Network Administration

CONCURRENT COMPUTER CORPORATION®

0890432-050

August 2001

Printed in U. S. A.

Revision History:		Level:	Effective With:
Original Release	-- August 1994	000	Power UX Version 1.1
Previous Release	-- March 1999	040	PowerMAX OS Version 4.3
Current Release	-- August 2001	050	PowerMAX OS Version 5.1

Preface

About This Book

Network Administration is directed to system administrators who are setting up and maintaining PowerMAX OS file sharing capabilities.

File sharing refers to the process of making file resources on your local system available to remote systems via a network, and conversely, to the process of accessing file resources on remote systems from your local system. The OS offers a file sharing package, also called a “distributed file system,” called the Network File System (NFS).

This guide tells you how to set up and administer NFS, and how to set up and administer Distributed File System (DFS), a software package that provides an interface to NFS. Also included in this guide is documentation that tells you how to set up and administer the OS implementation of TCP/IP, Remote Procedure Call (RPC), and Network Information System (NIS). Information about administering some mail service facilities associated with networking is also provided.

TCP/IP is a family of network protocols that determines how data is transferred across network media. TCP/IP supports NFS and is provided as the TCP/IP Internet package. It is necessary that you install TCP/IP to run NFS, because NFS requires UDP/IP as its transport. UDP/IP are protocols at the transport layer in the TCP/IP protocol family.

Included in TCP/IP at the application layer are a number of RPC commands and programs that allow users to perform remote operations on another host on the network as if were their own host. In addition to providing support for NFS, the RPC mechanism also supports NIS. NIS is a facility for distributed network administration.

NOTE

This guide is not intended to be an introduction to networking, nor to all the networking features of the OS.

How This Book is Organized

Because you may be setting up network services using a mix and match of applications and protocols, *Network Administration* is organized as seven parts that address the major topics listed below.

- Part 1, “Network Services Administration”

  This part provides an overview of networking, together with information about selecting a network, setting up name-to-address mapping, the connection server (which establishes connections for network services that communicate over TLI connection-oriented and dialup connections), using authentication schemes (for additional system security), setting up and administering ID mappings (for users on remote systems), administering and using the Basic Networking Utilities (BNU) (for communicating to other systems that support the Basic Networking Utilities), and interactive remote execution (REXEC) utilities (to allow remote administration of a machine).

- Part 2, “Mail Service Administration”

  This part describes administration of the online facility that allows users to exchange messages. Once basic networking is configured, you don't need to do any additional administration to use the mail facility. This part, however, will help you set up some special features, such as establishing a domain name, setting up mail directories to be shared across a networked file system, and setting up a connection to another site that uses the Simple Mail Transfer Protocol (SMTP). Also describes sendmail. Sendmail provides an alternative inter-network mail transport mechanism.

- Part 3, “TCP/IP Network Administration”

  This part provides information needed to set up and run TCP/IP on your system. The discussion includes information about configuring Internet addresses and describes how to use TCP/IP commands and files to implement a wide range of TCP/IP features. First, the part introduces you to important TCP/IP concepts you should be familiar with as an administrator. Next, it steps you through some basic TCP/IP administrative tasks. Finally, it describes some features in depth, such as domain name service and troubleshooting. This part is organized into the following chapters: “Introduction to Administering TCP/IP Networks” describes the concepts you need to understand to effectively do administration for your system. “Setting Up TCP/IP” contains step-by-step procedures for many basic tasks you need to do as an administrator. “Setting Up Routers and Subnetworks” describes how to expand and manage growing systems. “Managing TCP/IP Nodes Using SNMP” tells how to use SNMP to perform network monitoring and management functions. “Using Domain Name Service with TCP/IP” describes concepts and procedures relating to domain name service. “Troubleshooting and Tuning TCP/IP” describes how to diagnose problems and tune your system to improve TCP/IP performance. “Obtaining IP Addresses” describes how to obtain and complete IP address registration forms. “Obtaining Domain Names”, describes how to obtain and complete domain name registration forms. “Network Time Synchronization”, describes how to synchronize time among the machines on your network.

- Part 4, “Distributed File System Administration”

  This part describes the DFS command interface for NFS. For example, the DFS software provides you with the **`share`** command, which allows you to share a resource on your system using NFS. DFS Administration is covered in the following chapters: “Introduction to DFS Administration”, “Setting Up DFS”, “Using DFS Commands and Files”, and “DFS sysadm Interface”.

- Part 5, “Network File System Administration”

  This part tells you how to set up and maintain NFS on your system, including how to share and mount resources, how to mount resources automatically using a feature called the automounter, and how to set up Secure NFS. NFS Administration is covered in the following chapters: “Introduction to NFS Administration”, “Setting Up NFS”, “Sharing and Mounting NFS Resources Explicitly”, “Obtaining NFS Information”, “Troubleshooting and Tuning NFS”, “Setting Up Secure NFS”, “Using the NFS Automounter”, “Using the NFS Automounter”, and “Using the NFS sysadm Interface”.

- Part 6, “Remote Procedure Call Administration”

  This part tells you how to administer the files used by RPC, a mechanism for resource sharing between hosts used by NFS and NIS. Information about setting up and establishing secure RPC domains is also provided.

- Part 7, “Network Information Service Administration”

  This part explains how to set up, administer, and update NIS, a distributed database service used for password and host file administration.

- Glossary

  Contains definitions for terms and abbreviations used throughout this guide.

Related Documentation

The following manuals are referenced in this manual:

PowerMAX OS Release Notes	0890454-reln
Power Hawk Release Notes	0891058-reln
PowerMAXION Release Notes	0891063-reln
TurboHawk Release Notes	0891071-reln
Users Guide	0890428
Compilation System Volume 1	0890459
Compilation System Volume 2	0890460
System Administration (Volume 1)	0890429
System Administration (Volume 2)	0890430
Console Reference Manuals	
HN6200	0830047
HN6800	0830045
Power Hawk	0830050
PowerMAXION	0830052

reln = release number

Notation Conventions Used in This Book

This section describes the notation conventions used in this guide.

- References to literal computer input and output (such as commands entered by the user or screen messages produced by the system) are shown in a `monospace` font, as in the following example:

  ```
  $ ls -l report.oct17
  -rw-r--r--   1 jim   doc   3239 May 26 11:21 report.oct17
  ```

- Commands that are too long to fit on one line are separated by a backslash (\). This is not a character to be typed, but indicates that the command line continues on one line.

- Substitutable text elements (that is, text elements that you are expected to replace with specific values) are shown in an *italic* font, as in the following example:

  `$ cat` *file*

  The *italic* font is a signal that you are expected to replace the word *file* with the name of a file.

- Comments in a screen display-that is, asides from the author to the reader, as opposed to text that is not computer output-are shown in an *italic* font and are indented, as in the following example:

```
        .
        .
        .
   command interaction
        .
        .
        .
        .
Press RETURN to continue.
```

- Keyboard references are sometimes shown in a sans-serif font. Enter and Esc are two examples.

- Instructions to the reader to type input usually do not include explicit instructions to press the RETURN key at the appropriate times (such as after entering a command or a menu choice) because this instruction is implied for all UNIX system commands and menus.

  In one circumstance, however, an instruction to press the RETURN key is explicitly provided: when, during an interactive routine, you are expected to press RETURN without having typed any text, an instruction to do so will be provided, as follows:

  ```
      Type any key to continue: RETURN
      $
  ```

- Function key equivalents, which appear at the bottom of menus and are matched to the F keys across the top or side of your keyboard, are shown in ALL CAPITAL letters. MARK and SAVE are examples of function key equivalents. When instructed to press one of these keys, press the F key that corresponds with the label on your computer screen.

- Control characters are shown by the string CTRL-*char* where *char* is a character such as “d” in the control character CTRL-d. To enter a control character, hold down the CTRL key and press the letter shown. Be sure to type the letter exactly as specified: when a lowercase letter is shown (such as the “d” in the example above), enter that lowercase letter. If a character is shown in uppercase (such as CTRL-D), you should enter an uppercase letter.
- The system prompt signs shown in examples of interactive sessions are the standard default prompt signs :
  - the dollar sign (`$`) for an ordinary user
  - the pound sign (#) for the owner of the *root* login

Manual Pages

On-line manual pages can be viewed using the **`man`** command. References in text to “see manual page in *xxx Reference Manual*” implies displaying the applicable manual page using the **`man`** command. For example, entering “**`man cat`**” will display the **`cat(8)`** manual page on your terminal.

Contents

Part 3 TCP/IP Network Administration

Illustrations

Screens

Tables

1
Network Services Administration

Replace with Part 1 tab

1
Introduction to Network Services Administration

1
Introduction to Network Services Administration

An Overview of Network Services Administration

NOTE

If your system is being run in compliance with the security criteria described in the chapter "Trusted Facility Management" in the *System Administration* guide, network connections to the system are not allowed; although networking facilities have been enhanced to work with the Enhanced Security Utilities, networking facilities are not part of the Trusted Computing Base.

This section provides a brief overview to networking. This section is not intended to be a networking primer. Instead, it places the features of the OS in the context of a general network administration model and presents an overview of the tasks involved in the administration of user-level services.

A Model of Network Administration

The network facilities are built on the *client/server* model. A *client* is a networked machine that uses the resources of another machine on the network. A *server* is a system that provides resources to other systems on the network. A system can be both a client, utilizing another system's resources, and a server, making local resources available to other systems.

Typically, the steps involved in setting up a networked machine are the following:

1. Making the physical connection to the network and installing any software that drives the network hardware.
2. Installing the network software, which packages data according to a set of protocols. The network software includes a transport provider, which manages the transfer of data across the network connection.
3. On the server side, setting up continuous processes, called daemons, that listen for connection requests from other network machines.
4. On the client side, creating an address database that contains the addresses of all machines and services on the network to which the client can connect. The client consults the database before it sends a connection request to another network machine.

5. Setting up security. Typically, each network application has its own mechanism for performing a minimum of authentication.

The following section describes the network facilities and explains how they fit into the general model of network administration.

Networking Facilities

The OS includes a number of network facilities that give the administrator flexibility in building a network. Several facilities relieve the application of the need to know the underlying characteristics of the network. Thus, the application can run on different networks, and still present a consistent interface to the user.

Two distinct styles of network service provision are supported. One is the Berkeley style, implemented as “r-commands” (such as `rlogin`), which is described in *User's Guide.* The other style is the new Networking Applications Architecture (NAA), which is depicted in Figure 1-1 and described in the following list.

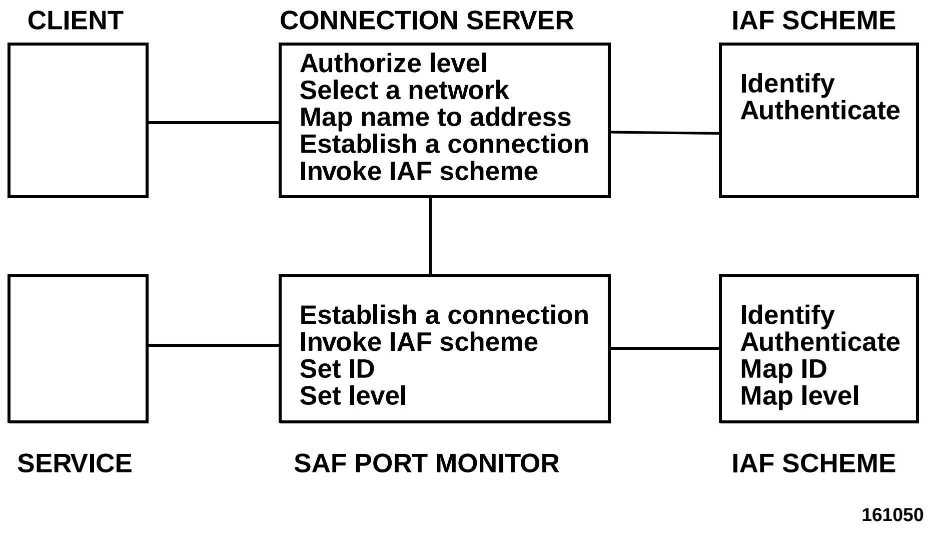


Figure 1-1. A Diagram of the Networking Applications Architecture

- The listen Port Monitor
  A port monitor is a continuous process, called a daemon, that listens to a given physical port or on a network address for connection requests. `listen,` also called the listener, is a network port monitor. It listens on addresses associated with a connection-oriented transport network. There may be multiple `listen` port monitors on the system, each monitoring multiple addresses.

  The listener listens on the server for incoming connection requests, accepts the requests, and starts services that have been requested. If a service is protected by an authentication scheme, the listener invokes the authentication scheme. If the authentication protocol completes successfully, the client user's identity is mapped to an identity on the server; then, the port monitor starts the requested service.

All port monitors, including `listen,` are managed by the Service Access Facility (SAF). The SAF generalizes service access procedures so login access on the local system and network access to local services are managed in similar ways. This means, from the system administrator's perspective, that local and network access are managed similarly, as is access over different networks. (For more information, see the chapter “Managing Ports” in the *System Administration* guide.)

- The Connection Server
  The Connection Server can be viewed as a counterpart on the client to the server's listener. When a client user attempts to access a service from another machine on the network, the Connection Server verifies the connection is allowable at the client's level. The Connection Server uses Network Selection to determine which network to use, and Name-to-Address Mapping to determine the address of the requested service on the requested host. The Connection Server sends a connection request to the server, manages the client's role in establishing the connection, and invokes the appropriate authentication scheme. If the scheme completes successfully, the Connection Server turns over the connection to the client process.

- Identification and Authentication Facility
  The Identification and Authentication Facility (IAF) authenticates network connections independently of the network application. It consists of three components—an invocation function, the cr1 and login authentication schemes, and the ID Mapping facility. The cr1 schemes (see the chapter entitled “cr1 Bilateral Authentication Scheme), and ID Mapping (see the chapter entitled “Administering ID Mapping), are discussed as separate features.

  Following the establishment of a connection, the Connection Server and the port monitor each invoke an authentication scheme. Once the client scheme has succeeded, the Connection Server passes the connection to the client application. Once the server scheme has succeeded, it searches the ID Mapping database to determine how the user on the client should be mapped to a user on the server. Minimally, the database consists of a map file that maps user logins on client systems to user logins on the server. Once the scheme maps the client user's identity into a local identity, the port monitor invokes the requested service.

  Authentication schemes are invoked by the IAF. This release of the OS contains the cr1 authentication scheme. By default, cr1 uses DES encryption, and can also be referenced as cr1.des. Because of export restrictions on DES, cr1 can also use ENIGMA encryption. When using ENIGMA encryption, cr1 is referenced as cr1.enigma. Other than the underlying encryption algorithm used, all cr1 schemes behave identically. The cr1 schemes operate as follows: The client and server schemes exchange a sequence of encrypted messages. Each system uses a secret key, which it retrieves from its local cr1 key database, to encrypt its own messages and decrypt the other's messages. (A secret key is a bit string known only to the client and server.) By successfully decrypting the other's messages, each machine authenticates the other's identity.The client scheme informs the server scheme of its user's security level and identity.

- Network Selection
  Network Selection is a facility that generalizes the way an application chooses a network. It allows an administrator or a user to specify an order of preference among available transport providers. When a user attempts to access a remote service, the system first attempts to make the connection over the primary network; if that attempt fails, it tries each network in order of preference until the connection is made.

- Name-to-Address Mapping
  Name-to-Address Mapping is a feature that allows applications to obtain transport-specific addresses in a transport-independent manner. The administrator maintains a configuration file with records describing each transport provider available to be used by applications. One of the fields in each record is the name of the shared library to use for the transport provider being described in the record. This library contains name-to-address mapping routines specific to the transport protocol. When a new transport protocol is being installed, a new entry in the configuration file is made, allowing network applications to make use of the new protocol without being changed.

- User-Level Services
  The facilities that provide services to end-users include the Basic Networking Utilities (BNU) and REXEC. BNU provides basic network communication (such as queued remote execution and file transfer capabilities) and REXEC is an interactive remote execution facility. Whereas many network packages provide remote execution capabilities, BNU and REXEC provide file transfer and remote execution independent of the transport provider. Both BNU and REXEC take advantage of the features of the Network Application Architecture (NAA) described in the previous sections.

  In addition to BNU and REXEC, the OS includes a distributed file system package, the Network File System (NFS). NFS provides access to files and directories across a network of machines running different operating systems. In addition, TCP/IP includes a number of application level services in its protocol suite.

  BNU and REXEC are described in this chapter. Distributed File Systems are described in “Introduction to DFS Administration,” “Setting Up DFS,” “Using DFS Commands and Files,” and “DFS sysadm Interface.” The user services that are provided by TCP/IP are discussed in *User's Guide*.

The following section leads you step-by-step through administration of BNU and REXEC.

Procedural Overview of BNU and REXEC Administration

Ultimately, the network administrator's objective is to provide services to users of networked machines. This section presents a high-level procedure for setting up REXEC and BNU, as well as all the network services on which they depend. The administration of each component mentioned in the following procedure is described in more detail later in the chapter.

This procedure makes the following assumptions:

1. Your network hardware and software have been installed and any administration specific to the network package has been completed.
2. Applications are to be set up to take advantage of the IAF in the OS. For compatibility, BNU can be configured as it was on previous versions of the OS. However, this section assumes BNU connections will be configured using enhanced authentication procedures (primarily cr1).

Set Up the Network Administrator's Login

If you are running the Least Privilege Module (LPM), which is provided as part of the Enhanced Security Utilities, the concept of the superuser is replaced by administrative roles. Each administrative role is associated with commands and privileges required to administer a certain aspect of the system. To administer network services, you must assume the role of the network administrator, NET. NET is predefined on the system; however, before you can log in as a network administrator, you must create a login and assign it the NET role. This is done using the **`adminuser`** command. For information about assigning roles, see the chapter “Securing User Accounts” in the *System Administration* guide or **`adminuser(1M)`**.

You must complete this task before beginning the following sequence of steps.

Step 1: Set Up Network Selection

Network Selection consists of the following elements:

- **`netconfig,`** a network configuration file [see **`netconfig(4)`**]
- `NETPATH,` an environment variable

To set up Network Selection, you need to create an entry in the **`netconfig`** file for each network available to the local system.

Any user, including the administrator, may use the `NETPATH` variable to modify the default order in which networks are tried by an application seeking a network connection. If a user does not set the `NETPATH` variable, networks are used in the order specified in **`netconfig.`**

For more information see Chapter 2, "Administering Network Selection".

Step 2: Set Up Name-to-Address Mapping

Name-to-Address Mapping consists of routines that application programs can call to determine the addresses of services and machines on the network. These routines are combined into libraries, with one library associated with each available transport provider. For every network available to the local system, the network administrator must create and maintain the following files used by the library routines:

- the **`hosts`** file, which lists the addresses of machines on the network
- the **`services`** file, which lists the port numbers of services available across the network

The files that support TCP/IP must be located in **`/etc`**. The files that support other networks must be located in **`/etc/net/`***transport*, where *transport* is the name of the transport provider.

For more information, see Chapter 3, “Administering Name-to-Address Mapping.”

Step 3: Set Up the listen Port Monitor

The listener is a port monitor under the control of the Service Access Controller (SAC), a daemon process that provides a consistent interface to all services, whether the user's connection to the system is local or across a network. (Because the purpose of the SAC is not limited to managing network access, it is not documented in this chapter. For more information about the SAC and setting up the listener, see the chapter “Managing Ports” in the *System Administration* guide.) To set up the listener, do the following:

1. Add a `listen` port monitor to the Service Access Controller's administrative file, using **`sacadm`** [see **`sacadm(1M)`**].
2. Set up the port monitor's administrative file, using **`pmadm`** [see **`pmadm(1M)`**]. Setting up the file involves
   - adding one or more services to the listener's administrative file
   - enabling the service on a port that the listener is monitoring
   - associating an authentication scheme with a service

Step 4: Set Up the Connection Server

The Connection Server is a daemon process running on the client that makes network connections. Applications obtain network connections by calling library routines that access the daemon. An error reporting routine (`cs_perror`), an associated service (`reportscheme`), and three optional administrative files are available. To set up the Connection Server, do the following:

1. Optionally, on the client, set up the file **`/etc/iaf/serve.allow.`** The **`/etc/iaf/serve.allow`** file enforces authentication, in the event the service is not protected by an authentication scheme on the server side.

2. Optionally, on the client, set up the file **`/etc/iaf/serve.alias.`** The **`/etc/iaf/serve.alias`** file maps aliases to service names, making it possible to register a service under two names with different authentication schemes associated with each name.
3. Optionally, on the client, set up the authentication file, **`auth.`** This file lists network services and their associated authentication schemes.
4. If your system is running Mandatory Access Control (MAC), use **`attradmin`** to set up LID authorization. The Connection Server uses attribute mapping to determine if a process at a given security level is authorized to make a connection request to the specified server. (Information on attribute mapping is provided below under Step 6: Setting Up ID Mapping.
5. Add an entry for `reportscheme` to the port monitor's administrative file on each server to which the Connection Server should be able to connect.

For more information about Connection Server administration, see Chapter 4, “Administering the Connection Server.”

Step 5: Set Up the cr1 Authentication Scheme

To set up cr1, do the following:

1. Start the **`keymaster`** daemon process [see **`keymaster(1M)`**].
2. Administer individual keys using **`cryptkey`** [see **`cryptkey(1)`**].

If you intend to set up REXEC or BNU, you need to add keys shared by a client machine and the server machines. Each client and server pair must share identical keys.

For more information, including detailed procedures, see Chapter 5, “cr1 Bilateral Authentication Scheme.”

Step 6: Setting Up ID Mapping

cr1 maps remote user identities to the local system using map files that are administered under the ID Mapping facility.

On a server, to set up ID Mapping to support cr1, do the following:

1. Use the **`idadmin`** command [see **`idadmin(1M)`**] to install cr1 name mapping.
2. If you are running MAC, set up attribute mapping for the `RLID` attribute, using the **`attradmin`** command. cr1 uses the `RLID` attribute to map level aliases from client systems to level aliases on the server.

On the client, to set up attribute mapping, do the following:

1. Set up attribute mapping for the `LIDAUTH` attribute. The Connection Server uses the `LIDAUTH` attribute to determine whether or not a process is authorized to make a connection request.

Specific instructions for setting up the `LIDAUTH` attribute appear in Chapter 4, “Administering the Connection Server.” Instructions for setting up ID mapping and attribute mapping appear in Chapter 6, “Administering ID Mapping.”

Instructions for setting up ID mapping appear in Chapter 6, “Administering ID Mapping.”

Step 7: Set Up BNU

The Basic Networking Utilities package is a collection of programs and support files that provide network communication services such as file transfer capabilities and remote execution. All BNU support files are located in **`/etc/uucp.`**

To set up BNU, do the following:

1. Create BNU logins.
2. On the server side, for every transport provider on which BNU will be supported, use **`pmadm`** to add the `uucico` service and associate the cr1 authentication scheme with the service.
3. Create or modify the BNU database files as needed. See the Database Support Files section of Chapter 7, “Administering the Basic Networking Utilities.
4. If your system supports MAC, verify that `LIDAUTH` attribute mapping has been set up for the Connection Server, and `RLID` attribute mapping has been set up for cr1.

Step 8: Set up REXEC

To set up REXEC, do the following:

1. Register REXEC with a network port monitor (the listener) under the Service Access Facility, specifying an associated authentication scheme that uses ID Mapping, that is, cr1.
2. Set up the file **`/etc/rexec/services`** on the server.

The **`services`** file is a database containing all the services on the server system that are available to remote users through REXEC. Typically, the list of available services consists of the pre-defined REXEC services `rl, rx,` and `rquery.`

For more information, see Chapter 8, "Administering REXEC".

Using the sysadm Menu Interface

Any of the functions associated with Network Services may be performed by selecting the appropriate “task” from a series of menus provided for administration. To access the system administration menu for using Network Services, type:

`sysadm network_services`

The following menu will appear on your screen:

```
    Network Services Management

attr_map                - Attribute Mapping Administration
basic_networking        - Basic Networking Utilities Management
cr1                     - IAF Scheme cr1 Key Management
name_map                - Name Mapping Administration
name_to_address         - Machine and Service Address Management
remote_files            - Distributed File System Management
selection               - Network Selection Management
```

Screen 1-1. Network Services Management Menu

NOTE

The Distributed File System Management option is described in the *Compilation System Manual*.

When you have chosen an option, self-explanatory submenus and instructions will be displayed on the screen to lead you through the appropriate procedures.

2
Administering Network Selection

2
Administering Network Selection

Introduction to Network Selection Administration

In order for network applications to be portable to different environments, the application process must have a standard interface into the various transport providers available in any current environment. Network Selection provides a simple and consistent interface that allows user applications to select networks (at the transport level), enabling applications to be protocol- and media-independent. Networking Services applications that allow a user to influence the choice of transports use the standard interface outlined here.

Tasks associated with Network Selection administration may be performed either by using the menu system or by entering shell commands on the command line. Screen 2-1 is the top-level menu for Network Selection. It can be brought up on the screen by typing the following:

```
sysadm selection
```

```
    Network Selection Management

display  -    Display Network Selection Configuration
modify   -    Modify Network Selection Configuration
```

Screen 2-1. Network Selection Management Menu

When you select an option, self-explanatory submenus and instructions lead you through the appropriate procedures.

You can also bypass the menu system by issuing commands directly to the shell. Where these commands involve editing sensitive system files, be sure to keep a backup copy of the file you are editing. When you have finished editing the file, use **`diff`** [see **`diff(1)`**] on the edited file and the backup copy to verify that only the changes you want have been made.

Table 2-1. Command Alternatives to the Network Selection Management Menu

Task Description	Menu Item	Shell Command

Display the contents of the **`netconfig`** file; display the entry for network *netid*	`display`	**`cat /etc/netconfig`**
		`grep` *netid* **`/etc/netconfig`**

Change a **`netconfig`** entry	`modify`	**`vi /etc/netconfig`**

If you want to add networks to the network configuration database, you need to edit **`/etc/netconfig`**, and add the appropriate entries. To remove networks from the network configuration database, you need to edit **`/etc/netconfig`**, and delete the appropriate entries.

Network Selection Overview

The Network Selection component is built around:

- a network configuration database (the **`/etc/netconfig`** file) that contains entries for each network available to the system, and
- an optional `NETPATH` environment variable, set by a user or the system administrator and containing an ordered list of network identifiers. These network identifiers match the **`netconfig`** `network ID` field and are used as links to the records in the **`netconfig`** file.

The Network Selection application programming interface consists of a set of network configuration database access routines. One group of these library routines accesses only the **`netconfig`** entries identified by the `NETPATH` environment variable; another group of routines accesses **`netconfig`** directly. The first group is also described in detail in **`getnetpath(3N)`**. The second group is described in **`getnetconfig(3N)`**.

Applications should use the routines that access `NETPATH.` These routines allow users to influence the selection of transports used by the application. If an application does not want the user to influence its decision, then the routines that access the **`netconfig`** database directly should be used.

The **`netconfig`** file, on which the Network Selection library routines depend, is maintained by the system administrator. The `NETPATH` environment variable is typically set or modified by application programmers and users, depending on the needs of their applications, but it may also be set by the system administrator in response to the needs of administrative applications.

The /etc/netconfig File

The system administrator is responsible for maintaining the network configuration database file **`/etc/netconfig`**. Entries in the **`netconfig`** file contain the following fields in the order shown. A sample **`netconfig`** file is shown in Table 2-2.

Table 2-2. Fields in netconfig Entries

network_id	*semantics*	*flags*	*protofamily*	*protoname*	*device*	*nametoaddr_libs*

The fields correspond to elements of the **`netconfig`** structure. Pointers returned by Network Selection library routines are pointers to **`netconfig`** entries in **`netconfig`** format. See **`netconfig(4)`** for more information. The **`netconfig(4)`** manual page also describes the elements of the `struct netconfig` structure. All symbolic names, structure definitions, and constant values for the Network Selection feature are defined in the header files **`/usr/include/netconfig.h`**, **`/usr/include/sys/netconfig.h`**, and **`/usr/include/netdir.h`**.

`netconfig` fields are defined as follows:

network_id — A string used to identify a transport provider. *network_id* consists of non-NULL characters, and has a length of at least 1. No maximum length is specified. This name space is locally significant and the local system administrator is the naming authority responsible for ensuring that all *network_id*s on a system are unique.

semantics — A string that identifies the "semantics" of the transport provider, that is, the set of services it supports, by identifying the service interface it provides. This is closely related to, but not identical with, the API (Application Programming Interface) with which applications are "supposed" to access the network. Typically, an application will specify its API by pushing an appropriate STREAMS module (such as `timod`) and using an appropriate user-level library (such as the TLI library). The *semantics* field is mandatory. The following semantics are recognized.

`tpi_clts`	Transport Provider Interface, connection-less
`tpi_cots`	Transport Provider Interface, connection-oriented
`tpi_cots_ord`	Transport Provider Interface, connection-oriented and supports orderly release
`tpi_raw`	Transport Provider Interface, raw

flags — The *flags* field contains two-valued ("true" and "false") attributes of transport providers. *flags* is a string of characters, each of which specifies the value of the corresponding attribute. If the character is present, the attribute is "true." If the character is absent, the attribute is “false.” A hyphen (`-`) indicates no attributes are present. The characters currently recognized are:

`v`	Visible network. Used to establish a default list of networks to search when the environment variable `NETPATH` is *unset*. See the section entitled “The NETPATH Environment Variable” on page 2-6 for a description of how the `v` flag is used.
`b`	Enable RPC broadcast. Used by `rpc_broadcast`() [see **`rpc_clnt_calls(3N)]`**.

protofamily

The *protofamily* and *protoname* fields are provided for protocol-specific applications. The *protofamily* field contains a string that identifies a protocol family. The *protofamily* identifier follows the rules for *network_id*s: It is a string of non-NULL characters with a length of at least `1`. No maximum length is specified.

A hyphen (`-`) in the *protofamily* field indicates that none of the available protocol family identifiers applies, that is, the transport provider is experimental. An application that wants to have family characteristics can match on the *protofamily* field when selecting a network. (For example, an application can search for an “osi” family.) In this case, the application is not protocol independent, since it has searched only for OSI entries. The following are examples of protocol family identifiers:

`loopback`	Loopback (local to host)
`inet`	Internetwork: UDP, TCP, and so on.
`implink`	ARPANET imp addresses
`pup`	PUP protocols: for example, BSP
`chaos`	MIT CHAOS protocols
`ns`	XEROX NS protocols
`nbs`	National Bureau of Standards (NBS) protocols
`ecma`	European Computer Manufacturers Association
`datakit`	Datakit protocols
`ccitt`	CCITT protocols, X.25, and so on.
`sna`	IBM SNA
`decnet`	DECNET
`dli`	Direct data link interface
`lat`	LAT
`hylink`	NSC Hyperchannel

`appletalk`	Apple Talk
`nit`	Network Interface Tap
`ieee802`	IEEE 802.2; also ISO 8802
`osi`	Umbrella for all families used by OSI
`x25`	CCITT X.25 in particular
`osinet`	AFI = 47, IDI = 4
`gosip`	U.S. Government OSI

protoname

The *protoname* field contains a string that identifies a protocol. This field is currently used only for the `inet` family. For any other family, the protocol name field contains a hyphen (-). The *protoname* identifier follows the same rules as *network_ids*: The string consists of non-NULL characters and has a length of at least `1`. No maximum length is specified. The *protoname* field may contain:

`icmp`	Internet Control Message Protocol
`tcp`	Transmission Control Protocol
`udp`	User Datagram Protocol

device

The *device* is the full pathname of the device used to connect to the remote machine via the transport provider. Typically, this device will be in the **`/dev`** directory. The *device* must be specified.

nametoaddr_libs

The *nametoaddr_libs* support a "directory service" (that is, a Name-to-Address Mapping service) for the network. This service is implemented by the Name-to-Address Mapping feature. If a transport is not provided with such a library, that is, if the *nametoaddr_libs* field in the **`netconfig`** file contains only a hyphen, the Network Selection request will fail. The *nametoaddr_libs* field consists of a comma-separated list of full pathnames to dynamically linked libraries.

Literal commas may be embedded as "`\,`"; backslashes as "`\\`." Lines in **`/etc/netconfig`** that begin with a pound sign (#) in the first column are comments.

The system administrator determines the order of the entries in the **`netconfig`** database. Because the Network Selection library routines that access **`netconfig`** directly return entries in order, beginning at the top of the **`/etc/netconfig`** file, the order in which networks are entered in the file by the system administrator becomes the default search path for applications choosing networks to which they will connect.

```
#network_id semantics flags protofamily protoname device nametoaddr_libs
#
ticlts    tpi_clts     v  loopback  - /dev/ticlts    /usr/lib/straddr.so
ticots    tpi_cots     v  loopback  - /dev/ticots    /usr/lib/straddr.so
ticotsord tpi_cots_ord v  loopback  - /dev/ticotsord /usr/lib/straddr.so
tcp       tpi_cots_ord v  inet    tcp /dev/tcp       /usr/lib/tcpip.so,\
                                                     /usr/lib/resolv.so
udp       tpi_clts     v  inet    udp /dev/udp       /usr/lib/tcpip.so,\
                                                     /usr/lib/resolv.so
icmp      tpi_raw      -  inet   icmp /dev/icmp      /usr/lib/tcpip.so,\
                                                     /usr/lib/resolv.so
rawip     tpi_raw      -  inet     -  /dev/rawip     /usr/lib/tcpip.so,\
                                                     /usr/lib/resolv.so
```

Screen 2-2. Sample netconfig File

The NETPATH Environment Variable

In most cases a user isn't interested in which transport provider handles a network operation, and the default network search path established by the system administrator (the **netconfig** file) is used to locate a transport provider available for connection. However, if a user or the system administrator wants to influence the choices made by applications, the search path can be modified using a new standard shell variable, NETPATH. NETPATH is similar to the PATH variable.

NETPATH consists of a colon-separated list of network IDs. Each network ID corresponds to the *network_id* field of a record in the **netconfig** database. A literal colon can be embedded as "\:" and a literal backslash as "\\." An empty component in NETPATH (signified by a beginning colon, an ending colon, or two successive colons) is not a valid entry because the empty string is not a valid network ID. NETPATH is described in **environ(5).**

The NETPATH environment variable is not set in **/etc/profile**. It can, however, be set in a user's **$HOME/.profile**.

Both users and system administrators should be aware that the set of "default" networks is different for routines that access **netconfig** directly and routines that access **netconfig** via the NETPATH environment variable. For routines that access **netconfig** directly [see **getnetconfig(3N)]**, the set of default networks is the entire **netconfig** file; the set of "default" networks for the routines that access **netconfig** via NETPATH is the visible networks in the **netconfig** file [see **getnetpath(3N)].** A network is "visible" if the system administrator has included a v flag in the flag field. If NETPATH is unset, these visible networks are the default search path for this second group of access routines.

3
Administering Name-to-Address Mapping

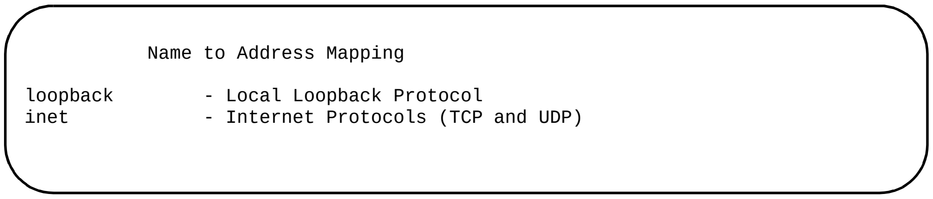
3
Administering Name-to-Address Mapping

Introduction to Name-to-Address Mapping Administration

The Name-to-Address Mapping feature allows an application to obtain the address of a service on a specified machine in a transport-independent manner.

Tasks associated with Name-to-Address Mapping administration may be performed using the menu system or shell commands entered on the command line. Screen 3-1 is the top-level menu for Name-to-Address Mapping. It can be brought up on the screen by typing.

```
sysadm name_to_address
```



```
           Name to Address Mapping

loopback        - Local Loopback Protocol
inet            - Internet Protocols (TCP and UDP)
```

Screen 3-1. Machine and Service Address Management Menu

NOTE

The menu screen may be different on your machine. The actual menu selections depend on the software packages and protocols installed on your system.

When you have selected the option you want, self-explanatory submenus and instructions will be displayed on the screen to lead you through the appropriate procedures.

If you want to bypass the menu system, you may issue commands directly to the shell, as shown in Table 3-1.

Name-to-Address Mapping consists of routines for use by application programs. These routines [described on **`netdir(3N)`**] are used to obtain addresses of services on given hosts. All routines are combined into a library, one for each transport provider address management mechanism. The library to use for a specific transport provider is named in the **`/etc/netconfig`** file. A call to any of these routines by an application dynamically links the library named in the *nametoaddr_libs* field of the **`/etc/netconfig`** file.

Table 3-1. Shell Commands for Name-to-Address Mapping

Task Description	Menu Item	Shell Command
Local Loopback Protocol	loopback	**vi /etc/net/***transport***/hosts** **vi /etc/net/***transport***/services**
Internet Protocols (TCP and UDP)	inet	**vi /etc/hosts** **vi /etc/services**

The functions described on **netdir(3N)** take a pointer to a `netconfig` structure and returns a list of addresses of the service and host names over a given transport provider.

The following libraries provide the routines shown on **netdir(3N)**:

tcpip.so	contains the **/etc/hosts** Name-to-Address Mapping routines for the TCP/IP protocol suite
resolv.so	contains the Domain Name Server (DNS) Name-to-Address Mapping for the TCP/IP protocol suite
straddr.so	contains the Name-to-Address Mapping routines for any protocol that accepts strings as addresses. The loopback driver is an example.

The tcpip.so Library

The routines in this dynamic library create addresses from the **/etc/hosts**(4) and **/etc/services**(4) files available with the TCP/IP package. The **/etc/hosts** file contains the machine's IP address as the first field followed by any number of machine names separated by white space as shown in Screen 3-2.

```
#IP address          machine name(s)        (optional comment)
#
192.11.108.01        bilbo
192.11.108.16        elvis                  # Located at Graceland
192.11.103.2         weeble wombat
```

Screen 3-2. Sample /etc/hosts File

The **/etc/services** file contains three fields, service name, port/protocol (with one of two protocol specifications either `tcp` or `udp`), and aliases as shown in Screen 3-3.

```
#service name     port/protocol     aliases
#
netstat           15/tcp
netstat           15/udp
time              37/tcp            mail
time              37/udp            mail
nntp              119/tcp           usenet readnews untp
```

Screen 3-3. Sample /etc/services File

For an application to use this library to request the address of a service on a particular host, the host name must appear in the **`/etc/hosts`** file and the service name must appear in the **`/etc/services`** file. If one or the other does not appear, an error will be returned by the name-to-address mapping routines.

The tcip_nis.so Library

The **`tcpip_nis.so`** library is identical to **`tcip.so`** with the exception that NIS support is built in for accessing the files **`/etc/hosts`** and **`/etc/services`**. This arrangement allows client systems to access NIS-managed versions of the hosts and services file from server system.

The resolv.so Library

The routines in this dynamic library create addresses similar to the **`tcpio.so`** file, except that it uses Domain Name Service (DNS), (see Chapter 15, "Using Domain Name Service with TCP/IP") instead of **`/etc/hosts`** to provide similar features.

The straddr.so Library

The routines in this dynamic library create addresses from files that have the same format as the **`tcpip.so`** file described above. The **`straddr.so`** files are **`/etc/net/`***transport***`/hosts`** and **`/etc/net/`***transport***`/services`**. *transport* is the local name of the transport provider that accepts string addresses (specified in the *network_id* field of the **`/etc/netconfig`** file). For example, the host file for `ticlts` would be **`/etc/net/ticlts/hosts`**, and the service file for `ticlts` would be **`/etc/net/ticlts/services`**. For `ticots`, the files would be **`/etc/net/ticots/hosts`** and **`/etc/net/ticots/services`**.

Even though most string addresses do not distinguish between "host" and "service," separating the string into a host part and a service part provides consistency with other transport providers. The **`/etc/net/`***transport***`/hosts`** file will therefore contain a string that is considered to be the machine address, followed by the machine name as shown in Screen 3-4.

```
#machine addr  machine name
#
bilboaddr      bilbo
elvisaddr      elvis
frodoaddr      frodo
```

Screen 3-4. Sample /etc/net/transport/hosts File

The **`/etc/net/`***transport***`/services`** file contains a service name followed by a string identifying the service port as shown in Screen 3-5.

```
#service name  service port
#
rpcbind        rpc
kerserv        kerserv
listen         serve
statd          statd
lockd          lockd
nfsd           nfsd
reportscheme   reportscheme
```

Screen 3-5. Sample /etc/net/transport/services File

The routines create the full string address by combining the "host address" and the "service port," separating the two with a dot (.). For example, the address of the "listen" service on `bilbo` would be `bilboaddr.serve` and the address of the "rpcbind" service on `bilbo` would be `bilboaddr.rpc`.

When an application requests the address of a service on a particular host on a transport provider that uses this library, the host name must appear in **`/etc/net/`***transport***`/hosts`** and the service name must appear in **`/etc/net/`***transport***`/services`**. If one or the other does not appear, the Name-to-Address Mapping routines return an error.

4
Administering the Connection Server

4 Administering the Connection Server

Introduction to Connection Server Administration

The Connection Server [see **cs(1M)**] is a standing process or daemon that runs on all client machines. It is used to establish connections for all network services that communicate over TLI connection-oriented and dialup connections. The Connection Server is started automatically (from **/etc/dinit.d/S80cs**) during initialization when the system goes to multi-user state. It receives requests for network services from client machine applications, establishes connections to the server-machine ports associated with the requested services, and passes the connections back to the application. Before passing a connection to an application, the Connection Server may invoke an authentication scheme.

The Connection Server is made up of the following components:

- An application interface to the standing server. The interface consists of library routines that make the connection over connection-oriented networks or using dialup connections, and an error reporting routine. The use of the application interface routines is described in The Connection Server Application Interface section in this chapter and on **cs_connect(3N)** and **dial(3N)**.
- An **/etc/iaf/serve.allow** file, maintained on the client machine. **/etc/iaf/serve.allow** contains a list of network services that client applications expect to use and the acceptable authentication scheme or schemes for each service. This file is not needed if client applications do not authenticate server identities, that is, if client applications will accept any authentication scheme imposed by server machines. **/etc/iaf/serve.allow** is described below.
- An optional file, **/etc/iaf/serve.alias**, also maintained on the client machine. **/etc/iaf/serve.alias** contains a list of server names, network service names and their aliases. The file is described below.
- If the Enhanced Security Utilities are installed, an **/etc/idmap/attrmap/LIDAUTH.map** file, maintained on the client machine. **LIDAUTH.map** indicates whether or not the client is authorized to connect to the remote system at the current security level. If **LIDAUTH.map** contains a line that authorizes a connection, the Connection Server sends the request to the server. If **LIDAUTH.map** does not authorize a connection at the security level of the local process, the Connection Server does not pass the connection request to the server. If a system is running the Enhanced Security Utilities and a **LIDAUTH.map** file does not exist, the

Connection Server fails all outgoing connection requests. **`LIDAUTH.map`** is described below.

- A non-standing network service, **`reportscheme`**, that tells client machine applications what authentication scheme to use for a requested network service. The **`reportscheme`** service must exist on each port monitor that offers network services if the server is to enforce authentication scheme invocation. **`reportscheme`** is described throughout this section and on **`reportscheme(1M)`**.
- A Connection Server log file.

In addition, the Connection Server may make use of the Service Access Facility's administrative command, **`pmadm`** [see **`pmadm(1M)`**], to install authentication schemes.

The Connection Server Application Interface

The application interface to the Connection Server consists of the library routines `cs_connect()`, `cs_perror()`, and `dial()`. They are described on **`cs_connect(3N)`** and **`dial(3N).`**

Connection Server Administration

An authentication scheme is a program called by `listen` on the server machine and by the Connection Server on the client machine. It is called after physical connection takes place and before the connection is handed over to the service. An authentication program allows client and server machines to verify each other's identities.

Authentication scheme administration is performed on both client and server machines.

Server Machine Administration

Registering Authentication Schemes

The authentication scheme for a network service is registered using the port monitor administrative command, **`pmadm`** [see **`pmadm(1M)`**]. Each network service under the Service Access Facility is associated with a port under a given port monitor. The port monitor administrative command adds, removes, and maintains services by adding, removing, or changing lines in a port monitor's administrative file. (See the chapter “Managing Ports” in the *System Administration* guide and **`pmadm(1M)`** for more information.)

The system administrator must “register” the authentication scheme for a service. If the service exists on the system, this is done using the port monitor administrative command, **`pmadm`**, with the **`-c`** and **`-S`** *scheme* options. If an authentication scheme has not already been associated with the service, *scheme* will be entered in the *scheme* field of the port monitor's administrative file. If an authentication scheme has already been associated with the service, the same command will replace the existing scheme with *scheme*. A **`pmadm`** command line in the following form will remove an authentication scheme for a service:

`pmadm -c -S “” -s` *svctag* **`-p`** *pmtag*

If the service does not yet exist, the name of the authentication scheme associated with it may be included on the command line that adds the service. In this case, **`-S`** *scheme* is included as one of the arguments on the **`pmadm -a`** command line.

After the authentication scheme name has been included in the *scheme* field of the port monitor's administrative file, it is available to the Connection Server on the client machine by way of the **`reportscheme`** service.

Client Machine Administration

If the administrator of the client machine wants to enforce the use of a given authentication scheme or schemes for a particular service and machine, the **`/etc/iaf/serve.allow`** file must be administered. If the **`/etc/iaf/serve.allow`** file is not administered, any authentication scheme specified by the server machine is used; if no scheme is specified, the NULL scheme is assumed. Any scheme named in the **`/etc/iaf/serve.allow`** file thus serves to enforce the type of authentication specified.

Maintaining the /etc/iaf/serve.allow File

The **`/etc/iaf/serve.allow`** file lists the names of the network services the client machine expects to use and the names of the authentication schemes acceptable to the client machine for use with each service. The system administrator is responsible for creating and maintaining the **`/etc/iaf/serve.allow`** file on the client machine. Each line in the file contains the name of a server machine, a network service name, and a comma-separated list of scheme names. The three fields are separated by white space.

```
#server name    service name    scheme name(s)
#
elvis           banking         cr1,other_scheme
elvis           uucico          cr1
```

Screen 4-1. Sample /etc/iaf/serve.allow File

In the example, `elvis` is a server machine name, `banking` is a network service name, and `cr1` is the name of authentication scheme acceptable to the client machine for the `banking` service on `elvis`. *other_scheme* could be any other secure authentication scheme (for example, `kerberos`). It must be registered with the `banking` service on the server machine for an application on the client machine to be able to access the service. If there is no entry in the **`/etc/iaf/serve.allow`** file for a network service, the client machine accepts any authentication scheme the server machine specifies for the service.

The system administrator maintains the **/etc/iaf/serve.allow** file using an editor such as vi.

Maintaining the /etc/iaf/serve.alias File

Before Name-to-Address Mapping is invoked, the Connection Server consults the **/etc/iaf/serve.alias** file, if it exists, to find out if the service requested by an application should be requested under another name. If the service name is found in the file, the Connection Server substitutes its alias for the name given by the application.

The administrator of a server machine may register a service under two names with two different authentication schemes to implement a gradual migration from one authentication scheme to another. For example, a server machine may register a new authentication scheme, newauth, with one of its network services, **date**. Not all client machines that use the server machine's **date** command have installed the newauth scheme. In this case the administrator of the server machine will add a **date.old** line to the service line in the appropriate port monitor administrative file and enter the old authentication scheme name, ns, in the *scheme* field. A client machine that wants to continue using the ns authentication scheme will add

```
#server name    service name     service name alias
#
elvis           date             date.old
```

to its **/etc/iaf/serve.alias** file.

In the example, the first field is the name of the server machine, elvis; the second field is the network service as it is known to the application. Field three is the network service's alias. The Connection Server will translate the application's request for the **date** command to a request for **date.old** and the ns authentication scheme will be used.

The **/etc/iaf/serve.alias** file is updated using one of the text editors such as vi.

Setting Up LIDAUTH.map

If the Enhanced Security Utilities are installed and running on your system, the client administrator must enable network communication from the client side by creating the **/etc/idmap/attrmap/LIDAUTH.map** file. **LIDAUTH.map** authorizes outgoing connection requests.

A map file that authorizes connection requests is an extension of the kind of attribute map described under Chapter 4, “Administering ID Mapping”. It gives the administrator control over the levels at which a local process can communicate with a remote system.

When a local process attempts to connect to a remote system, the Connection Server first consults the **LIDAUTH.map** file to determine whether the connection can be made at the level of the running process. If the map file contains an entry that authorizes the connection request, the Connection Server sends the request to the server.

If the Connection Server consults the authorization map and determines that a connection request at the process's security level is not authorized, it does not pass the connection request to the server.

If the Enhanced Security Utilities are installed and running on the system and the **`LIDAUTH.map`** file does not exist, the Connection Server fails all outgoing connection requests.

To create the **`LIDAUTH.map`** file, enter the following command:

```
attradmin -A LIDAUTH -I M2:M1
```

`M2:M1` is a format descriptor, which indicates that the attributes will be provided in the format *remote_system_name*:*local_LID*, with both fields mandatory. The form of the descriptor is dictated by the Connection Server, as is the name of the map file.

The following is a sample **`LIDAUTH.map`** file:

```
!M2:M1
sysA:SYS_PUBLIC %1
sysB:SYS_PUBLIC %1
sysB:* %i
sysC:* %1
```

To add an entry to the map file, use the **`attradmin`** command with the following syntax:

`attradmin -A` *attrname* **`-a -r`** *remoteval* **`-l`** *localval*

where *attrname* is `LIDAUTH,` and *localval* is either `%1` or `%i` to indicate whether or not the connection request should be authorized. `%1` indicates that processes with the specified LID are authorized to send a connection request and `%i` indicates that the outgoing connection is not allowed to the specified system at the specified LID.

A LID should be specified using its alias (such as `SYS_PUBLIC`) whenever an alias exists on both systems. For example, if you want to allow only local processes with LID `SYS_PUBLIC` to talk to a remote system called `sysA,` enter the following command line.

```
attradmin -A LIDAUTH -a -r sysA:SYS_PUBLIC -l %1
```

The entry

```
sysA:SYS_PUBLIC      %1
```

indicates that a local process with LID `SYS_PUBLIC` is authorized to send a connection request to a server named `sysA` and to attempt to connect to the server at its lowest security level (LID `SYS_PUBLIC`).

If the server does not support the Enhanced Security Utilities, it accepts incoming connection requests from all levels. The client administrator, however, may choose to restrict access to the server from the client side by including appropriate entries in **`LIDAUTH.map.`** The map entries

```
sysB:SYS_PUBLIC      %1
sysB:* %i
```

indicate that the client administrator has chosen to allow only processes at the lowest security level (LID `SYS_PUBLIC`) to connect to a server named `sysB.` The second entry in this example is needed only if the **`LIDAUTH.map`** file also has an entry

```
*:*      %1
```

which specifies transparent mapping for all other attribute pairs. If this entry is not in the file, lack of a match would cause the request to fail and the second entry above would not be required.

Examples

If you know the remote system `sysC` supports the same security levels and LIDs as the local system, and you want to allow communication at all levels, enter the following command line:

```
attradmin -A LIDAUTH -a -r "sysC:*" -l %1
```

NOTE

To protect characters that have special meaning to the shell, enclose them in double quotes.

If all the machines on the network support the same levels and LIDS, and you want your machine to have access to all network machines at all levels, enter the following:

```
attradmin -A LIDAUTH -a -r "*:*" -l %1
```

Remember that a server can still deny the connection request, but the request will fail on the server, not on the client.

In the following example, transparent mapping is allowed to `sysD` at all security levels on the system, with the exception of the LID `SYS_PRIVATE`. To authorize all connection requests with the exception of those with LID `SYS_PRIVATE`, add the following pair of entries to **`LIDAUTH.map`**:

```
sysD:SYS_PRIVATE     %i
sysD:* %1
```

To add the entries, first enter

```
attradmin -A LIDAUTH -a -r sysD:SYS_PRIVATE -l %i
```

Then enter

```
attradmin -A LIDAUTH -a -r "sysD:*" -l %1
```

The reportscheme Service

reportscheme(1M) is a network service invoked by the Connection Server to determine what authentication scheme is associated with a given service on the server machine. If the server is to enforce authentication scheme invocation, a **reportscheme** line must appear in the port monitor administrative file for every port monitor that offers network services. If the **reportscheme** service is not entered in a port monitor's administrative file, applications on client machines cannot successfully query scheme information and will assume a NULL authentication scheme is associated with the service to which they are trying to connect.

NOTE

The **reportscheme** entry itself must not have an authentication scheme associated with it.

The following is a sample **reportscheme** entry in the port monitor administrative file **/etc/saf/tcp/_pmtab**. The entry is a single line. It is shown wrapped around for readability.

```
reportscheme::root:reserved:reserved::\x00020AD0C00B02510000000000000000::c::
/usr/sbin/reportscheme#reportscheme service - no authentication scheme
```

There must also be a **reportscheme** entry in the **/etc/services** file:

```
#service name      port number/transport
#
reportscheme       2768/tcp
```

with the following entry in **/etc/hosts**:

```
#host address      host name
#
192.11.2.81        elvis
```

A **`reportscheme`** entry in the `ticots` port monitor administrative file **`/etc/saf/ticots/_pmtab`** will look like this:

```
reportscheme::root:reserved:reserved::elvis.reportscheme::c::/usr/sbin/reportscheme
#reportscheme service - no authentication scheme
```

In this case, the entry in the **`/etc/net/ticots/services`** file will be:

```
#service name      service port
#
reportscheme       reportscheme
```

The entry in the **`/etc/net/ticots/hosts`** file will be:

```
#machine addr      machine name
#
elvis              elvis
```

The Connection Server Authentication Scheme File

The Connection Server authentication file or **`/etc/cs/auth`** is an optional file, maintained by the system administrator, that lists the authentication scheme and role associated with a particular host, service, network tuple. The system administrator does not need to (and in most cases will not) put information into this file. Typically, the Connection Server obtains the initial authentication information about a particular host, service, network tuple from the **`reportscheme`** service. The Connection Server retains this data in an internal cache so the **`reportscheme`** service will not be called in subsequent network requests for the same host, service, network tuple.

If, for any reason, the system administrator does not want the **`reportscheme`** service to be called for a particular host, service, network tuple, the authentication scheme information can be stored in **`/etc/cs/auth`**. When the Connection Server is started, it uses the information in **`/etc/cs/auth`** to initialize its internal cache.

The Connection Server authentication file is read only once when the Connection Server is started up. If the system administrator changes the file while the Connection Server is running, the command

```
cs -x
```

must be issued from the command line to tell the Connection Server to read the authentication file again. See **`cs(1M)`** for further information on **`cs`**.

NOTE

Administrators of server machines that do not offer the **`reportscheme`** service must inform administrators of client machines of changes to the authentication scheme of a service from NULL to another scheme, such as cr1. Then client machine administrators must either update the **`/etc/cs/auth`** file with the new scheme information and execute the **`cs -x`** command, or kill the **`cs`** daemon and then restart it so the internal cache will be rebuilt with the correct information.

To change the Connection Server authentication file, the system administrator should edit the file manually. The format of the file comprises lines of tab separated fields:

host service transport authentication scheme imposer role

When no scheme is required (sometimes referred to as a NULL scheme), the administrator indicates this by putting a dash (`-`) in the authentication scheme field. The “role” field indicates the client will act as either the responder (`r`) to the authentication process or as the imposer (`i`).

An example of a small authentication file is shown in Figure 4-2:

```
#host    service     transport     authentication scheme     imposer role

 pelham  echo_tcp    tcp           cr1                       r
 pelham  cu          tcp           -                         r
 pelham  uucico      tcp           cr1                       r
 pelham  date        tcp           -                         r
```

Screen 4-2. Example of a Small Authentication File

For more information on imposer role and authentication schemes, see Chapter 5, “cr1 Bilateral Authentication Scheme”.

The Connection Server Log File

The Connection Server logs information in the file **`/var/adm/log/cs.log`** on the client machine. A message is logged on startup.

Any time a connection request fails, the reason for failure is written to the log file. In addition, for each connection requested through `cs_connect`() or `dial`(), a message is logged containing the following data: time, date, user ID, group ID, process level (if MAC is installed), network service requested, name of server machine, and status of request (success or failure). As shown in the example in Screen 4-3.

```
05/13/92 13:07:45;   229; *** CONNECTION SERVER starting ***
05/13/92 13:09:16;   313; connection not permitted by /etc/idmap/attrmap/LIDAUTH.map
05/13/92 13:09:16;   313; Request by process uid<0> gid<5> at level USER_LOGIN
05/13/92 13:09:16;   313;   for service<cu> on host<hulk> FAILED
05/13/92 13:09:38;   468; Request by process uid<0> gid<5> at level USER_LOGIN
05/13/92 13:09:38;   468;   for service<cu> on host<hulk> SUCCEEDED
```

Screen 4-3. Sample Output from a Connection Server Log File

The number immediately following each date and time (such as `468` in the last line) is the process ID of the process that the Connection Server daemon spawned to handle the connection request.

The Connection Server Debug File

If the Connection Server is invoked with the debug option, the Connection Server daemon will write debug information to **`/var/adm/log/cs.debug`**. See **`cs(1M)`** for further information on **`cs`**. The command line used is:

```
/usr/sbin/cs -d
```

A sample output file is shown in Screen 4-4.

```
10:40:58;  4404; cs: Debugging turned on
10:40:58;  4404; cs: chdir to ROOT
10:40:58;  4404; gs: reading schemes from /etc/cs/auth
10:40:58;  4404; cs: created CS pipe: /etc/.cs_pipe
10:41:08;  4404; cs: conn done; client talks to server
10:41:08;  4404; cs: fd to read request from: 3
10:41:08;  4412; cs: connection server child forked
10:41:08;  4412; cs: request-type: DIAL_REQUEST
10:41:08;  4412; cs: fd to read from <3>
10:41:08;  4412; sr: netpath<NULL>
10:41:08;  4412; cs: dial Call structure set up as follows:
10:41:08;  4412; cs: baud<-1>
10:41:08;  4412; cs: speed<-1>
10:41:08;  4412; cs: modem<-1>
10:41:08;  4412; cs: dev_len<-1>
10:41:08;  4412; cs: c_iflag<6182>
10:41:08;  4412; cs: c_oflag<6148>
10:41:08;  4412; cs: c_cflag<445>
10:41:08;  4412; cs: c_lflag<48>
10:41:08;  4412; cs: c_line<0>
10:41:08;  4412; cs: line<NULL>
10:41:08;  4412; cs: telno<sfsup>
10:41:08;  4412; cs: service<cu>
10:41:08;  4412; cs: class<NULL>
10:41:08;  4412; cs: protocol<NULL>
```

Screen 4-4. Sample Output from a Connection Server Debug File (cs -d)

5
cr1 Bilateral Authentication Scheme

5
cr1 Bilateral Authentication Scheme

Introduction to the cr1 Bilateral Authentication Scheme

cr1 is an identification and authentication scheme that protects a system from unauthorized access. By default, cr1 uses DES encryption, and can also be referenced as cr1.des. Because of export restrictions on DES, cr1 can also use ENIGMA encryption. When using ENIGMA encryption, cr1 is referenced as cr1.enigma. Other than the underlying encryption algorithm used, all cr1 schemes behave identically.

The cr1 scheme is bilateral, which means it authenticates both client and server identities. Generally, it authenticates a connection established by a Connection Server on the client side and a port monitor on the server. When a cr1 exchange is complete, the client, as well as the server, can be certain of the other party's identity.

cr1 requires a system to store a cryptographic key for every protected system with which it needs to communicate. The key is a bit string known only to the principals in the exchange (the client and server); the string is used to encrypt and decrypt messages passed between the two principals. Typically, when a remote client attempts to access a local service protected by cr1, the cr1 scheme on the server engages the client in a sequence of exchanges involving the shared cryptographic key and one-time challenges. If the remote client responds appropriately to the server's challenge, the server can be certain that the remote client is authorized to access the service. If the server does not engage the client in the exchange, the client can engage the server. If the server responds appropriately to the client's challenge, the client can be certain that it is connecting to the desired server, not an imposter.

cr1 consists of the following components:

- an executable program, **`/usr/lib/iaf/cr1/scheme`**
- the **`/etc/iaf/cr1/keys`** file, which is a database of shared keys, optionally encrypted with a master key
- **`keymaster,`** a daemon process that manages the key database
- **`/var/iaf/cr1,`** a multilevel directory, containing the log file **`log`**

The cr1 privileged user can create a key to be shared by the local system and any remote machine on the network. The privileged user is the owner of the **`keys`** file. Once a shared key is created for the two principals, the key management daemon stores the key in the key database, then uses a cryptographic key—called the master key—to encrypt the keys in the database. The keys in the database are re-encrypted every time a new key is added.

Shared keys can also be created for users. For example, `bob` on `system1` can share a key with `system2` or with user `joe` on `system2.` When a key is created for a user, the name of the client user and the system to which the user wants access must be specified.

The privileged user can create a shared key for any local user. A non-privileged user can create shared keys for their individual login name.

When a user attempts to run a service on the client, the command that executes the service initiates the following sequence of actions:

1. The Connection Server on the client requests the **`reportscheme`** service from the port monitor on the server. **`reportscheme`** is an internal service that determines the scheme that is protecting a service, then passes the information to the client.
2. The **`reportscheme`** service returns a message informing the Connection Server that the service is protected by cr1.

NOTE

> Once the client receives the name of the scheme, it caches the information for future use, so **`reportscheme`** need not be called every time the Connection Server requests a connection to the service.

3. The Connection Server then requests a connection to the service from the port monitor on the server.
4. Both the Connection Server and the port monitor invoke cr1. On the client, the Connection Server calls cr1 to play the role of the “responder” in the authentication exchange. (cr1 on a client is called the responder because it responds to the server's requirement that cr1 be used.) On the server, the port monitor calls cr1 to play the role of the “imposer.” (cr1 on the server is called the imposer because it protects a local service by imposing the use of the cr1 protocol on the client machine.)
5. The responder searches various local databases and sends pertinent information to the server. Included in that information is the responder's effective identity (that of the local user) and the name of the client machine. Also included in the message is a unique token, which the responder sends as a challenge to cr1 on the server.
6. The responder's message is received by cr1 on the server (the imposer). After the imposer receives the responder's message, it retrieves the shared key from its key database, and verifies that the key properly decrypts the message. It then sends the responder a message that includes the responder's token; by returning the token, the imposer proves its identity to the responder. The imposer's message also includes another unique token, which the imposer sends to challenge the responder.
7. The responder receives the message and validates its contents, proving that the sender properly decrypted the first message using the key known only to the two parties. The responder then sends a third message that includes the unique token provided by the imposer.
8. The imposer receives the third message and verifies that the token it included in the previous message was successfully decrypted and returned, proving that the responder is an active participant in the conversation.

When the authentication exchange is successfully completed, the user on the client connects to the service on the server.

An Overview of cr1 Administration

cr1 works with a number of facilities and databases to provide an authenticated connection between client and server. These facilities and databases must be administered prior to creating a shared key for two principals in a cr1 exchange. The following is an overview of setting up a cr1-protected service, first on the client side, then on the server side. This overview assumes a TLI connection.

To administer cr1 for a service on the client side, you must do the following:

1. Set up **`/etc/net/`***transport***`/hosts`** to include the remote system's name and address. If the connection is over TCP/IP, use the **`/etc/hosts`** and the **`/etc/services`** files.
2. Set up **`/etc/net/`***transport***`/services`**.
    a. Add the service and its port address.
    b. Add **`reportscheme`** and its address.

    NOTE

    For information about the **`hosts`** and **`services`** files, see **`hosts(4)`** and **`services(4)`**.

    If the connection is not over TLI, the Connection Server searches for the address of the service in the BNU **`Systems`** and **`Devices`** files. See the discussion of BNU files later in this chapter.

3. Make sure the Connection Server and the key management daemon are running.

    NOTE

    An initialization script starts both the Connection Server and the key management daemon. If the Connection Server has been stopped manually, use the **`cs`** command to restart it. For more information about starting and stopping the Connection Server, see Chapter 4, “Administering the Connection Server”. Starting and stopping the key management daemon is described later in this section.

4. Add a shared key for the server to the client's key database.

To set up cr1 to protect the service on the server side, you must do the following:

1. Set up the port monitor's **`_pmtab`** file.
   a. Create an entry for the service and specify

      `cr1` **`-s`** *servicetag*

      in the *scheme* field.

   b. Create an entry for **`reportscheme`**. Leave the *scheme* field empty to mean the `NULL` scheme.

2. Set up ID mapping to include a database entry that maps the name of the user on the client to a local name. The command lines are as follows:

   ```
   idadmin -S cr1 -I M1@M2          (to initialize the scheme)
   idadmin -S cr1 -a -r *@client -l %1     (transparent mapping)
   ```

3. If the remote user's LID is to be mapped, set up the ID mapping database that maps LIDs. The command lines are as follows:

   ```
   attradmin -A RLID -I M1@M2          (initialize the scheme)
   attradmin -A RLID -a -r *@client -l %1  (transparent mapping)
   ```

NOTE

For information about administration of the ID mapping databases, see Chapter 6, “Administering ID Mapping”.

4. Add the shared key to the server's key database.

The following tasks are specifically part of cr1 administration:

1. Setting up the key database initially and managing shared keys
2. Creating and managing the master key
3. Stopping and starting the key management daemon
4. Optionally, adding cr1 to the Connection Server's **`/etc/iaf/serve.allow`** file on the client

All administrative tasks can be performed through the system administration menu interface. When you select an option from a menu, self-explanatory submenus and instructions lead you through the appropriate procedures.

Most tasks can be done from the cr1 menu. The screen below is the top-level cr1 menu. It can be displayed by typing the following command:

```
sysadm network_services/cr1
```

```
    IAF Scheme cr1 Key Management

add      -   Add an Entry to the Key File
modify   -   Modify an Entry in the Key File
remove   -   Remove an Entry from the Key File
set      -   Set the Master Key for the Key File
start    -   Start the Keymaster Daemon
stop     -   Stop the Keymaster Daemon
```

Screen 5-1. cr1 Menu

The command alternatives to the cr1 menu interface are listed in Table 5-1.

Table 5-1. Command Alternatives to the cr1 Menu

Task Description	Menu Item	Shell Command
Add shared keys to the **keys** file.	`add`	**`cryptkey -a`**
Modify a `keys` entry.	`modify`	**`cryptkey -c`**
Remove a `keys` entry.	`remove`	**`cryptkey -d`**
Set a master key.	`set`	**`keymaster -c`**
Start the **keymaster** daemon.	`start`	**`keymaster`**
Stop the **keymaster** daemon.	`stop`	**`keymaster -k`**

Registering the scheme with a port monitor is done through the top-level Service Access Facility menu, shown below. It can be reached by typing the following command:

```
sysadm ports
```

```
    Service Access Management

port_monitors    - Port Monitor Management
port_services    - Port Service Management
quick_terminal   - Quick Terminal Setup
tty_settings     - Terminal Line Setting Management
```

Screen 5-2. SAF Menu

You can bypass the menu system by issuing commands directly to the shell. The command alternative to editing the **_pmtab** file through the SAF menu is **pmadm** [see **pmadm(1M)]**.

Instructions for administering facilities with which cr1 interfaces, such as the Connection Server and ID Mapping can be found in Chapter 4, "Administering the Connection

Server” and Chapter 6, “Administering ID Mapping”, respectively. Because the menu interface is self-explanatory, the following instructions assume you are administering cr1 using shell commands.

Registering cr1 with a Port Monitor

To protect a local service with cr1, you must instruct the port monitor associated with the service to use cr1 to authenticate any remote client attempting to gain access. To do this, you must add `cr1` to the *scheme* field of the service's entry in the port monitor's **`_pmtab`** file.

To specify `cr1` in the *scheme* field of a file entry, enter the **`pmadm`** command and include

`-S “cr1 -s` *servicetag*”

on the command line.

NOTE

> For more information about port monitors and the **`_pmtab`** file, see **`pmadm(1M)`** and the chapter “Managing Ports” in the *System Administration* guide. This chapter includes specific instructions for editing **`_pmtab`** using the **`pmadm`** command.

Typically, cr1 in the role of the imposer is invoked with the **`-s`** option; however, a number of options are supported. For a complete list of options, see **`cr1(1M)`**.

Registering cr1 with the Connection Server

Nothing needs to be done on the client side to register cr1. Once the cr1 executable program is installed, the Connection Server automatically invokes `cr1` with the **`-r`** option whenever it receives a message from the server that cr1 is protecting the requested service. The **`-r`** option tells the program on the client that it is to play the role of the responder in the authentication exchange.

When cr1 on the client is called, it searches local databases and sends pertinent information to the server, which the port monitor passes as arguments to the server's local cr1. Included in the arguments are the name of the user on the remote system and the remote system's machine name, which cr1 uses to locate the shared key in its key database. Setting up the key database is described in Setting Up the Key Database on page 5-8.

Although cr1 does not need to be administered on the client, the client administrator has the option to specify a list of acceptable schemes in the Connection Server's **`/etc/iaf/serve.allow`** file. If cr1 is specified, the Connection Server will fail the connection request if the use of cr1 is not mandated by the server. By forcing the server to use cr1, the client can verify the server's identity.

Instructions for setting up the **/etc/iaf/serve.allow** file can be found under Chapter 4, “Administering the Connection Server”.

Managing the Daemon and the Master Key

Before you can administer the key database, the key management daemon must be running. Typically, the daemon is started by default when you boot the system.

When you administer the key database—whether to add a new shared key to the **keys** file, delete a key that is no longer needed, or modify an existing key—the command you enter calls the key management daemon, which performs the requested operation. If you are supplying a new shared key, for example, the daemon takes the shared key from the **cryptkey** command [see **cryptkey(1)**] and stores it in the **keys** file. The daemon then uses the master key to encrypt the keys in the **keys** file. Every time the daemon modifies the **keys** file, it re-encrypts the keys—using the master key.

Both the key management daemon and the master key are managed using the **keymaster** command [see **keymaster(1M)],** as described in the following sections.

Starting and Stopping the Daemon

The key management daemon is started automatically through an initialization script whenever you boot the system; however, you can do this only if there is no master key.

NOTE

If you have a master key, the **keymaster** daemon cannot be started automatically at boot time; **keymaster** must be started manually. You must also edit the **init** script **/etc/dinit.d/S69keymaster** and comment out the line which reads **keymaster -n**.

If your system needs to be re-booted frequently, and you are unable to attend the system during a re-boot, you may want to store the **keys** file in unencrypted form. To store the **keys** file in unencrypted form, you need to change the master key to NULL.

In certain circumstances—if there is a problem with the **keys** file, for example—you may want to stop the daemon manually. Enter the **keymaster** command as follows to stop the daemon:

```
keymaster -k
```

No key is required to stop the daemon; however, the operation fails if you are not the privileged user (the owner of the **keys** file).

To re-start the daemon, enter the **`keymaster`** command.

Creating a Master Key

When cr1 is first installed and the system is booted, an initialization script runs the **`key-master`** command, which starts the key management daemon. Because the **`keys`** file is empty at this point, the master key is NULL.

To create the master key, simply enter the following command:

```
keymaster -n -c
```

`keymaster` prompts you for the key, which can be any alphanumeric string between zero and eight characters in length.

When you enter the key, the **`keymaster`** command does not echo it on the screen. Instead, **`keymaster`** prompts you to enter the key a second time. If the first and second entries match, the daemon stores the master key. If the entries do not match, the operation fails and the master key remains unchanged.

Once you create a master key, the daemon takes it and stores it. (For security reasons, it stores the key in cleartext in its process address space, not in a file. An encrypted copy of the master key is stored in the **`keys`** file.) The daemon then uses the master key to encrypt the shared keys in the **`keys`** file and to re-encrypt them every time the file is modified.

To change a master key once you have created it, enter

```
keymaster -c
```

The system then prompts you to enter the old master key. Once you enter the old master key, it prompts you to enter, and then re-enter the new master key. If the entries do not match, the operation fails and the master key remains unchanged.

Setting Up the Key Database

As described earlier, shared keys are stored in a **`keys`** file, which is set up and maintained by the cr1 administrator. Whenever you administer the **`keys`** file, the key management daemon must be running.

Shared keys are stored in **`keys`** files on both the client and server machines. The cr1 administrator of the client must enter a key to be shared by the local and remote systems. The cr1 administrator of the server machine must enter the same key into its key database. The database entry and the syntax of the command used to create the entry differ on the client and server machines; however, the entry always includes the names of the two principals in the exchange and, of course, the shared key. The key must be entered into both the client and server databases locally; there is no service available to propagate cr1 keys on the network.

Typically, authentication is done at the system level. Authorization of individual users is managed locally. If there is a need to deny a user access to a service on a machine, the user is denied access locally, through the standard permissions mechanism.

Although we recommend restricting access through cr1 at the system level, access by individual users can be controlled through cr1, as well. Shared keys can be entered into the key database for individual users. If access is controlled through cr1 at the user level, users can enter and maintain their own keys, in the same way that they maintain their own login passwords in **`/etc/shadow`** [see **`shadow(4)`**].

The interface to the **`keys`** file is the **`cryptkey`** [see **`cryptkey(1)`**] command. The **`cryptkey`** command allows privileged and non-privileged users to add, delete, and modify a key shared by two principals in a cr1 authentication exchange. A non-privileged user must be the local principal for whom the key is being added, deleted, or modified. The privileged user is the owner of the **`keys`** file. A privileged user can create, delete, or modify keys for any user on the local system.

The **`cryptkey`** command has the following syntax:

> **`cryptkey [-a | -c | -d] [`***local_principal***`]`** *remote_principal*

For further information on the options, see **`cryptkey(1).`**

If **`cryptkey`** is entered without options, the **`-c`** option is assumed and an existing key shared by the principals is modified.

When a user enters the command to add or change a key, he or she is prompted to enter a new key; the key can be any alphanumeric string between zero and eight characters in length.

When a non-privileged user is deleting an old key, he or she is prompted to enter the old key. When changing a key, a non-privileged user is prompted to enter the old key, followed by the new key.

As with passwords, an administrator may need to change keys for users who do not remember their old keys, or remove keys belonging to users who no longer have access to the system. Therefore a privileged user is not required to supply the old key when changing or deleting a key.

Assume you want to create a shared key for `adam` on your system, named `eden,` and for `tom` on a remote host named `utopia.` You want to enter `serpent` as the key. On the client, you must do the following:

1. Enter

   ```
   cryptkey -a adam utopia!tom
   ```

   The system then prompts you to enter a key.

2. In response to the prompt, type `serpent.` The system then prompts you to enter the key a second time.

3. Enter `serpent` a second time. If the first and second entries match, the key is added to the database. If they fail to match, the operation quits.

To add the key into the server's key database, you would do the following:

1. Access the server and enter

   ```
   cryptkey -a tom eden!adam
   ```

   Again, the system prompts you to enter a key.

2. Enter the same key that was added to the client's database—in this case, `serpent`

   The system then prompts you to enter the key a second time.

3. Enter the key a second time. When you enter the key a second time, and the first and second entries match, the key is added to the server's key database.

Now the two principals have a shared key.

6
Administering ID Mapping

6
Administering ID Mapping

Introduction to ID Mapping Administration

ID Mapping is a facility that can be used to establish local identities for users on remote systems. It provides a common mechanism through which applications can identify remote users and control remote access to local resources.

The ID Mapping module consists of two routines that map remote users into local identities, plus a database from which the routines retrieve the relevant mapping information. The ID Mapping database includes two types of map files. One type contains entries that map user logins; the other contains entries that map the values of user attributes, such as UIDs and GIDs.

When a remote user attempts to access a service on your system, the port monitor receives the connection request. It uses an authentication scheme to validate the user; the scheme then calls the ID Mapping routines. One routine checks the login maps associated with the ID Mapping scheme, then maps the user to a login on the local system. The other routine checks the local system's attribute maps, then maps the values of user attributes on the remote system to the specified local values.

Both login mapping and attribute mapping are provided as part of a general mapping facility. Some applications may require that users be mapped by login, some by attributes, and some by both. On a system running the Enhanced Security Utilities, users are mapped both by login and by security level ID (LID). If the Least Privilege Module (LPM) is running, the ID Mapping administrator must be in the NET role and be a member of group `sys.` LPM is part of the Enhanced Security Utilities.

ID Mapping administration entails setting up and maintaining the ID Mapping database; however, before you set up the database, it is assumed you have installed the authentication schemes you intend to use.

NOTE

The cr1 authentication scheme is the only authentication scheme provided at this time—with the exception of the traditional login/ password scheme, which doesn't rely on ID Mapping. Unless otherwise stated, examples throughout the discussion of ID Mapping assume cr1 is the authentication scheme.

We recommend you administer your system in the following sequence:

1. Check that authentication scheme executable programs are installed in the proper directories; cr1 should be installed in **`/usr/lib/iaf/cr1.`**

2. Set up the ID Mapping database, following instructions in this section.
3. Register services with port monitors and associate them with authentication schemes through the Service Access Facility. See the chapter “Managing Ports” in the *System Administration* guide for more information on registering services.
4. Update the ID Mapping database as needed.

If you enable a facility called “user-controlled mapping,” non-privileged users can help you maintain login maps. When user-controlled mapping is enabled, a user with logins on both the local and a remote system can access the local system and add a database entry that maps their own remote login to a local login. User-controlled mapping is described in “Enabling and Disabling User-Controlled Mapping” on page 6-14.

NOTE

The administrator of user-controlled mapping databases must be in group `sys`. whether or not the Enhanced Security Utilities are installed and running.

Administering the ID Mapping database can be done using either the **`sysadm`** command or ID Mapping commands. When you use **`sysadm,`** self-explanatory submenus and instructions lead you through the appropriate procedures. The screen below is the top-level Name Mapping menu. It can be displayed on the screen by typing:

```
sysadm name_map
```

```
    Name Mapping Administration

add              - Add a Name Mapping File
list             - List All Schemes with Defined Name Mapping Files
remove           - Remove a Name Mapping File
mappings         - Name Mapping Administration
```

Screen 6-1. Name Mapping Menu

You can bypass the menu system by issuing commands directly to the shell. The command alternatives to the name mapping menu options are listed below:

Table 6-1. Command Alternatives to the Name Mapping Menu

Task Description	Menu Item	Shell Command
Add a new name mapping file.	`add`	`idadmin` **`-S`** *scheme* **`-I`** *descr*
List all schemes with defined name-mapping files.	`list`	`idadmin`
Remove a name-mapping file.	`remove`	`idadmin` **`-S`** *scheme* **`-D`**

When you select `mappings` from the menu, the following sub-menu is displayed:

```
    Name Mapping Administration

add              - Add a Mapping
list             - List Specific Mappings
remove           - Remove One or More Mappings
enable           - Enable User-Controlled Mapping
disable          - Disable User-Controlled Mapping
check            - Check For Mapping File Problems
fix              - Fix Mapping File Problems
```

Screen 6-2. mappings sub-menu for Name Mapping Administration

Command alternatives to the second-level menu options are listed in Table 6-2.

Table 6-2. Command Alternatives to the Second-Level Name Mapping Menu

Task Description	Menu Item	Shell Command
Add a name mapping to a file.	`add`	`idadmin` **`-S`** *scheme* **`-a -l`** *logname* **`-r`** *g_name*
Check mapping consistency.	`check`	`idadmin` **`-S`** *scheme* **`-c`**
Fix mapping inconsistencies.	`fix`	`idadmin` **`-S`** *scheme* **`-f`**
List entries in a map.	`list`	`idadmin [`**`-S`** *scheme* `[`**`-l`** *logname*`]]`

Table 6-2. Command Alternatives to the Second-Level Name Mapping Menu (Cont.)

Task Description	Menu Item	Shell Command
Disable user-controlled mapping.	`disable`	`idadmin` **`-S`** *scheme* **`-s`**
Remove a specific map entry.	`remove`	`idadmin` **`-S`** *scheme* **`-d -l`** *logname* `[`**`-r`** *g_name*`]`
Enable user-controlled mapping.	`enable`	`idadmin` **`-S`** *scheme* **`-u`**

Attribute mapping is done from a separate menu and submenu. The top-level attribute mapping menu can be displayed by typing:

`sysadm attr_map`

The following is the top-level attribute mapping menu:

```
    Attribute Mapping Administration

add              - Add an Attribute Mapping File
list             - List All Attributes with Defined Attribute Mapping Files
remove           - Remove an Attribute Mapping File
mappings         - Attribute Mapping Administration
```

Screen 6-3. Attribute Mapping Menu

Command alternatives to the attribute mapping menu options appear below:

Table 6-3. Command Alternatives to the Attribute Mapping Menu

Task Description	Menu Item	Shell Command
Create a new attribute map file.	`add`	`attradmin` **`-A`** *attrname* **`-I`** *descr*
List all attribute mapping files.	`list`	`attradmin`
Delete an attribute map file.	`remove`	`attradmin` **`-A`** *attrname* **`-D`**

When you select `mappings` from the menu, the following sub-menu is displayed:

```
    Attribute Mapping Administration

add              - Add a Mapping
list             - List Specific Mappings
remove           - Remove One or More Mappings
check            - Check for Mapping File Problems
fix              - Fix Mapping File Problems
```

Screen 6-4. mappings sub-menu for Attribute Mapping Administration

Command alternatives to the second-level menu options are listed in Table 6-4.

Table 6-4. Command Alternatives to the Second-Level Attribute Mapping Menu

Task Description	Menu Item	Shell Command
Add an entry to an attribute map file.	`add`	`attradmin` **`-A`** *attrname* **`-a -l`** *localval* **`-r`** *remoteval*
Check mapping consistency.	`check`	`attradmin` **`-A`** *attrname* **`-c`**
Fix mapping inconsistencies.	`fix`	`attradmin` **`-A`** *attrname* **`-f`**
List entries in a map.	`list`	`attradmin [`**`-A`** *attrname* `[`**`-l`** *localval*`]]`
Remove a specific map entry.	`remove`	`attradmin` **`-A`** *attrname* **`-d -l`** *localval* `[`**`-r`** *remoteval*`]`

Because menus are self-explanatory, instructions in this guide for setting up and maintaining the ID Mapping database assume the use of shell commands.

Setting Up Login Maps

There are two types of login maps: one that contains map entries specified by the system administrator, and one that contains user-specified entries.

It is conceivable that you might have two or more sets of login maps on your system. Authentication schemes require a specific ID Mapping scheme. What this means, essentially, is that entries in maps that support a particular authentication scheme must have a specific format, which is dictated by the authentication scheme. If your system supports two authentication schemes, for example, and each scheme requires map entries to have a unique format, you will need to have two administrator-controlled login maps on your system.

The term “ID Mapping scheme” refers to the format of a set of maps. Generally, there is a one-to-one correspondence between ID Mapping schemes and authentication schemes, such that one set of maps supports one authentication scheme. When there is a one-to-one correspondence, the name of the ID mapping scheme and the name of the authentication scheme can be the same.

At the present time, cr1 is the only authentication scheme provided as part of the OS—with the exception of the traditional login/password scheme, which doesn't rely on ID Mapping. The ID Mapping scheme that supports the cr1 authentication scheme is also called cr1.

Administrator-specified map files are named **`idata`**. User-specified map files are named **`uidata`**. Both login map files are stored in the **`/etc/idmap`** directory, in a subdirectory named for the ID Mapping scheme. Below are the full pathnames of the map files that support cr1.

```
/etc/idmap/cr1/idata
/etc/idmap/cr1/uidata
```

NOTE

The name of the ID Mapping scheme is hard coded in the authentication scheme. If, in the future, your system supports an authentication scheme in addition to cr1, check the documentation that accompanies the scheme software to determine the name you need to assign to the maps that support the scheme.

Examples throughout the discussion of ID Mapping assume you are setting up maps to support cr1.

The following sections describe setting up and maintaining **`idata`** and **`uidata`** map files.

Administering an idata File

Each entry in **`idata`** maps a user login on a remote system to a login on the local system. Below is a sample **`idata`** file:

```
!M1@M2.M3.M4.M5
*@*.sf.buu.com %1
*@*.tm.buu.com guest
ilya@*.*.*.* ilya
```

Note that the example includes regular expressions, which are used to map remote users transparently or to map multiple users to a general login. The use of regular expressions in the **`idata`** file is explained in detail in “Adding an Entry to an idata File” on page 6-8.

All entries in an **`idata`** file are specified by the system administrator. When a remote user has an entry in **`idata`**, he or she can access a service on the local system and assume the user identity defined by the administrator.

The **`idadmin`** command is the command interface to **`idata.`** It allows a privileged user to do the following:

- create and delete an **`idata`** file for a particular ID Mapping scheme
- add and delete user entries in an **`idata`** file
- display information about the **`idata`** files
- check the consistency of file entries and fix inconsistencies
- enable and disable user-controlled mapping.

NOTE

> If the Least Privilege Module (LPM) is running, the ID Mapping administrator must be in the NET role and be a member of group `sys.` LPM is part of the Enhanced Security Utilities.

The **`idadmin`** command has the following syntax:

```
idadmin [-S scheme [-l logname]]
idadmin -S scheme -a -l logname -r g_name
idadmin -S scheme -d -l logname [-r g_name]
idadmin -S scheme -I descr
idadmin -S scheme [-Duscf]
```

See **`idadmin(1M)`** for an explanation of the options accepted by the **`idadmin`** command.

The options and command syntax required to execute a particular operation are described in the following sections.

Setting Up the idata File

When you want to set up an **`idata`** map file for a particular ID Mapping scheme, you must enter the **`idadmin`** command with the ID Mapping scheme name and specify the format that the global name should take in all file entries.

The **`idadmin`** command has the following syntax when used to set up a new file:

```
idadmin -S scheme -I descr
```

where *scheme* is the name of the ID Mapping scheme, and *descr* is a string called a format descriptor. The string specifies the form that the name of the remote user must take in the **`idata`** map file.

For example, cr1 expects global names to consist of the user's login, followed by the character `@`, then the system name. The global name of a user with the login `jeff` on a machine called `moon` would be

```
jeff@moon
```

The format descriptor itself consists of field numbers, the letter M, indicating the fields are mandatory, and the character(s) that are used as field separators.

The file descriptor you would enter to set up an **`idata`** file for cr1 would be

```
M1@M2
```

The field numbers indicate the order of significance of the fields, where higher numbered fields are more significant. In the example, `M2` means that the entity to be specified in the second field is of greater significance on the network than the entity specified in the first field. In this case, the system is of greater significance than the user. The letter `M` indicates that the fields are required when specifying `g_name.`

Assume that BNU services are registered with the authentication scheme cr1. To set up an **`idata`** file that would map remote users to the local system and give them access to BNU, you would enter the following command:

```
idadmin -S cr1 -I M1@M2
```

When the **`idadmin`** command in the example is executed, it creates the file **`/etc/idmap/cr1/idata`**. The first line of the file consists of the format descriptor. Except for the format descriptor, the file is empty until user entries are added, as described in the following section.

Adding an Entry to an idata File

To map a remote user to a login on the local system, add an entry for the remote user to the appropriate **`idata`** file. The local login must be in the **`/etc/passwd`** file on the local system.

When you add a user entry to **`idata,`** **`idadmin`** has the syntax

```
idadmin -S scheme -a -l logname -r g_name
```

where *scheme* is the name of the ID mapping scheme, *g_name* is the global name of the remote user, and *logname* is the local login. For example, given the format descriptor `M1@M2,` the following entry maps `jeff` on machine `moon` to the local login `guest:`

```
idadmin -S cr1 -a -l guest -r jeff@moon
```

You can set up transparent mapping of logins by using regular expressions in the *g_name* field and the `%` character in the *logname* field. The special characters supported by the **`idadmin`** command are explained on **`idadmin(1M).`**

File entries are sorted so that an entry that maps a login explicitly is found in a search before entries that implement transparent mapping. Likewise, entries that map logins transparently are sorted based on the position of the regular expression in the global name. Entries with a regular expression in place of a remote user login appear in the file before entries with a regular expression in place of a system name.

Examples

Assume the ID Mapping scheme is `idmp` and *g_name* has the form `M1@M2.M3`, where `M1` indicates the user's login. The following command line transparently maps all users on the remote system `man` in the domain `moon`. All users are mapped to the local login `guest`.

```
idadmin -S idmp -a -l guest -r “*@man.moon”
```

NOTE

To protect characters that are meaningful to the shell, enclose global names in quotes.

If you enter `*` in place of the remote user name and `%1` in place of *logname* on the **`idadmin`** command line, you map all remote logins to the same values indicated in `M1.` Assume the authentication scheme is called cr2 and the format descriptor is `M2:M1.` If you want to map all logins on `pluto` to identical logins on your machine, you would enter

```
idadmin -S cr2 -a -l %1 -r “pluto:*”
```

In this example, user `bob` on the remote system would be mapped to `bob` on the local system; `johnd` on the remote system would be mapped to `johnd`, and so on.

In the next example, assume you want to map all logins on remote system `mars` to identical logins on your system, with the exception of the remote login `guest.` Assuming cr1 is the authentication scheme, the map must contain the following pair of entries:

```
guest@mars %i
*@mars %1
```

To add the entries to the map, first enter

```
idadmin -S cr1 -a -l %i -r guest@mars
```

Then enter

```
idadmin -S cr1 -a -l %1 -r “*@mars”
```

These entries could have been added in reverse order. The **`idadmin`** command ensures that the most specific entry appears first in the file. If you enter `%i` in place of *logname* on the **`idadmin`** command line, the remote user specified by the **`-r`** would be rejected.

Other characters can appear in a field containing an asterisk. For example, all remote system names beginning with `ux` will match the pattern `ux*`. The following command adds transparent mapping for all users on machines with names being with `ux`.

```
idadmin -S cr1 -a -l %1 -r “*@ux*”
```

In addition, to exclude all `guest` users on these remote systems, enter

```
idadmin -S cr1 -a -l %i -r “guest@ux*”
```

The contents of an **`idata`** file with these four entries would be

```
M1@M2
guest@mars %i
*@mars %1
guest@ux* %i
*@ux* %1
```

When an **`idata`** file is searched for a global name, it is scanned sequentially. Therefore the ordering of global names in this file is critical.

Global names are sorted on the highest numbered field first. Entries with explicit values in this field appear first in the file. Entries with regular expressions in this field appear next and are sorted from most specific to least specific. For example, the remote system name `ux*` is more specific than the remote system name `u*` and therefore would appear first in the file.

If two or more entries are equally specific, the specificity of the next lower numbered field is examined. Fields are examined from highest to lowest until the global names can be differentiated.

NOTE

Remote IDs should not be mapped to **`uucp`**. Instead, they should be mapped to **`nuucp`** or some other login used exclusively for data transfer.

Deleting an Entry in an idata File

To delete an entry in **`idata,`** enter **`idadmin`** with the following syntax:

`idadmin` **`-S`** *scheme* **`-d -l`** *logname* `[`**`-r`** *g_name*`]`

where *scheme* is the name of the ID mapping scheme, *g_name* is the global name of the remote user, and *logname* is the local login.

The use of **`-r`** is optional. If you enter the command without the **`-r`** option, every entry associated with the local login is deleted. If *g_name* is specified with the **`-r`** option, only the entry that maps the specified remote user to the specified local login is deleted.

For example, to delete all entries in **`/etc/idmap/cr1/idata`** that map remote users to the local login `guest,` you would enter

```
idadmin -S cr1 -d -l guest
```

Deleting the idata and uidata Files

To delete the **`idata`** and **`uidata`** files (removing a *scheme*), enter the **`idadmin`** with the syntax

```
idadmin -S scheme -D
```

where *scheme* is the name of the ID Mapping scheme to be deleted. This command removes the following files:

```
/etc/idmap/cr1/idata
/etc/idmap/cr1/uidata
```

Checking Files and Fixing File Inconsistencies

Periodically you should run a check on the **idata** files and correct any problems that might exist. The **idadmin** command provides options that allow you to check a file, correct entries with syntax errors, sort entries that are out of order, and delete entries for unknown local logins.

NOTE

Deleting an entry does not necessarily invalidate the remote user login. The remote login might be matched by a regular expression in another entry further down in the file. To insure that the remote login is invalidated, change the local name to `%i`.

If you use the ID Mapping commands to update the mapping databases, inconsistencies will occur only if the ID Mapping files get out of sync with the password file.

To check the consistency of the file, enter the **idadmin** command with the **-c** option, as follows:

```
idadmin -S scheme -c
```

where *scheme* is the name of the ID Mapping scheme.

The report generated by the **idadmin -c** is shown in Screen 6-5.

```
Error on line number 2: Mandatory field missing
Error on line number 4: Duplicate entry
Error on line number 5: Line out of order
Error on line number 7: Mandatory field missing
Error on line number 8: Line out of order
Error on line number 9: Line out of order
Error on line number 10: Bad transparent mapping
Error on line number 15: Line out of order
Error on line number 18: Unknown mapped user
9 error(s) was (were) found in system map
```

Screen 6-5. Sample Output from idadmin -c

To correct problems (such as the ones shown above) in the file, enter the command with the **-f** option, as in the following syntax line:

```
idadmin -S scheme -f
```

When **`-f`** is entered, the command gives you the option to delete, skip, or change invalid entries. It also sorts entries in the file that are out of order. By entering the **`idadmin`** command with the **`-c`** option again will allow you to verify that all changes were made correctly.

Displaying Information

`idadmin` can be used to display information about the **`idata`** files on the system, or about the entries in a particular **`idata`** file. When used to display information, the command has the following syntax:

```
idadmin [-S scheme [-l logname]]
```

where *scheme* is the name of the ID Mapping scheme, and *logname* is a local login.

If **`idadmin`** is entered without options, it displays the names of all the ID Mapping schemes on the system.

If you enter **`idadmin`** with a scheme name only, the command displays the scheme's **`idata`** file. If you enter **`idadmin`** with a scheme name and a local login, the command displays all the entries that map to that particular login.

Administering a uidata File

Like an **`idata`** file, a **`uidata`** file maps remote user logins to local logins; however, entries in a **`uidata`** file are specified by users themselves, not the system administrator.

A sample **`uidata`** file is shown in Screen 6-6.

```
!M1@M2
joe@ulysses jfl
mike@alpha mickey
mike@beta mickey
```

Screen 6-6. Sample uidata File

NOTE

Unlike **`idata,`** the administrator-controlled login map file, **`uidata,`** cannot be used to set up transparent mapping; the use of regular expressions in **`uidata`** entries is not permitted.

When you enable user-controlled mapping, a user with logins on both a remote system and the local system can access the local system and make an entry in **uidata** that maps his or her remote login to a local login. By enabling user-controlled mapping and instructing users to update the database, an administrator can distribute the workload and minimize the administrative overhead. For more information about user-controlled mapping, see “Enabling and Disabling User-Controlled Mapping” on page 6-14.

The **uidadmin** command is the command interface to **uidata.** When entered by a user, assuming user-controlled mapping is enabled, the non-privileged user can do the following:

- add and delete his or her entries
- display his or her entries.

An administrator can use the **uidadmin** command to

- display all the entries in a **uidata** file
- check the consistency of file entries and fix inconsistencies
- add or delete an entry on behalf of a non-privileged user

NOTE

The administrator of user-controlled mapping must be in group `sys`. This is required whether or not the Enhanced Security Utilities are installed and running.

The **uidadmin** command has the following syntax:

```
uidadmin [-S scheme [-l logname]]
uidadmin -S scheme -a [-l logname] -r g_name
uidadmin -S scheme -d -l logname [-r g_name]
uidadmin -S scheme [-cf]
```

See **uidadmin(1)** for an explanation of the options accepted by the **uidadmin** command.

The options and command syntax required to execute a particular operation are described in the following sections.

Setting Up the uidata File

A **uidata** file is set up for a particular authentication scheme when you set up the scheme's **idata** file, using the **idadmin** command. For example, to set up an **idata** file for `cr1,` you enter the following:

```
idadmin -S cr1 -I M1@M2
```

When the command executes, a **uidata** file is created in **/etc/idmap/cr1,** along with an **idata** file; however, users cannot add entries to the file, nor will the name mapping function attempt to read the file, until user-controlled mapping is enabled, as described in the following section.

Enabling and Disabling User-Controlled Mapping

When user-controlled mapping is disabled, every time a remote user attempts to access the local system, an internal routine searches the appropriate **`idata`** file for a map entry for that user. When user-controlled mapping is enabled, the **`uidata`** file is searched first. Because any entry in **`uidata`** relevant to the remote user is found before an entry in **`idata,`** the entry in **`uidata`** takes precedence over the **`idata`** entry. In this way, the user-specified mapping overrides an administrator-specified mapping for that user. When user-controlled mapping is disabled, **`uidata`** is not searched at all.

Whenever maps for a new ID Mapping scheme are set up, user-controlled mapping is disabled. Before enabling user-controlled mapping, create local logins for your remote users.

To enable user-controlled mapping for a particular authentication scheme, you enter **`idadmin`** with the **`-u`** option, as in the following syntax line:

> **`idadmin -S`** *scheme* **`-u`**

where *scheme* is the name of the ID Mapping scheme. This command activates the `USER` mode of the authentication scheme.

To disable user-controlled mapping, you enter **`idadmin`** with the **`-s`** option, as in the following line:

> **`idadmin -S`** *scheme* **`-s`**

This command activates the `SECURE` mode of the authentication scheme. When **`idadmin`** is entered without options, it lists all schemes and indicates whether each is in `USER` or `SECURE` mode. A scheme in `SECURE` mode has user-controlled mapping disabled.

Adding an Entry to a uidata File

When user-controlled mapping is enabled, non-privileged users with logins on the local system can map their own logins on a remote system to their own local logins. A non-privileged user cannot add entries for other users.

When used to add an entry to a **`uidata`** file, the **`uidadmin`** command has the syntax.

> `uidadmin` **`-S`** *scheme* **`-a`** `[`**`-l`** *logname*`]` **`-r`** *g_name*

where *scheme* is the name of the ID mapping scheme, *g_name* is a global name that includes the remote login, and *logname* is the local login. A non-privileged user is permitted to map a remote login only to his or her local login; if the **`-l`** option is omitted, the user's local login is assumed.

NOTE

> If the administrator enters the **`uidadmin`** command to add an entry on behalf of a non-privileged user, the **`-l`** option is required.

In the following example, a user named Mike has logins on two systems. His login on a machine called `wizard` is `michael;` his login on a machine called `zooey` is `mike.` Mike wants to map his login on `wizard` to his login on `zooey.` The service he wants to access on `zooey` is associated with the `cr1` authentication scheme. To set up mapping from `wizard` to `zooey,` Mike would access `zooey` and enter the following command:

```
uidadmin -S cr1 -a -r michael@wizard
```

Deleting an Entry in a uidata File

When user-controlled mapping is enabled, any non-privileged user may choose to delete an entry mapping his or her login; however, a non-privileged user cannot delete entries made by another user.

If a user also has a map entry in the administrator-controlled **`idata`** file, he or she will be mapped to the login specified in that entry once the **`uidata`** entry has been deleted.

When used to delete an entry, the **`uidadmin`** command has the following syntax:

```
uidadmin -S scheme -d -l logname [-r g_name]
```

where *scheme* is the name of the ID Mapping scheme, and *logname* is the local login.

The use of *g_name* is optional. If *g_name* is omitted, every entry that maps a remote login to the specified local login is deleted from the **`uidata`** file. If *g_name* is specified, only the entry that maps that global name to the local login is deleted.

If user Mike wants to delete the entry that maps his login `michael` on `wizard` to `mike` on `zooey`, he would enter the following:

```
uidadmin -S cr1 -d -l mike -r michael@wizard
```

Checking Files and Fixing File Inconsistencies

Because there is no need for your users to edit the **`uidadmin`** file directly, there is little danger that the file will become cluttered with entries containing syntax errors; however, you should run a check on the file periodically and correct any problems that might exist. The **`uidadmin`** command provides options that allow an administrator to check the file, correct entries with syntax errors, and delete entries that map to unknown logins.

To check the consistency of the file, enter **`uidadmin`** with the following options:

```
uidadmin -S scheme -c
```

where *scheme* is the name of the ID Mapping scheme.

To correct the problems in the file, enter the command with the following options:

```
uidadmin -S scheme -f
```

When the **`-f`** option is entered, **`uidadmin`** gives you the opportunity to replace entries that contain syntax errors and to delete entries associated with logins that do not exist in **`/etc/passwd.`** It then sorts entries that are out of order. Entering **`uidadmin`** with the **`-c`** option again will verify that all changes were done correctly.

Displaying Information

Both the system administrator and a non-privileged user may use the **`uidadmin`** command to display information. When used to display information, the command has the following syntax:

```
uidadmin [-S scheme [-l logname]]
```

where *scheme* is the name of the ID Mapping scheme, and *logname* is a local login.

If *uidadmin* is entered without options, it displays the names of all the ID Mapping schemes on the system. If an administrator enters **`uidadmin`** with a scheme name only, the command displays the **`uidadmin`** file associated with the specified scheme. If the administrator enters **`uidadmin`** with a scheme name and a local login, the command displays all entries that map remote logins to the specified local login.

If a non-privileged user enters **`uidadmin`** with only a scheme name, the command displays only the entries in **`uidata`** associated with his or her local login.

Setting Up Attribute Maps

Attribute map files map the values of user attributes on a remote system to attribute values on the local system. Most likely, an authentication scheme that maps user attributes will map such attributes as UID, GID, and LID.

Attribute maps are created in the **`/etc/idmap/attrmap`** directory. Generally, you'll name each file for the attribute it maps.

NOTE

The names of the attribute map files must match the name used in the authentication scheme. For example, on a system on which the Enhanced Security Utilities are installed and running, cr1 requires a file that maps remote LIDs called **`RLID.map.`** If your system supports an application that uses attribute mapping, check the documentation that accompanies the scheme software to determine the names you should give the map files.

An example of a GID map is shown in Screen 6-7.

```
M1:M2
10:sysA 20
1:sysB 1
```

Screen 6-7. Example of a GID Map

Each entry in the map file maps one value to another. Note that the first entry in the sample file maps the GID value of 10 to the value of 20 on the local system. With this entry in the file, any user with GID 10 on a remote system who accesses a service on the local system remotely is mapped to GID 20 on the local system.

Attribute maps support transparent mapping. By using regular expressions when specifying attribute values, an administrator can set up transparent mapping such that a number of attribute values on the remote system are mapped with a single file entry. Transparent mapping is described in detail in “Adding an Entry to an Attribute Map” on page 6-18.

The **`attradmin`** command is the command interface to the attribute maps. It allows a privileged user to do the following:

- create and delete an attribute map file
- add and delete map entries in a file
- display file contents
- check the consistency of file entries and fix inconsistencies

NOTE

If the Least Privilege Module (LPM) is running, the ID Mapping administrator must be in the NET role and be a member of group `sys.` LPM is part of the Enhanced Security Utilities.

The **`attradmin`** command has the following syntax:

```
attradmin [-A attrname [-l localval]]
attradmin -A attrname -a -l localval -r remoteval
attradmin -A attrname -d -l localval [-r remoteval]
attradmin -A attrname -I descr
attradmin -A attrname [-Dcf]
```

See **`attradmin(1M)`** for an explanation of the options accepted by the **`attradmin`** command.

The options and command syntax required to execute a particular operation are described in the following sections.

Setting Up an Attribute Map

If you want to set up an attribute map file, you must enter the **`attradmin`** command with the attribute map filename and specify the format the remote value should take in all map entries.

When the **`attradmin`** command is used to set up a new file, it has the following syntax:

attradmin **-A** *attrname* **-I** *descr*

where *attrname* is the name of the attribute, and *descr* is a string called a format descriptor. The format descriptor specifies the format in which the remote attribute value must be entered when a map entry is created.

If you're using cr1 as your authentication scheme, the format descriptor you need to enter is `M1@M2.`

NOTE

At this time, the cr1 authentication scheme is the only authentication scheme provided—with the exception of the traditional login/password scheme, which doesn't rely on ID Mapping. Unless otherwise stated, examples throughout the discussion of ID Mapping assume cr1 as the authentication scheme.

Field numbers indicate the order of significance of the fields. Higher numbered fields are the more significant. `M2` in the descriptor `M1@M2` means that the second field contains an entity of greater significance on the network than does the first field. The letter `M` indicates that the fields are required. The `@` symbol is a field separator. For example, given the format descriptor `M1@M2,` UID 104 on the remote system `venus` would be specified as `104@venus.`

To set up a map file that maps UIDs on remote systems to UIDs on the local system, you might enter the following command line:

```
attradmin -A uid -I M1@M2
```

Once the command executes, the file **`uid.map`** appears in **`/etc/idmap/attrmap`**, and the format descriptor appears in the first line of the file.

Adding an Entry to an Attribute Map

To add an entry that maps a remote attribute value to a local value, you enter **`attradmin`** with the syntax

attradmin **-A** *attrname* **-a -l** *localval* **-r** *remoteval*

where *attrname* is the name of the attribute, *remoteval* is the combination of an attribute value on the remote system and the remote system name, and *localval* is the attribute value on the local system. For example, the following command line maps GID 10 on the remote system `moon` to a GID 20 on the local system:

```
attradmin -A gid -a -l 20 -r 10@moon
```

Once the command executes, any user with a GID of 10 on `moon` has a GID of 20 on the local system.

You can set up transparent mapping of attributes by using regular expressions in the *remoteval* and special characters in the *localval* field. The characters supported by the **`attradmin`** command are explained on **`attradmin(1M).`**

File entries are sorted so an entry that maps a value explicitly is found in a search before entries that implement transparent mapping. Likewise, entries that map values transparently are sorted based on the position of the regular expression in *remote_attr*. Entries with a regular expression in place of a remote attribute value appear in the file before entries with a regular expression in place of a system name.

The **`attradmin`** command sorts entries containing regular expressions in *remote_attr* in the same way the **`idadmin`** command sorts entries containing regular expressions in the global name, with one addition: fields with regular expressions containing brackets, `[@,` are considered more specific than fields with an asterisk and therefore will precede fields containing an asterisk. See "Adding an Entry to an idata File" on page 6-8.

Examples

The following command line transparently maps all GIDs on the remote system `mars` to GID 10 on the local system.

```
attradmin -A gid -a -l 10 -r "*@mars"
```

NOTE

To protect characters that are meaningful to the shell, enclose *remoteval* in quotes.

Given the format descriptor `M1@M2,` the following command line maps all UID values on the remote system `mars` to identical values on the local system.

```
attradmin -A uid -a -l %1 -r "*@mars"
```

In this example, UID 101 on `mars` is mapped to UID 101 on the local machine, UID 102 is mapped to UID 102, and so on.

By using a regular expression in place of a remote machine name, you can map values on every machine on the network with access to your system to identical local values. The following command line maps all remote UIDs to the same UIDs on the local system:

```
attradmin -A uid -a -l %1 -r "*@*"
```

Deleting an Entry in an Attribute Map

To delete an entry in an attribute map, enter **`attradmin`** with the syntax

```
attradmin -A attrname -d -l localval [-r remoteval]
```

where *attrname* is the name of the attribute, *remoteval* is the combination of an attribute value on the remote system and the remote system name, and *localval* is the attribute value on the local system.

The use of **`-r`** is optional. It you enter the command without the **`-r`** option, every entry that maps a value to the specified local value is deleted. If you include the **`-r`** option, only the entry that maps the specified remote value to the specified local value is deleted.

To delete an entry that maps GID 10 on a remote system `moon` to GID 20 on the local system, you might enter the following command line:

```
attradmin -A gid -d -l 20 -r 10@moon
```

NOTE

Deleting an entry does not necessarily invalidate the remote user login. The remote login might be matched by a regular expression in another entry further down in the file. To insure that the remote login is invalidated, change the local name to `%i`.

Deleting an Attribute Map

To delete an attribute map, enter **`attradmin`** with the syntax

```
attradmin -S attrname -D
```

where *attrname* is the name of the attribute to be deleted.

Checking Files and Fixing File Inconsistencies

Periodically you should run a check on the attribute map files and correct any problems that might exist. The **`attradmin`** command provides options that allow you to check a file, correct entries with syntax errors, and sort entries that are out of order.

NOTE

`attradmin` does not check the validity of the attributes it maps.

To check the consistency of the file, enter **`attradmin`** with the following syntax:

```
attradmin -A attrname -c
```

where *attrname* is the name of the attribute.

If the file contains errors, the command displays output similar to the output shown in Screen 6-8.

```
Error on line number 2: Mandatory field missing
Error on line number 4: Duplicate entry
Error on line number 5: Line out of order
Error on line number 7: Mandatory field missing
Error on line number 8: Line out of order
Error on line number 9: Bad transparent mapping
Error on line number 10: Bad transparent mapping
Error on line number 15: Line out of order
8 error(s) found in attribute map
```

Screen 6-8. Sample Output from attradmin -A attrname -c

To correct the problems in the file, enter the command with the **`-f`** option, as in the following syntax line:

`attradmin` **`-A`** *attrname* **`-f`**

When the **`-f`** option is entered, **`attradmin`** gives you the opportunity to replace entries that contain syntax errors. It then sorts entries that are out of order.

Entering **`attradmin`** with the **`-c`** option again will verify that all errors were fixed correctly.

Displaying Information

`attradmin` can be used to display information about all the attribute map files on the system, as well as the contents of a specific map file.

When used to display information, **`attradmin`** has the following syntax:

`attradmin [`**`-A`** *attrname* `[`**`-l`** *localval*`]]`

If **`attradmin`** is entered without options, it displays the names of the attribute map files on the system.

If you enter **`attradmin`** with an attribute map filename only, it displays the contents of the file. If you enter **`attradmin`** with an attribute map filename and a local attribute value, the command displays all file entries that map the values of the specified remote attribute to the specified local value.

Setting Up a LID Map on a System Running the ES Utilities

For network communication to be possible in an environment in which the Enhanced Security (ES) Utilities are installed and running, both sides of the network connection must have the same security level. To ensure this, LIDs on a remote machine must be mapped, through an attribute map file, to their equivalents on the server. When a client makes a connection request, the server receives the LID of the remote process with the request. The authentication scheme then calls the `attrmap` function, which uses a LID map file to map the remote LID to the equivalent local LID.

To set up networking on an ES server that uses the cr1 authentication scheme, the system administrator must create an attribute map file called **`RLID.map`** that maps remote LIDs. If the map file does not exist on the ES server, all incoming connection requests fail.

NOTE

In an Enhanced Security environment, the client administrator needs to create a map as well. The client map, called **`LIDAUTH.map,`** is a variation of an attribute map that authorizes an outgoing connection request on a client on which the Enhanced Security Utilities are installed. When the client user attempts to make a connection to a server, the Connection Server on the client uses the map to determine if the connection can be made at the process's security level. Because the Connection Server uses the map and dictates its format, instructions for setting up **`LIDAUTH.map`** appear in the section called “Administering the Connection Server”.

At this time, cr1 is the only authentication scheme provided—with the exception of the traditional login/password scheme, which doesn't rely on ID Mapping. This section explains how to set up the LID mapping file required by cr1 specifically.

Setting Up RLID.map

To enable basic network communication on a server on which the Enhanced Security Utilities are installed and running, you must set up an attribute map that maps remote LIDs. Because the names of map files are hard-coded in an authentication scheme, the scheme dictates the name of the map file. The LID mapping file used by cr1 must have the pathname **`/etc/idmap/attrmap/RLID.map.`**

NOTE

If the Least Privilege Module (LPM) is running, the ID Mapping administrator must be in the NET role and be a member of group `sys.` LPM is part of the Enhanced Security Utilities.

The **RLID.map** file is no different from the general model described in the preceding section, “Setting Up an Attribute Map” on page 6-18. To create the map, enter the **attradmin** command and specify RLID as the *attrname*, as in the following example:

```
attradmin -A RLID -I M1@M2
```

M1@M2 is a format descriptor, which indicates the format of the remote LID required by cr1; specifically, it indicates that the remote LID will be provided in the format *LID@remote_system_name*, with both fields mandatory.

Below is an example of an **RLID.map** file:

```
M1@M2
*@sysA   %1
SYS_PRIVATE@sysB USER_LOGIN
*@sysB   SYS_PUBLIC
*@sysC   SYS_PUBLIC
```

To add an entry to the map file, use the **attradmin** command with the following syntax:

attradmin **-A** *attrname* **-a -r** *remoteval* **-l** *localval*

where *attrname* is RLID, *remoteval* is the LID on the remote system that you want to map, and *localval* is the LID on the server into which the remote LID is to be mapped.

NOTE

For a complete description of the **attradmin** fields, see the preceding section, “Setting Up an Attribute Map”, or **attradmin(1M).**

A LID should be specified using its alias (such as SYS_PUBLIC) whenever an alias exists on both systems.

For example, to map LID SYS_PUBLIC on the remote system name memphis to LID SYS_PUBLIC on the local server, enter

```
attradmin -A RLID -a -r SYS_PUBLIC@memphis -l SYS_PUBLIC
```

When both the client and server are running the Enhanced Security Utilities, the remote system sends its LID across the network, and the server maps it to a local LID. If both systems support the same set of LIDs, the local LID is equal to the remote LID. For example, the entry

```
*@sysA   %1
```

indicates that all LIDs on the client are to be mapped to the same LIDs on the server. LID SYS_PUBLIC on the client gets mapped to LID SYS_PUBLIC on the server, LID X to LID X, and so on.

NOTE

The `*` is a special character that indicates, in this case, all LIDs on the system. `%1` is a field specifier that is used to set up transparent mapping; it indicates that remote LIDs should map to the same values on the server as specified in *remote_attr*.

For more information about the use of special characters and field specifiers, see the preceding section, “Setting Up an Attribute Map” on page 6-18.

If both client and server do not support the same LIDs, then LIDs must be indicated explicitly in the map entry. For example, assume the client system, `sysB`, has defined a security level called `private:sales,` which has been assigned LID 10. Assume also that the server has an equivalent security level defined, but it has been assigned a local LID of 12. In order for the client and server to communicate at the same security level, LID 10 must be mapped to LID 12 on the server. The following command creates an entry that maps LID 10 on `sysB` to LID 12 on the server:

```
attradmin -A RLID -a -r LID10@sysB -l LID12
```

NOTE

There is no utility available to verify which LIDs represent equivalent security levels on client and server. If you know that the client and server do not support identical LID assignments, meet with the administrator of the client machine to determine how LIDs map before you enable communication between the two systems.

If you want to allow communication with a client that is not running the Enhanced Security Utilities, you can leave the remote LID in *remote_attr* blank. If the LID is blank, the server accepts a connection request without a remote LID and assigns the remote process the specified local LID. The following command creates an entry that maps all requests without LIDS from `sysD` to LID `SYS_PUBLIC` on the server:

```
attradmin -A RLID -a -r “@sysD” -l SYS_PUBLIC
```

`attradmin` will accept a blank LID even though the field `M1` is mandatory. However, it does insist that the `M1` field delimiter, `@`, be present.

Examples

If you do not know what levels and LIDs are supported on the client machine, you may want to allow the client to communicate with your system only at `SYS_PUBLIC`, the lowest security level. The following command creates an entry that maps all LIDs on the remote system `sysC` to LID `SYS_PUBLIC` on the server:

```
attradmin -A RLID -a -r “*@sysC” -l SYS_PUBLIC
```

NOTE

To protect characters that are meaningful to the shell, enclose the *remote_attr* in quotes.

If you know that all clients on the network are secure and support the same levels and LID assignments, you can set up transparent LID mapping with all remote systems by creating an **RLID.map** file with the following single entry:

```
*@*      %1
```

To create the entry, enter

```
attradmin -A RLID -a -r "*@*" -l %1
```

However, the network will probably include systems that do not support the Enhanced Security Utilities; it may be desirable to allow communication with those systems at the lowest security level. To create a file entry that allows all non-ES machines on the network to connect to the server at the lowest security level, enter the following command:

```
attradmin -A RLID -a -r "@*" -l SYS_PUBLIC
```

7
Administering the Basic Networking Utilities

7
Administering the Basic Networking Utilities

Introduction to Basic Networking Utilities Administration

The Basic Networking Utilities package (BNU) allows any computer running the UNIX operating system to communicate with any other computer that supports the Basic Networking Utilities. This includes non-UNIX systems as well as systems running any release of the UNIX operating system. The Basic Networking Utilities range from programs used to copy files between computers (**uucp** and **uuto**) to routines used for remote login and command execution (**cu**, **ct**, and **uux**).

NOTE

The Enhanced Security Utilities may be used to restrict access to a local system or to restrict access by a local system to remote systems.

Most tasks associated with Basic Networking Utilities administration may be performed using either the menu system or shell commands entered on the command line. Screen 7-1 shows the top-level BNU menu. It can be brought up on the screen by typing

```
sysadm basic_networking
```

```
          Basic Networking Utilities Management

devices - Adding, Listing, and Removing Networking Devices
polling - Adding, Listing, and Removing Systems to Be Polled
systems - Adding, Listing, and Removing Remote Systems
```

Screen 7-1. The Basic Networking Utilities Management Menu

When you have selected the option you want, self-explanatory submenus and instructions are displayed on the screen to lead you through the appropriate procedures.

Overview of BNU

What BNU Does

The following is a general description of the BNU process. The component parts of this process are described either later in this chapter or in the places indicated. The phases are marked on the diagram that follows.

Phase A	A user on the local machine issues a command requesting file transfer or remote execution communication with a remote computer (phase A). Several BNU database support files are read to determine if the remote computer is accessible by the local computer and the priority of the user's request compared to other users' requests. This phase ends by queuing the user's request in a spool area on the local machine and triggering the next phase.
Phase B	The `uucico` routine is triggered automatically (phase B). It reads several BNU database support files to determine when the remote computer can be reached, how to establish the link to the remote computer, how to handle data flow between the local and remote computers, and the maximum number of simultaneous requests for communication to the remote computer.
Phase B'	The `uucico` routine calls the connection routine, **`dial`** [see **`dial(3N)`**], and passes the request and associated information to the server.
Phases C and C'	The `uucico` routine on the remote computer is triggered automatically when a call for communication is received from the local computer (phase C). In this phase, the remote `uucico` routine reads BNU database support files on its computer to determine if the calling computer is allowed access, and what action to take if the calling computer is not allowed access (phase C'). For calling computers that are allowed access to the remote computer, the level of access is determined in this phase.
Phase D	Requests initiated on the local computer may contain commands to be executed on the remote computer (phase D). When these commands arrive on the remote computer, they are stored in a spool area. During this phase, the remote `uuxqt` routine runs these commands on the remote computer.

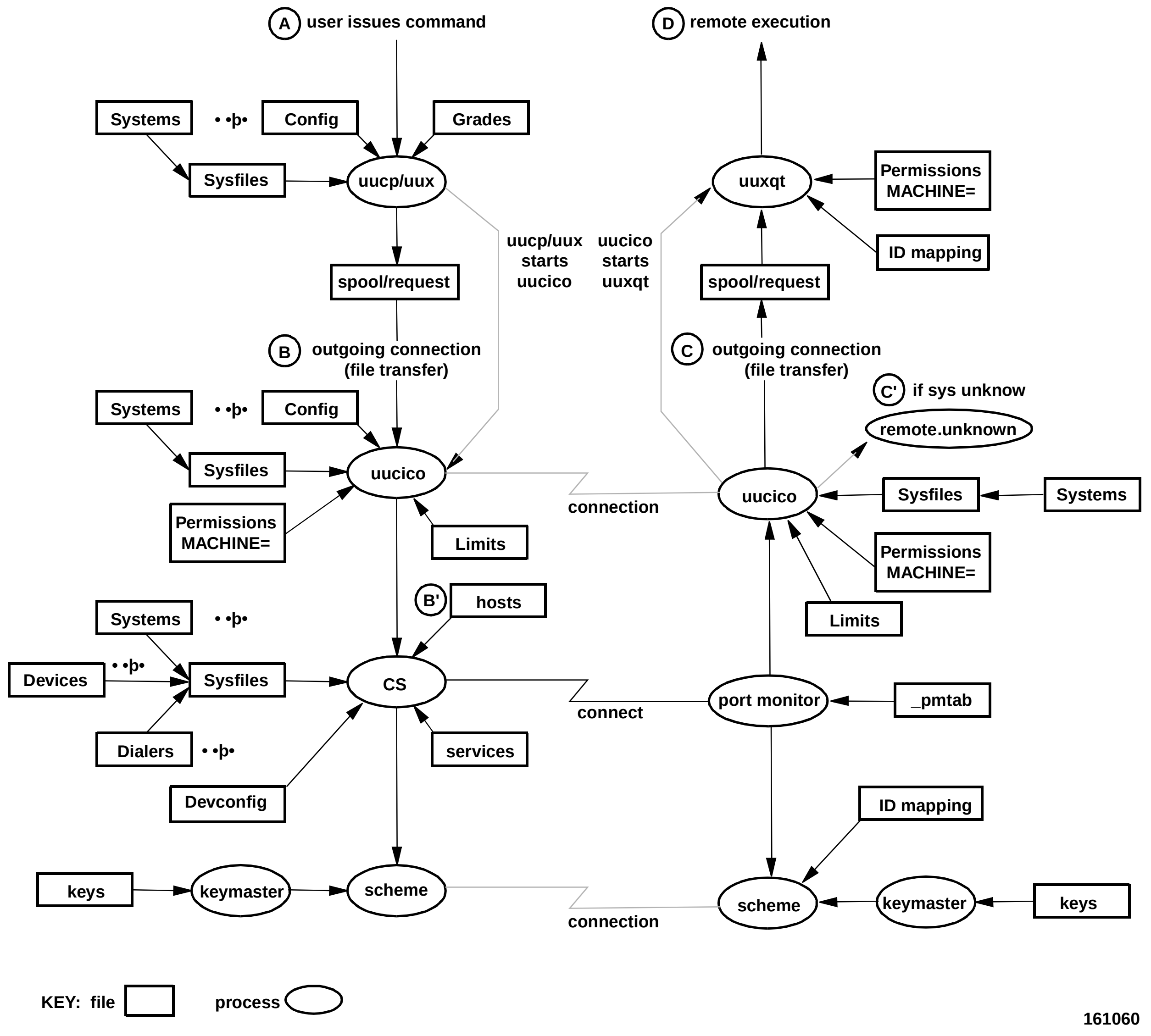


Figure 7-1. Basic Networking Process Diagram

BNU Components

Networking Hardware

Before a computer can communicate with other computers, hardware must be set up to complete the communications link. The hardware needed will depend on how you want to connect the computers. Computers may be connected using

- direct links
- telephone lines
- local area networks

Hardware installation is not the subject of this chapter.

Networking Programs

BNU is made up of programs and support files. BNU programs include user programs, administrative programs, and daemons. These three types of BNU programs are described in this section. They are described briefly with pointers to the manual pages that describe them in detail.

BNU support files are described in detail in the next section.

User Programs

The Basic Networking Utilities user programs are:

`cu`	Calls another UNIX system, a terminal, or possibly a non-UNIX system and manages an interactive conversation with possible transfers of files. See **`cu(1C)`** for more information.
`ct`	Dials the telephone number of a modem that is attached to a terminal and spawns a login process to that terminal. See **`ct(1C)`** for more information.
`uucp`	Copies files from a local UNIX machine to a destination either on the same machine or on a remote UNIX machine. **`uucp`** may also be used to copy files from a remote computer to another remote computer, to copy files from one location to another on a given remote computer, and to copy files from a remote computer to a local computer. Some remote copies will be restricted by the administrator of the remote machine. See **`uucp(1C)`** for more information.
`uuto`	Uses the **`uucp`** facility to send files to *PUBDIR* on another system. See **`uuto(1C)`** for more information.
`uupick`	Accepts or rejects files transmitted to a user. Specifically, **`uupick`** searches *PUBDIR* for files destined for the user. **`uupick`** is described on **`uuto(1C)`**.

uux	Gathers files from various systems, executes a command on a named system and then sends the output to a file on another named system. See **uux(1C)** for more information.
uustat	Displays the status of, or cancels, previously specified **uucp** commands; provides remote system performance information; and provides remote status of **uucp** connections to other systems. See **uustat(1C)** for more information.
uulog	Queries log files of **uucp** and **uuxqt** transactions. **uuxqt** is also described in “Networking Daemons” on page 7-6. **uulog** is described on **uucp(1C).**
uuglist	Prints the list of service grades available on a system for use with the **-g** option of **uucp** and **uux**. See **uuglist(1C)** for more information.
uuname	Lists the names of systems known to **uucp**. **uuname** is described on **uucp(1C)**.
uuencode	Converts a binary file into an ASCII-encoded file that can be transmitted using **mail** [see **mail(1)**]. See **uuencode(1C)** for more information.
uudecode	Reads an encoded file, strips off any leading and trailing lines added by mailer programs, and recreates the original binary data. See **uuencode(1C)** for more information.

The BNU user programs are located in **/usr/bin**. The BNU user programs are described in their respective manual pages. No special permissions are needed to use these programs.

Administrative Programs

The Basic Networking Utilities include the following three administrative programs:

uucheck	Checks for the presence of the **uucp** system-required files and directories. **uucheck** also checks the **/etc/uucp/Permissions** file for errors. See **uucheck(1M)** for more information.
uucleanup	Scans the spool directories for old files and removes them. The default values for file aging can be changed by command line options. An option also exists to restrict **uucleanup**'s action to the **system** spool directory. See **uucleanup(1M)** for more information.
Uutry	**Uutry** is a shell used to invoke **uucico** when calling a remote site. Debugging is initially turned on and the debugging output is put in a file. See **Uutry(1M)** for more information.

The BNU administrative programs are in **/usr/lib/uucp**, along with Basic Networking shell scripts. All Basic Networking Utilities administrative commands are fully described in their respective manual pages.

Since **uucp** owns the Basic Networking and spooled data files, the **uucp** login ID should be used only for BNU administration. The home directory for the **uucp** login is **/usr/lib/uucp**. Another Basic Networking login ID, **nuucp,** is used by remote computers to access the local machine. Calls to **nuucp** are answered by **uucico.**

Networking Daemons

The Basic Networking Utilities package includes three daemons. A daemon is a standing server or a routine that runs as a background process and performs system-wide public functions. The BNU daemons handle both file transfers and command execution. They may be invoked from the shell.

uucico Checks permissions, transfers files (if requested), logs results, and notifies the user (by **mail**) of transfer completion. When the local **uucico** daemon calls a remote computer, it invokes the **dial** routine [see **dial(3N)**], which then obtains authenticated connections from the Connection Server.

The Connection Server sets the security levels of all connections it establishes.

NOTE

Only files of the same security level can be transferred by any one **uucico.**

uucico is executed by the **uucp**, **uuto**, and **uux** programs, after all the required files have been created, to contact the remote computer. It is also executed by **uusched** and **Uutry**. See **uucico(1M)** for more information.

uuxqt Executes remote execution requests. **uuxqt** searches the spool directory for execute files (always named **X.file**) that have been sent from a remote computer. When an execute file is found, **uuxqt** opens it to get the list of data files required for the execution. It then checks to see if the required data files are available and accessible. If the files are present and can be accessed, **uuxqt** checks the **Permissions** file to verify that it has permission to execute the requested command. **uuxqt** is executed by the **uudemon.hour** shell script, which is started by **cron**. See **uuxqt(1M)** for more information.

uusched Schedules the queued work in the spool directory. Before starting **uucico**, **uusched** randomizes the order in which remote computers will be called. **uusched** is executed by the shell script **uudemon.hour**, which is started by **cron**. See **uusched(1M)** for more information.

The remote.unknown File

The **remote.unknown** file is a binary program that executes when a machine not found in any of the **Systems** files starts a conversation. It will log the conversation attempt and drop the connection. Its function is similar to that of a daemon.

CAUTION

If you change the permissions of the **`remote.unknown`** file so it cannot execute, your system will accept connections from any system.

Networking Support Files

There are three types of BNU support files: database files, administrative files, and log files.

- Database Files BNU database files are located in **`/etc/uucp`**. They are responsible for much of the actual networking activity associated with the Basic Networking Utilities package. In general, they determine which computers your computer will communicate with, the devices over which the communication will take place, the protocols for communicating with remote computers, and the security level access.

NOTE

BNU database support files used with previous releases of the UNIX operating system are compatible with the current release and may be used without alteration.

The following list describes each of the database support files and its function briefly. These files are described in detail in the "Database Support Files" on page 7-9.

`Config`	Contains a list of variable parameters within BNU. The administrator can set these parameters to configure the network manually.
`Devconfig`	This file is used to configure network connections on TCP/IP or some other network provider.
`Devices`	Contains information concerning the location and line speed of automatic call units, direct links, and network devices.
`Dialcodes`	Contains dial-code abbreviations that may be used in the *telephone number* field of the **`Systems`** and **`Dialers`** files.
`Dialers`	Contains character strings required to communicate with network devices, automatic calling units, and direct links.
`Grades`	Used to define the job grades, and the permissions associated with each job grade, that users may specify to queue jobs to a remote computer.
`Limits`	Defines the maximum number of simultaneous **`uucico`**s, **`uuxqt`**s, and **`uusched`**s permitted on a machine.

Permissions	Defines the levels of access granted to remote computers when they attempt to transfer files or execute remote commands on the local computer.
Poll	Defines computers that are to be polled by the local system and when they are polled.
Sysfiles	Assigns different or multiple files to be used as **Systems**, **Devices**, and **Dialers** files by various services (such as **uucico** and **cu**).
Systems	Contains information needed by the **uucico** daemon and the **cu** program to establish a link to a remote system such as the name of the remote system, the name of the connecting device associated with the remote system, when the system can be reached, a telephone number or network address, login ID, and password. The **Systems** file is also used by the client machine to verify that the client is allowed to connect to the server and by the server machine to verify that the server may accept calls from the client.
RLID.map	Contains the remote system name and incoming LID as input values and the local LID as an output value. When a request for access is initiated from a remote system, the authentication scheme checks this file and matches the incoming LID, then assigns the local LID to the request as a local level. This then determines what the requester can access. **RLID.map** is described in detail in Chapter 6, “Administering ID Mapping.”
LIDAUTH.map	The **LIDAUTH.map** file is consulted by the Connection Server to determine whether the local system is allowed to initiate a connection to a remote system at the current LID. The file contains the remote system name and local LID as input values and `%1` or `%i` as the output value to show whether the local LID is authorized or invalid.

- Administrative Files BNU administrative files are created in spool directories by network processes to hold temporary data or store information about remote transfers and command executions. They are described in detail in “Administrative Support Files” on page 7-41.
- Log Files BNU log files keep track of overall statistics for the computer network, particularly in the areas of security and accounting. The structure of each of the BNU log files is described in “Log Files” on page 7-43.

BNU Administration

This section outlines the steps a system administrator must take before using the Basic Networking Utilities and indicates where to find the information required to administer BNU once it has been set up.

NOTE

BNU database support files used with previous releases of the UNIX operating system are compatible with the current release and may be used without alteration.

Briefly, BNU administration involves the following procedures:

Setup	BNU setup is described in detail in the “Database Support Files” on page 7-9.
Maintenance	Basic Networking Utilities maintenance involves updating BNU database support files when necessary, tracking BNU transactions, and cleaning up BNU administrative and log files. BNU Maintenance is discussed in “BNU Maintenance” on page 7-63.
Debugging	BNU debugging procedures help identify and correct common problems in BNU operation and administration. BNU debugging procedures are discussed in “BNU Debugging” on page 7-66.

Database Support Files

Setting up the Basic Networking Utilities involves configuring the database support files. This section describes these files and their structure in detail. Some of the database support files are delivered with the system. In some cases these default files can be used without modification. In other cases they cannot. Each of the file descriptions below indicates whether the default file may be used and, if not, what changes must be made. BNU support files are in the **`/etc/uucp`** directory. The **`RLID.map`** and **`LIDAUTH.map`** files are in the **`/etc/idmap/attrmap`** directory.

NOTE

BNU database support files used with previous releases of the UNIX operating system are compatible with the current release and may be used without alteration.

Wherever possible, changes to the database support files should be made by using the system administration menus (accessed through the **`sysadm`** command). The BNU setup procedure provides the steps for initializing the BNU database files. This procedure uses the **`sysadm`** subcommands and a text editor, except for the security-related database files **`/etc/idmap/attrmap/RLID.map`** and **`/etc/idmap/attrmap/LIDAUTH.map`**. To maintain these two files, use the **`attradmin`** command. The use of **`attradmin`** to maintain **`RLID.map`** is described under Chapter 6, “Administering ID Mapping.” Its use in maintaining **`LIDAUTH.map`** is described in Chapter 4, “Administering the Connection Server.”

Begin by typing

```
sysadm basic_networking
```

This will give you the starting menu. From this point, you can begin setting up (initializing) the various database files. To get help along the way, press the HELP function key. This will give you a detailed description of a menu selection. Pressing the CANCEL function key exits help mode.

Setting up the **`Permissions`**, **`Devconfig`**, **`Sysfiles`** and **`Limits`** files, and adding **`uucp`** logins are principal functions in the initial Basic Networking setup process.

The permissions on networking devices should be 0600, with the owner set to `root`.

The level of the BNU administrative files should be set to `SYS_PUBLIC` so that processes running at all levels will be able to read these files.

The Permissions File

A default **`/etc/uucp/Permissions`** file is delivered with the system and contains the following entry, which provides maximum security:

```
LOGNAME=nuucp
```

`/etc/uucp/Permissions` specifies the permissions that remote computers have with respect to login, file access, and command execution. There are options that restrict the remote computer's ability to request files and its ability to receive files queued by the local site. Another option is available that specifies the commands that a remote site can execute on the local computer.

The following items should be considered when using the **`Permissions`** file to restrict the level of access granted to remote computers:

- All login IDs used by remote computers to log in for **`uucp`** communications must appear in one and only one `LOGNAME` entry.
- Any site that is called whose name does not appear in a `MACHINE` entry, will have the following default permissions and restrictions:
  - Local send and receive requests will be executed.
  - The remote computer can send files to your computer's **`/var/spool/uucppublic`** directory.
  - The commands sent by the remote computer for execution on your computer must be one of the default commands; usually **`rmail`**.
- When a remote machine calls you, unless you have a unique login and password for that machine or are using cr1 authentication, you don't know if the machine is who it says it is.

Each entry is a logical line. Physical lines may be terminated by a backslash (\) to indicate that the entry continues on the next line. Entries are made up of options delimited by white space. Each option is a name/value pair in the following format:

name=value

Note that no white space is allowed within an option assignment.

Comment lines begin with a pound sign (#) and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of **`Permissions`** file entries:

`LOGNAME`	Specifies the permissions that take effect when a remote computer logs in on (calls) your computer.
`MACHINE`	Specifies permissions that take effect when your computer logs in on (calls) a remote computer.

Options

This section describes each option, specifies how they are used, and lists their default values.

`REQUEST` When a remote computer calls your computer and requests the transfer of a file, this request is granted or denied based on the value of the `REQUEST` option. The string

```
REQUEST=yes
```

specifies that the remote computer can request the transfer of files from your computer. The string

```
REQUEST=no
```

specifies that the remote computer cannot request file transfers from your computer. `no` is the default value. It will be used if the `REQUEST` option is not specified. The `REQUEST` option can appear in either a `LOGNAME` (remote calls you) entry or a `MACHINE` (you call remote) entry.

`SENDFILES`

When a remote computer calls your computer and completes its work, it may attempt to take work your computer has queued for it. The `SENDFILES` option specifies whether your computer can send the work queued for the remote computer.

The string

```
SENDFILES=yes
```

specifies that your computer may send the work that is queued for the remote computer as long as it logged in as one of the names in the `LOGNAME` option. This string is mandatory if your computer is in a “passive mode” with respect to the remote computer.

The string

```
SENDFILES=call
```

specifies that files queued in your computer will be sent only when your computer calls the remote computer. The call value is the default for the

SENDFILES option. This option is only significant in LOGNAME entries, since MACHINE entries apply when calls are made out to remote computers. If the option is used with a MACHINE entry, it will be ignored.

PUBDIR This option defines the directory that **uucp** will use for public file transfers. The default value is **/var/spool/uucppublic**, and this value is used to initialize the value of the READ and WRITE options described below.

CAUTION

Changing this value is discouraged, as it may have an adverse affect on other system utilities.

READ and WRITE
These options specify the various parts of the file system that **uucico** can read from or write to. The READ and WRITE options can be used with either MACHINE or LOGNAME entries.

The default for both the READ and WRITE options is the PUBDIR directory, which would be equivalent to the following entries:

```
READ=/var/spool/uucppublic
WRITE=/var/spool/uucppublic
```

The strings

```
READ=/ WRITE=/
```

specify permission to access any file that can be accessed by a local user whose access permissions are set to “other.”

The value of these entries is a colon separated list of pathnames. The READ option is for requesting files, and the WRITE option for depositing files. One of the values must be a component of any full pathname of a file coming in or going out. To grant permission to deposit files in **/usr/news** as well as the public directory, the following values would be used with the WRITE option:

```
WRITE=/var/spool/uucppublic:/usr/news
```

It should be pointed out that if the READ and WRITE options are used, all pathnames must be specified because the pathnames are not added to the default list. For instance, if the **/usr/news** pathname was the only one specified in a WRITE option, permission to deposit files in the public directory would be denied.

You should be careful what directories you make accessible for reading and writing by remote systems. For example, you probably wouldn't want remote computers to be able to write over your **/etc/passwd** file, so **/etc** shouldn't be open to writes.

NOREAD and NOWRITE

The NOREAD and NOWRITE options specify exceptions to the READ and WRITE options or defaults. The strings

```
READ=/ NOREAD=/etc WRITE=/var/spool/uucppublic
```

would permit reading any file except those in the **/etc** directory (and its subdirectories—remember, these are prefixes) and writing only to the default **/var/spool/uucppublic** directory. NOWRITE works in the same manner as the NOREAD option. The NOREAD and NOWRITE can be used in both LOGNAME and MACHINE entries.

DIRECT

This option specifies whether files that have been received can be placed into a directory into the destination directory. If the value of DIRECT is no, files that are received are put into **uucp**'s private spool directory, and then copied to the destination directory. If the value of DIRECT is yes, files that are received are directly put into the destination directory. The default value for DIRECT is no.

CALLBACK

The CALLBACK option is used in LOGNAME entries to specify that no transaction will take place until the calling system is called back. CALLBACK can be used in the following two examples. From a security standpoint, if you call back a machine, you can be fairly certain it is the machine it says it is. If you are doing long data transmissions, you can choose the machine that will be billed for the longer call.

The string

```
CALLBACK=yes
```

specifies that your computer must call the remote computer back before any file transfers will take place.

The default for the CALLBACK option is

```
CALLBACK=no
```

The CALLBACK option is rarely used. Note that if two sites have this option set for each other, a conversation will never get started and no transfers will occur between them.

KEYS

The KEYS= entry defines the key management facility that obtains keys used in the authentication of remote command execution requests. This option provides an override capability for the global value specified in the **Config** file. At present, cr1 is the only key management facility available to BNU. There is no default key management facility. To specify the cr1 key management facility, the string

```
KEYS=cr1
```

must be entered in the **Permissions** file.

CRYPT

This option specifies the encryption type to use when authenticating remote execution requests generated by the **uux** command. The default value is des. The value enigma may be used if the export controlled Encryption Utilities

Package is not available. This option provides an override capability for the global value specified in the **`Config`** file.

`AUTH` The `AUTH=` entry determines whether authentication is required for remote requests. This option provides an override capability for the global value specified in the **`Config`** file. If either of the strings

```
AUTH=yes
```

or

```
AUTH=req
```

is included in the **`Permissions`** file, no remote command request will be accepted without authentication. When authenticated requests are executed, they are executed under the mapped id of the originator and all commands are allowed, that is, the `COMMANDS` value is ignored.

Either of the strings

```
AUTH=opt
```

or

```
AUTH=no
```

indicates that authentication is not required for remote command execution. In this case, commands are executed as in previous releases, limited by the `COMMANDS` value.

NOTE

If authentication is attempted but fails, the request will be rejected, regardless of the value of `AUTH`.

`COMMANDS`

CAUTION

The `COMMANDS` option can compromise the security of your system. Use it with extreme care.

The `COMMANDS` option is used only for unauthenticated remote command execution requests. See `KEYS` and `AUTH`, above, for use with authenticated remote command execution requests.

The **`uux`** program will generate remote execution requests and queue them to be transferred to the remote computer. Files and a command are sent to the target computer for remote execution. The `COMMANDS` option can be used in `MACHINE` entries to specify the commands that a remote computer can execute on the local computer. Note that `COMMANDS` is not relevant in a

`LOGNAME` entry; `COMMANDS` in `MACHINE` entries define command permissions whether we call the remote system or it calls us.

The string

```
COMMANDS=rmail
```

specifies the default commands that a remote computer can execute on your computer. If a command string is used in a `MACHINE` entry, the default commands are overridden. For instance, the entry

```
MACHINE=owl:raven:hawk:dove \
COMMANDS=rmail:rnews:lp
```

overrides the `COMMANDS` default so that the computers `owl`, `raven`, `hawk`, and `dove` can now execute **`rmail`**, **`rnews`**, and **`lp`** on your computer.

In addition to the names as specified above, there can be full path names of commands. For example,

```
COMMANDS=rmail:/usr/lbin/rnews:/usr/local/lp
```

specifies that the command **`rmail`** uses the default path. The default path for remote execution is **`/usr/bin.`** When the remote computer specifies **`rnews`** or **`/usr/lbin/rnews`** for the command to be executed, **`/usr/lbin/rnews`** will be executed regardless of the default path. Likewise, **`/usr/local/lp`** is the **`lp`** command that will be executed.

CAUTION

Including the `ALL` value in the list means that any command from the remote computer(s) specified in the entry will be executed. If you use this value, you give the remote computer full access to your computer. Be careful. This allows far more access than normal users have.

The string

```
COMMANDS=/usr/lbin/rnews:ALL:/usr/local/lp
```

illustrates two points:

- The `ALL` value can appear anywhere in the string.
- The pathnames specified for **`rnews`** and **`lp`** will be used (instead of the default) if the requested command does not contain the full pathnames for **`rnews`** or **`lp.`**

If commands are executed using the authenticated remote execution feature, the `COMMANDS` list is ignored and all commands are available to the authenticated user, as if the user had logged in directly.

The `VALIDATE` option described below should be used with the `COMMANDS` option whenever potentially dangerous commands like **`cat`** and **`uucp`** are specified with the `COMMANDS` option. Any command that reads or writes files is potentially dangerous to local security when executed by the **`uucp`** remote execution daemon (**`uuxqt`**).

`VALIDATE`

The `VALIDATE` option is used in conjunction with the `COMMANDS` option when specifying commands that are potentially dangerous to your computer's security. It is used to provide a certain degree of verification of the caller's identity. The use of the `VALIDATE` option requires that privileged computers have a unique login/password for **`uucp`** transactions. An important aspect of this validation is that the login/password associated with this entry be protected. If an outsider gets that information, that particular `VALIDATE` option can no longer be considered secure. `VALIDATE` is merely an added level of security on top of th `COMMANDS` option (though it is a more secure way to open command access than `ALL).`

Careful consideration should be given to providing a remote computer with a privileged login and password for **`uucp`** transactions. Giving a remote computer a special login and password with file access and remote execution capability is like giving anyone on that computer a normal login and password on your computer. Therefore, if you cannot trust users on the remote computer, do not provide that computer with a privileged login and password.

The `LOGNAME` entry

```
LOGNAME=uucpfriend VALIDATE=eagle:owl:hawk
```

specifies that if one of the remote computers that claims to be eagle, owl, or hawk logs in on your computer, it must have used the login **`uucpfriend.`** If an outsider gets the **`uucpfriend`** login/password, masquerading is trivial.

But what does this have to do with the `COMMANDS` option, which only appears in `MACHINE` entries? It links the `MACHINE` entry (and `COMMANDS` option) with a `LOGNAME` entry associated with a privileged login. This link is needed because the execution daemon is not running while the remote computer is logged in. In fact, it is an asynchronous process with no knowledge of what computer sent the execution request. Therefore, the real question is: How does your computer know where the execution files came from?

Each remote computer has its own “spool” directory on your computer. These spool directories have write permission given only to the UUCP family of programs. The execution files from the remote computer are put in its spool directory after being transferred to your computer. When the **`uuxqt`** daemon runs, it can use the spool directory name to find the `MACHINE` entry in the **`Permissions`** file and get the `COMMANDS` list, or a default list will be used if the computer name does not appear in the **`Permissions`** file.

The following example shows the relationship between the `MACHINE` and `LOGNAME` entries:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
COMMANDS=rmail:/usr/lbin/rnews \
READ=/ WRITE=/
```

```
LOGNAME=uucpz VALIDATE=eagle:owl:hawk \
REQUEST=yes SENDFILES=yes \
READ=/ WRITE=/
```

The value in the `COMMANDS` option means that remote mail and **`/usr/lbin/rnews`** can be executed by remote users.

In the first entry, you must make the assumption that when you want to call one of the computers listed, you are really calling either `eagle`, `owl`, or `hawk`. Therefore, any file put into one of the `eagle`, `owl`, or `hawk` spool directories is put there by one of those computers. If a remote computer logs in and says that it is one of these three computers, its execution files will also be put in the privileged spool directory. You therefore have to validate that the computer has the privileged login **`uucpz`**.

`MACHINE` Entry for “Other” Systems

You may want to specify different option values for the computers your computer calls that are not mentioned in specific `MACHINE` entries. This may occur when there are many computers calling in and the command set changes from time to time. The name `OTHER` for the computer name is used for this entry as shown below:

```
MACHINE=OTHER \
COMMANDS=rmail:rnews:/usr/lbin/Photo:/usr/lbin/xp
```

All other options available for the `MACHINE` entry may also be set for the computers that are not mentioned in other `MACHINE` entries.

Combining `MACHINE` and `LOGNAME` Entries

It is possible to combine `MACHINE` and `LOGNAME` entries into a single entry where the common options are the same. For example, the two entries

```
MACHINE=eagle:owl:hawk REQUEST=yes \
 READ=/ WRITE=/
```

and

```
LOGNAME=uucpz REQUEST=yes SENDFILES=yes \
 READ=/ WRITE=/
```

share the same `REQUEST`, `READ`, and `WRITE` options. These two entries can be merged as shown below:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
LOGNAME=uucpz SENDFILES=yes \
 READ=/ WRITE=/
```

As can be seen, the default entry

```
LOGNAME=nuucp
```

provides maximum security since it is equivalent to

```
LOGNAME=nuucp \
     MACHINE=OTHER \
     REQUEST=no \
SENDFILES=call \
READ=/var/spool/uucppublic \
WRITE=/var/spool/uucppublic \
AUTH=no \
COMMANDS=rmail
```

NOTE

Because KEYS is not specified, no keys will be available to authenticate requests. Attempts will therefore fail. Only **rmail** will be available, as it was in previous releases.

The Devconfig File

The **/etc/uucp/Devconfig** file is used if you are using BNU over a TCP network or some other TLI-conformant provider. **Devconfig** entries define the STREAMS modules that are used for a particular device. (The **push=** variable shows the modules and the order in which they are pushed onto a stream.) Different modules and devices can be defined for **cu** and **uucico** services.

Entries in the **Devconfig** file have the format:

```
service=x[:x...] device=y push=z[:z...]
```

where *x* can be **cu**, **uucico**, or both separated by a colon; *y* is the name of a network and must match an entry in the **Devices** file; and *z* is replaced by the names of STREAMS modules in the order in which they are to be pushed onto the Stream. Different modules and devices can be defined for **cu** and **uucico** services.

If you are using TCP/IP, the two entries shown below are all you need in this file.

```
service=cu      device=TCP  push=ntty:tirdwr
service=uucico  device=TCP  push=ntty:tirdwr
```

In the example, first ntty in pushed, then tirdwr. ntty is a hardware emulation module.

You must also create an entry for TCP in the **Devices** file. Descriptions in the **Devices** file define Transport Interface devices.

The **Devconfig** file cannot be modified using the **sysadm** menu interface. If you want to change the contents of the file, you must use a text editor.

The Sysfiles File

/etc/uucp/Sysfiles allows you to break the logical **Systems, Devices,** and **Dialers** files into multiple physical files. This makes it possible for different files to be used by **uucp** and **cu** as **Systems**, **Devices**, and **Dialers** files. These files may be useful in the following cases:

- You may want different **Systems** files so requests for **cu** services can be made to different addresses than the addresses used for **uucico** services.
- You may want different **Dialers** files to use different chat scripts for **cu** and **uucico**.
- You may want to have multiple **Systems**, **Dialers**, and **Devices** files. The **Systems** file in particular may become large, making it convenient to split it into several smaller files.

The format of the **Sysfiles** file is

```
service=w[:w...] systems=x[:x...] \
    dialers=y[:y...] \
    devices=z[:z...]
```

where *w* is replaced by **uucico**, **cu**, or both, separated by a colon; *x* is one or more files to be used as the **Systems** file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the **Dialers** file; and *z* is one or more files to be used as the **Devices** file. Each file is assumed to be relative to the **/etc/uucp** directory, unless a full path is given. A backslash (\) can be used to continue an entry on to the next line.

The following is an example of a **Sysfiles** file.

```
service=uucico  systems=Systems.cico:Systems\
                dialers=Dialers.cico:Dialers\
                devices=Devices.cico:Devices
service=cu      systems=Systems.cu:Systems\
                dialers=Dialers.cu:Dialers\
                devices=Devices.cu:Devices
```

When different systems files are defined for **uucico** and **cu** services, your machine will store two different lists of systems. To print the **uucico** list, use the **uuname** command; to print the **cu** list, use the **uuname -c** command.

The Limits File

The **/etc/uucp/Limits** file is used to limit the maximum number of simultaneous **uucicos**, **uuxqts**, and **uuscheds** that are running on your machine.

The format of the **Limits** file is

```
    service=x max=y
```

where *x* can be **`uucico`**, **`uuxqt`** or **`uusched`**, and *y* is the limit permitted for that service.

The fields are order insensitive and lowercase

The following entries should most commonly be used in the file:

```
    service=uucico max=5
    service=uuxqt max=2
    service=uusched max=2
```

The example allows five **`uucicos`**, two **`uuxqts`**, and two **`uuscheds`** to run simultaneously.

The **`Limits`** file cannot be modified using the system administration menus (accessed through the **`sysadm`** command). If you want to change the contents of the file, you must use one of the UNIX system text editors.

NOTE

Due to the interaction with security levels, the limits are imposed at any given level, although the total **`uucicos`**, **`uuzqts`**, and **`uuscheds`** running on your machine may exceed that limit.

The RLID.map File

If you are running the Enhanced Security Utilities or are using cr1 to authenticate connections, the system administrator must create a **`RLID.map`** file. A default file is not delivered with the system.

The **`/etc/idmap/attrmap/RLID.map`** file contains the information used to determine the LID assigned to an incoming request for access from a remote system. The file contains the client system name, the remote system LID which may be matched by the incoming request, and the local LID assigned to the request if there is a match.

Setting up an **`RLID.map`** file is described in the Chapter 6, "Administering ID Mapping".

The LIDAUTH.map File

If the Enhanced Security Utilities are installed and running on your system, the system administrator must create an **`LIDAUTH.map`** file. A default file is not delivered with the system.

The **`/etc/idmap/attrmap/LIDAUTH.map`** file contains the information used to determine whether the local system is allowed to initiate a connection to a remote system at the current LID, that is, the LID information for authorizing outgoing requests. The **`LIDAUTH.map`** file is consulted by the Connection Server to determine whether the local system is allowed to initiate a connection to a remote system at the current LID. The

LIDAUTH.map file must contain the remote system name and local LID as input values and a local LID or `INVALID` as output values.

Setting up a **LIDAUTH.map** file is described in Chapter 4, “Administering the Connection Server.”

The Config File

The **/etc/uucp/Config** file allows the administrator to override certain parameters within BNU by specifying global BNU options.

Each entry in the **Config** file has the following format:

parameter=*value*

Where *parameter* is one of the configurable parameters and *value* is the value to be assigned to that parameter. See the **Config** file provided with your system for a complete list of configurable parameter names.

The following **Config** file entry sets the default protocol ordering to “`Gge`” and changes the “`G`” protocol defaults to 7 windows and 512-byte packets. See “Protocols” on page 7-27.

```
Protocol=G(7,512)ge
```

If the system administrator requires authentication of **uux** requests for all or some systems, the entry

```
AUTH=yes
```

or

```
AUTH=req
```

can be included in the **Config** file. This establishes a global value for BNU. To override the global values specified in the **Config** file, entries may be included in the **/etc/uucp/Permissions** file for specific machines. See the section on the **Permissions** file.

If the `AUTH` option is not included, `AUTH` defaults to optional (`AUTH=opt` or `AUTH=no`).

To specify a key management mechanism, a line of the form

`KEYS=`*name*

can be included in the **Config** file. *name* is the name of the key management mechanism. For example, to specify that the `cr1` key management mechanism is to be used, the entry

```
KEYS=cr1
```

should be included in the **Config** file.

The Devices File

The **`Devices`** file (**`/etc/uucp/Devices`**) contains information for all the devices that may be used to establish a link to a remote computer. Provisions are made for several types of devices, such as automatic call units, direct links, and network connections.

NOTE

This file works closely with the **`Dialers`**, **`Systems`**, and **`Dialcodes`** files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to a related entry in another file.

Each entry in the **`Devices`** file has the following format:

Type Line Line2 Class Dialer-Token-Pairs

These fields are defined as:

Type This field contains the device name. The device name can be a name of the user's choosing, but it must match the name in the third field of the **`Systems`** file. The two names `Direct` and `ACU` are reserved. Generally the *Type* field is the name of a particular computer directly linked to the computer on which the **`Devices`** file in question is located (to differentiate it from other directly linked computers) or the name of a network accessible to the computer on which the **`Devices`** file is located (to differentiate it from other accessible networks).

`Direct`	Indicates a Direct Link to another computer or a switch.
`ACU`	Specifies that the link to a remote computer is made through an automatic call unit (Automatic Dial Modem). This modem may be connected either directly to your computer or indirectly through a Local Area Network (LAN) switch.
LAN_Switch	The name of the LAN or switch. For instance, `TCP` could be the name for the TCP/IP network.
Sys-Name	Specifies a direct link to a particular computer. (*Sys-Name* is replaced by the name of the computer.) This naming scheme is used to convey the fact that the line associated with this **`Devices`** entry is for a particular computer in the **`Systems`** file.
`CS`	TLI LAN-type connection with authentication scheme invocation.

The keyword used in the *Type* field is matched against the third field of **`Systems`** file entries, as shown in these sample files.

```
#Systems file
#
eagle Any ACU  1200 3251        ogin: nuucp ssword: XXXXXX
sys1  Any CS -    sys1,uucico
sys2  Any CS -
sys3  Any CS -    sys3,login  in:--in nuucp word:XX
sys4  Any DK   9600 sys4        INVOKE "cr1 -r"
sys5  Any DK   9600 sys5        in:--in nuucp word:XX
sys6  Any LAN - network_address
```

Screen 7-2. Sample Systems File

```
#Devices file
#
ACU term/11 - 2400 att4024
Direct term/12 - Any direct
sysb term/13 - Any uudirect
CS - - - CS
LAN,eg tcp - - TLI \D
```

Screen 7-3. Sample Devices file

The protocol to use for a device can be designated within the *Type* field. See the “Protocols” section on page 7-27.

Line This field contains the device name of the line (port) associated with the **`Devices`** entry. For instance, if the Automatic Dial Modem for a particular entry is attached to the **`/dev/term/11`** line, the name entered in the *Line* field will be **`term/11`**. There is an optional modem control flag, `M`, that can be used to indicate that the device should be opened without waiting for a carrier. The modem control flag is separated from the device name by a comma. For example,

```
term/11,M
```

`CS` entries must contain a hyphen (`-`) in the *Line* field.

Line2 If the keyword `ACU` is used in the *Type* field and the ACU is an 801-type dialer, *Line2* will contain the device name of the 801 dialer. Since 801-type ACUs do not contain a modem, separate modems are required and are connected to a different line, defined in the *Line* field. This means that one line will be allocated to the modem and another to the dialer. Non-801 dialers will not normally use this configuration. Although non-801 dialers therefore ignore the *Line2* field, it must contain a hyphen (`-`) as a placeholder. `CS` entries must also contain a hyphen in the *Line2* field.

Class If the `ACU` or `Direct` keywords are used in the *Type* field, the *Class* field need include only the speed of the device. It may, however, contain a letter and a speed, for example, C1200 (Centrex), or D1200 (Dimension PBX). This allows large organizations that have more than one type of telephone network to differentiate between classes of dialers. One network may be dedicated to serving only internal office communications, for example, while another handles external communications. In this case, it is necessary to distinguish which line(s) should be used for internal communication and which should be used for external communication. The keyword used in the *Class* field of the **`Devices`** file is matched against the fourth field of **`Systems`** file entries as shown in the **`Devices`** and **`Systems`** file lines below.

```
#Devices file
#
ACU tty11 - D1200 penril
```

```
#Systems file
#
eagle Any ACU D1200 3251 ogin: nuucp ssword: XXXXXX
```

Some devices can be used at any speed. In this case, the keyword `Any` may be entered in the *Class* field. If `Any` is used, the line will match any speed requested in a **`Systems`** file entry. However, if the **`Devices`** file *Class* field contains `Any` <u>and</u> the **`Systems`** file *Class* field contains `Any`, the speed defaults to 1200 bps.

Dialer-Token-Pairs

This field contains pairs of dialers and tokens. The *Dialer* portion of a dialer-token pair may be the name of an automatic dial modem, a LAN switch, or it may be **`direct`** or **`undirect`** for a Direct Link device. The *Token* portion of a dialer-token pair may be entered immediately following the *Dialer* portion, or, if it is not present, it will be taken from a related entry in the **`Systems`** file. There may be any number of dialer-token pairs. This field has the format:

dialer token [*dialer token*]

where the last pair may or may not be present, depending on the associated device (dialer). In most cases, the last pair contains only a *dialer* portion and the *token* portion is retrieved from the *Phone* field of the **`Systems`** file entry.

A valid entry in the *dialer* portion may be defined in the **`Dialers`** file or may be one of several special dialer types. These special dialer types are compiled into the software and are therefore available without having to be entered in the **`Dialers`** file.

CS A request for a TLI network. The specific network is chosen from the networks available to the system on which the requested service is available. Generally, the network connection returned supports **`t_snd(3N)`** and **`t_rcv(3N`**) semantics but may be modified using the **`Devconfig`** file.

801	Bell 801 auto dialer
TLI/TLIS	Transport Level Interface Network. Generally, the network connection returned supports **`t_snd(3N)`** and **`t_rcv(3N)`** semantics but may be modified to support **`read(2)`** and **`write(2)`**, using the **`Devconfig`** file.

The *Dialer-Token-Pairs* field may be structured differently, depending on the device associated with the entry:

- If an automatic dialing modem is connected directly to a port on your computer, the *Dialer-Token-Pairs* field of the associated **`Devices`** file entry will only have one pair. This pair will normally be the name of the modem and is used to match the **`Devices`** file entry with an entry in the **`Dialers`** file. The *dialer* field must therefore match the first field of a **`Dialers`** file entry as shown below:

```
#Devices file
#
ACU term/11 - 1200 att2212c
```

```
#Dialers file
#
att2212c =+-, "" atzod,o12=y,o4=n\r\c \
        \006 atT\T\r\c ed
```

  Notice that only the *dialer* portion (`att2212c`) is present in the *Dialer-Token-Pairs* field of the **`Devices`** file entry. This means that the *token* to be passed to the dialer (in this case the phone number) is taken from the *Phone* field of a **`Systems`** file entry. (`\T` is implied; backslash sequences are described below.)

- If a direct link is established to a given computer, the *Dialer-Token-Pairs* field of the associated entry will contain the keyword **`direct`** or **`uudirect`**. This is true for both types of direct link entries, `Direct` and `System-Name` (see the discussion of the *Type* field).

- If you want to communicate with a computer that is on the same local network switch as your computer, your computer must first access the switch and the switch can make the connection to the other computer. In this type of entry, there is only one pair. The *dialer* portion is used to match a **`Dialers`** file entry as shown below:

```
#Devices file
#
Datakit term/13 - 9600 datakit
```

```
#Dialers file
#
datakit ""   "" \d TION: - - :TION \D
```

In the example, the *token* portion is left blank. This indicates that it is retrieved from the **`Systems`** file. The **`Systems`** file entry for this particular computer will contain the token in the *Phone* field, which is normally reserved for the telephone number of the computer (see the discussion of the *Phone* field in the section “The Systems File” on page 7-31. This type of dialer-token pair contains an escape character (`\D`), to ensure that the contents of the *Phone* field will not be interpreted as a valid entry in the **`Dialcodes`** file.

- If an automatic dialing modem is connected to a switch, your computer must first access the switch and the switch will make the connection to the automatic dialing modem. This type of entry requires two dialer-token pairs. The *dialer* portion of each pair (the fifth and seventh fields of the entry) will be used to match entries in the **`Dialers`** file:

```
#Devices file
#
ACU term/14 - 1200 datakit dial att2212c
```

```
#Dialers file
#
datakit "" "" \d TION: - - :TION \D
att2212c =+-, "" atzod,o12=y,o4=n\r\c \006 atT\T\r\c ed
```

In the first pair, `datakit` is the dialer and `dial` is the token that is passed to the Datakit® switch to tell it which device (auto dial modem) to connect to your computer. This token will be unique for each LAN switch since each switch may be set up differently. Once the modem has been connected, the second dialer-token pair is accessed. The second dialer is `att2212c`; the token is retrieved from the **`Systems`** file.

There are two escape characters that may appear in a *Dialer-Token-Pairs* field:

`\T` Specifies that the *Phone* (*token*) field should be translated using the **`Dialcodes`** file. This escape character is normally placed in the **`Dialers`** file for each caller script associated with an automatic dial modem. The translation will not therefore take place until the caller script is accessed.

`\D` Indicates that the *Phone* (*token*) field should not be translated using the **`Dialcodes`** file. A `\D` is also used in the **`Dialers`** file with entries associated with network switches (develcon and micom).

If the dialer is an internal dialer, `\T` is the default. Otherwise, `\D` is the default.

Protocols

You can choose the protocol to use with each device. Usually, it is not needed since you can use the default. If you do specify the protocol, you must do so in the form *Type*,*Protocol*[(*parameters*)] (for example, `TCP,`eg). Available protocols are:

`g` Generic packet protocol. It provides error detection and retransmission intended for use over potentially noisy lines. By its nature, it is relatively slow. Two parameters characterize the `g` protocol, *windows* and *packetsize*. *windows* indicates the number of packets which may be transmitted without waiting for an acknowledgment from the remote host. *packetsize* indicates the number of data bytes in each packet. *windows* value is set at 7, and *packetsize* is set at 64 bytes.

`G` Identical to the `g` protocol in that it provides the same error detection and retransmission. However, in addition, the `G` protocol allows the number of windows and the packet size to be varied to match the characteristics of the transmission medium. When properly configured, performance can be significantly better than the `g` protocol. *windows* may range from 1 to 7, and *packetsize* may range from 32 to 4096 bytes, in powers of 2 (that is, 32, 64, 128, 256, 512, 1024, 2048, 4096).

`e` Assumes error-free transmission and performs no error checking or retransmission. It is therefore the fastest of the three protocols. It should be used for reliable local area networks. There are no parameters to be tuned within the `e` protocol.

The following example uses the `e` protocol over a TCP/IP local area network. If the `e` protocol is not available, `g` will be used.

```
TCP,eg tcp - - TLIS \D
```

This example uses the `G` protocol on a high-speed modem. The number of windows is set to 7, and the packet size is 512 bytes. If the `G` protocol is unavailable, the standard `g` protocol will be used.

```
ACU,G(7,512)g term/11 - 9600 att2296a
```

Presumably, seven windows with a packet size of 512 bytes will provide optimum throughput for the specified device.

For incoming connections, the preferred protocol priority and parameters may be specified in the **`Config`** file using the *Protocol* parameter.

A default **`Devices`** file containing only comment lines is delivered with the system. The system administrator must modify the **`Devices`** file if BNU is to be used.

Serial cu Connections and the Devices File

When **`cu`** is used with a serial connection, the following must be done:

- Add an entry in the **`Devices`** file for **`uudirect`**.
- Add an entry in the **`Systems`** file for each system that will be called. See “Serial cu Connections and the Systems File” for further information.
- Verify that the **`ttymon`** or **`uugetty`** port monitor(s) are in bi-directional mode. See **`ttymon(1M)`**, **`uugetty(1M)`**, and the chapter “Managing Ports” in the *System Administration* guide for further information.

When using **`cu`** with a serial connection, the **`uudirect`** dialer must be used. The following line is the format for the **`uudirect`** dialer in the **`/etc/uucp/Devices`** file:

`UUDirect` *Line Line2 Class* `uudirect`

See the above discussion for appropriate values for the *Line*, *Line2*, *Class* fields.

TCP/IP cu Connections and the Devices File

When using **`cu`** over TCP/IP, the comment character (#) needs to be deleted from the following line in the **`/etc/uucp/Devices`** file:

```
# CS - - - CS
```

so the line now reads,

```
CS - - - CS
```

An entry must be added to the **`Systems`** file for each system that will be called. See “TCP/IP cu Connections and the Systems File” on page 7-38 for further information.

The Dialers File

The **`Dialers`** file (**`/etc/uucp/Dialers`**) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of character strings that is transmitted and expected, and it is often used to dial a telephone number using an Automatic Call Unit.

NOTE

This file works closely with the **`Devices`**, **`Systems`**, and **`Dialcodes`** files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to a related entry in another file.

The fifth and subsequent odd numbered fields in the **`Devices`** file are indexes into the **`Dialers`** file or internal list of special dialer types (CS, 801, TLI, or TLIS). If a match is found, the **`Dialers`** entry is interpreted to perform the dialer conversation.

Each entry in the **`Dialers`** file has the following format:

dialer substitutions expect-send `. . .`

The *dialer* field matches the fifth and additional odd numbered fields in the **`Devices`** file. The *substitutions* field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate = and - into whatever the dialer requires for “wait for dial tone” and “pause.”

The remaining *expect-send* fields are character strings. Figure 7-4 shows some character strings distributed with BNU in the **`Dialers`** file.

```
att2212C =+-, "" atzod,o12=y,o4=n\r\c \006 atT\T\r\c ed
att2212c =+-, "" atzod,o12=y,o4=n\r\c \006 atT\T\r\c ed
att2224 =+-, "" \r\c :--: T\T\r\c red
att2224B =+-, "" atT\T\r\c ed
att2224CEO =+-, "" atzod,o12=y,o4=n,\\n3\\c1\\j0\\q0\\g0\r\c \006 atT\T\r\c
    Connected
att2224G =+-, "" atzod,o12=y,o4=n,o1=n\r\c \006 atz\\n3\\c1\\j0\\q0\\g0\r\c
   "" \datT\T\r\c Connected
att2224b =+-, "" atT\T\r\c ed
att2224ceo =+-, "" atzod,o12=y,o4=n,\\n3\\c1\\j0\\q0\\g0\r\c \006 atT\T\r\c
    Connected
att2224g =+-, "" atzod,o12=y,o4=n,o1=n\r\c \006 atz\\n3\\c1\\j0\\q0\\g0\r\c
   "" \datT\T\r\c Connected
att2248A =+-, "" atzod,o12=y\r\c \006 atT\T\r\c Connected
att2248a =+-, "" atzod,o12=y\r\c \006 atT\T\r\c Connected
att2296A =+-, "" atzod,o12=y,o50=y,o51=n,o55=n,o69=n\r\c \006
atz\\n3\\c1\\j0\\q0\\g0
   \r\c "" \datT\T\r\c Connected
att2296a =+-, "" atzod,o12=y,o50=y,o51=n,o55=n,o69=n\r\c \006
atz\\n3\\c1\\j0\\q0\\g0
   \r\c "" \datT\T\r\c Connected
att4000 =,-, "" ATZ\r\p\p OK\r ATZ\r OK\r\c \EATDT\T\r\c CONNECT
att4024 =+-, "" atzod,o12=y,o4=n\r\c \006 atT\T\r\c ed
datakit "" "" \d TION:--TION: \D
develcon "" "" \pr\ps\c est:\007 \E\D\e \n\007
direct
direct_cr1 "" "" \r\d INVOKE "cr1 -r"
direct_modem "" "" \M\d
hayes =,-, "" \M\dAT\r\c OK\r \EATDT\T\r\c CONNECT \r\m\c
HayesSmartm1200 =,-, "" \M\dAT\r\c OK\r ATZ\r\c OK\r ATM0\r\c OK\r ATE1\r\c OK\r
   ATS0=2\r\c OK\r \EATDT\T\r\c CONNECT \r\m\c
HayesSmartm1200B =,-, "" \M\dAT\r\c OK\r ATZ\r\c OK\r ATM1\r\c OK\r ATE1\r\c
    OK\r ATS0=2\r\c OK\r ATC1\r\c OK\r \EATDT\T\r\c CONNECT \r\m\c
HayesSmartm2400 =,-, "" \M\dAT\r\c OK\r AT&F\r\c OK\r ATZ\r\c OK\r ATM0\r\c OK\r
   AT&D2\r\c OK\r AT&C1\r\c OK\r ATS0=1\r\c OK\r \EATDT\T\r\c 00 \r\m\c
HayesSmartm2400B =,-, "" \M\dAT\r\c OK\r AT&F\r\c OK\r ATZ\r\c OK\r
   ATM0\r\c OK\r AT&D2\r\c OK\r AT&C1\r\c OK\r ATS0=1\r\c OK\r \EATDT\T\r\c 00
\r\m\c
micom "" "" \s\c NAME? \D\r\c GO
Multimodemv29 =w-, "" \MAT&F OK\r
AT$BA0&D3&E1&E5&E7$F0&R1$SB19200X4S0=1S11=55\r\c
   OK\r AT&W0 OK\r AT&E2 OK\r ATDT\T\r\c CONNECT \m\c
Multinormalv29 =w-, "" \MAT&F OK\r
AT$BA0&D3&E1&E5&E7$F0&R1$SB19200X4S0=1S11=55\r\c
   OK\r AT&W0 OK\r AT&E0 OK\r ATDT\T\r\c CONNECT \m\c
nls.cu "" "" NLPS:000:001:cu\N\c
nls.uucico "" "" NLPS:000:001:10103\N\c
penril =W-P "" \d > Q\c : \d- > s\p9\c )-W\p\r\ds\p9\c-) y\c : \E\TP > 9\c OK
rixon =&-% "" \r\r\d $ s9\c )-W\r\ds9\c-) s\c : \T\r\c $ 9\c LINE
safari =,-, "" \M\dATZ\r\c OK\r ATM0\r\c OK\r AT&D2\r\c OK\r AT&C1\r\c OK\r
   ATS0=1\r\c OK\r \EATDT\T\r\c CONNECT \r\m\c
tb =W-, "" A\pA\pA\pT OK AT~&F0S0=1S52=2S54=3S110=1S111=30DT\T "CONNECT"\s""
tbq =W-, "" A\pA\pA\pT OK AT~&F0S0=1S52=2S54=3S110=1S111=30M0DT\T "CONNECT"\s""
telebit =W-, "" A\pA\pA\pT OK ats50=255DT\T CONNECT\sFAST
telebit12 =W-, "" A\pA\pA\pT OK ats50=2DT\T CONNECT\s1200
telebit24 =W-, "" A\pA\pA\pT OK ats50=3DT\T CONNECT\s2400
telebitlong =W-, "" A\pA\pA\pT OK ats7=250 OK ats50=255DT\T CONNECT\sFAST
telebitmnp =W-, "" A\pA\pA\pT OK ats95=2 OK ats50=3DT\T CONNECT
uudirect "" "" \r\d in:--in:--in: \d
vadic =K-K "" \005\p *-\005\p-*\005\p-* D\p BER? \E\T\e \r\c LINE
ventel =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

Screen 7-4. Sample Dialers File Entries

The escape characters (those beginning with “\”) used in the **`Dialers`** are explained in the following section, “The Systems File.”

The `att2212c` entry in the **`Dialers`** file is processed in two steps. First, the telephone number argument is translated as follows:

- Any equals sign (=) is replaced by a plus sign (+)
- Any minus sign (`-`) is replaced by a comma (`,`)

The plus sign and comma in the translated telephone number argument have the following values:

+	Wait for dial tone.
,	Pause.

Second, the handshake given by the remainder of the line is interpreted as follows:

`""`	Wait for nothing, that is, proceed to the *expect-send* string.
`atzod`	Enter command mode, reset modem, set options to default.
`o12=y`	Set option `12` to `y` (transparent data mode).
`o4=n\r\c`	Set option `4` to `n` (don't disconnect on received spaces). Terminate with a carriage return but no newline.
`\006`	Wait for acknowledge signal (ACK).
`atT\T\r\c`	Enter command mode. Use tone dialing. Translate the phone number and terminate with a carriage return, but no newline.
`ed`	Expect “`ed`” (as in the last two letters of answer*ed*).

The Systems File

The **`Systems`** file (**`/etc/uucp/Systems`**) must contain an entry for each system BNU is allowed to communicate with. Each entry in the file represents a computer that can be called by the local machine in its role as client machine and contains the information the **`Connection Server`** daemon needs to establish a communication link. The Basic Networking software is configured to prevent any remote system not included in the local machine's **`Systems`** file from logging in. That is, the local machine can not communicate with any machine that is not listed in its **`Systems`** file, either in the role of client machine or server machine. See the section on the **`remote.unknown`** file. There may be more than one entry for a particular computer. The additional entries represent alternative communication paths that will be tried in sequential order.

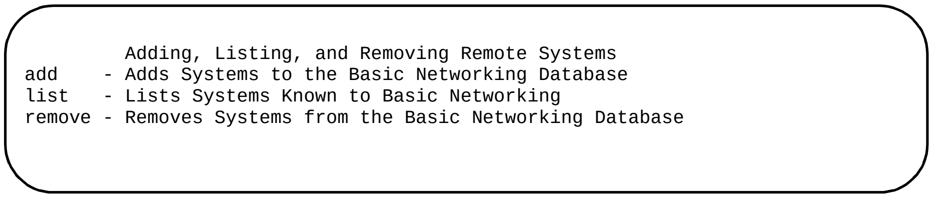
NOTE

> **`Systems`** files from previous releases may be installed without change. This provides backward compatibility without the administrative overhead involved in setting up the Connection Server and cr1.

> The **`Systems`** file works closely with the **`Devices`**, **`Dialers`** and **`Dialcodes`** files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to a related entry in another file.

The BNU System Management Menu, below, can be brought up on the screen by typing

```
sysadm systems
```



```
         Adding, Listing, and Removing Remote Systems
add    - Adds Systems to the Basic Networking Database
list   - Lists Systems Known to Basic Networking
remove - Removes Systems from the Basic Networking Database
```

Screen 7-5. Basic Networking System Management Menu

If the remote machine supports machine authentication and is connected via a non-Connection Server connection, the entry in the **`Systems`** file must include the appropriate `INVOKE`-*scheme* pair or pairs to enforce the use of an authentication scheme. For the current release, if the `cr1` authentication scheme is not included in the **`Systems`** file, passwords will be passed between machines in clear text. In this case, the entry in the **`Systems`** file will be:

```
INVOKE "cr1 -r"
```

NOTE

> More than one file can be defined as a **`Systems`** file by using the **`Sysfiles`** file. See the description of the **`Sysfiles`** file for details.

Each entry in the **`Systems`** file has the following format:

System-Name Time Type Class Phone Login

These fields are defined as:

System-name
: Contains the node name of the remote computer.

Time Specifies when the remote system can be used. *Time* is a string that specifies both the day of the week and the time of day when the remote system can be called. The format of the *Time* field is:

daytime[*;retry*]

day may be a list containing some of the following:

`Su Mo Tu We Th Fr Sa`
for individual days

`Wk` for any weekday (Mo Tu We Th Fr)

`Any` for any day

`Never` for a passive arrangement with the remote computer. If the *Time* field is `Never`, the local computer will never initiate a call to the remote computer. The call must be initiated by the remote computer. In other words, the local computer is passive with respect to the remote computer. (See “The Permissions File” on page 7-10.)

time should be a range of times specified in 24-hour notation, for example 0800-1230 for “8:30 A.M. to 12:30 P.M.” If no *time* portion is specified, any time of day is assumed to be allowed for the call. A time range that spans 0000 is permitted. For example, `0800-0600` means all times are allowed other than times between 6 A.M. and 8 A.M.

An optional subfield, *retry*, is available to specify the minimum time (in minutes) before a retry, following a failed attempt. The default wait is five minutes. The subfield separator is a semicolon (;). For example, `Any;9` is interpreted as call any time, but wait at least nine minutes before retrying after a failure occurs. The following is a sample *time* entry:

```
Wk1700-0800,Sa,Su
```

The example allows calls from 5:00 P.M. to 8:00 A.M., Monday through Friday, and calls any time Saturday and Sunday. In this example, calls are allowed only when telephone rates are low.

`Any` and `Never` are the most common entries in the *Time* field.

Type The *Type* field contains the device type used to establish a communication link with a remote computer. It is used to access the entry in the **`Devices`** file. The **`Systems`** file *Type* entry is matched against the first field of the **`Devices`** file entries.

The protocol used to contact the system can be defined by adding the protocol identifier to the *Type* field. If the *Type* is `CS` and the protocol string in the **`Devices`** file specified that the `e` or `g` protocol should be used, the string entered in the *Type* field of the **`Systems`** file could be **`CSeg`**. The use of the protocol information from the **`Devices`** file as part of the *Type* entry in the **`Systems`** file is optional but it is often included to make these files easier to understand. The device type and the protocols must be comma separated. In the **`Systems`** file example below, the protocol `g` is attached to the device type `ACU`. See “Protocols” on page 7-27.

```
#Systems file
#
eagle Any ACU,g D1200 3251 ogin: nuucp ssword: XXXXXX
```

NOTE

Replace `XXXXXX` in the examples with the appropriate password.

```
#Devices file
#
ACU tty11 - D1200 penril
```

Class Specifies the transfer speed of the device used in establishing the communication link. It may contain a letter and speed (for example, C1200, D1200) to differentiate between classes of dialers (see the discussion of the *Class* field under “The Devices File” on page 7-22). Some devices can be used at any speed. In this case the keyword `Any` may be used. The *Class* field in the **`Systems`** file must match the *Class* field in the associated **`Devices`** file entry. For example:

```
#Systems file
#
eagle Any ACU D1200 NY3251 ogin: nuucp ssword: XXXXXX
```

```
#Devices file
#
ACU tty11 - D1200 penril
```

If information is not required for this field, a `-` should be placed in the file as a placeholder for the field.

Phone Specifies the telephone number or network address of the remote computer. A telephone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the **Dialcodes** file. For example:

```
#Systems file
#
eagle Any ACU D1200 NY3251 ogin: nuucp \
           ssword: XXXXXX
```

```
#Dialcodes file
#
NY 9=1212555
```

In this string, an equal sign (`=`) tells the ACU to wait for a secondary dial tone before dialing the remaining digits. A dash (`-`) in the string instructs the ACU to pause four seconds before dialing the next digit.

If your computer is connected to a LAN switch, you may access other computers that are connected to that switch. The **Systems** file entries for these computers will not have a telephone number in the *Phone* field. Instead, this field will contain the token that must be passed to the switch so it will know which computer your computer wishes to communicate with (this is usually just the system name). The associated **Devices** file entry should have a **\D** at the end of the entry to ensure that this field is not translated using the **Dialcodes** file.

If the *Type* entry is `CS`, any information in the *Phone* field is passed to the `dial`() routine. In this case, the *Phone* field may consist of *system,service*, where either or both may be "`-`" or the *Phone* field may contain a single "`-`" character. If neither *system* nor *service* is specified, the *machine-name* (the first field in the line) and the *service-name* (**uucico** or **cu**) will be passed as parameters to `dial`().

Login This field contains login information given as a series of fields and subfields of the format:

expect send

where *expect* is the string that is received and *send* is the string that is sent when the *expect* string is received.

The *expect* field may be made up of subfields of the form:

expect[-*send*-*expect*] . . .

where the *send* is sent if the prior *expect* is not successfully read and the *expect* following the *send* is the next expected string. For example, with `ogin:--ogin:`, **uucp** will expect `login`. If **uucp** gets `ogin:`, it will go on to the next field. If it does not get `ogin:`, it will send nothing followed by a newline, then look for `ogin` again. If no characters are initially expected from the remote computer, the characters `""` (null string) should be used in the first *expect* field. Note that all *send* fields will be sent followed by a newline unless the *send* string is terminated with a `\c`.

NOTE

The “`l`” and other initial characters are often omitted from chat scripts since chat scripts are case-sensitive. Some systems capitalize the first letter of prompts while others do not.

Here is an example of a **`Systems`** file entry that uses an expect-send string:

```
owl Any ACU 1200 Chicago6013 "" \r ogin:-BREAK-ogin: \
uucpx word: xyzzy
```

This example says don't wait, just send a carriage return and wait for `ogin:` (for `Login:`). If you don't get `ogin`, send a `BREAK`. When you do get `ogin:` send the login name **`uucpx`**, then when you get `word:` (for `Password`:), send the password `xyzzy`.

There are several escape characters that cause specific actions when they are a part of a string sent during the login sequence. The following escape characters are useful when using BNU communications:

`\b`	Send or expect a backspace character
`\c`	If at the end of a string, suppress the newline that is normally sent. Ignored otherwise.
`\d`	Delay two seconds before sending or reading more characters
`\p`	Pause for approximately ¼ to ½ second
`\E`	Start echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.)
`\e`	Echo check off
`\M`	Turn on `CLOCAL` flag
`\m`	Turn off `CLOCAL` flag
`\n`	Send a newline character
`\r`	Send or expect a carriage-return
`\s`	Send or expect a space character
`\t`	Send or expect a tab character
`\\`	Send or expect a \ character
`BREAK`	Send or expect a BREAK character

`\D`	Telephone number or token without `Dialcodes` translation
`EOT`	Send or expect EOT newline twice
`\K`	Same as BREAK
`\N`	Send or expect a null character (ASCII NUL)
`\T`	Telephone number or token with `Dialcodes` translation
`\ddd`	Collapse the octal digits (ddd) into a single character
~*nn*	Specify the timeout by appending *nn* to the expect string, where *nn* is the timeout time in seconds.

The *Login* field may contain the keyword `INVOKE` followed by white space followed by the name of an authentication scheme. This pair may appear several times in the *Login* field of a **`Systems`** file entry if more than one authentication step is expected to take place. The following is an example of a **`Systems`** file entry that specifies more than one authentication step:

```
systemX Any ACU 1200 95551234 INVOKE "inap -r" \
    INVOKE "cr1 -r"
```

`systemX` is understood to be a valid system name.

If the `expect` string consists of the keyword `ABORT`, then the string after it is used to arm an abort trap. If that string is subsequently received any time prior to the completion of the entire **`expect/send`** script, then **`uucico`** aborts, as it would if the script had timed out. This is useful for trapping error messages—such as `Host Unavailable` or `System is Down`— from port selectors or front-end processors. Note that in an **`expect/send`** script, the format in which the `ABORT` appears is backward: `ABORT expect` instead of `expect ABORT`.

NOTE

The entire *Login* field is ignored for **`cu`** and **`ct`** service requests since the user is expected to enter the proper connect sequence manually. This means that *Login* chat scripts that include an `INVOKE` statement will also be ignored when the service being requested is **`cu`** or **`ct`**.

The system administrator must create the entries in the **`Systems`** file. A **`Systems`** file is delivered with the system but contains only comment lines.

Serial cu Connections and the Systems File

When **`cu`** is used with a serial connection, the following must be done:

- Add an entry in the **`Devices`** file for **`uudirect`**. See “Serial cu Connections and the Devices File” on page 7-28.
- Add an entry in the **`Systems`** file for each system that will be called.

- Verify that the **`ttymon`** or **`uugetty`** port monitor(s) are in bi-directional mode. See **`ttymon(1M`**), **`uugetty(1M)`**, and the chapter “Managing Ports” in the *System Administration* guide for further information.

When using **`cu`** with a serial connection, the **`uudirect`** dialer must be used.

System-Name *Time* `UUDirect` *Class* *Phone* *Login*

See the above discussion for appropriate values for the *System-Name*, *Time*, *Class*, *Phone*, and *Login* fields.

TCP/IP cu Connections and the Systems File

When using **`cu`** over TCP/IP, an entry for each system that is to be called must be added to the **`Systems`** file. The entry has the following format:

System-Name *Time* `CS - -,listen:cu`

See the above discussion for appropriate values for the *System-Name* and *Time* fields.

The **`Devices`** file must also be modified to support **`cu`** over TCP/IP. See “TCP/IP cu Connections and the Devices File” on page 7-28 for further information.

The Dialcodes File

The **`Dialcodes`** file (**`/etc/uucp/Dialcodes`**) contains the dial-code abbreviations that can be used in the *Phone* field of the **`Systems`** file.

NOTE

The **`Dialcodes`** file works closely with the **`Devices`**, **`Dialers`** and **`Systems`** files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to a related entry in another file.

Each **`Dialcodes`** entry has the format:

abb dial-seq

where *abb* is the abbreviation used in the **`Systems`** file *Phone* field and *dial-seq* is the dial sequence that is passed to the dialer when that particular **`Systems`** file entry is accessed.

The entry

```
jt 9=555-
```

would be set up to work with a *Phone* field in the **`Systems`** file such as `jt7867`. When the entry containing `jt7867` is encountered, the sequence `9=555-7867` would be sent to the dialer if the token in the dialer-token pair is `\T`.

The default **`Dialcodes`** file delivered with the system is empty. It is not necessary for the system administrator to do anything to the file.

The Poll File

The **`Poll`** file (**`/etc/uucp/Poll`**) contains information for polling remote computers. Each entry in the **`Poll`** file contains the name of a remote computer to call, followed by a tab character (a space won't work), and finally the hours the computer should be called.

The format for entries in the **`Poll`** file is:

> *sys-name*`<tab>`*hour*`. . .`

For example the entry:

```
eagle   0       4       8       12      16      20
```

polls the computer `eagle` every four hours.

The **`uudemon.poll`** script does not actually perform the poll. It merely sets up a polling work file (always named **`C.`***file*) in the spool directory. **`uudemon.hour`** starts the scheduler; it is the scheduler that examines all work files in the spool directory and does the polling.

A default **`Poll`** file is delivered with the system and does not need to be modified.

The Grades File

The **`Grades`** file (**`/etc/uucp/Grades`**) contains the definitions for the job grades that may be used to queue jobs to a remote computer. It also contains the permissions for each job grade. Each entry in this file represents a definition of an administrator defined job grade that allows users to queue jobs.

Each entry in the **`Grades`** file has the following format:

> *User-job-grade System-job-grade Job-size Permit-type ID-list*

Each entry in this file contains fields that are separated by white space. The last field in the entry is made up of sub-fields that are also white space separated. If a entry takes up more than one physical line, then a backslash (\), is used to continue the entry onto the following line. Comment lines begin with a pound sign (#) and occupy the entire line. Blank lines are always ignored. Here is a description of each field:

User-job-grade
: This field contains an administrative defined user job grade name of up to 64 characters.

System-job-grade
: This field contains a one character job grade to which *User-job-grade* will be mapped. The valid list of characters is A-Z, a-z, with A having the highest priority and z the lowest.

Job-size This field specifies the maximum job size that can be entered in the queue. *Job-size* is measured in bytes and may be a list of the following:

nnnn	where *nnnn* is an integer that specifies the maximum job size for this job grade
n`K`	where *n* is a decimal number that represents the number of kilobytes and `K` is an abbreviation for kilobyte
n`M`	where *n* is a decimal number that represents the number of megabytes and `M` is an abbreviation for megabyte
`Any`	a keyword to specify that there is no maximum job size

Here are some examples:

5000	represents 5000 bytes
10K	represents 10 kilobytes
2M	represents 2 megabytes

Permit-type
This field contains a keyword that denotes how to interpret the ID list. The following list contains the keywords and their meanings:

`User`	ID list contains the login names of users permitted to use this job grade.
`Non-user`	ID list contains the login names of users not permitted to use this job grade.
`Group`	ID list contains the group names whose members are permitted to use this group.
`Non-group`	ID list contains the group names whose members are not permitted to use this job grade.

ID-list This contains a list of login names or group names that are to be permitted or denied queuing to this job grade. The list of names are separated by white space and terminated by a newline character. The keyword `Any` is used to denote that anyone is permitted to queue to this job grade.

The user job grade may be bound to more than one system job grade. It is important to note that the **`Grades`** file will be searched sequentially for occurrences of a user job grade. Therefore, any multiple occurrences of a system job grade should be listed according to the restriction on the maximum job size.

While there is no maximum number for the user job grades, the maximum number of system job grades allowed is 52. The reason is that more than one *User-job-grade* can be mapped to a *System-job-grade*, but each *User-job-grade* job grade must be on a separate line in the **`Grades`** file. For example:

```
mail    N   Any     User    Any
netnews N   Any     User    Any
```

Given this configuration in a **`Grades`** file, these two *User-job-grade* grades will share the same *System-job-grade*. Since the permissions for a *Job-grade* are associated with a *User-job-grade* and not a *System-job-grade*, it is even possible for two *User-job-grades* to share the same *System-job-grades* and have two different sets of permissions for each one.

Default Grade

The binding of a default *User-job-grade* to a system job grade can be defined by the administrator. The administrator must use the keyword `default` as user job grade in the *User-job-grade* field of the **`Grades`** file and the system job grade that it is bound to. The restrictions and ID fields should be defined as `Any` so that any user and any size job can be queued to this grade. Here's an example:

```
default   a      Any      User      Any
```

If the default user job grade is not defined by the administrator, then the built-in default grade, `Z`, will be used. Because it is assumed that the restriction field is `Any,` multiple occurrences of the default grade are not checked.

The **`Grades`** file delivered with the system contains the following three default grades:

```
high    F      Any      User      Any
medium  S      Any      User      Any
low     n      Any      User      Any
```

The file must be administered only if you are using grades. If you are not using grades, it is not necessary to modify the file.

Administrative Support Files

The Basic Networking administrative files are described below. These files are created in spool directories to hold temporary data or store information about remote transfers or executions.

`TM.` (temporary data file)

These data files are created by Basic Networking processes under the spool directory (that is, **`/var/spool/uucp/`***X*) when a file is received from another computer. The directory *X* has the same name as the remote computer that is sending the file. The names of the temporary data files have the format:

`TM.`*pid.ddd*

where *pid* is a process-ID and *ddd* is a sequential three-digit number starting at 0.

When the entire file is received, the **`TM.`***pid.ddd* file is moved or copied to the pathname specified in the **`C.`***sysnxxxx* file (discussed below) that caused the transmission. If processing is abnormally terminated, the **`TM.`***pid.ddd* file may remain in the *X* directory. These files should be automatically removed by **`uucleanup`**.

`P.` (checkpoint file)

A checkpoint file is created when processing terminates abnormally. Unlike the **`TM.`***pid.ddd* file, it is not removed. Therefore, when the transfer session is re-established, the length of the **`checkpoint`** file serves as an appropriate place to restart the transfer of the file, instead of restarting from the beginning. When checkpointing is specified for a file being transferred from another computer, the checkpoint file is used instead of a **`TM`** file. **`checkpoint`** files are created in the spool directory. Checkpointing occurs only between two systems that have the BNU enhancements introduced in the OS.The names of the checkpoint files have the following format:

`P.`*systmxxxxyyy*

where *systm* is the first five characters in the name of the remote computer, *xxxx* is a four-digit job sequence number assigned by **`uucp`**. The four digit job sequence number may be followed by a sub-sequence number, *yyy*, which is used when there are several **`P.`** files created for a work (**`C.`**) file. When the entire file is received, the **`P.`***systmxxxxyyy* file is moved to the pathname specified in the **`C.`***sysnxxxx* file (discussed above) that caused the transmission.

`LCK.` (lock file)

These lock files prevent duplicate conversations and transfers. The names have the form:

`LCK.`*system.grade*

where *system* is the name of the remote system and *grade* is the *System-Job-grade* being processed. The lock file contains the process ID of the process holding the lock. This process ID remains valid as long as the process is active.

`C.` (work file)

Work files are created in a spool directory when work (file transfers or remote command executions) has been queued for a remote computer. The names of work files have the format:

`C.`*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the ASCII character representing the grade (priority) of the work, and *xxxx* is the four digit job sequence number assigned by **`uucp.`** Work files contain the following information:

- Type of request, `S` (send) or `R` (receive)
- Pathname of the file to be sent or received
- Pathname of the destination or user file name
- User login name
- List of options
- Name of associated data file in the spool directory. If the **`uucp -c`** or **`uuto -p`** option was specified, a dummy name may be used

- Mode bits of the source file
- Remote user's login name to be notified upon completion of the transfer

`D.` (data file)

Data files are created when it is specified in the command line to copy the source file to the spool directory. The names of data files have the following format:

`D.`*systmxxxxyyy*

where *systm* is the first five characters in the name of the remote computer and *xxxx* is a four-digit job sequence number assigned by **`uucp.`** The four digit job sequence number may be followed by a sub-sequence number, *yyy*`,` which is used when there are several **`D.`** files created for a work (**`C.`**) file.

`D.` (Authentication file)

The authentication file is an encrypted (that is, non-readable) file that contains information needed for the authentication and identification of remote users making **`uux`** requests. The file is interpreted by **`uuxqt`**.

`X.` (**`Execute`** file)

`Execute` files are created in the spool directory prior to remote command executions. The names of **`Execute`** files have the following format:

`X.`*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the character representing the grade (priority) of the work, and *xxxx* is a four digit number assigned by **`uucp`**. **`Execute`** files contain the following information:

- requester's login and computer name
- name of file(s) required for execution
- input file to be used as the standard input to the command string
- computer and file name to receive standard output and standard error from the command execution
- command string
- option lines for return status requests

Log Files

The BNU commands provide eight logs, some of which are optional. They are described below.

Command Log

The command log contains the commands issued by the user, the administrator, and the operator. It can help the system administrator in trouble-shooting. The full path name of the command log is **`/var/spool/uucp/.Admin/command`**. The format of each entry is as follows:

$$\underbrace{\texttt{user1}}_{\texttt{a}}\ \underbrace{\texttt{(5/16-19:00:31)}}_{\texttt{b}}\ \underbrace{\texttt{uucp test.c mach1!}{\sim}\texttt{/user2}}_{\texttt{c}}$$

KEY:

a	user login name
b	date and time the command was issued
c	command line

System History Log

The system history log contains a record of each action that alters the state of the system and queue. The system history log can be generated by **`uucp`**, **`uucico`**, **`uux`**, or **`uuxqt`** programs and put into the **`uucp`**, **`uucico`**, **`uux`**, **`uuxqt`** subdirectories of the **`/var/spool/uucp/.Log`** directory. Below is a sample entry that **`uucico`** writes into **`/var/spool/uucp/.Log/uucico/mach1`**.

```
uucp mach1 (2/12-8:18:42, 2427, 0) CONN FAILED
└──┘ └───┘└┘└────────────┘ └────┘ └┘ └──────────┘
 a     b   c       d          e    f       g
```

.

```
(NO DEVICES AVAILABLE)
└────────────────────┘
          h
```

KEY:

`a`	user who submitted the job
`b`	name of the remote machine
`c`	job ID if a job is currently being processed; otherwise null, as in this line
`d`	date and time that the entry was written to the file
`e`	process ID of **`uucico`** in this example
`f`	file transfer sequence number within the current invocation of **`uucico`**
`g`	status message
`h`	status message elaboration

Error Log

The error log contains the error messages in the network. Error messages appear in the **`/var/spool/uucp/.Admin/errors`** file. When the errors occur, the program aborts. In most cases, this results from file-system problems. A typical entry is as follows:

```
ASSERT ERROR (uucico) pid: 1513 (5/15-9:03:45) can’t open
     a          b         c            d            e

system 3 (FILE:pkl.c, LINE:356
   f   g       h          i
```

KEY:

`a`	error type
`b`	program name
`c`	process ID
`d`	date and time that this entry was written to the file
`e`	error message part 1
`f`	error message part 2
`g`	error number
`h`	name of calling module
`i`	line of calling module where the error occurred

Transfer Log

The transfer log contains information pertaining to file transfer. For example, it shows the number of bytes transferred and how long the transfer took. After both **`uucicos`** (master and slave) agree on the protocol, the transfer information of each file is written into **`/var/spool/uucp/.Admin/xferstats`**. A typical entry is as follows:

```
ihx! user M (5/20-6:10:10) (C, 6559, 1) [DKH]
 a     b  c       d         e     f    g    h


-> 1024/0.13 secs, 7877 bytes/sec ""
        i                j         k
```

KEY:

`a`	name of remote system
`b`	user login
`c`	M=master, S=slave
`d`	date and time that entry was written to log
`e`	C=uucico, U=uucp, X=uux, Q=uuxqt
`f`	process ID of **`uucico, uucp, uux,`** or **`uuxqt`**
`g`	sequential number for each file transferred in a session
`h`	device name of media
`i`	direction and data bytes / clock time to transfer
`j`	transfer rate (bytes/sec)
`k`	`"PARTIAL FILE"` if the file was not completed because of transmission error; `""` if the file was transmitted completely (as in this record)

Report Statistics of File Transfer

By specifying the **-s***file* option with **uucp**, you can have the statistics of your file transfer reported to *file*. For each file transfer request, *file* will contain three lines. For multiple file transfer requests, the size of *file* increases accordingly. A typical entry is:

First line:

```
uucp job: mach1N1131 (6/12-13:34:58) (0:0:19)
    a          b            c            d
```

KEY:

a	message header (always the same)
b	job ID assigned by **uucp**
c	date and time the file transfer completed
d	*hh:mm:ss* of elapsed time between time of creation of queue file and completion of the file transfer

Second Line:

```
ihnp1! /n15/user1/liblog -> mach1! ~/userid (user1)
   a           b          c    d       e       f
```

KEY:

a	sender node name
b	source file name
c	direction (always the same)
d	receiver node name
e	destination directory/file name
f	requester login ID

Third line (status):

```
copy succeeded
       a
```

KEY:

a	a message

Accounting Log

The accounting log contains information needed for network charging. Upon completing a successful job transaction, accounting information is written to file **`/var/spool/uucp/.Admin/account`**. If the job transaction is a file transfer, then accounting information is written to **`/var/spool/uucp/.Admin/account`** on the requesting site. If the job transaction is a remote execution, then accounting information is written to **`/var/spool/uucp/.Admin/account`** on the executing (target) site.

Accounting is optional. Accounting information is collected only if an account file exists and is writable by **`uucp`**; the file is not created automatically. The system administrator can therefore start or stop accounting by creating or removing the accounting log. The following is a typical accounting log entry:

```
5432 mach1N11152 4514 C S A ihnp1 user1 (5/20-3:10:12)
  a       b        c  d e f   g     h         i

mach1 user2 DKH "" xfer ""
  j     k    l  m   n   o
```

KEY:

`a`	user ID
`b`	job ID assigned by **`uucp`**
`c`	size of job in bytes if the job transaction is a file transfer; time of job in seconds if the job transaction is a remote execution
`d`	C = job completed. P = partial job completed.
`e`	service class of the job, namely: premium, standard, or economy. At present, only “standard” service class is supported.
`f`	job grade identification
`g`	originating system's name
`h`	originator's login name
`i`	date and time the job originated
`j`	destination system's name
`k`	destination user's login name
`l`	device name of media
`m`	ID of physical network (always `“”`)
`n`	type of transaction: xfer = file transfer, rexe = remote execution.
`o`	The command if the job transaction is a remote execution.

Security Log

The security log contains the job transactions that attempt to violate system and user security measures. It is used to aid in detecting attacks on the systems. An attempted security violation is detected when the requester fails to pass the security checks specified in **`/etc/uucp/Permissions`** or tries to access a protected source or destination file. The occurrence is logged for further analysis in **`/var/spool/uucp/.Admin/security`**. Two different entries can appear in the security log.

`xfer`	file transfer
`rexe`	remote execution

Their formats are as follows:

File Transfer (xfer) Security Log

```
xfer ihnp1 user1 mach1 user2 uucp.c ihnp1 user1 uucp.c
 a     b     c     d     e     f      g     h     i

34567 (5/19-16:10) (5/20-11:10:29) (5/20-11:18:20)
  j        k              l                m
```

KEY:

`a`	record type (always `xfer`)
`b`	requester node name
`c`	requester user login
`d`	destination node name
`e`	destination user login
`f`	destination file name
`g`	source node name
`h`	source file owner login
`i`	source file name
`j`	source file size in bytes
`k`	modification date and time of source file
`l`	date and time that transfer started
`m`	date and time that transfer completed

Remote Execution (rexe) Security Log

$$\underbrace{\texttt{rexe}}_{a}\ \underbrace{\texttt{ihnp1}}_{b}\ \underbrace{\texttt{user1}}_{c}\ \underbrace{\texttt{user2}}_{d}\ \underbrace{\texttt{(5/20-15:28:32)}}_{e}\ \underbrace{\texttt{(pwd)}}_{f}$$

KEY:

`a`	record type (always rexe)
`b`	client (requesting) node name
`c`	client (requesting) user login
`d`	server (destination) user login
`e`	date and time that command was executed by server
`f`	command name and options

Performance Log

The performance log contains statistics about the operation of **`uucico`**. **`uucico`** writes the log entries to **`/var/spool/uucp/.Admin/perflog`**. Statistics are collected only if **`perflog`** exists when **`uucico`** starts and is writable by **`uucico`**. The **`perflog`** file is not created automatically. The system administrator can therefore start or stop the collection of performance statistics by creating or removing the performance log file, or by making it writable or not writable by **`uucico`**. In processing terms, if the performance log file can be opened for writing, logging is performed; if the file cannot be opened for writing, logging is not performed. In either case, processing continues.

Two types of records are written to the file; each is identified by a mnemonic type at the beginning of the record. The fields of a record are separated by the "`|`" character. The record types are:

`conn`	contains statistics about the successful establishment of a connection
`xfer`	contains statistics about a file transfer

NOTE

Fields that are not applicable or unknown are marked by double quotes (`""`).

Their formats are as follows:

Connection (conn) Performance Log

$$\underbrace{\texttt{conn}}_{a}\ |\ \underbrace{\texttt{860516175047}}_{b}\ |\ \underbrace{\texttt{23517}}_{c}\ |\ \underbrace{\texttt{mach1}}_{d}\ |\ \underbrace{\texttt{M}}_{e}\ |\ \underbrace{\texttt{ihnp1}}_{f}\ |\ \underbrace{\texttt{DKH}}_{g}\ |$$

$$\underbrace{\texttt{d}}_{h}\ |\ \underbrace{\texttt{""}}_{i}\ |\ \underbrace{\texttt{20.85}}_{j}\ |\ \underbrace{\texttt{3.15}}_{k}\ |\ \underbrace{\texttt{0.94}}_{l}$$

KEY:

`a`	record type (always conn)
`b`	time stamp *YYMMDDhhmmss* for sorting
`c`	**`uucico`**'s process ID
`d`	the name of the machine where the record was written
`e`	M = master, S = slave
`f`	name of remote system
`g`	device name of media
`h`	the protocol that was used for communications
`i`	physical network ID (always `""`)
`j`	real time to connect
`k`	user time to connect
`l`	system (kernel) time to connect

File Transfer (xfer) Performance Log

```
xfer | N | 860516175051 | 23517 | mach1 | M | ihnp1 | DKH |
└───┘  └─┘  └────────────┘  └─────┘  └─────┘  └┘  └─────┘  └───┘
  a     b         c            d        e      f     g       h

d | "" | ihnp1N2c76 | 118.00 | 121.00 | 1000 | -dc | 0.08 |
└┘  └──┘  └──────────┘  └──────┘  └──────┘  └────┘  └───┘  └────┘
i    j        k            l         m        n      o      p

0.00 | 0.04 | 0.91 | 0.06 | 0.13 | 0.22 | 0.02 | 0.06 | "" |
└───┘  └────┘  └────┘  └────┘  └────┘  └────┘  └────┘  └────┘  └──┘
  q      r       s       t       u       v       w       x      x
```

KEY:

`a`	record type (always `xfer`)
`b`	job grade ID
`c`	time stamp (*YYMMDDhhmmss*) for sorting
`d`	**`uucico`**'s process ID
`e`	the name of the machine where the record was written
`f`	M = master, S = slave
`g`	name of remote system
`h`	device name of media
`i`	the protocol that was used for communications
`j`	physical network ID (always `""`)
`k`	job ID if master, `""` if slave
`l`	time in seconds that job was in queue if master, `""` if slave
`m`	turn around time in seconds if master, `""` if slave
`n`	size of the file that was actually transferred successfully or partially because of transmission error
`o`	command line options if master, `""` if slave
`p`	real time to start up transfer
`q`	user time to start up transfer
`r`	system (kernel) time to start up transfer
`s`	real time to transfer file

`t`	user time to transfer file
`u`	system (kernel) time to transfer file
`v`	real time to terminate the transfer
`w`	user time to terminate the transfer
`x`	system (kernel) time to terminate the transfer
`y`	`"PARTIAL FILE"` if the file was not completed because of transmission error; `""` if the file was transmitted completely (as in this record)

NOTE

Fields that are not applicable or unknown are marked by double quotes (`""`).

Start-up time includes the time for the master to search the queues for the next job, for the master and slave to exchange work vectors, and the time to open files.

Transfer time is the time it takes to transfer the data, close the file, and exchange confirmation messages.

Termination time is the time it takes to send mail notifications and write status files.

Turnaround time is the difference between the time that the job was queued and the time that the final notification was sent.

Foreign Log

The foreign log is a list of unknown systems that attempted to connect to the current machine. The list appears in the **`/var/spool/uucp/.Admin/Foreign`** file. The format produced by **`remote.unknown`** is as follows:

$$\underbrace{\texttt{Wed Jan 16 15:44:20 1987:}}_{\texttt{a}}\ \underbrace{\texttt{call from system } \textit{machine} \texttt{ login } \textit{logname}}_{\texttt{b}}$$

KEY:

a	date and time that the entry was written to the file
b	the message logged by **`remote.unknown`**

Adding uucp Logins

You may add one or more logins to your system so incoming **`uucp`** (**`uucico`**) requests from different remote machines can be handled differently. Each remote machine should have an entry in its **`Systems`** file for your machine that contains the login ID and password that you add to your **`/etc/passwd`** file.

The default entry in the **`/etc/passwd`** file is shown below.

```
uucp:x:5:5:0000-uucp(0000):/usr/lib/uucp
nuucp:x:10:10:0000-uucp(0000):/var/spool/uucppublic:/usr/lib/uucp/uucico
```

This entry shows that a login by **`nuucp`** is answered by **`/usr/lib/uucp/uucico`**. The home directory is **`/var/spool/uucppublic`**. The *x* indicates that the encrypted password is stored in **`/etc/shadow`**.

Notice that the standard shell is not given to the **`nuucp`** login. The shell that **`nuucp`** receives is the **`uucico`** daemon that controls the conversation when a remote machine logs in to your machine.

The assigning of passwords for the **`uucp`** and **`nuucp`** logins is left up to the administrator. The passwords should be at least six to eight characters. Only the first eight characters of the passwords are significant. If the password for the access login is changed for security reasons, make certain that the remote machines that are a part of your network are properly notified of the change.

NOTE

The **`uucp`** login is for local administration only. The password should not be given to remote administrators.

If a login is used to transfer files at multiple levels, those levels must be valid login levels for that login.

BNU Setup Procedures

This section describes the steps in the BNU setup procedure. These steps fall into three groups:

- The steps required in every case; see section (1), below.
- The additional steps required if you are using the new CS dialer type (the Connection Server); see sections (2a) and (2b), below.
- The additional steps required if you are not using the new CS dialer type; see sections (3a) and (3b), below.

BNU setup thus always requires the steps in section (1) and, in addition, will require the steps in either section (2) or section (3).

Sections (2) and (3) are divided into client and server machine subsections; (2a) and (3a) describe client machine administration and (2b) and (3b) describe server machine administration. Each subsection presents essential steps first, followed by optional or conditional steps.

(1) For All Types of Connection

- To use BNU, you must set up the necessary BNU database support files. This is true whether you are using the Connection Server or not. The following database files must be modified for your system or you must determine that the delivered default files are appropriate. If these files exist and were used on your system with a previous release, you may use the existing files without alteration.
  - **`Sysfiles`**
  - **`Systems`**
  - **`Devices`**
  - **`Dialers`**
  - **`Devconfig`**
  - **`Config`**

There can be multiple **`Systems`**, **`Devices`**, and **`Dialers`** files registered in the **`Sysfiles`** file.

- If the Enhanced Security Utilities are installed and running on your system, the following two files must also be created. You will be unable to use BNU on a secure system without them.
  - **`RLID.map`**
  - **`LIDAUTH.map`**

  See Chapter 6, “Administering ID Mapping” and Chapter 4, “Administering the Connection Server” for a description of how to set up these files.

(2a) For Connections Using the New CS Dialer Type - the Client Machine

Essential Steps

- Set up **`hosts`** and **`services`** files. These files are needed to perform the authentication required by the server machine. See Chapter 3, “Administering Name-to-Address Mapping” for a description of how to administer the **`hosts`** and **`services`** files.

Optional or Conditional Steps

- Set up a **`/etc/iaf/serve.allow`** file. Set up an **`/etc/iaf/serve.allow`** file that includes a line associating an authentication scheme with the **`uucico`** network service.The **`/etc/iaf/serve.allow`** file allows the administrator of the client machine to enforce the use of specific authentication schemes for specific services and machines. The following line from a sample **`/etc/iaf/serve.allow`** file associates the cr1 bilateral authentication scheme with the **`uucico`** network service.

```
# machine   service   scheme
elvis       uucico    cr1
```

  Without this entry, any authentication scheme specified by the server will be accepted. This may result in the transmission of sensitive data (passwords) across the network in clear text.

  See Chapter 4, “Administering the Connection Server” for a description of the **`/etc/iaf/serve.allow`** file.

- Specify authentication for **uux**. If you want to use the new features of the **uux** command, specify the authentication you want performed by making the appropriate entries in the **Config** file. For example:

  ```
  AUTH=req KEYS=cr1
  ```

  For the current release, cr1 key management is the only key management mechanism available to BNU for use in authenticating remote execution requests.

  BNU uses the `KEYS=` and `AUTH=` values in the **Config** or **Permissions** files. If **uux** can obtain a key for the user, it spools a request including all the necessary authentication information. If it cannot obtain a key and the value of `AUTH` is `opt` or `no`, it spools the request without authentication information. Otherwise, the request fails.

- Perform cr1 key administration. If you are using cr1 as your authentication scheme or for **uux** key management, you must perform cr1 key administration. See Chapter 5, “cr1 Bilateral Authentication Scheme.”

- `LIDAUTH` Mapping. If you have installed the Enhanced Security Utilities, you must create the LID authorization file **LIDAUTH.map.** See Chapter 4, “Administering the Connection Server.”

(2b) For Connections Using the New CS Dialer Type - the Server Machine

Essential Steps

- Register services. Set up the **_pmtab** files for port monitors that expect requests for network services. The **_pmtab** files are administered using the **pmadm** [see **pmadm(1M)]** command. Two services in particular must be registered on the server machine: **uucico** and **reportscheme**. See the chapter “Managing Ports” in the *System Administration* guide for a description of registering services and Chapter 4, “Administering the Connection Server” in this chapter for a description of the **reportscheme service**.

- Register **uucico** services. A system that expects to be a server in BNU transactions must register a **uucico** service for each port monitor that receives BNU service requests. Each **uucico** service registered should include an authentication scheme name in the *scheme* field of the port monitor administrative file for the port.

  The **uucico** service is added using the **pmadm** command. **pmadm** is described in detail in the chapter “Managing Ports” in the *System Administration* guide and on **pmadm(1M).** The following **pmadm** command adds a line for **`uucico`** to the port monitor's **`_pmtab`** file. In this case, the port monitor is `tcp`.

```
pmadm -a -p tcp -s uucico -m "`nlsadmin -c \"/usr/lib/uucp/uucico -r 0 \
-A Addr\"`" -v `nlsadmin -V`-S "cr1 -suucico" -y"uucp"
```

NOTE

Replace *Addr* in the example with the address on which the **`uucico`** service will be offered.

- Register **`reportscheme`** services. Add a **`reportscheme`** service entry to the appropriate port monitor administrative files for each transport on which you want to support BNU. See Chapter 4, “Administering the Connection Server” and **`reportscheme(1M)`** for a description of the **`reportscheme`** service.

Optional or Conditional Steps

- Perform cr1 key administration. If you are using the cr1 authentication scheme, you must perform cr1 key administration. See Chapter 5, “cr1 Bilateral Authentication Scheme.”
- Perform ID mapping. See Chapter 6, “Administering ID Mapping.”

NOTE

Remote IDs should not be mapped to **`uucp`**. Instead, they should be mapped to **`nuucp`** or some other login used exclusively for data transfer.

- Specify authentication for **`uux`**. If you want to use the new features of the **`uux`** command, specify the authentication you want performed by making the appropriate entries in the **`Config`** file. For example,

  ```
  AUTH=req KEYS=cr1
  ```

  For the current release, the cr1 authentication scheme is the only scheme available to BNU for authenticating remote command execution.

  BNU looks for the `KEYS` entry in the **`Config`** file, then gets the key for the user from `cr1`. If `KEYS` has no assigned value, or if **`uux`** fails to get a key and `AUTH=yes` or `AUTH=req`, **`uux`** will fail. Otherwise **`uux`** will proceed without the key, although the results on the remote system may not be the same.

- RLID Mapping. If the Enhanced Security Utilities are installed and running on your system and you want to use BNU, you must create and administer an **RLID.map** file to tell you at what level services run. The **RLID.map** file is created using the **attradmin** command described under Chapter 6, “Administering ID Mapping.”

NOTE

The **nuucp** login must be authorized for each level alias for which you want to allow file transfer. See **usermod(1M)** for more information.

(3a) For Connections Not Using the CS Dialer Type- the Client Machine

Optional or Conditional Steps

- Include an *invoke-scheme* Pair. Include an *invoke-scheme* pair in the appropriate **Systems** or **Dialers** file entry. For example:

```
#Systems file
#
sysa Any DK speed sysa INVOKE "cr1 -r"
sysb Any DKcr1 speed sysb
```

```
#Devices file
#
DK term/11 - Any datakit \D
DKcr1 term/11 - Any datakitcr1 \D
```

```
#Dialers file
#
datakit ""       "" \d TION:—TION: \D
datakitcr1 ""    "" \d TION:—TION: \D INVOKE "cr1 -r"
```

Whether calling sysa or sysb, a Datakit® VCS connection will be made and the cr1 authentication scheme will be invoked in the responder role. See “The Dialers File” on page 7-29 and “The Systems File” on page 7-31 below for field definitions.

- Specify authentication for **`uuxqt`**. If you want to allow the use of the new features of the **`uux`** facility, you must perform this step. For the current release, `cr1` key management is the only key management scheme available to BNU for use in authenticating remote execution requests.

  BNU uses the `KEY=` and `AUTH=` values in the **`Config`** or **`Permissions`** files. If **`uuxqt`** can obtain and successfully decode the authentication information supplied with a request, the request is executed *as the mapped user*. (See Chapter 6, "Administering ID Mapping.") If no authentication information is supplied and the value of `AUTH` is `opt` or `no`, **`uuxqt`** will execute the request as it did in previous releases. If either the decoding of authentication information fails or no authentication was attempted, and the value of `AUTH` is `req` or `yes`, the request will be rejected.

- Perform cr1 key administration. If you are using the cr1 authentication scheme, you must perform cr1 key administration. See Chapter 5, "cr1 Bilateral Authentication Scheme."

(3b) For Connections Not Using the CS Dialer Type - the Server Machine

Optional or Conditional Steps

- Register **`uucico`** services. A system that expects to be a server in BNU transactions must register a **`uucico`** service for each port monitor that receives BNU service requests. Each **`uucico`** service registered should include an authentication scheme name in the *scheme* field of the port monitor administrative file for the port.

  The **`uucico`** service is added using the **`pmadm`** command. **`pmadm`** is described in detail in the chapter "Managing Ports" in the *System Administration* guide and on **`pmadm(1M).`** The following **`pmadm`** command adds a line for **`uucico`** to the port monitor's **`_pmtab`** file. In this case, the port monitor is `tcp`.

```
pmadm -a -p tcp -s uucico -m "`nlsadmin -c \"/usr/lib/uucp/uucico -r 0 \
-A Addr\"`" -v `nlsadmin -V`-S "cr1 -suucico" -y"uucp"
```

  In the example above, replace *Addr* with the address on which the **`uucico`** service will be offered.

NOTE

When using cr1 on serial connections, you should configure the port to start on carrier (that is, **-c**). This is necessary because the responder sends no data until the imposer indicates it is ready to start the protocol. For this reason, autobaud will not work on parts protected by cr1.

BNU Maintenance

Maintenance involves modifying BNU database support files when necessary, tracking transactions, and cleaning up BNU administrative and log files. Maintenance may be automatic or manual.

The following four shell scripts are delivered as part of the BNU software:

```
uudemon.poll
uudemon.hour
uudemon.admin
uudemon.clean
```

These scripts poll remote machines, reschedule transmissions, and clean up old log files and unsuccessful transmissions. They should be run regularly. Normally, they are run automatically by **cron** [see **cron(1M)]** although they can also be run manually.

Automated Networking Maintenance (cron)

BNU is delivered with entries for shell scripts in the **/var/spool/cron/crontabs/uucp** file. These entries will automatically handle some BNU administrative tasks for you. The shell scripts are in **/usr/lib/uucp**.

The **uucp** jobs that are started by **cron** should be placed in the file **/var/spool/cron/crontabs/uucp**. On a system on which the Enhanced Security Utilities are installed and running, the **tfadmin** database must be updated accordingly so user **uucp** has the appropriate privileges.

In multi-user state, tasks scheduled by the **crontab** command are automatically performed by cron. The **crontab** file for **uucp** is now used to schedule regular BNU maintenance using the following shell scripts:

uudemon.poll

The **uudemon.poll** shell script delivered with the system does the following:

- Reads the **Poll** file (**/etc/uucp/Poll**).

- If any of the machines in the **`Poll`** file is scheduled to be polled, places a work file (**`C.sysnxxxx`**) in the **`/var/spool/uucp/`***nodename* directory (*nodename* is the name of the machine being polled).

By default, the shell script is scheduled to run twice an hour, just before **`uudemon.hour`**, so the work files will be there when **`uudemon.hour`** is called. The following is the default **`uucp crontab`** entry for **`uudemon.poll`**:

```
1,30 * * * * $TFADMIN /usr/lib/uucp/uudemon.poll \
> /dev/null
```

uudemon.hour

The **`uudemon.hour`** shell script delivered with the system does the following:

- Calls the **`uusched`** program to search the spool directories for work files (**`C.files`**) that have not been processed and schedules these files for transfer to a remote machine.
- Calls the **`uuxqt`** daemon to search the spool directories for execute files (**`X.files`**) that have been transferred to your computer and were not processed at the time they were transferred.

The following is the default **`uucp crontab`** entry for **`uudemon.hour`**:

```
41,11 * * * * $TFADMIN /usr/lib/uucp/uudemon.hour\
> /dev/null
```

On the system as delivered, **`uudemon.hour`** is run twice an hour. You may want it to run more often if you expect high failure rates.

uudemon.admin

The **`uudemon.admin`** shell script delivered with the system does the following:

- Runs the **`uustat`** command with the **`-p`** and **`-q`** options. The **`-q`** option reports on the status of work files (**`C.files`**), data files (**`D.files`**), and execute files (**`X.files`**) that are queued. The **`-p`** option prints process information for networking processes listed in the lock files (**`/var/spool/locks`**).
- Sends the resulting status information to the **`uucp`** administrative login via **`mail`** [see **`mail(1)`**].

There is no default **`crontab`** entry for **`uudemon.admin`**. To set up administrative reporting it is recommended that the following line be entered in the **`uucp crontab`** file.

```
48 8,12,16 * * * $TFADMIN /usr/lib/uucp/uudemon.admin\
> /dev/null
```

uudemon.clean

The delivered **`uudemon.clean`** shell script does the following:

- Takes log files for individual machines from the **`/var/spool/uucp/.Log`** directory, merges them, and places them in the **`/var/spool/uucp/.Old`** directory with other old log information. If log files become large, the `ulimit` parameter may need to be increased.
- Removes work files (**`C.files`**) seven days old or older, data files (**`D.files`**) seven days old or older, and execute files (**`X.files`**) two days old or older from the spool directory.
- Returns mail that cannot be delivered to the sender.
- Mails a summary of the status information gathered during the current day to the **`uucp`** administrative login.

The following is the default **`uucp crontab`** entry for **`uudemon.clean:`**

```
45 23 * * * $TFADMIN /usr/lib/uucp/uudemon.clean\
> /dev/null 2>&1
```

`uudemon.clean` is described in detail in “Cleaning Up Log Files” on page 7-65.

NOTE

If your system handles a heavy **`uucp`** load, it may be necessary to run **`uudemon.clean`** with an increased `ulimit`. For example:

```
45 23 * * * ulimit 5000; $TFADMIN /usr/bin/su uucp -c \
"/usr/lib/uucp/uudemon.clean" > /dev/null 2>&1
```

Manual Maintenance

Some files may grow indirectly from **`uucp`** and other Basic Networking activities. The following file should be checked and deleted if it has become too large:

`/usr/lib/cron/log`	This file is a log of **`cron`** activities. While it grows with use, it is automatically truncated when the system goes to the multi-user state.

Cleaning Up Log Files

`uudemon.clean` is a shell script that should be invoked daily by **`cron`** to clean up **`uucp`**'s spool directory and to consolidate and dispose of the log files that currently exist for **`uucp`**. Currently, **`uudemon.clean`** uses two techniques to clean up the log files:

1. The *Multi-day* (multi) technique. Three days of logs are kept in files called **`Old-1, Old-2,`** and **`Old-3`**. Whenever **`uudemon.clean`** is run,

Old-2 is renamed **Old-3**, **Old-1** is renamed **Old-2**, and the current log is renamed **Old-1** and stored in **/var/spool/uucp/.Old/Old-1**.

2. The *Single-day* (single) technique. The current log is moved to **/var/spool/uucp/.Old**. This means that the log is preserved for one day only (assuming that **uudemon.clean** is run daily).

Although **uudemon.clean** does take care of removing old log entries, as an administrator, you should monitor the size of the log files. The exact procedure **uudemon.clean** uses to remove old log entries varies according to the type of log file. Here, shown in Table 7-1, is a description of how the **uucp** log files will be cleaned up.

Table 7-1. Summary of BNU Log Files

File Use	File Name (beginning with **/var/spool/uucp**)	Cleanup Technique
Command	.../.Admin/command	single
History	.../.Log/[uucp|uucico|uux|uuxqt]/system	multi
Foreign	.../.Admin/Foreign	single
Error	**.../.Admin/errors**	single
Transfer	**.../.Admin/xferstats**	single
Accounting	**.../.Admin/account**	multi
Security	**.../.Admin/security**	multi
Performance	**.../.Admin/perflog**	single

BNU Debugging

The *Debug* procedures are intended to help identify and correct common problems in Basic Networking operations and administration. The following monitoring tools are available for detecting and solving Basic Networking problems:

uustat(1C)

cu(1C)

Uutry(1M) **See uucp(1C)**

uuname [see **uucp(1C)] [**See **uucp(1C)]**

uulog [see **uucp(1C)]**

uucheck(1M)

Check Basic Information

There are several commands you can use to check for Basic Networking information.

`uuname`	Lists the machines your machine can contact.
`uulog`	Displays the contents of the log directories for particular hosts.
`uucheck -v`	Run to check for the presence of files and directories needed by **`uucp`**. **`uucheck`** also checks the **`Permissions`** file and provides information on the permissions you have set up.

Check for Faulty ACU/Modem

You can check if the automatic call units or modems are not working properly in several ways.

- Run **`uustat -q`**. This will give counts and reasons for contact failure.
- Run **`cu -d -l`***line*. This will let you call over a particular communications line and print debugging information on the attempt. If the communications line, *line*, is connected to an autodialer, you must add a telephone number at the end of the command line you execute. Otherwise, *line* must be defined as `direct` in the **`Devices`** file.

NOTE

`cs` must be in debug mode. See Chapter 4, “Administering the Connection Server” for more information.

Check Systems File

Check that you have up-to-date information in your systems file if you are having trouble contacting a particular machine. Some things that may be out of date for a machine are its:

- phone number
- login
- password

Debug Transmissions

If you cannot contact a machine, check communications to that machine with **`Uutry`** and **`uucp`**. For a listing of the options accepted by **`Uutry`** and **`uucp`**, see **`Uutry(1M)`** and **`uucp(1C),`** respectively.

Step 1: Simply to try to make contact, run:

`/usr/lib/uucp/Uutry -r` *machine*

where *machine* is replaced with the node name of the machine you are having problems contacting. This command will:

1. Start the transfer daemon (**`uucico`**) with debugging. You will get more debugging information if you are `root`.
2. Direct the debugging output to **`/tmp/`***machine*,
3. Print the debugging output to your terminal, using **`tail -f`**. When the conversation has completed, the **`tail -f`** process is killed.

You can copy the output from **`/tmp/`***machine* if you want to save it.

Step 2: If **`Uutry`** doesn't isolate the problem, try to queue a job by running:

`uucp -r` *file machine***`!`***/dir/file*

where *file* is replaced by the file you want to transfer, *machine* is replaced by the machine you want to copy to, and *dir/file* is where the file will be placed on the other machine. The **`-r`** option will queue a job but not start the transfer.

Now use **`Uutry`** again. If you still cannot solve the problem, you may need to call support personnel. Save the debugging output; it will help diagnose the problem.

NOTE

Since the connection phase is now performed by the Connection Server, information previously found in **`/tmp/machine`** is now maintained in the Connection Server debug log (if it is enabled). For more information, see Chapter 4, “Administering the Connection Server.”

8
Administering REXEC

8
Administering REXEC

Introduction to REXEC Administration

Many commercially available transport protocols provide remote execution capabilities that are tied to the protocols. In the OS, the **`rlogin`** command, for example, allows remote users to log into a host on a TCP/IP network. REXEC is a remote execution facility that is independent of transport protocol. It allows a user to execute a process on a remote host, independent of the transport provider.

REXEC increases productivity across a set of machines and makes it possible to share expensive hardware on a network. REXEC saves space on a client machine because a copy of an executable file needs to exist only on the server. Hardware can be better utilized by allowing client users to run processes remotely that require the hardware.

REXEC also makes it possible for users to access a centralized database—without significantly increasing network traffic. If a local user runs a database query on a remote machine, the remote machine sends only the response to the query across the network. From the user perspective, remote execution of a process through REXEC is transparent. Once the user executes a process on a remote machine, the user interfaces with the process as if it were running locally. However, all files referenced by a remote command are relative to the server. If a user executes the **`who`** command remotely, for example, a list of users logged in to the remote machine is displayed, not a list of users on the local machine.

For the system administrator, REXEC makes it possible to administer a set of machines remotely from a single location, without the overhead of logging in to the remote machines. REXEC gives administrators the ability

- to monitor the system activity of several machines from a single machine
- to stop and start network applications, such as NFS
- to tune the performance of a set of machines by specifying that certain processes are to execute on a less active system.

REXEC is service-based. A server defines services that a client can execute remotely. By default, when REXEC is first installed, a server has the following standard services defined:

`rx`	A service that allows a user on a client to execute a command or shell script on the server.
`rl`	A service that allows a user on a client to log in to the server.
`rquery`	A service that allows a client user to list the services available on a server for remote execution.

General instructions for invoking an REXEC service, as well as specific instructions for using the standard services, appear in *User's Guide* and on **`rexec(1)`**.

Overview of REXEC Administration

Although REXEC is independent of other network applications, it utilizes components of the Network Applications Architecture, specifically ID Mapping and the Service Access Facility on the server side, the Connection Server on the client side, and an IAF authentication scheme on both the server and client. The client also utilizes Name-to-Address Mapping.

This section first describes the components of the Network Applications Architecture that need to be administered before you set up REXEC. It then gives an overview of the steps that are specifically part of REXEC administration.

Prior to setting up REXEC on the server, you must

1. Install an authentication scheme.
2. Set up ID Mapping.
3. Set up the Connection Server.

The authentication scheme, cr1, will be used to protect the REXEC service from unauthorized remote access. Setting up the authentication scheme will include the use of the cr1 administrative commands to control the key management daemon, and to administer the key database.

Using the ID Mapping facility, map the logins of users on a client to logins on the server. Before client logins can be mapped, the server logins to which client logins will be mapped must exist on the server system with valid entries in the server's **`/etc/passwd`** file. When this is done, use the ID Mapping administrative commands with a cr1 mapping scheme specification to complete the set up of the facility.

NOTE

Currently, cr1 is the only authentication provided that uses ID Mapping. cr1 administration involves setting up and maintaining the cr1 key database, as described in Chapter 5, “cr1 Bilateral Authentication Scheme.”

Setting up the Connection Server on the server will include the installation of the **`reportscheme`** service for each port monitor being used to offer the **`rexec`** service.

Before you use REXEC from a client, you must do the following:

1. Set up a host address database.
2. Install the same authentication scheme used on the server to protect REXEC.

3. Set up the Connection Server.

How you set up a host address database depends on your network. Host addresses are stored in any of several databases, depending on the type of network connection the client has to the server. If a client can reach a server over a TCP/IP transport, for example, then that client should have the server's name and address in its **`/etc/hosts`** file. See Chapter 3, "Administering Name-to-Address Mapping" for information about setting up a host address database for your network.

Setting up the authentication scheme on the client will include the use of the cr1 administration command **`cryptkey`** to specify the key to be shared between the server and the client.

Setting up the Connection Server on the client will include the specification of the authentication scheme to be used. For example, cr1 needs to be specified in **`/etc/iaf/serve.allow`**. This will help ensure enforcement of bilateral authentication.

Instructions for setting up and administering Name-to-Address Mapping, the cr1 Bilateral Authentication Scheme, ID Mapping, and the Connection Server appear in preceding sections of this chapter, and should be consulted for more details.

Once you've set up the network services on which REXEC depends, you're ready to set up REXEC itself. Administering the REXEC facility on a server involves the following tasks:

1. Registering REXEC with a port monitor under the Service Access Facility.
2. Maintaining a database of services available for remote execution through REXEC.

Administering REXEC on a client is just a matter of installing the REXEC software. However, the client administrator may choose to create links from REXEC services to the REXEC command interface to improve the ease with which local users can execute remote services.

The following sections describe REXEC administrative tasks.

Registering REXEC with a Port Monitor

As with any service, you must register REXEC with a port monitor under the Service Access Facility (SAF). By registering the service with a port monitor, you associate the service with a specific port and an authentication scheme. When a client machine attempts to access the service, it calls the port monitor, which informs the client that the service is protected by the scheme. If the client supports the scheme, it begins the authentication process.

The authentication scheme authenticates the user on the client machine and maps the user's login to a local login before REXEC is invoked. REXEC obtains the information established by the authentication scheme and sets up the environment of the mapped user before executing the requested service.

NOTE

As explained earlier, cr1 is the only authentication scheme provided that uses ID Mapping. It is assumed throughout the REXEC documentation that your authentication scheme is cr1.

To register REXEC with a port monitor and protect it with the cr1 authentication scheme, you must add REXEC to the port monitor's **`_pmtab`** file. If the port monitor is `tcp`, the **`rexec`** service is added to the **`_pmtab`** file by using the command:

```
pmadm -a -p tcp -s rexec -f u \
-m "`nlsadmin -c /usr/lib/rexec/rxserver -p tirdwr`"\
-v `nlsadmin -V` -S "cr1 -srexec" -y "remote execution"
```

The port monitor-specific command, the **`-m`** operand, specifies the server `rxserver` to be executed and the module to be pushed on the Stream. `rxserver` is invoked by a network listener process after a connection has been established between a client and the server's port associated with the **`rexec`** service, and performs REXEC server functions.

The **`-S`** operand of the **`pmadm`** command specifies the authentication scheme to be associated with the service tag **`rexec`**.

For more information about registering services under the SAF, see the chapter “Managing Ports” in the *System Administration* guide for more information. For additional information about cr1, see Chapter 3, “cr1 Bilateral Authentication Scheme.”

Adding and Removing Services

The server administrator controls which local services are available for remote execution by adding and removing services from the REXEC database **`/etc/rexec/services.`**

Although REXEC is installed with three standard services defined, these services can be removed from the REXEC database at any time. In addition, the server administrator can add other services to the database and make them available to clients for remote execution.

You don't need to edit **`/etc/rexec/services`** explicitly; instead, the **`rxservice`** command is provided as an interface to the file. **`rxservice`** allows a privileged user to add and remove services from the REXEC database.

Adding a Service

To make a service on your machine available to remote users through REXEC, you add the service to the REXEC database, using the **`rxservice`** command.

The syntax of the **`rxservice`** command, when used to add a service, is as follows:

```
rxservice -a servicename [-d description] [-u] servicedef
```

See **`rxservice(1M)`** for an explanation of the options accepted by the **`rxservice`** command.

The REXEC facility defines the following macros, which can be used by any service. When you specify any of the following macros, REXEC substitutes appropriate values for the macros when a remote user makes a request for the service. The remote user is not required to enter the information as arguments to the invoking command.

`%m`	The address of the client machine.
`%t`	The name of the transport provider used to connect to the server.
`%s`	The mapped user's shell, obtained from the **`/etc/passwd`** file on the server.

If the value of a macro cannot be obtained, the macro is expanded to a dash (`-`).

Examples

Assume you want to define the **`shutdown`** command so that it can be used remotely. The syntax of **`shutdown`** is as follows:

```
shutdown [-y] [-g grace_period] [-i init_state]
```

Let's call the remote shutdown service **`rshutdown.`** If you want **`rshutdown`** to support all the parameters to **`shutdown`** and in the same way (as options), then you would define **`rshutdown`** as follows:

```
rxservice -a rshutdown -d "rshutdown usage: rshutdown \
      shutdown_options" 'cd /;/sbin/shutdown %*'
```

The macro `%*` means that **`rshutdown`** supports all options to **`shutdown.`** Suppose you were to define the **`rshutdown`** service as follows:

```
rxservice -a rshutdown -d "rshutdown usage: rshutdown \
      minutes initstate" 'cd /;/sbin/shutdown -g%1 -i%2'
```

When defined this way, the macros indicate that the **`-g`** and **`-i`** operands (that is, values for *%1* and *%2*, respectively) are required. If the remote user fails to supply a grace period operand or an init state operand, the **`rshutdown`** command will fail. Note that the service definition omits the **`-y`** option. Because the option has been omitted in the service definition, the **`rshutdown`** user cannot by-pass the prompt that asks him or her to confirm that the system should be shut down.

The following command line defines a service called **`rlookup,`** which accesses a local database via a command called **`dblook:`**

```
rxservice -a rlookup -d 'Remote database lookup' \
      '/usr/bin/dblook %*'
```

The following command line defines a service called **`rsetup,`** which modifies database tables via a local command called **`setdb.`** The **`setdb`** command takes the address of the client machine as a parameter.

```
rxservice -a rsetup -d 'remote setup service' \
      '/usr/bin/setdb %m'
```

Removing a Service

To disable a service so that it is no longer available for remote execution, you remove the service from the REXEC database. In addition to providing a means to add services to the database, the **`rxservice`** command allows you to remove services.

The syntax of the **`rxservice`** command when used to remove a service is

```
rxservice -r servicename . . .
```

where *servicename* can be one or more service names.

Example

To remove the services **`rshutdown,`** **`rsetup,`** and **`rlookup`** from the REXEC database, enter the following command line:

```
rxservice -r rshutdown rsetup rlookup
```

NOTE

If you want to remove a service temporarily, you can remove the service's execute permissions.

Listing Defined Services

As a system administrator, you may need to view the contents of **`/etc/rexec/services`** from time to time, to remind yourself which services you have made available for remote execution. To display the **`services`** file, enter the **`rxservice`** command with the **`-l`** option, as follows:

```
rxservice -l
```

In addition to providing an option to **`rxservice`** that allows the administrator to display **`/etc/rexec/services,`** REXEC installs a command on the server that allows a remote user to display the file. When a remote user displays the file, however, only the services available to that user are displayed.

The command on the server that allows remote users to display the **`services`** file is the **`rxlist`** command. Remote users use the standard REXEC service **`rquery`** to run **`rxlist`** remotely.

If the **`rxlist`** command is removed from the server, client users cannot determine which services are available to them for remote execution; however, they can still invoke those services, provided they already know the names of the services and have appropriate permissions.

Instructions for using **`rquery`** appear in *User's Guide* and on **`rexec(1)`**.

Linking Services to REXEC

Administration of REXEC on a client machine is simply a matter of installing the REXEC software.

You may choose, however, to create links from the **`rexec`** command to the REXEC services defined on a remote machine. The **`rexec`** command is the user interface to the REXEC facility. When you link services to **`rexec,`** your users can enter abbreviated commands to invoke a remote service.

The standard REXEC services are linked to the **`rexec`** command by default when the software is installed. If a user wants to run **`rquery`** on a server named `sftig,` for example, the user enters

```
rquery sftig
```

With the link removed, the user enters

```
rexec sftig rquery
```

For information about creating and removing hard or symbolic links, see **`ln(1)`**.

2
Mail Service Administration

Replace with Part 2 tab

9 Administering the Mail Service

9
Administering the Mail Service

Introduction to Mail Service Administration

NOTE

If your system is being run in compliance with the security criteria described in the chapter “Trusted Facility Management” in the guide *System Administration*, the **`mail`** software will be used only to provide electronic communication between users on the same computer; none of the networking facilities described in this chapter should be made available on such systems.

The mail service provides electronic communications between users on the same computer, or between computers connected together on a network, and supports two addressing schemes, known as “bang” style and “domain” style. You do not need to do anything for mail to work in the default manner.

You can take advantage of various options within the mail service that enable you to set up a smarter host, establish a domain name, administer a set of sites so they all send mail with the same system name, set up the mail directory to be shared across a networked file system, such as NFS, fill in alias information, and set up a connection to another site that uses the Simple Mail Transfer Protocol (SMTP). This chapter provides information on these options.

This chapter covers **`mail(1)`** and **`smtp(1M)`**. **`Sendmail(1M)`** provides an alternate inter-network mail transport mechanism. **`Sendmail(1M)`** is covered in the next chapter

Multilevel Mail Files

As an administrator, you should be aware that the mail directory, **`/var/mail`**, is a Multilevel Directory (MLD). This should be transparent to a user as long as the directory remains an MLD. (See “Multilevel Directories” in the chapter “Administering File Security Attributes” in the *System Administration* guide for further information.) If there are problems, make sure **`/var/mail`** is an MLD of mode `775`, is owned by `root`, and belongs to group **`mail`**. While in real mode, also check the subdirectories for the same permissions and ownership.

Mail Administration Files

There are four files that are important to mail administration.

- The surrogate file, **/etc/mail/mailsurr**, describes how to rewrite addresses and how to deliver messages through the networks. [See **mailsurr(4)**.]
- The configuration file, **/etc/mail/mailcnfg**, which is optional, permits various per-site options to be established. This file is not delivered with the Mail service: you must create it yourself if you want to define your own options. [See **mailcnfg(4)**.]
- The master alias path file, **/etc/mail/namefiles**, by default points to one alias file, **/etc/mail/names**, and one alias directory, **/etc/mail/lists**. Further (directories and files may be added to the list.) These files and directories are used to define name mappings and address lists. [Both files are described on **mailalias(1)**.]

Note that all files under the directory **/etc/mail** are installed at the level SYS_PUBLIC. If your mail system has problems, check the levels of the files there (using the **ls -z** command) and make sure they are at SYS_PUBLIC. You can change the levels of the files using:

chlvl SYS_PUBLIC *file*

There are additional administration files that need to be considered if **sendmail(1M)** is used for delivery of remote mail. Refer to the chapter on “Sendmail”.

Editing Mail Administration Files in a Secure Environment

Because the mail administration files must be visible at all security levels, they are installed at the level SYS_PUBLIC. They can be edited only when the system is in maintenance mode, and should be edited using the trusted editor, **ed(1)**. After editing, the level must be restored on the file using:

chlvl SYS_PUBLIC *file*

for the mail service to work correctly when the system is brought back to multi-user mode.

For more information, see “Maintenance Mode and the Trusted Systems Programmer” and “Editing System Files” in the chapter “Trusted Facility Management” in the guide *System Administration.*

Note also that the **mail** and **mailalias** commands pre-compile the surrogate, configuration, and alias files. Therefore, after editing the surrogate or configuration file (remember to do this in maintenance mode), send a test message using the **mail** command; after editing one of the alias files (again, while in maintenance mode), use the **mailalias** command to access one of the aliases within that alias file. Following this, check the level of the files under **/etc/mail** to make sure they are at the SYS_PUBLIC level. If necessary, set the levels using the **chlvl** command:

```
chlvl SYS_PUBLIC /etc/mail/*
```

This guarantees the compiled surrogate and alias files are maintained at the proper levels.

Mail Addressing Styles

The default surrogate file contains command entries for translating between domain style addresses and bang style addresses. Bang style addressing is characterized by exclamation points (or "bangs") within the address and looks like *host*!*user* or *host1*!*host2*!*user*. Domain style addressing is characterized by the commercial at sign (@) and looks like *user@host.domain* or *user@host*.

Establishing a Smarter Host

Although it is possible to maintain the data files for the UUCP network so the system knows about hundreds or thousands of other systems that can be contacted, it is impractical to do so. It is often much easier to set up what is known as a "smarter host," that is, a reference to a UNIX system where remote mail will be shipped if the mail destination is not known to the local computer. For example, assume you need to send a mail message to `hosta!tony`, but your local computer does not know about `hosta`. The mail message can be automatically routed to the computer `worldly`, which has a more extensive list of UUCP connections, if `worldly` is set up as a smarter host.

This is done in two steps:

1. Add a line to `mailcnfg` that says

   SMARTERHOST=*smhost*

   where *smhost* is replaced with the name of the smarter system (`worldly`).

2. Remove the # character from the line within `mailsurr` that looks like this:

   ```
   #'.+'   '!(.+)'     'Translate T=1; R=!%X!\\1'
   ```

If you are also using a cluster setup, and the cluster name is the same as the smarter host's name, there will be problems with the above because mail will subsequently strip off the cluster's name. Such a setup will often happen when many sites share a common **`/etc/mail/mailcnfg`** file. For such a setup, instead of using the smarter host surrogate entry shown above, use the commented out entry which looks like this

```
#'.+'  '!(.+)'  '< sh -c "if [
    then uux -a%R - %X!rmail '(1)';
    else uux -a%R - %x!rmail '(1)';fi"'
```

Now add another entry to **`/etc/mail/mailcnfg`** describing the smarter host which will be used by the cluster's smarter host, using

```
x=alternate-smarter-host
```

At sites where **`uname`** does not match the smarter host name, the mail will be forwarded (using **`uucp`**) to the real smarter host. On the smarter host itself, any addresses which it cannot handle will be sent to the alternate smarter host.

Establishing Domain Addresses

As distributed, mail knows about two forms of domain style addresses:

user@*host* `and` *user*@*host*`.UUCP`

It does not know about

user@*host*`.`*domain*

A domain name is an internationally recognized and registered name for a set of computers. Commercial entities may be registered under domain names similar to .*company-name*.COM and educational entities may be registered under domain names similar to .*school*.EDU.

NOTE

`.UUCP` is not a true domain name. The high-level domain names of `.COM` and `.EDU` are assigned by a central authority.

A system generally knows how to establish direct connections to other computers within the local domain, but you may want to make use of a smarter host to take care of other domains.

To establish the local domain name, type

```
/usr/sbin/domainname domain
```

where *domain* is replaced with the domain name, such as .*company-name*.COM (or whatever is appropriate) and contains a leading period. (Preceding backslashes are added to any periods in the domain name, changing "`.`" to "`\.`", before being passed to the regular expressions in the surrogate file.)

The domain name is also used by the SMTP router when header files are being rewritten into RFC 822 format. (See "Administering SMTP" on page 9-9.)

Establishing a Mail Cluster or Gateway

With the arrival of inexpensive personal computers, it is often desirable to assign a single name to a set (or a cluster) of computers by which all the computers in the cluster are known to external computers, for purposes of mail delivery. For example, a cluster of computers known internally under names such as `Xsysa`, `Xsysb`, and `Xsysc` could be assigned the cluster name of `Xsys`. Mail sent from any of these computers would be shown as being from `Xsys`; that is, the internal names would not be known outside the cluster.

To establish a cluster name, add a line such as the following to `mailcnfg`:

```
CLUSTER=extname
```

where *extname* is the name by which the computer is known externally (such as `Xsys`). In the surrogate file, **`/etc/mail/mailsurr`**, you can then use `%L` to refer to the value of `CLUSTER`.

Establishing Mail Service on a Networked File System

With the arrival of inexpensive Local Area Networking (LAN) and networked file systems (such as NFS), it is often desirable to set up clusters of computers that share file systems.

It is also possible to share **`/var/mail`** across the computers.

In you choose to establish a mail cluster with a shared **`/var/mail`** you can arrange to have all users' mailboxes created on only one computer, but accessible from all computers.

As an example, assume you want the systems `Xsysa`, `Xsysb`, and `Xsysc` to share a single mail directory under `Xsysa`. In addition, the entire file system for each system is mounted under the names **`/Xsysa`**, **`/Xsysb`**, and **`/Xsysc`**. All users have home directories under file systems named **`/homea`**, **`/homeb`**, and **`/homec`**, which are mounted on the corresponding computers.

To establish a shared **`/var/mail`** file system, complete the following steps:

1. Make sure **`/var/mail`** from `Xsysa` is advertised.
2. Remove the directory **`/var/mail/:saved`** from systems that will not have a local **`/var/mail`** (`Xsysb` and `Xsysc`).
3. Add a line to `mailcnfg` that says

   ```
   FAILSAFE=Xsysa
   ```

   With this specified, **`mail`** looks for the presence of **`/var/mail/:saved`**. If the directory is not there (an indication that the network connection to `Xsysa` has been lost), **`mail`** requeues the file to be delivered to `Xsysa` via other means (such as by UUCP or SMTP).
4. Move any mailboxes from **`/var/mail`** on `Xsysb` and `Xsysc` to `Xsysa`. (If you don't, the files will be inaccessible.)
5. Mount **`/var/mail`** from `Xsysa`.
6. To allow the **`notify`** program to identify where the user is logged on (so it can notify the user when new mail arrives), create, on all systems, a file named **`/etc/mail/notify.sys`** with contents similar to the following:

```
    Xsysa   /Xsysa
    Xsysb   /Xsysb
    Xsysc   /Xsysc
```

The first column lists the name of the system; the second gives the path of the `root` file system for each computer.

7. To allow the **`notify`** program to handle a network failure, create, on all computers, a file named **`/etc/mail/notify.fsys`** with contents similar to the following:

```
    /homea Xsysa
    /homeb Xsysb
    /homec Xsysc
```

   The first column lists a file system name; the second contains the name of the system (computer) on which that file system is normally mounted. If **`notify`** cannot open the mail file for writing, it looks up the file system in this list and requeues the file to be delivered to the corresponding system via other means (such as by UUCP or SMTP).

Administering alias Lists

Before delivering a local mail message, **`mail`** looks up the user name to see if the user name has been assigned an alias. An alias may reference simply another name or a list of names.

The master alias path file (**`/etc/mail/namefiles`**) contains a list of files that **`mail`** searches for aliases. As distributed, this list contains two files, **`/etc/mail/names`** and **`/etc/mail/lists`**, to be searched for aliases.

If the named alias is found at the beginning of a line within an alias file, the rest of that line is used as the alias. This may contain a single name or a list of names separated by blank spaces. For example, if you want to set up a group mailing list (such as **`andy.group`**) that is to be expanded, add a line similar to the following to the alias file:

```
    andy.group tony paul john ned gary hailey mike
```

Recursive references are permitted, as in this reference to **`andy.group`** within another alias:

```
    armida.dept andy.group danielle.group bob.group \
        lee.group pier.group
```

Several alias files can be listed in **`namefiles`**, which may be kept anywhere on the computer. This permits different alias files to be owned by different administrators.

Note the **`sendmail(1M)`** does not use the **`mail(1)`** alias listed above. Generally, if **`sendmail(1M)`** is used for delivery of remote mail, then the **`mail(1)`** alias file(s) can be disregarded. Refer to the chapter on “Sendmail”.

NIS and Mail Aliases

NIS support is available for the aliases file **`/etc/mail/names`** used by **`mail(1)`**. The advantage of using NIS is that it eliminates the need for maintaining an alias file on each system in a computer network. Instead it is the only necessary to maintain a single copy of the mail aliases file - on the NIS master system.

By using NIS, client systems would only need a small number of host specific aliases. Other aliases would be common throughout the network and can be in the NIS-managed set (which is the set of aliases from the file **`/etc/mail/names`** on the NIS master system). The full set of aliases for a client would be the union of the host specific aliases and the NIS-managed set of aliases.

NIS is enabled on a client, by adding a line beginning with the NIS special token '+' to the mail address file (**`/etc/mail/names`**). The token indicates that the NIS-managed set of mail aliases should be "included" at this point in the aliases file.

Note that this is the only NIS special token supported for mail aliases (and it is only possible to include the entire NIS set of mail aliases).

Surrogate Files

Logging Mail

Occasionally it may be necessary to keep a log of traffic going through the system. For example, if you were to write a program called **`/usr/lib/mail/surrcmd/logmail`** that takes three arguments (the names of a log file, a sender, and a recipient), it could log all external mail flowing through the system by using this surrogate entry:

```
'.+!.+' '.*' '> logmail W=1;B=*; /var/adm/mailtransport\
%R' '%n'
```

Because the path used for surrogate commands includes **`/usr/lib/mail/surrcmd`**, there is no need to use a full path to reference `logmail`. Another example would be to log traffic to or from a particular system. In the following example, we're logging traffic from our system to system `xyz` and traffic from `abc` to our system:

```
'.*' 'xyz!.+' '> W=1;B=*; logmail /var/adm/mailto-xyz\
    %R' '%n'
'abc!.+' '.*' '> W=1;B=*; logmail /var/adm/mailfrom-abc\
    %R' '%n'
```

The W=1; indicates that **`mail`** should not wait for the logging program to complete before continuing its work. The B=*; indicates that **`mail`** should execute a single call to `logmail`, passing it a list of all recipients to whom mail was delivered.

Using pathrouter for Smart Routing

On many systems, you can find a path translation program that gives the shortest route to a given system, based on various criteria or a database. An example of this is the **pathrouter** program. For example, if you wish to setup a machine to act as a mail gateway between the UUCP zone and the Internet zone, then the **pathrouter** program can be used to facilitate this. The **pathrouter** program is used to lookup the route information for UUCP and domain addresses. As an alternative to using a smarter host, the **pathrouter** program can be invoked as a final step in the **mailsurr** file:

```
'.+'  '.*[!@].*'  'Translate T=1;B=*; R=|pathrouter -p' '%n'
```

If you know that the majority of the addresses used on the system will have to be translated, this statement can also be placed just prior to the invocation lines for the **uux** program.

The **pathrouter** program in turn uses the file **/etc/uucp/paths** to decide how to rewrite addresses. For example, the following **paths** file is an sample of the setup for a gateway machine. This example prefixes the destination routes with an exclamation mark (`!`) to reduce the number of translations done within the **mailsurr** file.

The following example is used to rewrite the domain addresses **defgmbh.ge** and any domain address based in the **uk** to be routed via the `abcinc` machine. The `xyzinc.com` domain address is converted to `!xyzinc!%s` in order to utilize the UUCP transport rather than SMTP.

```
    abcinc       !abcinc!%s 0
    defgmbh.ge   !abcinc!defgmbh.ge!%s 0
    windsor      !abcinc!windsor!%s 0
    xyzinc       !xyzinc!%s 0
```

Note that with this example, you would not have a smarter host defined, and so any addresses that are not matched by **pathrouter** are left unchanged. The SMTP DNS lookup will resolve the address and perform delivery over SMTP, and mail to any unknown address would be returned.

Controlling Mail Resource Access

It is often necessary to control access to commercial services. One method of doing this is to prevent any non-local users from sending mail to the commercial site using the **Accept** and **Deny** commands:

```
    '[^!]+' 'commail!.+' 'Accept'
    '.+'    'commail!.+' 'Deny'
```

Another method is to use an external program to check the sender's path to see if it is a valid user of the service. For example, the following shell script returns `0` if the sender is a valid system, and `1` otherwise:

```
case "$1" in
    abc | def | ghi ) exit 0 ;;
    * ) echo "$1 is not permitted to send mail to external service"
        exit 1 ;;
esac
```

If the script were installed as **`/usr/lib/mail/surrcmd/chksender`**, it would be invoked as a delivery agent that either continues or fails:

```
# check senders more than one hop away
'.+!(.+)![^!]+' 'commail!.+' '< C=0;F=*; chksender \\1'
# check senders one hop away
'(.+)![^!]+'    'commail!.+' '< C=0;F=*; chksender \\1'
```

Administering SMTP

The Simple Mail Transfer Protocol (SMTP) mail service is delivered as a group of programs that allows UNIX system mail to send and receive mail using the SMTP protocol. This protocol is typically used over TCP/IP networks. However, as delivered, the SMTP processes can connect to a remote SMTP server over any TLI or XTI-based, connection-oriented transport medium that has been administered to have an SMTP service. However, because this version of the system depends on the standard system port monitors for inbound SMTP, inbound connections can be made only over the TCP/IP network.

To establish SMTP service requires these steps:

1. The Internet Utilities package from the Networking Set must be installed. This establishes a call to the SMTP queuer program in the mail surrogate file **`/etc/mail/mailsurr`**:

   ```
   '.+' '!([^!]+)!(.+)' '< B=4096; smtpqer -N %R \\1' '\\2'
   ```

   Mail is addressed using the standard UNIX system mail formats of *host*!*user* or *user*@*host*. If *host* is known to support SMTP mail delivery, the mail is queued for delivery using SMTP. If not, **`smtpqer`** does not accept the message, and delivery is done by subsequent surrogates in the **`mailsurr`** file.

   All messages spooled for SMTP delivery are stored in the directory **`/var/spool/smtpq/`***host*, where *host* is the name of the computer to which mail is being sent.

2. The list of computers that will accept SMTP mail is determined by querying `netdir_getbyname()`. See **`netdir_getbyname(3N)`** and **`netconfig(4)`** for further information.

3. By default the SMTP daemon program **`in.smtpd`** only runs when there are incoming SMTP service requests. This is invoked by the intd internet port monitor. An entry for the SMTP daemon program will be in the inetd configuration file **`/etc/inet/inetd.config`**.

   Alternately, **`in.smtpd`** can be disabled and the SMTP daemon **`/usr/lib/mail/surrcmd/smtpd`** can be started when the system is booted.This version of the daemon is always running and it will wait for incoming SMTP requests when there is nothing to process.

   When **`smtpd`** receives a piece of mail, it does three things:

   - It inserts a valid UNIX system `From` header line;
   - It converts the recipient address to *host*!*user* form; and

   It hands the message to **`rmail`** for delivery.

   Note that by default, in.smtpd is invoked with the “r” option. This option is used to prevent smtpd from rewriting the addresses in the “From:”, “To:”’ “Cc:” and “Bcc:” from Domain style addressing to UUCP style. If this conversion is desired, then the “-r” option should be removed.

4. The **`crontab`** file for the pseudo-user `smtp` has the following contents:

```
25 * * * * /usr/lib/mail/surrcmd/smtploop -B
55 1 * * * /usr/lib/mail/surrcmd/smtploop -c -w 1 -r 7
```

   By default, mail that cannot be delivered immediately (as it is sent) is queued and **`smtploop`** repeatedly tries to deliver it at one hour intervals. You can change the interval between delivery attempts by modifying the entry for **`smtpsched`** in **`smtp`**'s **`crontab`** file. The **`smtp`** **`crontabl`** file is in **`/usr/spool/cron/crontabs/smtp.`**

SMTP logs all SMTP activity, including incoming mail messages, in the log file **`/var/spool/smtpq/LOG`**. It is backed up once a day by **`smtpsched`**; the log files for previous days are located in **`/var/spool/smtpq/LOG.`***n*, where *n* specifies the day of the week (from `0` to `6`). The **`smtpsched`** program also returns undeliverable mail messages. [See **`smtpsched(1M)`** in the *Command Reference*.]

The **`-N`** option is used by both the **`smtpqer`** and **`smtp`** programs. This option prevents these programs from attempting to query a nameserver for MX records and internet addresses. If you are using a nameserver, remove the **`-N`** option.

Setting Up SMTP to Listen Over Multiple Networks

`smtpd` will listen to any connection-oriented TLI or XTI network that provides the SMTP service. TLI/XTI networks are specified in **`/etc/netconfig`**. For each network that is connection-oriented, **`smtpd`** will use **`netdir_getbyname(3)`** to determine if the SMTP service exists for that network. If the service does exist, a port is opened at the address returned by this function. To make the listener listen to a new network, first administer the `netdir` databases, and then restart the listener.

The mailR Language and Header Rewriting

As you've seen from working with the mail surrogate file, the surrogate rules control how mail addresses are interpreted and how the mail message is to be delivered to its recipients. However, nothing that you've done so far in the surrogate file has affected the contents of the mail message. That is the purpose of the **`/etc/mail/rewrite`** file and the **`mailR`** language. As a small example of what can be done in the **`rewrite`** file, consider this small function:

```
function main()
{
    delete_headers("Foo-Bar");
    if (!exists("Date"))
        preend_header("Date", "To", rfc822date());
}
```

This function says to delete all `Foo-Bar` headers in the message, including those that are spelled `FOO-BAR`, `foo-bar`, `FoO-bAr`, or any other similar header. It then checks to see if the message has a `Date` header and, if not, creates one. The new header will be placed in front of any existing `To` headers and will have the value returned by the function `rfc822date()`. If no `To` header exists, the new header will be placed at the beginning of the headers.

The **`mailR`** language is a full featured language, with a variety of control structures and variables. It uses a mail message as its input, generating another mail message as its output. It is similar in nature to **`awk(1)`** in that variables are automatically converted from strings to doubles as needed, and arrays are indexed by any arbitrary string (associative arrays). However, unlike **`awk`**:

- Variables must be declared before being used.
- There are no global variables.
- All statements must end with a semi-colon.
- String concatenation is performed using the `@` operator.
- Function arguments can be passed by value or by reference.
- `for` statements can loop over the values of arrays as well as the indices of arrays.
- Special functions known as generators are available.

Now let's look at a more extensive example:

```
# call rewrite to begin header modification
function main()
{
    rewrite("To");   # modify To: headers
    rewrite("Cc");   # modify Cc: headers
    rewrite("Bcc");  # modify Bcc: headers
    rewrite("From"); # modify From: header
}

# rewrite address headers by guaranteeing that a
# domain exists and by deleting sub-domain information
function rewrite(var hdr_name)
{
    var old_Hdr;

    # loop through all headers of the given name
    for (old_Hdr from headers(hdr_name))
         {
         var hdrval, new_Hdr, address_count = 0;

         # loop through all addresses in the header
         for (hdrval from addrparse(old_Hdr))
             {
             var old_domain = hdrval["domain_part"], new_domain;

             # if no domain exists, tack on our domain
             if (!old_domain)
                  {
                  old_domain = domain();

                  # convert !-style addresses to domain-style addresses
                  if (hdrval["local_part"] ~ "!")
                      {
                      var new_address =
                      bangtodomain(hdrval["address"] @ old_domain, "@r!l%r");
                      var new_hdrval = addrparse(new_address)[0];
                      hdrval["route"] = new_hdrval["route"];
                      hdrval["local_part"] = new_hdrval["local_part"];
                      old_domain = hdrval["domain_part"] =
                           new_hdrval["domain_part"];
                      }
                  }

             # does the name say xyz.foo.com?
             if (old_domain ~ ".+\\\.foo\.com")
                  new_domain = "foo.com";

             # does the name say xyz.bar.com?
             else if (old_domain ~ ".+\\\.bar\.com")
                  new_domain = "bar.com";

             # use the domain information as it stands
```

```
              else
                  new_domain = old_domain;

              # if any group name information exists, preserve it
              if (hdrval["group_name"])
                  new_Hdr[address_count] = hdrval["group_name"] @ ": " @
                      hdrval["route"] @ hdrval["local_part"] @
                      new_domain @ " " @ hdrval["comment"] @
                      hdrval["errors"] @ "; ";
              else
                  new_Hdr[address_count] =
                      hdrval["route"] @ hdrval["local_part"] @
                      new_domain @ " " @ hdrval["comment"] @
                      hdrval["errors"];
              address_count = address_count + 1;
              }

          # Replace the value of the old header with the new header.
          # Separate addresses with commas and put each address on
          # a line by itself.
          old_Hdr = "";
          if (address_count > 0)
             {
             var j;
            old_Hdr = new_Hdr[0];
            for (j = 1; j < address_count; j = j + 1)
                  old_Hdr = ",\n" @ new_Hdr[j];
            }

          else
              old_Hdr = new_Hdr[0];
          }

}
```

This example begins with the `main()` function, which calls the function `rewrite()` to modify, in turn, the `To`, `Cc`, `Bcc` and `From` headers. The `rewrite()` function starts out by looping through all of the headers of the given type, using a built-in function `headers()`. The `for` loop is executed once for each header, with the variable *old_hdr* set as a reference to each value in turn. The list of addresses from each header is then parsed (into its separate pieces) using the built-in function `addrparse()`. The domain portion of the address is then examined to see if a value is present. If not, the current domain is used as the setting for `domain`.

The address is then checked to see if it contains a `!`-style address. If so, it is converted to `@`-style, using the built-in function `bangtodomain()` and reparsed; the reparsed address is then assigned as the mail address.

The domains are then examined to see if they contain subdomains (of the domains foo.com or bar.com), and if so, the domain is changed to the upper level domain. The address is then recreated, along with any RFC 822 group names, the comments, and any errors which may have been detected during the parsing. The new address is stored into the array `new_Hdr`.

After all of the addresses have been parsed, the header's value is changed by looping over the array `new_Hdr` and putting the addresses together (separated by `commas` and `newlines`). Each continuation line of a header must begin with white space, and the **`mailR`** language oversees this requirement and guarantees that it is done.

Many other built-in **`mailR`** functions are available, each described on the **`mailR(5)`** manual page. It is also possible to pipe the entire message or the body of the message through an external command, instead of only being able to modify the headers of the message.

The **mail** command automatically calls the **mailR** function `main()` before doing any parsing of the mail surrogate file. Before the message is delivered to a local mailbox, the **mailR** function `local()` is invoked. It is also possible to invoke a **mailR** function just prior to the message being passed to a delivery command. For example, the surrogate entry

```
'.+' 'xxgate!.+' '< H=rewrite_for_xxgate;B=1024; \
            uux -a %R -p -- \\1!rmail' '(\\2)'
```

says to use **uux** to send the message to the system `xxgate`, but before doing so, `rewrite` it according to the **mailR** function `rewrite_for_xxgate()`. This might be useful in cases where there are special requirements for mail coming into the system `xxgate`, such as requiring `!`-style addresses in all `To` headers.

Note that this rewriting does not affect the message as seen by other recipients. Sometimes, however, it is necessary to have a message rewritten based on who it is coming from or going to, but irrespective of the delivery mechanism. This is accomplished using a `Rewrite` surrogate entry, as in

```
'xxgate!.+' '.+' 'Rewrite rewrite_coming_from_xxgate'
```

This says that any messages coming from the system xxgate should be rewritten according to the **mailR** function `rewrite_coming_from_xxgate()`. This might be useful if it is known that messages coming from the system `xxgate` have certain idiosyncrasies, such as using a special format for encoding attachments within the message.

See the file **/etc/mail/rewrite.samples** for other examples of **mailR** functions.

The localmail Command

Consider a user population spread across a group of machines (`sysa`, `sysb` and `sysc`) that has a disjoint address space. That is, a given user only has a login on one of the machines and none of the others. For example, if user `foo` had a login on `sysa`, the user would not have a login on `sysb` or `sysc`. This normally means that sending mail to a user on one of those machines will require knowing which machine the user resides on. However, it is possible to create a flat name space across the entire set of machines, such that you can send mail to user `xyz` on any of the machines and have the mail go to the user on the proper machine. This can be done in several different ways:

1. Set up accounts on all machines for all of the users, and establish forwarding files for each user so the mail is sent to the proper machine. This quickly becomes an administrative headache.
2. Create a mailing list that will be placed on each machine which lists the user and the system that user belongs on. For example, such a mailing list might read:

   ```
   usera usera@sysb
   userb userb@sysa
   ```

```
userc userc@sysb
userd userd@sysc
```

One advantage of this is that mail to a given user will stay on the machine if it belongs there, and go to the proper machine if it isn't local. However, this also requires that whenever a user is added to any of the systems, the mailing list must be updated on all machines.

3. Alternatively, the **`localmail`** program may be used. It takes a user name and looks it up on the system to see if it is to be considered a local user name. If it is local, it just prints the user name, but if not, it rewrites the user name according the options passed to it.

This makes **`localmail`** ideal for use as a translation surrogate. For example, the following surrogate entry could be placed at the end of the surrogate file:

```
&'.+' '[^!@]+' 'Translate T=1;B=*; R=|localmail -p -S\
@sysa' '%n
```

Any user names that are not local will be returned with the suffix `@sysa` added, which causes the mail to be forwarded to the system `sysa`. That single system would then need a master mailing list as described above which specifies where the user actually lives.

Note that this is similar to how a smarter host is handled. With a smarter host, any remote address that is not recognized is rewritten to go to another host, which presumably knows how to handle such addresses. In this case as well, user names that are not recognized as local user names are rewritten to go to another host, which presumably knows how to handle such addresses.

If you look at the **`/etc/mail/mailsurr`** file, you will find a commented out invocation of **`localmail`**.

MIME, Metamail, and /etc/mail/mailcap

The mail system supports the Multipurpose Internet Mail Extensions (MIME) standard in several ways. The introduction to the MIME standard says:

> RFC 822 defines a message representation protocol which specifies considerable detail about message headers, but which leaves the message content, or message body, as flat ASCII text. This document redefines the format of message bodies to allow multi-part textual and non-textual message bodies to be represented and exchanged without loss of information. This is based on earlier work documented in RFC 934 and RFC 1049, but extends and revises that work. Because RFC 822 said so little about message bodies, this document is largely orthogonal to (rather than a revision of) RFC 822.
>
> In particular, this document is designed to provide facilities to include multiple objects in a single message, to represent body text in charactersets other than US ASCII, to represent formatted multi-font text messages, to represent non-textual material such as images and audio fragments, and generally to facilitate later extensions defining new types of Internet mail for use by cooperating mail agents.

MIME is supported by the mail system in several ways:

- When reading a mail message that uses MIME capabilities beyond simple text, a MIME interpreter known as `metamail` will be invoked automatically.
- To create a mail message, either **`mailx`** or another program, known as **`mailto`**, may be used to compose messages which contain enhanced text, or have multiple parts.
- Support for additional message or attachment types, such as audio or visual attachments, is easily administered through the file **`/etc/mail/mailcap`**.

As an administrator, you will only have to become involved with MIME if you wish to add support for a new mail attachment format. To add support for a new attachment format, you will need to know several things:

- What command can be invoked to create a file of this type?
- What command can be invoked to display a file of this type?
- How would you describe this file type?
- Does the terminal need to be accessed when showing a file of this type?

As an example, consider the attachment format known as message/external-body. This is a special type of attachment which actually refers to a file found on some other system or across an Internet link. To create one, you would use the command extcompose with a single argument, which specifies where to put the output. This command generates the output as a series of headers followed by a body which gives additional information needed by some external access methods. This is specified in the **`mailcap`** file by an entry which says

```
composetyped = extcompose %s;
```

The keyword *composetyped* indicates that headers are output as well as a body. If there were no headers in the output, the keyword *compose* would be used instead. The *%s* indicates where to put the output and will be filled in with a filename.

To display a message/external-body, you would invoke the **`showexternal`** program, passing in various parameters found on the `Content-type` header of such a message. This is the first field of the **`mailcap`** entry for message/external-body and would be specified by

```
showexternal %s %{access-type} %{name} %{site}\
    %{directory} %{mode} %{server};
```

This program interacts with the user, so a terminal is needed. This requires that the *needs-terminal* keyword be specified.

We can call this type of attachment “Reference to data in external location.” This would be specified as part of a `label` field.

Putting this together produces a **`mailcap`** entry which looks like this:

```
message/external-body; showexternal %s %{access-type}\
 %{name} %{site} %{directory} %{mode} %{server}; \
    needsterminal; composetyped = extcompose %s; \
    description="Reference to data in external location"
```

As you can see, each field is separated by a semi-colon, and continuation lines are specified by putting a backslash at the end of each line. A complete description of the **`mailcap`** file can be found in **`mailcap(4).`**

Locales and Character Sets

MIME requires access to a character set name to describe the character set used by the different locales being used on the system. For example, the locale `german` would typically use the character set `iso8859-1`.

When showing a message created with the character set `iso8859-1`, the mail program will check the character set against your current character set and, if they don't match, lodge a complaint.

The character set in use by a user can be specified by setting the environment variable `MM_CHARSET` to the correct value. However, this requires that the user be informed about yet another environment variable (in addition to the `LOCALE` environment variable).

Eventually, the system incorporates character set information in the locale-specific system files. Until that time, however, character set information may be made available through a file named **`/etc/mail/charset/`***locale*, which may be created by the administrator for each locale in use on the system.

These files each have a single line, which specifies the character set for that locale. For example, the file **`/etc/mail/charset/german`** would contain the single line which says `iso8859-1`.

10 Sendmail

10
Sendmail

Introduction to Sendmail

As an alternative to mail (and uucp), **sendmail(1M)** may be used to send or receive remote mail from hosts connected via a network.

Sendmail(1M) was designed as a centralized mail facility used to route mail by interpreting addresses according to a set of well-defined (and configurable) rules. Message headers can also be rewritten to conform to a number of standards or specialized targets.

Sendmail(1M) does not interface directly with users nor does actual mail delivery. Rather, it collects mail messages generated by such mail interface programs as **mail(1)** or **mailx(1)**, edits the message as directed and then calls the appropriate mailer to do the mail delivery.

There is also a sendmail daemon that can be invoked as an SMTP server for incoming remote mail.

Mail vs. Sendmail

Usually, the option for processing remote mail will come down to either *mail* (i.e. **smtp**) or s*endmail*. From a basic functional standpoint both facilities are similar. Often the decision will be based on the environment that the system will be in. For example, it might be desirable to choose the mail facility that is used by the majority of the local hosts within the network.

Sendmail Aliases

The sendmail alias database exists in two forms. One in a text form, maintained in the file **/etc/ucbmail/aliases** (there is also a soft link at **/usr/ucblib/aliases**).

Aliases are of the form:

alias: recipient1, recipient2, ...

Aliases may be continued by starting any continuation lines with a space or tab. Blank lines and lines beginning with a sharp sign (“#”) are comments.

Note that aliases are case-insensitive. As an example, if the alias file contained the line:

fred: name1, ...

then mail addressed to fred, Fred or FRED would all be delivered according to the above alias.

This would also mean that if the aliases file has two lines such as:

fred: name1
Fred: name2

then only one of these alias entries would be used (generally, the last alias overrides any previous alias when there are duplicate names). Note that **`sendmail(1M)`** prints a warning when building the aliases database when it detects duplicate alias names.

The second form of the sendmail alias database is processed by the **`dbm(3N)`** subsystem. This form is in the files **`/etc/ucbmail/aliases.dir`** and **`/etc/ucbmail/aliases.pag`**. For performance reasons, this is the form that sendmail actually uses to resolve aliases.

Rebuilding the Sendmail Alias Database

The dbm version of the *sendmail* alias database may be rebuilt by executing the command:

```
/usr/ucb/newaliases
     -or-
/usr/ucblib/sendmail -bi
```

NIS and Sendmail Aliases

NIS support is available for the aliases used by **`sendmail(1M)`**. The advantage of using NIS is that it eliminates the need for maintaining an alias file on each system in a computer network. Instead, it is only necessary to maintain a single copy of the sendmail aliases - on the NIS master system.

By using NIS, client systems would only need a small number of host specific aliases. Other aliases would be common throughout the network and can be in the NIS-managed set (which is the set of sendmail aliases from the file **`/etc/ucbmail/aliases`** on the NIS master system). The full set of aliases for a client would be the union of the host specific aliases and the NIS-managed set of aliases.

NIS is enabled on a client by adding a line beginning with the NIS special token '+' to the *sendmail* aliases file (**`/etc/ucbmail/aliases`**). The token indicates that the NIS-managed set of sendmail aliases should be “included” at this point in the aliases file.

Note that this is the only NIS special token supported for sendmail aliases (and it is only possible to include the entire NIS set of sendmail aliases).

Pre-User Forwarding

As an alternative to the alias database, a user may put a file with the name ".forward" in his/her home directory. If this file exists, **`sendmail(1M)`** redirects mail for that user to the list of addresses listed in the .forward file.

Sendmail Configuration File

`Sendmail(1M)` is distributed with two configuration files: **`/etc/ucbmail/main.cf`** and **`/etc/ucbmail/subsidiary.cf`**. One of these files should be installed at **`/etc/ucbmail/sendmail.cf`**. The type of configuration will determine which file to use.

If the system will be the master host of the local network then the configuration file **`main.cf`** should be used.

If the system is NOT the master host of the local network then the configuration file **`subsidiary.cf`** should be used (the default).

The configuration file contains header declarations, mailer definitions and address rewriting rules. The syntax of this file is designed to be easy to parse.

More detail about the syntax of the configuration file provided later in this chapter.

Configuration File Syntax and Semantics

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

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse. Since this is done every time sendmail starts up, it is rather easy for a human to read or write.

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

The Syntax

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

R and S - Rewriting Rules

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

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

The syntax of these two commands are:

`*n*`

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

`R``*lhs rhs comments*`

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

D - Define Macro

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

The syntax for macro definitions is:

D*x val*

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

$*x*

C and F - Define Classes

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

The syntax is:

C*c word1 word2*
F*c file*[*format*]

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

CHmonet ucbmonet

and

CHmonet
CHucbmonet

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

named *file*; the *format* is a *scanf(3)* pattern that should produce a single string.

M - Define Mailer

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

M*name, {field=value}**

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

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

Only the first character of the field name is checked.

H - Define Header

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

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

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

Some headers have special semantics that will be described below.

O - Set Option

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

O*o value*

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

T - Define Trusted Users

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

T*user1 user2...*

There may be more than one of these lines.

Precedence Definitions

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

P*name=num*

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

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

The Semantics

This section describes the semantics of the configuration file.

Special Macros, Conditionals

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

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

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

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

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

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

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

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

a	The origination date in Arpanet format
b	The current date in Arpanet format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	Sendmail's pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user

v	The version number of sendmail
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

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

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

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

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

The **$p** and **$t** macros are used to create unique strings (e.g., for the “Message-Id:” field). The **$i** macro is set to the queue id on this host; if put into the timestamp line it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of *sendmail*; this is normally put in timestamps and has been proven extremely useful for debugging. The **$w** macro is set to the name of this host if it can be determined. The **$c** field is set to the “hop count”, i.e., the number of times this message has been processed. This can be determined by the **-h** flag on the command line or by counting the timestamps in the message.

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

Conditionals can be specified using the syntax:

$?x text1 $| text2 $.

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

Special Classes

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

The Left Hand Side

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

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

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

$\-:$+

is applied to the input:

UCBARPA:eric

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

$1	UCBARPA
$2	eric

The Right Hand Side

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

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

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

A host name enclosed between **$[** and **$]** is looked up in the */etc/hosts* file and replaced by the canonical name. For example, “$[csam$]” would become “lbl-csam.arpa”.

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

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

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

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

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

The **$@** and **$:** prefixes may precede a **$>** spec; for example:

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

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

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

Semantics Of Rewriting Rule Sets

There are five rewriting sets that have specific semantics. These are related as depicted by Figure 10-1.

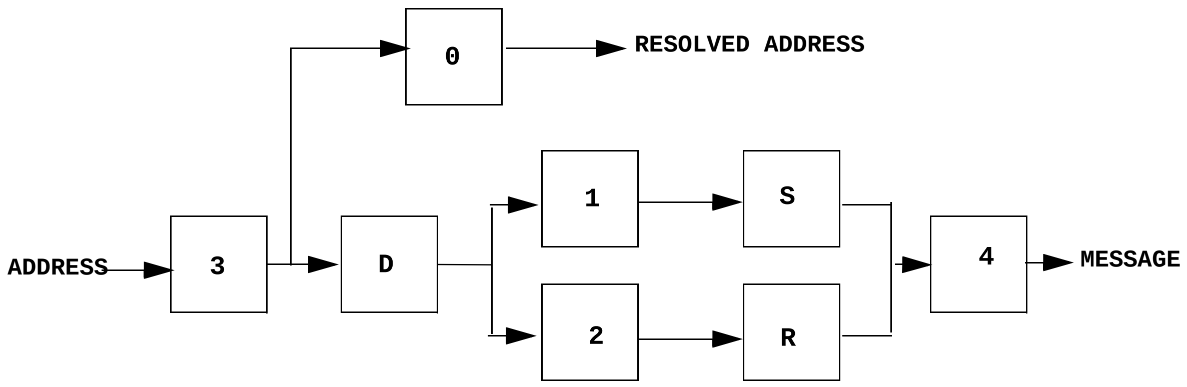


Figure 10-1. Rewriting Set Semantics

Where:

D - sender domain addition,
S - mailer-specific sender rewriting, and
R - mailer-specific recipient rewriting.

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

local-part@host-domain-spec

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

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

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

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

Mailer Flags etc

There are a number of flags that may be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. These are detailed in Table 10-1. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers.

Table 10-1. Mailer Flags

Flag	Flag Description
e	This mailer is expensive to connect to, so try to avoid connecting normally; any necessary connection will occur during a queue run.
f	The mailer wants a **-f** *from* flag, but only if this is a network forward operation (i.e., the mailer will give an error if the executing user does not have special permissions).
h	Upper case should be preserved in host names for this mailer.
l	This mailer is local (i.e., final delivery will be performed).
m	This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users.
n	Do not insert a UNIX-style “From” line on the front of the message.
p	This mailer wants a “Return-Path:” line.
r	Same as **f**, but sends a **-r** flag.
s	Strip quote characters off of the address before calling the mailer.
u	Upper case should be preserved in user names for this mailer.
x	This mailer wants a “Full-Name:” header line.
A	This is an Arpanet-compatible mailer, and all appropriate modes should be set.

Table 10-1. Mailer Flags (Cont.)

Flag	Flag Description
C	If mail is *received* from a mailer with this flag set, any addresses in the header that do not have an at sign ("@") after being rewritten by ruleset three will have the "@domain" clause from the sender tacked on. This allows mail with headers of the form: From: usera@hosta To: userb@hostb, userc to be rewritten as: From: usera@hosta To: userb@hostb, userc@hosta %automatically.
D	This mailer wants a "Date:" header line.
E	Escape lines beginning with "From" in the message with a ">" sign.
F	This mailer wants a "From:" header line.
I	This mailer will be speaking SMTP to another *sendmail* - as such it can use special protocol features. This option is not required (i.e., if this option is omitted the transmission will still operate successfully, although perhaps not as efficiently as possible).
L	Limit the line lengths as specified in RFC821.
M	This mailer wants a "Message-Id:" header line.
P	Use the return-path in the SMTP "MAIL FROM:" command rather than just the return address; although this is required in RFC821, many hosts do not process return paths properly.
S	Don't reset the userid before calling the mailer. This would be used in a secure environment where *sendmail* ran as root. This could be used to avoid forged addresses. This flag is suppressed if given from an "unsafe" environment (e.g, a user's mail.cf file).
U	This mailer wants UNIX-style "From" lines with the ugly UUCP-style "remote from <host>" on the end.
X	This mailer want to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This insures that lines in the message containing a dot will not terminate the message prematurely.

The error Mailer

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

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

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

Building a Configuration File

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

Purpose

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

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer. *Sendmail* does this in three subphases. Rulesets one and two are applied to all sender and recipient addresses respectively. After this, you may specify per-mailer rulesets for both sender and recipient addresses; this allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

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

Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. A few possible philosophies are presented here.

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

Large Site, Many Hosts - Minimum Information

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

For example, a typical site on their local ethernet network is “monet”. Monet has a list of known ethernet hosts; if it receives mail for any of them, it can do direct delivery. If it receives mail for any unknown host, it just passes it directly to “ucbvax”, their master host. Ucbvax may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new ethernet host is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated as convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, their TCP connection is currently on “ucbarpa”. However, monet “*does not*” know about this; the information is hidden totally between ucbvax and ucbarpa. Mail going from monet to a TCP host is transferred via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to the Arpanet. Although this involves some extra hops, they feel this is an acceptable trade-off.

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

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

Small Site - Complete Information

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

Single Host

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

Relevant Issues

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

RFC822 describes the format of the mail message itself. *Sendmail* follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

< > ( ) " \\

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

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially each host is given a name which is a right-to-left dot qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Berkeley one legal host is "a.cc.berkeley.arpa"; reading from right to left, "arpa" is a top level domain (related to, but not limited to, the physical Arpanet), "berkeley" is both an Arpanet host and a logical domain which is actually interpreted by a host called ucbvax (which is actually just the "major" host for this domain), "cc" represents the Computer Center, (in this case a strictly logical entity), and "a" is a host in the Computer Center; this particular host happens to be connected via berknet, but other hosts might be connected via Ethernet or some other network.

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

How To Proceed

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

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since *sendmail* likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Berkeley configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

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

Testing The Rewriting Rules - The -bt Flag

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

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

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

```
rwset address
```

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

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

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

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

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

turns on an incredible amount of information; a single word address is probably going to print out several pages worth of information.

Building Mailer Descriptions

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

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

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

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

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

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

will be modified to:

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

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

Other flags are described in Table 10-1.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses respectively. These are applied after the sending domain is appended and the general rewriting sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or

```
From: ucbvax!eric
```

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

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

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

IPC $h [*port*]

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

For example, the specifications:

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

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called “local”, is located in the file “/bin/mail”, takes a picky **-r** flag, does local delivery, quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the message and ruleset twenty should be applied to recipient addresses; the argv to send to a message will be the word “mail”, the word “-d”, and words containing the name of the receiving user. If a **-r** flag is inserted it will be between the words “mail” and “-d”. The second mailer is called “ether”, it should be connected to via an IPC connection, it can handle multiple users at once, connections should be deferred, and any domain from the sender address should be appended to any receiver name without a domain; sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000 byte limit on messages passed through this mailer.

Enabling Sendmail

Changes are required to the **`mailsurr(4)`** file (**`/etc/mail/mailsurr`**) in order to have **`sendmail(1M)`** process outgoing remote mail.

Note that **`sendmail(1M)`** typically uses **`mail(1)`** to deliver local mail. Hence, it is not advisable to configure the **`mailsurr(4)`** file so that local mail is sent to **`sendmail(1M)`**. This would generally lead to **`sendmail(1M)`** recusively calling **`mail(1)`** and vice versa until eventually an error occurs.

There are several different ways that **`mailsurr(4)`** might be changed to enable **`sendmail(1M)`**. The following lists typical changes required - (these are also documented in the **`mailsurr(4)`** file).

In Address Translation section (i.e. part 2) :

- Disable -ALL- address translation commands. (Note that it is expected that similar translations will be done by **`sendmail(1M)`** as specified by the configuration file.
- Enable the translation command that invokes ucb mail alias. This is used to do **`sendmail(1M)`** alias processing on all mail - local and remote.

In Delivery Commands section (i.e. part 3) :

- Disable uux and smtpqer. Enable **`sendmail(1M)`**.

It is also possible to have **`sendmail(1M)`** process incoming remote mail instead of **`smptd(1M)`** (SMTP daemon).

- Disable **`smtpd(1M)`** by removing the entry for it in the inetd configuration file **`/etc/inet/inetd.conf`**.
- Enable sendmail daemon in file **`/etc/inet/rc.inet`**.

Sendmail Utilities

A brief description of **`sendmail(1M)`** utilities:

`/usr/ucblib/sendmail`	- **`sendmail(1M)`**
`/usr/ucb/mailq`	- displays contents of sendmail queue.
`/usr/ucb/mailstats`	- displays sendmail statistics.

/usr/ucb/mconnect	- connect to mail server on remote host.
/usr/ucb/newaliases	- rebuilds sendmail alias database.
/usr/ucblib/ucbmailalias	- translate sendmail alias name.

Sendmail Mail Queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary.

The command

/usr/ucblib/mqueue (or **/usr/ucblib/sendmail -bp**)

will produce a listing of the queued mail with: id numbers, message size, message date, time message was queued, sender and recipients.

The queue files are located in the directory **/var/spool/mqueue**. Files have the form xfAA99999 where AA99999 is the id of the file. x is the type and can be one of:

d	- date file.
l	- lock file.
n	- id creation temporary file.
q	- queue control file.
t	- temporary file.
x	- transcript file.

qf files are structured as a series of lines beginning with a code letter. The code letters are as follows:

D	- name of date file.
H	- header definition.
M	- message.
P	- message priority.
R	- recipient address.
S	- sender address.
T	- job creation time.

Sendmail Log

If your system is running **syslog(1M)** (enabled by default), then a log of sendmail operations can be maintained. The location of the log can be specified in the file **/etc/syslog.conf**. By default, log data associated with sendmail is kept at **/var/spool/mqueue/syslog**.

Each line in the log consists of a timestamp, the name of the system that generated it, the word sendmail and a message.

The log can grow to be quite large. The amount of information that is logged can be controlled by indicating a logging level. At the lowest level only extremely strange situations are logged. At the highest level, even mundane and uninteresting events are logged.

The following table lists the definitions for the log levels. By default, the log level specified is 2 in the subsidiary configuration file (**`subsidiary.cf`**) and 1 in the main configuration file (**`main.cf`**).

0	- no logging.
1	- major problems only.
2	- message collections and failed deliveries.
3	- successful deliveries.
4	- deferred messages.
5	- normal queued messages.
6	- unusual but benign incidents.
9	- queue id to external message id mappings.
12	- debugging messages.
16	- verbose information about the mail queue.

3
TCP/IP Network Administration

Replace with Part 3 tab

11
Introduction to Administering TCP/IP Networks

11
Introduction to Administering TCP/IP Networks

About TCP/IP Network Administration

The TCP/IP administration chapters explain how to set up and administer a network built on the TCP/IP Internet protocol family. The administrative steps needed are simple and few, once you understand the concepts underlying them. Therefore, this guide concentrates on the concepts that will allow you to install the most appropriate network for your particular needs.

How This Part is Organized

The TCP/IP administration chapters are organized as follows:

- “Introduction to Administering TCP/IP Networks” describes the concepts you need to understand to effectively do TCP/IP administration for your systems.
- “Setting Up TCP/IP” contains step-by-step procedures for many basic tasks you need to do as a TCP/IP administrator.
- “Setting Up Routers and Subnetworks” describes how to expand and manage growing systems.
- “Managing TCP/IP Nodes Using SNMP” tells how to use SNMP to perform network monitoring and management functions.
- “Using Domain Name Service with TCP/IP” describes concepts and procedures relating to domain name service.
- “Troubleshooting and Tuning TCP/IP” describes how to diagnose problems and tune your system to improve TCP/IP performance.
- “Obtaining IP Addresses” describes how to obtain and complete IP address registration forms.
- “Obtaining Domain Names” describes how to obtain and complete domain name registration forms.
- “Network Time Synchronization” describes how to synchronize time among the machines on your network.

Introducing the Internet Protocol Suite

A network is a configuration of machines that exchange information among themselves. In order for the network to function properly, the information originating at a sender must be transmitted along a communication line and delivered to the intended recipient in an intelligible form. Because different types of networking software and hardware need to interact to perform this function, network designers developed the concept of the communications protocol family (or suite). A network protocol is a set of formal rules explaining how software and hardware should interact within a network in order to transmit information. The Internet protocol family is one such group of network protocols. It is centered on the Internet Protocol (IP). The other members of the Internet protocol family are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), and Internet Control Message Protocol (ICMP).

The entire family is popularly referred to as TCP/IP, reflecting the names of the two main protocols. This is also the terminology used in this document.

TCP/IP provides service to many different types of host machines connected to heterogeneous networks. These networks may be wide area networks, such as X.25-based networks, but they may also be local area networks, such as one you might install in a single building.

TCP/IP was originally developed by the United States Department of Defense to run on the ARPANET, a packet-switching wide area network first demonstrated in 1972. Today the ARPANET is part of a wide area network known as the DoD (Department of Defense) Internet, or, for short, the Internet. Many popular texts use the term Internet to describe both the protocol family and the wide area network. This text uses the term TCP/IP to refer to the Internet protocol suite and Internet when referring to the network itself.

The TCP/IP protocol structure can be conceptualized as being formed of a series of layers as shown in Table 11-1.

Table 11-1. TCP/IP Protocol Layers

Layer	Network Services
Application	telnet, ftp
Transport	tcp, udp
Network	ip, icmp
Data Link	arp, rarp, device driver (such as Ethernet)
Physical	cable or other device (such as an Ethernet board)

TCP/IP involves communication between pairs of hosts on a network, or peers. Each protocol layer on one peer has a corresponding layer on the other peer. Each layer is required, by design, to handle communications in a pre-determined fashion. Each protocol formats communicated data and appends information to or removes information from it.

Then the protocol passes the data to a lower layer on the sending host or a higher layer on the receiving host, as illustrated in Figure 11-1.

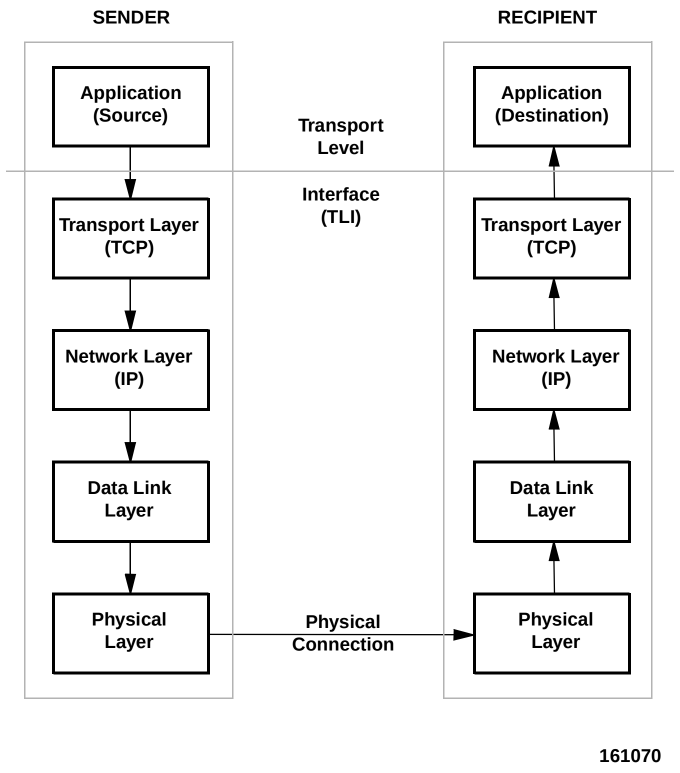


Figure 11-1. Sender/Receiver Interaction

In the OS implementation, the interface between the application and the transport layers is the Transport Level Interface (TLI), an interface that eliminates the need for applications to know particulars about the transport layer. Any application written to the TLI can run on a TCP/IP network.

The next subsections briefly explain how each protocol layer handles messages. For more detailed information, refer to the manual page for the appropriate protocol.

Physical Layer

The physical layer is the hardware level of the protocol model that is concerned with electronic signals. Physical layer protocols send and receive data in the form of packets.

A packet contains a source address, the transmission itself, and a destination address.

TCP/IP supports a number of physical layer protocols, including Ethernet. Ethernet is an example of a packet-switching network; its communications channels are occupied only for the duration of the transmission of a packet. On the other hand, the telephone network is an example of a circuit-switching network.

Data Link Layer

The data link layer is concerned with addressing at the physical, machine level. Protocols at this layer are involved with communications controllers, their chips, and their buffers. Ethernet is supported at this layer by TCP/IP.

Two additional TCP/IP protocols, ARP and RARP, can be viewed as existing between the network and data link layers. ARP is the Ethernet address resolution protocol. It maps known IP addresses (32 bits long) to Ethernet addresses (48 bits long).

RARP (or Reverse ARP) is the IP address resolution protocol. It maps known Ethernet addresses (48 bits long) to IP addresses (32 bits long), the reverse of ARP.

Network Layer

IP (Internet Protocol) and Internet Control Message Protocol (ICMP) are the protocols present at the network layer.

IP provides machine-to-machine communication. It performs transmission routing by determining the path a transmission must take, based on the receiving machine's IP address. IP also provides transmission formatting services; it assembles data for transmission into an *internet datagram*.

The datagram defines the structure and form in which the information is sent. If the datagram is outgoing (received from the higher layer protocols), IP attaches an IP header to it. This header contains a number of parameters, including the IP addresses of the sending and receiving hosts. For more information about IP, refer to the **`IP(7)`** manual page.

ICMP sends error or control messages to other hosts. For more information about ICMP, refer to the **`ICMP(7)`** manual page.

Transport Layer

The TCP/IP transport layer protocols enable communications between processes running on separate machines. Protocols at this level are TCP and UDP.

TCP (Transmission Control Protocol) enables applications to talk to one another via virtual circuits, as if connected by physical circuits. TCP is a connection-oriented, reliable delivery protocol; any data written to a TCP connection will be received by its peer in sequence, or an error indication will be returned.

UDP (User Datagram Protocol) is the alternative protocol available at the transport layer. UDP is a connectionless datagram protocol. Datagrams are groups of information transmitted as a unit to and from the upper layer protocols on sending and receiving hosts. UDP datagrams use port numbers to specify sending and receiving processes. However, no attempt is made to recover from failure or loss; packets may be lost with no error indication returned.

Whether TCP or UDP is used depends on the network application invoked by the user. For example, if the user invokes **`telnet`**, that application passes the user's request to TCP. If the user's request involves the Domain Name Service, that application passes the request to UDP.

For information about the TCP and UDP protocols, refer to the **`TCP(7)`** and **`UDP(7)`** manual pages.

Application Layer

A variety of TCP/IP protocols exist at the application layer. Here is a description of some of the more widely used.

Telnet

The Telnet protocol enables terminals and terminal-oriented processes to communicate on a network running TCP/IP. It is implemented as the program **`telnet`** on the local machine and the daemon **`telnetd`** on the remote machine. **`telnet`** provides a user interface through which two hosts can open communications with each other, then send information on a character-by-character or line-by-line basis. The application includes a series of commands, which are documented in the **`telnet(1)`** manual page. The **`telnetd`** daemon on the remote host handles requests from the telnet command. For more information about telnetd, see the **`telnetd(1M)`** manual page.

FTP

The File Transfer Protocol (FTP) transfers files to and from a remote network. The protocol includes the **`ftp`** command on the local machine and the **`ftpd`** daemon on the remote machine. **`ftp`** lets you specify on the command line the host with whom you want to initiate file transfer and options for transferring the file. The **`ftpd`** daemon on the remote host handles the requests from your **`ftp`** command.

The options to **`ftp`**, as well as the commands you invoke through the **`ftp`** command interpreter, are described on the **`ftp(1)`** manual page.

The services provided by the **`ftpd`** daemon are described on the **`ftpd(1M)`** manual page.

Domain Name Service

The Domain Name Service (DNS) is a protocol that provides domain-name-to-address-mapping of hosts and mail recipients on a network. DNS is described on the **`named(1M)`** manual page.

For information on how to implement the Domain Name Service on your network, see the chapter “Using Domain Name Service with TCP/IP”.

Other Application Layer Protocols

Other application layer protocols exist that are also implemented as a program on the local machine and a daemon on the remote machine. Examples of these are **`rlogin`** and **`rlogind`**, which permit a user to log on to a remote machine; **`rsh`** and **`rshd`**, which enable the user to spawn a shell on a remote machine; and **`finger`** and **`fingerd`**, which permit a user to obtain information about users on remote machines.

To avoid having an excess of daemons running at all times, the daemon **`inetd`** is initiated at start-up time. After consulting the **`/etc/inetd.conf`** file, **`inetd`** runs the appropriate daemons as needed. For example, the daemon **`rlogind`** will be run by **`inetd`** whenever there is a request for a remote login from another machine, and only at that time and for the duration of the remote login.

TCP/IP Concepts

As you use the procedures and descriptions in this guide, there are concepts that come up again and again. The following are descriptions of the most important concepts you should understand throughout this guide.

IP Addresses and Domain Names

In order for all computers connected to the Internet to be able to communicate with each other, the Network Information Center (NIC) controls addresses and names in a way that they are unique across the entire Internet. If your organization wants to connect to the Internet, you apply to the NIC for an Internet network ID and a domain name. You then assign host numbers and system names, respectively, within the assigned network ID and domain.

IP Addresses

A TCP/IP network routes a packet according to the destination IP address, an address provided by the IP protocol on the sending host. IP addresses are 32-bit numbers that uniquely identify every host computer connected to the Internet. This 32-bit address is typically represented as four numbers that are separated by dots. For example, the following is an IP address:

```
137.65.208.51
```

Because each of the four numbers is represented by an eight-bit field (or octet), each number must fall between 0 and 255. (The last field, however, can only be assigned numbers from 1-254 for a Class C address, or for a Class B address with an 8-bit subnet. This is because the host ID part of the address cannot be all 0's or all 1's.)

NOTE

The numbers 0 and 255 in the host ID part of an IP address are reserved.

The tricky part about IP addresses is that the network ID (the part of the address that identifies your network) and the host ID (the part that identifies an individual host on your network) must all fit into this 32-bit address space. And, because the size of the network ID is different for different network "classes," the number of hosts you can configure for your network varies. To further complicate matters, if your network consists of several physical networks you must further partition your host ID to use part of it as a subnet ID.

Descriptions of the parts of an IP address (network ID, host ID, and optional subnet ID) are contained below.

Network ID

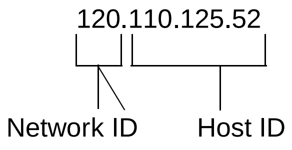
The network ID is the first part of the IP address for every host in the network you manage. The network ID was designed to identify part of a total IP address as the network, and leave the rest of the IP address to identify a particular host on that network.

The Network Information Center (NIC) that assigns network IDs for the Internet takes into account the fact that different organizations manage different numbers of hosts. NIC therefore assigns you a network ID belonging to one of the following three classes, based on how many hosts you have, or expect to have, on your network: Class A, Class B, or Class C. The three classes use progressively more of the IP address to identify the network, and therefore leave you with fewer numbers to uniquely identify your hosts.

Network Classes

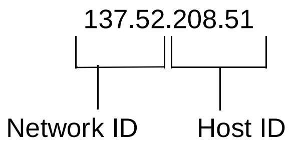
Here are descriptions of the three common network classes available from the NIC. Once you understand how classes are identified, you can tell at a glance the class of a network for any IP address.

Class A For a Class A address, the first part of the four-part IP address is between 1 and 127. The network ID assigned for a Class A address consumes only the first part of the IP address and the rest can be used for the host ID. For example, the following is a Class A IP address.



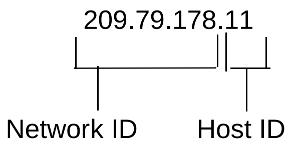
In this example, the NIC assigned the first part (120) as your network ID and you supplied the remaining three parts (.110.125.52) as the host ID. Notice that by controlling the final three parts of the address, you would have access to a huge number of host IDs. For that reason, Class A networks are rarely given out.

Class B For a Class B address, the first part of the four-part IP address is between 128 and 191. The network ID assigned for a Class B address consumes the first two parts of the IP address and the rest can be used for the host ID. For example, the following is a Class B IP address.



In this example, the NIC assigned the first two parts (137.52) as your network ID and you supplied the remaining two parts (.208.51) as the host ID. Even a Class B address provides a large number of possible host addresses (more than 65,000). Because Class B addresses are becoming scarce, you need a convincing reason for needing a Class B address.

Class C For a Class C address, the first part of the four-part IP address is between 192 and 223. The network ID assigned for a Class C address consumes the first three parts of the IP address and the last part can be used for the host ID. For example, the following is a Class C IP address.



In this example, the NIC assigned the first three parts (209.79.178) as your network ID and you supplied the remaining part (.11) as the host ID. A Class C address allows you to connect 254 hosts on your network (numbers 0 and 255 are reserved). Chances are you will be assigned a Class C network ID. If you expect to have more than 254 hosts on your network, NIC will probably assign you several Class C network IDs.

Host ID

The host ID is the part of the IP address that uniquely identifies a host within your network (network ID). A host ID cannot be all 0's or 1's. The numbers available to use as a host ID depend on the class of your network (see class descriptions above) and whether or not you are doing subnetting (see the section “Subnet ID”).

Subnet ID

If you have more than one physical network within your organization, but only one network ID, you can take part of your host ID area and make it into a subnet ID. A subnet ID is optional. You assign your own subnet IDs (the NIC does not assign subnet IDs).

Typically, you would subnet if you have a Class B address. Rather than have up to 65,000 hosts connected to a single physical network, you would use the third part of the IP address to indicate the subnetwork number and the fourth part as the host ID.

NOTE

Technically, you can take any bit (or bits) from a host ID and use it for subnetting by masking that bit (or bits). For example, you could use 9 bits for the subnet, leaving 7 for the hosts.

Figure 11-2 is an example where a Class B network number is divided into several subnetworks:

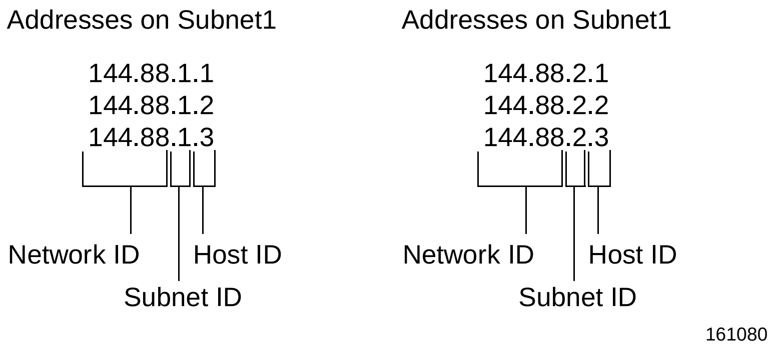


Figure 11-2. Using Subnet IDs in IP Addresses

In Figure 11-2, your Class B address is 148.88, your subnet ID is 1 or 2, and host IDs from 1 to 254 can be assigned for each subnet. To actually use subnetting, you must configure routing as described in the chapter “Setting Up Routers and Subnetworks”.

Domain Names

A domain is a set of machines that is administered and maintained as a single entity. Generally, all the machines on a local network constitute a domain on the larger network; however, you may choose to break the local network into several administrative entities, called subdomains. For example, you may want all the machines in the accounting department to comprise one subdomain, and all the machines in marketing to comprise another.

To join the Internet, your domain must be named according to Internet naming conventions, and you must register your domain name with the NIC. Even if you do not intend to join the Internet at this time, it is recommended that you follow the Internet naming conventions and register with the NIC to avoid problems in the future, should you decide to join the network.

This section explains the Internet naming conventions. See the chapter “Obtaining Domain Names” for further information on getting domain name assignments from the NIC. (If you intend to join a public network other than the Internet, such as BITNET or CSNET, contact the organization that administers the network for information about naming conventions and for a registration form.)

Organization of Internet Domains

The Internet consists of four levels of domain: the root level, the top level, the second level, and the local level. Each level branches from the level above it. Below the local level are hosts and, optionally, subdomains.

Figure 11-3 shows the different domain levels of the Internet and how they relate to each other.

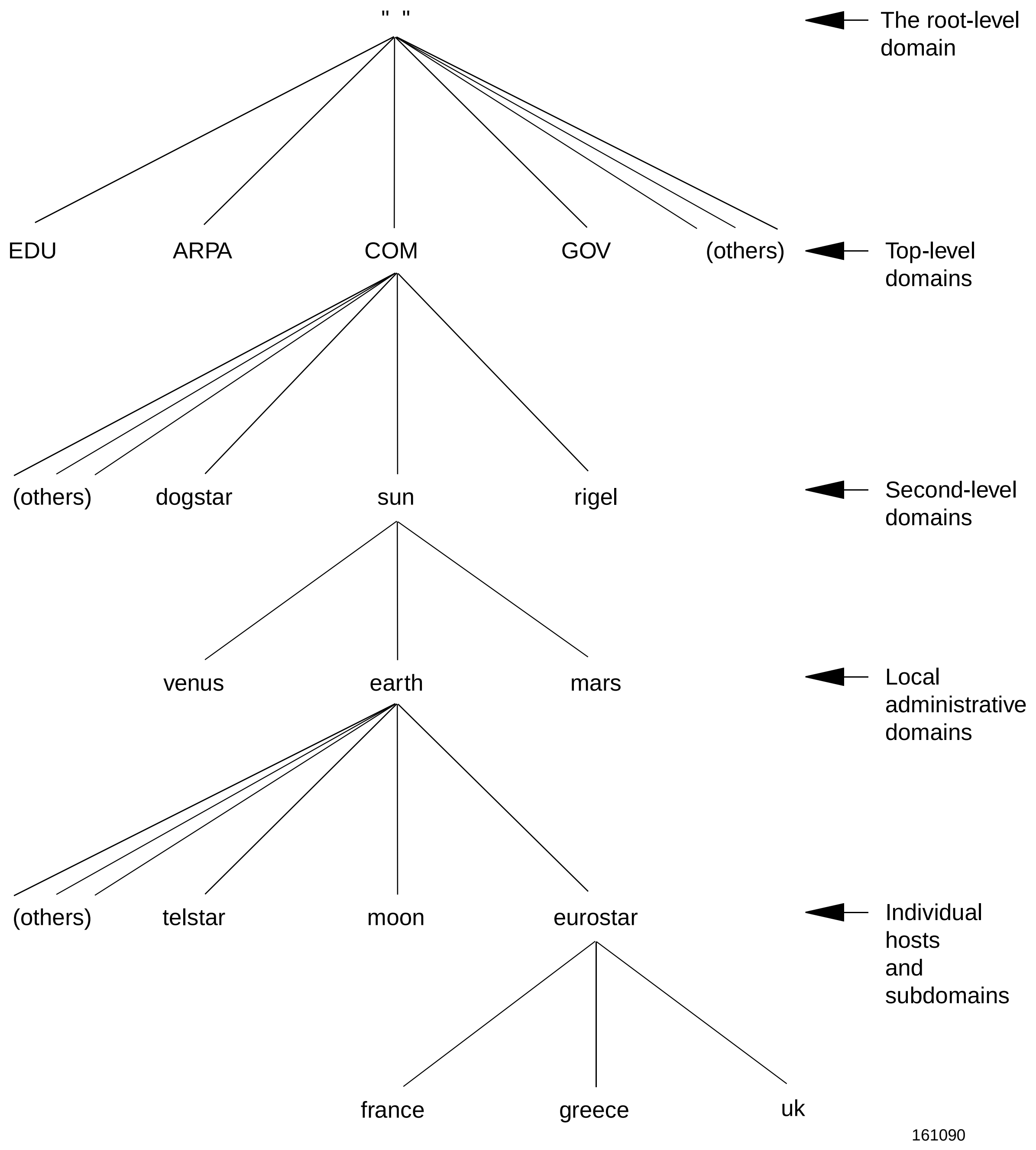


Figure 11-3. Typical Internet Domain Hierarchy

At each level of the Internet are name servers. A name server is a machine that maintains information about machines at the next lower level and facilitates name-to-address mapping.

The different levels of the Internet are described below:

Root-level domain

The root level is the top of the entire Internet; it is maintained by the NIC. (Organizations in charge of other public networks, such as BITNET and CSNET, also administer the roots of their networks.)

At the root level, the NIC administers root domain name servers, which maintain information about name servers at the next lower level.

Top-level domains

The root level branches into top level domains. Currently, some Internet top level domains are EDU, ARPA, COM, GOV, MIL, NET, ORG, and US.

The names of the top level domains are assigned by the NIC. They reflect the types of networks that make up the domain. For example, EDU (an abbreviation of Educational) is a top level domain that consists of networks administered by universities. COM (an abbreviation of Commercial) consists of networks belonging to business organizations.

When you register with the NIC, it assigns your network to one of these domains, depending on your organization's function.

Second-level domains

Each top level domain branches into second level domains. Member organizations at the second level assign “domain administrators” to manage their name servers, and the NIC assigns a “technical contact” to coordinate administration across domains. Depending upon your site's size and requirements, you may have domain administrator responsibilities.

Local administrative domains

Within each second-level domain are local administrative domains. These are the domains you administer for your organization. A local domain can be as small as one host, or large enough to include many hosts and additional name servers. It may also have other administrative domains (called subdomains) nested within it.

For more information about name servers and administrative responsibilities, see Chapter 15, "Using Domain Name Service with TCP/IP".

Selecting a Domain Name

The Internet is organized as a hierarchy of domains. The name space is organized as a tree according to organizational or administrative boundaries. Each node of the tree is a domain, which is given a label. The name of the domain is the concatenation of all the labels of the domain from the root to the current domain, listed from right to left and separated by dots. A host address for a host named monet at Podunk University might look like this:

```
monet.podunk.EDU
```

The top level domain for educational institutions is `EDU`, `podunk` is a subdomain of `EDU` and `monet` is the name of the host.

When you name your domain, you must select a label that is unique within its domain. For example, you could use `podunk` as a label for a domain consisting of all machines at Podunk University, provided that another university in the domain `EDU` has not used this name first. The label you assign a host within your domain must be unique as well, but only within domain `podunk`.

Once you choose a label, you must concatenate it with the labels of all the higher level domains to create your domain name.

Domains are not limited to any fixed number of levels. For example, the Amalgamated Widgit Company might be registered with the NIC as belonging to the domain **`Widgit.COM`**; it might also have a subdomain called **`eng.Japan.Widgit.COM`** for the engineering group within their Japan subsidiary, and another subdomain called **`mktg.Japan.Widgit.COM`** for marketing in Japan. However, machines at company headquarters might belong to the subdomain called **`HQ.Widgit.COM`**.

NOTE

Upper- and lowercase letters are not significant in domain names. The traditional UNIX system convention is to use all lowercase; many other systems use uppercase.

At many sites, users name their individual hosts while the administrators name the servers. The administrator should ensure that there are no duplicate names within a domain by using the **`nslookup`** program [see the **`nslookup(1M)`** manual page]. **`nslookup`** is an interactive program that you can use (among other things) either to produce a listing of all the hosts in the domain (by using the **`ls`** command at the **`>`** prompt), or to obtain information about a host. At the prompt, you enter the host's name. If the host is in the database, **`nslookup`** produces its address; otherwise, it informs you that the host does not exist.

Registering Your Domain

After choosing the name of your upper domain (the equivalent of `Widgit.COM` in the example in the preceding section), you need to register it with a higher-level domain. To do this, you should get in touch with the administrator of the higher domain. The top level domains are administered by the NIC at SRI International. To register a domain with NIC, you request the Domain Registration Form from the NIC and complete it according to the instructions. See the chapter “Obtaining Domain Names” for information on getting and completing NIC domain registration forms. Once your domain is registered, you can divide it into subdomains as the need arises.

12
Setting Up TCP/IP

12
Setting Up TCP/IP

Introduction to TCP/IP Setup

This chapter contains procedures you need to configure and run TCP/IP on your computer. The chapter includes information about controlling access to your computer by remote users. It also contains a section explaining security issues related to the use of Internet network services with the Enhanced Security package.

Follow the step-by-step examples in this chapter. If there is something you don't know how to do, refer to the other chapters in this guide for more details on the facility.

Planning Your TCP/IP Network

Whether you are connecting a few computers in a single building or accessing thousands of computers world wide, some planning is needed to configure your TCP/IP network. Before you install and configure TCP/IP, you should choose the networking media and determine the ways of identifying your TCP/IP network (system names, IP addresses, and domain names).

Choosing Networking Media

You can use Ethernet network media with TCP/IP. In addition, you can use serial lines (dedicated serial lines and telephone lines) to reach a remote system.

The installation guide that accompanies your TCP/IP software describes the Ethernet network media drivers that are provided with TCP/IP. The configuration described in this chapter lets you connect to the supported network media. You should also consult your OS Installation Guide and the documentation that accompanies the networking hardware for further information.

Getting an IP Network Number and Domain Name

Before you set up your TCP/IP network, you should have a pool of IP addresses and a domain name assigned to your organization. If you ever expect to communicate on the Internet, as opposed to just connecting a few hosts together locally with no outside connections, you must apply to the Network Information Center (NIC) for an IP network ID and a domain name.

See the chapter “Introduction to Administering TCP/IP Networks” for detailed descriptions of IP addresses and domain names. See the chapter “Using Domain Name Service with TCP/IP” for further information on using domain name service. To obtain an IP network ID and a domain name, you must apply for them from the NIC as described in the chapters “Obtaining IP Addresses” and “Obtaining Domain Names” respectively.

IP addresses

Once you have received your network ID(s) from the NIC, you can create IP addresses for network hosts. You may want to do this on paper or in an ASCII file for reference, before supplying these addresses to the programs and files that use them.

Domain Names and System Names

Once you have registered a domain name with the NIC, you can develop a scheme for naming the hosts and optional subdomains for your network.

A host name is typically the system name set during system installation. To determine the system name (also called the network node name) for an existing system, open the System Status window or type the following from a Terminal window:

```
uname -n
```

You will use the system names later when you set up addressing and use TCP/IP commands, such as **`rlogin`**, **`telnet`**, or **`ftp`**. If you are administering several systems at one location, you may want to set up some naming conventions for the computers. For example, computers at your New York City office may be named `nycbob`, `nycbev`, and `nycmary`.

NOTE

If possible, set up your naming conventions before you install the systems. Because different networking software keys off of the system name, changing the name later might cause some networking software to fail.

Installing TCP/IP

Once you have the domain name, system names, and IP addresses assigned, you can begin to install TCP/IP on each system you are supporting. Check the *Release Notes* for details on installing TCP/IP.

NOTE

Strictly speaking, you don't need a network ID and domain name to install TCP/IP. Domain name service can be set up after your network is up and running. IP addresses, however, are more trouble to change later.

When TCP/IP installation is complete, the following are defined for your system:

- System name (part of basic installation)
- LAN driver (Ethernet)
- IP address (for local system)

NOTE

If you need to add additional LAN drivers or change one after you have installed TCP/IP, you must run the **`configure`** command. See the chapter "Setting Up Routers and Subnetworks" for descriptions of how to use the **`configure`** command.

Creating a Simple TCP/IP Network

After TCP/IP is installed and the hardware is connected, the minimum you need to do to set up TCP/IP is configure the host names and addresses to which you will communicate. Host name and address information is stored in the **`/etc/hosts`** file.

There are two ways to add host names and addresses to your system:

- Editing the **`/etc/hosts`** hosts (as `root`) from a Terminal window.

To edit the **`/etc/hosts`** file, do the following:

1. Open a Terminal window.
2. Type **`su`** then the `root` password to obtain `root` permission.
3. Open the **`/etc/hosts`** file for editing using a standard UNIX editor, such as **`vi`** or **`ed`**.

4. Add the host names and IP addresses for the systems on your network. For each entry, there are three fields (separated by tabs) and an optional comment, as follows:

```
address hostname [alias] # comment
```

   You can uses aliases to define other names by which the host at the address can be accessed.

The following is an example of an **`/etc/hosts`** file:

```
    # Internet host table

127.0.0.1    localhost
#
147.2.101.1      mouse    mus      #Larry Local a-123
147.2.101.2      cat      felis    #Chris Jones b-124
147.2.101.4      dog      canis    #Lisa Someone a-432
147.2.101.5      horse    equus    #Janet Green d-343
```

If you are connected to a single local area network with a limited number of host computers, you can start using TCP/IP. Read the following assumptions about this simple configuration and how to expand beyond its bounds.

This simple configuration assumes:

- All hosts can be reached on your local area network. To reach hosts over serial connections (phone lines or direct connections), see “Setting Up Serial Connections” on page 12-5.
- Only one physical network, with no routing to other networks. To set up a gateway to route packets to other local area networks, see “Setting Up Routing” on page 12-21.
- Each host keeps a list of all host names and IP addresses it can reach. To centrally manage names and addresses for groups of computers, see “Setting Up Domain Name Service” on page 12-22.
- The default TCP/IP services available from your host to remote hosts is acceptable. To see the services available from your system and find out how to change them, see “Changing Available Services” on page 12-23.
- No administration of remote systems is required. To be able to diagnose and fix problems on remote systems, see Chapter 14, "Managing TCP/IP Nodes Using SNMP".
- The default TCP/IP performance is acceptable. To tune your TCP/IP system, see Chapter 16, "Troubleshooting and Tuning TCP/IP".

Expanding Your TCP/IP Network

Only the smallest, most self-contained network configurations will find the procedure described in “Creating a Simple TCP/IP Network” on page 12-3 sufficient. The following sections lead you into more advanced configuration topics.

Setting Up Serial Connections

If you want to reach a remote system that is not connected to your LAN, but is reachable over serial lines (telephone or direct connection), you have two methods from which you can choose: Serial Line IP (SLIP) and Point-to-Point Protocol (PPP).

SLIP lets you use direct serial line connections to connect to remote systems with TCP/IP. PPP allows you to connect to remote systems over telephone lines, direct connections, or any other types of serial connection you can configure using the standard UNIX **`uucp`** facility.

Configuring Point-to-Point Protocol Connections (PPP)

The procedure below describes how to set up a `ppp` connection between two systems: `SystemA` and `SystemB`. `SystemA` is configured to receive incoming `ppp` calls; `SystemB` is configured to make outgoing `ppp` calls. Follow the two procedures to set up `ppp` communications between two systems.

NOTE

If you are using PPP to connect to a SCO TCP/IP, there are several parameters that must be turned on. The parameters are: DLC Address Control Field Compression, IP Address negotiation, Old IP address negotiation (RFC1172), Van Jacobson TCP header compression, and Compatibility with old PPP. You can set these parameters during the **`pppconf`** procedure. The resulting entry in the **`/etc/inet/ppphosts`** file would look like the following:

```
137.65.208.51 - remote1 idle=30 old ipaddr
rfc1172addr VJ accomp
```

where `137.65.208.51` is replaced by your local `ppp` IP address and `remote1` is replaced by the name of the remote `ppp` system as it appears in your **`/etc/uucp/Systems`** file. (The `idle=30` is not required, but is a good idea so your modem connection times out after a set number of minutes of inactivity.)

Configuring SystemA (incoming ppp connection)

To configure a system for incoming `ppp` calls, do the following:

1. Configure an incoming **`uucp`** connection. Any valid **`uucp`** connection is acceptable. In the example below, we used a ZyXEL Hayes-compatible modem configured to accept incoming calls and transfer data at 9600 bps.

   a. Open a Terminal Window.

   b. As `root` user, add the following entries to **`/etc/uucp/Devices`** for the serial port 2 (use `tty00h` for serial port 1):

```
Direct tty01h - 9600 direct_modem
```

   We use the `tty01h`, rather than `tty01`, because `tty01h` lets us use hardware flow control.

   c. Connect the modem to the serial port 2.

   d. Change the **`tty`** settings in your Terminal window to incorporate the following:

```
stty -parenb cs8 9600
```

   (The reason these values are important is that when you **`cu`** directly to the port later, **`cu`** uses the current **`tty`** settings to make the connection. **`cu`** might change your modem settings automatically based on these settings. Note that you may want to add this entry to your **`.profile`** file.)

   e. Type the following from the Terminal window as `root` user to change the settings on the modem:

```
cu -l tty01h
AT S00=001
AT S15=130
AT S20=7
ATQ2
ATQ1
AT&W
~.
```

   These settings may vary for each modem you use. In our case, `AT S00=001` turns on auto answer, `AT S15=130` sets the parity properly, and `AT S20=7` sets the line speed to 9600. Then `ATQ1` and `ATQ2` set the modem to not send certain strings that confuse the login process later. The setting `AT&W` writes the change to the modem and `~.` returns you to the shell. These settings may be lost if you turn off the modem later. To check that the settings are correct, **`cu`** to the modem again and type `AT&V0`.

Read your modem manual to set your settings in a similar fashion! The settings you should watch for include serial port speed locking, hardware or software flow control setup (most modems do not do this by default), DCD, DTR and DSR handshake configuration, transmission mode selection (V32.bis, V42.bis, and so on) result code set, result code suppression (for answering modem), verbose mode, auto answer, character size and parity (for XON/XOFF recognition) and EEprom write command.

f. Change the `9600` entry in the **`/etc/ttydefs`** file to disable parity (**`-parenb`**) and use 8-bit characters (`cs8`) read as follows:

```
9600: 9600 opost onlcr tab3 ignpar ixon ixany
    -parenb istrip echo echoe erase echok isig cs8 cread :
    9600 opost onlcr sane tab3 ignpar ixon ixany -parenb
    istrip echo echoe erase echok isig cs8 cread ::4800
```

g. Configure a process to listen to the serial port 2 by typing the following from a Terminal window as `root` user:

```
/usr/X/adm/pmadm
```

h. Edit the **`_pmtab`** file in **`/etc/saf/ttymon1`**, changing the device to **`/dev/tty01h`** and the word `auto` to read `9600`. (If the `ttymon1` directory is not there, check other directories at that level for the entry shown below.)

```
tty01:u::reserved:reserved:login: /dev/tty01:::/usr/bin/shserv::
    auto:ldterm:login:::: #
```

change to

```
tty01:u::reserved:reserved:login: /dev/tty01h:::/usr/bin/shserv::
    9600:ldterm:login:::: #
```

2. Run the **`pppconf`** command. To do this, open a Terminal window with `root` permissions (type **`su`** and password) and type the following command:

```
/usr/sbin/pppconf
```

The following menu appears:

```
PPP Configuration Menu

1) Configure PPP hosts
2) Configure incoming PPP parameters
3) Configure PPP authentication parameters
q) Quit

Enter choice:
```

3. Type `1`. The following message appears:

```
Enter IP address for local PPP end:
```

4. Type the local system's IP address (for example, `128.212.44.44`). The following message appears:

```
Enter IP address for remote PPP end:
```

5. Type the remote system's IP address (for example, `128.212.44.90`). The following message appears:

```
Enter netmask [default]:
```

6. Press `Enter` to use the current `netmask` for your system (for example, `255.255.255.0`). The following message appears:

```
The UUCP name is used by PPP to dial into the remote system.
If a UUCP name is not specified, outgoing calls will be disabled.
Enter UUCP name ("-" for null):
```

7. Type the **`uucp`** system name for the remote host you set up in Step 1. (In this case, because `ppp` is not making outgoing calls from this host, type a dash ('`-`') to make the entry null.) If you specified a name other than null ('`-`'), the following message appears:

```
Do you want to specify PPP negotiation parameters? (y/n):
```

8. Type `n` to not add any additional parameters. (The parameters you could set at this point only apply to outgoing `ppp` connections.) If you type `n`, the following message appears:

```
Do you want to add an entry for the remote host
    to /etc/hosts? (y/n):
```

9. Type `y` to add the remote host to **`/etc/hosts`**. The following message appears:

```
Enter remote PPP host name:
```

10. Type the name of the remote `ppp` host. (This does not necessarily have to match the **`uucp`** name. Use something like `ppp`_*host1*, where *host1* is replaced by the remote system's system name). The following menu appears:

```
Adding host newhost
to /etc/inet/hosts
Configure PPP Hosts

1) Add/modify PPP hosts
2) Remove PPP hosts
q) Quit
Enter choice:
```

11. Type `q` to quit. The following menu appears:

```
PPP Configuration Menu
1) Configure PPP hosts
2) Configure incoming PPP parameters
3) Configure PPP authentication parameters
q) Quit
Enter choice:
```

12. Type `2` to configure an incoming PPP host. The following menu appears:

```
Configure Incoming PPP Parameters

1) Add/modify incoming PPP setup
2) Remove incoming PPP setup
q) Quit
Enter choice:
```

13. Type `1` to add incoming PPP setup. The following message appears:

```
Enter PPP login name: ppplogin
```

14. Type the login name used by the remote `ppp` host to connect to the local system. (In this case we used `ppplogin` as the login id.) Remember this login name! This name must match the login name used in the remote system's **`Systems`** file for the outgoing host. The following message appears:

```
Do you want to create PPP login account ppplogin? (y/n):
```

15. Type `y` to create a login account for the login name you just entered. The following message appears:

```
Enter a password for login "ppplogin"
New password:
Re-enter new password:
```

16. Type a password (twice) for the login. Remember this password! This password must match the password used in the remote system's **`Systems`** file for the outgoing host. The following message appears:

```
Do you want to specify PPP negotiation parameters? (y/n):
```

17. Type `y` to specify negotiation parameters. The following questions relate to values affecting the incoming connection:

```
PPP connection inactivity timeout in minutes
    ("-" for forever) [forever]:
```

18. Type in the number of minutes a `ppp` connection can be inactive (that is, no packets are sent across it) before the connection it is using is dropped (`idle=idle_time`). The default is for the connection to never time out. The following message appears:

```
Timeout per PPP protocol request in seconds [3]:
```

19. Type the number of seconds after which a configure request or terminate request will time out (`tmout=timeout`). The default is `3` seconds. Increase this number if your serial connection is slow or prone to errors. The following message appears:

```
Maximum number of PPP configure requests [10]:
```

20. Type the number of configure requests to make (without receiving a response from the remote side) before dropping the connection (`conf=num`). The default is `10` times. Increase this number if your serial connection is slow or prone to errors. The following message appears:

```
Maximum number of PPP termination requests [2]:
```

21. Type the number of termination requests to make before dropping the connection (without receiving a response from the remote side) (`term=num`). The default is `2`. The following message appears:

```
Maximum allowable number of remote PPP configure requests [10]:
```

22. Type the number of `configure-nak` retries allowed before dropping the connection. This indicates the number of `configure-nak` packets sent without sending a `configure-ack` before assuming that the configuration negotiation is not converging (`nak=num`). The default is `10`. The following message appears:

```
Maximum receive unit size in bytes [296]:
```

23. Type the maximum size of the units you will receive (`mru=num`). A larger number could send data more efficiently across a reliable network. The default is `296` bytes. The following message appears:

```
Async control character map in hex [00000000]:
```

24. Type the hexadecimal number representing the asynchronous control character map (`accm=num(hex)`). The default is `0x00000000`. PPP can map any ASCII control character (hex `0` to hex `20`) into an appropriate two-character sequence. The `ACCM` field consists of four octets with each bit representing one control character which should be mapped. The least significant bit corresponds to ASCII `0` (NULL) and so on. If a bit is zero, then that control character is not mapped. If a bit is one, then the corresponding control character must be mapped. If you wish to use software flow control, you will need to map `Control-S` (ASCII `19` decimal) and `Control-Q` (ASCII `17` decimal) to avoid confusing data with flow control characters. The resulting value of the `ACCM` field, in this case, is `0x000a0000`. The following message appears:

```
PPP peer authentication timeout in minutes [1]:
```

25. Type the number of minutes the host will wait on authentication requests before timing out (`paptmout=tmout`). The default is `1` minute. The following message appears:

```
PPP password authentication ("on" or "off") [off]:
```

26. Type `on` to turn on PPP password authentication (`pap`). Note that, if you turn password authentication on, you must set up `ppp` authentication from both sides of the connection using the `Configure PPP Authentication Parameters` selection from the `PPP Configuration Menu`. The following message appears:

```
Magic number negotiation ("on" or "off") [on]:
```

27. Type `on` to turn on magic number negotiation, or `off` to not use magic number negotiation (`nomgc`). Magic number negotiation is on by default and is recommended. Magic number negotiation provides a way to detect loop-back (echo) in the link. The following message appears:

```
LCP protocol field compression ("on" or "off") [off]:
```

28. Type `on` to turn on protocol field compressions or `off` to turn it off (`protcomp`). (If on, this will compress the protocol portion of `ppp` frames that are sent.) It is off by default. The following message appears:

```
HDLC address-control field compression ("on" or "off") [off]:
```

29. Type `on` to turn on address-control field compression or `off` to turn it off (`accomp`). (If on, this will compress the HDLC portion of `ppp` frames that are sent.) It is off by default. The following message appears:

```
IP address negotiation ("on" or "off") [off]:
```

30. Type `on` to turn on IP address negotiation or `off` to turn it off (`ipaddr`). (If on, the remote host must support address negotiation.) The default is `off`. The following message appears:

```
Old IP address negotiation (RFC1172) ("on" or "off") [off]:
```

31. Type `on` to turn on IP address negotiation based on RFC1172 or `off` to turn it off (`rfc1172addr`). The default is `off`. The following message appears:

```
Van Jacobson TCP header compression ("on" or "off") [off]:
```

32. Type `on` to turn on Van Jacobson TCP header compression or `off` to turn it off (`VJ`). (If on, TCP performance is improved.) The default is `off`. The following message appears:

```
Compatibility with old PPP ("on" or "off") [off]:
```

33. Type `on` to maintain compatibility with TCP or `off` to maintain compatibility with the current release (`old`). The default is `off`. The following message appears:

```
Enter remote IP address("-" for null):
```

34. Enter the IP address of the remote `ppp` host. If there are multiple incoming PPP connections handled by the local host, you must provide this address or use IP address negotiation (see above). The following menu appears:

```
Configure Incoming PPP Parameters
1) Add/modify incoming PPP setup
2) Remove incoming PPP setup
q) Quit

Enter choice:
```

35. Type `q` to quit. The following menu appears:

```
PPP Configuration Menu

1) Configure PPP hosts
2) Configure incoming PPP parameters
3) Configure PPP authentication parameters
q) Quit

Enter choice:
```

36. If you requested `ppp` authentication to be turned on earlier in this procedure, type `3` and configure `ppp` authentication parameters. Otherwise, type `q` to quit.

37. Reboot your computer.

Configuring SystemB (outgoing ppp connection)

To configure a system for outgoing `ppp` calls, do the following:

1. Configure an outgoing **`uucp`** connection. Any valid **`uucp`** connection is acceptable. In the example below, we used a ZyXEL Hayes-compatible modem configured to accept incoming calls and transfer data at 9600 bps. Many other types of modems can be used. See Chapter 7, "Administering the Basic Networking Utilities" for further information on the Systems, Devices, and Dialers files for configuring **`uucp`** connections.

   a. Open a Terminal Window.

   b. As `root` user, add the following entries to **`/etc/uucp/Devices`** for the serial port 2 (use `tty00h` for serial port 1):

```
Direct tty01h - 9600 direct_modem
ACU tty01h - Any hayes
```

   c. Connect the modem to the serial port 2.

   d. Change the **`stty`** settings in your terminal window to incorporate the following:

```
stty -parenb cs8 9600
```

   (The reason these values are important is that when you **`cu`** directly to the port later, **`cu`** uses the current **`stty`** settings to make the connection. **`cu`** might change your modem settings automatically based on **`stty`** settings.)

e. Type the following from the Terminal window as `root` user for the ZyXEL modem:

```
cu -l tty01
AT S15=130
AT S20=7
ATQ0
AT&W
~.
```

These settings may vary for each modem you use. In our case, `AT S15=130` sets the modem to no parity, then `ATQ0` sets the modem to return the result code. `AT S20=7` sets the line speed to 9600. The setting `AT&W` writes the change to the modem and `~.` returns you to the shell. These settings may be lost if you turn off the modem later. To check that the settings are correct, **`cu`** to the modem again and type `at&v0`.

Make sure you read your modem manual to set your settings in a similar fashion!

f. Add an entry to the **`/etc/uucp/Systems`** file with the following attributes: system name (`SystemA`, the name of the incoming system), Device Type (`ACU`), Speed (`9600`), Phone Number (the phone number of the remote `ppp` host's modem), and Password (password for remote `ppp` user). An example appears below:

```
systema Any ACU 9600 555-1212 "" \r\d "" \r\d in:
    --in: ppplogin word: abc opyrigh
```

(Notice the letters `opyrigh` at the end of the `chat` script. After the outgoing side has logged into the remote system with `ppplogin` and a password of `abc`, it waits for the letters `opyright` to appear before `ppp` takes over the connection. The incoming system prints a `Copyright` message which matches the `opyrigh` the outgoing system is waiting for. This prevents the `ppp` connection from timing out before a successful underlying connection is ready.)

g. If the system only makes outgoing connections on this port, make sure there is no **`ttymon`** running on the local port. (To do this, as `root` user, type **`pmadm -l`** to see which port monitors are active.)

2. From a Terminal window as `root` user, type **`pppconf`**. The following menu appears:

```
Configure PPP Hosts
1) Add/modify PPP hosts
2) Remove PPP hosts
q) Quit
Enter choice:
```

3. Type `1`. The following message appears:

```
Enter IP address for local PPP end:
```

4. Type the local system's IP address (for example, `128.212.44.90`). The following message appears:

```
Enter IP address for remote PPP end:
```

5. Type the remote system's IP address (for example, `128.212.44.44`). The following message appears:

```
Enter netmask [default]:
```

6. Press `Enter` to use the current `netmask` for your system or enter another `netmask` for the network number being used by the `ppp` interface. The following message appears:

```
The UUCP name is used by PPP to dial into the remote system.
If a UUCP name is not specified, outgoing calls will be disabled.
Enter UUCP name ("-" for null):
```

7. Type the **`uucp`** system name for the remote host you set up in Step 1 (`systema` in this case). The following message appears:

```
Do you want to specify PPP negotiation parameters? (y/n):
```

8. Type `y` to add PPP negotiation parameters or `n` to not add any additional parameters. If you type `y`, the parameters you can set are described in “Configuring SystemA (incoming ppp connection)” on page 12-6. If you type `n`, the following message appears:

```
Do you want to add an entry for the remote host
    to /etc/hosts? (y/n):
```

9. Type `y` to add the remote host to **`/etc/hosts`**. The following message appears:

```
Enter remote PPP host name:
```

Type the name of the remote `ppp` host. (This does not necessarily have to match the **`uucp`** name.) The following menu appears:

```
Adding host newhost to /etc/inet/hosts
Configure PPP Hosts

1) Add/modify PPP hosts
2) Remove PPP hosts
q) Quit
Enter choice:
```

10. Type `q` to quit.
11. Reboot your computer.
12. Before you try using this port with TCP/IP utilities (such as **`ping`** or **`rlogin`**), make sure you can use uucp to talk across the port. Type **`cu systema`**, where systema is the name of the remote `ppp` host you added in your **`Systems`** file. If you can successfully login to the remote system, type **`~.`** to return to the shell.
13. Now type:

```
ping -s systema
```

The modem should dial the remote system and **`ping`** should indicate that the remote system is alive.

For information on configuring PPP parameters, see the **`ppphosts(4)`** manual page.

Configuring Serial Line IP Connections (SLIP)

The SLIP interface lets you connect directly to a remote host from a serial port on your computer. No **`uucp`** configuration is needed to use SLIP.

Enabling SLIP Connections

To create a direct connection over a serial line for use by TCP/IP, do the following:

1. Make sure there are no port monitors (that is, **`ttymon`**) running on the SLIP port. (To do this, as `root`, type **`pmadm -l`** and look for the device name to see if there is an active port monitor on the device.)
2. Create a physical connection from the serial port on your system to one on a remote system.
3. Add the remote host name and IP address to your local **`/etc/hosts`** file. Because SLIP is a separate network interface, you should also add a new host name and IP address for your local computer to **`/etc/hosts`** that is on the same subnet as the remote host.
4. Open a Terminal window with `root` permissions (type **`su`** and password), and enter the **`/usr/sbin/slattach`** command. The following is an example of the **`slattach`** command:

```
/usr/sbin/slattach tty01 192.9.200.1 192.9.200.5 9600
```

In this example, TCP/IP serial line communication is enabled to the remote system connected to your serial port 2 (`tty01`). The local IP address is `192.9.200.1` and the remote IP address is `192.9.200.5`. The baud rate on the line is set at `9600`.

For information on configuring SLIP parameters, see the **`slattach(1M)`** manual page.

Disabling SLIP Connections

To disable a SLIP connection, do the following:

1. Open a Terminal window with `root` permissions (type **`su`** and password).

2. Kill the running **`slattach`** process.
3. Manually remove the routing information associated with the SLIP connection using the following command:

```
/usr/sbin/route delete 192.9.200.5 192.9.200.1
```

In this example, `192.9.200.5` is the IP address of the remote host and `192.9.200` is the local IP address as your computer is known to the SLIP interface.

Setting Up Routing

As your network expands, you will want to reach hosts that are connected to other networks. You can connect your LAN to another LAN by designating one of your systems as a gateway to do routing. Likewise, if a computer on your LAN has a connection to a remote SLIP or PPP host, you can use that computer to route packets to the remote host from other hosts on your LAN.

In a simple case, you may have two local area networks in your building. You designate one system as a router. You must use the procedure described below to set up the gateway system and all client systems as follows:

1. On the gateway system do the following:
   a. Install and configure two network interface cards on the system: one connecting to the first network and one to the second. (See the command **`/etc/confnet.d/configure`** and the **`/etc/confnet.d/inet/interface`** file.)
   b. Assign two names and network addresses to the routing system. Assign different names for each network. (See the **`/etc/hosts`** file.)
   c. Configure the router to route packets between the two networks. (See the daemon **`in.routed`**, as well as the **`networks`** and **`gateways`** files, and Chapter 13, "Setting Up Routers and Subnetworks".)
2. On the client system, configure the router daemon so packets for systems not on the local network are directed to the router. Or, instead, you could hard code the location of the router.

Complete descriptions and procedures for setting up a router are contained in Chapter 13, "Setting Up Routers and Subnetworks".

Setting Up Domain Name Service

If you want to communicate with other hosts on the Internet, you must set up domain name service (DNS). Even if you are not connecting to the Internet, in an organization where systems are constantly being added and changed, it is often inconvenient for every computer to have its own complete list of system names.

By setting up domain name service, you centralize the administration of system names and addresses. When a system tries to communicate with another system, it checks the name server for the remote system's address.

Once you have obtained a domain name from the NIC for your network and figured out a scheme for using subdomains, use the procedure described below to set up domain name service for your domain:

1. On the client systems do the following:
   a. Configure the addresses for each domain name server the client can reach. (See the **`/etc/resolv.conf`** file and the **`resolv.conf(4)`** manual page.)
   b. Make sure all transport providers that provide IP connections have DNS name-to-address routing defined. (See the **`/etc/netconfig`** file and the **`netconfig(4)`** manual page.)
2. On the domain name server systems do the following:
   a. Configure domain name server addresses and name-to-address resolving, as you would for a client system.
   b. Configure the boot and data files for the name service daemon (see the **`named(1M)`** manual page).
   c. Start up the name service daemon (**`in.named`**) in the system's Internet start-up script (**`/etc/inet/rc.inet`**). If **`named.boot`** is configured, **`in.named`** will start automatically.

Complete descriptions and procedures for setting up domain name service are contained in Chapter 15, "Using Domain Name Service with TCP/IP". Descriptions of domains and subdomains are contained in Chapter 11, "Introduction to Administering TCP/IP Networks".

Setting Up Subnetworks

To be able to join together several physical networks together under one network ID, you can divide up your assigned pool of IP addresses into subnets. See Chapter 11, "Introduction to Administering TCP/IP Networks" for descriptions of IP addresses. See Chapter 13, "Setting Up Routers and Subnetworks" for descriptions of subnetwork configuration.

Changing Available Services

A standard set of services is available on your TCP/IP system when it is installed. As requests come into your system on particular TCP/IP or UDP ports, your system takes the requests and responds by fulfilling or rejecting the service requests.

TCP/IP services your system knows about are contained in the **`/etc/inet/services`** file. The daemon processes that handle requests for those services are listed in the file **`/etc/inet/inetd.conf`**.

Examples of services provided by TCP/IP that can be turned off and on include the following commands: **`ftp`**, **`telnet`**, and **`named`**. Some services, such as **`ntp`** and **`snmp`**, are turned on automatically if configuration files are set up for these services. Some services are not turned on by default.

The following procedure shows how you would allow a remote system to run the **`finger`** command to check on users on your system:

1. Open the **`/etc/inet/inetd.conf`** file for editing as `root` user.
2. Remove the comment character (`#') from in front of the line containing the **`fingerd`** daemon, so it appears as follows:

```
finger stream tcp nowait nobody /usr/sbin/in.fingerd in.fingerd
```

3. Save and exit the file.
4. Type the following from the command line, as `root` user, to obtain the process number of the **`inetd`** daemon:

```
# ps -ef | grep inetd
```

5. Using the process id of the **`inetd`** daemon, type the following command:

```
# kill -HUP pid
```

   where *pid* is replaced by the process ID of the running **`inetd`** daemon.

A remote user could now run **`finger`** to get information about users on your system. You don't have to reboot your system. (Note that **`fingerd`**, as well as other services you allow, may pose some security risks.)

Managing Remote Systems

Using Simple Network Management Protocol (SNMP), you can monitor and control other TCP/IP systems from your system. Information on configuring and using SNMP is contained in Chapter 14, "Managing TCP/IP Nodes Using SNMP".

Synchronizing Time

Time clocks on your computers will drift apart over time. To ensure that time is synchronized on the computers on your network, follow procedures described in Chapter 19, "Network Time Synchronization".

Maintaining Security

The OS includes various user programs that enable network users to log in to remote systems (**`rlogin`**), create and use a shell on a remote machine (**`rsh`**), and copy files to and from a remote machine (**`rcp`**).

For a remote user to use the above commands to access your system, he or she must have a user account on your system. Although remote users can log in to your machine once they have entries in your password database, they cannot run certain remote processes (such as **`rsh`** and **`rcp`**) unless their machines are listed in the **`/etc/hosts.equiv`** file, or the user's **`.rhosts`** file, located in the user's home directory. In addition to allowing remote users to run certain remote processes on your machine, these files grant users access to your machine without their having to supply passwords.

This section explains how your system uses the **`hosts.equiv`** and **`.rhosts`** files, then gives you instructions for setting up and maintaining these files.

Administering the hosts.equiv and .rhosts Files

On a TCP/IP-based network, security is implemented at two levels: first, at the host level, and second, at the user level.

The **`/etc/hosts.equiv`** file is a general database that controls access at the host level. The **`.rhosts`** file controls access to your machine at the user level. It is a file located in the home directory of a specific remote user on your machine, and it is used to allow or deny access to that specific user. There may be multiple instances of **`.rhosts`** on your machine, one for each remote user with a home directory.

When a remote user attempts to log in to your machine, the security-checking process goes like this:

- A remote user initiates an **`rlogin`**, and the **`rlogin`** daemon on your machine checks for the user's user name in the password database. If no entry is found, the remote user is denied access.

If the remote user has an entry in your password database, the daemon next checks for the remote machine's hostname in your **/etc/hosts.equiv** file. If the hostname is found, the remote user gains access.

- If no **/etc/hosts.equiv** entry is found, the system checks for a line with the remote machine's hostname (and, optionally, the remote user's user name) in the **.rhosts** file in the user's home directory on your machine. If the entry is found, the remote user gains access.
- If no entry is found for the remote machine in either your **/etc/hosts.equiv** file or the remote user's **$HOME/.rhosts** file on your machine, the remote user can **rlogin** to the machine after giving the correct password. However, the remote user will receive the message `Permission denied` when attempting to run remote processes like **rcp** or **rsh**.

By creating and maintaining a **/etc/hosts.equiv** file, you can allow everyone on a particular host to log in to your machine and run remote processes like **rcp** and **rsh**.

If you want certain users on a particular host to access your machine, but not everyone, do not include the host in your **/etc/hosts.equiv** file. Instead put the host's name in the **.rhosts** file in each remote user's home directory on your machine.

The following sections tells you how to set up the **/etc/hosts.equiv** and **.rhosts** files.

The /etc/hosts.equiv File

In the simplest case, the **/etc/hosts.equiv** file is a list of machines, or hosts, from which users are permitted to log in (using **rlogin**) without supplying a password **/etc/hosts.equiv** is a local file, pertaining only to the system on which it is found. The machine's system administrator can modify **/etc/hosts.equiv** by logging in as a privileged user, and using a text editor to make changes.

A typical **/etc/hosts.equiv** file has the following structure:

```
host1
host2
```

If a user attempting to log in from a remote host listed in **/etc/hosts.equiv** has a valid user account, he or she will be granted access without having to enter a password. If the same user attempts to log in from a host that is not listed in **/etc/hosts.equiv** or in **$HOME/.rhosts**, he or she will be granted access only after entering a correct password.

A single '**+**' on a line in a machine's **/etc/hosts.equiv** file means that all known hosts (that is, all hosts listed in the `hosts` database) are trusted.

Example

Suppose the **`/etc/hosts.equiv`** file on your machine looks like this (note that the file is just a list of host names, one per line):

```
raks
dancers
jazz
```

Now you want to allow anyone on host `ballet` to have access to your machine. To do this, edit the file **`/etc/hosts.equiv`** and add `ballet` to the list, as follows:

```
raks
dancers
jazz
ballet
```

Now, add all the users on `ballet` to your machine using the User Setup window. After you complete the add, all users who can log in to `ballet` can also freely **`rlogin`** to your machine, without having to supply a password. Furthermore, users on `ballet` can remote copy from your machine and use a remote shell on your machine.

Refer to the **`hosts.equiv(4)`** manual page for more information.

The .rhosts File

The .rhosts file is located in a specific remote user's home directory on your machine; it is used to allow or deny access to that specific user.

The format of the **`.rhosts`** file is similar to that of **`/etc/hosts.equiv`**, that is:

```
host1
host2
```

The above entries in a user's **`.rhosts`** file mean that the user can log in from `host1` or `host2`, without supplying a password.

A user can allow others to log in to the remote machine using his or her login by adding user names to the **`.rhosts`** file.

For example, user `steve` has an **`.rhosts`** file on your machine that looks like this:

```
host1
host2
host1 jane
```

This means that `steve` can log in remotely to your machine as himself from `host1` and `host2`, and user `jane` can log in to your machine as `steve` from `host1`.

If your site does not require stringent security measures, the easiest way to administer machine access at the user level is to give each user an account in the password database and a home directory on your machine(s). Then ask the trusted users to create their own **`.rhosts`** files in their home directories on the machine(s).

Example 1

You want user `chris` to have access to your machine without restrictions. To do this, have `chris` create the file **`.rhosts`** in her home directory on your machine. If `chris` wants other users to have access to her login on your machine, her **`.rhosts`** file might look like this:

```
adams
monroe
jackson
adams bob
jackson jenny
```

Now user `chris` can access your machine remotely from `adams`, `monroe`, and `jackson`; user `bob` can access your machine as `chris` from `adams`; and user `jenny` has access as `chris` from `jackson`.

Example 2

Suppose you want only user `chris` to have access to your machine from host `samba`. To set up your machine, you would follow these steps:

1. Make sure `samba` is not in your **`/etc/hosts.equiv`** file.
2. Create the file **`.rhosts`** in `chris`'s home directory on your machine. The file should look like this:

```
samba
```

Security Issues

The only way to achieve anything resembling security in a sensitive environment is to exclude users from your password database; once someone knows a password, he or she can access your machine.

Some TCP/IP user services are especially problematic in an environment that requires strict security. For example, the **`finger`** command allows a user to display users on a remote host. The command starts the **`finger`** daemon (**`fingerd`**) on the remote machine, which then reports to the user the user name and full name of everyone who is logged in to the machine. There is no authentication of the requestor and no auditing of the requests. Similarly, the **`rwhod`** daemon (on whose information the **`ruptime`** command bases its reports) freely passes around information about who is logged in to a machine.

There are no files or databases to restrict access to the information provided by these daemons; therefore, by default, they are not run. If you do not require strict security, you can run **`fingerd`** by deleting the comment character from the line that starts **`fingerd`** in the file **`/etc/inet/inetd.conf`**. To start **`rwhod`**, add the line:

```
/usr/sbin/in.rwhod
```

to the end of the start-up script **`/etc/inet/rc.inet`**.

Disabling the TCP/IP Network Services

You may want to use an application over the network and disable the TCP/IP network service commands. System administrators can configure TCP/IP solely as a transport provider and disable TCP/IP services including **`telnet`**, **`rsh`**, **`rlogin`**, and **`rcp`**.

This would allow other networking services (such as RPC and NFS) to continue to work over TCP/IP. Use the **`inet.priv`** shell script to disable or restore TCP/IP services as shown in the following syntax:

To disable TCP/IP services, type:

```
/sbin/sh /etc/inet/inet.priv -d
```

To enable TCP/IP services, type:

```
/sbin/sh /etc/inet/inet.priv -e
```

Once disabled, TCP/IP service commands will remain disabled even after system reboots. The enable option must be run in System Maintenance Mode with privilege. The shell script must be started by the appropriate user ID with privilege.

TCP/IP Services and Enhanced Security Issues

Using TCP/IP network services with the Enhanced Security package produces a potential security risk by exposing systems to more points of entry. Although running Mandatory Access Control (MAC) over your network will not affect the TCP/IP services, MAC itself cannot restrict the access of users logging in to server machines with TCP/IP service commands. For more information on MAC, see the chapters in the “Security Administration” part of the *System Administration.* guide. For securely authenticated network commands, see the chapter “Remote Services Tutorial” in the *User's Guide.*

The purpose of this section is to point out the effect of using TCP/IP services over networks containing the Enhanced Security package, and to recommend steps for minimizing the security risk.

Minimizing TCP/IP Security Risks

The TCP/IP daemons minimize potential access security risks by stripping process privileges before invoking remote requests. Stripping process privileges occurs regardless of the security level at which you logged in. Even if you are a system administrator logged in at the `SYS_PRIVATE` (remote login) level, your remote login request (**`rlogin`** or **`telnet`**) will produce a shell without privileges.

TCP/IP and Secure Device Handling

The Enhanced Security package allocates devices for the TCP/IP daemons. The TCP/IP devices are initialized to a private state and a user-accessible security level.

NOTE

Changing the default device allocation attributes may affect the functionality of TCP/IP services. We recommend that system administrators not change the device allocation. Any changes may conflict with the ability of the TCP/IP services to use the secure devices.

SYS_PRIVATE and USER_LOGIN Security Levels

When used with the enhanced security package, TCP/IP network services react the same as other network services by permitting you to log in without privileges, even if you log in as a privileged user such as `root`. A system administrator can log on to server machines at the `SYS_PRIVATE` or `USER_LOGIN` security level. The level at which administrators login will dictate the TCP/IP services they can execute and the files on which the commands will work. If you are a system administrator logged on to server machines at the `SYS_PRIVATE` level, you can not use TCP/IP services to access files at the `USER_LOGIN` level, even when performing administrative tasks. Conversely, logging on at the `USER_LOGIN` level will let you use certain TCP/IP services, but they will fail if executed on `SYS_PRIVATE` level files.

Most of the TCP/IP administrative database files must be set for use at the `SYS_PUBLIC` level. The exception is the user's **`$HOME/.rhosts`** file, which operates at the `USER_LOGIN` level.

If you need to access multiple levels, we recommend that you use the network services provided by the Basic Networking Utilities (BNU), REXEC, Remote File Sharing, or the Connection Server. For more information on remote access at several levels, see Chapter 1, "Introduction to Network Services Administration". The following sections list the files contained in the `SYS_PRIVATE` and `USER_LOGIN` levels.

Setting Up the Network Administrator's Login

If you are running the Enhanced Security Utilities, which is provided as part of the Enhanced Security add-on package, the concept of the superuser has been replaced by administrative roles. Each administrative role is associated with commands and privileges required to administer a certain aspect of the system. To administer network services, you must assume the role of the network administrator `NET`. The `NET` role is predefined on the system, but before you can log in as a network administrator, you must create a login and assign it the `NET` role. This is done using the **`adminuser`** command.

For information about assigning roles, see the chapter “Trusted Facility Management” in the *System Administration* guide or **`adminuser(1M)`**.

SYS_PRIVATE Level Executables

The following executables depend on inheritable privileges and must be run at the `SYS_PRIVATE` security level from a privileged process. To gain privilege to run a command, your user ID must be in the `NET` role. Use the **`adminuser(1M)`** command to add users to the `NET` role. Use the **`tfadmin(1M)`** command to acquire privilege.

To remotely execute these binaries (with privilege), we recommend that you invoke them using the Basic Networking Utilities (BNU) network services or the REXEC service,

rexec. Other options are to use network services administered by the Connection Server. The SYS_PRIVATE-level executables are listed below.

```
arp          gettable     htable
ifconfig     inetd        in.named
in.routed    in.rwhod     route
slink        trpt
```

USER_LOGIN Level Executables

TCP/IP accesses the following operations only at the USER_LOGIN level. Note that files such as **/etc/hosts.equiv** and **$HOME/.rhosts** also must be readable by USER_LOGIN.

```
finger  ftp         netstat
rcp     rdate       rlogin
rsh     ruptimer    who
talk    telnet      whois
```

SYS_PUBLIC-level Files

The following TCP/IP set-up files are located at the SYS_PUBLIC level. An administrator logged in at any other security level must have the appropriate privileges to edit these files.

```
/etc/hosts   /etc/hosts.equiv    /etc/networks
/etc/ethers  /etc/protocols      /etc/services
```

Since these files can be written to only by root, members of the NET role may use **tfadmin cp** to copy in updated versions of these files.

NOTE

One exception to these set-up files is **$HOME/.rhosts**, which is set at the USER_LOGIN level. An administrator must be logged in at USER_LOGIN to edit this file.

13
Setting Up Routers and Subnetworks

13
Setting Up Routers and Subnetworks

Introduction to Router and Subnetwork Administration

Routing and subnetworking are important features as your organization's networking needs grow. Subnetworking lets you divide up your pool of IP addresses so you can split those addresses across several physical networks within your organization. Routing provides a means of passing information from one physical network to another. In this chapter, we describe how to set up a system to route packets from one network to another, as well as how to assign addressing and network masking to do subnetworking.

Configuring Routing

A TCP/IP network usually interconnects a number of hosts. Your host is connected to a TCP/IP network via a hardware network interface. Individual TCP/IP networks are in turn interconnected via IP routers. IP routers forward IP packets from one TCP/IP network to another, and exchange routing information with each other to deliver packets across a number of networks. Other types of routers may forward traffic for protocol families other than TCP/IP.

NOTE

> We refer to the term “router” throughout this chapter. Though technically a router refers to a machine that does nothing but routing, we are usually referring to a system that does routing among its other activities. Therefore, we sometimes refer to the system as a gateway system.

If all of the hosts at your site are connected to a single TCP/IP network that is not interconnected with any other TCP/IP networks, an IP router is unnecessary. If your site comprises many TCP/IP networks, or if you want to interconnect your network with other TCP/IP networks, you must configure the interconnections with IP routers in order for all hosts to communicate.

Many types of machines may serve as IP routers. A number of vendors offer machines dedicated entirely to the function of IP routing. A system may act both as a host (offering network services such as remote login) and a router.

Using the routed Daemon

The **`routed`** (routing daemon) program implements a standard routing protocol. If a system has only one network interface, **`routed`** will passively monitor the routing traffic (if that network is a broadcast network). If a system has two or more network interfaces and is set up to be a gateway, the **`routed`** program will actively participate in the exchange of routing information with other routers.

For more information about **`routed`**, see **`routed(1M)`** and “Setting up the Route Daemon” on page 13-12.

Preparing for Routing

Before actually configuring your system to do routing, make the following preparations.

- Assign the router a unique host name and a unique IP address for each network it is on. The Internet Protocol architecture requires each interface to have a unique IP address.
- Make sure you have acquired registered IP network numbers for each network the router is to connect. (This may require subnetworking within your existing network.)
- If you have not installed the TCP/IP Internet package, install it following the instructions in the *Release Notes*.

Once you have TCP/IP installed, do the following from a Terminal window as the `root` user:

1. Access **`/etc/networks`** and add the network names and network numbers of all the networks that the router can reach.
2. If you have more than one network card (device) installed in your machine, you must run **`/etc/confnet.d/configure -i`** [see generic **`configure(1M)]`**. (**`/etc/confnet.d/configure`** will be referred to as **`configure`** in the following explanation).

   When you run **`configure -i`**, you are interactively configuring the device(s) in your machine. You can run the **`configure`** script once for each device that is installed, or you can specify the number(s) of the device(s), separated by white space, when you are asked which devices(s) you want to configure/reconfigure. Before you run configure, your network cards should already be installed and set up.

   The **`configure`** script runs a protocol-specific **`configure`** script (for example, **`inet`**, [see INET-specific **`configure(1M)]`**. The protocol-specific **`configure`** scripts are not meant to be run by a user.

   An entry in **`/etc/hosts`** should be added for each network card that is installed in your machine before you run **`configure -i`**. If an entry does not exist for a network card in **`/etc/hosts`**, you will be prompted to enter a valid name and IP address for the device that is being configured.

If you populate **`/etc/hosts`**, one entry should be for the actual name of your machine, along with a valid IP address. Use **`uname -n`** to display the name of your machine. The additional entries (for multiple network cards) should be variations of the name displayed by **`uname -n`**. The entry for each network card installed must have both a unique name and unique IP address.

The following explanations and examples will be for the **`inet`** protocol suite. For the first device, the configure script uses the IP address from **`/etc/hosts`** that corresponds to the system name displayed by **`uname -n`**. For the second device, the configure script defaults to the IP address that corresponds to the system name displayed by **`uname -n`**, followed by a `2`. For example, if the name of your machine is `hulk` and you have two network cards installed, the first device would default to the IP address for `hulk`, and the second device would default to the IP address for `hulk2`. You have the ability to change the defaults when you run **`configure -i`**. The address for each of these two system names must include the network number associated with that network. You will then be asked to select the **`ifconfig`** options for the network card that is being configured. The valid choices are shown in Table 13-1:

Table 13-1. Choices for Setting the ifconfig Options with configure -i

Choice	Description of **`ifconfig`** options
yes	`arp` (This option enables the use of the Address Resolution Protocol to map between IP addresses and link level addresses.)
no	Enter no options or customize the **`ifconfig`** options. (This option allows you to customize the network interface.)
ClassC	`netmask 0xffffff00 broadcast` *network_address.255* `arp` (You would use this option if you have a Class B address and are netmasking to use the third byte of the IP addresses as the subnet ID.)
BerkeleyC	`netmask 0xffffff00 broadcast` *network_address.0* `arp` (Same as Class C, except the broadcast address has `0` rather than `255` in the final field. Only use this for compatibility with certain Berkeley UNIX systems.)
info	Prints an informational message about common **`ifconfig`** options.

After you choose the **`ifconfig`** option(s), the INET-specific **`interface`** file is updated with the information for the device being configured and the **`netdrivers`** file is updated by the **`netinfo`** command. See **`netdrivers(4)`** for information on the **`/etc/confnet.d/netdrivers`** file. See **`netinfo(1M)`** for information on **`netinfo`**.

When a second (or subsequent) device is being configured, you are asked if you want to set up the machine as a gateway (this enables the machine to forward packets between the two networks). The `IPFORWARDING` variable will be updated in the **`/etc/conf/cf.d/stune`** file. The valid choices are shown in Table 13-2.

Table 13-2. Choices for Setting up a Gateway Machine with configure -i

Choice	**`configure -i`** options
yes	If `IPFORWARDING=0`, change it to `1`, set kernel to be rebuilt on next reboot. If `IPFORWARDING=1`, make no changes. (This option lets your system act as a gateway for other hosts on your network.)
no	If `IPFORWARDING=1`, change it to `0`, set kernel to be rebuilt on next reboot. If `IPFORWARDING=0`, make no changes. (This option prevents other hosts from routing to another network through your system.)
unchanged	make no changes

Once you have selected the option you want, the INET-specific **`configure`** script will finish setting up the specified network card(s).

An example run of the generic **`configure -i`** script, and the resulting files, is shown in “Configuring Multiple Network Cards” on page 13-6.

3. If there are interfaces that you want certain network applications to ignore, configure the **`/etc/inet/if.ignore`** file. The file consists of a list of interfaces plus lists of associated daemon processes that should ignore those interfaces. The following is an example of an **`if.ignore`** file that lists daemons that will ignore any messages from the associated interfaces:

```
ppp0     rwhod timed
wd0      routed
wd1      routed timed rwhod
```

4. Change the route daemon to active mode by deleting the **`-q`** from the line:

```
/usr/sbin/in.routed -q
```

   in the **`/etc/inet/rc.inet`** file, so the line reads:

```
/usr/sbin/in.routed
```

   In the first case (**`-q`**) routing information is not broadcast. In the second case (without a **`-q`**), routing information is broadcast if `IPFORWARDING` is turned on.

5. Once you have your router set up, add the host name, IP address, and network number of the router to the **`/etc/hosts`** and **`/etc/networks`** files on each machine on your local network.
6. If you specified that your machine should be set up as a gateway, and the `IPFORWARDING` variable was modified by **`configure`**, you will see a message stating that the UNIX kernel will be rebuilt the next time you reboot your machine. If you see this message, you must reboot your machine after you are done configuring the network cards for the changes to take effect. To reboot and rebuild the UNIX kernel on your machine (if needed), either double-click on the Shutdown icon, or, as `root` user, type the following:

```
cd / ;
shutdown -i6 -g0 -y
```

Your UNIX kernel will be rebuilt, and your machine will be rebooted.

Options for the Generic configure Command

The generic **`configure`** command has the following syntax:

`configure [`**`-i`**`] [`**`-p`** *protocol* **`-d`** *device* `[`**`-d`** *device*`...]]`
`configure [`**`-p`** *protocol* **`-d`** *device*`] [`**`-O`** `"`*protocol specific opts*`"]`
`configure [`**`-r -d`** *device*`] [`**`-r -p`** *protocol*`] [`**`-r -p`** *protocol* **`-d`** *device*`]`

The **`-i`** option runs **`configure`** in the interactive mode. You can specify the protocol and device(s) that you want to configure at this time. If you don't use the **`-i`** option, you must specify all of the protocol-specific options with the **`-O`** option, enclosed in a pair of double-quotes (“ “). See the protocol-specific **`configure`** manual page for the options needed by the protocol-specific **`configure`** command. For example, to find out the protocol-specific options for **`inet`**, you would need to see INET-specific **`configure(1M)`**.

The **`-r`** option allows you to remove a specified device, a specified protocol, or both. This command would be used if you were going to remove a protocol and/or device from the machine.

See generic **`configure(1M)`** for information on the **`configure`** command.

Configuring Multiple Network Cards

This example of **`configure -i`** will be for two network cards (devices), using the **`inet`** protocol, with the machine to be set up as a gateway. The name of the machine is `hulk`. The following are the contents of the **`/etc/confnet.d/netdrivers`** file:

```
el16_0
wd_0
```

The above example shows that there are two network cards installed, and that they are not mapped to any protocol.

The following example shows the active entries in the unmodified INET-specific **`interface`** file.

```
lo:0:localhost:/dev/loop::add_loop:
```

The only entry in the **`/etc/confnet.d/inet/interface`** file is for `lo:0` (`lo0/localhost`). `lo0/localhost` is the loopback driver; it can be used as an argument to network applications without using network hardware.

After the generic **`configure -i`** is finished, entries are added or updated.

Configuring the First Network Device

Run the generic **`configure`** script in interactive mode by typing:

```
/etc/confnet.d/configure -i
```

The following text is the question and answer session from the generic **`configure`** and the INET-specific **`configure`** scripts, along with the resulting **`netdrivers`** and INET-specific **`interface`** files:

```
These are the device(s) available on your system:
1 el16_0
2 wd_0
Type the number of the device(s) you wish to
configure with inet [?,??,q]:1
```

If you enter `?,` the help message will be displayed. If you enter `??`, the list of available devices will be displayed. If you enter `q`, the **`configure`** script will either display the next available protocol (if any are installed) or terminate execution. At this point, you can enter the number of one device, or both devices, separated by white space. (**`configure -i`** was executed for each device in this example.) For this example, we specify the first (`el16_0`) device (by entering the value `1`).

You will be prompted to enter the IP host name for `el16_0`, as shown below:

```
Please enter the IP host name for device el16_0
(default is hulk):
```

If you press `Return` the default name will be used. The default name is generated by the **`uname -n`** command. If you want to use a name other than the default, you can enter it now.

For this example, we used the default name. You are then asked for the IP address for the IP host name that was just entered. The **`/etc/hosts`** file is searched for the IP address for the IP host name that was specified. If a match for the specified name is found, it becomes the default entry. If a match is not found, you will be prompted to enter a valid IP address with the following message:

```
Please initialize the IP address for host hulk
(default: 174.2.110.56):
```

If you press `Return` the default IP address will be used. If you want to use an IP address other than the default, you can enter it now. If you change the IP address at this time, the change will also be made to the **`/etc/hosts`** file.

For this example, we used the default IP address. You are then asked for the **`ifconfig`** options for the device that is being configured:

```
Configure host hulk with default Ethernet ifconfig options?
(yes no ClassC BerkeleyC info;
default: info): yes
```

For this example, we answered `yes`. At this point, the INET-specific **`interface`** file and the **`netdrivers`** file is updated to show that the `el16_0` board is mapped to the **`inet`** protocol. The partially configured **`interface`** file and **`netdrivers`** file (for the first network card) are shown below.

The partially configured **`/etc/confnet.d/inet/interface`** file contains the following:

```
lo:0:localhost:/dev/loop::add_loop:
el16:0::/dev/el16_0:arp::
```

The partially configured **`/etc/confnet.d/netdrivers`** file contains the following:

```
el16_0 inet
wd_0
```

Configuring the Next Network Device

Now, type the following:

```
sh /etc/confnet.d/configure -i
```

a second time to configure the second network card (device). The following message appears.

```
These are the device(s) available on your system:
 1 el16_0
 2 wd_0
Type the number of the device(s) you wish to
configure with inet [?,??,q]:2
```

Note that the first network card (`el16_0`) still appears in the list of available network cards. This allows you to reconfigure an existing network card. For this example, we will specify the second (`wd_0`) device (by entering the value `2`).

You will then be prompted to enter the IP host name for `wd_0` with the following message:

```
Please enter the IP host name for device wd_0
(default: hulk2):
```

If you press `Return` the default name will be used. The default name is generated by the **`uname -n`** command, followed by a `2`, since this is the second device that is being configured. If you want to use a name other than the default, you can enter it now.

For this example, we used the default name. You will then be asked for the IP address for the IP host name that you just specified. The **`/etc/hosts`** file will be searched for the IP address for the IP host name that was specified.

If an IP address for the specified name is found, it will become the default entry. If a match is not found, you will be prompted to enter the IP address with the following message:

```
Please initialize the IP address for host hulk2
(default: 174.2.110.124):
```

If you press Return the default IP address will be used. If you want to use an IP address other than the default, you can enter it now. If you change the IP address at this time, the change will also be made to the **/etc/hosts** file.

For this example, we used the default IP address. You will then be asked for the **ifconfig** options for the network card that is being configured:

```
Configure host hulk with default Ethernet ifconfig options?
(yes no ClassC BerkeleyC info;
default: info): yes
```

We answered yes. At this point, the INET-specific **interface** file and the **netdrivers** file are updated to show that the wd_0 board is mapped to the **inet** protocol. The following message appears:

```
lo:0:localhost:/dev/loop::add_loop:
el16:0::/dev/el16_0:arp::
wd:0:hulk2:/dev/wd_0:arp::
```

The fields in the **interface** file are separated by colons (":"). The first field is the identifier for the driver's **netstat** statistics [see **netstat(1M)]**. The second field contains the index number for that device (the value for the first device is 0, for the second device 1, with the maximum value of 9). The third field contains either the IP host name or IP address for the device. If this field is null, the value is expanded to the system nodename. There should only be one "null" address per **interface** file. The fourth field is the device name (as it appears in the **/dev** directory) for the transport provider that is being used. The fifth field is used for customized **ifconfig** options. This field may be null, but it usually contains **arp** for Ethernet devices. The sixth field should not be modified by the user. It specifies the procedure used to set up the network interface when the default is not appropriate. This field may be null.

See **interface(4)** for detailed information on the **/etc/confnet.d/inet/interface** file.

The configured **/etc/confnet.d/netdrivers** file contains the following:

```
el16_0 inet
wd_0 inet
```

The fields in the **netdrivers** file are separated by white space. The first field is the name of the network card as it appears in the **/dev** directory. The second field is the protocol that has been mapped to the device that is shown on the same line.

See **`netdrivers(4)`** for detailed information on the **`netdrivers`** file.

Configuring the System as a Gateway

Since we are configuring the second (or subsequent) card in this machine, we are asked the following question:

```
Do you want to set this machine as a gateway?
[y/n/unchanged]: y
```

For our example, we answered `y`. At this point, the **`configure`** script updates the `IPFORWARDING` variable (if needed) within the file **`/etc/conf/cf.d/stune`**. If the `IPFORWARDING` variable had to be updated, the UNIX system kernel will need to be rebuilt. You will see the following message:

```
The UNIX Operating System kernel will be rebuilt
to include your configuration changes during the
next system reboot.
```

Before you make any other changes, you should reboot your system by double-clicking on the Shutdown icon or by typing the following (as the `root` user):

```
cd / ;
shutdown -i6 -g0 -y
```

Your UNIX kernel will be rebuilt, and your machine will be rebooted.

Sample Router Files

Consider a router called `jekyll` that connects two Class C networks. Like all routers, `jekyll` must have two hardware network interfaces. For TCP/IP to work properly, the router must have a unique host name in the **`/etc/hosts`** file for each interface.

Because the internetwork on which `jekyll` resides is a simple one, `jekyll` doesn't need to run the routing daemon. It knows about all the available hosts by virtue of being directly connected to both local networks. Notice in the sample file that, on the second network, `jekyll` has another name: `jekyll-hyde`. (`jekyll` is the router's primary name.) For ease of administration, similar host names are used for each network. Users on both networks can address the machine by the primary name `jekyll`, while the administrator can tell the difference between the two.

```
#
# sample hosts file
#
# 192.9.200 -- eng -- Engineering Network
#
192.9.200.1      jekyll
192.9.200.2      usher
192.9.200.3      lenore
192.9.200.5      raven
# 192.9.201 -- mktg -- Marketing Network
#
192.9.201.11     quasimoto
192.9.201.12     godzilla
192.9.201.13     rodan
192.9.201.4      jekyll-hyde
```

The following screen is an excerpt from `jekyll`'s **`/etc/networks`** file. The networks file contains just the names and IP addresses of the networks to which `jekyll` belongs.

```
#
# sample networks file
#
eng      192.9.200    # Engineering Network
mktg     192.9.201    # Marketing Network
```

Setting up the Route Daemon

Routers manage network traffic by maintaining routing tables. These routing tables contain information that specifies which networks and hosts can be reached by which routes. A routing table entry can be either static or dynamic.

On a more complex network, that is, a network in which a router connects a local network to other routers and gateways, the router should be configured to use dynamic routing tables. Dynamic routing tables allow the router to route traffic to the most current gateway destinations.

A router builds and maintains dynamic tables by running the routing daemon **`routed(1M).`** The routing daemon manages its routing table by exchanging routing information with gateways and other routers. When **`routed`** runs on a router, it broadcasts its routing table and listens for broadcasts from other directly connected routers. It continually updates its routing table based on those broadcasts. A routing daemon that both broadcasts its routing tables and listens for broadcasts from other routers is called an “active” **`routed`**.

NOTE

`routed` can also run on a client. However, when it runs on a client, **`routed`** simply listens for broadcasts and updates its local routing table; it does not broadcast to other machines. This is called a “passive” **`routed`**. For information about running **`routed`** on a client, see **`routed(1M)`** and “Setting Up Router Clients” on page 13-14.

When **`routed`** is first initialized on a router, it builds its table using the contents of the file **`/etc/gateways`** (if it exists), which contains the address and distance to various networks and gateways. Once it starts to run, **`routed`** immediately begins to update its table based on broadcasts from other routers and gateways.

You need to create **`/etc/gateways`** if you want to tell other networks about a router that doesn't broadcast information to you. You can use any text editor to create **`/etc/gateways`**. The **`/etc/gateways`** file consists of a series of lines, each in the following format:

`[net | host]` *hostname1* `gateway` *hostname2 value* `[passive | active]`

where the operands have the following meanings:

`net	host`	This keyword indicates whether the route is to a network or to a specific host, respectively.
hostname1	Specifies the name of the destination network or host. This may be a net name or host name, as specified in **`/etc/networks`** or **`/etc/hosts`**, or an IP address.
hostname2	Specifies the name or address of the gateway to which messages should be forwarded.
value	This is a number indicating the number of “hops” to the destination host or network.
`passive	active`	This keyword indicates whether the gateway should be treated as passive or active gateway, respectively.

To run **`routed`** on a router automatically whenever TCP/IP is initialized, do the following:

1. Set up the **`/etc/gateways`** file, if necessary. The changes will become active when you reboot your system.

2. Access the start-up script **`/etc/inet/rc.inet`** and delete the **`-q`** from the following line:

```
/usr/sbin/in.routed -q
```

so that the line becomes:

```
/usr/sbin/in.routed
```

For more information about **`/etc/gateways`** and **`routed`**, see **`routed(1M)`**.

Setting Up Router Clients

After you configure a router to access new networks, the clients of that router need to be informed of the path to those networks. This is done through the **`route`** command or by the routing daemon, **`routed`**.

The client uses the information you supply in the **`route`** command to build or update its internal routing table, or its routing daemon creates its routing table based on broadcasts from the router. The client then uses the table to find the “next hop” to the destination it is trying to reach.

Either the **`route`** command or **`routed`** must be run on the client every time TCP/IP is initialized. To ensure that either the command or the daemon is run automatically after booting, you must modify the client's start-up script, **`/etc/inet/rc.inet`**. You can edit the script to specify different kinds of routing behavior, as described below:

- If there is only one router on the local network, you can set up the `rc.inet` script so that, by default, your client's attempts to reach outside the local network are directed to this router. Since there is only one router to access external networks, no additional processing is needed on the client.
- If your client has more than one router available to it on the local network, you can direct traffic to specific networks through specific routers.
- If your client is connected to a router that runs an active routing daemon, the client can take advantage of the router's dynamic routing tables by running a passive **`routed -q`**.

The following sections tell you how to specify the different kinds of routing described above.

Specifying a Default Router

If there is only one router connected to a local network, clients should direct all external traffic through the router by default. (Once you do this procedure, you no longer have dynamic routing. You should not do this procedure if you may have other machines doing routing at some time in the future.)

To specify default routing, do the following:

1. Access **`/etc/inet/rc.inet`** on the client and locate the following line.

```
#/usr/sbin/route add default your_nearest_gateway hops_to_gateway
```

2. Enable default routing by editing the line, as follows:
   - Delete the comment character (#).
   - Substitute the router's host name on your local network for `your_nearest_gateway`.
   - Substitute the argument `1` for `hops_to_gateway`.
3. Disable the route daemon by inserting the comment character (#) into the following line:

```
/usr/sbin/in.routed -q
```

   so the line now reads:

```
#/usr/sbin/in.routed -q
```

Using an example from a preceding section, assume that networks named `eng` and `mktg` are connected by a router. The router has two network interfaces, one for each network, and each interface has a different host name. On the network named eng, the router's host name is `jekyll`; on `mktg`, its name is `jekyll-hyde`.

If your client is on the network named `eng`, you would edit the **`route`** command in your **`/etc/inet/rc.inet`** file to look like this:

```
/usr/sbin/route add default jekyll 1
```

If your client is on the network named `mktg`, your **`route`** command would look like this:

```
/usr/sbin/route add default jekyll-hyde 1
```

In both examples, the argument **`default`** indicates that all communication directed outside the local network from your client should be routed through the router.

Using Specific Routers

If your machine is a client of more than one router that uses static routing tables, you may want to specify which routers should route traffic to which networks.

For example, assume your system has two routers, `barker` and `lovett`, available to it on the local network. Your system needs to communicate regularly with networks named `sales` and `adv`. You know that the router `barker` maintains information about `sales` in its routing table, and `lovett` knows how to get to `adv`. To specify that traffic from your system should follow these routes, you would edit the **`route`** command in **`/etc/inet/rc.inet`** as follows:

```
/usr/sbin/route add sales barker 1
```

In the above case, `barker` is directly connected to the network `sales`. Below the statement in **`/etc/inet/rc.inet`** you add a second route command:

```
/usr/sbin/route add adv lovett 1
```

These commands set up your system so that it sends messages intended for different networks through different routers, rather than sending all traffic through the same router. If all of your routes are defined in **`/etc/inet/rc.inet`**, you can comment out the

/usr/sbin/in.routed -q line in the **/etc/inet/rc.inet** file to improve system performance.

A powerful way to set up routing for more complex networks is to provide several specific "**route add**" commands for frequently used networks, as well as a "**route add default**" command to handle traffic to all other networks. This is particularly useful when routers, in turn, use other routers.

For more information, see the **route(1M)** manual page.

Running the Route Daemon on a Client

If your machine is a client to a router that maintains a dynamic routing table, you can set up your client to update its own tables based on the router's broadcasts.

If the router is running the **routed** daemon, **routed** listens for broadcasts from other routers and gateways and continually updates its routing tables based on the information in those broadcasts. It also broadcasts its own routing tables so that other machines can use them for updating their own routing tables. Because **routed** on a router both broadcasts its own routing table and listens for broadcasts from other routers, it is known as an "active" **routed**. When **routed -q** is run on a client (that is, on a machine with only one network interface), it listens on the network and waits for the local router to broadcast its routing table. It then uses the broadcast information to update its own table. Because **routed -q** on a client does not broadcast its routing table, it is called a "passive" or "quiet" **routed**.

NOTE

Even if the local router uses dynamic routing tables, you can choose not to run **routed** on the client. In this case, the client's tables remain static, even though the router's tables are dynamic. You might choose not to run **routed** on the client if the client is on a simple internetwork, and the routes to its usual destinations do not change.

If the client's router uses static routing tables (that is, it does not run **routed**), do not run **routed** on the client.

Setting Up Subnets

Using subnets is a way to divide the addresses for a single network to accommodate the fact that the network consists of several physical networks. For administrative or technical reasons, many organizations choose to divide one network into several subnets.

Routing can get very complicated as the number of networks grows. For example, a small organization might give each local network a Class C number. As the organization grows, administering network numbers may get out of hand. A better idea is to allocate a Class B network for the entire company. Then, divide the Class B network into physical networks using subnets. In this way, you can isolate hosts from changes you might make to the network in remote parts of the organization.

Subnets allow you more flexibility when assigning network addresses. The Internet Protocol allows 127 Class A networks with 24-bit host fields; 16,383 Class B networks with 16-bit host fields; and over two million Class C networks with eight bit host fields.

Network Masks

Typically, you create subnets by using a subnetting scheme called the “network mask” or netmask. When setting up your network, you should select a network mask that applies to each network interface. A network mask extends the network ID portion of an address by taking some bits from the host portion to create a subnet portion. The remaining bits represent the host within the subnet. For example, if an organization has a Class B network, you could assign each physical network in that network a subnet number within that network. The 16 bits for the host ID could be allocated as eight for subnet and eight for host, or nine for subnet and seven for host, and so on. Your decision would be transparent to everyone outside that organization. Figure 13-1 illustrates the effect of the network mask:

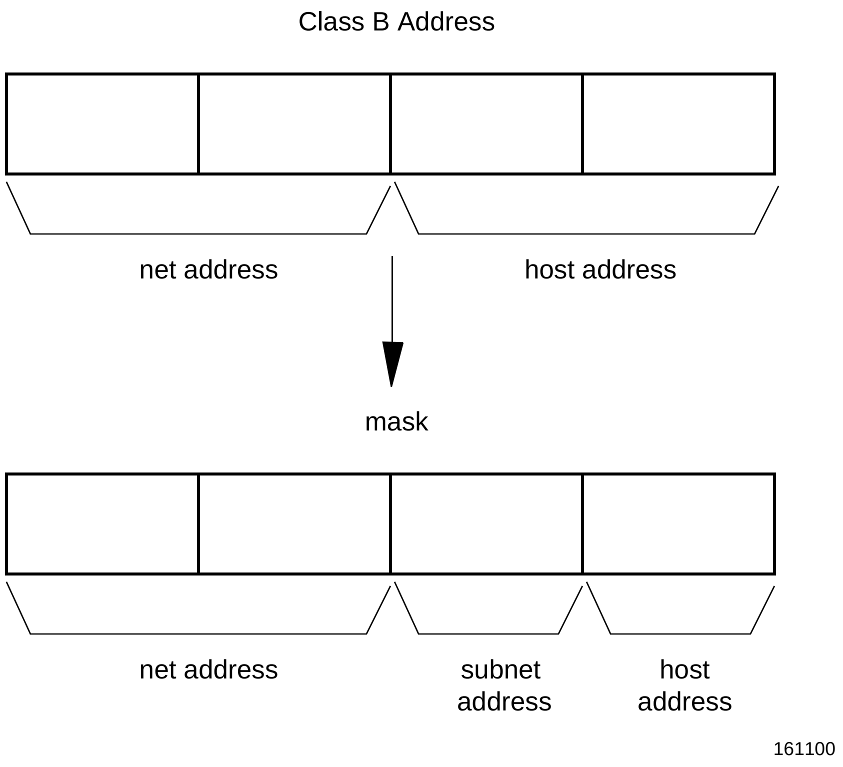


Figure 13-1. Network Mask

You can express network masks as a single hexadecimal number, or as four octets of decimal numbers, as described earlier in this guide. The default is a mask of `0xFF000000` (`255.0.0.0`) for Class A networks, `0xFFFF0000` (`255.255.0.0`) for Class B networks, and `0xFFFFFF00` (`255.255.255.0`) for Class C networks. You only have to specify network masks explicitly when they are wider (that is, have more one-bits) than the default values. Ones correspond to the network and zeros correspond to the host.

One common case is a Class C type subnet mask on a Class B network. A Class B network provides you with 256 possible subnets, each one of which can accommodate 254 possible hosts (remember, within an 8-bit host field `0` and `255` are not acceptable). But you may know that none of your subnets will ever have more than, say, 128 hosts, while you may need more than 256 subnets. In that case, you could decide to use nine bits for the subnet number instead of eight, and seven for the host addresses. The appropriate mask for this would be `0xFFFFFF80`, or `255.255.255.128`.

Given the above scheme, and a network address of, for instance, `131.60`, the address for the first host of the first subnet would be `131.60.0.129`.

Changing to a Subnetted Network

Follow these steps to change from an internetwork that does not use subnets to one that is subnetted.

1. Decide on the new subnet topology, including considerations for subnet routers and locations of hosts on the subnets.
2. Assign all subnet and host addresses.
3. Edit **`/etc/hosts`** on all hosts to change host address.

Examples of Subnets

The following examples show network installations where subnets are (and are not) in use:

```
128.32.0.0 Berkeley Class B network (subnetted) netmask 255.255.255.0
36.0.0.0 Stanford Class A network (subnetted) netmask 255.255.0.0
10.0.0.0 Arpanet Class A network (non-subnetted) netmask 255.0.0.0
```

All of the University of California at Berkeley is assigned the network number `128.32.0.0`, so that any external router only needs to know one route to reach Berkeley. Within the campus, a Class C type subnet mask is used to give each local network a subnet number, with 254 hosts on each of the 254 possible subnets (remember that zero and all ones, that is `255`, are reserved). Stanford University uses a Class A network number with a Class B type subnet mask, for 254 subnets of 65534 hosts each. The Arpanet is a Class A network without subnets; therefore, the default Class A netmask is used.

14
Managing TCP/IP Nodes Using SNMP

14 Managing TCP/IP Nodes Using SNMP

Introduction to SNMP Administration

The Simple Network Management Protocol (SNMP) is a protocol used to monitor and control TCP/IP-based networks. The SNMP allows the retrieval and alteration of networking information maintained by hosts and routers attached to a network. A network administrator can use the SNMP to diagnose and correct network problems from remote hosts.

Using the OS SNMP implementation, a network administrator can gather information such as routing entries, interface status, and protocol statistics. If problems are encountered, the administrator can manipulate items such as the ARP cache and the routing table to add, delete, and modify incorrect entries.

This chapter provides a brief history of the SNMP and describes the implementation of SNMP.

SNMP History

As the Internet expanded, its character changed. As the number of nodes at a site increased, the amount of time administrators had available to monitor their hosts and networks decreased. Additionally, many sites were being connected whose administrators were relatively new to internet working. Also, many companies and universities had large networks, with equipment located at remote sites. Finally, the growing number of sites connecting to the Internet placed an ever-increasing strain on the organizations that managed the Internet itself.

It became obvious to many people that more automated tools were needed in order to allow network administrators to manage their networks and quickly diagnose problems. Several of these people got together in 1987 and developed the Simple Gateway Monitoring Protocol (SGMP), which is the direct ancestor of the SNMP.

After a year or so of operational experience with the SGMP, several enhancements were made. The most notable was the adoption of the international data encoding standard, Abstract Syntax Notation One (ASN.1). The use of ASN.1 enabled the new protocol to convey information about any arbitrary data type, and allowed for much greater flexibility than the SGMP provided. The new protocol was re-named the Simple Network Management Protocol.

Now, only a few years after its introduction, the SNMP is the standard management protocol for managing TCP/IP-based networks. Although the SNMP protocol itself is stable, the information that can be processed via the SNMP constantly evolves and expands. In this way, the SNMP is able to keep up with new networking hardware and software advances.

Basic SNMP Concepts

There are three basic components of the SNMP: the protocol, the Structure of Management Information (SMI), and the Managements Information Base (MIB).

SNMP Protocol

The first basic component of the SNMP is the protocol itself. This is currently defined in RFC 1157. The SNMP is a very simple protocol with four basic operations: **`get`**, **`get next`**, **`set`**, and **`trap`**. Each operation is encoded in a separate Protocol Data Unit (PDU). An additional PDU is defined for replying to **`get`** and **`get next`** operations. The PDUs are carried through the network in the User Datagram Protocol (UDP).

Although using an unreliable protocol such as UDP may seem non-intuitive at first, it has definite advantages. When attempting to get management packets through a network that is damaging or corrupting packets, it is possible to get UDP packets through faster than establishing a TCP connection. This is because the SNMP requires only one packet to get through for the request, and an additional one for the response. Most common TCP implementations would require three packets just to establish a connection.

Structure of Management Information (SMI)

The next fundamental component of SNMP-based network management is the Structure of Management Information, or SMI. This is a framework that describes the basic types of information that can be manipulated by the SNMP. The SMI is described in detail in RFC 1155. The SMI is a skeleton that specifies the basic format and hierarchy of management data. It does not describe the objects that can be managed, but rather the building blocks from which the managed objects are constructed.

A fundamental concept of the SNMP is the notion of object identifiers. An object identifier (OID) is a tag that allows a management entity to refer unequivocally to a particular object. Object identifiers are allocated in a tree fashion. The value of the object identifier is a series of integers that refer to a particular traversal of the object tree.

The root of the OID tree has no label. Currently, there are three children of the root, **`CCITT(0)`**, **`ISO(1)`**, and **`Joint-ISO-CCITT(2)`**. The **`ISO(1)`** node has many children, one of which is **`org(3)`**, which is allocated for international organizations. Under **`org(1)`** is the U.S. Department of Defense, **`dod(6)`**, which has the child **`internet(1)`**. Figure 14-1 shows the object identifier hierarchy. The name `{ iso org dod internet }` is a symbolic representation for the integer series `1.3.6.1`. Both refer to the object identifier of the Internet sub-tree. In practice, `1.3.6.1` can simply be referred to as `internet`. That is to say, `{ iso org dod internet }`, `1.3.6.1`, `internet`, and `{ dod 1 }` are all different ways of identifying the same object. In SNMP PDUs, only the numeric sequences are used.

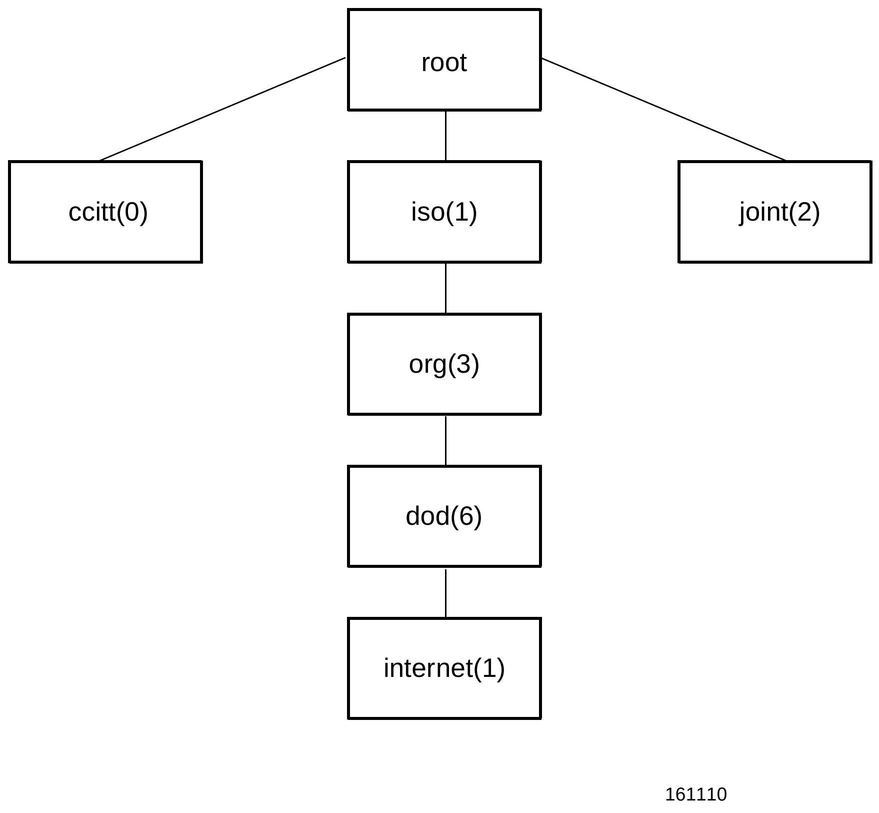


Figure 14-1. Object Identifier Hierarchy

To ensure that object identifiers are unique, each organization is responsible for a particular section of the OID tree. Just as ISO and CCITT have responsibility for their portions, the Internet Activities Board (IAB) has responsibility for the Internet portion. To allow vendors to support objects that may not be defined in the standard Management Information Base, or MIB, the IAB reserves a portion of the OID tree for enterprise MIBs. This provides vendors with the flexibility they need.

The SMI defines basic types that make it convenient to describe managed objects. Among these types are `ipAddress`, `Counter`, `Gauge`, `TimeTicks`, and `Opaque`. Readers who are interested in the precise semantics of these objects are encouraged to read the RFC for further information.

Management Information Base (MIB)

The final piece of the SNMP is the MIB. The MIB lists the standard objects that any SNMP implementation should support. The original MIB is defined in RFC 1156. An expanded version now in common use is defined in RFC 1213.

Each object in the MIB has two important characteristics. The first is its OID. The second is its type. Object types are constructed from the fundamental types defined in the SMI.

The MIB is divided into logically related groups of objects. The important groups are: `system`, `interfaces`, `tcp`, `udp`, `ip`, `egp`, `icmp`, and `snmp`.

The `system` group contains information about the network node. An example of this would be the physical location of the node, for example, “R&D Facility, 3rd floor machine room.”

The `interfaces` group contains information about network interfaces, such as Ethernet and point-to-point links. Information is kept about such items as interface status (for example, up or down), packet counts, and so on.

The rest of the groups contain information about the particular protocol to which they refer. This includes items like number of packets received with a particular protocol type, and number of packets received with incorrect checksums.

Agents and Management Stations

Systems using the SNMP are divided into two categories: agents and management stations. The management station is the system that issues a query. The agent is the system that is being queried.

Although there is no architectural reason that a system cannot have both types of functionality, most SNMP agents are usually found in dedicated routers and other specialized network equipment that is not capable of acting as a management station.

Using the SNMP implementation, time-sharing systems and workstations are able to act as both agents and management stations.

Traps

Although the SNMP is normally a synchronous protocol in which systems poll each other periodically, it does include a mechanism for asynchronous events. In the SNMP, these are referred to as “traps.” Traps are sent by agents to management stations when an abnormal event, such as an interface state-change, occurs.

Authentication

In the current implementation of the SNMP, authentication is based on a field in the PDU called the community name. When an agent receives a request, it validates the community name/IP address pair of the sender by comparing it against a specified list of acceptable management stations. If the sender is not found in the list, the packet is discarded. Optionally, an authentication trap may be sent to a specified group of management stations, informing them that an invalid access has been attempted.

The Internet Engineering Task Force (IETF) is currently working on a proposal for better authentication.

Overview of SNMP

As stated above, the SNMP product provides the functionality of both agent and management station. A system running the software can monitor other systems and be monitored as well.

The SNMP agent is implemented as a daemon. There are three configuration files associated with the daemon. These files consist of the following: **`/etc/inet/snmpd.conf`**, **`/etc/inet/snmpd.comm`**, and **`/etc/inet/snmpd.trap`**. The agent is started when the system enters the multi-user state (traditionally run-level 2). Upon start-up, the agent reads configuration information from its configuration files and then begins listening for SNMP requests on the SNMP port. If configured to send traps, the agent will also notify the appropriate management stations that it is up. This is done by sending a "cold-start" trap to the systems listed in **`/etc/inet/snmpd.trap`**.

`/etc/inet/snmpd.conf` contains the basic information that the agent needs as part of the system group. These are the types of the SNMP and TCP/IP software, the name of the person responsible for the system, the location of the system, and the object identifier of the agent. Typically, only the name of the responsible person and the system location would need to be changed.

`/etc/inet/snmpd.comm` contains a list of community/IP address pairs from whom the agent will accept queries. Along with each pair is an access field that controls whether access is allowed, and if so, whether the access granted is read-only, write, or none (no access).

In addition to the agent, there are user commands that provide management station functionality to the system administrator. These are: **`getid(1M)`**, **`getmany(1M)`**, **`getnext(1M)`**, **`getone(1M)`**, **`getroute(1M)`**, **`setany(1M)`**, and **`snmpstat(1M)`**.

The **`snmpstat`** command provides the administrator with an easy way to retrieve information such as routing tables, address translation tables, and interface status. Using **`snmpstat`**, a network administrator can quickly detect error conditions on a misbehaving network node.

The remaining commands provide the administrator with the ability to retrieve or modify arbitrary objects. Many vendors have their own enterprise MIBs, in addition to the standard MIB. Since these continually evolve, it may well be the case that a particular node maintains information above that which is conveniently accessed via **`snmpstat`**. The additional commands provide this functionality.

Configuring the SNMP Agent

As stated above, there are three configuration files that need to be modified before starting the SNMP agent. The first file that must be changed is the SNMP configuration file, **`/etc/inet/snmpd.conf`**.

There are four variables defined in the configuration file: `descr`, `objid`, `contact`, and `location`. Of these four, only `location` and `contact` should need modification on a per-system basis.

The `contact` variable defines what will be returned by the agent when a management station asks for the `sysContact` MIB variable (which is part of the **`system`** group). Change this string to be the name of the person responsible for the system. Note that it is also a valid practice to include additional information in the `contact` definition, such as a phone number to call if there are problems with the system.

The `location` variable defines what will be returned in response to a query for the `sysLocation` MIB variable. This should be changed to reflect the physical location of the system.

Screen 14-1 shows a sample configuration file.

```
descr=Concurrent Computer Corporation PowerMAX OS SNMP Agent
objid=ConcurrentComputerCorporation
contact=Steve Alexander x256
location=1901 N.  Naper Blvd., Naperville, IL, 1 West
```

Screen 14-1. Sample snmpd.conf File

The next set of information to adjust is the list of community/host pairs from which the agent will accept requests. This information is stored in the community file **`/etc/inet/snmpd.comm`**. Each line of the community file is a separate access record. Each record has three fields: `community` name, IP `address`, and `access`. The fields must be separated by white space.

Community names can be arbitrarily long, with the restriction that the agent has a 255 character limit on the length of any single record, including the `address` and `access` fields.

The IP address `0.0.0.0` is a special case; it signifies that any host can access the agent using the specified community. The type of access is defined by the third field. The `access` field has three possible values: `read`, `write`, and `none`. `read` allows read-only access; `write` allows read-write access; `none` blocks all access.

In essence, **`snmpd.comm`** is a permissions file that states whom to accept requests from, and what to let them do.

Screen 14-2 shows a sample community configuration file.

```
# community      address          access
#
public           0.0.0.0          none
private          128.212.64.90    read
ops              128.212.64.2     write
```

Screen 14-2. Sample snmpd.comm file

The last file that needs to be modified is the trap information file **`/etc/inet/snmpd.trap`**. This file contains a list of hosts to whom the agent will send its trap PDUs. Each line in the file is a separate record, and each record has three fields: `community` name, destination IP `address`, and destination `port`.

The first two fields are defined as they are in the communities file. The third field should normally be `162`, which is the standard SNMP trap port on which a management station will listen for trap requests.

Screen 14-3 shows a sample trap configuration file.

```
# community      address          port
#
public           128.212.64.90    162
private          128.212.65.12    162
```

Screen 14-3. Sample snmpd.trap file

For additional information about the SNMP agent and its configuration files, consult the **`snmpd(1M)`**, **`snmpd.comm(4)`**, **`snmpd.conf(4)`**, and **`snmpd.trap(4)`** manual pages.

Using the SNMP Commands

This section explains how to use the SNMP commands.

From the `system` group we'll look at the following variables: `sysDescr`, `sysContact`, and `sysName`. These are the description of the node, the name of the person responsible for the node, and the name of the node.

From the `interfaces` group we'll look at the `ifDescr`, `ifOperStatus`, `ifType`, and `ifPhysAddress` variables. These are the name, status, type, and physical address associated with a particular interface.

From the `ip` group we'll look at routing table information. The variables used will be `ipRouteDest`, `ipRouteNextHop`, and `ipRouteType`. These are the destination and next hop of the route entry, plus the routing protocol (such as RIP or EGP) that provided the routing information.

getone Command

The easiest command to use for retrieving specific variables is **`getone(1M).`** The syntax of **`getone`** is:

`getone` *node community variable-name [variable-name]* . . .

So, in order to ask node `grinch` for its system description, one would use the command:

```
getone grinch public sysDescr.0
```

This assumes that `grinch` will accept requests with a community of `public`. Screen 14-4 shows two examples of the **`getone`** command:

```
compost# getone grinch public sysDescr.0
Name: sysDescr.0
Value: Concurrent Computer Corporation PowerMAX OS SNMP Agent
  .
  .
  .
  .
Name: sysName.0
Value: grinch
```

Screen 14-4. Getone Example

getid Command

The **`getid(1M)`** command retrieves a subset of the system group from a node. The syntax of **`getid`** is:

`getid` *node community*

Screen 14-5 shows an example of the **`getid`** command:

```
Name: sysDescr.0
Value: Concurrent Computer Corporation PowerMAX OS SNMP Agent
Name: sysObjectID.0
Value: enterprises.1457
Name: sysUpTime.0
Value: 61971755
```

Screen 14-5. getid Example

The `sysUpTime` variable is the number of clock ticks since the system was restarted; each tick is 1/100th of a second.

getmany Command

The **`getmany(1M)`** command can retrieve multiple variables at a time. The syntax of the **`getmany`** command is:

`getmany` *node community variable-class [variable-class] . . .*

For example, you can retrieve the entire `system` group using the command:

```
getmany node community system
```

Or, you could get information about all interfaces attached to the node with the command:

```
getmany node community interfaces
```

Screen 14-6 shows **getmany** being used to get the `system` group:

```
Name: sysDescr.0
Value: Concurrent Computer Corporation PowerMAX OS SNMP Agent
Name: sysObjectID.0
Value: enterprises.1457
Name: sysUpTime.0
Value: 61976949
Name: sysContact.0
Value: Local System Administrator
Name: sysName.0
Value: grinch
Name: sysLocation.0
Value: Computer Center
Name: sysServices.0
Value: 72
```

Screen 14-6. getmany Example

snmpstat Command

The **snmpstat(1M)** command provides a subset of the functionality of **getone** and **getmany**, but has a much friendlier user interface. The **snmpstat** command can display routing, address translation, system, interface, and TCP connection information in a form that is easier to understand.

The syntax of the **snmpstat** command is:

`snmpstat [`*options*`] [`*node* `[`*community*`]`

If no *node* argument is specified, **snmpstat** queries the local system. If no *community* argument is specified, the default community of `public` is used. With no options specified, **snmpstat** displays the table of transport (TCP and UDP) endpoints.

Table 14-1 lists the options available to **`snmpstat`**.

Table 14-1. snmpstat Options

Option	Description
`-S`	Show the `snmp` group
`-a`	Show the address translation table
`-i`	Show the `interface` group
`-n`	Print addresses numerically
`-r`	Show the routing table
`-s`	Show the `system` group
`-t`	Show all connections

Screen 14-7 shows an example of using **`snmpstat`** to retrieve the `system` group.

```
System Group
Description:Concurrent Computer Corporation PowerMAX OS SNMPAgent
ObjectID: ConcurrentComputerCorporation
UpTime: 7 days 4 hor, 11 minutes, 10 seconds, 12 hundredths
Contact: Local System Administrator
Name: grinch
Location: Computer Center
Services: applications, end-to-end
```

Screen 14-7. snmpstat System Group Example

In this example, notice that **`snmpstat`** takes care of formatting the information in a way that is easier to understand. For example, notice how **`snmpstat`** converts the raw timeticks of the system uptime into a human-readable time.

Screen 14-8 shows an example of using **`snmpstat`** to retrieve routing information.

```
compost# snmpstat -r grinch public
Routing table
Destination      Gateway      Metric   Type     Proto Interface
0.0.0.0          houdini      3        rem      rip ie0
localhost        localhost    0        dir      local lo0
isc-i88-backbone grinch       0        dir      local ie0
isc-i88-dev-net  mcfeely      2        rem      rip ni0
```

Screen 14-8. snmpstat Routing Example

In this example, four routes are returned from `grinch`. Two of the routes are directly attached. These are the routes for `localhost` and the `isc-i88-backbone` subnet. Two other routes have been acquired by the RIP routing protocol from the hosts `houdini` and `mcfeely`.

setany Command

The **`setany(1M)`** command can be used to modify variables. The basic syntax for the **`setany`** command is:

`setany` *node community variable type value*

The valid values for `type` are given in Table 14-2. For a complete explanation of these types, refer to RFC 1155.

Table 14-2. setany Variable Types

Object Type	setany Flag
Integer	**`-i`**
Octet	**`-o`**
Object Identifier	**`-d`**
IP address	**`-a`**
Counter	**`-c`**
Gauge	**`-g`**
Time_ticks	**`-t`**
Display String	**`-s`**
Null	**`-n`**

Screen 14-9 shows an example of using **`setany`** to alter the status of an interface. Notice that, in this example, the status is verified before and after the change to ensure that the change took place correctly. Note that changing the status from `1` to `2` marks the interface down.

```
compost# getone grinch public ifAdminStatus.2
Name: ifAdminStatus.2
Value: 1
 .
 .
 .
compost# setany grinch public ifAdminStatus.2 -i 2
Name: ifAdminStatus.2
Value: 2
 .
 .
 .
compost# getone grinch public ifAdminStatus.2
Name: ifAdminStatus.2
Value: 2
```

Screen 14-9. Altering interface status with setany

Using SNMP to Correct Problems

This section includes several examples of using SNMP.

Obtaining Remote System Contacts

Suppose that the administrator notices a number of connection attempts to a restricted service (such as NNTP) from an unknown host. In this case, the administrator can use the **`snmpstat`** command to get information about the unknown host. Screen 14-10 shows a possible scenario.

```
compost# snmpstat grinch public
Active Internet connections
Proto Local Address Foreign Address (state)
tcp grinch.nntp bach.CS.ESU.EDU.1021 ESTABLISHED
 .
 .
 .
compost# snmpstat -s bach.CS.ESU.EDU
System Group
Description: SNMPD VERSION 9.4.0.0 SUN 3/260 - SUNOS3.5
ObjectID: SNMP_Research_UNIX_agent.9.4.0.3
UpTime: 4 days, 15 hours, 56 minutes, 39 seconds, 74 hundredths
Contact: Operations 1-999-555-1040
Name: bach.CS.ESU.EDU
Location: Comp Center
Services: applications, end-to-end
```

Screen 14-10. Verifying the Origin of a Network Connection

Removing an Incorrect Routing Entry

Suppose that, due to a configuration error, a system in another part of the building had an incorrect entry in its routing table, and that this was causing problems communicating with other hosts in other company offices. In this case, the network administrator could use SNMP commands to correct the incorrect routing entry remotely. Screen 14-11 shows a possible scenario. Notice that, for clarity, the command line in the example is split with a backslash to stay within the page margins. In actual practice, the command line should be typed to the shell as a single line of input. Notice also that, in addition to the new next hop, the administrator specifies the type of route. In this case, the route is a remote route (type `4`), which was reported by the **`snmpstat`** command as `rem`.

```
compost# snmpstat -r grinch public | grep 123.0.0.0
123.0.0.0    fonebone     2        rem      rip      ie0
 .
 .
 .
compost# setany grinch public ipRouteNextHop.123.0.0.0 \
     -a 128.212.64.1 ipRouteType.123.0.0.0 -i 4
Name: ipRouteNextHop.123.0.0.0
Value: 128.212.64.1
Name: ipRouteType.123.0.0.0
Value: 4
```

Screen 14-11. Correcting Routing Entries

Note that the administrator could also have deleted the route by setting its `ipRouteType` to invalid (`2`). This is useful if a routing protocol will reacquire the route information automatically.

Marking an Interface Down

Suppose that an interface attached to a router is reporting many errors. In this case, the administrator might wish to mark the link down while the phone company checks the line. The SNMP could be used to do this remotely.

First, the error count can be obtained using the **`snmpstat`** command. Next, the interface can be marked down using the **`setany`** command. Screen 14-12 shows both operations.

```
Interface statistics
                    Type       InOctet    InPckts      InErrs        IfMtu
Name   Address      Speed      OutOctet   OutPckts     OutErrs       OutQlen
lo0    localhost    loop       20419918   63080        0             2048
                    0          20419918   63080        0             0
ie0    grinch       0          1011588253 13242145     0             1500
       0000c30268d6 0          142235789  15409487     107           0
```

Screen 14-12. Marking an Interface Down

When the line is repaired, the interface can be reactivated in a similar fashion.

Removing an Incorrect ARP Entry

Suppose that a misconfigured system had an invalid IP address which caused bad ARP information to be installed on another node. The SNMP could be used to remove the invalid IP address remotely. This can be very useful if the node in question has a very long time-out on the ARP cache, or if some important connection is being established with the wrong machine. Screen 14-13 shows a possible scenario.

```
compost# snmpstat -a grinch public
Address Translation table
Host                 Physical Address     Flags    Interface
128.212.64.9         8:0:14:40:1:21       dyna     ie0
128.212.64.79        0:0:c0:6d:82:23      dyna     ie0
 .
 .
 .
compost# setany grinch public ipNetToMediaType.1.128.212.64.9 -i 2
 .
 .
 .
compost# snmpstat -a grinch public
Address Translation table
Host                 Physical Address     Flags    Interface
128.212.64.79        0:0:c0:6d:82:23      dyna     ie0
```

Screen 14-13. Removing an ARP Entry

15
Using Domain Name Service with TCP/IP

15
Using Domain Name Service with TCP/IP

Introduction to Domain Name Service Administration

Domain Name Service (DNS) is an application layer protocol that is part of the standard TCP/IP protocol suite. Specifically, DNS is a naming service; it obtains and provides information about hosts on a network.

Domain Name Service performs naming between hosts within your local administrative domain and across domain boundaries. It is distributed among a set of servers, commonly known as name servers, each of which implements DNS by running a daemon called **`named`**.

NOTE

The **`named`** daemon is also called the Berkeley Internet Name Domain service, or BIND, because it was developed at University of California at Berkeley.

On the client's side, DNS is implemented through the resolver. The resolver is neither a daemon nor a particular program; rather, it is a library compiled into applications that need to know machine names. The resolver's function is to resolve users' queries. To do that, it queries a name server, which then returns either the requested information or a referral to another server.

DNS is dependent on the hierarchical structure of the Internet, and adds to that structure the concept of domain zones. This chapter describes the organization of domains as it relates to DNS on the Internet, then tells you how to set up and maintain DNS on your network machines.

The Domain Hierarchy and DNS Administrative Zones

Domain names used by DNS reflect the hierarchical organization of the public networks. Before you set up DNS for your organization, you should understand the organization of the network, your place in the overall structure of the network, and how domain names reflect that structure.

For information about the structure of the Internet and domain naming conventions, see Chapter 18, "Obtaining Domain Names".

The Domain Name Service introduces the concept of zones to the Internet network model. A zone is a hierarchical community of hosts, administered by a single authority and served by a set of name servers. This community can include individual hosts, plus every name server and its clients (a subzone) nested beneath the authoritative set of name servers for the zone. Zones usually represent administrative boundaries, such as your local administrative domain. However, a zone is not limited to just one administrative domain. A zone is a domain, plus all or some of the domains under it.

Figure 15-1 shows how some fictional zones might be delegated. The illustration shows four zones: the `root` zone, served by the Internet root domain name servers, and three zones served by name servers at domain `sun.COM`. The dotted lines point to the areas making up each zone.

Zones take their names from the label of the domain at the top of the zone hierarchy. In Figure 15-1, the names of the four domains within dotted lines are:

```
sun.COM venus.sun.COM mars.sun.COM .
```

where the "." (dot) at the end of the line signifies the root zone. Zone `sun.COM` takes its name from the label of the second level domain `sun.COM`. However, zone `sun.COM` does not have the same administrative authority as the domain of the same name. Zone `sun.COM` consists only of:

- domain `sun`
- local administrative domain `earth`
- subdomains `telstar`, `moon`, `eurostar`, `france`, `greece`, and `uk`

The zone `sun.COM` does not include domains `venus` and `mars`. They have their own separately administered zones: `venus.sun.COM` and `mars.sun.COM`, respectively. However, `venus` and `mars` are part of the `sun.COM` domain hierarchy.

The domain hierarchy and name space described so far keeps track of information by host name. This enables the **named** daemon to perform name-to-address mapping.

In addition, there is a special domain recognized by DNS called `IN-ADDR.ARPA`, which facilitates address-to-name mapping. `IN-ADDR.ARPA` contains essentially the same information as the hosts name space, but it is expressed in terms of IP addresses.

A name in the `IN-ADDR.ARPA` domain has four labels preceding it, corresponding to the four octets of an IP address. This host address is listed from right to left. For example, a host whose IP address is `128.32.0.4` has the following `IN-ADDR.ARPA` domain name:

```
4.0.32.128.IN-ADDR.ARPA.
```

Therefore, if **named** knows the IP address of a host, it can find out the host's fully qualified domain name by consulting a file representing the `IN-ADDR.ARPA` domain. (This file, commonly called **`hosts.rev`**, is explained in a later section.)

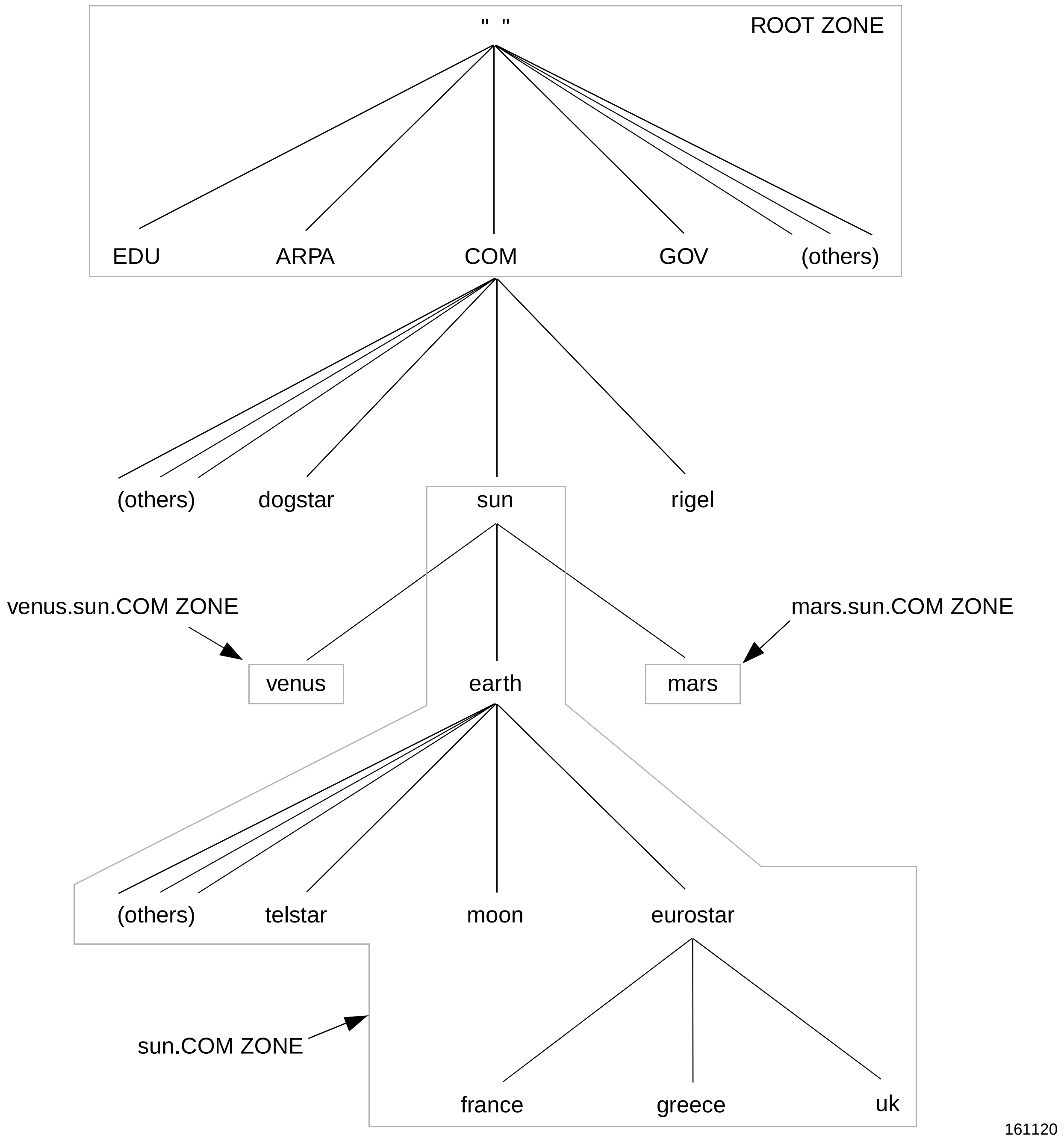


Figure 15-1. Administrative Zones

The Administrator's Role

Administering DNS involves not only running the appropriate programs on the servers and clients, but also determining domain names, answering complaints, and filling out registration and other forms, should you join a public network. This section contains overall procedures for setting up DNS, based on the responsibilities you might have as a DNS administrator.

Your administrative responsibilities for DNS depend on your domain's position within the overall network hierarchy. For example, managing one set of name servers in a small administrative domain entails less responsibility than managing the authoritative set for a large zone. Responsibilities depend on whether you are the chief authority for a domain or zone, or an administrator reporting to the chief authority.

The Network Information Center (NIC) divides administrators on the Internet into domain administrators, who have primary responsibility for a domain, and technical contacts, who work with domain administrators to maintain a zone. Descriptions of each position appear below.

The Domain Administrator

The domain administrator (DA) is a coordinator, manager, and technician for a second-level or lower domain. The DA's responsibilities include the following:

- Registering the domain.

  If your domain is going to be on a public network, you must register it (if you have not done so already). The domain must have a unique name within its level of the network hierarchy. To register your domain with the Internet, obtain the Domain Registration Form from NIC and complete it according to the instructions in the chapter “Obtaining Domain Names”. To join other public networks, contact the organization in charge of the network and request the appropriate domain registration form.

NOTE

> To belong to multiple networks such as the Internet, BITNET, or CSNET, you need to register with only one network, as domain names are independent of a network's physical topology.

- Naming hosts and verifying that names within the domain are unique.

  At many sites, users name their individual hosts while the administrators name the servers. The administrator should ensure that there are no duplicate names within a zone.

- Understanding the functioning of the name servers and making sure the data is current at all times.

  This includes either setting up the DNS-related files and programs, or delegating this authority to other technically competent people (such as the technical contact). This chapter should provide you with the basic information you need for understanding DNS. However, as the domain administrator, you should gain more specific technical knowledge of your particular network. Contact the public network to which you belong and ask for technical papers regarding it.

The Technical Contact

The primary responsibility of a technical contact is to maintain DNS programs and files for a zone and to keep the zone's name servers running. Technical contacts interact with their domain administrators and with DA's of other domains to solve network problems.

DNS Clients and Servers

As mentioned earlier, there are two sides to DNS: clients running the resolver and name servers running the **named** daemon.

Clients

A name server running **named** can also run the resolver; therefore, there can be two kinds of clients:

- client-only
- client/server

A client-only client does not run the **named** daemon; instead, it consults the resolver, which provides a list of possible name serving machines to which queries should be directed.

A client/server is a machine that uses the domain name service provided by **named** in order to resolve a user's queries. However, if the daemon dies or hangs, the client/server might be able to solve queries through its resolver.

Servers

You implement DNS for a zone not on a single server, but on a set of servers. This set must include two master servers, and may or may not include other servers.

- Master servers

  The master name servers maintain all the data corresponding to the zone, making them the authority for that zone. These are commonly called authoritative name servers. The data corresponding to any given zone must be available on at least two authoritative servers. You should designate one name server as the primary master server and at least one as a secondary master server, to act as a backup if the primary is unavailable or overloaded.

  The primary master server is the name server where you make changes for the zone. This server loads the master copy of its data from disk when it starts up **named**. The primary server may also delegate authority to other servers in its zone, as well as to servers outside of it.

  The secondary master server is a name server that maintains a copy of the data for the zone. The primary server sends its data and delegates its authority to the secondary server. When the secondary server boots **`named`**, it requests all the data for the given zone from the primary. The secondary server then periodically checks with the primary to see if it needs to update its data.

A server may function as a master for multiple zones: as a primary for some zones, and as a secondary for others.

A server at the root level of the network is called a root domain name server. On the Internet, root domain name servers are maintained by the NIC. If a network is not connected to the Internet, primary and secondary name servers must be set up and administered for the `root` level of the local network.

- Caching and caching-only servers

  All name servers are caching servers. This means that the name server caches received information until the data expires. (The expiration process is regulated by the `time to live` field attached to the data when it is received from another server.)

  Additionally, you can set up a caching-only server that is not authoritative for any zone. This server handles queries and asks other name servers who have the authority for the information needed. But the caching-only server does not maintain any authoritative data itself.

Setting Up DNS on a Client

Setting up DNS on a client involves performing the following two tasks:

1. Creating the file **`resolv.conf`**
2. Modifying the file **`netconfig`**

NOTE

If you are setting up DNS on a name server, you need to complete these tasks, in addition to setting up boot and data files and editing the startup script.

Creating resolv.conf

The domain name server uses several files to load its database. At the resolver level, it needs a file called **`/etc/resolv.conf`**, which lists the addresses of the servers where the resolver can obtain the information needed. Whenever the resolver has to find the address of a host (or the name corresponding to an address) it builds a query package and sends it to the name servers it knows of (from **`/etc/resolv.conf`**).

The servers either answer the query locally, or use the services of other servers and return the answer to the resolver.

The **`resolv.conf`** file is read by the resolver to find out the name of the local domain and the location of name servers. It sets the local domain name and instructs the resolver routines to query the listed name servers for information. Every DNS client-only system on your network must have a **`resolv.conf`** file in its **`/etc`** directory.

NOTE

On a system where a name server is running, an **`/etc/resolv.conf`** file is unnecessary.

The first line of the file lists the domain name in the form:

`domain` *domain_name*

where *domain_name* is the name registered with the NIC. Succeeding lines list the IP addresses that the resolver should consult to resolve queries. IP address entries have the form:

`nameserver` *IP_address*

A sample **`resolv.conf`** file is shown in Screen 15-1:

```
; Sample resolv.conf file
domain Podunk.Edu
; try local name server
nameserver 127.0.0.1
; if local name server down,
try these servers
nameserver 128.32.0.4
nameserver 128.32.0.10
```

Screen 15-1. Sample /etc/resolv.conf File

NOTE

It is recommended that you do not have an **`/etc/resolv.conf`** file on a system that is running a name server, because **`/etc/resolv.conf`** causes the system to be less efficient. However, when the name server is not running, the resolver consults other name servers to answer queries. Therefore, if it is important that your system have little or no down time, create **`/etc/resolv.conf`** on your system and include the loopback address, followed by the addresses of other name servers.

Modifying netconfig

Once you have created the appropriate **/etc/resolv.conf** file on each machine that is to run the resolver, you need to edit the **netconfig** file. Look at the file and make sure that all transport providers listed in the **/etc/netconfig** file that provide IP connections have the name-to-address resolving library that uses DNS (**resolv.so**) listed in the name `lookups` field. (See **netconfig(4)** for more information about **netconfig**.)

For example, if your **/etc/netconfig** file lists the following entries:

```
netid    semantics    flags    family   proto    device       nametoaddr_libs
tcp      tpi_cots_ord v        inet     tcp      /dev/tcp     /usr/lib/tcpip.so,
                                                              /usr/lib/resolv.so
udp      tpi_clts     v        inet     udp      /dev/udp     /usr/lib/tcpip.so
                                                              /usr/lib/resolv.so
icmp     tpi_raw      -        inet     icmp     /dev/icmp    /usr/lib/tcpip.so,
                                                              /usr/lib/resolv.so
rawip    tpi_raw      -        inet     -        /dev/rawip   /usr/lib/tcpip.so,
                                                              /usr/lib/resolv.so
```

you will have to modify the `nametoaddr_libs` field so it says:

```
netid    semantics    flags    family   proto    device       nametoaddr_libs
tcp      tpi_cots_ord v        inet     tcp      /dev/tcp     /usr/lib/resolv.so,
                                                              /usr/lib/tcpip.so
udp      tpi_clts     v        inet     udp      /dev/udp     /usr/lib/resolv.so,
                                                              /usr/lib/tcpip.so
icmp     tpi_raw      -        inet     icmp     /dev/icmp    /usr/lib/resolv.so,
                                                              /usr/lib/tcpip.so
rawip    tpi_raw      -        inet     -        /dev/rawip   /usr/lib/resolv.so,
                                                              /usr/lib/tcpip.so
```

NOTE

In single-user mode, **named** is not available, so the **/etc/netconfig** file must include a lookup routine that uses local text files (**tcpip.so** in the example above is the default). You should therefore put the local machines' interface addresses and at least a couple of machine names and addresses into the **/etc/hosts** file.

Setting Up DNS on a Name Server

Because name servers also use the resolver, it is necessary to set up **/etc/netconfig** and, optionally **/etc/resolv.conf**, as described in the section “Setting Up DNS on a Client” on page 15-6. If you create an **/etc/resolv.conf** file on a name server, you should include the line:

```
nameserver 127.0.0.1
```

before any other `nameserver` lines. Next, you need to create the boot and data files that **named** needs. Instructions for performing both of these tasks appear in this section.

If your local network is not on the Internet, you must set up primary and secondary servers in the root-level domain on the local network. Instructions for setting up a root domain name server appear at the end of this section.

Creating Boot and Data Files

In addition to the daemon **named**, DNS on a name server consists of a boot file and local data files. The default location of the boot file is **/etc/inet/named.boot**. Common names for the local data files are **named.ca**, **named.local**, **host**s, and **hosts.rev**. In the descriptions of these files that follow, these names are used. However, you can name these files whatever you like. In fact, it is recommended that you use names other than the ones used in this guide, to avoid confusion with files with similar names.

- **named.boot**

  The boot file **named.boot** establishes the server as a primary, a secondary, or a caching-only name server. It also specifies the zones over which the server has authority, and which data files it should read to get its initial data.

  The boot file is read by **named** when the daemon is started by the server's startup script **/etc/inet/rc.inet**. The boot file directs **named** either to other servers, or to local data files for a specified domain.

- **named.ca**

  The **named.ca** file establishes the names of root servers and lists their addresses. If you are connected to the Internet, **named.ca** lists the Internet name servers; otherwise, it lists the root domain name servers for your local network. **named** cycles through the list of servers until it contacts one of them. It then obtains from that server the current list of root servers, which it uses to update **named.ca**.

  To avoid confusion with the caching process, with which this has very little to do, you may want to call your file something like **named.root** or **boot.root**.

- **named.local**

  The **named.local** file specifies the address for the local loopback interface, or localhost, with the network address 127.0.0.1. The name of this file is specified in the **named.boot** file. As with all other files, it can be called something other than the name used in this guide.

- **hosts**

  The **hosts** file contains all the data about the machines in the local zone. The name of this file is specified in the **named.boot** file. To avoid confusion with **/etc/hosts**, name the file something other than hosts. You may want to give the file name the suffix zone; in the following illustration, the file is called **A.puzone**.

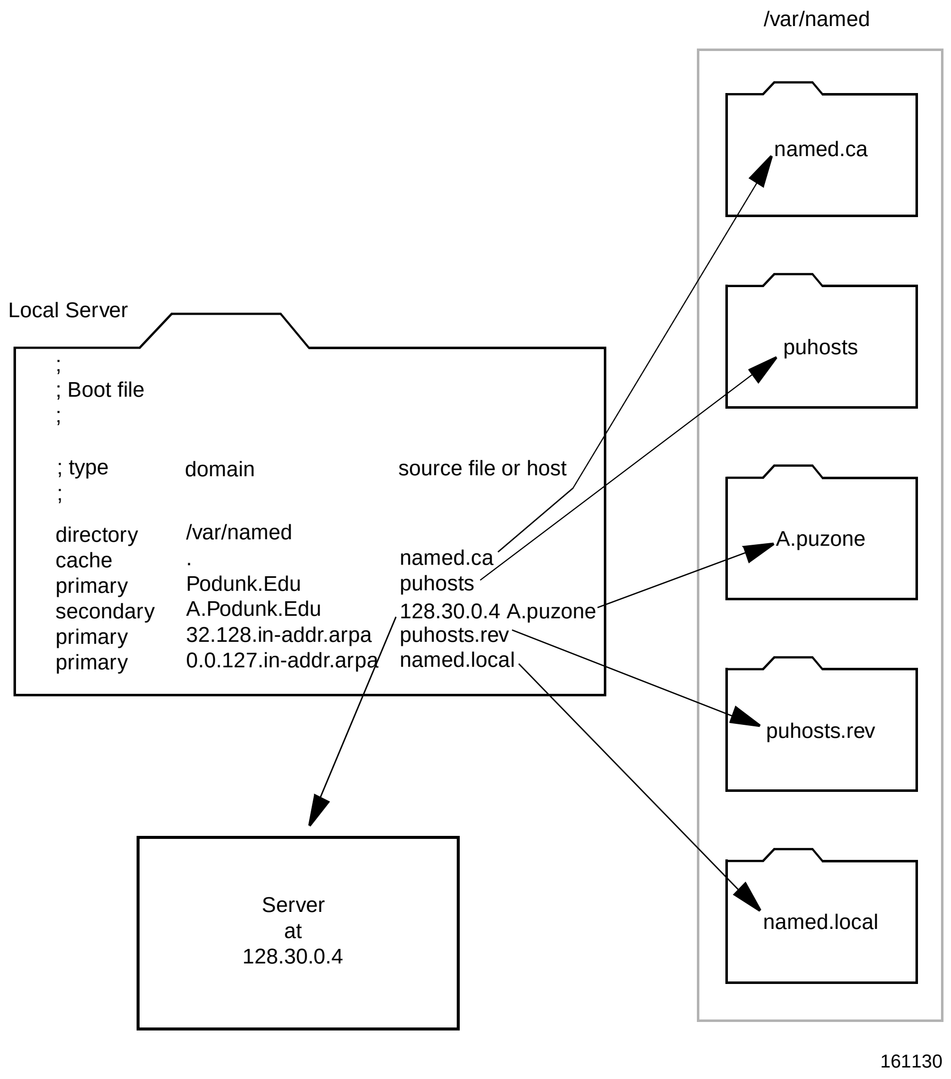
- **hosts.rev**

  The **hosts.rev** file specifies a zone in the IN-ADDR.ARPA domain (the special domain that allows inverse mapping). The name of this file is specified in the **named.boot** file. In the illustration, the file is called **puhosts.rev**.

The relationship between the **named.boot** file and the data files (or, alternatively, other servers) is illustrated in Figure 15-2.

Setting Up the Boot File

The contents of the boot file vary, depending on the type of server. This section describes boot files for primary and secondary master servers and caching-only servers.



```
;
; Boot file
;

; type          domain                  source file or host
;

directory       /var/named
cache           .                       named.ca
primary         Podunk.Edu              puhosts
secondary       A.Podunk.Edu            128.30.0.4 A.puzone
primary         32.128.in-addr.arpa     puhosts.rev
primary         0.0.127.in-addr.arpa    named.local
```

Figure 15-2. Boot File and Data Files

Boot File For a Primary Master Server

A sample boot file for a primary server is shown in Screen 15-2.

```
; Sample named.boot file for Primary Master Name Server
;
; type domain source file or host
;
directory /var/named
cache .  named.ca
primary Podunk.Edu puhosts
primary 32.128.in-addr.arpa puhosts.rev
primary 0.0.127.in-addr.arpa named.local
```

Screen 15-2. Sample Boot File For a Primary Master Server

The entries in the file are explained below.

`directory`

The following line in the boot file designates the directory in which you want the name server to run:

```
directory /var/named
```

This allows the use of relative path names for the files mentioned in the boot file or, later, with the **`$INCLUDE`** directive. It is especially useful if you have many files to be maintained and you want to locate them all in one directory dedicated to that purpose.

If there is no directory line in the boot file, all file names listed in it must be full pathnames.

`cache` A name server needs to know which servers are the authoritative name servers for the root zone. To do this, you have to list the addresses of these higher authorities.

All servers should have the following line in the boot file to find the root name servers:

```
cache . named.ca
```

The first field indicates that the server will obtain root server hints from the indicated file, in this case **`named.ca`** (located in the directory **`/var/named`**).

`primary` To set up a primary server, you need to create a file that contains all the authoritative data for the zone. Then you create a boot file that designates the server as a primary server and tells it where to find the authoritative data.

The following line in the boot file names the server and the data file:

```
primary     Podunk.Edu   puhosts
```

The first field designates the server as primary for the zone stated in the second field. The third field is the name of the file from which authoritative data is read. Note that the domain name in the second field must not end with a "." (dot).

The lines

```
primary 32.128.in-addr.arpa puhosts.rev
primary 0.0.127.in-addr.arpa named.local
```

show that the server is also a primary server for the domains `32.128.in-addr.arpa` (that is, the reverse address domain for `Podunk.Edu`) and `0.0.127.in-addr.arpa` (that is, the local host loopback), and that the data for them is to be found, in the files **`puhosts.rev`** and **`named.local`**, respectively.

Boot File For a Secondary Master Server

A sample boot file for a secondary server in the same domain as the above primary server is shown in Screen 15-3.

```
; Sample named.boot file for Secondary Master Name Server
;
;type    domain   source file or host
;
directory /var/named
cache .  named.ca
secondary Podunk.Edu 128.32.0.4 128.32.0.10 128.32.136.22 puhosts.zone
secondary 32.128.in-addr.arpa 128.32.0.4 128.32.0.10 128.32.136.22 purev.zone
primary 0.0.127.in-addr.arpa named.local
```

Screen 15-3. Sample Boot File for a Secondary Master Server

In appearance, this file is very similar to the boot file for the primary server; the main difference is to be found in the lines:

```
secondary Podunk.Edu 128.32.0.4 128.32.0.10 128.32.136.22 puhosts.zone
secondary 32.128.in-addr.arpa 128.32.0.4 128.32.0.10 128.32.136.22 purev.zone
```

The keyword `secondary` establishes that this is a secondary server for the zone listed in the second field, and that it is to get its data from the listed servers (usually the primary server followed by one or more secondary ones); attempts are made in the order in which the servers are listed. If there is a filename after the list of servers (as in the example above), data for the zone will be put into that file as a backup. When the server is started, data is loaded from the backup file, if it exists, and then one of the servers is consulted to check whether the data is still up to date.

This ability to specify multiple secondary addresses allows for great flexibility in backing up a zone. The interpretation of the other "secondary line" is similar to the above. Note also that, although this is a secondary server for the domain `Podunk.Edu` and `32.128.in-addr.arpa`, this is a primary server for `0.0.127.in-addr.arpa` (the local host).

NOTE

A server may act as the primary server for one or more zones, and as the secondary server for one or more zones. The mixture of entries in the boot file determines whether the name server is a primary or secondary server for each zone it serves.

Boot File For Caching-only Server

A sample boot file for a caching-only server is shown in Screen 15-4.

```
; Sample named.boot file for Caching Only Name Server
;
; type domain source file or host
;
cache .  named.ca
primary 0.0.127.in-addr.arpa named.local
```

Screen 15-4. Sample Boot File For a Caching-only Server

You do not need a special line to designate a server as a caching-only server. What denotes a caching-only server is the absence of authority lines, such as `secondary` or `primary` in the boot file. As explained above, a caching-only server does not maintain any authoritative data; it simply handles queries and asks the servers listed in the **`named.ca`** file for the information needed.

Setting Up the Data Files

All the data files used by the DNS daemon **`named`** are written in Standard Resource Record Format. In Standard Resource Record Format, each line of a file is a record, called a Resource Record (RR). Each DNS data file must contain certain Resource Records. This section describes the DNS data files, and the Resource Records each file should contain. Following this section is a discussion of the Standard Resource Record Format, including an explanation of each Resource Record relevant to the DNS data files.

The hosts File

The **`hosts`** file contains all the data about all the machines in your zone, including server names, addresses, host information (hardware and operating system information), canonical names and aliases, the services supported by a particular protocol at a specific address, and group and user information related to mail services. This information is represented in the records NS, A, HINFO, CNAME, WKS, MX, MB, MR, and MG. The file also includes the SOA record, which indicates the start of a zone and includes the name of the host on which the **hosts** data file resides. A sample hosts file is shown in Screen 15-5.

```
; sample hosts file
@        IN       SOA      ourfox.Sample.Edu.  kjd.monet.Sample.Edu.  (
                           1.1 ; Serial
                           10800 ; Refresh
                           1800 ; Retry
                           3600000 ; Expire
                           86400 ) ; Minimum
         IN       NS       ourarpa.Sample.Edu.
         IN       NS       ourfox.Sample.Edu.
ourarpa  IN       A        128.32.0.4
         IN       A        10.0.0.78
         IN       HINFO    3B2 UNIX
arpa     IN       CNAME    ourarpa
ernie    IN       A        128.32.0.6
         IN       HINFO    3B2 UNIX
ourernie IN       CNAME    ernie
monet    IN       A        128.32.7
         IN       A        128.32.130.6
         IN       HINFO    Sun-4/110 UNIX
ourmonet IN       CNAME    monet
ourfox   IN       A        10.2.0.78
         IN       A        128.32.0.10
         IN       HINFO    Sun-4/110 UNIX
         IN       WKS      128.32.0.10 UDP syslog route timed domain
         IN       WKS      128.32.0.10 TCP ( echo telnet
                           discard rpc sftp
                           uucp-path systat daytime
                           netstat qotd nntp
                           link chargen ftp
                           auth time whois mtp
                           pop rje finger smtp
                           supdup hostnames
                           domain
                           nameserver )
fox      IN       CNAME    ourfox
toybox   IN       A        128.32.131.119
         IN       HINFO    3B2 UNIX
toybox   IN       MX       0 monet.Sample.Edu
miriam   IN       MB       vineyd.Neighbor.COM.
postmistress      IN       MR       miriam
bind     IN       MINFO    bind-request kjd.Sample.Edu.
         IN       MG       ralph.Sample.Edu.
         IN       MG       zhou.Sample.Edu.
         IN       MG       painter.Sample.Edu.
         IN       MG       riggle.Sample.Edu.
         IN       MG       terry.pa.Xerox.Com.
```

Screen 15-5. Sample hosts File

The named.local File

The **named.local** file sets up the local loopback interface for your name server. It needs to contain the host name of the machine, plus a pointer to the host name localhost, which represents the loopback mechanism. The server name is indicated in the NS resource record, and the pointer to localhost by the PTR record. The file also needs to include a SOA record, which indicates the start of a zone, and includes the name of the host on which the **named.local** data file resides.

A sample **named.local** file is shown in Screen 15-6.

```
; sample named.local file
@ IN SOA ourhost.Podunk.Edu.  kjd.monet.Podunk.Edu.  (
 1 ; Serial
 3600 ; Refresh
 300 ; Retry
 3600000 ; Expire
 3600 ) ; Minimum
 IN NS ourhost.Podunk.Edu.
 1 IN PTR localhost.
```

Screen 15-6. Sample named.local File

The hosts.rev File

hosts.rev is the file that sets up inverse mapping. It needs to contain the names of the primary and master name servers in your local domain, plus pointers to those servers and to other, non-authoritative name servers. The names of the primary and secondary master servers are indicated by NS records, and the pointers by PTR records. The file also needs a SOA record to show the start of a zone and the name of the host on which hosts.rev resides. A sample **hosts.rev** file is shown in Screen 15-7.

```
; sample hosts.rev file
@ IN SOA ourhost.Podunk.Edu.  kjd.monet.Podunk.Edu.  (
 1.1 ; Serial
 3600 ; Refresh
 300 ; Retry
 3600000 ; Expire
 3600 ) ; Minimum
 IN NS ourarpa.Podunk.Edu.
 IN NS ourhost.Podunk.Edu.
4.0 IN PTR ourarpa.Podunk.Edu.
6.0 IN PTR ernie.Podunk.Edu.
7.0 IN PTR monet.Podunk.Edu.
10.0 IN PTR ourhost.Podunk.Edu.
6.130 IN PTR monet.Podunk.Edu.
```

Screen 15-7. Sample hosts.rev File

The named.ca File

The **named.ca** file contains the names and addresses of the root servers. Server names are indicated in the record NS and addresses in the record A. You need to add an NS record and an A record for each root server you want to include in the file. A sample **named.ca** file is shown in Screen 15-8.

```
;Initial cache data for root domain servers.
;
; list of servers...
 99999999 IN NS NS.NASA.GOV.
 99999999 IN NS NIC.DDN.MIL.
 99999999 IN NS A.ISI.EDU.
 99999999 IN NS TERP.UMD.EDU.
 99999999 IN NS C.NYSER.NET.
 99999999 IN NS BRL-AOS.ARPA.
 99999999 IN NS GUNTER-ADAM.ARPA.

; ...and their addresses
NIC.DDN.MIL.  99999999 IN A 10.0.0.51
NIC.DDN.MIL.  99999999 IN A 26.0.0.73
C.NYSER.NET.  99999999 IN A 192.33.4.12
BRL-AOS.ARPA.  99999999 IN A 128.20.1.2
BRL-AOS.ARPA.  99999999 IN A 192.5.22.82
NS.NASA.GOV.  99999999 IN A 128.102.16.10
TERP.UMD.EDU.  99999999 IN A 10.1.0.17
A.ISI.EDU.  99999999 IN A 26.3.0.103
GUNTER-ADAM.ARPA.  99999999 IN A 26.1.0.13
```

Screen 15-8. Sample named.ca File

Standard Resource Record Format

In the Standard Resource Record Format, each line of a data file is a record called a Resource Record (RR), containing the following fields, separated by white space:

[name] *[ttl]* *class* *type* *data*

The order of the fields is always the same; however, the first two fields are optional (as indicated here by the brackets), and the contents of the last field can vary, depending on the value of the *type* field.

name The first field specifies the name of the domain that applies to the record. This field is optional. If left blank in a given RR, it defaults the name of the previous RR.

A domain name in a zone file can be either a fully qualified name, terminated with a dot, or a relative name, in which case the name of the current domain is appended to it.

ttl The second field is an optional time-to-live field. The value of this field specifies how long (in seconds) this data will be cached in the database before it is disregarded and new information is requested from a server. If this field is left blank, *ttl* defaults to the minimum time specified in the Start Of Authority (`SOA`) Resource Record.

If the *ttl* value is set too low, the server will experience a lot of repeat requests for data refreshment. If, on the other hand, the *ttl* value is set too high, changes in the information will not be distributed in a timely manner.

Most *ttl*'s should be initially set to between a day (`86400`) and a week (`604800`); then, depending on the frequency of actual change of the information, the *ttl*'s should be modified to reflect that frequency. Also, if you have some *ttl*'s that have very high values because you know they relate to data that rarely changes, and you know that the data is now about to change, you should reset the *ttl* to a low value (`3600` to `86400`) until the change takes place, and then change it back to the original high value.

Note that all RRs with the same *name*, *class*, and *type* should have the same *ttl* value.

class	The third field is the record class. Only one class is currently in use: `IN` for the TCP/IP Internet protocol family.
type	The fourth field identifies the type of the Resource Record. There are many types of RRs; the most commonly used types are discussed in the section “Resource Record Types” on page 15-19.
data	The contents of this field depend on the type of the particular Resource Record.

Note that, although case is preserved in names and data fields when loaded into the name server, all comparisons and lookups in the name server database are case insensitive. However, this situation may change in the future, and you are advised to be consistent in your use of lowercase and uppercase.

Special Characters

The following characters have special meanings:

`.`	A single free-standing dot in the name field refers to the current domain.
`@`	A free standing `@` in the name field denotes the current origin.
`..`	Two free-standing dots in the name field represent the null domain name of the root domain.
\char	Escapes the meaning of *char*, where *char* is any character other than the digits 0-9. For example, you can use `.` (backslash-dot) to place a dot character in a label.
\digit	Escapes the meaning of the decimal number *digit*, where *digit* is a decimal octet. The octet is assumed to be text and is not checked for special meanings.
`( )`	Use parentheses to group data that crosses a line. In effect, line terminations are not recognized within parentheses.
`;`	A semicolon starts a comment. The remainder of the line is ignored.
`*`	An asterisk signifies wildcarding.

Most Resource Records have the current origin appended to names if they are not terminated by a dot (“.”). This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb is to use a fully qualified name ending with a dot if the name is not in the domain for which you are creating the data file.

Control Entries

The only lines that do not conform to the standard RR format in a data file are control entry lines. There are two kinds of control entries: include lines and origins lines.

$INCLUDE

An include line begins with **$INCLUDE** in column one, followed by a file name. The **$INCLUDE** command is particularly useful for separating different types of data into multiple files. For example, the line:

```
$INCLUDE /etc/named/data/mailboxes
```

is interpreted as a request to load the file **/etc/named/data/mailboxes** at that point.

Note that the **$INCLUDE** command does not cause data to be loaded into a different zone or tree. It is simply a way to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

$ORIGIN An origin line begins with **$ORIGIN** in column one, followed by a domain name. The **$ORIGIN** command is a way of changing the origin in a data file. It resets the current origin for relative domain names (that is, not fully qualified names) to the stated name. This is useful for putting more than one domain in a data file.

Resource Record Types

Table 15-1 shows some of the most commonly used types of Resource Records.

Table 15-1. Common Types of Resource Records

Type	Description
SOA	Start of Authority
NS	Name Server
A	Internet Address
CNAME	Canonical Name (nickname)
HINFO	Host Information
WKS	Well Known Services
PTR	Pointer
MX	Mail Exchange

We will use the following **hosts** file example to describe the different resource record types. For an explanation of each resource record type, refer to sections that follow Screen 15-9.

```
; sample hosts file
@        IN       SOA      ourfox.Sample.Edu.  kjd.monet.Sample.Edu.  (
                           1.1              ; Serial
                           10800            ; Refresh
                           1800             ; Retry
                           3600000          ; Expire
                           86400 ) ; Minimum
         IN       NS       ourarpa.Sample.Edu.
         IN       NS       ourfox.Sample.Edu.
ourarpa  IN       A        128.32.0.4
         IN       A        10.0.0.7
         IN       HINFO    3B2 UNIX
arpa     IN       CNAME    ourarpa
ernie    IN       A        128.32.0.6
         IN       HINFO    3B2 UNIX
ourernie IN       CNAME    ernie
monet    IN       A        128.32.7
         IN       A        128.32.130.6
         IN       HINFO    Sun-4/110 UNIX
ourmonet IN       CNAME    monet
ourfox   IN       A        10.2.0.78
         IN       A        128.32.0.10
         IN       HINFO    Sun-4/110 UNIX
         IN       WKS      128.32.0.10 UDP syslog route timed domain
         IN       WKS      128.32.0.10 TCP ( echo telnet
                           discard rpc sftp
                           uucp-path systat daytime
                           netstat qotd nntp
                           link chargen ftp
                           auth time whois mtp
                           pop rje finger smtp
                           supdup hostnames
                           domain
                           nameserver )
fox      IN       CNAME    ourfox
toybox   IN       A        128.32.131.119
         IN       HINFO    3B2 UNIX
toybox   IN       MX       0 monet.Sample.Edu
miriam   IN       MB       vineyd.Neighbor.COM.
postmistress      IN       MR       miriam
bind     IN       MINFO    bind-request kjd.Sample.Edu.
         IN       MG       ralph.Sample.Edu.
         IN       MG       zhou.Sample.Edu.
         IN       MG       painter.Sample.Edu.
         IN       MG       riggle.Sample.Edu.
         IN       MG       terry.pa.Xerox.Com.
```

Screen 15-9. Sample Resource Record Types

SOA - Start Of Authority Resource Record

The format of a Start of Authority (SOA) Resource Record is shown in Figure 15-3.

```
name    [ttl]    [class]    SOA       origin    person in charge (
                            serial
                            refresh
                            retry
                            expire
                            minimum)
```

Figure 15-3. Start of Authority Resource Record Format

The Start of Authority record designates the start of a zone. The zone ends at the next SOA record. There should only be one SOA record per zone.

The SOA record fields are:

name	Specifies the name of the zone. In the example below, `@` indicates the current zone or origin.
ttl	Specifies an optional time-to-live value.
class	Specifies an optional address class.
SOA	Identifies the type of this Resource Record.
origin	Specifies the name of the host where this data file resides.
person in charge	Specifies the mailing address for the person responsible for the name server.
serial	Specifies the version number of this data file. You must increment this number whenever you make a change to the data. A secondary server uses the value of the *serial* field to detect whether the data file has been changed since the last time it copied the file from the master server. Note that the name server cannot handle numbers greater than `9999` after the decimal point.
refresh	Specifies how often, in seconds, a secondary name server should check with the primary name server to see if an update is needed.
retry	Specifies how long, in seconds, a secondary server is to retry after a failure to check for a refresh.
expire	Specifies the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh.
minimum	Specifies the default number of seconds to be used for the time-to-live (*ttl*) field on resource records that don't have a *ttl* specified.

A sample SOA Resource Record is shown in Screen 15-10.

```
@ IN SOA ourfox.Sample.Edu.  kjd.monet.Sample.Edu.(
 1.1 ;Serial
 10800 ;Refresh
 3600 ;Retry
 432000 ;Expire
 86400)
;Minimum )
```

Screen 15-10. Sample Start of Authority Resource Record

NS - Name Server Resource Record

The format of a Name Server (NS) resource record is shown in Figure 15-4.

[name] *[ttl]* *class* `NS` *name-server name*

Figure 15-4. Name Server Resource Record Format

The Name Server record identifies the server responsible for a particular domain. The *name* field is optional and specifies the domain that is serviced by the **`named`** name server. If no *name* field value is specified, the server name defaults to the last name listed. One NS record should exist for each primary and secondary master server for the domain. A sample NS resource record is shown in Screen 15-11.

```
        IN   NS   ourarpa.Sample.Edu.
```

Screen 15-11. Sample Name Server Resource Record

A - Address Resource Record

The format of an Address (A) resource record is shown in Figure 15-5.

[name] *[ttl]* *class* `A` *address*

Figure 15-5. Address Resource Record Format

The Address record defines the address for a given machine. The *name* field is the machine name, and the *address* is the IP address. If no *name* field value is specified, the server name defaults to the last machine name listed. One A record should exist for each address of the machine (in other words, gateways should be listed twice, once for each address). A sample Address resource record is shown in Screen 15-12.

```
ourarpa  IN       A       128.32.0.4
         IN       A       10.0.0.7
```

Screen 15-12. Sample Address Resource Record

HINFO - Host Information Resource Record

The format of a Host Information (HINFO) resource record is shown in Figure 15-6.

[name] *[ttl]* *class* `HINFO` *hardware* *os*

Figure 15-6. Host Information Resource Record Format

The Host Information resource record contains host specific data. It identifies the type of machine (*hardware* field) and operating system (*os* field) that are running at the **named** host. Note that, if you want to include any white space in the *hardware* field values, you must surround the values with quotes. The *name* field is optional and specifies the name of the host. If no *name* field value is specified, the name defaults to the last host name listed. One HINFO record should exist for each host. A sample HINFO resource record is shown in Screen 15-13.

```
        IN       HINFO    Sun-4/110 UNIX
```

Screen 15-13. Sample Host Information Resource Record

WKS - Well Known Services Resource Record

The format of a Well Known Services (WKS) resource record is shown in Figure 15-7.

[name] *[ttl]* *class* `WKS` *address* *protocol* *list of services*

Figure 15-7. Well Known Services Resource Record Format

The Well Known Services record describes the well known services supported by a particular *protocol* at a specified *address*. The *list of services* and port numbers comes from the list of services specified in the **services** database. Only one WKS record should exist per protocol, per address. A sample WKS resource record is shown in Screen 15-14.

```
IN      WKS      128.32.0.10  UDP      syslog route timed domain
IN      WKS      128.32.0.10  TCP      (echo telnet
                                       discard rpc sftp
                                       uucp-path systat daytime
                                       netstat qotd nntp
                                       link chargen ftp
                                       auth time whots mtp
                                       pop rje finger smtp
                                       supdup hostnames
                                       domain
                                       nameserver)
```

Screen 15-14. Sample Well Known Services Resource Record

CNAME - Canonical Name Resource Record

The format of a Canonical Name (CNAME) resource record is shown in Figure 15-8.

nickname *[ttl]* *class* `CNAME` *canonical-name*

Figure 15-8. Canonical Name Resource Record Format

The Canonical Name resource record specifies a *nickname* for a machine's *canonical-name*. A nickname should be unique. All other resource records should be associated with the canonical name and not with the nickname. Do not create a nickname and then use it in other resource records. Nicknames are particularly useful during a transition period, when a machine's name has changed but you want to permit people using the old name to reach the machine. A sample CNAME resource record is shown in Screen 15-15.

```
ourmonet         IN       CNAME    monet
```

Screen 15-15. Sample Canonical Name Resource Record

PTR - Domain Name Pointer Resource Record

The format of a Domain Name Pointer (PTR) resource record is shown in Figure 15-9.

```
special-name [ttl]   class   PTR     actual-name
```

Figure 15-9. Domain Name Pointer Resource Record Format

A Pointer record allows special names to point to some other location in the domain. PTRs are used mainly in the `IN-ADDR.ARPA` records for the translation of an address (the *special-name*) to a real name (*actual-name*). PTR names should be unique to the zone. The PTR records in Screen 15-16 set up reverse pointers for the special `IN-ADDR.ARPA` domain.

```
7.0                         IN       PTR      monet.Podunk.Edu.
2.2.18.128.in-addr.arpa     IN       PTR      blah.junk.COM.
```

Screen 15-16. Sample Domain Name Pointer Resource Record

MX - Mail Exchanger Resource Record

The format of a Mail Exchanger (MX) resource record is shown in Figure 15-10.

name *[ttl]* *class* `MX` *pref mailer* *exchanger*

Figure 15-10. Mail Exchanger Resource Record Format

The Mail Exchanger (MX) resource record is used to specify a machine that knows how to deliver mail to a domain or to machines in a domain. There may be more than one MX resource record for a given name. In Screen 15-17, `Seismo.CSS.GOV.` (note the fully qualified domain name) is a mail gateway that knows how to deliver mail to `Munnari.OZ.AU.` Other machines on the network cannot deliver mail directly to `Munnari`. `Seismo` and `Munnari` may have a private connection or use a different transport medium. The value specified in the *pref*(erence) field indicates the order a mailer should follow when there is more than one way to deliver mail to a single machine. The value `0` (zero) indicates the highest preference. If there is more than one MX resource record for the same name, they may or may not have the same preference value.

Note that you can use names with the wildcard asterisk (`*`) for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. In Screen 15-17, all mail to hosts in domain `foo.COM` is routed through `RELAY.CS.NET`. You do this by creating a wildcard resource record, which states that the mail exchanger for `*.foo.COM` is `RELAY.CS.NET`. Note that the asterisk will match any host or subdomain of `foo.COM`, but it will not match `foo.COM` itself.

```
Munnari.OZ.AU.        IN      MX      0       Seismo.CSS.GOV.
foo.COM.              IN      MX      10      RELAY.CS.NET.
*.foo.COM.            IN      MX      20      RELAY.CS.NET.
```

Screen 15-17. Sample Mail Exchanger Resource Record

Modifying the Data Files

When you add or delete a host in one of the data files in the master DNS server, or otherwise modify the data files, you must also change the serial number in the SOA resource record so the secondary servers can modify their data accordingly. You should then inform **named** in the master server that it should re-read the data files and update its internal database, as explained below.

When **`named`** successfully starts up, it writes its process ID to the file **`/etc/inet/named.pid`**. To have **`named`** re-read **`named.boot`** and reload the database, type:

```
# kill -HUP 'cat /etc/inet/named.pid'
```

Note that all previously cached data is lost, and the caching process starts over again.

Starting Up the Name Service Daemon

Once you create the **`/etc/inet/named.boot`** file, TCP/IP knows to start up the **`in.named`** daemon automatically.

Setting Up a root Server for a Local Network

If you are not connecting your local network to the Internet, you must set up primary and secondary name servers in the root-level domain on the local network. You must do this so that all domains in the network have a consistent authoritative server to which to refer; otherwise, machines may not be able to resolve queries.

Since a single machine can be the primary domain name server for more than one domain, the easiest way to create a root domain name server is to have a server be the name server for all the domains that make up its own domain name. For example, if a server is named `x.sub.dom.`, then it should be designated the primary name server for “.”, `dom., and sub.dom.`

Since the root name server provides an authoritative name server at the root level of the network, all top-level domains should have their name server records (IN NS) defined in the root domain.

NOTE

It is strongly recommended that the root domain server name be the primary name server for all top-level domains in the network.

Troubleshooting named

From time to time, you may need to debug **`named`**. You can do this by sending it signals through the **`kill`** command [see **`kill(1)]`**. Depending on the signal received, **`named`** will change its behavior.

To get a good idea of what **named** thinks the database is, type:

```
kill -INT `cat /etc/inet/named.pid`
```

Upon receiving this signal, **named** dumps the current database and cache to **/var/tmp/named_dump.db**. This should give you an indication of whether the database was loaded correctly.

When **named** is running incorrectly, you can also look in **/var/adm/messages** and check for any messages logged by `syslog`. For instance, if there is a data file that lists a hostname as a nickname, and also has other data under that name, instead of under the machine's canonical name, you may see a message similar to this:

```
May 4 02:35:26 hostname named[4804]: hazy.widget.junk.COM
has CNAME and other data (illegal)
```

Or, if there is a problem with the database, you may see a message like this:

```
May 1 11:02:33 hostname named[17808]: /etc/named/junk.zone:
line 759: database format error ()
```

To turn on debugging, you can start **named** with the **-d** option, or, if **named** is already running, you can type:

```
kill -USR1 `cat /etc/inet/named.pid`
```

Note that each subsequent `USR1` will increment the debug level. The output goes to **/var/tmp/named.run**.

To turn off debugging completely, type:

```
kill -USR2 `cat /etc/inet/named.pid`
```

A Practical Example

We can now start building up the files that an imaginary network would need.

Let's assume that our network is composed of three networks, and all of them have access to the Internet. Each network has a Class C Network Number:

```
junk     223.100.100
widget   223.100.101
zap      223.100.102
```

The names of the zones are also the names of the hosts that we are designating as the master servers.

Our imaginary network can therefore be represented in Figure 15-11.

Let's further assume that, after careful consideration, we decide that we want to set up the Domain Name Service in our network so that each master server is the primary server for its zone, and a secondary server for the other zones. All this can be summed up as shown in Table 15-2, Table 15-3, and Table 15-4.

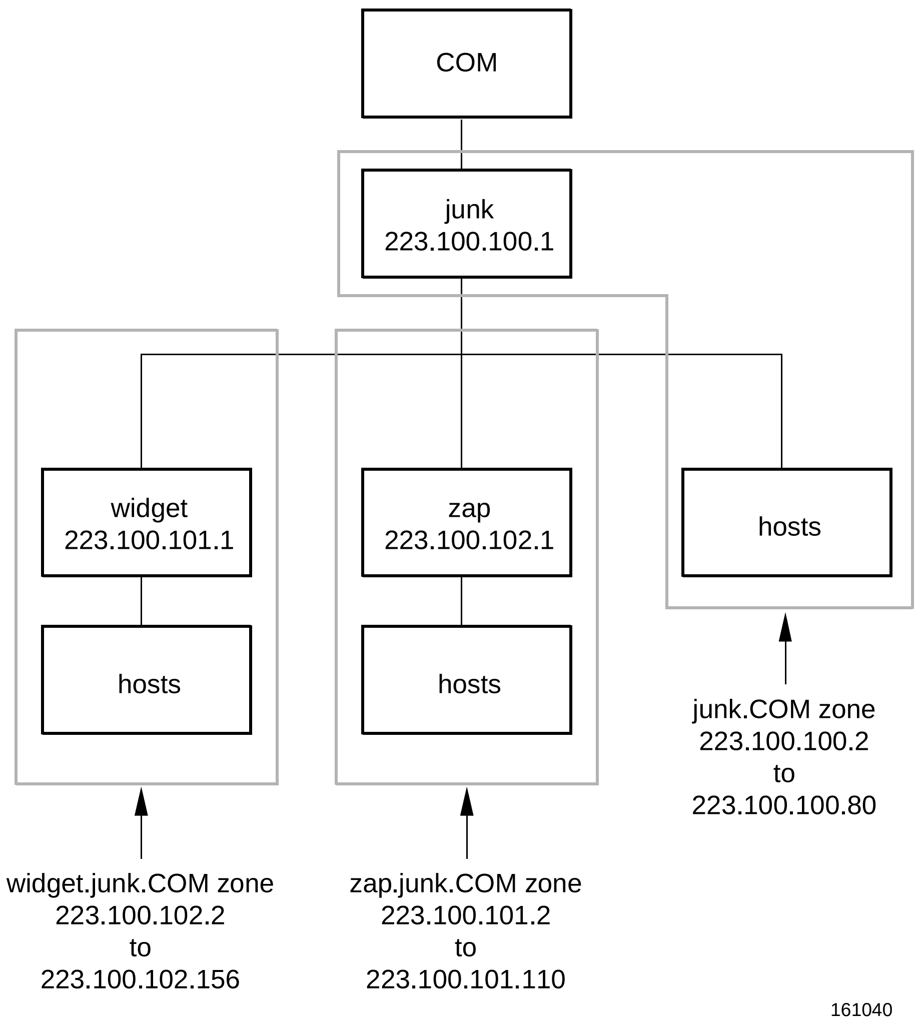


Figure 15-11. An Imaginary Network

Table 15-2. Example: junk Zone

Hostname	Function	Address
junk	primary	223.100.100.1
widget	secondary	223.100.101.1
zap	secondary	223.100.102.1
	hosts	223.100.100.2-80

Table 15-3. Example: widget Zone

Hostname	Function	Address
widget	primary	223.100.101.1
junk	secondary	223.100.100.1
zap	secondary	223.100.102.1
	hosts	223.100.101.2-110

Table 15-4. Example: zap Zone

Hostname	Function	Address
zap	primary	223.100.102.1
junk	secondary	223.100.100.1
widget	secondary	223.100.101.1
	hosts	223.100.102.2-156

boot Files

The following are the boot files for the three servers in the network. Note that the domain names in the boot files must not end with a dot (“.”).

```
;
;   Boot file for server junk
directory   /var/named
cache       .                           named.root
primary     junk.COM                    junk.zone
primary     100.100.223.in-addr.arpa    junk.revzone
primary     0.0.127.in-addr.arpa        named.local

secondary   widget.junk.COM             223.100.101.1 223.100.102.1 widget.zone
secondary   zap.junk.COM                223.100.102.1 223.100.101.1 zap.zone
secondary   101.100.223.in-addr.arpa    223.100.101.1 widget.rev
secondary   102.100.223.in-addr.arpa    223.100.102.1 zap.rev
    .
    .
    .
;
;   Boot file for server widget
directory   /var/named
cache       .                           named.root
primary     widget.junk.COM             widget.zone
primary     101.100.223.in-addr.arpa    widget.revzone
primary     0.0.127.in-addr.arpa        named.local

secondary   junk.COM                    223.100.100.1 223.100.102.1 junk.zone
secondary   zap.junk.COM                223.100.102.1 223.100.100.1 zap.zone
secondary   100.100.223.in-addr.arpa    223.100.100.1 junk.rev
secondary   102.100.223.in-addr.arpa    223.100.102.1 zap.rev
    .
    .
    .
;
;   Boot file for server zap
directory   var/named
cache       .                           named.root
primary     zap.junk.COM                zap.zone
primary     102.100.223.in-addr.arpa    zap.revzone
primary     0.0.127.in-addr.arpa        named.local

secondary   junk.COM                    223.100.100.1 223.100.101.1 junk.zone
secondary   widget.junk.COM             223.100.101.1 223.100.100.1 widget.zone
secondary   100.100.223.in-addr.arpa    223.100.100.1 junk.rev
secondary   101.100.223.in-addr.arpa    223.100.101.1 widget.rev
```

Screen 15-18. Sample boot Files

resolv.conf Files

The following are some sample **`resolv.conf`** files. Note that, if the host in question is not running **`named`**, the local host address should not be used as a name server.

```
;
; resolv.conf file for server junk
;
domain          junk.COM
nameserver      127.0.0.1
nameserver      223.100.101.1
nameserver      223.100.102.1
    .
    .
    .
;
; resolv.conf file for a host in zone junk not running named
;
domain          junk.COM
nameserver      223.100.100.1
nameserver      223.100.101.1
nameserver      223.100.102.1
    .
    .
    .
;
; resolv.conf file for a host in zone widget.junk not running named
;
domain          widget.junk.COM
nameserver      223.100.101.1
nameserver      223.100.100.1
nameserver      223.100.102.1
    .
    .
    .
;
; resolv.conf file for a host in zone zap.junk not running named
;
domain          zap.junk.COM
nameserver      223.100.102.1
nameserver      223.100.101.1
nameserver      223.100.100.1
```

Screen 15-19. Sample resolv.conf Files

named.local Files

The following are sample **named.local** files:

```
;
; named.local for server junk
;
@     IN   SOA      junk.COM.   ralph.sysad.zap.junk.COM. (
           1.1      ;Serial
           10800    ;Refresh
           3600     ;Retry
           432000   ;Expire
           86400)   ;Minimum

      IN   NS       junk.COM.
1     IN   PTR      localhost.
    .
    .
    .
;
; named.local for server widget
;
@    IN   SOA      widget.junk.COM.   ralph.sysad.zap.junk.COM. (
          1.1      ;Serial
          10800    ;Refresh
          3600     ;Retry
          432000   ;Expire
          86400)   ;Minimum

     IN   NS       widget.junk.COM.
1    IN   PTR      localhost.
    .
    .
    .
;
; named.local for server zap
;
@   IN   SOA      zap.junk.COM.   ralph.sysad.zap.junk.COM. (
         1.1      ;Serial
         10800    ;Refresh
         3600     ;Retry
         432000   ;Expire
         86400)   ;Minimum

    IN   NS       zap.junk.COM.
1   IN   PTR      localhost.
```

Screen 15-20. Sample named.local Files

hosts Files

The following is the **`hosts`** file for server **`junk`**, followed by its **`$INCLUDE`** file.

```
;
; junk zone hosts file for server junk
;
                IN   SOA       junk.COM.         ralph.sysad.zap.junk.COM. (
                      1.1      ;Serial
                      10800    ;Refresh
                      3600     ;Retry
                      432000   ;Expire
                      86400)   ;Minimum

                IN   NS        junk.COM.
                IN   NS        widget.junk.COM.
                IN   NS        zap.junk.COM.
widget.junk.COM IN   NS        widget.junk.COM.
                IN   NS       junk.COM.
                IN   NS       zap.junk.COM.
zap.junk.COM    IN    NS       zap.junk.COM.
                IN   NS       junk.COM.
                IN   NS        widget.junk.COM.
junk.COM.       IN   MX       10 junk.COM.
*.junk.COM.     IN   MX       10 junk.COM.

; junk.COM hosts
$INCLUDE /var/named/hosts/junk
     .
     .
     .
;
; junk zone hosts file for server junk
;
                IN   SOA       junk.COM.         ralph.sysad.zap.junk.COM. (
                      1.1      ;Serial
                      10800    ;Refresh
                      3600     ;Retry
                      432000   ;Expire
                      86400)   ;Minimum

                IN   NS        junk.COM.
                IN   NS        widget.junk.COM.
                IN   NS        zap.junk.COM.
widget.junk.COM  IN   NS       widget.junk.COM.
                IN   NS        junk.COM.
                IN   NS        zap.junk.COM.
zap.junk.COM.   IN    NS       zap.junk.COM.
                IN   NS        junk.COM.
                IN   NS        widget.junk.COM.
junk.COM.       IN   MX        10 junk.COM.
*.junk.COM.     IN   MX        10 junk.COM.

; junk.COM hosts
$INCLUDE /var/named/hosts/junk
```

Screen 15-21. Sample hosts and $INCLUDE Files For Server junk

The following is the **`hosts`** file for server `widget`, followed by its **`$INCLUDE`** file.

```
;
; widget zone hosts file for server widget
;
   IN     SOA       widget.junk.COM.   ralph.sysad.zap.junk.COM. (
         1.1       ;Serial
         10800     ;Refresh
         3600      ;Retry
         432000    ;Expire
         86400)    ;Minimum

    IN    NS        widget.junk.COM.
    IN    NS        junk.COM.
    IN    NS        zap.junk.COM.

; junk.COM hosts
$INCLUDE /var/named/hosts/widget
         .
         .
         .
; hosts in widget zone as listed in /var/named/hosts/widget

widget     A         223.100.101.1
           HINFO   "Sun 3/180"     UNIX
           MX        10 junk.COM.
whatsit    CNAME   widget.junk.COM
smelly     A         223.100.101.2
           HINFO   3B2             UNIX
           MX        10 junk.COM.
stinky     A         223.100.101.3
           HINFO   3B2             UNIX
           MX        10 junk.COM.
dinky      A         223.100.101.4
           HINFO   “Sun 3/160"     UNIX
           WKS       223.100.101.4     UDP who route
           WKS       223.100.101.4     TCP (echo telnet
                                     discard sftp
                                     uucp-path systat daytime netstat
                                     ftp finger domain nameserver)
           MX        10 junk.COM.

; all other hosts follow, up to 223.100.101.110
```

Screen 15-22. Sample hosts and $INCLUDE Files For Server widget

Reverse Addresses File

The following is the sample file of reverse addresses for hosts in the zone `junk`. Note that the name of the domain is fully qualified, so that the addresses of the hosts (without the network address) are sufficient in this case.

The reverse address files for servers `widget` and `zap` should be written in a manner similar to the file shown for server `junk`.

```
; reverse address file for server junk, in /var/named/junk.revzone

100.100.223.in-addr.arpa.  IN     SOA     junk.COM. ralph.sysad.zap.junk.COM.
(
                                1.1i;Serial
                                10800;Refresh
                                3600;Retry
                                432000     ;Expire
                                86400)     ;Minimum

                           IN     NS      junk.COM.
1                                 PTR     junk.COM.
2                                 PTR     lazy.junk.COM.
3                                 PTR     crazy.junk.COM.
4                                 PTR     hazy.junk.COM.

; all other hosts follow, up to 223.100.100.80
```

Screen 15-23. Sample Reverse Addresses File For Server junk

16
Troubleshooting and Tuning TCP/IP

16
Troubleshooting and Tuning TCP/IP

Introduction to TCP/IP Troubleshooting and Tuning

TCP/IP has several facilities you can use to troubleshoot problems and maximize system performance. These facilities include special TCP/IP commands and TCP/IP tunable parameters.

TCP/IP Troubleshooting Commands

TCP/IP includes several commands that help you troubleshoot, should you have problems with the network. These commands are:

- **`ping`**
- **`ifconfig`**
- **`netstat`**

The following sections describe these commands and give suggestions for using them.

The ping Command

The **`ping`** command offers the simplest way to find out if a host on your network is responding to TCP/IP packets or not. The basic syntax of **`ping`** is as follows:

`/usr/sbin/ping` *host* [*timeout*]

where *host* is the hostname of the machine in question. The optional *timeout* argument indicates the time, in seconds, for **`ping`** to keep trying to reach the machine (the default is 20 seconds). See **`ping(1M)`** for further information on the **`ping`** command.

When you run **`ping`**, the ICMP protocol sends a datagram to the host you specify, asking for a response. (Remember that ICMP is the protocol responsible for error handling on a TCP/IP network.)

Suppose you typed:

```
ping elvis
```

If host `elvis` is up, you would receive the following message:

```
elvis is alive
```

indicating that `elvis` responded to the ICMP request. However, if host `elvis` is down, or cannot receive the ICMP packets, you would receive the following message from **`ping`**:

```
no answer from elvis
```

If you suspect that a machine may be losing packets even though it is up, you can use the **`-s`** option of **`ping`** to try and detect the problem. For example, if you type:

```
ping -s elvis
```

ping sends packets to host `elvis` continuously, until you press the `Break` key or a time-out occurs. Sample output from **`ping -s`** elvis is shown in Screen 16-1.

```
PING elvis: 56 data bytes
64 bytes from elvis 129.144.50.21: icmp_seq=0. time=80. ms
64 bytes from elvis 129.144.50.21: icmp_seq=1. time=0. ms
64 bytes from elvis 129.144.50.21: icmp_seq=2. time=0. ms
64 bytes from elvis 129.144.50.21: icmp_seq=3. time=0. ms
         .
        .
        .
----elvis PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms) min/avg/max = 0/20/80
```

Screen 16-1. Sample Output From ping -s

Note that the statistical message appears after you press the Delete key, or a timeout is reached. The packet loss statistic indicates whether the host has dropped any packets.

NOTE

The nature of the TCP connection used for most client/server computing can cause local processes to hang while the remote machine to which the processes are connected is being reset. Since no packets can be sent to the local machine during the time it takes to reset the remote machine, the connection is maintained and the local process simply waits for incoming data.

There are advantages to maintaining the connection in this manner. For example, if a user does a **rlogin** to a remote machine and then goes away from their terminal, and if, during the time the user was away from their terminal the sole gateway to the remote machine is rebooted and brought back on line, when the user returns to their terminal the session with the remote machine would continue normally. This is because both sides of the connection were silent during the time the gateway was down. Since no packets were sent, no packets could be lost, and, consequently, there is no indication that the connection needs to be broken.

If **ping** fails, you can check the status of the network itself using the **ifconfig** and **netstat** commands. These commands are described in the following sections.

The ifconfig Command

The **ifconfig** command displays information about the configuration for a named network interface.

The **ifconfig** command has the following syntax:

```
/usr/sbin/ifconfig { -a | interface} [[address [dest_addr]] \
   [up] [down] [netmask mask] [arp | -arp] [metric n] \
   [trailers | -trailers][broadcast address] [protocol_family]
```

If you want to find out the configuration for all of the network devices that are currently installed, type:

```
ifconfig -a
```

You may receive a message like the one shown in Screen 16-2. Note that the figure shows sample **ifconfig** output for an Ethernet interface. When using **ifconfig** on a different type of interface, the characteristics of that interface type are displayed, and therefore different output is seen.

```
lo0: flags=449<UP,LOOPBACK,RUNNING,MULTICAST>
        inet 127.0.0.1 netmask ff000000
egl0: flags=c43<UP,BROADCAST,RUNNING,MULTICAST,MULTI_BCAST>
        inet 174.2.110.56 netmask ffff0000 broadcast 174.2.255.255
```

Screen 16-2. Sample Output from ifconfig -a

This tells you several things about the Ethernet interface. First, the `flags` section shows that the interface is up, that it allows broadcasts, that it is capable of using IP multicast, and that the interface is running with no problems. Information displayed on the second line includes the Internet address of the host you are using, the netmask currently being used, and the address of the network that broadcasts are sent over.

If you get **`ifconfig`** output that indicates an interface is not running, it might mean a problem with that interface. In this case, refer to the **`ifconfig(1M)`** manual page.

The netstat Command

The **`netstat(1M`**) command generates displays that show network status. You can run **`netstat`** to display the status of network traffic in table format, including routing table information (available routes and their status), and interface information.

`netstat` can display, according to the command line options chosen, one of the various network data structures available. These displays are the most useful for system administration. The syntax for this form is

`netstat [`**`-an`**`] [`**`-f`** *address-family*`] [`*system core*`]`

or

`netstat [`**`-mnrsv`**`] [`**`-f`** *address-family*`] [`**`-p`** *protocol*`] [`*system core*`]`

or

`netstat [`**`-gin`**`] [`**`-I`** *interface*`] [`*interval*`] [`*system core*`]`

The most frequently used network status options are **`-I`**, **`-s`**, **`-i`**, **`-r`**, and **`-rs`**. The **`-I`** option displays the number of packets sent and received on a given interface every *interval* seconds. For example, the following command would show the activity on the `imx586_0`driver every 5 seconds:

```
netstat -I ie0 5
```

The other **`netstat`** network status options are described in the following sections.

Displaying Per Protocol Statistics

The **`-s`** option displays per-protocol statistics for the UDP, TCP, ICMP, and IP protocols. To display statistics for all protocols, type:

```
netstat -s
```

To display statistics for a particular protocol, type:

```
netstat -p protocol
```

For example, typing:

```
netstat -p ip
```

produces statistics that look like the following:

```
ip:
                    91189 total packets received
                    215 bad IP headers
                    3761 fragments received
                    0 fragments dropped (dup or out of space)
                    0 fragments dropped after timeout
                    0 packets forwarded
                    285 packets not forwarded
```

Screen 16-3. Sample Output From netstat -p ip

The statistical information can show areas where a protocol is having problems. For example, statistical information from ICMP can indicate where this protocol has found errors.

Displaying Communications Controller Status

The **`-i`** option of **`netstat`** shows the state of the communications controllers that are configured with the machine where you ran the command. Here is a sample display produced by **`netstat -i`**:

```
Name     Mtu      Network Address  Ipkts    Ierrs    Opkts    Oerrs    Collis
ie0      1500     129.134.36cleo   13238710 0        15411607 107      0
lo0      2048     loopback localhost163101  0        63101    0        0
```

Screen 16-4. Sample Output From netstat -i

Using this display, you can find out how many packets a machine thinks it is transmitting and receiving on each network. For example, the input packet count (**`Ipkts`**) displayed for a server may increase each time a client tries to boot, while the output packet count (**`Opkts`**) remains steady. This suggests that the server is seeing the boot request packets from the client, but does not realize it is supposed to respond to them. This might be caused by an incorrect address in the `hosts` or `ethers` database.

On the other hand, if the input packet count is steady over time, it means that the machine does not see the packets at all. This suggests a different type of failure, possibly a hardware problem.

Displaying Routing Table Status

The **`-r`** option of **`netstat`** displays the IP routing table. Here is a sample display produced by **`netstat -r`** run on machine named `ballet`.

```
Routing tables
Destination        Gateway            Flags    Use Interface  Pmtu
*                  localhost          U          0 lo0        8232
localhost          localhost          UH         0 lo0        8232
eng1-00            eng1-00            UH         0 cnd00      8232
eng-main           eng-main           UH         0 ie0        8232
site2-eng          router1            UG         0 ie0        1500
test1              router1            UG         0 ie0        1500
mktng-1            router1            UG         0 ie0        1500
admin-pc           async-1            UH         3 ppp1       536
```

Screen 16-5. Sample Output from netstat -r

The first column shows the destination network, and the second column shows the router through which packets are forwarded. The `U` flag indicates that the route is up; the `G` flag indicates that the route is to a gateway. The `H` flag indicates that the destination is a fully qualified host address, rather than a network.

The `Use` column shows the number of packets sent per route and the `Interface` column shows the network interface that the route uses. The `Pmtu` (Path MTU) column indicates the maximum transmission unit for that path.

Adjusting Your Netmask to Correct Routing Problems

If you are unable to communicate with networks that your gateways can reach, you may need to adjust your netmask. This may be the case if a **`netstat -r`** command displays your local network with one or two numbers (such as `126` or `135.7`) when it should be three numbers (such as `126.7.9` or `135.7.234`). This could prevent you from exchanging broadcast messages with a gateway configured for a class C network mask with three numbers (such as `126.7.9` or `135.7.234`).

If your IP network address is a class A address (first number is between `1` and `127`) or a class B address (first number is between `128` and `191`), then you should try changing your netmask to class C. (If you are already using a class C netmask if the first number of your address is between `192` and `223` and you have not customized your netmask.) To adjust your netmask, you edit your Ethernet's IP address entry at the end of the file **`/etc/confnet.d/inet/interface`**. It is recommended that you make a copy of the **`interface`** file before you edit it.

The following example shows the address entry for an `ie0` Ethernet card. The entry should be changed from:

```
ie:0:cleo:/dev/ie0:arp
```

so that it becomes:

```
ie:0:cleo:/dev/ie0: arp netmask 255.255.0::
```

NOTE

There is no need to change the `localhost` entry in the **`interface`** file.

Your netmask update will take effect on reboot. The netmask will adjust what broadcast messages are sent and received by your machine, and allow your **`in.routed -q`** process to receive the correct information for updating the route table. If **`in.routed -q`** is unable to update the route table, you should refer to the instructions for static routing in Chapter 13, "Setting Up Routers and Subnetworks".

The netinfo Command

The **`netinfo(1M)`** command allows you to add, list, or remove entries from the **`/etc/confnet.d/netdrivers`** file [see **`netdrivers(4)`**]. The syntax of the **`netinfo`** command is:

```
netinfo [-l dev] [-l proto] [-d device] [-p protocol]
netinfo [-u -l dev] [-u -l proto]
netinfo [-a -d device] [-a -d device -p protocol]
netinfo [-r -d device][-r -p protocol] [-r -d device -p protocol]
```

Adding a Device Entry to the netdrivers File

To add an entry for a network card (device) to the **`netdrivers`** file, type:

```
netinfo -a -d device
```

where *device* is the name of the transport provider as it appears in the **`/dev`** directory.

Adding Device-to-Protocol Mapping Entry to the netdrivers File

To add a device-to-protocol mapping in the **`netdrivers`** file, type:

```
netinfo -a -d device -p protocol
```

where *device* is the name of the transport provider as it appears in the **`/devs`** directory and *protocol* is the name of a protocol that has been installed on the machine (for example, **`inet`**).

Listing Entries in the netdrivers File

There are a variety of ways that information in the **`netdrivers`** file can be displayed.

To list all of the devices that are installed in your machine, type:

```
netinfo -l dev
```

To list all of the protocols that are installed in your machine, type:

```
netinfo -l proto
```

To list all of the devices that are not mapped to any protocol(s), type:

```
netinfo -u -l dev
```

To list all of the protocols that are not mapped to any device(s), type:

```
netinfo -u -l proto
```

To show what protocol is mapped to a specific device, type:

```
netinfo -d device
```

where *device* is the name of the transport provider as it appears in the **/dev** directory.

To show what devices are mapped to a specific protocol, type:

```
netinfo -p protocol
```

where *protocol* is the name of a protocol that has been installed on the machine (for example, **inet**).

Removing Entries in the netdrivers File

There are three ways to properly remove information from the **netdrivers** file.

To remove a device from the **netdrivers** file, type:

```
netinfo -r -d device
```

where *device* is the name of the transport provider as it appears in the **/dev** directory.

To remove all device-to-protocol mapping for a specific protocol, type:

```
netinfo -r -p protocol
```

where *protocol* is the name of a protocol that has been installed on the machine (for example, **inet**).

To remove a specific device-to-protocol mapping, type:

```
netinfo -r -d device -p protocol
```

where *device* is the name of the transport provider as it appears in the **/dev** directory and *protocol* is the name of a protocol that has been installed on the machine (for example, **inet**).

Verifying the Contents of Protocol-Specific Interface File

Since **inet** is the only networking protocol supported under MP, the only interface file needing verification is in **/etc/confnet.d/inet/interface**.

For more information on the **/etc/confnet.d/inet/interface** file, see the INET-specific **interface(4)** manual page.

Running Software Checks

If there is trouble on the network, here are some actions you can take to diagnose and fix software-related problems.

1. Use the **netstat** command to determine network status as described previously.
2. Check the **/etc/hosts** file to make sure that the entries are correct and up to date.
3. Check the accuracy of the line(s) in the **/etc/confnet.d/inet/interface** file. See INET-specific **interface(4)** for the format of an entry in the interface file.
4. Try to **rlogin** to yourself by entering one of the following commands:
   - **rlogin \`uname -n\`**
   - **rlogin localhost**

   If the first bullet item is successful, then you are able to send TCP packets and process them on the **\`uname -n\`** machine. The \`uname -n\` system name is usually associated with first network device installed in the machine. If the first bullet item is successful and other machines cannot **rlogin** into the machine you are troubleshooting, then check the hardware and cables associated with the first network device.

   If the second bullet item is successful and the first is not, verify the `localhost` entry in the **/etc/confnet.d/inet/interface** file. Some configuration files are date dependent. If you have restored some old files or updated a number of files manually, type:

```
touch /etc/confnet.d/inet/interface
```

and reboot the machine so that the INET-specific files will be updated.

If you do not have an entry for yourself in your **`$HOME/.rhosts`** file, you will be prompted for your password.

5. Make sure the network daemon **`inetd`** is running. Then, log in and type:

```
ps -ef | grep inetd
```

The resulting display should resemble the following if the **`inetd`** daemon is running:

```
root     57       1        0        Apr 04 ? 3:19     /usr/sbin/inetd
root     4218     4198     0        17:57:23 pts/3    0:00     grep inetd
```

Screen 16-6. Sample Output From ps -ef | grep inetd

If **`inetd`** is not running, it may be restarted by typing:

```
sacadm -s -p inetd
```

or by rebooting the machine.

Troubleshooting Point-to-Point Protocol

The Point-to-Point Protocol (PPP) for using TCP/IP over serial connections is built on the **`uucp`** facility. If PPP appears to not be working, it may simply be that **`uucp`** or your modem are not configured properly. The following section helps you troubleshoot your PPP connections.

To check if PPP is able to reach the remote PPP system, type:

```
ping -s remote
```

where *remote* is replaced by the TCP/IP name of the remote PPP system. If you do not receive an acknowledgment that the remote PPP system is alive, check that you can contact the remote system using **`uucp`**.

To check your **`uucp`** connection to the remote system, type:

```
cu remote
```

where *remote* is the **`uucp`** name (in the **`/etc/uucp/Systems`** file) of the remote system. If you can establish a connection with the remote system, recheck your PPP configuration. If you cannot establish a connection, check the connection from the bottom up as described in the following paragraphs.

CAUTION

The **`cu`** command picks up the current **`stty`** settings when it tries to establish a connection with the remote system. This can cause undesirable results, such as changing your modem settings invisibly.

Checking Modems and Cables

Check the cables and modem settings you are using. If you are directly connecting two systems, make sure the cable is a null-modem cable. If you are using modems to connect, make sure that modem settings match on the remote and local sides. Line speed and parity are two important settings.

If you think you need to change modem settings, use the **`cu`** command to directly call the modem attached to your computer. The procedure for changing your modem settings is contained in the PPP section of the chapter “Setting Up TCP/IP.”

Before you use the **`cu`** command to connect to the modem, check the **`stty`** settings by typing:

```
stty -a
```

Once you connect to your local modem, list out the entire profile of the modem (if possible) and compare the settings.

If you feel the modems are set up correctly, try dialing directly across the connection. On a Hayes-compatible modem, type `ATDT` and the phone number to place the call.

Checking uucp Connection

If the modem or direct connection seem to be working properly, try calling across the connection again with the **`cu`** command. Running **`cu`** from different shells (the console, a Terminal window, or a remote login) can sometimes cause **`cu`** to fail, and possibly reset the modem settings.

If the **`cu`** works, but PPP still doesn't work, there may be a problem in the **`chat`** script you created for the remote system in the **`/etc/uucp/Systems`** file. This script must not only connect to the remote system, but it must also login as the PPP user. To check the script, type:

```
/usr/lib/uucp/Uutry -r remote
```

where *remote* is replaced by the remote **`uucp`** system name in your **`/etc/uucp/Systems`** file. This will actually connect to the remote system and log in based on the `chat` script in the **`Systems`** file.

If the remote system is very slow, it is possible that PPP could time out before the connection is completed. To avoid this, make sure the last part of the `chat` script matches some string that the remote system sends you after login is complete. For example, in the PPP procedure in the chapter “Setting Up TCP/IP” we recommend using the string `opyright` as the last part of the `chat` script. After logging into the remote system, the `dial` function (see the **`dial(3N)`** manual page) passes the connection to PPP after it sees the Copyright message after login.

If the full connection and login `chat` script seems to work once, but then not work, go back and check your modem settings again and see if **`uucp`** reset them automatically again. You may have to change them back again.

Checking PPP Connection

Once you are convinced that the **uucp** connection is working properly, try running **ping -s** *remote* to the remote PPP system again. If the call is going through, but the **ping** does not show packets received from the remote system, run the **ppstat** command.

The following are some of the errors that can occur with PPP that can be determined by running **pppstat**:

- Password authentication failure. If this occurs, make sure the passwords match on the local and remote systems.
- Received a packet with bad FCS. You received a packet with a checksum error. Try turning on hardware flow control on the connection. This may require requesting hardware flow control on one of the modem settings and using the hardware flow control device on your **uucp** connection. (For example, use **/dev/tty00h** instead of **/dev/tty00** for your connection.)
- Received packets with bad id field or loopback packets. Make sure the cable connection is correct. You may be receiving your own packets back, which could indicate that some of the pin connections on your cable are not correct.
- Received state table errors. This would indicate a PPP connection to a remote system that has a different and incompatible PPP implementation.

Another way to check the PPP connection is by typing:

```
netstat -i
```

Look for incoming and outgoing packet errors for your PPP connection in the **netstat** output. If there are no packets, the connection is not working. If there are lots of errors, you may simply have a dirty line.

Finally, you may want to check that the PPP interface was set up correctly in the first place. To do this, type:

```
ifconfig -a
```

The PPP interface should appear as one of the interfaces and should be listed as **UP**. If the PPP interface is not there, you may need to add it by typing:

```
configure -i.
```

If the PPP interface is there, but not up, type:

```
ifconfig ppp0 up
```

This would make the first PPP interface on your computer (`ppp0`) available for use.

Restarting TCP/IP

If you are a privileged user and need to stop and restart TCP/IP, use the following procedure.

CAUTION

The following should only be done if you are willing to lose all current network connections. You should only stop TCP when debugging the interface or other boot sensitive files. Make sure that any remotely mounted or shared resources are not in use when you do the following procedure.

1. To stop TCP/IP, do the following:
   - If you are running NFS, stop NFS by typing:

```
sh -x /etc/init.d/nfs stop
```

   - Stop other processes using TCP/IP.
   - Kill **`inetd`** and the TCP/IP listener by typing the commands:

```
sacadm -k -p inetd
sacadm -k -p tcp
```

- Shut down and unlink STREAMS by typing:

```
sh -x /etc/init.d/inetinit stop
```

2. To restart TCP/IP, do the following:

   - Start and link the STREAMS by typing:

```
sh +x /etc/init.d/inetinit start
```

- Restart **`inetd`** and the listener by typing the commands:

```
sacadm -s -p inetd
sacadm -s -p tcp
```

- If you use NFS on your system, restart NFS typing:

```
sh -x /etc/init.d/nfs start
```

Improving System Performance

If you are having performance problems, your problems may be the result of running the daemons **`routed`** and **`rwhod`**.

The system status daemon **`rwhod`** is not run by default, because it can seriously degrade system and network performance. You are strongly advised not to run it.

The routing daemon **`routed`** is run by default. If your machine has limited memory and there are no routers on your network, then you do not need to run **`routed`**. You can disable **`routed`** by commenting out (inserting the “#” character) the line:

```
/usr/sbin/in.routed -q
```

in the start-up script **`/etc/inet/rc.inet`**.

If your machine has limited memory and only one router in your network, then you can run **`routed`** only on the router, and disable it on all other machines.

For more information about configuring a router, see Chapter 13, "Setting Up Routers and Subnetworks".

Increasing System Determinism

Concurrent’s TCP/IP implementation is STREAMS based. As such, TCP/IP STREAMS service procedures may usually be executed on any CPU in the system. These STREAMS service procedure executions can cause a reduction in the determinism of applications when and if those applications are preempted by the kernel to execute queued STREAM’s service procedures.

On systems that are configured for real-time applications, it may be desirable to restrict the CPUs that may execute STREAMS service procedures. It may also be necessary to enable a kernel feature that provides a STREAMS service procedure CPU bias mask on a per-STREAM basis in order to provide even greater control over which CPU(s) execute each STREAM’s service procedures. For more information on this subject, see the section “Controlling STREAMS Scheduling” in the *PowerMAX OS Real-Time Guide* (Pubs No. 0890466).

Tuning TCP/IP Kernel Parameters

To maximize your system resources, you can modify three types of TCP/IP tunable parameters:

- General-purpose TCP/IP parameters are located in the **`/etc/conf/cf.d/stune`** file.
- Tunable parameters associated with allocating and initializing configuration-dependent data structures for the individual TCP/IP kernel modules are located in **`Space.c`** files in subdirectories of **`/etc/conf/pack.d`**.
- Tunable parameters used by ARP, IP, RAWIP, UDP and TCP accessible via the **`ndd`** program.

TCP/IP tunable parameters in the **stune** file should not be edited directly. To change the value of a parameter in the **stune** file, you should use the **config(1M)** program. For more information see the chapter “Tunable Parameters” in the guide *System Administration.* You may also want to refer to the **stune(4)** manual page.

TCP/IP tunable parameters in **Space.c** files may be edited directly. The parameters with upper case names are defined constants; those with lower case names are variables. To change a defined constant, you can change the constant itself. To change a variable, you change the value the variable is set to. For information about the **Space.c** file, see the **Space.c(4)** manual page.

TCP/IP tunable parameters can be set and retrieved using a set of functions called Named Dispatch. An interface to the Named Dispatch is the **ndd** program.

The **ndd** program provides a simple command line interface to the Named Dispatch facility. The program accepts arguments from the command line or may be run interactively. The general form of the command line is:

```
ndd [-set| -get] device_name ndd_name [value]
```

The tunable parameters accessed by the **ndd** command are for advanced users. Please see the *TCP/IP Program Interfaces Guide* (0890468) for more information.

Note that once you modify a TCP/IP tunable parameter in the **stune** file or in a **Space.c** file, you need to rebuild the kernel and then reboot the system so that your changes will take effect. See the **idbuild(1M)** manual page for more information.

The following sections describe the tunable parameters for the TCP/IP kernel modules. Each module's **stune** file parameters (if any) are described first, followed by its **Space.c** file parameters (if any).

asyhdlc Tunable Parameters

This section describes the tunable parameters for the asynchronous HDLC protocol driver `asyhdlc.`

asyhdlc stune Parameters

`ASYHMTU`	Specifies the default packet size used during connection initialization. The value of this parameter will be overridden by PPP option negotiation. It is recommended that the value be equal to a power of 2 plus 40 bytes for the IP header.

ip Tunable Parameters

This section describes the tunable parameters for the Internet Protocol driver **ip**.

ip stune Parameters

`IPFORWARDING`	Controls whether **`ip`** will forward packets (in other words, whether it will act as a gateway). This should be turned on (non-zero) only if the machine is intended to be a gateway.

ip ndd Parameters

Please see the *TCP/IP Program Interfaces Guide* for **`ip`** **ndd** parameters.

ppp Tunable Parameters

This section describes the tunable parameters for the Point-to-Point Protocol driver **ppp**.

ppp Space.c Parameters

`PPPDEVCNT`	Specifies the maximum number of connections between **`ip`** and **ppp**. Each configured **ppp** host requires one connection.
`PPPHIWAT`	Establishes the STREAMS high water mark for the **ppp** driver.
`PPPMTU`	Specifies the default packet size used by **ppp** for both transmitting and receiving. The value of this parameter may also be changed through **ppp** configuration (see the **`pppconf(1M)`** manual page) and by **ppp** option negotiation. It is recommended that the value be equal to a power of 2 plus 40 bytes for the IP header.
`PPPWFACK`	Specifies the configuration acknowledgment timeout value in seconds. The value of this parameter may also be changed through **ppp** configuration (see the **`pppconf(1M`**) manual page).
`PPPCNFRETRIES`	Specifies the number of configuration acknowledgment retries to attempt. The value of this parameter may also be changed through **ppp** configuration (see the **`pppconf(1M)`** manual page).

arp, rawip, tcp, and udp Tunable Parameters

The current **`arp`**, **`rawip`**, **`tcp`**, and **udp** tunable parameters can only be accessed via the **ndd** program. Please see the *TCP/IP Program Interfaces Guide* for further information about these tunable parameters.

Logging Network Problems

If you suspect a routing daemon malfunction, you may log its actions—and even all the packet transfers. To create a log file of routing daemon actions, you simply supply a file name when you start up the **`routed`** daemon in **`/etc/inet/rc.inet`**.

CAUTION

On a busy network this generates almost constant output.

For example:

```
/usr/sbin/in.routed -q /var/routerlog
```

Whenever a route is added, deleted, or modified, a log of the action and a history of the previous packets sent and received will be printed in the log file.

To force full packet tracing, specify the **`-t`** option when the **`routed`** daemon is started up.

For more information, refer to the **`routed(1M)`** manual page.

17
Obtaining IP Addresses

17 Obtaining IP Addresses

IP Number Registration Form

The following questions appear on the IP Number Registration Form provided by InterNIC Registration Services. You can obtain the form by anonymous FTP from RS.INTERNIC.NET (file **`templates/internet-number-template.txt`**), or by submitting an electronic mail request to HOSTMASTER@INTERNIC.NET, or, if FTP and electronic mail is not available to you, by calling the NIC at (800) 444-4345, or writing to:

Network Solutions
InterNIC Registration Services
505 Huntmar Park Drive
Herndon, VA 22070

1. If the network will be connected to the Internet, you must provide the name of the governmental sponsoring organization, and the name, title, mailing address, phone number, net mailbox, and NIC Handle (if any) of the contact person (POC) at that organization who has authorized the network connection. This person will serve as the POC for administrative and policy questions about authorization to be a part of the Internet. Examples of such sponsoring organizations are DISA DNSO, the National Science Foundation (NSF), or similar military or government sponsors.

NOTE

If the network will not be connected to the Internet, then you do not need to provide this information.

Example:

```
1a. Sponsoring Organization:              DARPA
1b. Contact name (Lastname, Firstname):          Jones, John
1c. Contact title:              Program Manager
1d. Mail Address:               DARPA/ISO Office
                                1400 Wilson Boulevard
                                Arlington, VA 22209
1e. Phone:          (609) 555-1212
1f. Net mailbox:                progmgr@VAX.DARPA.MIL
1g. NIC handle (if known):                AA12
```

2. Provide the name, title, mailing address, phone number, and organization of the technical POC. The online mailbox and NIC Handle (if any) of the technical POC should also be included. This is the POC for resolving technical problems associated with the network and for updating information about the network. The technical POC may also be responsible for hosts attached to this network.

   Example:

2a. NIC handle (if known):	CC56
2b. Technical POC name (Lastname, Firstname):	Jones, Jenny
2c. Technical POC title:	Computer Scientist
2d. Mail address:	SRI International 14200 Park Meadow Dr. Chantilly, VA 22021
2e. Phone:	(908) 555-1212
2f. Net Mailbox:	cs@SRI-NIC.ARPA

3. Supply the short mnemonic name for the network (up to 12 characters). This is the name that will be used as an identifier in Internet name and address tables.

   Example:

   3. Network name: ALPHA-BETA

4. Identify the network geographic location and the responsible organization establishing the network.

   Example:

   4a. Postal address for main/headquarters network site:
   Socrates Enterprises
   P.O. Box 999999
   Sandy, Utah 84094
   4b. Name of Organization: Socrates Enterprises

5. Question #5 is for military or DOD requests only. If you require that this connected network be announced to the NSFNET please answer questions 5a, 5b, and 5c.

   Example:

   5a. Do you want MILNET to announce your network to the NSFNET?
   (Y/N): N

   5b. Do you have an alternate connection, other than MILNET, to the NSFNET?
   (please state alternate connection if answer is yes): N

   5c. If you've answered yes to 5b, please state if you would like the
   MILNET connection to act as a backup path to the NSFNET? (Y/N): N

6. Estimate the number of hosts that will be on the network.

   Example:

6a. Initially:	5
6b. Within one year:	25
6c. Within two years:	50
6d. Within five years:	200

7. Unless a strong and convincing reason is presented, the network (if it qualifies at all) will be assigned a Class C network number. If a Class C network number is not acceptable for your purposes, state why. Note: If there are plans for more than a few local networks, and more than 100 hosts, you are strongly urged to consider subnetting (See RFC 950).

   Example:

   7. Reason: Class C is fine

8. Networks are characterized as being either Research, Defense, Government - Non Defense, or Commercial, and the network address space is shared among these four areas. Which type is this network?

   Example:

   8. Type of network: Research

9. What is the purpose of the network?

   Example:

   9. Purpose: To economically connect computers used
   in DARPA sponsored research project FROB-BRAF
   to the DARPA Internet or the DDN Internet
   to provide communication capability
   with other similar projects and UNIV-X and CORP-Y.

Additional Reading

The following recommended reading is available from the InterNIC Registration Services.

Bjork, S.; Marine, A., eds. "*Network Protocol Implementations and Vendors Guide*" Menlo Park, CA: SRI International, DDN Network Information Center; 1990 August; NIC 50002 (August 1990). 242 p. (NIC.DDN.MIL NETINFO:VENDORS-GUIDE.DOC).

Braden, R.T.; Postel, J.B. "*Requirements for Internet Gateways*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 June; RFC 1009. 55 p. (RS.INTERNIC.NET POLICY RFC1009.TXT).

Defense Advanced Research Projects Agency, Internet Activities Board. "*IAB Official Protocol Standards*" 1991 April; RFC 1200. 31 p. (RS.INTERNIC.NET POLICY

RFC1200.TXT). Feinler, E.J.; Jacobsen, O.J.; Stahl, M.K.; Ward, C.A., eds. "*DDN Protocol Handbook*" Menlo Park, CA: SRI International, DDN Network Information Center; 1985 December; NIC 50004 and NIC 50005 and NIC 50006. 2749 p.

Garcia-Luna-Aceves, J.J.; Stahl, M.K.; Ward, C.A., eds. "*Internet Protocol Handbook*" The Domain Name System (DNS) Handbook. Menlo Park, CA: SRI International, Network Information Systems Center; 1989 August; NIC 50007. 219 p. AD A214 698.

Kirkpatrick, S.; Stahl, M.K.; Recker, M. "*Internet Numbers*" Menlo Park, CA: SRI International, DDN Network Information Center; 1990 July; RFC 1166. 182 p. (RS.INTERNIC.NET POLICY RFC1166.TXT).

Mogul, J.; Postel, J.B "*Internet Standard Subnetting Procedure*". Stanford, CA: Stanford University; 1985 August; RFC 950. 18 p. (RS.INTERNIC.NET POLICY RFC950.TXT).

Postel, J.B. "*Internet Control Message Protocol*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 792. 21 p. (RS.INTERNIC.NET POLICY RFC792.TXT).

Postel, J.B. "*Transmission Control Protocol*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 793. 85 p. (RS.INTERNIC.NET POLICY RFC793.TXT).

Postel, J.B. "*Address Mappings*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 796. 7 p. (RS.INTERNIC.NET POLICY RFC796.TXT). Obsoletes: IEN 115 (NACC 0968-79)

Postel, J.B. "*User Datagram Protocol*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1980 August 28; RFC 768. 3 p. (RS.INTERNIC.NET POLICY RFC768.TXT).

Postel, J.B. "*Internet Protocol*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 791. 45 p. (NIC.DDN.MIL RFC:RFC791.TXT).

Reynolds, J.K.; Postel, J.B. "*Assigned Numbers*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1990 March; RFC 1060. 86 p. (RS.INTERNIC.NET POLICY RFC1060.TXT).

Reynolds, J.K.; Postel, J.B. "*Official Internet Protocols*" Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 May; RFC 1011. 52 p. (RS.INTERNIC.NET POLICY RFC1011.TXT).

18
Obtaining Domain Names

18 Obtaining Domain Names

Domain Registration Forms

To establish a domain, there are two forms you must complete and send to the NIC. The first form registers your domain and the second form registers your address-to-host lookup information. These two forms are described in the following sections.

Registering Your Domain Name

The following information must be sent to register your domain name with the NIC Domain Registrar. Questions may be addressed to the NIC Hostmaster by electronic mail at `HOSTMASTER@SRI-NIC.ARPA`, or by phone at (415) 859-3695 or (800) 235-3155. You can obtain the domain registration form by anonymous FTP from `RS.INTERNIC.NET` (file **`templates/domain-template.txt`**), or by submitting an electronic mail request to `HOSTMASTER@INTERNIC.NET`, or, if FTP and electronic mail are not available to you, by calling the NIC at (800) 444-4345, or by writing to:

Network Solutions
InterNIC Registration Services
505 Huntmar Park Drive
Herndon, VA 22070

NOTE

The key people must have electronic mailboxes and NIC "handles," unique NIC database identifiers. If you have access to "WHOIS," please check to see if you are registered and if so, make sure the information is current. Include only your handle and the changes (if any) that need to be made in your entry. If you do not have access to WHOIS, please provide all the information indicated and a NIC handle will be assigned.

Completing the Domain Registration Form

The following is an example of filling out the NIC Domain Registration Form.

1. The name of the top-level domain to join (EDU, COM, MIL, GOV, NET, ORG).

   Example:

Top-level domain:	COM

2. The name of the domain (up to 12 characters). This is the name that will be used in tables and lists associating the domain with the domain server addresses. (While, from a technical standpoint, domain names can be quite long we recommend the use of shorter, more user-friendly names.)

   Example:

Complete Domain Name:	TNC

3. The name and address of the organization establishing the domain.

   Example:

Organization name:	The NetWorthy Corporation
Organization address:	The NetWorthy Corporation
	4676 Andrews Way, Suite 100
	Santa Clara, CA 94302-1212

4. The date you expect the domain to be fully operational.

   Example:

Date operational:	September 9, 1999

5. The NIC handle of the administrative head of the organization, or this person's name, mailing address, phone number, organization, and network mailbox. This is the contact point for administrative and policy questions about the domain. In the case of a research project, this should be the principal investigator. Both the Administrative and Technical/Zone contact of a domain must have a network mailbox, even if the mailbox is to be within the proposed domain.

   Example:

Administrative Contact	
Handle (if known):	PQS
Name (Last, First):	Sassafrass, Penelope Q.
Organization:	The NetWorthy Corporation
Mail Address:	The NetWorthy Corporation
	4676 Andrews Way, Suite 100
	Santa Clara, CA 94302-1212
Phone Number:	(801) 555-1212
Net Mailbox:	Sassafrass@ECHO.TNC.COM NIC

6. The NIC handle of the technical contact for the domain, or the person's name, mailing address, phone number, organization, and network mailbox. This is the contact point for problems concerning the domain or zone, as well as for updating information about the domain or zone.

   Example:

   Technical and Zone Contact

Handle (if known):	AAA2
Name (Last, First):	Aardvark, Ansel A.
Organization:	The NetWorthy Corporation
Mail Address:	The NetWorthy Corporation 4676 Andrews Way, Suite 100 Santa Clara, CA. 94302-1212
Phone Number:	(408)555-1212
Net Mailbox:	Aardvark@ECHO.TNC.COM

7. Domains must provide at least two independent servers on Government-sponsored networks that provide the domain service for translating names to addresses for hosts in this domain.

   If you are applying for a domain and network number assignment simultaneously, and a host on your proposed network will be used as a server for the domain, you must wait until you receive your network number assignment and have given the server(s) a netaddress before sending in the domain application. Sending in the domain application without providing complete information in Sections 7 and 8 of this template will result in the delay of the domain registration.

   Also, establishing the servers in physically separate locations and on different PSNs and/or networks is strongly recommended.

   NOTE

   All new hosts acting as servers will appear in the DNS root servers but will not appear in the **`HOSTS.TXT`** file unless otherwise requested.

   The preferred format for this information is:

   Primary Server: HOSTNAME, NETADDRESS, HARDWARE, SOFTWARE

   Example:

Primary Server Hostname:	BAR.FOO.COM
Primary Server Netaddress:	10.9.0.13
Primary Server Hardware:	VAX-11/750
Primary Server Software:	UNIX

8. The Secondary server information.

   Example:

Secondary Server Hostname:	XYZ.ABC.COM
Secondary Server Netaddress:	128.4.2.1
Secondary Server Hardware:	IBM-PC
Secondary Server Software:	MS-DOS

9. If any currently registered hosts will be renamed into the new domain, please specify old hostname, netaddress, and new hostname.

   Example:

   BAR-FOO2.ARPA (10.8.0.193) -> FOO2.BAR.COM
   BAR-FOO3.ARPA (10.7.0.193) -> FOO3.BAR.COM
   BAR-FOO4.ARPA (10.6.0.193) -> FOO4.BAR.COM

10. Please describe your organization briefly.

    Example:

    The NetWorthy Corporation is a consulting organization of people working
    with the UNIX system
    and the C language
    in an electronic networking environment.
    It sponsors two technical conferences annually
    and distributes a bimonthly newsletter.

NOTE

HOSTS.TXT whose names will change as a result of this domain registration. (Also be sure to answer question # 7, above.)

Registering for Internet Address to Host Mapping

The Internet uses a special domain to support gateway location and Internet address to host mapping. The intent of this domain is to provide a guaranteed method to perform host address to host name mapping, and to facilitate queries to locate all gateways on a particular network in the Internet.

The following information must be sent to the NIC Domain Registrar (`HOSTMASTER@SRI-NIC.ARPA`). Questions may be addressed to the NIC Hostmaster by electronic mail at the above address, or by phone at (415) 859-3695 or (800) 235-3155. You can obtain the Internet address to host mapping form by anonymous FTP from `RS.INTERNIC.NET` (file **`templates/in-addr-template.txt`**), or by submitting an electronic mail request to `HOSTMASTER@INTERNIC.NET`, or, if FTP and electronic mail are not available to you, by calling the NIC at (800) 444-4345, or by writing to:

Network Solutions
InterNIC Registration Services
505 Huntmar Park Drive
Herndon, VA 22070

The following information is needed for delegation of registered networks in your domain for inclusion in the `IN-ADDR.ARPA` zone files:

- The `IN-ADDR.ARPA` domain
- The Network name
- The Hostnames of the two hosts on networks that will be acting as `IN-ADDR` servers. `IN-ADDR` domains are represented using the network number in reverse. For example, network `123.45.67.0`'s `IN-ADDR` domain is represented as `67.45.123.IN-ADDR.ARPA`.

  Example:

IN-ADDR domain	Network NameI	N-ADDR Servers
(Hostname)		
(NetAddress)		
(CPUType/OpSys)		
41.192.IN-ADDR.ARPA	NET-TEST-ONE	BAR.FOO.EDU
123.45.67.89		
VAX-II/VMS		
ONE.ABC.COM		
98.76.54.32		
PowerMAX OS		

NOTE

Unless specified, new hosts registered as `IN-ADDR` servers will be registered in the root servers only and will not appear in the **`HOSTS.TXT`** file.

Please have the Network Coordinator complete and return the following information for those networks needing `IN-ADDR` registration.

IN-ADDR domain   Network Name   IN-ADDR Servers

Additional Reading

Feinler, E.J.; Jacobsen, O.J.; Stahl, M.K.; Ward, C.A., eds. *DDN Protocol Handbook* Menlo Park, CA: SRI International, DDN Network Information Center; 1985 December; NIC 50004 and NIC 50005 and NIC 50006. 2749 p.

Garcia-Luna-Aceves, J.J.; Stahl, M.K.; Ward, C.A., eds. *Internet Protocol Handbook: The Domain Name System (DNS) Handbook* Menlo Park, CA: SRI International, Network Information Systems Center; 1989 August; 219 p. AD A214 698.

Postel, J.B.; Reynolds, J.K. *Domain Requirements* Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1984 October; RFC 920. 14 p. (RS.INTERNIC.NET POLICY RFC920.TXT).

Harrenstien, K.; Stahl, M.K.; Feinler, E.J. *DoD Internet Host Table Specification* Menlo Park, CA: SRI International, DDN Network Information Center; 1985 October; RFC 952. 6 p. (RS.INTERNIC.NET POLICY RFC952.TXT). Obsoletes: RFC 810

Harrenstien, K.; Stahl, M.K.; Feinler, E.J. *Hostname Server* Menlo Park, CA: SRI International, DDN Network Information Center; 1985 October; RFC 953. 5 p. (NIC.DDN.MIL RFC:RFC953.TXT). Obsoletes: RFC 811

Partridge, C. *Mail Routing and the Domain System* Cambridge, MA: BBN Labs., Inc.; 1986 January; RFC 974. 7 p. (RS.INTERNIC.NET POLICY RFC974.TXT).

Lazear, W.D. *MILNET Name Domain Transition* McLean, VA: MITRE Corp.; 1987 November; RFC 1031. 10 p. (RS.INTERNIC.NET POLICY RFC1031.TXT).

Stahl, M.K. *Domain Administrators Guide* Menlo Park, CA: SRI International, DDN Network Information Center; 1987 November; RFC 1032. 14 p. (RS.INTERNIC.NET POLICY RFC1032.TXT).

Lottor, M. *Domain Administrators Operations Guide* Menlo Park, CA: SRI International, DDN Network Information Center; 1987 November; RFC 1033. 22 p. (RS.INTERNIC.NET POLICY RFC1033.TXT).

Mockapetris, P. *Domain Names - Concepts and Facilities* Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 November; RFC 1034. 55 p. (RS.INTERNIC.NET POLICY RFC1034.TXT). Updated-by: RFC 1101 Obsoletes: RFC 973; RFC 882; RFC 883

Mockapetris, P. *Domain names - Implementation and Specification* Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 November; RFC 1035. 55 p. (RS.INTERNIC.NET POLICY RFC1035.TXT). Updated-by: RFC 1101 Obsoletes: RFC 973; RFC 882; RFC 883

Mockapetris, P. *DNS Encoding of Network Names and Other Types* Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1989 April; RFC 1101. 14 p. (RS.INTERNIC.NET POLICY RFC1101.TXT). Updates: RFC 1034; RFC 1035

19
Network Time Synchronization

19
Network Time Synchronization

Introduction to Network Time Synchronization Administration

Time synchronization utilities provided with TCP/IP can be used to keep the clocks synchronized among all the computers on your network.

- Timekeeping synchronization among a set of time servers and clients can be performed using the Network Time Protocol (NTP) daemon **`xntpd`**.
- Clock synchronization and dynamic selection of a time server when the master time server is not available can be performed using the time server daemon **`timed`**.

Normally, you would only use one of the two utilities (you shouldn't use both utilities at the same time). This chapter describes both **`xntpd`** and **`timed`**.

Administering the xntpd Network Time Protocol Daemon

`xntpd` is a complete implementation of the Network Time Protocol (NTP), Version 2, and is also compatible with NTP Version 1. NTP is used to synchronize timekeeping among a set of distributed time servers and clients. NTP does this by synchronizing the clock to Coordinated Universal Time (UTC) using the best available source.

To take full advantage of NTP you will need access to a good source of UTC. If you have access to the Internet, then see the section “The clock.txt File” on page 19-7 for information about finding NTP servers on the Internet.

This chapter describes:

- guidelines for configuring **`xntpd`**
- how to configure **`xntpd`**
- how NTP operates conceptually
- advanced features of **`xntpd`**
- where to find more information on existing NTP servers
- and full configuration examples

Configuring Clock Synchronization

Before you configure a host to use **`xntpd`**, it is best that you understand how NTP works.

If you are configuring a single NTP host, you need to locate three servers to obtain time from. Ideally you would pick two primary servers and one secondary server to poll. It is best to choose servers that are close to the machine in terms of network topology, but don't worry if you are unable to determine which server is close and which is not.

If, however, your intention is to synchronize a larger number of hosts via NTP, then you will want to synchronize a few hosts to primary servers. Then you synchronize some of the rest of your hosts to these secondary servers. Some good guidelines to follow are:

- Choose three of your local hosts to operate as stratum 2 servers. Then choose six stratum 1 servers and configure each of your stratum 2 servers to peer with a pair of these. If you can, try to ensure that each stratum 2 server has at least one close stratum 1 peer. In addition, peer each of the stratum 2 servers with the other two (each configuration will list four peers, two remote, two local).
- From the remaining hosts, choose those you would like to operate at stratum 3. This might be all of the rest if you want to synchronize at most a couple of dozen hosts, or it might be things like file servers and machines with good clocks if you want to synchronize many hosts. Configure each stratum 3 server to peer with the three stratum 2 servers.
- Configure all other hosts to peer with three or four of the above nearby stratum 3 servers.
- If you have a radio clock, simply turn one of the stratum 2 servers mentioned above into a stratum 1 server by attaching the radio clock. (This version of **`xtnpd`** doesn't provide direct support for radio clocks at this time.) Leaving the associations with the other servers intact will provide a good backup in case the clock dies.
- Avoid configuring servers above stratum 4. Generally, topologies with strata greater than 4 are unnecessarily complex and are harder to manage.
- Don't synchronize a server to a same stratum peer unless the latter is receiving time from lower stratum sources the former doesn't talk to directly.
- Don't configure peer associations with higher stratum servers. Let the higher strata configure lower stratum servers.

The above arrangement provides a good robust time service with minimal traffic to distant servers and with manageable loads on all local servers. A server with a stratum above 15 may be unreachable because of loops in the topology. Following the above guidelines helps prevent loops in the topology.

Time Synchronization Example

In the following example:

- `Zite`, `HGWells`, and `DrWho` are stratum 2 servers for `Sci.com`.
- `Zite` peers with two stratum 1 servers: `percival.cs.purdue.edu` and `deidinator.psyc.chi.il.us`.
- `HGWells` peers with two stratum 1 servers: `asmus1.genetics.uga.edu` and `bedivere.cs.purdue.edu.`
- `DrWho` peers with two stratum 1 servers: `nehd.gov` and `berdie.athens.ga.us`.
- `Zite` peers with `HGWells` and `DrWho`.
- `HGWells` peers with `Zite` and `DrWho`.
- `DrWho` peers with `Zite` and `HGWells`.
- `moondoggie` and the rest of the machines at `sci.com` are at stratum 3 and peer with the three stratum 2 machines.

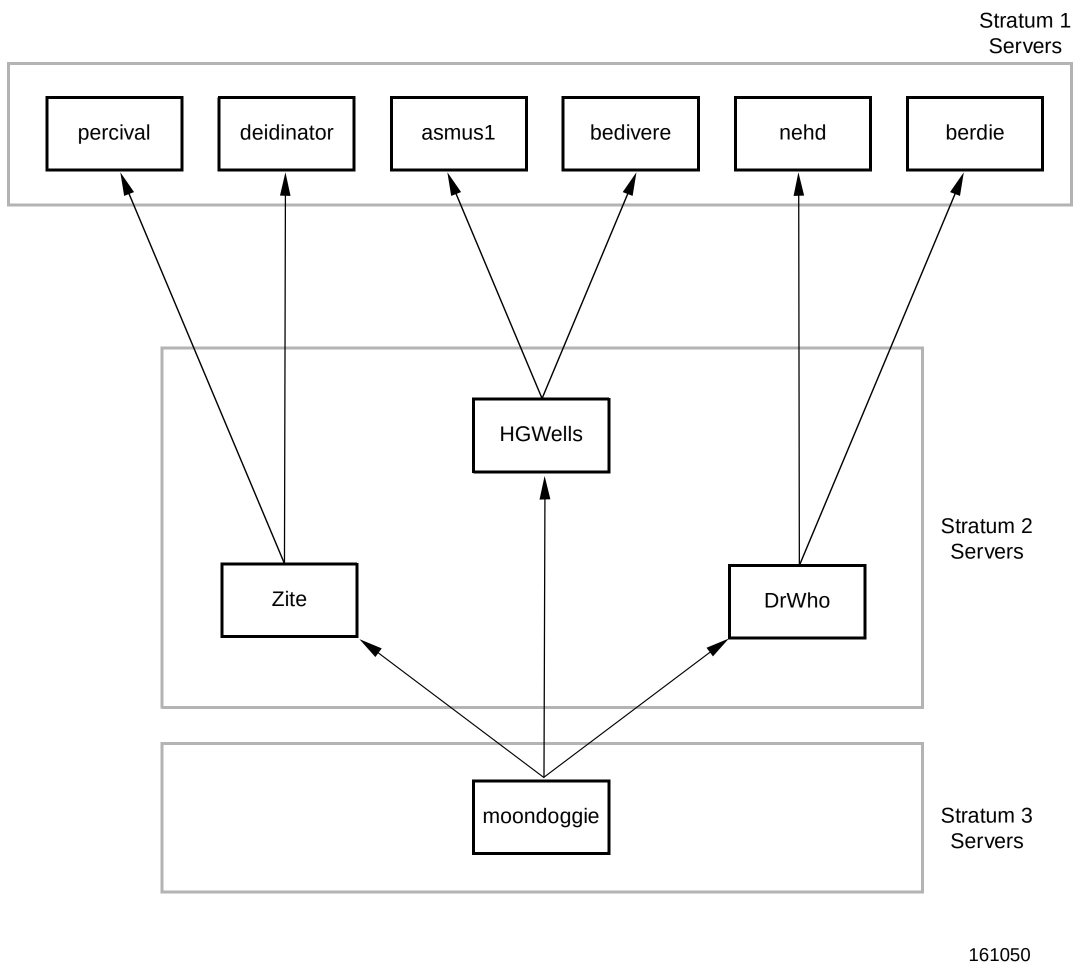


Figure 19-1. Time Synchronization Example

Configuring NTP With ntp.conf

The file **`ntp.conf`** contains information **`xntpd`** uses at startup, including the peers of this server, host restrictions, whether to broadcast NTP packets or not, whether to listen for broadcast NTP packets or not, what the roundtrip delay is for a peer whose broadcast packets are being listened to, where the `driftfile` is located, where the key ID file is located, whether to monitor NTP connections or not, whether or not to allow run-time reconfiguration of server, and what peer selection algorithm to use.

The configuration file is relatively free format. Comments begin with the character "#" and extend to the end of the line. Comments may be inserted freely. Blank lines are ignored. Configuration statements include an initial keyword followed by white space separated arguments. Configuration statements may not be split over multiple lines. A list of the configuration statements with their arguments are listed below. Optional arguments are delimited by square brackets ([]), while alternatives are separated by pipe signs (|).

Configuration Statements

`peer` *host_address* [`key` *keyid*] [`minpoll`]
: Specifies that the named host is to be polled in symmetric active mode. *host_address* is the IP address of the host to poll. The argument `key` indicates that all packets sent to the address are to include authentication fields encrypted using the key number *keyid*. The default is to not include authentication fields. The argument `minpoll` indicates that the polling interval should be kept at the minimum. This argument should only be used for debugging purposes.

`server` *host_address* [`key` *keyid*] [`minpoll`]
: Specifies that the host at the given *host_address* is to be polled in client mode. The arguments are the same as described for the `peer` statement above.

`broadcast` *host_address* [`key` *keyid*] [`minpoll`]
: Specifies that **`xntpd`** is to broadcast NTP to the specified *host_address*. *host_address* should be the broadcast address on (one of) your local network(s). The arguments are the same as described for the `peer` statement above.

`precision` *precision_value*
: Indicates the precision of the local timekeeping. See the section “Testing and Tuning Your NTP Subnet” on page 19-18 for further information on setting the precision.

`driftfile` *filename*
: Specifies that the file *filename* is to be used to record the drift (frequency error) computed by **`xntpd`**.

`monitor yes | no`
: Indicates whether or not **`xntpd`** monitoring function should be enabled. The default is `no`.

`broadcastclient yes | no`
: Indicates whether or not the local server should listen for, and attempt to synchronize to, broadcast NTP. The default is `no`.

`broadcastdelay` *seconds*
: Specifies the round trip delay to the host whose broadcasts are being synchronized to, where *seconds* is the number of seconds to delay. The default is 0.008 seconds.

`authenticate yes | no`
: Indicates whether or not the local server operates in authenticate mode. The default is `no`.

`authdelay` *seconds*
: Indicates the amount of time it takes to encrypt an NTP authentication field on the local computer, where *seconds* is the number of seconds to delay.

`keys` *filename*
: Specifies the name of the file that contains the encryption keys to be used by **`xntpd`**.

`trustedkey` *keyid* [...]
: Specifies which encryption key IDs are the trusted ones for the purpose of determining peers suitable for time synchronization, when authentication is enabled.

`requestkey` *keyid*
: Specifies which key ID is to be used to authenticate run time reconfiguration requests made by **`xntpdc`** (mode 7 control messages). If no `requestkey` statement is included, the run-time reconfiguration facility is disabled.

`controlkey` *keyid*
: Specifies which key ID is to be used to authenticate run time reconfiguration requests made by **`ntpq`** (mode 6 control messages). If no `controlkey` statement is included, then mode 6 control messages are ignored.

`restrict` *address* [*mask*] [*flag*] [...]
: Specifies which hosts and networks are allowed to interoperate with the daemon.

`trap` *host_address* [port *port_number*] [`interface` *interface_address*]
: Configures a trap receiver at the given host address and port number.

`maxskew` *seconds*
: Sets the system maximum skew parameter to *seconds*.

`select` *algorithm_number*
: Selects one of five selection algorithms. The default is `1`.

`resolver` *path*/`xntpres`
: Indicates the full path to the **`xntpres`** program. (This should be set to **`/usr/sbin/xntpres`**.)

Example ntp.conf File

Here is an example of a minimal **`ntp.conf`** file for the host `zite.sci.com`. The four `peer` declarations tell **`xntpd`** the servers it needs to poll. The last declaration tells **`xntpd`** the name of the file in which to store the frequency error of the system clock. For now, suffice it to say that every **`ntp.conf`** file should contain this declaration. And lastly, this server is expected to operate as a stratum 2 server, since two of the four servers (the first two) declared have hardware clocks and typically run at stratum 1.

```
# Peer configuration for 128.212.66.90 (zite.sci.com)
# (expected to operate at stratum 2)
#
peer        128.211.1.4      # percival.cs.purdue.edu (stratum 1 server)
peer        192.35.53.67     # deidinator.psyc.chi.il.us (stratum 1 server)
peer        128.212.66.67    # HGWells.Sci.com
peer        128.212.66.118   # DrWho.Sci.com
driftfile /etc/inet/ntp.drift
```

Screen 19-1. Example ntp.conf File

Advanced NTP Topics

The following sections discuss some advanced NTP topics.

How the NTP Daemon Operates

NTP operates in a distributed hierarchal-master-slave configuration. Each server adopts a stratum based on its source of time. If the server has direct access to an external source of UTC, then it will be a stratum 1 server. A stratum `n` server is one which is currently obtaining time from a stratum `n`-1 server. For example, a stratum 3 server gets its time from a stratum 2 server, and a stratum 2 server gets its time from a stratum 1 server.

An NTP client polls an NTP server which will respond. Using information it receives from the server, the client can calculate the difference between its (the client's) clock and the server's clock. In the ideal situation the client will poll several servers. Now using information such as the stratum of the server and the round trip delay to the server, the client picks which server it believes has the best estimate of the current time. The client will then attempt to synchronize its clock to that of the server which it believes has the best estimate of the current time.

The algorithms that decide how all of the these decisions are made and the ideas (and experience) behind them are described in the RFCs mentioned on the **`xntpd(1M)`** manual page.

The clock.txt File

The file **`/pub/ntp/clock.txt`** on host `louie.udel.edu` is the canonical source for information concerning the location of stratum 1 and good quality stratum 2 servers. **`clock.txt`**, slightly over 20 pages long (1407 lines), also contains some general information about configuring NTP servers, most, if not all, of which is covered in this document you are currently reading. The file also lists the serial timecode formats used by the various manufacturers as given in the instruction manuals of their clocks, sources for timecode receivers, sources for precision oscillators and timing receivers, the call, location, frequencies, and coordinates for timecode transmitters, an algorithm for great-circle distance and bearing computation, and information on timecode transmitters in Germany, United Kingdom, and France.

For most people the most useful information in the **`clock.txt`** file is the location of primary and secondary NTP servers. The information on primary and secondary servers consists of the following:

```
hostname # IP address
Location: # organization Name and geographic location
Synchronization: # machine type, clock type, synchronization protocol (NTP)
Service area: # networks served (e.g.,  CICNET, NSFNET, SURA, ...)
Access policy: # whether or not access is open to the public
Contact: # name and email address of the person responsible for this host
Note: # miscellaneous information
```

If you are on the Internet, you can FTP the **`clock.txt`** file from `louie.udel.edu` by entering the following commands:

```
$ ftp louie.udel.edu
Connected to louie.udel.edu
220 louie.udel.edu.
Name (louie.udel.edu:muckel): anonymous
331 Guest login ok, send ident as password.
Password: guest
230 Guest login ok, access restrictions apply.
ftp> cd /pub/ntp/doc
250 CWD command successful.
ftp> get clock.txt
200 PORT command successful.
150 ASCII data connection for clock.txt (128.212.66.118,1198) (57002 bytes).
local: clock.txt remote: clock.txt
58409 bytes received in 8 seconds (7.1 Kbytes/s)
ftp> quit
221 Goodbye.
$
```

The driftfile

The `driftfile` entry in **`/etc/inet/ntp.conf`** tells **`xntpd`** the name of the file where it can find and store the frequency error (clock drift) of the system clock. If the file exists at startup, it is read and the value used to initialize **`xntpd`**'s internal value of the frequency error. The file is updated once every hour by **`xntpd`**. It usually takes a day or so after the daemon is started to compute a good estimate of the clock drift. Once the initial value is computed it will change only by relatively small amounts during the course of continued operation. Because **`xntpd`** needs a good estimate to synchronize closely to its server, there should always be a driftfile declaration in **`/etc/inet/ntp.conf`**.

Association Modes

There are a number of modes in which NTP servers can associate with each other, with the mode of each server in the pair indicating the behavior the other server can expect from it. In particular, when configuring a server to obtain time from other servers, there is a choice of two modes which may be alternatively used. Configuring an association in symmetric active mode (usually indicated by a `peer` declaration in configuration files) indicates to the remote server that your system wants to obtain time from the remote server and that your system also is willing to supply time to the remote server if need be. Configuring an association in client mode (usually indicated by a server declaration in configuration files) indicates that the system wants to obtain time from the remote server, but that the system is not willing to provide time to the remote. It is recommended that symmetric active mode should be used for configuring all peer associations between NTP daemons (or stateful NTP entities). Client mode is for use by boot time date-setting programs and the like, which really have no time to provide and which don't retain state about associations over the longer term.

Address-and-mask Configuration Facility

The address-and-mask configuration facility supported by **`xntpd`** is quite flexible and general. However the user interface is not so nice. For this reason it is probably worth doing an example here.

Briefly, the facility works as follows. There is an internal list, each entry of which holds an address, a mask, and a set of flags. On receipt of a packet, the source address of the packet is compared to each entry in the list, with a match being posted when the following condition is true:

`(`*source_addr & mask*`) == (`*address & mask*`)`

A particular source address may match several list entries. In this case, the entry with the most one bits in the mask is chosen. The flags associated with this entry are returned.

In the current implementation the flags always add restrictions. In effect, an entry with no flags set leaves matching hosts unrestricted. An entry can be added to the internal list using a `restrict` statement. The flags associated with the entry are specified textually. The `ignore` flag indicates that all packets from hosts matching this entry will be ignored. The `noquery` flag indicates that hosts matching this entry should not be allowed to send mode 6 and mode 7 packets. The `nomodify` flag indicates that hosts matching this entry should not be allowed to do run-time configuration. The `notrap` flag indicates that hosts matching this entry will not be allowed to register as trap receiver. The `lowpriotrap` flag indicates that hosts matching this entry will be given a low priority for the use of traps. The `noserve` flag indicates that hosts matching this entry will not be allowed to send packets other than mode 6 or mode 7 (they are denied time service). The `nopeer` flag indicates that hosts matching this entry will not be considered as a peer. The `notrust` flag indicates that hosts matching this entry, while treated normally in other respects, shouldn't be trusted for synchronization.

Now the example. Suppose you are running the server on a host whose address is `128.212.66.67`. You would like to ensure that run-time reconfiguration requests can only be made from the local host, and that the server only ever synchronizes to one of a pair of off-site servers or, failing that, a time source on net `128.212`. The following entries in the configuration file would implement this policy:

```
# By default, don't trust and don't allow modifications
restrict default notrust nomodify
# These guys are trusted for time, but no modifications allowed
restrict     128.212.0.0      mask 255.255.0.0 nomodify
restrict     128.192.8.4      nomodify
restrict     192.211.1.24     nomodify
# The local addresses are unrestricted
restrict     128.212.66.67
restrict     127.0.0.1
```

The first entry is the default entry, which all hosts match and, hence, which provides the default set of flags. The next three entries indicate that matching hosts will only have the nomodify flag set and, hence, will be trusted for time. If the mask isn't specified in the `restrict` statement, it defaults to `255.255.255.255`. Note that the address `128.212.66.67` matches three entries in the table, the default entry (mask `0.0.0.0`), the entry for net `128.212` (mask `255.255.0.0`), and the entry for the host itself (mask `255.255.255.255`). As expected, the flags for the host are derived from the last entry since the mask has the most bits set.

Also, note that the `restrict` statements apply to packets from all hosts, including those that are configured elsewhere in the configuration file, and even including your clock pseudopeer(s), if any. Hence, if you specify a default set of restrictions which you don't want to be applied to your configured peers, you must remove those restrictions for the configured peers with additional `restrict` statements mentioning each peer.

Authentication

`xntpd` supports the optional authentication procedure specified in the NTP Version 2 specification. Briefly, when an association runs in authenticated mode, each packet transmitted has appended to it a 32-bit key ID and a 64-bit crypto checksum of the contents of the packet computed using the DES algorithm. The receiving peer recomputes the checksum and compares it with the one included in the packet. For this to work, the peers must share a DES encryption key and, furthermore, must associate the shared key with the same key ID.

This facility requires some minor modifications to the basic packet processing procedures to actually implement the restrictions to be placed on unauthenticated associations. These modifications are enabled by the `authenticate` configuration statement. In particular, in authenticated mode, peers which send unauthenticated packets, peers which send authenticated packets which the local server is unable to decrypt, and peers which send authenticated packets encrypted using a key we don't trust are all marked untrustworthy and unsuitable for synchronization. Note that, while the server may know many keys (identified by many key IDs), it is possible to declare only a subset of these as trusted. This allows the server to share keys with a client which requires authenticated time and which trusts the server, but which is not trusted by the server. Also, some additional configuration language is required to specify the key ID to be used to authenticate each configured peer association. Hence, for a server running in authenticated mode, the configuration file might look similar to the following:

```
# Peer configuration for 128.212.66.67
# (expected to operate at stratum 2)
# Fully authenticated this time
#
peer     128.212.66.90    key 22       # Zite.Sci.com
peer     128.212.66.118   key 22       # DrWho.Sci.com
peer     128.211.1.21     key 4        # bedivere.cs.purdue.edu
peer     128.192.8.4      key 6        # asmus1.genetics.uga.edu
authenticate     yes
trustedkey       4 6 22
keys             /etc/inet/ntp.keys
authdelay        0.000323
driftfile        /etc/inet/ntp.drift
```

The keys Statement

The `keys` statement declares the location of a file containing the list of keys and associated key IDs the server knows about. Because the file contains unencrypted passwords, it is better left unreadable by anyone except the server.

Key File

The format of the key file is:

key_id   *key_format* *key*

The key ID (the first token) is that used by the `trustedkey`, `requestkey`, and `controlkey` statements in the configuration file. The key IDs are arbitrary, unsigned 32-bit numbers. The key format (the second token) identifies the format of the next field, the key. The third token is the key. A DES encryption key is 56 bits long, and is written as an 8-octet number, with 7 bits of the key in each octet. The eighth bit in the octet is a parity bit, and is set to maintain odd parity in each octet. There are three key formats. An `A` indicates that the key is an ASCII character string from one to eight characters long, and that the 7-bit ASCII representation of each character should be used as an octet of the key. An `S` indicates that the key is written as a hex number in the DES standard format, with the low order bit of each octet being the parity bit. An `N` indicates that the key is again a hex number, but that it is written in NTP standard format, with the high order bit of each octet being the parity bit. For the `S` and `N` key formats, the daemon **`xntpd`** requires that the parity bits be set correctly to maintain odd parity.

Here is an example key file:

```
# Keys 4, 6 and 22 are actually identical
#
4       A       DonTTelL
6       S       89dfdca8a8cbd998
22      N       c4ef6e5454e5ec4c
#
# The following three keys are also identical
#
100     A       SeCReT
1000    N       d3e54352e5548080
100000  S       a7cb86a4cba80101
```

The authdelay Statement

The `authdelay` statement is an estimate of the amount of processing time taken between the freezing of a transmit timestamp and the actual transmission of the packet when authentication is enabled (in other words,an estimate of the time it takes for the DES routine to encrypt a single block), and is used as a correction for the transmit timestamp. This can be computed for your CPU by the **`authspeed`** program. The usage is similar to the following:

```
authspeed -n 30000 auth.samplekeys
```

Name Resolution

A recent addition to **`xntpd`** is the ability to specify host names requiring resolution in `peer` and `server` statements in the configuration file. There are several reasons why this was not permitted in the past. Chief among these is the fact that name service is unreliable, and the interface to the UNIX system resolver routines is synchronous. The hangs this combination can cause are unacceptable once the NTP server is running (and remember it is up and running before the configuration file is read).

Instead of running the resolver itself, the daemon defers this task to a separate program, **`xntpres`**. When the daemon comes across a `peer` or `server` entry with a non-numeric host address, it records the relevant information in a temporary file and continues on. When the end of the configuration file has been reached, and one or more entries requiring the resolver have been found, the server runs an instance of **`xntpres`** with the temporary file as an argument. The server then continues on normally, but with the offending peers/ servers omitted from its configuration.

`xntpres` attempts to resolve each name. When it successfully gets one, it configures the `peer` entry into the server using the same mode 7 run-time reconfiguration facility that **`xntpd`**c uses. If temporary resolver failures occur, **`xntpres`** will periodically retry the offending requests until a definite response is received. The program will continue to run until all entries have been resolved.

There are several configuration requirements if **`xntpres`** is to be used. The path to the **`xntpres`** program must be made known to the daemon via a resolver configuration entry, and mode 7 run-time reconfiguration must be enabled. The following fragment might be added to **`/etc/inet/ntp.conf`** to accomplish this:

```
resolver /usr/sbin/xntpres
keys     /etc/inet/ntp.keys
requestkey 65535
```

Note that **`xntpres`** sends packets to the server with a source address of `127.0.0.1`. You should obviously avoid restricting modification requests from this address, or **`xntpres`** will fail.

Clock Support Overview

`xntpd` was designed to support radio (and other external) clocks and does some parts of this function nicely. Clocks are treated by the protocol as favored NTP peers, even to the point of referring to them with an (invalid) IP host address. Clock addresses are of the form

`127.127.`*type*`.`*unit*

where *type* specifies the particular type of clock (or a particular clock driver), and *unit* is a unit number whose interpretation is clock driver dependent. This is analogous to the use of major and minor device numbers by the UNIX system.

Because clocks look much like peers, both configuration file syntax and run-time reconfiguration commands can be adopted to control clocks unchanged. Clocks are configured via `server` declarations in the configuration file (`peer` declarations can also be used, but the `server` form is preferred). Clocks can be started and stopped using **`xntpdc`**, and are subject to address-and-mask restrictions much like a normal peer. As a concession to the need to sometimes transmit additional information to clock drivers, however, an additional configuration file statement was added, the `fudge` statement. This enables you to specify the values of two time quantities, two integral values and two flags, the use of which is dependent on the particular clock driver the values are being given to.

For example, to configure a PST radio clock which can be accessed through the serial device **`/dev/pst1`**, with propagation delays to `WWV` and `WWVH` of 7.5 and 26.5 milliseconds, respectively, on a machine with an imprecise system clock and with the driver set to disbelieve the radio clock once it has gone 30 minutes without an update, one might use the following configuration file entries:

```
server 127.127.3.1
fudge 127.127.3.1 time1 0.0075 time2 0.0265
fudge 127.127.3.1 value2 30 flag1 1
```

NOTE

The implementation of **`xntpd`** does not currently support any hardware clocks.

Converting ntpd Configuration Files to xntpd

Though **`ntpd`** is not a supported product, this section describes how to convert an **`ntpd`** configuration file for use with **`xntpd`**. There are several items to watch out for.

The first is the `precision` entry, if there is one. This entry should be removed. There was a time when the `precision` claimed by a server was mostly commentary, with no particularly useful purpose. This is no longer the case. Thus, changing the `precision` a server claims should only be done with some consideration of how this alters the performance of the server.

Next is the use of numeric IP addresses instead of host names in the `peer` declarations. It is possible to use host names instead of numeric IP addresses, but this requires additional attention.

Finally, passive and client entries in an **`ntpd`** configuration file have no useful semantics for **`xntpd`** and should be deleted. For example, **`xntpd`** won't reset the kernel variable `tickadj` when it starts, so you can remove anything dealing with this in the configuration file. The configuration of radio clock peers is done using different configuration language in **`xntpd`** configuration files, so you will need to change these entries from your **`ntpd`** configuration file.

Dealing With a Mixture of xntpd and ntpd Servers

`xntpd` is an NTP Version 2 implementation. As such, when no additional information is available concerning the preferences of the peer, **`xntpd`** claims, by default, to be Version 2 in the packets it sends.

`ntpd`, while implementing most of the Version 2 algorithms, still believes itself to be a Version 1 implementation. The problem is that when **`ntpd`** receives a packet claiming to be from a Version 2 server, it silently discards the packet. Hence there is a situation where **`ntpd`** will ignore packets from **`xntpd`**.

On the other hand, when **`xntpd`** is polled by a host claiming to be a Version 1 implementation, **`xntpd`** claims to be a Version 1 implementation in the packets returned to the poller. This allows **`xntpd`** to service **`ntpd`** clients transparently.

The problem occurs when an **`ntpd`** server is configured in an **`xntpd`** configuration file. To get around the problem, **`xntpd`** allows a qualifier to be added to configuration entries to indicate which version to use when polling. Hence the entry:

```
peer 192.35.54.2 version 1 # nehd.gov (running ntpd)
```

will cause Version 1 packets to be sent to the host `192.35.54.2`. If you are running **`xntpd`** against existing **`ntpd`** servers, you will need to be careful when configuring **`xntpd`**.

NTP Examples

This first example would be used when configuring a single host to use NTP. In the example, the host `rainbow.sci.com` will be expected to run as a stratum 2 server, since it is configured to peer with two machines that have radio clocks: `bedivere.cs.purdue.edu` and `asmus1.genetics.uga.edu`. **`xntpd`** will use **`/etc/inet/ntp.drift`** to store the frequency error of `rainbow`'s clock. This is a good example to use if you just want to get **`xntpd`** up and running quickly. Since it can take a day or two for a system to get in sync with its peer, you will still need to be patient. In other words, don't start **`xntpd`** and expect to see your clock sync up and stay in sync with a peer in a matter of seconds, minutes, or even hours.

The following is an example of an **`ntp.conf`** configuration file for `rainbow`:

```
# Peer configuration for 128.212.66.13
# (rainbow.sci.com)
#
peer 128.211.1.24 # bedivere.cs.purdue.edu
peer 128.192.8.4 # asmus1.genetics.uga.edu
driftfile /etc/inet/ntp.drift
```

The next example is the complete configuration for `sci.com`. The configuration files here configure `sci.com` to match the topology shown earlier in this chapter. All hosts can be configured dynamically with either **`xntpdc`** or **`ntpq`**. And there are no restrictions on who can peer with any of the hosts.

The following is an example of an **`ntp.conf file`** for host `zite`.

```
# Peer configuration file for 128.212.66.90
# (zite.sci.com)
#
peer 128.211.1.4 # percival.cs.purdue.edu
peer 192.35.53.67 # deidinator.psyc.chi.il.us
peer HGWells.Sci.com
peer DrWho.Sci.com
broadcast 128.212.66.255
driftfile /etc/inet/ntp.drift
resolver /usr/sbin/xntpres
keys /etc/inet/ntp.key
requestkey 65534
controlkey 65535
```

The following is an example of the **`keys.conf`** file for the host `zite`.

```
2313     A        APassword
65534    A        NoSecret
65535    A        BadKey
```

The following is the **`ntp.conf`** configuration file for `HGWells`:

```
# Peer configuration for HGWells
# (128.212.66.67)
#
peer         128.192.8.4 # asmus1.genetics.uga.edu
peer         128.211.1.24 # bedivere.cs.purdue.edu
peer         zite
peer         DrWho
broadcast    128.212.66.255
driftfile    /etc/inet/ntp.drift
resolver     /usr/sbin/xntpres
keys         /etc/inet/ntp.keys
requestkey   65534
controlkey   65535
```

The following is the **`keys.conf`** configuration file for `HGWells`:

```
2313     A        APassword
65534    A        NoSecret
65535    A        BadKey
```

The following is the **`ntp.conf`** configuration file for `DrWho`:

```
# Peer configuration file for 128.212.66.118
# DrWho.Sci.com
#
peer 192.35.54.2 # nehd.gov
peer 192.35.55.2 # berdie.athens.ga.us
peer HGWells.Sci.com
peer Zite.Sci.com
broadcast 128.212.66.255
driftfile /etc/inet/ntp.drift
resolver /usr/sbin/xntpres
keys /etc/inet/ntp.keys
requestkey 65534
controlkey 65535
```

The following is the **`keys.conf`** configuration file for `DrWho`:

```
2313     A        APassword
65534    A        NoSecret
65535    A        BadKey
```

The following is the **`ntp.conf`** configuration file for `Moondoggie` and all other peers:

```
# Peer configuration used by all stratum 3
# servers at Sci.com
#
broadcastclient yes
broadcastdelay 0.0500
driftfile /etc/inet/ntp.drift
resolver /usr/sbin/xntpres
keys /etc/inet/ntp.keys
requestkey 65534
controlkey 65535
```

The following is the **`keys.conf`** configuration file for `Moondoggie`:

```
2313     A       APassword
65534    A       NoSecret
65535    A       BadKey
```

Testing and Tuning Your NTP Subnet

There are several parameters available for tuning your server. These are set using the `precision` and `maxskew` configuration statements. A fragment, which would simply reset these to their default values, follows:

```
precision -6
maxskew 0.01
```

The `precision` statement sets the value of `sys.precision` while the `maxskew` statement sets the value of `NTP.MAXSKW`.

The `sys.precision` statement is defined in the NTP specification to be the base 2 logarithm of the expected precision of the system clock. It used to be set by reading the kernel's clock interrupt period and computing a value of `sys.precision`, which gave a precision close to this, in essence making the value rather unpredictable. This was unfortunate since, for NTP version 2, `sys.precision` acquired several quite important functions and is useful as a tuning parameter. The current behavior is to default this value to -6 in all servers.

The NTP protocol makes use of `sys.precision` in several places . `sys.precision` is included in packets sent to peers, and is used by them as a sort of quality indicator. When faced with selecting one of several servers of the same stratum and about the same network path delay for synchronization purposes, clients will tend to prefer to synchronize to those claiming the smallest (most negative) `sys.precision`. The effect is particularly pronounced when all the servers are on the same LAN. Hence, if you run several stratum 1 servers, or three or four stratum 2 servers, but you would like the client machines to prefer one of these over the other(s) for synchronization, you can achieve this effect by decreasing `sys.precision` on the preferred server and/or increasing this value on the others.

The other tuning parameter is the antihop aperture, and is derived from `sys.precision` and `NTP.MAXSKW` using the following equation:

```
2**sys.precision + NTP.MAXSKW
```

Making the antihop aperture larger will make the server less likely to hop from its current system peer (synchronization source) to another, while increasing the probability of the server remaining synchronized to a peer which has malfunctioned. Making the antihop aperture smaller allows the server to hop more freely from peer to peer, but this can also cause it to generate a fair bit more NTP packet traffic than necessary for no good purpose.

Given the agreement among current stratum 1 NTP servers and the performance typical of the Internet, it is recommended that the antihop aperture be maintained with a value of between 0.020 and 0.030 (if you calculate the default value you will find it is about 0.026). You can change the antihop aperture by changing the value of `NTP.MAXSKW` via a `maxskew` configuration statement. Note, however, that if you wish to change `sys.precision` via a `precision` configuration statement, but don't wish to alter the antihop aperture, you must change `NTP.MAXSKW` to compensate.

Related xntp Commands

Two separate query programs are included with the **`xntp`** distribution: **`ntpq`** and **`xntpdc`**.

`ntpq` sends queries and receives responses using NTP standard mode 6 control messages. Since it uses the standard query protocol, it may be used to query the NTP servers as well as **`xntpd`**.

`xntpdc` is a program which uses NTP private mode 7 messages to make requests to the server. The format and content of these messages are specific to **`xntpd`**. The program allows inspection of a wide variety of internal counters and other state data. **`xntpdc`** also provides a user interface to the run-time reconfiguration facility.

Both of these programs are nonessential to the operation of **`xntpd`**. The main reason for describing them here is to point out an important inconsistency. Both **`xntpdc`** and **`xntpd`** demand that anything which has dimensions of time be specified in units of seconds, both in the configuration file and when doing run-time reconfiguration. Both programs also print the values in seconds. **`ntpq`**, on the other hand, obeys the standard by printing all time values in milliseconds. This makes the process of looking at values with **`ntpq`**, and then changing them in the configuration file or with **`xntpdc`**, very prone to errors (by three orders of magnitude).

`xntpd` was written specifically to allow its configuration to be fully modifiable at run time. Indeed, the only way to configure the server is at run time. The configuration file is read only after the rest of the server has been initialized into a running, but default unconfigured, state. This facility was included not so much for the benefit of the UNIX system, where it is handy but not strictly essential, but rather for limited platforms where the feature is more important for maintenance. Nevertheless, run-time configuration works very nicely for UNIX servers as well.

Nearly all of the things it is possible to configure in the configuration file may be altered via NTP mode 7 messages using the **`xntpdc`** program. Mode 6 messages may also provide some limited configuration functionality (though the only thing you can currently do with mode 6 messages is set the leap second warning bits) and the **`ntpq`** program provides generic support for the latter.

Mode 6 and mode 7 messages which would modify the configuration of the server are required to be authenticated using standard NTP authentication. To enable the facilities you must, in addition to specifying the location of a **`keys`** file, indicate in the configuration file, the key IDs to be used for authenticating reconfiguration requests. Hence, the following fragment might be added to a configuration file to enable the mode 6 (**`ntpq`**) and mode 7 (**`xntpdc`**) facilities in the daemon:

```
keys     /etc/inet/ntp.keys
requestkey   65535    # for mode 7 requests (xntpdc)
controlkey   65534    # for mode 6 requests (ntpq)
```

If the `requestkey` and/or the `controlkey` configuration statements are omitted from the configuration file, the corresponding run-time reconfiguration facility is disabled.

The query programs require the user to specify a key ID, and a key to use for authenticating requests to be sent. The key ID provided should be the same as the one mentioned in the configuration file, while the key should match that corresponding to the key ID in the **`keys`** file. It is recommended that you make the request and control authentication keys text characters (in ASCII format). This is because the query programs prompt you for them as passwords you must type from the keyboard.

Examples

This example demonstrates the use of **`ntpq`** and **`xntpd`** to list the servers that `moondoggie.sci.com` is currently polling. The server marked with an `*` is the server that we are currently peering with. We are also listening for broadcast packets on the `128.212.66` subnet, but no one is currently broadcasting packets there. Note the difference in the units of the `delay`, `offset`, and `dispersion` between the two different commands. **`ntpq`** displays milliseconds while **`xntpdc`** displays seconds.

```
sjc@moondoggie $ ntpq
ntpq> peer
 remote         refid         st   when   poll  reach   delay    offset  disp
==============================================================================
128.212.66.255  0.0.0.0       16   never  64      0     0.0       0.00   64000
*Zite.sci.com   deidinator.ps  2   53     64    376     81.1     -1.00     12.4
+DrWho.sci.com  berdie.athens  2   11     64     37     41.2     30.00   7504.3
sjc@moondoggie $ xntpdc
xntpdc> peer
 remote         local         st   poll   reach delay   offset     disp
==============================================================================
*Zite.sci.com   192.35.53.67   2   64     376   0.0811  -0.001003  0.0124
-DrWho.sci.com  192.35.55.2    2   64      37   0.041   0.030001   7.504
^128.212.66.255 0.0.0.0       16   64       0   0.000   0.000000   64.000
```

This information tells us, among other things, that `moondoggie` is currently peering with `zite`, that `zite` is at stratum 2 (and thus we are at stratum 3), that we are polling `zite` once every 64 seconds, that the reachability register for `zite` is 376, in octal, that `moon-doggie` is 1 millisecond (actually 1003 microseconds) behind its peer `zite`, that the round trip delay to `zite` is 81 milliseconds, and that the dispersion is 12.4 milliseconds.

The next example shows how to turn on monitoring, and how to list the statistics that are gathered when monitoring is being done.

```
xntpdc> monitor on
Keyid: 65534
Password: NoSecret
done!
    .
    .
    .
xntpdc> monlist
 address        port    count   mode   version  lasttime  firsttime
===================================================================
127.0.0.1       1417    6        7      2         0        475668
128.212.66.118   123    3818     1      2        92        475332
128.212.66.90    123    7432     5      2        48        475632
```

This tells us who has connected to the daemon (**`xntpd`**), on what `udp` port the connection was made, the number of packets received, the mode of this connection, the NTP Version, the time since the last packet exchange (in seconds), and the time since the first packet exchange (in seconds).

The last example shows how we can dynamically delete and add peers.

```
xntpdc> host rainbow
current host set to rainbow.sci.com
xntpdc> peer
remote            local        st  poll  reach  delay    offset    disp
==========================================================================
*zite.sci.com     128.211.1.4   2    64  376    0.0512  -0.010000   0.0021
+HGWells.sci.com  0.0.0.0      16  1024    0    0.0000   0.000000  64.000
xntpdc> unconfig HGWells
Keyid: 65534
Password: NoSecret
done!
xntpdc> peer
remote            local        st  poll  reach  delay    offset    disp
=========================================================================
*zite.sci.com     128.211.1.4   2    64  376    0.0512  -0.010000  0.0021
xntpdc> addpeer DrWho.sci.com
done!
xntpdc> peer
remote            local        st  poll  reach  delay    offset    disp
==========================================================================
+DrWho.sci.com    0.0.0.0      16  64      0    0.0000   0.000000  64.000
*zite.sci.com     128.211.1.4   2  64    376    0.0512  -0.010000   0.0021
xntpdc>

    .
    .
    .
xntpdc> peers
 remote         local          st  poll  reach  delay    offset    disp
=========================================================================
*DrWho.sci.com  128.212.66.118  2  64    377    0.0312  -0.004488  0.0011
~zite.sci.com   128.211.1.4     2  64    376    0.0512  -0.010000  0.0021
xntpdc>
```

Troubleshooting xntpd

`xntpd` uses **`syslog`** to log all problems and errors. Excessive STEPs in the **`syslog`** file indicates that the daemon is having a problem synchronizing the clock. The most likely cause is that some other program (such as **`timed`**) is also setting/skewing the clock. Another sign of more than one entity playing with the clock is the following syslog message:

```
Previous time adjustment didn't complete
```

The message:

```
Clock appears to be # seconds fast|slow, something may be wrong
```

is issued when the clock is off by more than 1000 seconds. **`xntpd`** will not try to synchronize with a host under this circumstance. If you want to synchronize with this host, you will need to set the system clock to within 1000 seconds of the host by hand (by using **`date`** or **`ntpdate`**).

Administering the timed Time Synchronization Daemon

As an alternative method to using NTP for time synchronization, the OS supports the Berkeley Time Synchronization Protocol. The Berkeley Time Synchronization Protocol service is composed of a collection of time daemons (**`timed`**) running on the machines in a local area network. The algorithms implemented by the service are based on a master-slave scheme. The time daemons communicate with each other using the Time Synchronization Protocol (TSP), which is built on the UDP protocol.

A time daemon has a twofold function. First, it supports the synchronization of the clocks of the various hosts in a local area network. Second, it starts (or takes part in) the election that occurs among slave time daemons when, for any reason, the master disappears. The synchronization mechanism and the election procedure employed by the program **`timed`** are described in other documents. The next paragraphs are a brief overview of how the time daemon works. This document is mainly concerned with the administrative and technical issues of running **`timed`** at a particular site.

A master time daemon measures the time differences between the clock of the machine on which it is running, and the clocks of all other machines. The master computes the network time as the average of the times provided by non-faulty clocks.

NOTE

A clock is considered to be faulty when its value is more than a small specified interval apart from the majority of the clocks of the other machines.

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

To ensure that the service provided is continuous and reliable, it is necessary to implement an election algorithm to elect a new master, should the machine running the current master crash, the master terminate (for example, because of a run-time error), or the network be partitioned. Under our algorithm, slaves are able to realize when the master has stopped functioning and to elect a new master from among themselves. It is important to note that, since the failure of the master results only in a gradual divergence of clock values, the election need not occur immediately.

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

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

Guidelines

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

If we exclude submasters, there will normally be, at most, one master time daemon in a local area internetwork. During an election, only one of the slave time daemons will become the new master. However, because of the characteristics of its machine, a slave can be prevented from becoming the master. Therefore, a subset of machines must be designated as potential master time daemons. A master time daemon will require CPU resources proportional to the number of slaves, in general, more than a slave time daemon, so it may be advisable to limit master time daemons to machines with more powerful processors or lighter loads. Also, machines with inaccurate clocks should not be used as masters. This is a purely administrative decision; an organization may well allow all of its machines to run master time daemons.

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

There are some limitations to the current implementation of the time daemon. For example, the maximum number of machines that may be directly controlled by one master time daemon is fixed at 99.

In addition, there is a pathological situation to be avoided at all costs, that might occur when time daemons run on multiply-connected local area networks. In this case, as we have seen, time daemons running on gateway machines will be submasters, and they will act on some of those networks as master time daemons. For example, consider machines A and B that are both gateways between networks X and Y. If time daemons were started on both A and B without constraints, it would be possible for submaster time daemon A to be a slave on network X and the master on network Y, while submaster time daemon B is a slave on network Y and the master on network X. This loop of master time daemons will not function properly or guarantee a unique time on both networks, and will cause the submasters to use large amounts of system resources in the form of network bandwidth and CPU time. In fact, this kind of loop can also be generated with more than two master time daemons, when several local area networks are interconnected.

Options

The flags are:

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

Daily Operation

The **`timedc`** command is used to control the operation of the time daemon. **`timedc`** may be used to:

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

See the **`timed(1M)`** and **`timedc(1M)`** manual pages for more detailed information. Note that the **`netdate(1M)`** command can be used to set the network date.

4
Distributed File System Administration

Replace with Part 4 tab

20
Introduction to DFS Administration

20
Introduction to DFS Administration

About DFS Administration

The DFS Administration chapters, “Introduction to DFS Administration”, “Setting Up DFS”, “Using DFS Commands and Files” and “DFS sysadm Interface”, describe the tasks you can perform using Distributed File System Administration—a set of commands and files that allows you to administer NFS (as well as other distributed file system types that may be available in the future) in a consistent way. The files and commands that support common administration of NFS are provided in the Distributed File System (DFS) Administration Utilities package.

Organization

The DFS Administration chapters are organized by the tasks that can be performed using DFS Administration commands and files.

In addition to the command line interface, a menu interface is provided through which you can enter DFS commands. The menus are included with System Administration Menus (**`sysadm`**), a menu-based administration interface that is standard with the OS. Once you access a **`sysadm`** menu, help screens provide you with all the information you need to complete a task. Therefore, the “DFS sysadm Interface” chapter does not lead you step-by-step through the menu-based procedures. Instead, it documents DFS commands as you would enter them at the command line. This chapter describes the DFS menus and directs you to “Using the sysadm Interface” in the *System Administration* guide for instruction in using the menu interface.

The organization of the DFS Administration chapters are as follows:

- “Introduction to DFS Administration” presents an overview of DFS Administration, and describes all the commands and files that the Distributed File System Administration Utilities package either installs or utilizes.
- “Setting Up DFS” describes the software that must be installed before you can use DFS Administration and directs you to installation instructions.
- “Using DFS Commands and Files” tells you how to share and unshare file system resources using DFS commands and files; how to mount and unmount remote resources; and how to display information about resources that are shared and mounted on your local system and on network clients.
- “DFS sysadm Interface”, describes the System Administration Menu (**`sysadm`**) interface to DFS Administration.

File Sharing

File sharing employs a *client/server* model. A computer that wishes to share file systems with other computers on a network acts as a *server*. Files are physically owned and managed by the server machine. A computer that wants to access file systems shared by the server acts as a *client* of the server machine. Acting on behalf of its applications, the client makes requests to a server to access data in a file or to perform file manipulations. A single machine may be both a client and a server, making it possible to share its local file systems and to access remote file systems.

A server may offer any directory tree for access over the network. Once shared, an authorized client may mount the remote file system on any of its local directories. The mount procedure behaves in a manner similar to mounting local file systems.

Transparency is the key to the usefulness of file sharing. Once mounted, remote file systems look like local file systems from a user or application perspective. Applications, in most cases, run unchanged.

Since remote file systems may be mounted anywhere in the local tree, existing programs can run on several different computers while still having the same files and directory structure available to them. Creating the file environment is now mainly an administrative task, not one requiring program changes.

Servers do not need to make all their files accessible to network clients. In the following illustration, the server is sharing **`/public/tkit.`** The client mounts **`/public/tkit`** on its local directory **`/usr/tools.`** The remote directory tree now appears to be a directory tree under **`/usr/tools,`** and files in that tree may be accessed as though they were local. Note that the client cannot access **`/public/tkit2.`**

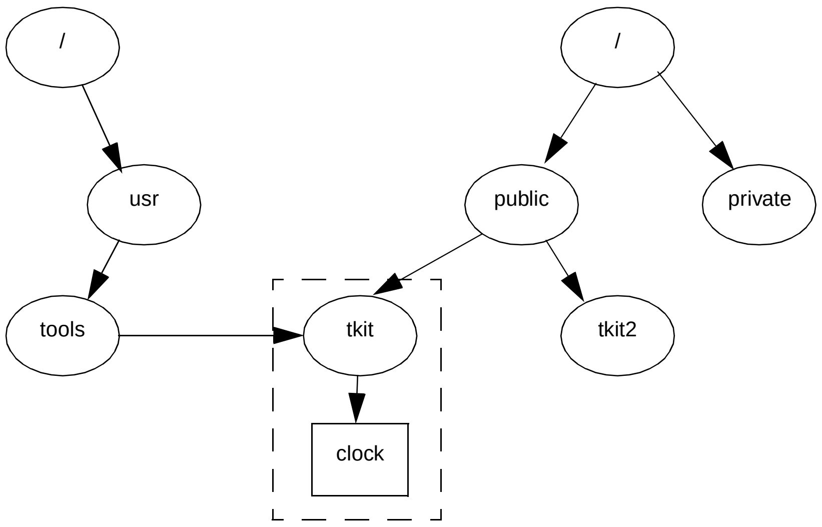


Figure 20-1. File Sharing Example

In a file sharing environment, a large number of users can access a program as though it were on their local machines, when actually the program resides on a single file server. This is a great benefit to small workstations, where disk space is at a premium. A user can have access to a much larger program repertoire than could fit on a private disk.

By having a resource reside physically on a single server, then distributed throughout the network, you can greatly simplify administration. First, you reduce the number of copies of various programs that need to be maintained on the network. Second, you reduce the problems involved in performing backups for a number of machines dispersed over a wide geographical area. By keeping files in a single location, this task becomes comparable to backing up a single machine.

Centralizing files on a few file servers not only simplifies administration, it helps maintain consistency of shared data files. When changes are made to a shared file, they become available to all users immediately.

As an alternative to centralizing files on a few file servers, files may be shared in a peer manner. When a single computer runs out of capacity, more computers can be added to a configuration. Files can be moved to the new computers, while a consistent view of the file system from the user's perspective is maintained.

An Overview of DFS Administration

Some tasks involved in the administration of a distributed file system package are the same, whether you are using NFS or some other file system package. The Virtual File System architecture provides a mechanism for administering file system types in a general, or generic, way. Administrators of NFS or other file system types are provided with a common set of commands which can be used to administer NFS (as well as any distributed file system type that may be supported in the future).

For example, to share a resource on your computer with remote systems, you must make the resource available and communicate its availability. To make a local resource available running earlier versions of NFS, you would “export” it, using the **`exportfs`** command. If you were to run another distributed file system package, keeping the corresponding package-specific commands straight might be difficult. Multiply this by the number of corresponding package-specific commands in both packages, and the job becomes even more difficult.

DFS Administration provides generic commands that call package-specific commands, freeing you of the need to learn two sets of corresponding commands.

DFS Commands and Files

The DFS Administration Utilities package installs seven commands: **`share`**, **`unshare`**, **`shareall`**, **`unshareall`**, **`dfshares`**, and **`dfmounts`**. In addition to these commands, DFS Administration utilizes a number of files and commands installed by the other packages.

All the files and commands relevant to DFS administration are described in this section. Files and commands that operate on all file system types are described in this section only as they relate to administering distributed file systems.

The DFS Administration files are:

- **`/etc/dfs/fstypes`**, which is used to register the distributed file system packages you have installed on your system and establishes the default package (the first line in this file is the default package). The **`fstypes`** file is created by the distributed file system package you install first.
- **`/etc/dfs/dfstab`**, which allows you to share a resource or a set of resources automatically when you enter system state 3.
- **`/etc/vfstab`**, which allows you to mount resources automatically when you enter system state 3.
- **`/etc/dfs/sharetab`**, which logs the resources currently shared on your system. The **`sharetab`** file is created by the **`share`** command and does not require handling by an administrator.
- **`/etc/mnttab`**, which logs the file systems currently mounted by your system, including remote file systems and directories. The **`mnttab`** file is created by the **`mount`** command and does not require handling by an administrator.

DFS Administration commands are:

- **`share(1M),`** which allows you to make a resource available for mounting by clients, or to display a list of the resources on your system that are currently shared.
- **`unshare(1M),`** which allows you to make a previously available resource unavailable for mounting by clients.
- **`shareall(1M)`**, which executes a script that shares a pre-determined set of resources, listed in **`/etc/dfs/dfstab`**.
- **`unshareall`** [see **`shareall(1M)]`**, which executes a script that unshares all currently shared resources, listed in **`/etc/dfs/dfstab`**.
- **`mount(1M)`**, which allows you to mount a remote resource on your system, or to display a list of resources, both local and remote, that are currently mounted on your system.
- **`umount`** [see **`mount(1M)`**], which allows you to remove a remote resource you previously mounted.
- **`mountall(1M)`**, which executes a script that mounts a pre-determined set of resources.
- **`umountall`** [see **`mountall(1M)],`** which executes a script that unmounts all currently mounted resources.
- **`dfshares(1M)`**, which displays a list of remote resources that are available to you, as well as a list of local resources that are currently shared.

- **`dfmounts(1M),`** which shows you which local resources are mounted by which clients.

21
Setting Up DFS

21 Setting Up DFS

Introduction to DFS Setup

This chapter describes the tasks you must complete before you execute DFS Administration commands. These tasks are installing the software, possibly editing a file to establish a default file sharing package, and starting distributed file system operation.

Installing the Software

If you installed all the software when you received the operating system, DFS Administration is already installed on your system. If you did not install all the add-on utilities, you'll need to install DFS Administration Utilities, as well as NFS Utilities and all the hardware and software required to run NFS.

Installation instructions for all operating system software—including DFS Administration Utilities and NFS Utilities—appear in the *Release Notes*. Hardware and software prerequisites for NFS appear in Chapter 25, "Setting Up NFS".

Once all your network hardware and software are installed, you must set up your file sharing packages before you can enter DFS commands.

Logging in As the Network Administrator

If the Enhanced Security Utilities are installed, the role of the superuser is divided into multiple roles, each with the authority to perform specific administrative tasks. There no longer exists a single login that allows a user to override all system restrictions and perform any task associated with system administration or maintenance.

In place of the superuser, the Enhanced Security Utilities provide five distinct administrative roles, each with responsibility for a certain aspect of the system. Each role is associated with the commands and privileges required to complete the relevant administrative tasks. Logins are then created and assigned an administrative role.

The database that associates commands with roles is the Trusted Facility Management (TFM) database. A command entry in the database includes an alias for the command, the path to the executable file, and the privileges to be granted to an authorized user. A user identified in the database as an administrator can execute the command with the privileges listed. To execute the command, however, the user must preface it with **`tfadmin.`** For example, to shut down the system, a user must access the system using a login associated with the appropriate administrative role and enter the following command:

```
tfadmin shutdown -g0 -y -i6
```

The role with responsibility for network administration is NET. Before you can use any of the commands associated with DFS or NFS, you must create a login and assign it the administrative role NET. Once the login has been created, you must access the system using that login, then include **`tfadmin`** on every DFS or NFS command line.

NOTE

For convenience, **`tfadmin`** does not appear in command line examples throughout this guide.

In addition to the roles pre-defined on the system, Trusted Facility Management provides a privileged user with the ability to define other administrative roles. If you choose, you can divide network administration among several people in your organization. For example, you may want to have one person responsible for BNU, and another responsible for distributed file systems. Using TFM commands, you can create a BNU administrative role and a DFS administrative role, then assign each role to a different login.

The following section tells you how to assign the NET role to a user login, and how to define additional administrative roles.

For more information about Trusted Facility Management and administrative roles in general, see the chapter “Trusted Facility Management” in the *System Administration* guide.

Assigning the NET Role

To assign a role to a user login, you enter the **`adminuser`** command with the following syntax:

```
adminuser [-n] -o role user
```

The **`-n`** option indicates that the user does not exist as yet in the TFM database. If an existing entry assigns a different role to the same user login, omit the **`-n`** option.

Assume you want to assign the NET role to the login `netadmin.` If `netadmin` has no entries in the database, enter the following:

```
adminuser -n -o NET netadmin
```

If you want to verify that NET is assigned to `netadmin,` enter the following.

```
adminuser netadmin
```

Now, to set up distributed file system operation, you would log out and log in again as `netadmin.`

Defining Additional Roles

If you want to divide responsibility for network administration among several users, you can create additional administrative roles and associate commands with each role. For example, you may want to create a role that has responsibility only for NFS administration. To create a role and associate commands with the role, you need to become `root` in single-user mode. Then you enter the **`adminrole`** command with the following syntax:

```
adminrole -n -a cmd:path[:priv[:priv]][, . . . ] role
```

The **`-n`** option indicates that *role* is a new administrative role.

Let's call the NFS administrative role `nfsadmin.` When you create the `nfsadmin` role, you'll need to associate all the NFS commands with it. The following command line creates the role and associates the command **`nfstart`** and **`nfstop`** with the role:

```
adminrole -n -a nfstart:/usr/sbin/nfstart:allprivs,\
     nfstop:/usr/sbin/nfstop:allprivs nfsadmin
```

To assign additional NFS commands to `nfsadmin,` enter the **`adminrole`** command again, omitting the **`-n`** option, as in the following command line:

```
adminrole -a dname:/usr/sbin/dname:allprivs nfsadmin
```

After you associated all the NFS commands with the NFS administrative role, you need to create a login and assign it the `nfsadmin` role. To assign the role to a login, you use the **`adminuser`** command, as described in the previous section. For example, if you want to assign the role to the login `mike,` you would enter the following:

```
adminuser -n -o nfsadmin mike
```

Once you've assigned `nfsadmin` to the login `mike,` a user can log in as `mike` and set up NFS.

Changing the File Sharing Package Default

So that the system can distinguish between local and remote file system types, the distributed file system packages you have installed must be “registered.” When you install NFS, the installation script creates the **`fstypes`** file and populates it. If the file already exists when you install the package, the installation script adds a line to the file:

```
nfs Network File System Utilities: Version number
```

where *number* indicates the version number of the package installed.

The package indicated in the first line of the file becomes your default file sharing package. If you enter DFS Administration commands without specifying a package (or file system type), the system assumes the first entry in the **`/etc/dfs/fstypes`** file. Therefore, if you know that you will administer one package more frequently than the other, make that the first entry in the file. This saves you from specifying the package every time you enter a DFS command.

To change the default package, edit the file using any supported text editor.

Starting Distributed File System Operation

NFS normally becomes operational automatically whenever you take your system to init state 3; however, NFS can be started in other run levels by entering commands at the command prompt.

To start NFS automatically, use the **`init`** command to take your system to init state 3. If you choose, you can have your system enter init state 3 automatically when you boot by changing the `initdefault` line in the **`/etc/inittab`** file to read:

```
is:3:initdefault:
```

As explained earlier, DFS Administration allows you to share and mount resources *explicitly*, by entering commands at the command line, and *automatically*, by editing files that share and mount resources when you take the system to init state 3. Commands can be entered explicitly in any init state once you start distributed file system operation. Automatic sharing and mounting is done <u>only</u> when your system enters init state 3. When you exit init state 3, any resource you shared is automatically unshared, and any remote resource you mounted is automatically unmounted.

22 Using DFS Commands and Files

22
Using DFS Commands and Files

Sharing and Unsharing Resources

This section tells you how to share and unshare NFS (or other file system type) resources in a consistent way, using a common commands and files. The commands and files described in this chapter are the **`share`**, **`unshare`**, **`shareall`**, and **`unshareall`** commands; and the **`/etc/dfs/dfstab`** file.

Before you unshare a resource, using either the **`unshare`** or **`unshareall`** command, be sure you understand the effect of unsharing on existing mounts. For example, when you unshare an NFS resource, access to existing mounts is inhibited; for more information, see Chapter 26, "Sharing and Mounting NFS Resources Explicitly".

Sharing a Resource Explicitly—the share Command

To make a resource on your computer available to clients, you use the **`share`** command, which has the following syntax:

```
share [-F fstype] [-o fs_options] [-d description]
      [pathname [resourcename]]
```

See the **`share(1M)`** manual page for an explanation of the options accepted by the **`share`** command.

For an explanation of the options available for sharing a resource over NFS, see the NFS-specific **`share(1M)`** manual page. For more information about NFS-specific options, see Chapter 26, "Sharing and Mounting NFS Resources Explicitly".

If no argument is specified when the **`share`** command is entered, then the command displays all resources on your system that are currently shared. If only a file system type is specified, then the **`share`** command displays all resources of the specified type that are currently shared. The **`share`** command can be used in this capacity by users as well as administrators.

Example

You want to share a partial file system on your computer. The root directory of the branch of the file system is **`graphics`**, which is a subdirectory to your **`/export`** directory. The directory is an NFS resource. You want to share the directory read-only to all clients except a client named “art.dept,” with which you want to share the directory read/write. At the command line, type the following:

```
share -F nfs -o ro,rw=art.dept /export/graphics
```

Sharing a Resource Automatically—the /etc/dfs/dfstab File

The **`/etc/dfs/dfstab`** file allows you to share a set of resources automatically whenever your system enters init state 3. For example, if you want a directory to be available to clients on a regular basis, and you can anticipate few occasions when you would need to make it unavailable, you can enter a **`share`** command for that directory into the **`dfstab`** file. Then, whenever you take the system to init state 3, the directory becomes available to clients automatically.

Each line of the file consists of the **`share`** command line needed to **`share`** a particular resource; the **`share`** command you enter in the file has the same syntax as the **`share`** command you enter at the command line. (See the description of the **`share`** command in the preceding section.)

If you want to add or delete a resource from a list of sharable resources, or to modify the way the sharing is done, edit the file with your text editor. The next time you enter init state 3 from another init state, the changes you made to the file will take effect.

Example

You want the directory **`editors`** to be available to all clients at all times. The directory is an NFS resource, located in your **`/export`** directory. Use your text editor to add the **`share`** command to the **`/etc/dfs/dfstab`** file. The file entry should look like this:

```
share -F nfs /export/editors
```

Unsharing a Resource—the unshare Command

If you want to reclassify a resource you have shared so that it is no longer available for mounting by remote systems, you “unshare” it. To unshare a resource, you use the **`unshare`** command.

The **`unshare`** command can be used to unshare any resource—whether the resource was shared explicitly with the **`share`** command or automatically through the **`dfstab`** file. If you use the **`unshare`** command to unshare a resource that you shared through the **`dfstab`** file, remember that it will be shared again when you exit and re-enter init state 3. The syntax for the **`unshare`** command is:

```
unshare [-F fstype] [-o fs_options] {pathname | resourcename}
```

See **`unshare(1M)`** for an explanation of the options accepted by the **`unshare`** command.

Example

Although the directory **/export/templates** on your machine is shared continually through the **dfstab** file, you need to unshare the directory temporarily. The directory is an NFS resource. Type the following:

```
unshare -F nfs /export/templates
```

Now you can share the directory immediately using the **share** command, or the directory will be shared again automatically when you exit and re-enter init state 3.

Sharing a Set of Resources—the shareall Command

DFS Administration lets you share a set of resources by entering a single command—the **shareall** command. To use the command, you first create a file that lists the resources you want to share. The syntax of the entries in your file is the same syntax as the **share** command and the entries in the **dfstab** file, as follows:

share [**-F** *fstype*] [**-o** *fs_options*] [**-d** *description*]
        [*pathname* [*resourcename*]]

Once you create the file, you specify it as the input file when you enter the **shareall** command.

If you do not specify an input file, the **/etc/dfs/dfstab** file is used by default.

The syntax of the **shareall** command is as follows:

shareall [**-F** *fsys*[,*fsys* . . .]] [*file*]

See the **shareall(1M)** manual page for an explanation of the options accepted by the **shareall** command.

Example

You create an input file called **misc** that contains commands to share three separate resources. The file looks like this:

```
#cat misc
share -F nfs -o ro,rw=art.dept /export/graphics
share -F nfs /usr/man
```

To share all the resources listed in the file you created, type the following:

```
shareall misc
```

To share only the NFS resources listed in the **misc** file, type the following:

```
shareall -F nfs misc
```

Unsharing a Set of Resources—the unshareall Command

DFS Administration provides you with the **unshareall** command, which lets you unshare all the shared resources on your system, or all the shared resources of a specified file system type. The syntax of the **unshareall** command is

```
unshareall [-F fsys[,fsys . . . ]]
```

When you specify a file system type, the command unshares all local resources of the distributed file system type you specified. If no **-F** option is specified, then the command unshares all local resources currently shared.

For **unshareall**, see the **shareall(1M)** manual page for an explanation of the options accepted by the **unshareall** command.

Example

You want to unshare all currently shared local resources, regardless of file system type. Type the following:

```
unshareall
```

Mounting and Unmounting Remote Resources

This section tells you how to mount and unmount remote resources explicitly, using the generic **mount, umount, mountall,** and **umountall** commands. It also tells you how to mount remote resources automatically, using the **/etc/vfstab** file. The command and files described in this section are described only as they relate to the administration of remote resources.

If you have the Enhanced Security Utilities installed, it is important that you understand that privileges are assigned to executables locally. When you mount an executable from a server, it is mounted without privileges, regardless of the privileges assigned to the executable on the server. For example, **nfstart** may have appropriate privileges assigned to it on the server, however; when you mount the server's **/usr, nfstart** does not have privilege automatically on your system. If the executable requires privilege, you must assign privileges to the executable using the **filepriv** command, just as you would if the executable were local to your system. Once you assign privileges on the client, you can mount the resource or add it to the **vfstab** file. If, later, you unmount the resource or remove the relevant entry from **vfstab,** you should also remove the executable's privileges on the client, using the **filepriv** command with the **-d** option. If you fail to remove privileges on the client, the system displays an error message every time you re-boot.

Mounting a Remote Resource Explicitly—the mount Command

The **`mount`** command allows you to mount both local and remote file systems. To mount a remote resource using the **`mount`** command, you must be able to reach, via a network, the server that is sharing the resource you want to mount.

The syntax of the **`mount`** command as it relates to mounting distributed file systems is:

```
mount [-F FSType] [-V] [current_options] [-o specific_options]
     {special | mount_point}
mount [-F FSType] [-V] [current_options] [-o specific_options]
     special mount_point
```

See the **`mount(1M)`** manual page for an explanation of the options accepted by the **`mount`** command.

For information about NFS-specific options, see Chapter 26, "Sharing and Mounting NFS Resources Explicitly".

Resources mounted explicitly with the **`mount`** command stay mounted unless you unmount them, using **`umount`**, or if you exit init state 3. If you exit init state 3 and re-enter it, the resource is no longer mounted (unless you edited the **`/etc/vfstab`** file to include the mount automatically).

Example 1

You want to mount an NFS resource residing on a server called “tools.srv.” The resource is a directory called **`graphics`** in the server's **`/usr`** directory. You created a mount point called **`graphics`** in your **`/usr/tools`** directory. You want to mount the resource read/write, and you want set-uid bits to be ignored. Type the following:

```
mount -F nfs -o nosuid tools.srv:/usr/graphics\
 /usr/tools/graphics
```

Because no access rights are specified, the directory is mounted read/write.

Example 2

You want to hard mount a directory containing manual pages that resides on a server called “docgroup.” You want the directory mounted read-only, with interrupt enabled. The directory is an NFS resource named **`/usr/man`**. You want to mount the directory to a mount point of the same name on your machine. Type the following:

```
mount -F nfs -o ro,intr docgroup:/usr/man /usr/man
```

Example 3

Assume you have the Enhanced Security Utilities installed and you want to mount an NFS resource residing on a server called “tools.srv.” The resource is a directory called **`graphics`** in the server's **`/usr`** directory. You created a mount point called **`graphics`** in your **/usr/tools** directory. You want the level ceiling of the resource to be a level that has the alias **Tools.** Type the following:

```
mount -l Tools -F nfs tools.srv:/usr/graphics \
   /usr/tools/graphics
```

Mounting a Remote Resource Automatically—the /etc/vfstab File

The **/etc/vfstab** file allows you to mount a local file system automatically when you boot the system. If you edit the file to include a remote file system or directory, the remote resource is mounted automatically when you take the system to init state 3.

To mount a remote resource automatically, create a mount point for the resource using the **mkdir** command, then edit the **vfstab** file. Entries in the **vfstab** file require the following syntax:

special fsckdev mountp fstype fsckpass automnt mntopts fsceiling

See the **vfstab(4)** manual page for a description of the fields in the **/etc/vfstab** file.

Once you create the mount point and edit the **/etc/vfstab** file, the file system or directory will be mounted automatically when you enter init state 3. It will be mounted automatically every time you enter init state 3 until you modify the **vfstab** file (unless, of course, the server sharing the directory unshares it). The contents of **/etc/vfstab** remain the same until you edit the file.

Example

Assume you want to mount the resource **/usr/share** from a server named “frontoffice.” You want the resource to be mounted automatically every time you take your system to init state 3. The directory is an NFS resource. You want to mount the directory on the mount point **/usr/local/tmp** with read-only access. First create the mount point by typing the following:

```
mkdir /usr/local/tmp
```

Make sure the mode and permissions of the new mount point match those of the resource you want to mount on it. Then edit the **/etc/vfstab** file, using any supported text editor. The file entry should look like this:

```
frontoffice:/usr/share - /usr/local/tmp nfs - yes ro
```

Unmounting a Remote Resource—the umount Command

The **umount** command allows you to unmount both local and remote file systems. You can unmount a resource with **umount** whether is was mounted explicitly using the **mount** command or automatically through the **vfstab** file.

The syntax of the **umount** command, as it relates to unmounting distributed file systems, is:

umount [**-V**] [**-o** *specific_options*] {*resource* **|** *mount_point*}

For **umount**, see the **mount(1M)** manual page for an explanation of the options accepted by the **umount** command.

Example

You want to unmount a remote NFS resource by specifying the server sharing the resource and the pathname of the resource on the server. Type the following command:

```
umount docgroup:/usr/man
```

Mounting a Set of Resources—the mountall Command

DFS Administration lets you mount a set of resources by entering a single command—the **mountall** command. To use the command, you first create a file that lists the resources you want to mount. The syntax of the entries in your file is the same syntax as the entries in the **/etc/vfstab** file, as follows:

special fsckdev mountp fstype fsckpass automnt mntopts fsceiling

See the **vfstab(4)** manual page for a description of the fields in the **/etc/vfstab** file.

Once you create a file, you specify it as the input file for the **mountall** command.

If you do not specify an input file, the **/etc/vfstab** file is used by default.

The syntax of the **mountall** command is as follows:

mountall [**-F** *FSType*] [**-l** **|** **-r**] [*file_system_table*]

See the **mountall(1M)** manual page for an explanation of the options accepted by the **mountall** command.

If the **mountall** command is entered without arguments, it mounts all local file systems and remote resources in the **vfstab** file that have the *automnt* field set to “yes.”

Example

Your system mounts resources automatically when you entered init state 3, using the **/etc/vfstab** file. While resources are still mounted, you add entries to the **vfstab** file. Now you want to mount the resources you added to the file, without exiting init state 3 and unmounting currently mounted resources. Type the following command:

```
mountall -r
```

The system uses the updated **vfstab** file as the input file. It mounts all remote resources in the file that are not currently mounted and displays error messages for those resources that are mounted already.

Unmounting a Set of Resources—the umountall Command

DFS Administration utilizes the **umountall** command, which lets you unmount mounted resources on your system or all the mounted resources of a specified file system type. The syntax of the **umountall** command is the following:

```
umountall [-F FSType] [-k] [-l | -r]
```

For **umountall**, see the **mountall(1M)** manual page for an explanation of the options accepted by the **umountall** command.

Example

You want to unmount all remote file systems currently mounted on your system. You want to kill all processes that have open files. Type the following:

```
umountall -k -r
```

Displaying Information

This section tells you how to determine what resources are available to you from network servers, what resources are shared and mounted on your system, and what resources shared on your system have been mounted by network clients. The commands described in this chapter are the **share**, **mount**, **dfshares**, and **dfmounts** commands.

Displaying Shared Local Resources—the share Command

You use the **share** command to share resources on your system with network clients; however, you can also use the command to display information about shared resources on your system. The command's syntax when used in this capacity is:

```
share [-F fstype]
```

If you enter the command without arguments, it displays all NFS resources on your system that are currently shared. If you specify a file system type and no other options, the command displays all resources of the specified type that are currently shared.

Displaying Mounted Resources—the mount Command

Generally, you use the **`mount`** command to mount local and remote resources on your system. However, if you enter the command without arguments, it displays a list of local and remote resources that are currently mounted on your system.

NOTE

You cannot limit the display to remote resources or to resources of a specific file system type by entering arguments. If you enter **`mount -F`** nfs, for example, the system displays an error message.

Browsing Shared Remote Resources—the dfshares Command

To “browse” shared resources means to determine what resources are available from remote systems. For browsing, DFS Administration gives you the **`dfshares`** command, which displays information about the NFS resources that are available from network servers. The syntax of the **`dfshares`** command is:

```
dfshares [-F fstype] [-h] [-o fs_options] [server . . .]
```

See **`dfshares(1M)`** for an explanation of the options accepted by the **`dfshares`** command.

The syntax of the *server* option depends on the server's file system type. You can use a *system* name, which specifies a system in the domain of the local system, a *domain* name, which specifies *all* the systems in the domain, or you can enter both the domain and the system, in the form *domain.system*, to display resources on a specific remote system in a remote domain.

To display resources on an NFS server, you can enter a *system* name for the server option, which specifies a system on the network.

When you enter the command, your system displays output consisting of the fields:

resource server access transport description

where *resource* is the shared resource name; *server* is the system from which the resource is available; *access* refers to the access granted to client systems—either read/write or read-only; *transport* is the transport provider over which communication is carried, such as tcp; and *description* is a description of the resource provided by an administrator when the resource was shared.

For NFS resources, the command cannot determine the access rights granted to client systems or the transport provider; therefore, the access and transport fields are filled with hyphens (-) when **`dfshares`** displays a list of NFS resources. The description field does not appear at all in a display of NFS resources.

If you enter the **dfshares** command without arguments, the system displays all resources currently shared on your system, along with remote resources that are currently available for mounting.

Example

You want to browse NFS resources on a server called “main.” You do not want the display to include a header. Type the following:

```
dfshares -F nfs -h main
```

The system displays the following output:

```
main:/export        main     -     -
main:/usr/man       main     -     -
main:/usr/share     main     -     -
```

Monitoring Local Resource Use by Remote Systems—the dfmounts Command

The **dfmounts** command shows you which local resources are mounted by clients. The command accepts the following syntax:

dfmounts [**-F** *fstypes*] [**-h**] [**-o** *fs_options*] [*restriction*]

See **dfmounts(1M)** for an explanation of the options accepted by the **dfmounts** command.

If you enter **dfmounts** without arguments, the system displays all local resources currently mounted by remote systems.

The **dfmounts** command displays a header, which is optional, followed by lines consisting of the fields

resource server pathname clients . . .

where *resource* is the shared resource name; *server* is the system from which the resource is available; *pathname* is the pathname of the shared resource as it was given to the **share** command; and *clients* is a list (separated by commas) of clients that have mounted the resource.

Example

You want to see which clients are mounting the NFS resources on the system called “main.” You do not want a header to be displayed. Type the following:

```
dfmounts -F nfs -h main
```

The system displays the following output:

```
-      main      /export         dancer, runner
-      main      /usr/man        jogger
-      main      /usr/share      dancer, runner
```

23
DFS sysadm Interface

23
DFS sysadm Interface

Introduction to the DFS sysadm Interface

The OS provides a menu interface to system administration called System Administration Menus (or *sis-adam*, because it is accessed through the **`sysadm`** command). When you install DFS Administration Utilities, menus are added to the interface for administering distributed file systems. These menus are known collectively as the Distributed File System Management menus. This section describes the DFS Management menus and the options the menus provide.

A complete description of the menu interface, including a tutorial, appears in the chapter “Using the sysadm Interface” in the *System Administration* guide.

The DFS Management Menu Tree

All DFS Administration commands can be accessed through the Distributed File System Management menus in the System Administration Menus.

To display the DFS Management menus, do the following:

1. Type **`sysadm network_services`**.

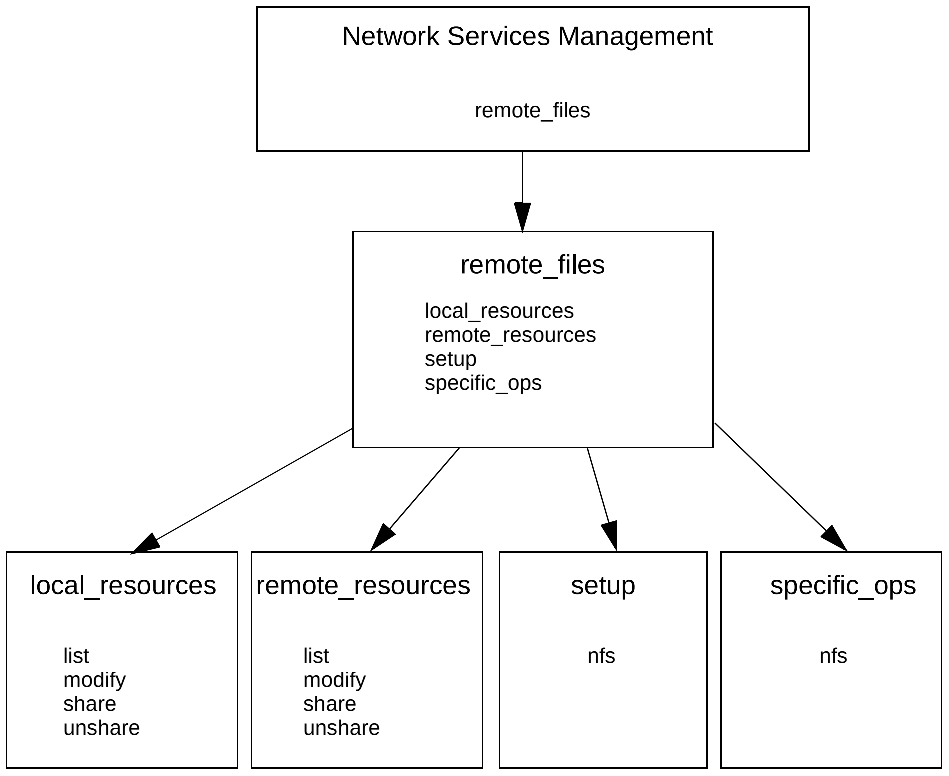
   The Network Services Management menu is displayed.

2. From the Network Services Management menu, select the option `remote_files`.

   The Distributed File System Management menu is displayed.

From the Distributed File System Management menu, you can access submenus that provide you with commands that correspond to the commands described in this guide.

Here is an illustration of **`sysadm`** menus that relate to distributed file system administration:



DFS Menu Options

The commands that correspond to the commands described in this chapter are the commands for administering local and remote resources, which are accessed through the `local_resources` and the `remote_resources` submenus. The `setup` submenu gives you package-specific commands for setting up NFS. The `specific_ops` submenu gives you package-specific commands for administering NFS.

Menu options on the `local_resources` and the `remote_resources` submenus allow you to share, unshare, mount, and unmount resources *immediately*, or to specify resources to be shared and mounted *automatically* whenever your system enters system state 3.

The `local_resources` submenu provides you with the following menu options.

- `list`, which allows you to list the resources on your system currently available to network clients, or that are shared automatically whenever distributed file system operation begins.
- `modify`, which allows you to change the options with which resources on your system are currently shared, as well as the options with which resources are shared when they are shared automatically.

- `share`, which allows you to share selected resources on your system with network clients. The menu option lets you share resources immediately or specify resources to be shared automatically.
- `unshare`, which allows you to unshare resources on your system that are currently shared with network clients. The menu option also lets you cancel the automatic sharing of local resources.

The `remote_resources` submenu provides you with the following menu options:

- `list`, which allows you to list the remote resources that are currently mounted on your system and the remote resources that are mounted automatically.
- `modify`, which allows you to change the options with which remote resources are currently mounted on your system, as well as the options with which remote resources are mounted when they are mounted automatically.
- `mount`, which allows you to mount remote resources on your system and make them available to your users. The menu option lets you mount resources immediately, and specify resources to be mounted automatically every time distributed file system operation begins.
- `unmount`, which allows you to unmount currently mounted remote resources. The menu option also lets you cancel the automatic mounting of remote resources.

Once you select a menu option from either submenu, screen messages and prompts lead you through the task you want to perform. If a message or a prompt is unclear to you, you can display a help screen, which gives you additional information for completing the task.

5
Network File System Administration

Replace with Part 5 tab

24
Introduction to NFS Administration

24 Introduction to NFS Administration

About NFS Administration

The OS allows you to share files, file hierarchies, and entire file systems across a network using the Network File System (NFS).

The NFS administration chapters explain how to administer the implementation of the Network File System. They explain how NFS works, and provide the procedures for sharing resources on the network, and for accessing resources shared by other machines.

How This Part is Organized

The NFS administration chapters assume you are setting up your machine both to share local resources on the network, and to mount remote resources shared by other systems. If you are administering a machine that will mount but not share resources, you can limit your reading to the NFS overview in this chapter, and to the chapters that explain how to mount resources, set up network security, and diagnose problems on an NFS client.

The NFS chapters are organized as follows:

- “Introduction to NFS Administration,” which describes this guide and provides an introduction to NFS.
- “Setting Up NFS,” which directs you to installation instructions and tells you how to start NFS operation and set up automatic sharing and mounting. If you have the Enhanced Security Utilities installed, you are directed to instructions for setting up security on a server running Mandatory Access Control (MAC) and the Least Privilege Mechanism (LPM).
- “Sharing and Mounting NFS Resources Explicitly,” which tells you how to share and mount resources on an “as needed” basis, using the command line.
- “Obtaining NFS Information,” which tells you how to get information about resources that have been shared and mounted across the network.
- “Troubleshooting and Tuning NFS,” which provides solutions to problems you may encounter when using NFS.
- “Setting Up Secure NFS,” which introduces options to NFS that provide for encryption and other security measures.
- “Using the NFS Automounter,” which tells you how to use the automatic mounting facility of NFS.

- "The NFS Network Lock Manager," which discusses how to prevent multiple processes from modifying the same file at the same time.
- "Using the NFS sysadm Interface," which tells you how to set up and administer NFS through a series of **`sysadm(1M)`** menus.

Introduction to NFS

NFS is a service that enables machines of different architectures running different operating systems to share resources across a network. NFS makes it possible for a machine to share local files and directories, and permits remote users to access those files and directories as though they were local to the user's machine.

NFS provides file sharing in a heterogeneous environment, potentially containing many different operating systems; it has been implemented on operating systems including MS-DOS and VMS.

NFS can be implemented on different operating systems because it defines an abstract model of a file system. On each operating system, the NFS model is mapped into the local file system semantics. As a result, normal file system operations, such as read/write, operate in the same way that they operate on a local file system.

The benefits of NFS are significant. NFS allows multiple machines to use the same files, so the same data can be accessible to everyone on the network. Storage costs can be kept down by having machines share applications.

NFS Version 3

NFS Version 3 was an enhancement provided by PowerMAX OS Version 5.1. NFS Version 3 offers substantial enhancements over NFS Version 2. These enhancements include, but are not limited to, the following:

Improved client write throughput.

Reduced server load resulting in increased scalability and performance.

Support for large (multi-gigabyte) files on NFS servers.

To address backward compatibility with the installed base using NFS Version 2, Concurrent has implemented NFS in this release so that both protocols are supported concurrently.

Changes for NFS Version 3 are shown below. See the table below additional information.

- The NFS3 maximum file size is 17,179,869,184GB - 1 = 18,446,744,073,709,551,615 bytes.
- The NFS2 maximum file size is 2GB - 1 =
  2,147,483,647 bytes.

- The NFS3 default write size (wsize) and read size (rsize) = 32KB.
- The NFS2 default write size (wsize) and read size (rsize) = 8KB.

Client PowerMAX OS Version and NFS Version	**Server PowerMAX OS Version and NFS Version**	**Maximum (Default) Write and Read Sizes Supported**
Release 5.1 - NFS3	Release 5.1 - NFS3	32KB[1]
Release 4.3 & below - NFS2	Release 5.1 - NFS3	8KB[1]
Release 5.1 - NFS3	Release 4.3 & below - NFS2	8KB[1]
[1] Concurrent recommends users do not deviate from the default sizes specified for wsize and rsize.		

The maximum file size is of course what NFS itself is capable of. If the user's NFS mounted disk partition is 100MB, then the maximum file size allowed by NFS to that partition is 100MB.

A new feature has been added to the **-o** option of the **mount nfs** command to allow users to specify the particular NFS version, 2 or 3, they which to use. Refer to the **mount_nfs** manpage for more details.

vers=<NFS version

By default, the version of NFS protocol used between the client and the server is the highest one available on both systems. If the NFS server does not support NFS Version 3 protocol, then the NFS mount will use NFS Version 2 protocol. If NFS Version 2 and 3 are both supported and the user wants to use NFS Version 2, specify **vers=2** in the **-o** option of the mount command.

NFS Resources

The objects that can be shared through NFS include any whole or partial directory tree or file hierarchy—including a single file. However, a machine cannot share a file hierarchy that overlaps one that is already shared, and peripheral devices such as modems and printers cannot be shared.

In most UNIX system environments, a file hierarchy that can be shared corresponds to a file system or to a portion of a file system; however, NFS works across operating systems, and the concept of a file system may be meaningless in other, non-UNIX system environments. Therefore, the term *resource* is used throughout the NFS chapters to refer to a file or file hierarchy that can be shared and mounted over NFS.

When you mount a remote file system on a local mount point, you mount the entire file system, starting at its root. When you mount a remote resource through NFS, you are not restricted to mounting the entire file system. You can mount any directory or file in the file system tree, and gain access only to that directory or file and anything beneath it. For example, in the illustration, Machine A is sharing its entire **/usr** file system. If Machine B wants access only to the files and subdirectories in **/usr/man** on Machine A, it can mount **/usr/man,** rather than **/usr;** then, nothing above **usr/man** on Machine A appears in Machine B’s directory tree.

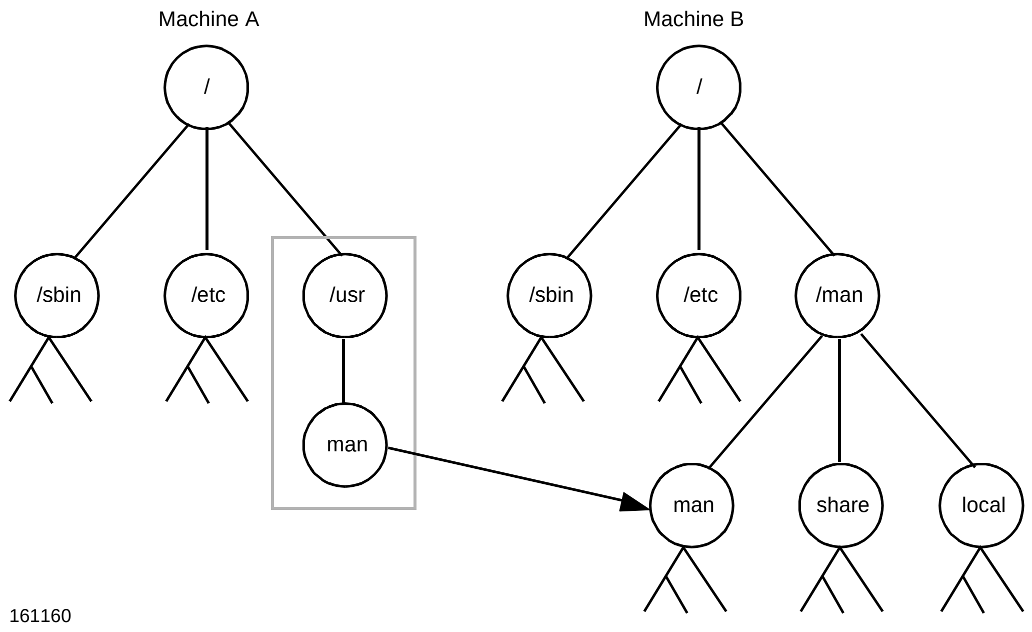


Figure 24-1. Mounting a Remote Resource

It would be invalid for Machine A to share both **`/usr`** and **`/usr/man`** if both resources reside on the same disk partition. In this case, you would have to share **`/usr`** and leave it to network machines to decide whether to mount **`/usr`** or **`/usr/man.`**

If you want to mount a single file, you must mount the file on a directory. Once it is mounted, you cannot remove it (using **`rm`**) or move it to another directory (using **`mv`**); you can only unmount it.

NFS Servers and Clients

A machine that makes its' local resource available for mounting by remote machines is called a *server*. A machine that mounts a resource shared by a remote machine is a *client* of that machine. Any machine with a disk can be a server, a client, or both at the same time.

A server can support a *diskless* client, a machine that has no local disk. A diskless client relies completely on the server for all its file storage. A diskless client can act only as a client—never as a server.

Clients access files on the server by mounting the server's shared resources. When a client mounts a remote resource, it does not make a copy of the resource; rather, the mounting process uses a series of remote procedure calls that enable the client to access the resource transparently on the server's disk. The mount looks like a local mount, and users enter commands just as if the resources were local.

An Overview of NFS Administration

Your responsibilities as an NFS administrator depend on your site's requirements and the role of your machine on the network. You may be responsible for all the machines on your local network, in which case you may be responsible for installing the software on every machine, determining which machines, if any, should be dedicated servers, which should act as both servers and clients, and which should be clients only.

If your site has a network administrator, and you are the administrator of a client-only machine, you may be responsible only for mounting and unmounting remote resources on that machine.

Maintaining a machine once it has been set up involves the following tasks:

- Starting and stopping NFS operation.
- Sharing and unsharing resources.
- Mounting and unmounting resources.
- Modifying administrative files to update the lists of resources your machine shares and/or mounts automatically.
- Checking the status of the network.
- Diagnosing and fixing NFS-related problems as they arise.
- Maintaining security.
- Setting up maps to use the optional automatic mounting facility called the Automounter.

25 Setting Up NFS

25 Setting Up NFS

Introduction to NFS Setup

Setting up NFS involves the following tasks:

- Installing NFS software on your servers and clients (for information on installing the NFS software, see the *Release Notes*).
- Setting up automatic sharing on servers
- Setting up automatic mounting on clients
- Starting NFS operation

The NFS software can be configured in multiple ways to suit the needs of your machine. It can be configured as:

- a client only NFS system
- a server only NFS system
- both a client and server NFS system

There are four NFS modules:

• nfss	- code and structures common to both client and server NFS, static portion
• nfscmn	- code and structures common to both client and server NFS, can be a DLM module
• nfs	- code and structures for client NFS, can be a DLM module
• nfssrv	- code and structures for server NFS, can be a DLM module

The nfss and nfscmn modules must always be enabled to use NFS. At least one of the nfs or nfssrv modules must be enabled. When the NFS software is installed on a machine, it is configured by default as both a client and a server NFS machine (all four modules enabled).

The kernel lock manager module klm must be enabled for client NFS configurations, but it is not used by the server NFS module. Networking must be enabled for all NFS configurations.

Remember, a machine can be both a server and a client—both sharing local resources with remote machines and mounting remote resources. Throughout the NFS chapters, it is assumed you are setting up your machine as both a server and a client. If you are setting up machines that are to be dedicated servers or clients, follow the instructions for sharing and mounting as appropriate.

Starting and Stopping NFS Operation

If NFS is installed on your system, your system will come up in init state 3, and NFS will be started automatically.

If your system is not in init state 3, you can take your system to init state 3 by entering **`init 3,`** or you can set up your system to enter init state 3 immediately on rebooting (for information, see the chapter “Booting and System States” in the *System Administration* guide.)

NFS allows you to share and mount resources *explicitly*, by entering commands at the command line. You can also set up the system to share and mount resources *automatically,* by editing a few of administration files. If you set up automatic sharing and mounting, a pre-determined set of resources is shared and/or mounted whenever you start NFS operation.

To start NFS using the command line, type the following command:

```
sh /etc/init.d/nfs start
```

To stop NFS using the command line, type the following command:

```
sh /etc/init.d/nfs stop
```

When NFS is stopped, the resources you shared and mounted are automatically unshared and unmounted.

If your system does not automatically enter init state 3 upon start-up or reboot, edit the **`/etc/inittab`** file and change the line

```
is:2:initdefault:
```

to the following:

```
is:3:initdefault:
```

See **`inittab(4)`** for more information on the **`/etc/inittab`** file.

Setting Up Automatic Sharing

If you want to set up your system so that certain files and directories are shared automatically whenever you start NFS operation, you need to edit an administrative file. It is recommended that you set up automatic sharing when you first set up NFS if you need to share the same set of resources on a regular basis. For example, if your machine is a server that supports diskless clients, you need to make your clients' root directories available at all times.

The file you need to edit to set up automatic sharing is **`dfstab`**, located in **`/etc/dfs`**. **`dfstab`** is installed by the Distributed File System Administration Utilities.

The **`dfstab`** file lists all the resources that your server shares with its clients and controls which clients may mount a resource. If you want to modify **`dfstab`** to add or delete a resource, or to modify the way sharing is done, simply edit the file with your text editor. The next time the machine enters init state 3, the system reads the updated **`dfstab`** to determine which resources should be shared automatically.

Each line in the file consists of a **`share`** command—the same command you enter at the command line to share a resource explicitly. The **`share`** command is located in **`/usr/sbin`**. When it is used to share a resource over NFS, it has the following syntax:

share [**-F** nfs] [**-o** *specific-options*] [**-d** *description*] *pathname*

See the **`NFS-specific share(1M)`** manual page for the options supported by the **`share`** command.

You can omit the **`-F`** option if

```
nfs    Network File System Utilities: Version x.x
```

is the first line in **`/etc/dfs/fstypes`** or if it is the only Distributed File System (DFS) package installed. If NFS is the only DFS package installed, it becomes the default for the **`-F`** option.

If you enter the **`-d`** option, the description is stored in your **`sharetab`** file. However, clients will not see the description displayed when they use the **`dfshares`** command to list the resources shared on your system.

Under NFS, a server shares resources it owns so clients can mount them. However, a user who becomes the superuser on a client may be denied superuser access to mounted remote resources, unless the appropriate option is used when the resource is shared. Otherwise, when a user logged in as `root` on one host requests access to a remote file shared through NFS, the user's ID is changed from 0 to the user ID of the username `nobody.` The access rights of user `nobody` are the same as those given to the public for a particular file. For example, if the public only has execute permission for a file, then user `nobody` can only execute that file.

When you share a resource, you can permit `root` on a particular machine to have root access to that resource (provided you do not have the Enhanced Security Utilities installed) by editing **`/etc/dfs/dfstab`** on the server, or by specifying the appropriate options on the command line. For example, suppose you want the machine “samba” (but no others) to have superuser access to the shared directory **`/usr/src.`** You enter the following command in **`/etc/dfs/dfstab,`** or on the command line.

```
share -F nfs -o root=samba /usr/src
```

If you want more than one client to have root access, you can specify a list, as follows:

```
share -F nfs -o root=samba:raks:jazz /usr/src
```

If you want all client processes with uid 0 to have superuser access to **`/usr/src,`** you enter

```
share -F nfs -o anon=0 /usr/src
```

`anon` is short for “anonymous.” Anonymous requests, by default, get their user ID changed from its previous value (whatever it may be) to the user ID of username `nobody.` NFS servers label as anonymous any request from a root user (someone whose current effective user ID is 0) who is not in the list following the `root=` option in the **`share`** command. The command above tells the kernel to use the value 0 for anonymous requests. The result is that all root users retain their user ID of 0.

Setting Up Automatic Mounting

Once a resource has been shared on a server through NFS, it can be accessed from a client. Mounting can be done automatically when NFS operation begins on the client (when the client enters init state 3), or explicitly, by executing the NFS start-up script (**`/etc/init.d/nfs`**) on the command line. If you need to mount certain remote resources on a regular basis, it is recommended that you set up automatic mounting when you first set up NFS operation. Automatic mounting is handled through the **`/etc/vfstab`** file.

NOTE

A server can be a client of another server on a local network, in which case, the server's **`vfstab`** may need to include both local and remote mounts. For information about adding local mounts to the **`vfstab`** file, see the chapter “Managing File System Types” in the *System Administration* guide.

If you have the Enhanced Security Utilities installed and you're supporting the Least Privilege Mechanism (LPM), be aware that privileges are assigned to executables locally. When you mount an executable from a server, it is mounted without privileges, regardless of the privileges assigned to the executable on the server. If the executable requires privilege, assign privileges to the executable using the **`filepriv`** command. If you unmount the resource, you should also remove the executable's privileges on the client, using the **`filepriv`** command with the **`-d`** option. If you fail to remove privileges on the client, the system displays an error message every time you re-boot. For more information, see the **`filepriv(1M)`** manual page.

Entries in the **`/etc/vfstab`** file have the following syntax:

special fsckdev mountp fstype fsckpass automnt mntopts fsceiling

See the **`vfstab(4)`** manual page for an explanation of the fields in the **`/etc/vfstab`** file.

The contents of **`/etc/vfstab`** remain the same until you change them.

Automatic Mounting Example

To specify that the directory **`/usr/local`** of the server `dancer` should be mounted automatically on the client's directory **`/usr/local/tmp`** with read/only permission, you add the following line to the client's **`vfstab.`**

```
dancer:/usr/local - /usr/local/tmp nfs - yes ro
```

If you have the Enhanced Security Utilities package installed, you might enter

```
dancer:/usr/local - /usr/local/tmp nfs - yes ro USER_PUBLIC
```

where `USER_PUBLIC` is the level ceiling.

NFS as a Dynamically Loadable Module

NFS can be configured into the system as Dynamically Loadable Modules (DLMs). DLMs are kernel modules (such as device drivers, file systems, and streams modules) that can be loaded into and unloaded from the system during runtime.

By using DLMs, system administrators can avoid the work that was previously necessary (doing an `idbuild,` shutting down the system, bringing it back up, and so on) every time they install or remove a new kernel module.

DLMs also conserve system resources by providing a mechanism for loading kernel modules only when they are needed in the system. When a loaded kernel module is no longer in-use, the DLM feature can unload the module to reclaim the resources it occupied.

Loading and unloading of the NFS DLMs is slightly different than other DLMs. Because NFS daemons will be accessing the NFS DLM modules on a continual basis, the NFS DLM kernel modules will not be dynamically unloaded by the kernel DLM mechanism even though NFS activity becomes infrequent. The NFS DLM modules will be unloaded on demand when the system goes from init state 3 to init state 2. Dynamic and demand unloading of NFS will only take place when all NFS daemons have been killed and all remotely mounted file systems have been unmounted.

Dynamic loading and unloading can be performed on demand by the system administrator, or automatically by the kernel. For more information about the DLM feature, refer to the chapter “Managing Dynamically Loadable Modules” in the *System Administration* guide.

26
Sharing and Mounting NFS Resources Explicitly

26
Sharing and Mounting NFS Resources Explicitly

Sharing and Unsharing NFS Resources

In addition to sharing and unsharing resources automatically, you can share and unshare explicitly by entering commands on the command line. To share resources explicitly, use the **share** and **shareall** commands; to unshare explicitly, use the **unshare** and **unshareall** commands.

Sharing NFS Resources with the share Command

The **share** command lets you share resources as needed. You should use **share** at the command line prompt when you want to share a resource for a brief period of time, or on an irregular basis.

The **share** command is located in **/usr/sbin** and has the following syntax:

share [**-F** nfs] [**-o** *specific-options*] [**-d** *description*] *pathname*

See NFS-specific **share(1M)** for the options supported by the **share** command.

If you enter the **-d** option, the description is stored in your **sharetab** file. However, clients will not see the description displayed when they use the **dfshares** command to list the resources shared on your system.

NFS share Examples

1. To share the directory **/usr** with all the server's clients, enter the following command:

   ```
   share -F nfs /usr
   ```

   The resource is shared with read/write permissions by default.

2. If you want to make sure that client dancer can access **/usr** with read-only permission, enter the following command:

   ```
   share -F nfs -o rw,ro=dancer /usr
   ```

3. If you want to see a listing of the local resources currently available through NFS, enter the following command:

   ```
   share -F nfs
   ```

If you have more than one distributed file system package installed, the **share** command without arguments displays a list of all the resources shared on your system through the default file sharing package, with the default being the first file system type listed in **/etc/dfs/fstypes**.

4. Assume you are running the Enhanced Security Utilities. If you want users or processes on clients jogger and jumper with an effective UID of 0 to access **/usr** with superuser permission, enter the following command:

```
share -F nfs -o root=jogger:jumper /usr
```

5. If you want to permit root access on **/usr** by any user or process whose user ID is 0, enter the following command:

```
share -F nfs -o anon=0 /usr
```

   You would share a resource this way only if you are in a trusting environment.

Sharing a Set of NFS Resources with the shareall Command

The **shareall** command allows you to share a set of resources. To use the command, you first create a file that lists the resources you want to share. The syntax of the entries in your file is the same syntax as the NFS-specific **share** command and the entries in the **dfstab** file, as follows:

share [**-F** nfs] [**-o** *specific_options*] [**-d** *description*] [*pathname*]

Once you create the file, you specify it as the input file when you enter the **shareall** command.

If you do not specify an input file, the **/etc/dfs/dfstab** file is used by default.

The syntax of the **shareall** command is as follows:

shareall [**-F** nfs] [*file*]

See **shareall(1M)** for the options supported by the **shareall** command.

NFS shareall Example

You need to share the same set of resources on a fairly regular basis, but you do not want the resources shared automatically. You create an input file called **misc** that looks like this:

```
#cat misc
share -F nfs -o ro,rw=artdept /export/graphics
share -F nfs /usr/man
share -F nfs -o rw,ro=dancer,root=jogger:jumper /local
```

To share the resources listed in the **misc** file, type

```
shareall misc
```

This example assumes that NFS is the only package installed; the **`-F`** option is omitted from the **`shareall`** command and from the **`share`** commands in the input file.

Unsharing NFS Resources with the unshare Command

Resources that are shared either explicitly or automatically (through the **`/etc/dfs/dfstab`** file) can be unshared at any time by using the command **`unshare.`**

`unshare` is located in **`/usr/sbin`**, and has the following syntax:

```
unshare [-F nfs] pathname
```

where **`-F`** `nfs` indicates the kind of share, and *pathname* is the full name of the shared resource, beginning with root (/). For information on the **`unshare`** command, see NFS-specific **`unshare(1M)`**.

NFS unshare Example

If you want to unshare the directory **`/usr`**, enter this command:

```
unshare -F nfs /usr
```

Unsharing NFS Resources with the unshareall Command

When you want to unshare all the resources that are currently shared on your system through NFS, use the **`unshareall`** command, located in **`/usr/sbin.`** If NFS is the only distributed file system installed, or it is the default file sharing package on your system, simply enter the following command:

```
unshareall
```

If you have more than one distributed file system installed and NFS is not the default, include the **`-F`** option, as follows:

```
unshareall -F nfs
```

For information on the **`unshareall`** command, see **`shareall(1M).`**

Mounting and Unmounting NFS Resources

Once a resource has been shared on a server through NFS, you can explicitly mount and unmount the resource, using the **`mount`** and **`umount`** commands.

Mounting NFS Resources with the mount Command

You can use the **`mount`** command any time during client operations to mount a remote resource, provided the resource is shared and you can reach the server over the network. However, you must be a privileged user to use **`mount`**.

NFS supports two types of mounts—hard mounts and soft mounts. If a mount is a hard mount, an NFS request affecting any part of the mounted resource is issued repeatedly until the request is satisfied (for example, the server crashes and comes back up at a later time). When a mount is a soft mount, an NFS request returns an error if it cannot be satisfied (for example, the server is down), then quits.

Before you issue the **`mount`** command, use the **`mkdir`** command to create a mount point for the remote resource. As with a local mount, if you mount a remote resource on an existing directory that contains files and sub-directories, the contents of the directory are obscured.

If you have the Enhanced Security Utilities installed and you're running the Least Privilege Mechanism (LPM), be aware that privileges are assigned to executables locally. When you mount an executable from a server, it is mounted without privileges, regardless of the privileges assigned to the executable on the server. If the executable requires privilege, assign privileges to the executable using the **`filepriv`** command, just as you would if the executable were local to your system. If, later, you unmount the resource, remove the executable's privileges on the client, using the **`filepriv`** command with the **`-d`** option. If you fail to remove privileges on the client, the system displays an error message every time you re-boot. For more information, see **`filepriv(1M)`**.

The syntax of the **`mount`** command, as it relates to an NFS mount, is:

```
mount [-F fstype] [-o specific_options] {special | mount_point}
mount [-F fstype] [-o specific_options] special mount_point
```

See NFS-specific **`mount(1M)`** for the options supported by the **`mount`** command.

Resources accessed through the **`mount`** command stay mounted, unless you unmount them with the **`umount`** command or exit init state 3. Also, if you exit and re-enter init state 3, the resource will no longer be mounted (unless you edited the **`vfstab`** file to mount the resource automatically.)

When you mount an NFS resource, it is suggested that you do the following:

- Use the `hard` option with any resource you mount read-write. Then, if a user is writing to a file when the server goes down, the write will continue when the server comes up again, and nothing will be lost.
- Use the `nosuid` option with any resource you mount read-write, unless you have good reasons to do otherwise.

NFS mount Examples

1. You want to soft mount on-line manual pages from remote machine `dancer` on the local directory **`/usr/man`**. You want the pages mounted read-only. Type the following command:

```
mount -F nfs -o ro,soft dancer:/usr/man /usr/man
```

2. You want to hard mount the resource **`/usr/local`** from the remote machine `dancer` on the mount-point **`/usr/local/dancer`**. You want the resource mounted read-write, with the set-uid bits ignored and the keyboard interrupt enabled. Enter the following, all on one line:

```
mount -F nfs -o hard,nosuid,intr \
     dancer:/usr/local /usr/local/dancer
```

Unmounting NFS Resources with the umount Command

The **`umount`** command allows you to unmount a remote resource, whether the resource was mounted automatically or explicitly. However, you must be a privileged user to use **`umount`**.

The syntax of **`umount`**, as it relates to NFS, is:

> `umount [`**`-V`**`]` {*server*:*path* **|** *mount-point*}

See **`mount(1M)`** for the options supported by the **`umount`** command.

27
Obtaining NFS Information

27
Obtaining NFS Information

Obtaining Information About NFS Resources

System administrators have at their disposal a variety of tools to obtain information about the status of the networked file system services.

This chapter tells you how to determine what resources are available to you from remote servers, what resources you have shared and mounted on your machine, and what shared resources have been mounted by remote clients.

Browsing Available Resources with dfshares

The command **`dfshares`** allows you to "browse" remote servers to find out the names of remote resources available to your client machine. The syntax of the command is:

```
dfshares [-F nfs] [-h] [server ...]
```

See **`NFS-specific dfshares(1M)`** for the options supported by the **`dfshares`** command.

If NFS is the only file sharing package you have installed, you can omit the **`-F`** option.

Unless it is suppressed with the **`-h`** option, the output from the **`dfshares`** command is preceded by the header:

resource server access transport

where *resource* is of the form *server*:*pathname*, and *server* is the name of the server sharing the resource.

At this time, NFS does not populate the *access* and *transport* fields; a hyphen appears in place of access and transport information.

Displaying Shared Local Resources with share

You can obtain a list of all the NFS resources currently shared on your machine by entering the following command:

```
share
```

If you have more than one distributed file system package installed, the **`share`** command without arguments displays all the resources shared on your system, including the resources shared through NFS.

You do not have to be a privileged user to use the **`share`** command in this capacity. See NFS-specific **`share(1M)`** for the options supported by the **`share`** command.

Monitoring Shared Local Resources with dfmounts

The command **`dfmounts`** lets you display a list of shared resources by server that tells you which resources are currently mounted over NFS, and by which clients. The syntax of the command is shown below:

```
dfmounts [-F nfs] [-h] [server ...]
```

See NFS-specific **`dfmounts(1M)`** for the options supported by the **`dfmounts`** command.

If NFS is the only file sharing package you have installed, you can omit the **`-F`** option.

Unless it is suppressed with the **`-h`** option, the output from the **`dfmounts`** command is preceded by the header:

resource server pathname clients

where *resource* is of the form *server:pathname*; *server* is the name of the server from which the resource was mounted; *pathname* is the pathname of the shared resource as it appears in the second part of *resource*; and *clients* is a commaseparated list of clients that have mounted the resource.

The *server* can be any system on the network. If a *server* is not specified, **`dfmounts`** displays the resources of the local system that have been mounted by remote clients.

28
Troubleshooting and Tuning NFS

28
Troubleshooting and Tuning NFS

Introduction to NFS Troubleshooting and Tuning

This chapter describes problems that may occur on machines using NFS services. It covers the following topics:

- A summary of NFS sequence of events
- Strategies for tracking NFS problems
- NFS-related error messages
- NFS tunable parameters

Before trying to clear NFS problems, you need some understanding of the issues involved. The information in this chapter contains enough technical details to give experienced network administrators a thorough picture of what is happening with their machines. If you do not yet have this level of expertise, note that it is not important to understand fully all the daemons, system calls, and files. However, you should be able to at least recognize their names and functions. Before you read this chapter, familiarize yourself with the following man pages:

```
NFS-specific mount(1M)
NFS-specific share(1M)
mountd(1M)
nfsd(1M)
biod(1M)
```

An Overview of the Mount Process

This section describes the remote mount process. It is not critical that you understand in depth how the daemons mentioned here work; however, you need to know that they exist, because you may need to restart them should they stop for any reason.

Here is the sequence of events when you first enter init state 3. (The **/etc/rc3.d/S22nfs** script contains other commands not related to the mounting process, and they are not shown in the following list.)

1. The **S22nfs** file in **/etc/rc3.d** starts the **nfsd** daemon. The **nfsd** daemon is used in server operations.

2. The same file starts the **biod** daemon. The **biod** daemon is used in client operations.
3. **S22nfs** then starts **mountd.** This is used to process mount requests.
4. It then starts **lockd,** which is used to process lock requests, and **statd,** which interacts with **lockd** to provide crash and recovery functions for the locking services.
5. The same file executes the **shareall** program, which reads the server's **/etc/dfs/dfstab** file, then tells the kernel which resources the server can share and what access restrictions—if any—are on these files.
6. The same file starts the **mountall** program, which reads the client's **vfstab** file and mounts all NFS-type files mentioned there in a manner similar to an explicit mount (described below).

Here is the sequence of events when an administrator mounts a resource during a work session, using the command line:

1. The administrator enters a command, such as

   ```
   mount -F nfs -o ro,soft dancer:/usr/src \
      /usr/src/dancer.src
   ```

2. The **mount** command verifies that the mount point is a full pathname and that there is a file **/usr/lib/fs/nfs/mount. /sbin/mount** then passes all the relevant arguments and options to **/usr/lib/fs/nfs/mount,** which takes control of the process (when we say **mount** from now on in this section, we will be referring to this last file.)
3. **mount** then opens **/etc/mnttab** and checks that the mount was not previously done.
4. **mount** parses the argument **dancer:/usr/src** (*server*:*path*)into host dancer and remote directory **/usr/src.**
5. **mount** calls dancer's **rpcbind** to get the port number of dancer's **mountd.**
6. **mount** calls dancer's **mountd** daemon and passes it **/usr/src,** requesting it to send a file handle (fhandle) for the directory.
7. The server's **mountd** daemon handles the client's mount requests. If the directory **/usr/src** is available to the client (or to the public), the **mountd** daemon does a NFS_GETFH system call on **/usr/src** to get the fhandle, and it sends it to the client's mount process.
8. **mount** checks if **/usr/src/dancer.src** is a directory.
9. **mount** does a **mount(2)** system call with the fhandle and **/usr/src/dancer.src.**
10. The client kernel looks up the directory **/usr/src/dancer.src** and, if everything is in order, ties the file handle to the hierarchy in a mount record.
11. The client kernel looks up the directory **/usr/src** on dancer.

12. The client kernel does a **`statvfs(2)`** call to `dancer`'s NFS server **`nfsd.`**
13. The **`mount(2)`** system call from step 9 returns.
14. **`mount`** opens **`/etc/mnttab`** and adds an appropriate entry to the end, reflecting the new addition to the list of mounted files.
15. When the client kernel does a file operation, once the resource is mounted, it sends the NFS RPC information to the server, where it is read by one of the **`nfsd`** daemons to process the file request.
16. The **`nfsd`** daemons know how a resource is shared from the information sent to the server's kernel by **`share.`** These daemons allow the client to access the resource according to its permissions.

Determining Where NFS Service Has Failed

When tracking down an NFS problem, keep in mind that there are three main points of possible failure: the server, the client, or the network itself. The strategy outlined in this section tries to isolate each individual component to find the one that is not working.

For NFS to operate properly, the NFS daemons shown below must be running. To check the status of the NFS daemons, use **`nfsping`**. The syntax of the **`nfsping`** is:

```
nfsping [-a|-s|-c|-o name]
```

For more information about **`nfsping`**, see **`nfsping(1M)`**.

The **`mountd`** daemon must be running on the server for a remote mount to succeed. Make sure **`mountd`** will be available for an RPC call by checking that **`/etc/init.d/nfs`** has lines similar to the following:

```
if [ -x /usr/lib/nfs/mountd ]
then
        echo > /etc/rmtab
        $TFADMIN /usr/lib/nfs/mountd > /dev/console 2>&1
fi
```

Remote mounts also need the **`nfsd`** daemon to execute on NFS servers. Check the server's file **`/etc/init.d/nfs`** for the following (or similar) lines:

```
if [ -x /usr/lib/nfs/nfsd ]
then
    $TFADMIN /usr/lib/nfs/nfsd -a > /dev/console 2>&1
fi
```

To enable this daemon without rebooting, log in as the network administrator (or as a privileged user) and type:

```
/usr/lib/nfs/nfsd -a
```

The client's **biod** daemon is not necessary for NFS to work, but it improves performance. Make sure the following (or similar) lines are present in the client's file **/etc/init.d/nfs:**

```
    if [ -x /usr/lib/nfs/biod ]
    then
        $TFADMIN /usr/lib/nfs/biod > /dev/console 2>&1
    fi
```

To enable this daemon without rebooting, log in as the network administrator and type:

```
    /usr/lib/nfs/biod
```

Clearing Startup Problems

A potential problem can occur if the name of the machine is changed with the **uname** command.

When the system hostname is changed with the **uname** command, the hostname entries in the **/etc/net/*/hosts** files are not updated. This causes the NFS daemons and/or the **rpcbind** daemon to fail. To check if the NFS daemons and the **rpcbind** daemon are running, use **nfsping -a**. For more information on **nfsping**, see **nfsping(1M)**. Next, check to see if the system hostname (**uname -n**) matches the entry in the **/etc/net/*/hosts** files. For example, if the machine's hostname was `hulk` (use **uname -n**), the entry in each of the **/etc/net/*/hosts** files should look like:

```
    hulkhulk
```

If the first entry in each of the **/etc/net/*/hosts** files do not match the machine's hostname, you must stop NFS, update the first entry in the **/etc/net/*/hosts** files, and restart NFS. For information on starting and stopping NFS, see “Starting and Stopping NFS Operation” on page 25-2. You must also restart the **rpcbind** daemon, to do this, enter:

```
    /usr/sbin/rpcbind
```

Clearing Server Problems

When the network or server has problems, programs that access hard mounted remote files will fail differently than those that access soft mounted remote files. Hard mounted remote resources cause the client's kernel to retry the requests until the server responds again. Soft mounted remote resources cause the client's system calls to return an error after trying for a while. **mount** is like any other program: if the server for a remote resource fails to respond, and the `hard` option has been used, the kernel retries the mount until it succeeds. When you use **mount** with the `bg` option, it retries the mount in the background if the first mount attempt fails.

When a resource is hard mounted, a program that tries to access it hangs if the server fails to respond. In this case, NFS displays the message

```
NFS server hostname not responding, still trying
```

on the console. When the server finally responds, the message

```
NFS server hostname ok
```

appears on the console.

Any program accessing a soft mounted resource whose server is not responding may or may not check the return conditions. If it does, it prints an error message in the form

```
. . . hostname server not responding: RPC: Timed out
```

If a client is having NFS trouble, check first to make sure the server is up and running. From the client, type

```
/usr/bin/rpcinfo server_name
```

to see if the server is up. If it is up and running, it prints a list of program, version, protocol, and port numbers, similar to the following:

```
program   version   netid     address        service     owner
 100000      3      icmp      0.0.0.0.0.111  rpcbind     superuser
 100000      2      icmp      0.0.0.0.0.111  rpcbind     superuser
 100000      3       udp      0.0.0.0.0.111  rpcbind     superuser
 100000      2       udp      0.0.0.0.0.111  rpcbind     superuser
 100000      3       tcp      0.0.0.0.0.111  rpcbind     superuser
 100000      2       tcp      0.0.0.0.0.111  rpcbind     superuser
 100000      3     ticotsord  hulk.rpc       rpcbind     superuser
 100000      3     ticots     hulk.rpc       rpcbind     superuser
 100000      3     ticlts     hulk.rpc       rpcbind     superuser
   .         .        .           .             .            .
   .         .        .           .             .            .
   .         .        .           .             .            .
```

If the server fails to print a list, try to log in at the server's console. If you can log in, check to make sure the server is running **`rpcbind.`**

If the server is up but your machine cannot communicate with it, check the network connections between your machine and the server.

Clearing Remote Mounting Problems

This section deals with problems related to mounting. Any step in the remote mounting process can fail—some of them in more than one way. Below are the error messages you may see and detailed descriptions of the failures associated with each error message.

`mount` can get its parameters explicitly from the command line or from **`/etc/vfstab.`** The examples below assume command line arguments, but the same debugging techniques work if mounting is done automatically through **`/etc/vfstab.`**

- `mount: . . . server not responding: RPC:Program not registered`

  Either the server sharing the resource you are trying to mount is down, or is at the wrong init state, or its **`rpcbind`** is dead or hung, or **`mount`** got through to **`rpcbind,`** but the NFS mount daemon **`mountd`** is not registered. You can check the server's init state by entering (at the server) the following command:

  ```
  who -r
  ```

  If the server is at init state 3, try going to another init state and back, or try rebooting the server to restart **`rpcbind.`** Try to log in to the server from your machine, using the **`rlogin`** command. If you can't log in, but the server is up, try to log in to another remote machine to check your network connection. If that connection is working, check the server's network connection. See **`who(1)`** for more information on the **`who`** command. If **`mountd`** is not running, try restarting NFS. See "Starting and Stopping NFS Operation" on page 25-2 for information on restarting NFS.

- `mount: . . . : No such file or directory`

  Either the remote directory or the local directory does not exist. Check the spelling of the directory names. Use **`ls`** on both directories.

- `mount: not in share list for . . .`

  Your machine name is not in the list of clients allowed access to the resource you want to mount. From the client, you can display the server's share list by entering the following command:

  ```
  dfshares -F nfs server
  ```

  If the resource you want is not in the list, log in to the server and run the **`share`** command without options.

- `mount: . . . : Permission denied`

  This message indicates that the user does not have the appropriate permissions or that some authentication failed on the server. It may be that you are not in the share list (see the preceding error message and explanation), or that the server could not verify that you are who you say you are. Check the server's **`/etc/dfs/sharetab`** file.

- `mount: . . . : Not a directory`

  The local path is not a directory. Check the spelling in your command, and try to run **`ls`** on both directories.

- `mount: mount-point does not exist.`

  The local directory where the resource in being mounted (mount-point) does not exist.

- `nfs mount: access denied for` *host:path*

  This is similar to the `mount: . . . : Permission denied` error message.

- `nfs mount: nfs file system; use` *host:path*

  The command used to mount a resource is missing information.

The **`mount`** command hangs indefinitely if there are no **`nfsd`** daemons running on the NFS server. This happens when there is no **`/etc/dfs/dfstab`** file on the server when it enters init state 3. To clear the problem, restart the **`nfsd`** daemon by executing the following command:

```
/usr/lib/nfs/nfsd
```

If you are mounting resources from a fast server, it is advised that you use the NFS-specific **`mount`** options `rsize` and `wsize`. These options should be set to `rsize=1024` and `wsize=1024` because fast servers cause data overruns on the ethernet driver on slow client machines. One symptom of this problem has the following message being written to the console of the client machines:

```
RPC: Timed out.
```

Another symptom of this problem may be that the client machine appears to be hung, with the following message being written to the console of the client machine:

```
NFS server hostname not responding, still trying
```

Fixing Hung Programs

If programs hang while doing file-related work, your NFS server may be dead. You may see the following message on your console:

```
NFS server hostname not responding, still trying
```

This message indicates that NFS server *hostname* is down, or that there is a problem with the server or with the network.

If your machine hangs completely, check the server(s) from which you mounted the resource. If one or more are down, do not be concerned. When the server comes back up, programs resume automatically. No files are destroyed if the resource was `hard` mounted.

If you soft mount a resource and the server dies, other work should not be affected. Programs that time out trying to access soft mounted remote files will fail with `errno ETIMEDOUT`, but you should still be able to access other resources.

If all servers are running, ask someone else using these same servers if they are having trouble. If more than one machine is having problems getting service, there is a problem with the server. Log in to the server. Use **`nfsping -o nfsd`** to see if **`nfsd`** is running. Enter **`ps -ef | grep nfsd`** a few times to see if **`nfsd`** is accumulating CPU time (let some time pass between each call). If it is not accumulating CPU time, you may be able to kill and then restart **`nfsd.`** If this does not work, you have to reboot the server. If **`nfsd`** is not running, it may be that the server has been taken to a init state that does not support file sharing. Use **`who -r`** to obtain the server's current init state.

If other systems seem to be up and running, check your network connection and the connection of the server.

If programs on the client are hung but the server is up, NFS requests to the server other than reads and writes are succeeding, and messages of the form

```
xdr_opaque: encode FAILED
```

are appearing on either the client or the server console, you may be requesting more data than the underlying transport provider can provide. Try remounting the file system using the **`-o`** `rsize=`*nnn*,`wsize=`*nnn* options to **`mount(1M)`** to restrict the request sizes the client will generate. The maximum NFS Version 2 `rsize` and `wsize`, which is also the default, is 8KB. The maximum NFS Version 3 `rsize` and `wsize`, which is also the default, is 32KB.

Fixing a Machine That Hangs Part Way through Boot

If your machine boots normally, then hangs when it tries to mount resources automatically, most likely one or more servers are down. Use **`init`** to go to single user mode or to a init state that does not mount remote resources automatically. Then start the appropriate daemons in the background and use the **`mount`** command to mount each resource usually mounted automatically through the **`/etc/vfstab`** file. By mounting resources one at a time, you can determine which server is down. To restart a server that is down or hung, see the preceding section.

If you cannot mount any of your resources, most likely your network connection is bad.

Improving Access Time

If access to remote files seems unusually slow, enter the following command:

```
ps -ef
```

on the server and look for abnormal (high) execution times for NFS daemon processes to check that the access time is not being affected adversely by a runaway daemon. If there is nothing unusual (processes with excessive CPU time) in the display, and other clients are getting good response, make sure your **`biod`** daemons are running. At the client, type the following command:

```
ps -ef | grep biod
```

Look for **`biod`** daemons in the display, then enter the command again. If the **`biods`** do not accumulate excessive CPU time, they are probably hung. If they are dead or hung, follow these steps:

1. Kill the daemon processes by determining their process ID (*pid*) numbers:

   ```
   ps -ef | grep biod
   ```

The second field contains the *pid* number. Then enter the following command:

```
kill -9 pid
```

where *pid* is the process ID of the **`biod`** daemon.

2. Restart the **`biod`** daemon by typing:

```
/usr/lib/nfs/biod
```

If **`biod`** is running, check your network connection. The **`netstat -i`** command can help you determine if you are dropping packets; for more information, see **`netstat(1M).`**

Tuning NFS Tunable Parameters

To maximize your system resources, you can modify the following NFS tunable parameters in the **`/etc/conf/cf.d/stune`** file. Note that the **`stune`** file should not be edited directly. To change the value of a parameter in the **`stune`** file, you can use the **`config(1M)`** program. For more information about modifying parameters in the **`stune`** file, see the chapter "Tunable Parameters" in the guide *System Administration.* You may also want to refer to the **`stune(4)`** manual page.

`NRNODE` Specifies the maximum number of `rnode` structures to be allocated for NFS. An `rnode` is a node specific to the NFS file system type. The default is `300` nodes.

`NFS_ASYNC_MAX`
Specifies the maximum number of Light Weight Processes (LWPs) which can be created for asynchronous I/O over NFS. The default value of `8` LWPs is expected to be sufficient for most systems.

`NFS_ASYNC_TIMEOUT`
Specifies, in seconds, the time-to-live of the Light Weight Processes (LWPs) which do asynchronous I/O over NFS. If an asynchronous request is not generated in the system in `NFS_ASYNC_TIMEOUT` seconds, the LWPs exit. The default is `4` seconds.

`NFS_MMAP_TIMEOUT`
Specifies, in seconds, the interval between the time the NFS `mmap` LWP wakes up and the time the NFS `mmap` LWP updates file attributes for ll `mmap`ed files. The default is `30` seconds. Given the large amount of CPU time the `mmap` LWP can potentially consume, in cases where a significant number of files have been `mmap`ed over NFS, this value should not be made too small.

`NFS_MAXCLIENTS`
Specifies the number of active RPC client handles which NFS can cache. The default is `6` clients. This number should be increased only in cases when more memory is added to the system.

`NFSD_TIMEOUT`
Specifies, in seconds, the time-to-live of the Light Weight Processes (LWPs)

which service NFS requests from clients. If an NFS request is not received by the system in `NFSD_TIMEOUT` seconds, the LWPs exit. The default is `8` seconds.

`NFSD_MAX`
: Specifies the maximum number of Light Weight Processes (LWPs) which can be created for servicing NFS requests from clients. The default is `8` LWPs. For dedicated NFS servers, this value should be increased.

`NFSD_MIN`
: Specifies the minimum number of Light Weight Processes (LWPs) which should always exist in the system for servicing NFS requests from NFS clients. The default is `2` LWPs. For dedicated NFS servers, this value should be increased.

`NFS_RETRIES`
: Specifies the maximum number of NFS retries before failing the NFS operation for a soft mount. The default is `5` retries. Typically, this value should not be changed.

`NFS_TIMEO`
: Specifies, in seconds, the maximum initial time to wait for an NFS server to respond to a client request. The default is `10` seconds. On machines connected to “lossy” networks (networks where packets tend to get lost), this value should be changed. On some “lossy” networks you will have greater success getting the packet if you retry earlier; on other networks, you will have greater success if you wait longer.

`NFS_NRA` Specifies the number of pages to read ahead for each `read` operation, if possible. The default is `1` page. Read-ahead over NFS may be turned off by changing this value to `0`.

29
Setting Up Secure NFS

29 Setting Up Secure NFS

Introduction to Secure NFS

NFS is a powerful and convenient way to share resources on a network of different machine architectures and operating systems. However, the same features that make sharing resources through NFS convenient also pose some security problems. An NFS server authenticates a file request by authenticating the machine making the request, but not the user.

When the Enhanced Security Utilities are installed, NFS supports the Enhanced Security features, including Mandatory Access Control (MAC) and the Least Privilege Mechanism (LPM). On a server running MAC and LPM, a client process is allowed access to resources at the same security level, but the user, by default, is granted no privileges to override security restrictions. Privileges granted to the user on the client side are not filtered to the server.

Systems that are not running MAC and LPM do not place similar restrictions on a user process. If superuser privilege is not restricted when a resource is shared, a client user could run **su** and impersonate the owner of a file.

Systems should take advantage of an authentication mechanism provided with NFS. The authentication system exists at the level of Remote Procedure Call (RPC)—the mechanism on which NFS is built. The system, known as Secure RPC, greatly improves the security of network environments and provides additional security to NFS. The security features that Secure RPC provides to NFS are known collectively as Secure NFS.

Because Secure RPC is at the core of Secure NFS, it is necessary to understand how authentication works in RPC to understand Secure NFS. This chapter first presents an overview of Secure RPC, then tells you how to set up Secure NFS. Following the set up procedure are a few important points you should be aware of if you plan to use Secure NFS.

An Overview of Secure RPC

The goal of Secure RPC is to build a system at least as secure as a normal time-sharing system. The way a user is authenticated in a time-sharing system is through a login password. With Data Encryption Standard (DES) authentication, the same is true. Users can log in on any remote machine, just as they can on a local terminal, and their login passwords are used for authentication on the remote machines. For time-sharing systems, the trusted person is the system administrator, who has an ethical obligation not to change a password in order to impersonate someone. In secure RPC, the network administrator is trusted not to alter entries in a database that stores “public keys.

You need to be familiar with two terms to understand an RPC authentication system: *credentials* and *verifiers*. Using ID badges as an example, the credential is what identifies a person: a name, address, birth date, and so on. The verifier is the photo attached to the badge: you can be sure the badge has not been stolen by checking the photo on the badge against the person carrying it. In RPC, the client process sends both credentials and a verifier to the server with each RPC request. The server sends back only a verifier, since the client already knows the server's credentials. If the server doesn't recognize the client's credentials, the server sends back an error message stating that the client sent invalid credentials.

RPC's authentication is open-ended, which means that a variety of authentication systems may be plugged into it. Currently, there are two such systems: the UNIX system, and the Data Encryption Standard (DES) system.

When the UNIX system authentication is used by a network service, the credentials contain the client's machine-name, UID, `gid,` and group-access-list, but the verifier contains nothing. This presents a problem. Because there is no verifier, users could deduce appropriate credentials. Another problem with UNIX system authentication is that it assumes all machines on a network are UNIX system machines. UNIX system authentication breaks down when applied to other operating systems in a heterogeneous network.

To overcome the problems of UNIX system authentication, Secure RPC uses DES authentication—a scheme that employs verifiers, yet allows Secure RPC to be general enough to be used by most operating systems.

DES Authentication

DES authentication uses the Data Encryption Standard (DES) and public key cryptography to authenticate both users and machines in the network. DES is a standard encryption mechanism; public key cryptography is a cipher system that involves two keys: one public and one private.

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 timestamp is encrypted with DES. Two things are necessary for this scheme to work: 1) the two agents must agree on the current time, and 2) the sender and receiver must be using the same encryption key.

If a network runs a time synchronization program, then the time on the client and the server is synchronized automatically. If a time synchronization program is not available, timestamps can be computed using the server's time instead of the network time. The client asks the server for the time 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 timestamps. If the client and server clocks get out of sync to the point where the server begins to reject the client's requests, the DES authentication system resynchronizes with the server.

The client and server arrive at the same encryption key by generating a random *conversation key*, then using public key cryptography to deduce a *common key*. The common key is a key that only the client and server are capable of deducing. The conversation key is used to encrypt and decrypt the client's timestamp; the common key is used to encrypt and decrypt the conversation key.

A Secure RPC Client/Server Session

Here is the series of transactions effected in a client/server session using Secure RPC.

Step 1 Sometime prior to a transaction, the user runs a program that generates a *public key* and a *secret key.* (Each user has a unique public key and secret key.) The public key is stored, in encrypted form, in a public database; the secret key is stored, also in encrypted form, in a private database.

Step 2 The user logs in and runs the **`keylogin`** program (or the **`keylogin`** program may be included in **`$HOME/profile`** so that it runs automatically whenever the user logs in). The **`keylogin`** program prompts the user for a secure RPC password and uses the password to decrypt the secret key. The **`keylogin`** program then passes the decrypted secret key to a program called the Keyserver. (The Keyserver is an RPC service with a local instance on every machine.) The Keyserver saves the decrypted secret key, and waits for the user to initiate a transaction with a server.

Step 3 When the user initiates a transaction with a server

a. the Keyserver randomly generates a *conversation key.*

b. the kernel uses the conversation key to encrypt the client's timestamp (among other things).

c. the Keyserver looks up the server's public key in the public database.

d. the Keyserver uses the client's secret key and the server's public key to create a *common key.*

e. the Keyserver encrypts the conversation key with the common key.

Step 4 The transmission including the timestamp and the conversation key is then sent to the server. The transmission includes a credential and a verifier. The credential contains three things:

- the client's machine name.
- the conversation key, encrypted with the common key.
- a “window,” encrypted with the conversation key.

The window is the difference the client says should be allowed between the server's clock and the client's timestamp. If the difference between the server's clock and the timestamp is greater than the window, the server should reject the client's request. (For secure NFS, the window currently defaults to 30 minutes.)

The client's verifier contains

- the encrypted timestamp.
- an encrypted verifier of the specified window, incremented by 1

The reason the window verifier is needed is this: Suppose somebody wants to impersonate a user and writes a program that, instead of filling in the encrypted fields of the credential and verifier, just stuffs in random bits. The server will decrypt the conversation key into some random key, and use it to try to decrypt the window and the timestamp. The result will just be random numbers. After a few thousand trials, however, there is a good chance that the random window/timestamp pair will pass the authentication system. The window verifier makes guessing the right credential much more difficult.

Step 5 When the server receives the transmission from the client

1. the Keyserver local to the server looks up the client's public key in the public database.
2. the Keyserver uses the client's public key and the server's secret key to deduce the common key—the same common key computed by the client. (No one but the server and the client can calculate the common key, since doing so requires knowing one secret key or the other.)
3. the kernel uses the common key to decrypt the conversation key.
4. the kernel calls the Keyserver to decrypt the client's timestamp with the decrypted conversation key.

Step 6 After the server decrypts the client's timestamp, it stores four things in a credential table:

- the client's machine name.
- the conversation key.
- the window.
- the client's timestamp.

The server stores the first three things for future use. It stores the timestamp to protect against replays. The server will only accept timestamps that are chronologically greater than the last one seen, so any replayed transactions are guaranteed to be rejected.

Step 7 The server returns a verifier to the client, which includes

- the index ID, which the server records in its credential table.
- the client's timestamp minus one, encrypted by conversation key.

The reason for subtracting one from the timestamp is to insure that it is invalid and cannot be reused as a client verifier.

Step 8 The client receives the verifier and authenticates the server. The client knows that only the server could have sent the verifier, since only the server knows what timestamp the client sent.

Step 9 The client returns the index ID to the server in its second transaction and sends another encrypted timestamp.

Step 10 The server sends back the client's timestamp minus 1, encrypted by the conversation key.

With every transaction after the first, the client sends its index ID and another encrypted timestamp, and the server returns the timestamp minus 1.

NOTE

Implicit in these procedures is the name of caller, who must be authenticated in some manner. The Keyserver cannot use DES authentication to do this, since it would create a 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 a set-UID process, owned by root, which makes the request on the part of the client, telling the Keyserver the real UID of the client.

Administering Secure NFS

To use Secure NFS, all the machines for which you are responsible must have a domain name. A *domain* is an administrative entity, typically consisting of several machines, that joins a larger network. If you are running the Network Information Service (NIS), you should also establish the NIS name service for the domain. See Chapter 34, "Network Information Service Administration" for information on setting up NIS.

With UNIX system authentication, the name of a domain is the UID. UIDs are assigned per domain. A problem with this scheme is that UIDs clash when domains are linked across the network. Another problem with UNIX system authentication has to do with superusers; with UNIX system authentication, the superuser ID (uid 0) is assigned one per machine, not one per domain, which means that a domain can have multiple superusers—all with the same uid.

DES authentication corrects these problems by using netnames. A *netname* is simply a string of printable characters created by concatenating the name of the operating system, a user ID, and a domain name. For example, a UNIX system user with a user ID of 508 in the domain `eng.acme.COM` would be assigned the following netname: `unix.508@eng.acme.COM.` Because user IDs are unique within a domain, and because domain names are unique on a network, this scheme produces a unique netname for every user.

To overcome the problem of multiple super-users per domain, netnames are assigned to machines as well as to users. A machine's netname is formed much like a users—by concatenating the name of the operating system and the machine name with the domain name. A UNIX system machine named `hal` in the domain `eng.acme.COM` would have the netname `unix.hal@eng.acme.COM.`

1. Assign your domain a domain name by following the instructions in Chapter 12, "Setting Up TCP/IP", and make the domain name known to each machine in the NIS/RPC domain by using **`domainname`** on each machine. See “The SRPC_DOMAIN Variable” on page 33-3 for further information on **`domainname`**.

2. Either establish public keys and secret keys for your clients' users using the **`newkey`** command, or have each user establish their own public and secret keys using the **`chkey`** command.

   When public and secret keys have been generated, the public keys are stored in the `publickey` database, and users' secret keys are stored in **`/etc/keystore.`** Secret keys for root users are stored in **`/etc/.rootkey.`**

NOTE

For information about these commands, see **`newkey(1M)`** and the **`chkey(1).`**

3. If you choose, put the **`keylogin`** program in **`/etc/profile`** so that it runs automatically whenever a user logs in. Otherwise, make sure users know to run **`keylogin`** when they log in.

4. If you are running NIS, verify that the **`ypbind`** daemon is running and that there is a **`ypserv`** running in the domain.

5. Verify that the **`keyserv`** daemon (the Keyserver) is running by typing

   ```
   ps -ef | grep keyserv
   ```

   If it isn't running, start the Keyserver by typing

   ```
   /usr/sbin/keyserv
   ```

6. Edit the **`/etc/dfs/dfstab`** file and add the `secure` option to the appropriate entries (those that indicate resources you want clients to mount using DES authentication).

7. On each client machine, edit **`/etc/vfstab`** to include `secure` as a mount option in the appropriate entries (those that indicate resources that should be mounted using DES authentication).

NOTE

If a client does not mount as `secure` a resource that is shared as secure, everything works, but users have access as user `nobody,` rather than as themselves.

If you are not running NIS, all users must keep their secret keys synchronized with their login passwords by invoking **`chkey`** if they change their entries in the password database.

When you reinstall, move, or upgrade a machine, remember to save **`/etc/keystore`** and **`/etc/.rootkey.`**

Important Considerations

The following are points you should be aware of if you plan to use Secure RPC:

- If a server crashes when no one is around (after a power failure for example), all of the secret keys that are stored on the system are wiped out. Now no process is able to access secure network services, or mount an NFS file system. The important processes at this time are usually root processes, so things would work if root's secret key were stored away, but nobody is around to type the password that decrypts it. If the machine has a local disk, the solution to the problem is to store root's decrypted secret key in a file, which the Keyserver can read.
- Some systems boot in single-user mode, with a `root` login shell on the console and no password prompt. Physical security is imperative in such cases.
- A problem with diskless clients is that diskless machine booting is not totally secure. It is possible for somebody to impersonate the boot-server, and boot a devious kernel that, for example, makes a record of your secret key on a remote machine. Secure NFS provides 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. This is not considered a serious problem, because it is highly unlikely that somebody would be able to write this compromised kernel without source code. Also, the crime is not without evidence. If you polled the network for boot-servers, you would discover the devious boot-server's location.
- Most set-UID programs are owned by root; since root's secret key is always stored at boot time, these programs will behave as they always have. If a set-UID program is owned by a user, however, it may not always work. For example, if a set-UID program is owned by `dave,` and `dave` has not logged into the machine since it booted, then the program would not be able to access secure network services.
- If you log in to a remote machine (using **`login, rlogin,`** or **`telnet)`** and use **`keylogin`** to gain access, you give away access to your account. This is because your secret key gets passed to that machine's Keyserver, which then stores it. This is only a concern if you don't trust the remote

machine. If you have doubts, however, don't log in to a remote machine if it requires a password. Instead, use NFS to mount resources shared by the remote machine. If you used **`keylogin`** to gain access, you should use **`keylogout(1)`** (for security reasons) on the machine where **`keylogin`** was executed to remove your unencrypted secret key from the Keyserver's address space.

- Using secure NFS can result in some degrading of performance. However, that is the impact on network performance. Not all file operations go over the network, so the impact on total system performance is actually lower. Secure NFS is an optional feature; environments that require higher performance at the expense of security can turn it off.

30
Using the NFS Automounter

30
Using the NFS Automounter

The NFS Automounter

Resources shared through NFS can be mounted using a method called “automounting.” The **`automount`** program (located in **`/usr/lib/nfs/automount`**) mounts and unmounts remote directories on an as-needed basis. Whenever a user on a client running the automounter invokes a command that needs to access a remote file, the shared resource to which that file belongs is mounted automatically. When a certain amount of time has elapsed without the resource being accessed, it is automatically unmounted. For additional information on **`automount`**, see **`automount(1M).`**

Once the automounter is invoked, an administrator does not have to set up automatic mounting with the **`vfstab`** file, or use the **`mount`** and **`umount`** commands at the command line. All mounting is done automatically and transparently.

NOTE

> Mounting some resources with **`automount`** does not exclude the possibility of mounting others with **`mount`**; in fact, in a diskless machine you must mount root (`/`) and **`/usr`** with **`mount`**.

This chapter explains how the automounter works and provides instructions for setting it up.

How the Automounter Works

Unlike **`mount, automount`** does not consult the file **`/etc/vfstab`** for a list of resources to mount automatically. Rather, it consults a series of maps. These maps enable the automounter to locate the appropriate NFS file server, the exported file system, and the mount options for the requested resource.

The automounter mounts everything under the directory **`/tmp_mnt,`** and provides a symbolic link from the requested mount point to the actual mount point under **`/tmp_mnt.`** For instance, if a user wants to mount a remote directory **`src`** under **`/usr/src,`** the actual mount point will be **`/tmp_mnt/usr/src,`** and **`/usr/src`** will be a symbolic link to that location.

When **`automount`** is called, it forks a daemon (**`automountd`**) to serve each mount point in the maps and makes the kernel believe that the mount has taken place. The daemon sleeps until a request is made to access the corresponding resource. In the mount state, the daemon intercepts the request, mounts the remote resource, creates a symbolic link between the requested mount point and the actual mount point under **`/tmp_mnt,`** passes the symbolic link to the kernel, and steps aside.

When a predetermined amount of time has passed with the link not being touched (generally five minutes), the daemon unmounts the resource and resumes its previous position.

Preparing the Automounter Maps

A server never knows, nor cares, whether the files it shares are accessed through **`mount`** or **`automount.`** Nothing, therefore, needs to be done to a server to run the automounter.

A client, however, needs special files for the automounter. As mentioned earlier, **`auto-mount`** does not consult **`/etc/vfstab;`** rather, it consults a map file (or files) specified at the command line.

There are three basic kinds of automounter maps:

- a master map
- a direct map
- an indirect map

The master map is a general map that uses other maps that contain more specific information. When you enter a command to invoke the automounter and you specify the master map, the master map is used to reference direct and indirect maps and get the information it needs to give to the **`automount`** command.

A direct map contains all the information **`automount`** needs to do the mount. It can be called directly by the automounter, or through a master map.

An indirect map allows you to specify alternate servers for a set of resources. It also allows you to specify resources to be mounted as a hierarchy under the same mount point. The indirect map is called only through a master map, which specifies the mount point on which all resources listed in the indirect map should be mounted.

You should create all your automounter maps in your **`/etc`** directory, using any supported text editor.

Conventions

When creating any of the automounter maps, follow these conventions:

- Use a backslash before characters that may confuse the automounter's parser. For example, suppose you want to mount a directory whose name includes a colon, such as `rc0:dk1.` This resource name might result in the map entry

```
/junk -ro vmsserver:rc0:dk1
```

The automounter will be confused by the second colon in this entry. To avoid a problem, the entry should look like this:

```
/junk -ro vmsserver:rc0\:dk1
```

- Use double quotes around a resource name to hide white space.
- If you enter a comment into a map, make sure that it is preceded by #.
- If you include a long entry in a map, use backslashes to split the line into two or more shorter lines. For example, the entry

```
/usr/frame -ro,soft redwood:/usr/frame1.3 balsa:/export/frame
```

could be written as

```
/usr/frame -ro,soft redwood:/usr/frame1.3 \
balsa:/export/frame
```

Writing a Master Map

Each line in a master map has the syntax:

mountpoint map [*mount-options*]

where *mountpoint* is the full pathname of a directory on which resources should be mounted; *map* is the name of the map that lists the resources to be mounted and their locations; and *mount-options* is a comma-separated list of options that regulates the mounting of the entries mentioned in the *map* (unless the *map* entries list other options).

The options that can be specified for *mount-options* are the same options that can be specified with the **`mount`** command, except for **`fg`** and **`bg.`** See the NFS-specific **`mount(1M)`** manual page for information.

The *mountpoint* can be any mount point you have created for a remote mount. Here is a sample entry in a master map:

```
/usr/man /etc/libmap -ro
```

This entry tells the automounter to look in the indirect map **`/etc/libmap`** and to mount everything listed there on **`/usr/man`** on the local system. This entry also tells the automounter to mount resources on **`/usr/man`** read-only; however, if **`libmap`** indicates that a resource should be mounted read-write, it will be mounted read-write.

If the master map calls a direct map, *mount-point* should be `/-`. This tells the automounter to mount the entries in a direct map on the mount point(s) specified in that map (the *location* field in a direct map contains a full pathname).

Below is a sample master map:

```
#Mount-point   Map               Mount-options
/usr/reports   /etc/reportmap    -rw,intr,secure
/usr/man       /etc/libmap       -ro
/-             /etc/direct.map   -ro,intr
```

Writing a Direct Map

All entries in a direct map have the syntax

key [*mountoptions*] *location*

where *key* is the full pathname of the mount point; *mountoptions* is a comma-separated list of options that regulates the mounting of the resource specified in the entry; and *location* is the location of the resource, specified as *server*:*pathname.*

The *mountoptions* can be any of the options that can be specified with the **mount** command, except for the options **fg** and **bg**. For a list of valid options, see the NFS-specific **mount(1M)** manual page.

The following is a sample entry in a direct map:

```
/usr/fun -ro,soft peach:/usr/games
```

This entry means that the remote resource **/usr/games** on the server named peach should be soft mounted read-only on the local mount point **/usr/fun.** Whenever a user tries to access a file or directory that is part of the **/usr/games** directory tree, the automounter reads the direct map, mounts the resource from server peach onto the mount point **/tmp_mnt/usr/fun** on the local system, then creates a symbolic link between **/tmp_mnt/usr/fun** and **/usr/fun.** The user is unaware that the mount operation is taking place, and the resource appears to the user to be at **/usr/fun.**

The following is a typical 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
/var/spool/news            -ro,soft   pine:/var/spool/news
/usr/frame                 -ro,soft   redwood:/usr/frame1.3 \
                                      balsa:/export/frame
```

Note that, in the first entry, more than one mount point is specified, and more than one location is specified. Specifying multiple mounts and locations is discussed later in this chapter.

Writing an Indirect Map

Entries in an indirect map have the syntax

key [*mountoptions*] *location*

where *key* is the name (not the full pathname) of the directory that will be used as the mount point; *mountoptions* is a comma-separated list of options that regulates the mount; and *location* is the location of the resource, specified as *server*:*pathname*.

Once the key is obtained by the automounter, it is suffixed to the mount point associated with it either on the command line or in the master map.

For example, suppose one of the entries in the master map reads:

```
/usr/reports /etc/reportmap -rw,intr,secure
```

Here **`/etc/reportmap`** is the name of the indirect map that lists the remote resources to be mounted under **`/usr/reports.`**

Example of an Indirect Map

Assume that the following map is the master map on our system.

```
/home         /etc/indirect.map
/-            /etc/direct.map
```

The indirect map has been specified as being mounted on **`/home`**, and the indirect map has the following entries:

```
#key      mount-options   location
willow    -rw             willow:/home/willow
cypress   -rw             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 this map resides on host `oak`. If the login `laura` exists on the machine `oak`, and her home directory is specified as **`/home/willow/laura`**, every time user `laura` logs into the system `oak`, the automounter will mount the resource specified in the indirect map (**`willow:/home/willow`**). If the directory `laura` exists on the remote system under **`/home/willow`**, user `laura` will be in her home directory. Thus, when she logs off (and if no one else is using this resource), the resource will be unmounted automatically, after the specified time-out interval (the default is 5 minutes).

NOTE

Any option in the indirect map overrides all options specified in the master map or on the command line.

Specifying Multiple Mounts

An entry in a direct or indirect map can describe multiple mounts, where the mounts can be from different locations and with different mount options. In the sample map shown below, the first entry (which is actually one long entry whose readability has been improved by splitting it into three lines) mounts **`/usr/local/bin`**, **`/usr/local/share`** and **`/usr/local/src`** from the server `ivy`, with the options “readonly” and “soft.” The entry could also read:

```
/usr/local \
               /bin     -ro,soft     ivy:/export/local/sun3 \
               /share   -rw,secure   willow:/usr/local/share \
               /src     -ro,intr     oak:/home/jones/src
```

where the options are different and more than one server is used.

Multiple mounts can be hierarchical. When resources are mounted hierarchically, each resource is mounted on a subdirectory within another resource. When the root of the hierarchy is referenced, the automounter mounts the entire hierarchy. The concept of *root* here is very important. In the case of a single mount, there is no need to specify the root of the mount point, because it is assumed that the location of the mount point is at the mount root or `/`. When mounting a hierarchy, however, the automounter must have a mount point for each mount within the hierarchy. The following illustration shows a true 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
```

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 will issue **`mkdir`** commands to build a path to the first mount point if it is not at the mount root.

NOTE

A true hierarchical mount can be a problem if the server for the root of the hierarchy goes down; any attempt to unmount the lower branches will fail, since the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down.

Specifying Multiple Locations

A mount entry in a direct or indirect map can include more than one location in its *location* field. This means that the mounting can be done from any of the locations specified.

Specifying multiple locations makes sense when you are mounting a resource readonly, since you generally want to have control over the locations of files you write or modify. An example of a read-only resource you might mount from a variety of locations is on-line documentation. In a large network, the current set of on-line manual pages may be available from more than one server. It doesn't matter which server you mount them from, as long as the server is up and running and sharing its files. If manual pages reside in a directory **`/usr/man`** on three different hosts, called `oak,` `rose,` and `willow,` you can mount from any location by specifying the following in the direct map:

```
/usr/man -ro,soft oak:/usr/man rose:/usr/man \
    willow:/usr/man
```

This could also be expressed as a comma-separated list of servers, followed by a colon and the pathname (as long as the pathname is the same on all servers):

```
/usr/man -ro,soft oak,rose,willow:/usr/man
```

From the list of servers, the automounter first selects those that are on the local network and queries or “pings” them. The first server to respond is selected, and an attempt is made to mount from it.

If the server goes down while the mount is in effect, the resource becomes unavailable. A choice here is to wait five minutes until the autounmount takes place and try again; next time around, the automounter will choose one of the available servers. (Another choice is to use the **`umount`** command, inform the automounter of the change in the mount table, and retry the mount; see “Updating the Mount Table” on page 30-13 for more information.)

Specifying Subdirectories

Until now we have used the form *server*:*pathname* to specify a location in a map entry; you can also specify a subdirectory in the *location* field, using the syntax *server*:*pathname*:*directory.*

Assume you have a master map on host `oak` that contains the following entry:

```
/home   /etc/auto.home   -rw, intr, secure
```

Here **`/etc/auto.home`** is the name of an indirect map that contains the entries to be mounted under **`/home.`**

Below is the **`auto.home`** map:

```
#key      mount-options   location

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
john                      willow:/home/willow:john
mary                      willow:/home/willow:mary
joe                       willow:/home/willow:joe
```

Screen 30-1. The auto.home map

Look at the first entry in the map, and assume user Adam has his home directory on host `cypress.` If Adam has an entry in host `oak's` password database specifying his home directory as **`/home/cypress/adam,`** then he can log in to `oak` and the automounter will mount (as **`/tmp_mnt/home/cypress)`** the directory **`/home/cypress`** residing on `cypress.` If one of the directories is `adam,` then Adam will be in his home directory. Because of the options specified in `oak's` master map, Adam's home directory is mounted read/write, interruptible, and secure.

Now look at the last three entries in the indirect map. Note that these entries refer to the same resource on the same host (**`/home/willow`**). They also specify subdirectories in the *location* field (`john`, `mary`, and `joe`).

Now when a user logs in to host `oak` and requests access to the home directory `john` on host `willow,` the automounter mounts **`willow:/home/willow.`** It then places a symbolic link between **`/tmp_mnt/home/willow/john`** and **`/home/john.`**

If user Mary then tries to access her home directory from host `oak,` the automounter sees that **`willow:/home/willow`** is already mounted, so all it has to do is return the link between **`/tmp_mnt/home/willow/mary`** and **`/home/mary.`**

In general, it's a good idea to provide a *subdirectory* entry in the *location* field when different map entries refer to the same resource shared by the same server.

Using Substitutions

If you have a map with a lot of subdirectories specified, as in the following indirect map

```
#key      mount-options   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
         [. . .]
```

you should consider using string substitutions. The ampersand character (&) can be used to substitute the key wherever it appears. Using the ampersand, the above map would look like this:

```
#key      mount-options   location

john                      willow:/home/willow:&
mary                      willow:/home/willow:&
joe                       willow:/home/willow:&
able                      pine:/export/home:&
baker                     peach:/export/home:&
         [. . .]
```

If the name of the server is the same as the key itself, for instance:

```
#key      mount-options   location

willow                    willow:/home/willow
peach                     peach:/home/peach
pine                      pine:/home/pine
oak                       oak:/home/oak
poplar                    poplar:/home/poplar
         [. . .]
```

the use of the ampersand would result in:

```
#key      mount-options   location

willow                    &:/home/&
peach                     &:/home/&
pine                      &:/home/&
oak                       &:/home/&
poplar                    &:/home/&
         [. . .]
```

If all entries in a map have the same format, you can use the catch-all substitute character, the asterisk (`*`), as in the following example:

```
#key      mount-options   location

oak                       &:/export/&
poplar                    &:/export/&
*                         &:/home/&
```

The catch-all key (`*`) matches on everything that reaches this point in the map. Once the automounter reads the catch-all key, it does not continue to read the map.

Using Environment Variables

You can use the value of an environmental variable by prefixing a dollar sign (`$`) to its name. Braces can also be used to delimit the name of the variable from appended characters.

The environmental variables can be inherited from the environment or can be defined explicitly with the **`-D`** option on the command line. For example, if you want each client to mount clientspecific files in the network in a replicated format, you could create a specific map for each client according to its name, so that the relevant line for host `oak` would be:

```
/mystuff   cypress,ivy,balsa:/export/hostfiles/oak
```

and for `willow`:

```
/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 usually isn't feasible. The solution in this case would be to invoke the automounter with a command line similar to the following:

```
automount -D HOST=`ame -n` . .
```

and have the entry in the direct map read:

```
/mystuff   cypress,ivy,balsa:/export/hostfiles/$HOST
```

Now each host would find its own files in the `mystuff` directory, and the task of centrally administering and distributing the maps becomes easier.

Invoking the Automounter

Once the maps are written, you should make sure that there are no equivalent entries in **`/etc/vfstab,`** and that all the entries in the maps refer to NFS shared resources.

The syntax to invoke the automounter is:

```
automount [-mnTv] [-D name=value] [-M mount-directory] \
[-f master-file] [-t sub-options] [directory map [-mount-options]] . . .
```

See **`automount(1M)`** for a complete description of options. The suboptions are the same as those for NFS **`mount,`** with the exception of `bg` (background) and `fg` (foreground), which do not apply.

The default mount point for all mounts is **`/tmp_mnt.`** Like the other names, this is an arbitrary name. It can be changed when you enter the automount command with the **`-M`** option. For example,

```
automount -M /auto . . .
```

causes all mounts to happen under the directory **`/auto,`** which the automounter creates if it doesn't already exist.

The automounter can be invoked in one of the following ways:

1. You can specify all arguments to the automounter without reference to the master map:

   ```
   automount /net /home /etc/indirect_map \
         -rw,intr,secure /- /etc/direct_map -ro,intr
   ```

   `/net` is the directory/mount-point, **`/home`** is the mount point for the indirect map, **`/etc/`***indirect_map* is the location for the *indirect_map* that is to be used, **`-rw`**,intr,secure are the mount options for the *indirect_map*, `/-` means use a *direct_map*, **/etc/***direct_map* is the location for the *direct_map* that is to be used, and **-ro**,intr are the mount options for the *direct_map*.

2. You can specify all arguments in the master map and instruct the automounter to look in it for instructions:

   ```
   automount -f /etc/master_map
   ```

3. You can specify more mount points and maps in addition to those mentioned in the master map:

   ```
   automount -f /etc/master_map /src /etc/auto.src -ro,soft
   ```

4. You can nullify one of the entries in the master map (particularly useful if you are using a map that you cannot modify but does not meet the needs of your machine):

   ```
   automount -f /usr/lib/master_map /home -null
   ```

5. You can override an entry in the master map by specifying a different indirect map on the command line, as follows:

   ```
   automount -f /usr/lib/master_map /home \
       /myown/indirect_map -rw,intr
   ```

   This command tells the automounter to mount **/home** according to instructions in **/myown/***indirect_map*, not according to instructions in the indirect map specified in the master map.

Killing the Automounter

To kill the automounter, first find out the **automount**'s *pid* number by entering:

```
ps -ef | grep automount
```

The second field contains the *pid* number. Then enter the following command:

```
kill pid
```

where *pid* is the automount process ID.

NOTE

automount must not be terminated with the SIGKILL signal (**kill -9**). Without an opportunity to unmount itself, **automount**'s mount points will appear to the kernel as belonging to a non-responding NFS server. The recommended way to terminate **automount** services is to send a SIGTERM (**kill -15**, the default signal) to the daemon. This allows the automounter to catch the signal, and to unmount not only the mount points associated with its daemon, but also to unmount any mounts in

/tmp_mnt. Note that any mounts in **/tmp_mnt** that are busy will not be unmounted.

Updating the Mount Table

If you use **umount** to unmount explicitly one of the automounted resources, have the automounter re-read the **/etc/mnttab** file to update the internal database used by the automounter.

This can be accomplished by sending `SIGTERM` signal to the automount daemon. First, find out the **automount**'s *pid* number by entering:

```
ps -ef | grep automount
```

The second field contains the *pid* number. Then enter the following command:

```
kill pid
```

where *pid* is the automount process ID.

Modifying the Maps

You can modify the automounter maps at any time.

Modifying Master and Direct Maps

The automounter looks at the master and direct maps only when it is invoked. To make changes to a master map or a direct map take effect, perform the following procedure:

1. Terminate the **automount** process, following the procedure described in the section “Killing the Automounter” on page 30-12.
2. Restart the **automount** process, following the procedure described in the section “Invoking the Automounter” on page 30-11.

Modifying Indirect Maps

You do not have to stop and restart the automounter to make changes to an indirect map take effect. Changes to an indirect map will take effect the next time the automounter mounts the modified entry.

Troubleshooting the Automounter

The following are error messages you may see if the automounter fails.

- *mapname*: `Not found`

  The required map cannot be located. This message is produced only when the **`-v`** (verbose) option is given. Check the spelling and pathname of the map name.

- `dir` *mountpoint* `must start with '/'`

  The automounter mount point must be given as full pathname. Check the spelling and pathname of the mount point.

- *mountpoint*: `Not a directory`

  The *mountpoint* exists but it is not a directory. Check the spelling and pathname of the mount point.

- `hierarchical mountpoint:` *mountpoint*

  The automounter will not allow itself to be mounted within an automounted directory. You will have to think of another strategy.

- `WARNING:` *mountpoint* `not empty!`

  The mount point is not an empty directory. This message is produced only when the **`-v`** (verbose) option is given, and it is only a warning. All it means is that the previous contents of *mountpoint* will not be accessible.

- `Can't mount` *mountpoint*: *reason*

  The automounter cannot mount itself at *mountpoint*. The *reason* should be self-explanatory.

- *hostname*:*filesystem* `already mounted on` *mountpoint*

  The automounter is attempting to mount a resource on a mount point, but the resource is already mounted on that mount point. This happens if an entry in **`/etc/vfstab`** is duplicated in an automounter map (either by accident or because the output of **`mount -p`** was redirected to **`vfstab).`** Delete one of the redundant entries.

- `WARNING:` *hostname:filesystem* `already mounted on` *mountpoint*

  The automounter is mounting itself on top of an existing mount point.

- `couldn't create` *directory*: *reason*

  The system could not create a directory. The *reason* should be selfexplanatory.

- `bad entry in map` *mapname* "*map entry*"

  The *map entry* in map *mapname* is incorrect.

- `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.

- *hostname*: `exports:` *rpc_err*

  There is an error when the automounter tries to get a share list from *hostname*. This indicates a server or network problem.

- `host` *hostname* `not responding`

  A resource was trying to be mounted from host *hostname* while the host was down.

- *hostname*:*filesystem* `server not responding`

  A resource that is currently mounted from host *hostname* (that is down) has been accessed.

- `Mount of` *hostname*:*filesystem* `on` *mountpoint*: *reason*

  You will see these error messages after the automounter attempts to mount from *hostname* but gets no response or fails. This may indicate a server or network problem.

- *mountpoint* - *pathname* `from` *hostname*`: absolute symbolic link`

  When mounting a resource, the automounter has detected that *mountpoint* is an absolute symbolic link (beginning with `/`)`.` The content of the link is *pathname*. This may have undesired consequences on the client; for example, the content of the link may be **`/usr.`**

- `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.

- `Remount` *hostname:filesystem on mountpoint:* `server not responding`

  An attempted remount after an unmount failed. This indicates a server problem.

- `NFS server (pid`*n@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 overloaded temporarily, or dead. Wait a few minutes; if the condition persists, the easiest solution is to reboot the client. If you don't want to reboot, exit all processes that make use of automounted resources (or change to a non-automounted resource, in the case of a shell), kill the current automount process, and restart it again from the command line. If this fails, you must reboot.

31
The NFS Network Lock Manager

31 The NFS Network Lock Manager

An Overview of the Network Lock Manager

NFS takes advantage of a network locking facility, accomplished via a user-level daemon called the Network Lock Manager. The Lock Manager supports the style of advisory and mandatory file and record locking—provided via **`lockf()`** and **`fcntl().`** Mandatory file and record locking is not supported over NFS.

Locking prevents multiple processes from modifying the same file at the same time, and allows cooperating processes to synchronize access to shared files. The user interfaces with the network locking service by way of the standard **`fcntl()`** system-call interface, and rarely requires any detailed knowledge of how it works. The kernel maps user calls to the **`fcntl()`** system call or the **`lockf()`** library call into RPC-based messages to the local lock manager. The fact that the file system may be spread across multiple machines is really not a complication—until a crash occurs.

All computers crash from time to time, and in an NFS environment, where multiple machines can have access to the same file at the same time, the process of recovering from a crash is necessarily more complex than in a non-network environment. First of all, locking is inherently stateful (it requires information about locks to be maintained on the server). If a server crashes, clients with locked files must be able to recover their locks. If a client crashes, its servers must release the locks held by processes running on the client. Second, to preserve NFS's overall transparency, the recovery of lost locks must not require the intervention of the applications themselves. This is accomplished as follows:

- Basic file access operations, such as read and write, use a stateless protocol (the NFS protocol). All interactions between NFS servers and clients are atomic—the server doesn't remember anything about its clients from one interaction to the next. In the case of a server crash, client applications will simply sleep until the server comes back up and their NFS operations can complete.
- Stateful services (those that require the server to maintain client information from one transaction to the next), such as the locking service, are not part of the NFS itself. They are separate services that use the status monitor to ensure that their implicit network state information remains consistent with the real state of the network. There are two specific state-related problems involved in providing the locking in a network context:
  1. If the client has crashed, the lock can be held forever by the server.
  2. If the server has crashed, it loses its state (including all its lock information) when it recovers.

The Network Lock Manager solves both of these problems by cooperating with the Network Status Monitor to ensure that it is notified of relevant machine crashes. Its own protocol then allows it to recover the lock information it needs when crashed machines recover.

The lock manager [**`lockd(1M)`**] and the status monitor **`[statd(1M)`**] are both network-service daemons—they run at user level, but they are essential to the kernel's ability to provide fundamental network services, and they are therefore run on all network machines. Like other network-service daemons—which provide, for example, remote-login services (**`rlogind`**)—they are best seen as extensions to the kernel, which, for reasons of space, efficiency, and organization, are implemented as daemons. Application programs that need a network service can either call the appropriate daemon directly with RPC/XDR, or use a system call to call the kernel. In this later case, the kernel will use RPC to call the daemon. The network daemons communicate among themselves with RPC. (See “The Locking Protocol” on page 31-3 for some details about the lock manager protocol.)

It should be noted that the daemon-based approach to network services allows for tailoring by users who need customized services. It is possible, for example, for users to alter the lock manager to provide locking in a different style.

Figure 31-1 depicts the overall architecture of the locking service.

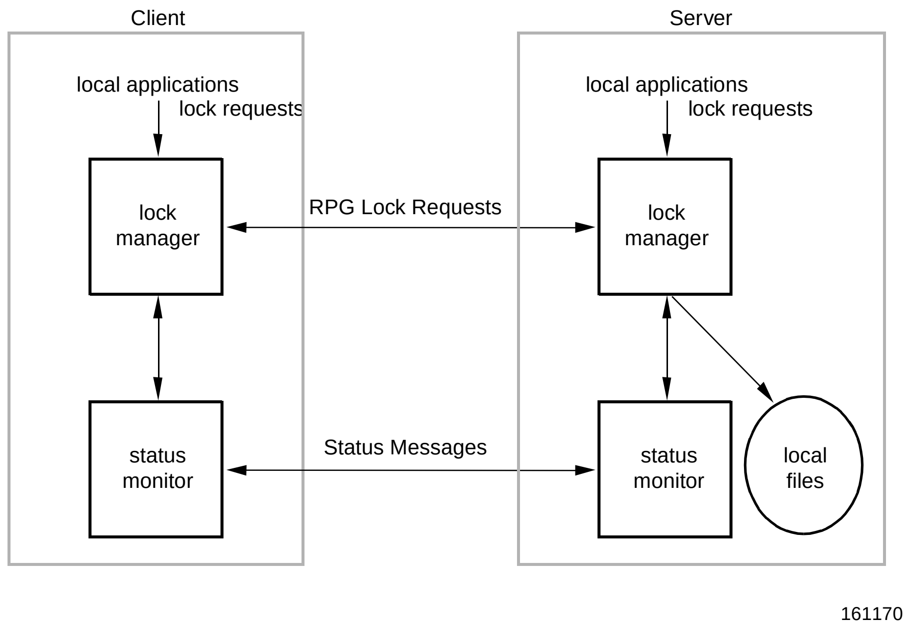


Figure 31-1. Architecture of the Locking Service Over NF

At each server site, a lock manager process accepts lock requests, made on behalf of client processes by a remote lock manager, or on behalf of local processes by the kernel. The client and server lock managers communicate with RPC calls. Upon receiving a remote lock request for a machine that it doesn't already hold a lock on, the lock manager registers its interest in that machine with the local status monitor, and waits for that monitor to notify it that the machine is up. The monitor continues to watch the status of registered machines, and notifies the lock manager if one of them is rebooted (after a crash). If the lock request is for a local file, the lock manager tries to satisfy it, and communicates back to the application along with the appropriate RPC path.

The crash recovery procedure is very simple. If the failure of a client is detected, the server releases the failed client's locks, on the assumption that the client application will request locks again as needed. If the recovery (and, by implication, the crash) of a server is detected, the client lock manger retransmits all the lock requests previously granted by the recovered server. This retransmitted information is used by the server to reconstruct its locking state.

The locking service, then, is essentially stateless. Or to be more precise, its state information is carefully circumscribed within a pair of system daemons that are set up for automatic, application-transparent crash recovery. If a server crashes, and thus loses its state, it expects that its clients will be notified of the crash and that they will send it the information it needs to reconstruct its state. The key in this approach is the status monitor, which the lock manager uses to detect both client and server failures.

For more information about the lock manager daemon, see the **`lockd(1M)`** manual page.

The Locking Protocol

There are four basic kernel-to-Lock Manager requests:

`KLM_LOCK`	Lock the specified record.
`KLM_UNLOCK`	Unlock the specified record.
`KLM_TEST`	Test if the specified record is locked.
`KLM_CANCEL`	Cancel an outstanding lock request.

Despite the fact that the network lock manager adheres to the **`lockf()`** and **`fcntl()`** semantics, there are a few subtle points about its behavior that deserve mention. These arise directly from the nature of the network:

- The first and most important point to be made about the lock manager's behavior has to do with crashes. When an NFS client goes down, the lock managers on all of its servers are notified by their status monitors, and they simply release their locks, on the assumption that the client will request them again when it wants them. When a server crashes, however, matters are different. Clients will wait for the server to come back up. When it does, the server's lock manager will give the client lock managers a grace period to submit lock reclaim requests; during this period, the server will accept only reclaim requests. The client status monitors will notify their respective lock managers when the server recovers. The default grace period is 45 seconds.

- It is possible that, after a server crash, a client will not be able to recover a lock that it had on a file on that server. This can happen for the simple reason that another process may have beaten the recovering application process to the lock. In this case the `SIGLOST` signal will be sent to the process (the default action for this signal is to kill the application).

- The local lock manager does not reply to the kernel lock request until the server lock manager has gotten back to it. Further, if the lock request is on a server new to the local lock manager, the lock manager registers its interest in that server with the local status monitor and waits for its reply. Thus, if either the status monitor or the server's lock manager are unavailable, the reply to a lock request for remote data is delayed until it becomes available.

For more information about OS locking, see the **`fcntl(2)`** and **`lockf(3C)`** manual pages.

The Network Status Monitor

The lock manager relies heavily on a service called the Network Status Monitor to maintain the inherently stateful locking service within the stateless NFS environment. However, the status monitor, implemented as **`statd(1M)`**, is very general and can also be used to support other kinds of stateful network services and applications. Normally, crash recovery is one of the most difficult aspects of network application development, and requires a major design and installation effort. The status monitor simplifies crash recovery.

The status monitor works by providing a general framework for collecting network status information. Implemented as a daemon that runs on all network machines, it provides a simple protocol that allows applications to monitor easily the status of other machines. Its use improves overall robustness, and avoids situations in which applications running on different machines (or even on the same machine) come to disagree about the status of a site—a potentially dangerous situation that can lead to inconsistencies in many applications.

Applications using the status monitor do so by registering with it the machines that they are interested in. The monitor then tracks the status of those machines, and when one of them crashes (actually, when one of them recovers from a crash), it notifies the interested applications to that effect. Then they take whatever actions are necessary to reestablish a consistent state.

This approach provides the following advantages:

- Only applications that use stateful services must pay the overhead—in time and in code—of dealing with the status monitor.
- The implementation of stateful network applications is eased, since the status monitor shields application developers from the complexity of the network.

For more information about the status monitor, see the **`statd(1M)`** manual page.

32
Using the NFS sysadm Interface

32
Using the NFS sysadm Interface

NFS sysadm Interface

The OS provides you with an interface, called **`sysadm`**, that lets you set up and administer NFS through a series of menus. The **`sysadm`** interface not only lets you enter basic NFS configuration information, but it also acts as a tutorial by introducing and explaining key NFS concepts. Once you access a **`sysadm`** menu, help screens provide you with background information and explanations regarding menu selections. Access the help screens by using the HELP function key; use the CANCEL function key to exit the help mode. Continue making menu selections until you complete the particular task.

NOTE

For instruction on moving through the interface, selecting menu options, and so on, see the tutorial in the chapter “Using the sysadm Interface” in the *System Administration* guide.

Here are the NFS procedures you can perform through the **`sysadm`** menu interface.

Procedure 1	Set Up Network File System To perform initial NFS setup.
Procedure 2	Start/Stop Network File System To start and stop NFS and check if it is currently running.
Procedure 3	Local Resource Sharing To manage the local resources you make available to other machines.
Procedure 4	Remote Resource Mounting To manage remote resources made available to your machine.

Procedure 1: Set Up Network File System

This procedure is used to set up NFS on your machine. When the procedure is done, you will have completed everything needed to run NFS on your system. This procedure assumes NFS software is installed on your system, as well as all the software utilities on which NFS depends. For information about installation, see the OS *Release Notes.*

If you have the Enhanced Security Utilities installed, it is assumed you have set up the DFS security database `lid_and_priv.` There is no menu interface to the database or to the **`lidload`** command, which loads the contents of the database into the kernel.

a. Type

   `sysadm network_services`

b. Select `remote_files`; then select `local_resources`. You are now at the following screen:

```
        Local Resource Sharing Management

list          - List Automatically-Currently Shared Local Resources
modify        - Modify Automatic-Current Sharing of Local Resources
share         - Share Local Resources Automatically-Immediately
unshare       - Stop Automatic-Current Sharing of Local Resources
```

Select `list`, then `nfs` to list the local resources currently shared by NFS. Select `modify`, then `nfs` to modify sharing permissions of local resources via NFS. Select `share`, then `nfs` to share local resources via NFS. Select `unshare`, then `nfs` to unshare local resources currently shared via NFS.

Remember, the HELP function key will provide you with help messages along the way. Also, see Chapter 26, "Sharing and Mounting NFS Resources Explicitly" and Chapter 27, "Obtaining NFS Information" for more information about these tasks.

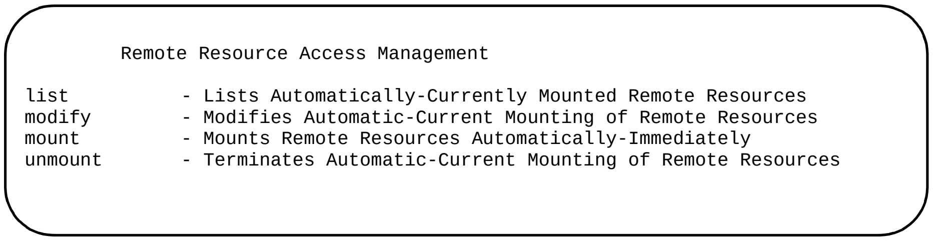
Procedure 4: Remote Resource Mounting

This procedure allows you to make remote resources available or unavailable (mount/unmount) to your local computer, via NFS. With this procedure, you can specify resources to be mounted or unmounted automatically, whenever NFS operation stops and starts, or you can mount and unmount a resource immediately during a work session. You can also modify the options by which remote resources are mounted on your local computer. Finally, this procedure enables you to list the resources of remote systems that are currently available, via NFS, to your machine.

1. Type

   `sysadm network_services`

2. Select `remote_files`, then select `remote_resources`. You are now at the following screen:



```
        Remote Resource Access Management

list          - Lists Automatically-Currently Mounted Remote Resources
modify        - Modifies Automatic-Current Mounting of Remote Resources
mount         - Mounts Remote Resources Automatically-Immediately
unmount       - Terminates Automatic-Current Mounting of Remote Resources
```

Select `list`, then `nfs` to list remote resources mounted via NFS. Select `modify`, then `nfs` to modify mount permissions of remote resources. Select `mount`, then `nfs` to mount remote resources. Select `unmount`, then `nfs` to terminate mounting of remote resources currently shared via NFS.

Remember, the HELP function key will provide you with help messages along the way. Also see the chapters “Sharing and Mounting NFS Resources Explicitly” and “Obtaining NFS Information” for details about these tasks.

6

Remote Procedure Call Administration

Replace with Part 6 tab

33
Remote Procedure Call (RPC) Administration

33
Remote Procedure Call (RPC) Administration

Introduction to RPC Administration

Remote Procedure Call (RPC) administration consists of configuring administration files that:

- Establish name-to-address mapping relationships
- Start server daemons at boot time
- Prompt users for a network password at login (secure RPC)
- Edit a master machine **`/etc/publickey`** file that determines who can access secure RPC services (secure RPC)
- Start ypdaemons

Servers are started at boot time by editable system RC scripts. The file **`/etc/profile`** is edited to call **`keylogin`** to query for a network password at login time.

NIS is currently the recommended default mechanism for administering secure RPC (see Chapter 34, "Network Information Service Administration" for more information).

RPC in the Enhanced Security Environment

RPC administration will be different if you are running the Enhanced Security Utilities. This section describes what you, as RPC administrator, must do to execute RPC administrative commands on a secure system.

Becoming the RPC Administrator

Because of the Least Privileged Mechanism (LPM) of the Enhanced Security Utilities, the concept of the superuser has been replaced by administrative roles. Each administrative role is associated with commands and privileges to administer certain aspects of the system. To administer RPC, you must assume the role of the network administrator, `NET`.

The `NET` role is predefined on your system, but before you can log in as RPC administrator, the system administrator must create the login `root` and assign it the `NET` role. This is done using the **`adminuser`** command.

Any time you want to administer RPC, you must log in as the `root` ID with the appropriate `NET` privileges. For information about assigning roles, see “Basic Security” in *System Administration*, or the **`adminuser(1M)`** manual page.

The tfadmin Command

To perform RPC administrative functions on an LPM system, you must precede each administrative command that requires special privileges with the **`tfadmin`** command. **`tfadmin`** checks the Trusted Facility Management (TFM) databases for the appropriate privileges associated with the `NET` role or other TFM roles to which you may belong. If the privileges are not associated with the command, **`tfadmin`** aborts the command.

For example, if you are administering a non-LPM system and you want to assign a machine to a domain, type

`domainname` *name*

On an LPM system, you would type

`tfadmin domainname` *name*

`tfadmin` is needed only for administrative commands in the TFM database.

NOTE

The examples used in this chapter assume the user is executing RPC commands on a non-LPM system, unless specified otherwise.

RPC Administration Files

RPC and Name-to-Address Mapping

Name-to-address mapping must be in effect for RPC (secure or otherwise) to work. Refer to Chapter 3, "Administering Name-to-Address Mapping" for name-to-address mapping administrative procedures.

System RC File /etc/rc2.d/S75rpc

RPC servers can be started at system boot time. When the system comes up in init state 2, all of the scripts in **`/etc/rc2.d`** are executed. One of these scripts, **`/etc/rc2.d/S75rpc,`** starts the RPC servers.

The system administrator can edit the script to start additional servers.

NOTE

The server **`keyserv`** must also be running for secure RPC to work properly. Administrators may wish to edit the **`/etc/rc2.d/S75rpc`** script and remove the comment character (#) for the **`keyserv`** lines so that **`keyserv`** will start at boot time. If not, **`keyserv`** will have to be started manually.

The /etc/publickey File

Secure RPC information is kept in this file, which is controlled by a domain master server. For each secure RPC user known to a master, this file contains:

- operating system name
- user ID
- RPC domain name
- public key
- secret key

The triple (*operating system, user ID, domain*) forms a unique key into this database of public/secret key pairs that are required by the RPC built-in security protocol.

NOTE

The user ID field in **`/etc/publickey`** may also be a host name. This allows more than one `root` user per domain.

The SRPC_DOMAIN Variable

All machines supporting secure RPC must have what is known as a secure RPC domain name. By default, a machine's secure RPC domain name is null and (because it is null), secure RPC will not work on the machine.

A domain name can be set using the **`domainname(1M)`** command, but it will not be remembered across reboots. For preservation of the name across reboots, administrators need to edit their **`/etc/conf/pack.d/name/space.c`** file to set the `SRPC_DOMAIN` tunable to their desired secure RPC domain name. For example, to change a machine's domain name from null to `finance,` the system administrator would find the line:

```
#define SRPC_DOMAIN=""
```

in **`/etc/conf/pack.d/name/space.c`** and change it to

```
#define SRPC_DOMAIN="finance"
```

After this is done, the kernel must be rebuilt and the system rebooted for the change to take effect. To rebuild the kernel and reboot your system, type:

```
/etc/conf/bin/idbuild -B
cd /; shutdown -y -g0 -i6
```

The above steps will rebuild the kernel, and reboot your system immediately. For more information on the **idbuild** command, see **idbuild(1M).** For more information on the **shutdown** command, see **shutdown(1M)**.

Secure RPC Overview

There is a security protocol, based on DES encryption, built into the RPC package. Remote programs that use secure RPC expect client users to have a public/secret key entry in a shared master **/etc/publickey** file. Access to secure RPC programs is controlled by the **keyserv** daemon which accesses the **/etc/publickey** file when users invoke **keylogin.** One **/etc/publickey** database exists for each secure RPC domain.

Secure RPC users must be given entries in **/etc/publickey** by the RPC administrator before they can use secure RPC programs.

In addition, the administrator of every client machine should edit **/etc/profile** to remove the comment character that has commented out the **keylogin** command; in this way, **keylogin** will be invoked for each user at login time. Thereafter secure RPC commands and programs can be used in the same way ordinary commands and programs are used.

NOTE

Every machine that allows use of secure RPC is a client machine.

One of the secure RPC commands, **chkey,** allows users to change their secure RPC passwords.

The **.profile** files of secure RPC users should be set up to call **keylogout(1)** automatically at the end of a terminal session. For example:

```
# .profile code fragment
trap "keylogout" 0
```

CAUTION

A secure RPC user should always execute **keylogout** before logging off the system. Failure to do so is a serious security infraction.

[See **`sh(1)`** for details on use of `trap` for executing commands at the end of a terminal session.]

NOTE

The presence of secure RPC has no effect on remote programs that do not use the secure protocol. Such programs work normally, whether or not the user is also a secure RPC user.

RPC Domains

All machines using secure RPC must have a secure RPC domain name. One machine per domain acts as master server for the domain. The **`domainname(1M)`** command is used to set a machine's domain name. The machine's `SRPC_DOMAIN` tunable should also be set to the secure RPC domain name. Otherwise, the name is forgotten across reboots. See "The SRPC_DOMAIN Variable" on page 33-3 for information on setting the `SRPC_DOMAIN` variable.

Secure RPC identifies users using a triple (*operating system*, *uid*, *domain*). Thus, users may have multiple registrations with RPC, provided all such triples are unique. For example, a user may belong to more than one *domain*, with *operating system* and *uid* identical for each.

Secure RPC Administration

In general, administering secure RPC is accomplished as follows:

1. A domain name is chosen (for multiple domains, more than one domain name is chosen). Secure RPC domain names are set on participating machines, using the **`domainname(1M)`** command, and the `SRPC_DOMAIN` tunable is set to the secure RPC domain name. See "The SRPC_DOMAIN Variable" on page 33-3.
2. For each user or host to be allowed access to secure RPC services, domain master machine administrators add entries to their master **`/etc/publickey`** file.
3. **`keyserv`** and NIS daemons are started.
4. Administrators start **`keyserv`**, either manually or by means of a boot-time script.
5. The system administrators of client machines remove the comment character that has commented out the **`keylogin`** command from their machine's **`/etc/profile`** and they direct their secure RPC users to add a trap to their **`$HOME/.profile`** so that **`keylogout`** will be called when their sessions end.

The following sections detail this procedure.

NOTE

When slave servers are in use, master servers may have clients as well as slaves.

Establishing Secure RPC Domains

For many networked systems, a single secure RPC domain will suffice. Administrators are notified of the domain name, and they use the **`domainname(1M)`** command to establish that name as the secure RPC domain name for their machine. For example, to set the a machine's domain name to `research`:

```
# domainname research
```

For networked systems having multiple domains, the process is the same, except that two or more different domains will be in use in the network.

Administrators should also set their machine's `SRPC_DOMAIN` tunable to their secure RPC domain name, as described in “The SRPC_DOMAIN Variable” on page 33-3. If this is not done, the domain name will be forgotten across reboots.

A machine can be part of only one domain at any given time. The decision to use single or multiple domains depends on need. In general, the advantages of multiple domains include:

- Duplicate operating system/user ID pairs can be using secure RPC (provided they are in different domains).
- Access to secure RPC programs can be made selective, if some programs are not available to all domains.

The primary advantage of using a single domain is simplified administration.

Master /etc/publickey File

The **`/etc/publickey`** file is a database of public/secret key pairs. The file contains pairs for users and hosts authorized to use secure RPC. Remote procedures that use the DES authentication protocol (built into the RPC package) expect to find public/secret key pairs (for the processes that call them) in **`/etc/publickey.`** A system administrator must therefore add an entry to **`/etc/publickey`** for each user/host to be granted access to secure RPC resources. A single **`/etc/publickey`** file (on a master server or on a collection of master and slave servers) is used and shared over the network by machines having access to the file.

NOTE

Secure RPC programs are not required to be hosted by the same machine that hosts the master **`/etc/publickey`** file. The master **`/etc/publickey`** machine is not necessarily the server for *any* of the secure RPC application programs or commands.

Adding RPC Users with the newkey Command

On the domain master server machine (only), the system administrator grants a user or host access to secure RPC in that domain by adding an entry to the **`/etc/publickey`** file. This is accomplished using the **`newkey(1M)`** command.

NOTE

The **`newkey`** command must be executed on the master server machine by the RPC administrator. Furthermore, prior to using **`newkey,`** the machine's secure RPC domain name must have been set.

For example, to add an entry for the user `alice` the system administrator would enter the following on the master server:

```
master# newkey -u alice
password: password
Re-enter new passwd: password
```

The **`-u`** option signifies that `alice` is a user ID. The domain field for this entry is the domain of the master server on which this command is executed. This is the only way that user `alice` can get access to this particular secure RPC domain.

The **`newkey`** command can also be used with the **`-h`** option to give access to hosts, that is, to `root` users on hosts on the network:

```
master# newkey -h client
password: password
Re-enter new passwd: password
```

Within the domain of secure RPC users having entries in a master **`/etc/publickey`** file, all user names and IDs must be unique. The **`-h`** option is provided to allow more than one `root` user to have access to secure RPC. Because `root` users on different machines have the same name and ID, it would be impossible for more than one of them to be a secure RPC user. The **`-h`** option solves this problem, allowing `root` users to use their unique machine name and address as a user name and ID for RPC purposes.

Network Passwords and the chkey Command

If you are using NIS, client users should be notified of their passwords when they are given access to secure RPC. Their **`.profile`** files should be modified to execute **`keylogout`** when they log off.

Users are prompted for their secure RPC passwords when **`keylogin`** is executed by **`/etc/profile.`**

For example, a user can set up a password as follows:

```
master$ chkey
New password: password
Retype passwd: password
```

Troubleshooting Note

If all administration procedures have been performed correctly and trouble occurs, suspect that an RPC server daemon process (in particular, **`rpcbind)`** may not have been started, may have died, or may have been killed.

7
Network Information Service Administration

Replace with Part 7 tab

34
Network Information Service Administration

34 Network Information Service Administration

Introduction to NIS Administration

This chapter explains how to administer the Network Information Service (NIS)—a distributed network lookup service formerly known as Yellow Pages.

Information in the chapter includes:

- The NIS environment
- Setting up NIS servers
- Setting up an NIS client
- Creating and updating maps
- NIS-related commands
- Fixing NIS problems

What Is NIS?

The Network Information System (NIS) is a distributed name service that replaces copies of commonly replicated configuration files with a centralized management facility. Instead of having to manage each host's files (like **`/etc/hosts`**, **`/etc/passwd`**, etc.), it is only necessary to maintain one version of the file on the central server. Hosts that are using NIS retrieve information as needed from a server system. Changes to an NIS-managed configuration file will be automatically propagated to clients systems on the same network

NIS is best suited for files that have no host-specific information in them and hence are generally the same on all hosts in a network. Examples would be the files **`/etc/group`** and **`/etc/services`**. Some files, like the services files, are ignored entirely once NIS is running. NIS provides all information to hosts from its centralized database. For other files, like **`/etc/group`**, the local file is augmented by information from the centralized database

NIS may also be used for any general data file that is accessed on one or more key fields. Site-specific database files may also be managed by NIS.

The NIS Elements

The NIS service is composed of the following elements:

- domains
- maps
- daemons:
  - **ypserv** -server process [see **ypserv(1M)**]
  - **ypbind** -binding process [for **ypbind**, see **ypserv(1M)]**
  - **ypupdated** - server for changing map entries [see **ypupdated(1M)]**
  - **yppasswdd**-server for changing /etc/passwd entries [see **yppasswdd(1M)]**
- utilities:
  - **ypcat** - lists data in a map [see **ypcat(1)]**
  - **ypwhich** - lists name of NIS server [see **ypwhich(1)]**
  - **ypmatch** - finds a key in a map [see **ypmatch(1)]**
  - **ypinit** - builds and installs an NIS database, or initializes a client Also used to terminate and tear-down NIS**.** [see **ypinit(1M)]**
  - **yppoll** - gets protocol version from server [see **yppoll(1M)]**
  - **yppush** - propagates data from master to slave NIS server [see **yppush(1M)]**
  - **ypset** - sets binding to a particular server [see **ypset(1M)]**
  - **ypxfr** - transfers data from master to slave NIS server [see **ypxfr(1M)]**
  - **makedbm** — creates **dbm** file for an NIS map [see **makedbm(1M)]**

NIS in the Enhanced Security Environment

NIS administration will be different if you are running the Enhanced Security Utilities. This section describes what you, as NIS administrator, must do to execute NIS administrative commands on a secure system.

Becoming the NIS Administrator

Because of the Least Privileged Mechanism (LPM) of the Enhanced Security Utilities, the concept of the NIS administrator has been replaced by administrative roles. Each administrative role is associated with commands and privileges to administer certain aspects of the system. To administer NIS, you must assume the role of the network administrator, `NET`.

The `NET` role is predefined on your system, but before you can log in as NIS administrator, the system administrator must create the login `root` and assign it the `NET` role. This is done using the **`adminuser`** command.

Any time you want to administer NIS, you must log in as the `root` ID with the appropriate `NET` privileges. For information about assigning roles, see “Basic Security” in *System Administration*, or the **`adminuser(1M)`** manual page.

The tfadmin Command

To perform NIS administrative functions on an LPM system, you must precede each administrative command with the **`tfadmin`** command. Before executing a command, **`tfadmin`** checks the Trusted Facility Management (TFM) databases for the appropriate privileges associated with the `NET` role or other TFM roles to which you belong. If the privileges are not associated with the command, **`tfadmin`** aborts the command.

For example, if you are administering a non-LPM system and you want to assign a machine to a domain, type

`domainname` *name*

On an LPM system, you would type

`tfadmin domainname` *name*

Note that **`tfadmin`** is only needed for administrative commands in the TFM database.

The ypbuild Command

Use the **`/var/yp/ypbuild`** command only on systems running the Enhanced Security Utilities. **`/var/yp/ypbuild`** replaces the **`make(1)`** program for converting the NIS maps from ASCII to non-ASCII format. Enter **`ypbuild`** on the command line with its full pathname and the type of shell to be invoked as shown in the following example.

```
/var/yp/ypbuild SHELL=/sbin/sh
```

NOTE

The examples used in this chapter assume the user is executing NIS commands on a non-LPM system, unless specified otherwise.

The NIS Environment

NIS service is based on information contained in NIS maps. Maps are non-ASCII administrative files, which usually derive from ASCII files traditionally found in the **`/etc`** directory. Each NIS map has a mapname used by programs to access it. On a network running NIS, at least one NIS server per domain maintains a set of NIS maps for other hosts in the domain to query.

The service is mediated by the daemons **`ypserv`** and **`ypbind`,** and updates are facilitated by the daemons **`ypupdated`** and **`yppasswdd`**.

The NIS Domain

An NIS domain is an arbitrary name that designates which machines will make use of a common set of maps. Maps for each domain are located in separate directories, **`/var/yp/`***domainname*, on the NIS server (see “NIS Servers” on page 34-4). For example, the maps for machines that belong to the domain `accounting` will be located in the directory **`/var/yp/accounting`** on their corresponding NIS server.

No restrictions are placed on whether a machine can belong to a given domain. Assignment to a domain is done at the local level of each machine by the system administrator logged in as the NIS administrator. See “Establishing the NIS Domain” on page 34-7 for instructions on how to establish the NIS domain.

NIS Machine Types

There are three types of NIS machines:

- master server
- slave server
- client

Any machine can be an NIS client. Servers are generally also clients.

NIS Servers

Servers are divided into master and slave servers. The master server contains the master set of maps that are updated as necessary. Slave servers have copies of the master's set of NIS maps, but cannot modify NIS maps directly. The master server is responsible for all map maintenance and distribution to slave servers.

Either type of server may handle client requests. If you have only one NIS server on the network, it should be designated the master server. Otherwise, the master server should be the system that can best propagate NIS updates with the least performance degradation.

Slave servers allow the system administrator to distribute evenly the load required in answering NIS requests from client systems. A stylized representation of the relationship between master, slaves and clients is shown in Figure 34-1.

A server may be a master in regard to one map, and a slave in regard to another. However, randomly assigning maps to NIS servers can cause a great deal of administrative confusion. You are strongly urged to make a single server the master for all the maps you create within a single domain. The examples in this chapter assume that one server is the master for all maps in the domain.

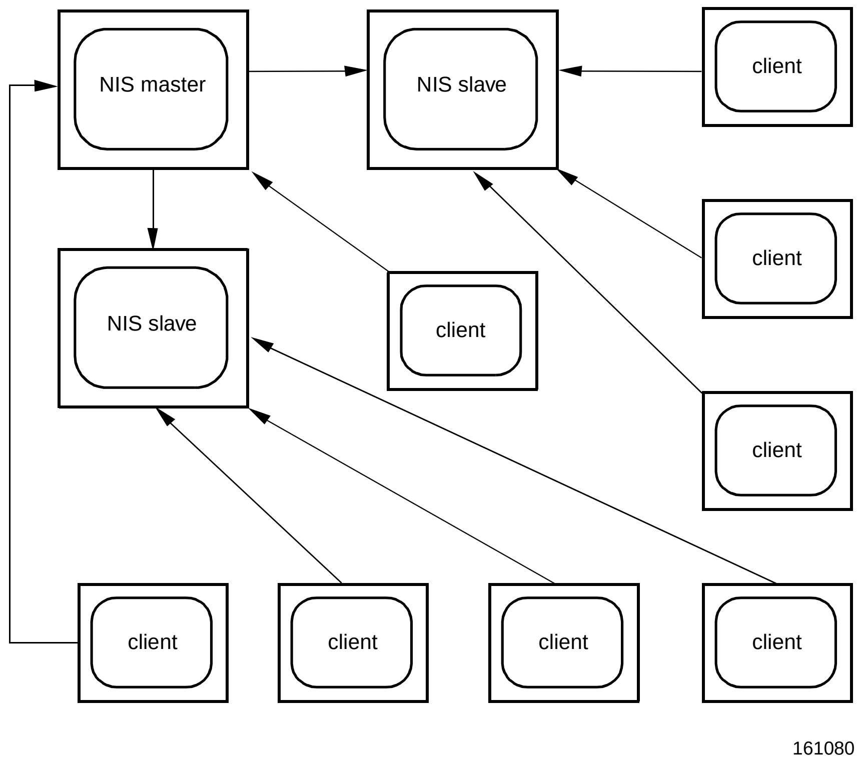


Figure 34-1. Relationships between Master, Slave(s) and Client(s) Machines

NIS Clients

NIS clients run processes that request data from maps on the servers. Clients do not care which server is the master in a given domain, since all NIS servers have the same information. The distinction between master and slave server only applies to where you make the updates.

NIS Binding

NIS clients get information from the NIS server through the binding process. Here is what happens during NIS binding:

1. A program running on the client (that is, a client process) and needing information that is normally provided by an NIS map, asks **`ypbind`** for the name of a server.
2. **`ypbind`** looks in the file **`/var/yp/binding/`***domainname***`/ypservers`** to get a list of the servers for the domain (see “Establishing the NIS Domain” on page 34-7).
3. **`ypbind`** initiates binding to the first server on the list. If the server does not respond, it tries the next, and so on until it finds a server or exhausts the list.

4. **`ypbind`** tells the client process which server to talk to. The client then forwards the request directly to the server.

5. The **`ypserv`** daemon on the NIS server handles the request by consulting the appropriate map.

6. **`ypserv`** then sends the requested information back to the client.

The binding between a client and a server can change with the network's load as the service tries to compensate for current activity; that is, a client may get information from one server at one time and from another server at a different time.

To find out which NIS server is currently providing service to a client, use the **`ypwhich`** command

`ypwhich` *hostname*

where *hostname* is the name of the client. If no *hostname* is mentioned, **`ypwhich`** defaults to the local host (the machine on which the command is entered).

Files Managed by NIS

By default, the following system administrative files will be managed by NIS.

File	Contains
`/etc/passwd`	User names, user IDs and password information.
`/etc/group`	User groups and group IDs.
`/etc/hosts`	Hostnames and IP addresses.
`/etc/networks`	Network addresses.
`/etc/rpc`	Remote procedure call program numbers.
`/etc/services`	Network port numbers and services names.
`/etc/protocols`	Network protocol names and numbers.
`/etc/ethers`	Ethernet numbers.
`/etc/publickey`	Public and private keys used for secure RPC.
`/etc/netgroup`	Netgroup definitions (used by NIS).
`/etc/netmasks`	Network masks.
`/etc/mail/names`	Mail alias names.
`/etc/ucbmail/alias`	Sendmail alias names.

Once NIS is running, it will replace or supplement all of these files. The system administrator may choose to not have one or more of these files managed by NIS. Or, the administrator may want to add additional files to this set. The procedure for doing this is described in “Administering NIS Maps”.

NIS Maps

NIS itself, does not use the above files directly. Instead it will use “maps”. Information in NIS maps is organized in **`dbm(3)`** format. Input to **`makedbm(1M)`** must be in the form of key/value pairs where key is the first word of each line and value is whatever follows in that line. The pairs of keys and values are preserved in the NIS maps so that programs can use the keys to look up the values.

Sometimes an NIS managed file will have more than one map. Each map will use a different key that can be used to locate key/value pairs. For example, there are two maps associated with the **`/etc/passwd`** file

passwd.byname :	looks up data in the passwd database based on username.
passwd.byuid :	looks up data in the passwd database based on user ID.

The most commonly used maps will often have “nicknames” which correspond directly to the name of the original file: for example, the nickname for passwd.byname is simply passwd.

Implementing the NIS

Implementation of the NIS service consists of the following steps:

1. Establishing the domain(s) for your machines.
2. Writing or preparing the maps in ASCII form.
3. Making the maps
4. Choosing NIS servers
5. Setting the master server
6. Setting slave servers
7. Starting up an NIS client
8. NIS masters as clients
9. Terminating NIS
10. Starting daemons in the slave server(s)
11. Initializing the clients

The following sections describe each of these steps.

Establishing the NIS Domain

Before you configure machines as NIS servers or clients, you must prepare the NIS domain by:

- Giving it a name.

  A domain name can be up to 256 characters long. The name of the database directory **`/var/yp/`***domainname* will correspond to the shortened alias for the domain name.

- Designating which machines will serve or be served by the NIS domain.

  Once you have chosen a domain name, make a list of network hosts that will give or receive NIS service within that domain.

- Determining which machine should be master server (you can always change this at a later date).
- Listing which hosts on the network, if any, are to be slave servers.
- Finally, listing all the hosts that are to be NIS clients.

You will probably want all hosts in your network's administrative domain to receive NIS services, although this is not strictly necessary. If this is the case, give the NIS domain the same name as the network administrative domain.

There are several methods used to establish the domain

1. By using the **`domainname(1M)`** command directly such as:

   ```
   domainname 'name'
   ```

   This method is only temporary.

   A better way would be to modify the domain definition in the system start-up script **`/etc/rc2.d/S80nis`**. This script invokes the utility **`domainname(1M)`** with the indicated domain specified.

2. By use of the shell script **`ypinit(1M)`**. **`ypinit(1M)`** is highly recommended as the mechanism used to initialize NIS masters, slaves and clients.

   `ypinit(1M)` will prompt for the name of the domain. With this information it will do -both- steps listed in #1 (i.e. call domainname(1M) and modify **`/etc/rc2.d/S80nis`**).

3. By modifying the file **`/etc/conf/pack.d/name/space.c`** and assigning a name to SRPC_DOMAIN as follows:

   ```
   #define    SRPC_DOMAIN 'name'
   ```

   where name is the name of the domain.

   After this is done, the kernel must be rebuilt and rebooted for the change to take effect.

Preparing the Maps

As released, all files listed in the table in "Files Managed by NIS" will be built into NIS maps. As a site administrator, it may be desirable to not have some of these files handled by NIS or to add additional ones. This can be done by editing the file **`/var/yp/Makefile`** as needed. More on customizing NIS maps will be provided in "Administering NIS Maps".

Making the Maps

It is necessary to convert the ASCII versions of the NIS managed files into the nonASCII files in dbm format that the NIS service expects. The prescribed method to do this is to use the make program using **`/var/yp/Makefile`**.

Initially, this step is not required as the shell script ypinit will do this automatically. ypinit will be described in the following sections.

Choosing NIS Servers

It is important to decide which systems will be NIS servers. Because client systems get almost all of its configuration information from NIS, servers must be highly available and efficient in handling requests. An interruption in NIS service affects all NIS clients.

It is also important that slave servers contain the same data in their copies of the NIS maps as on the master server. NIS contains several mechanisms for making changes to map files and distributing these changes to servers on a regular basis. Described in more detail later in "Administering NIS Maps."

Setting the Master Server

The program that helps you establish NIS masters, slave servers and clients, and permits the initial mapping of ASCII files and their propagation, is **`/usr/sbin/ypinit.`**

You use the shell script **`ypinit`** to build a fresh set of NIS maps on the master server in the following way:

1. Bring the machine that is going to be your master server to single-user mode, or to a mode that is not defined as running the NIS service, and log in as NIS administrator.
2. Type

   ```
   cd /var/yp
   /usr/sbin/ypinit -m
   ```

3. **`ypinit`** prompts for a domain name. The name will be set as described in *"Establishing the Domain Name"*.

4. **`ypinit`** prompts for a list of other hosts that are to become NIS servers. Enter the name of the server you are working on and the names of all other NIS servers.

5. **`ypinit`** will ask if the yppasswdd daemon should be started. When this daemon is enabled, users on server and client may update passwd information on the master using **`yppasswd(1)`** or **`passwd(1)`**.

6. **`ypinit`** asks whether you want the procedure to die at the first non-fatal error or to continue despite non-fatal errors.

   If you choose the first option, **`ypinit`** will exit at the first problem; you can then fix the problem and restart **`ypinit`**.This is recommended if you are running **`ypinit`** for the first time.

If successful, **`ypinit`** will do the following:

- rebuild the maps. This is done using the **`make(1)`** utility with the Makefile in **`/var/yp`**. This builds the map versions of the local administrative files using the **`makedbm(1M)`** utility.
- modify start-up script **`/etc/rc2.d/S80nis`** according to the domain name and options as specified. This script is used to invoke the proper daemons when the system is rebooted.
- invoke the start-up script with the command **`“sh /etc/rc2.d/S80nis start”`**. This immediately gets the proper set of NIS daemons running (i.e. so that a reboot isn't required).
- adds the library **`tcpip_nis.so`** to **`/etc/netconfig.`** This enables name to address translation for hostnames via NIS managed **`/etc/hosts`** files. Refer to chapter called “Administering Name-to-Address Mapping”.

NOTE

For security reasons, you may want to restrict access to the master NIS server.

Setting Slave Servers

Your network can have one or more NIS slave servers. Before actually running **`ypinit`** to create the slave servers, you should take several precautions.

Make sure that the network is working properly before you set up a slave NIS server. In particular, check that you can use **`rcp(1)`** to send files from the master NIS server to NIS slaves. If you cannot, follow the procedures outlined in *User's Guide* to permit the use of **`rcp(1).`**

Now you are ready to create a new slave server.

1. Log in to each slave server as NIS administrator and bring the slave server to a runlevel, preferably single user, that does not imply running the NIS service. **`ypserv`** must not be running.
2. Change directory to **`/var/yp.`**
3. **`ypinit`** will prompt for the domain name. The domain name for each slave must be the same as the domain name of the NIS master server.
4. Type

   `/usr/sbin/ypinit -s` *master*

   where *master* is the host name of the existing NIS master server. Ideally, the named host is the master server, but it can be any host with a stable set of NIS maps, such as another slave server.
5. **`ypinit`** will prompt you for a list of other servers. Enter the names of the NIS servers in order of preference. **`ypinit`** will not run make as it does on the master server. Instead it will call the program **`ypxfr`**, which transfers a copy of the master's NIS map set to the slave server.
6. **`ypinit`** will also invoke the proper daemons for a slave server with a similar procedure as discussed for master servers.

Repeat the procedures above for each machine you want configured as an NIS slave server.

Setting up an NIS Client

The NIS master system must be running to set up clients. To establish a machine as an NIS client, do the following:

1. Edit the client's local files, as you did for the local files in the slave servers, so that processes consulting those files are sent to the NIS maps.
2. Run

   ```
   /usr/sbin/ypinit -c
   ```

   to initialize the client.

With the relevant files in **`/etc`** abbreviated and **`ypbind`** running, the processes on the machine will be clients of the NIS servers.

At this point, you must have configured a NIS server on the network and have given that server's name to **`ypinit.`** Otherwise, processes on the client hang if no NIS server is available while **`ypbind`** is running.

NIS Masters as Clients

As the site administrator, you must decide whether or not the NIS master server should also be an NIS client.

It may be possible to disable NIS clients operations on the master (since all of the local files correspond to the NIS managed files).

By default, all servers are also clients and this is recommended. However, in some situations, it might be desirable to disable NIS clients on the master. This would be done by modifying the /**`etc/rc2.d/S80nis`** script on the master system and setting “startypbind” to 0.

Terminating NIS

The recommended mechanism to terminate NIS on a server or a client is by using the ypinit command with the **`-t`** option.

`ypinit -t` will:

- terminate all of the NIS daemons.
- modify **`/etc/rc2.d/S80nis`** so that NIS daemons are disabled.
- remove NIS libraries from **`/etc/netconfig`**.
- remove domain and bind directories on servers.

Relationship of NIS Maps to Local Files

This section will discuss how the NIS maps relate to the files that they replace.

Default NIS Maps

The following table lists the default set of maps managed by NIS:.

File Name	Accessed By	Map Name	Map Alias	Integration
`/etc/passwd`	User name	passwd.byname	passwd	append
`/etc/passwd`	User id	passwd.byuid		append
`/etc/group`	Group name	group.byname	group	append
`/etc/group`	Group id	group.bygid		append
`/etc/hosts`	Host name	hosts.byname	hosts	replace
`/etc/hosts`	IP address	hosts.byaddr		replace
`/etc/networks`	Net address	networks.byname		replace
`/etc/networks`	IP address	networks.byaddr		replace
`/etc/rpc`	RPC number	rpc.bynumber		replace

File Name	Accessed By	Map Name	Map Alias	Integration
`/etc/services`	Server. name	services.byname	services	replace
`/etc/protocols`	Proto. name	protocols.byname	protocols	replace
`/etc/protocols`	Port number	protocols.bynumber		replace
`/etc/ethers`	Host name	ethers.byname	ethers	replace
`/etc/ethers`	Address	ethers.byaddr		replace
`/etc/publickey`	Key id	publickey.byname		replace
`/etc/netgroup`	User id	netgroup.byuser		replace
`/etc/netgroup`	Host name	netgroup.byhost		replace
`passwd + group`	User id	netid.byname		derived
`/etc/netmasks`	Net address	netmasks.byaddr		-
`/etc/mail/names`	Alias name	mailnames.aliases	mailnames	append
`/etc/ucbmail/alias`	Alias name	mail.alias	aliases	append
`/etc/ucbmail/alias`	Expanded	alias mail.byaddr		append
`NIS servers`	Host name	ypservers		-

As listed, there are often several maps for each file. Each map corresponds to a unique way of accessing a file. For example, the **`passwd.byname`** map looks up data in the passwd database by username. There's also a **`passwd.byuid`** map that is used to look up passwd entries using user id numbers.

Most commonly used maps have nicknames which correspond directly to the name of the original file. For example, the nickname for **`passwd.byname`** is simply **`passwd`**. Nicknames are only recognized by two NIS utilities: **`ypmatch`** and **`ypcat`**.

Integration of Maps with Local files

Its important to understand how NIS maps are integrated into the interface routines that are used to access corresponding local files. Generally, a map will either replace the local file or append to it.

A replaced local file is totally ignored once NIS is operational. All information corresponding to that file will be supplied by NIS. As an example, the interface routines that access **`/etc/hosts`** (i.e. **`gethostent(3N)`**) will totally ignore the local version of **`/etc/hosts`** if NIS is running. Instead, these routines will access the hosts NIS maps via the appropriate server.

Local files that are appended are always consulted first. The information in the local file can be augmented using NIS. Special tokens are inserted into the local file that specify NIS lookups. As an example, the file **`/etc/passwd`** will usually contain a small number of host-specific users. To have the full set of NIS managed users appended, the special token '+' should be added to **`/etc/passwd`**. The interface routines that access the passwd file **getpwent(3C)**) will then use both the small number of host-specific users in addition to the set of NIS users.

Replaced Local Files

The following lists the local files that are replaced when NIS is operational. Note that it is not possible to disable the corresponding interface routines from exclusively using NIS.

```
/etc/hosts
/etc/networks
/etc/rpc
/etc/services
/etc/protocols
/etc/ethers
/etc/publickey
/etc/netgroup
```

Appended Local Files

This section discusses the local files that are appended by NIS.

In general, these files should be stripped to the minimum number of entries needed for booting, in addition to the host-specific entries.

Password File

The following syntax is recognized within the file **/etc/passwd**:

'+'	add entire NIS-managed passwd file.
'+username'	add NIS entry for user username.
'+@group'	add NIS entries for all users in netgroup group.
'-username'	remove NIS entry for user username.
'-@group'	remove NIS entries for users in netgroup group.

Note that if **/etc/passwd** does NOT contain the NIS token '+' then NIS is disabled for **passwd** file entries.

Netgroups, as maintained in **/etc/netgroup**, provides a method for defining a group of users (or groups) for administrative purposes. In the **/etc/passwd** file it can be used to include or exclude an entire set of NIS users previously defined as a netgroup.

It will be necessary to have the same number of fields for an NIS entry in the file **/etc/passwd** as the other entries. Otherwise, the common set of utilities that access the password file may get confused. In addition, it is important that an NIS entry not be considered valid when NIS is not running.

It is recommended that the utilities **`useradd(1M)`**, **`usermod(1M)`** and **`userdel(1M)`** be used for adding NIS entries to the passwd file. These utilities recognize NIS syntax and will do proper checking in addition to building correct NIS entries for the password file.

Group File

The following syntax is recognized within the file **`/etc/group:`**

'+'	add entire NIS-managed group file.
'+groupname'	add NIS entry for group groupname.
'-groupname'	remove NIS entry for group groupname.

Note that if **`/etc/group`** does NOT contain the NIS token '+' then NIS is disabled for group file entries.

As with the passwd file, it is possible to include or exclude an entire set of groups that is defined as a netgroup (within **`/etc/netgroup`**).

It is recommended that the utilities **`groupadd(1M)`** and **`groupdel(1M)`** be used for adding NIS entries to the group file. These utilities recognize NIS syntax and will do proper checking in addition to building correct NIS entries for the group file.

Mail and Sendmail Aliases File

The aliases files used by mail and sendmail (**`/etc/mail/names`** and **`/etc/ucbmail/alias`** respectively) can be augmented with NIS managed aliases.

The syntax recognized in these two files is:

'+'	add entire NIS-managed name/alias file.

Note that if these files do NOT contain the NIS token '+' then NIS is disabled.

Administering NIS Maps

This section describes how to maintain the maps of an existing NIS domain. Subjects discussed include:

- Updating NIS maps
- Propagating an NIS map
- Adding maps to an additional NIS server
- Moving the master map set to a new server

Updating Existing Maps

After you have installed NIS, you will discover that some maps require frequent updating while others never need to change. For example, the hosts map may change frequently on a large company's network. On the other hand, the protocols map probably will change little, if at all.

When you need to update a map, you can use one of two updating procedures, depending on whether the map is standard or non-standard. A standard map is a map in the default set created by **`ypinit`** from the network databases. Non-standard maps may be any of the following:

- A map included with an application purchased from a vendor
- A map created specifically for your site
- A map existing in a form other than ASCII

The following text explains how to use various updating tools. In practice, you probably will use them only if you add non-standard maps or change the set of NIS servers after the system is up and running.

Modifying Standard Maps

Use the following procedure for updating all standard maps:

1. Become NIS administrator on the master server. (Always modify NIS maps on the master server.)
2. Edit the file in **`/etc`** that has the same name as the map you want to change.
3. Type the following:

   ```
   cd /var/yp
   make mapname
   ```

   The **`make`** command will then update your map according to the changes you made in its corresponding file. It will also propagate it among the servers (see "Propagating an NIS Map" on page 34-18 for more information).

NOTE

Do not use this procedure with the **`publickey`** map. Instead, use the **`newkey`** and **`chkey`** commands, as described in "Administering Secure NFS" on page 29-5."

Creating and Modifying Non-Standard Maps

To update a non-standard map, you edit its corresponding ASCII file. Then you rebuild the updated map using the **`/usr/sbin/makedbm`** command. (For information on the

makedbm command, see **makedbm(1M).** If the map has an entry in the **/var/yp/ Makefile,** simply run **make.** If the map does not have an entry, try to create one following the instructions in "Making the Maps" on page 34-9. Using **make** is the preferred method; otherwise, you will have to use **makedbm** by hand.

There are two different methods for using **makedbm:**

- Redirect the command's output to a temporary file, modify the file, then use the modified file as input to **makedbm.**
- Have the output of **makedbm** operated on within a pipeline that feeds into **makedbm** again directly. This is appropriate if you can update the disassembled map with either **awk, sed,** or a **cat** append.

You can use either of two possible procedures for creating new maps. The first uses an existing ASCII file as input; the second uses standard input.

Updating Maps Built from Existing ASCII Files

Assume that an ASCII file **/var/yp/mymap.asc** was created with an editor or a shell script on the NIS master (See "Preparing the Maps" on page 34-9 for information on creating NIS maps). You want to create an NIS map from this file, and locate it in the **home_domain** subdirectory. To do this, you type the following on the master server:

```
cd /var/yp
/usr/sbin/makedbm mymap.asc home_domain/mymap
```

The **mymap** map now exists in the directory **home_domain.**

Adding entries to **mymap** is simple. First, you must modify the ASCII file **mymap.asc.** (If you modify the actual **dbm** files without modifying the corresponding ASCII file, the modifications are lost.) Type the following:

```
cd /var/yp
edit mymap.asc
/usr/sbin/makedbm mymap.asc home_domain/mymap
```

where *edit* is any editor that will allow you to modify an ASCII file. When you finish updating the map, propagate it to the slave servers, as described in "Propagating an NIS Map" on page 34-18.

Updating Maps Built from Standard Input

When no original ASCII file exists, create the NIS map from the keyboard by typing input to **makedbm,** as shown below:

```
ypmaster# cd /var/yp
ypmaster# /usr/sbin/makedbm - home_domain/mymap
key1 value1
key2 value2
key3 value3
<ctl D>
ypmaster#
```

If later you need to modify a map that is not based on an existing file, you can use **makedbm -u** to disassemble the map and create a temporary ASCII intermediate file. You type the following:

```
cd /var/yp
/usr/sbin/makedbm -u home_domain/mymap > mymap.temp
```

The resulting temporary file **mymap.temp** has one entry per line. You can edit it as needed, using your preferred editing tools.

To update the map, you give the name of the modified temporary file to **makedbm** as follows:

```
/usr/sbin/makedbm mymap.temp home_domain/mymap
rm mymap.temp
```

When **makedbm** finishes, propagate the map to the slave servers, as described in “Propagating an NIS Map” on page 34-18.

The preceding paragraphs explained how to use some tools. In reality, almost everything you have to do can be done by **ypinit** and **/var/yp/Makefile,** unless you add non-standard maps to the database or change the set of NIS servers after the system is already up and running.

Whether you use the makefile in **/var/yp** or some other procedure, a new pair of well-formed **dbm** files must end up in the domain directory on the master NIS server.

Propagating an NIS Map

When you *propagate* an NIS map, you move it from place to place—most often from the master to all NIS slave servers. Initially **ypinit** propagates the maps from master to slaves, as described previously. From then on, you must transfer updated maps from master to slaves by running the **ypxfr** command. You can run **ypxfr** three different ways: periodically through the root **crontab** file; by the **ypserv** daemon; and interactively on the command line.

ypxfr handles map transference in tandem with the **yppush** program. **yppush** should always be run from the master server. The makefile in the **/var/yp** directory automatically runs **yppush** after you change the master set of maps.

yppush's function is to copy, or “*push*,” a new version of a NIS map from the NIS master to the slave(s). After making a list of NIS servers from the **ypservers** map built by **ypinit, yppush** contacts each slave server in the list and sends it a "transfer map" request. When the request is acknowledged by the slave, the **ypxfr** program transfers the new map to the slave.

Using crontab with ypxfr

Maps have differing rates of change. For instance, **/etc/services** may not change for months at a time, but **/etc/hosts** may change several times a day in a large organization. When you schedule map transference through the **crontab** command, you can designate the intervals at which individual maps are to be propagated.

To run **ypxfr** periodically at a rate appropriate for your map set, edit root's **crontab** file on each slave server and put the appropriate **ypxfr** entries in it [see the manual page for **crontab(1)**]. **ypxfr** contacts the master server and transfers the map only if the master's copy is more recent than the local copy.

Using Shell Scripts with ypxfr

As an alternative to creating separate **crontab** entries for each map, you may prefer to have root's **crontab** periodically run shell scripts that update the maps. You can easily modify these shell scripts to fit your site's requirements or replace them. Here is an example shell script:

```
#! /sbin/sh
#
# ypxfr_1perday.sh - Do daily yp map check/updates
#

PATH=/bin:/usr/bin:/usr/etc:/usr/etc/yp:$PATH
export PATH

# set -xv
ypxfr ypservers
ypxfr publickey.byname
```

This shell script will update once per day the maps mentioned in it, as long as root's **crontab** executes it once a day (preferably at times of low network load). You can also have scripts update maps once a week, once a month, once every hour, and so on, but be aware of the performance degradation implied in propagating the maps.

Run the same shell scripts through root's **crontab** on each slave server configured for the NIS domain. Alter the exact time of execution from one server to another to avoid bogging down the master.

Three scripts are provided that may be used as templates to update several maps periodically. These scripts are in /**usr/sbin/yp** and are (mnemonically named) **:** **ypxfr_1day**, **ypxfr_2day** and **ypxfr_1hour**. These scripts must be modified to include the maps to be updated. These scripts would also need to be added to root's crontab.

If you want to transfer the map from a particular slave server, use the **-h** *host* option of **ypxfr** within the shell script. The syntax of the commands you put in the script is

```
    /usr/sbin/ypxfr -h host mapname
```

where *host* is the name of the server with the maps you want to transfer, and *mapname* is the name of the requested map. If you use the **-h** option without specifying *host*, **ypxfr** will try to get the map from the master server.

You can use the **-s** *domain* option to transfer maps from another domain to your local domain. These maps should be essentially the same across domains.

Directly Invoking ypxfr

The third method of invoking **ypxfr** is to run it as a command. Typically, you do this only in exceptional situations - for example, when setting up a temporary NIS server to create a test environment, or when trying to make an NIS server that has been out of service consistent with the other servers.

Logging ypxfr's Activities

ypxfr's transfer attempts and the results can be captured in a log file. If a file called **/var/yp/ypxfr.log** exists, results are appended to it. No attempt to limit the size of the log file is made. To prevent it from growing indefinitely, empty it from time to time by entering:

```
    cp /var/yp/ypxfr.log /var/yp/ypxfr.log.old
    cat /dev/null > /var/yp/ypxfr.log
```

You can have **crontab** execute these commands once a week.

To turn off logging, remove the log file.

Password File Updates

The exception to the **yppush** push-on-demand strategy are the passwd maps. Users need to be able to change their passwords without intervention from the system administrator. To change a password when NIS is running, the change must be made on the master server and distributed to all slave servers before the change is visible to all clients.

The **yppasswd(1)** utility is provided for a user to change a password on the NIS master server. For the change to take affect, the **yppasswdd(1M)** daemon must be running on the master server. The script ypinit when invoked for the master will ask whether this daemon should automatically be run. To manually enable or disable **yppasswdd(1M)** the script **/etc/rc2.d/S80nis** should be modified and the "startyppass" set appropriately.

The **yppasswdd(1M)** can also force the password maps to be automatically rebuilt and pushed to the slave servers. Some sites may instead choose not to rebuild and distribute the maps automatically. The script ypinit when invoked for the master will ask whether the automatic propagation should be done. To manually enable or disable **yppasswdd(1M)** the script **/etc/rc2.d/S80nis** should be modified and the **"yppass_make"** set appropriately.

Note also that **passwd(1)** will check to see if the specified password is managed by NIS and will invoke yppasswd if this is the case.

Adding New NIS Maps to the Makefile

Adding a new NIS map entails getting copies of the map's **dbm** files into the **/var/yp/***domain_name* directory on each of the NIS servers in the domain. The actual mechanism is described above in “Propagating an NIS Map” on page 34-18. This section only describes how to update the makefile so that propagation works correctly.

After deciding which NIS server is the master of the map, modify **/var/yp/Makefile** on the master server so that you can conveniently rebuild the map. As indicated previously, different servers can be masters of different maps. This can, however, lead to administrative confusion, and it is strongly recommended that you set only one server as the master of all maps. Actual case-by-case modification is too varied to describe here, but typically a human-readable ASCII file is filtered through **awk, sed,** and/or **grep** to make it suitable for input to **makedbm.** Refer to the existing **/var/yp/Makefile** for examples.

Use the mechanisms already in place in **/var/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, you can have **make** run **yppush** on the NIS master server. The map must be available globally before clients begin to access it.

NOTE

If the map is available from some NIS servers, but not all, you will encounter unpredictable behavior from client programs.

Adding a New NIS Server to the Original Set

After NIS is running, you may need to create an NIS slave server that you did not include in the initial set given to **ypinit.** The following procedure explains how to do this:

1. Log in to the master server as NIS administrator.
2. Go to the NIS domain directory by typing:

   cd /var/yp/*domain_name*

3. Disassemble **ypservers,** as follows:

   /usr/sbin/makedbm -u ypservers > /tmp/*temp_file*

   makedbm converts **ypservers** from **dbm** format to the temporary ASCII file **/tmp/***temp_file.*

4. Edit **/tmp/***temp_file* using your preferred text editor. Add the new slave server's name to the list of servers. Then save and close the file.

5. Run the **makedbm** command with **temp_file** as the input file and **ypservers** as the output file.

   ```
   /usr/sbin/makedbm /tmp/temp_file ypservers
   ```

   Here **makedbm** converts **ypservers** back into **dbm** format.

6. Verify that the **ypservers** map is correct (since there is no ASCII file for **ypservers)** by typing the following:

   ```
   ypslave# /usr/sbin/makedbm -u ypservers
   ```

   NOTE

   If a host name is not in **ypservers**, it will not be warned of updates to the NIS map files.

   Here **makedbm** will display each entry in **ypservers** on your screen.

7. Set up the new slave server's NIS domain directory by copying the NIS map set from the master server. To do this, log in to the new NIS slave as NIS administrator and run the **ypinit** command:

```
ypslave# cd /var/yp
ypslave# ypinit -c
< enter the list of servers >
ypslave# /usr/lib/netsvc/yp/ypbind
ypslave# /usr/sbin/ypinit -s ypmaster
```

When you are finished, complete Steps 5 and 6 in “Setting Slave Servers” on page 34-10.

Changing a Map's Master Server

To change a map's master, you first have to build it on the new NIS master. The old master's name occurs as a key-value pair in the existing map (this pair is inserted automatically by **makedbm).** Therefore, using the existing copy at the new master or transferring a copy to the new master with **ypxfr** is insufficient. You have to reassociate the key with the new master's name. If the map has an ASCII source file, you should copy it in its current version to the new master.

Here are instructions for remaking a sample NIS map called **jokes.bypunchline.**

1. Log in to the new master as NIS administrator and type the following:

   ```
   newmaster# cd /var/yp
   ```

2. **`/var/yp/Makefile`** must have an entry for the new map before you specify the map to **`make.`** If this isn't the case, edit the makefile now (see "Making the Maps" on page 34-9).

3. Type the following:

   ```
   newmaster# make jokes.bypunchline
   ```

4. If the old master will remain an NIS server, **`rlogin`** in to it and edit **`/var/yp/Makefile.`** Comment out the section of **`/var/yp/Makefile`** that made **`jokes.bypunchline`** so that it is no longer made there.

5. If **`jokes.bypunchline`** only exists as a **`dbm`** file, remake it on the new master by disassembling a copy from any NIS server, then running the disassembled version through **`makedbm:`**

   ```
   newmaster# cd /var/yp
   newmaster# ypcat -k jokes.bypunchline | \
   /usr/sbin/makedbm - domain/jokes.bypunchline
   ```

   Don't forget that **`jokes.bypunchline`** should be in the **`alias`** file too.

After making the map on the new master, you must send a copy of it to the other slave servers. However, do not use **`yppush,`** as the other slaves will try to get new copies from the old master, rather than the new one. A typical method for circumventing this is to transfer a copy of the map from the new master back to the old master. Become NIS administrator on the old master server and type:

```
oldmaster# /usr/sbin/ypxfr -h newmaster jokes.bypunchline
```

Now it is safe to run **`yppush.`** The remaining slave servers still believe that the old master is the current master. They 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 this method fails, you can try this cumbersome but sure-fire option. Log in as NIS administrator on each NIS server and execute the **`ypxfr`** command shown above.

Summary of NIS-Related Commands

In addition to maps, NIS service also includes specialized daemons, system programs, and commands, which are summarized below.

`ypserv`	Looks up requested information in a map. **`ypserv`** is a daemon that runs on NIS servers with a complete set of maps. At least one **`ypserv`** daemon must be present on the network for NIS service to function.
`ypbind`	Initiates binding. **`ypbind`** is the NIS binder daemon. It must be present on both clients and servers. It initiates binding by finding a **`ypserv`** process that serves maps within the domain of the

	requesting client. **`ypserv`** must run on each NIS server. **`ypbind`** must run on all servers and clients.
`ypinit`	Automatically creates maps for an NIS server from files located in **`/etc.`** **`ypinit`** also constructs the initial maps that are not built from files in **`/etc,`** such as **`ypservers.`** Use **`ypinit`** to set up the master NIS server and the slave NIS servers for the first time, as well as to initialize all clients. **`ypinit`** is also used to terminate NIS on servers and clients.
`make`	Updates NIS maps by reading the **`Makefile`** in **`/var/yp.`** You can use **`make`** to update all maps based on the files in **`/etc`** or to update individual maps. The manual page **`ypmake(1M)`** describes **`make`** functionality for NIS.
`makedbm`	Takes an input file and converts it into **`dbm .dir`** and **`.pag`** files—valid **`dbm`** files that NIS can use as maps. You can also use **`makedbm -u`** to "disassemble" a map, so that you can see the key-value pairs that comprise it.
`ypxfr`	Moves an NIS map from one server to another, using NIS itself as the transport medium. You can run **`ypxfr`** interactively, or periodically from a **`crontab`** file. It is also called by **`ypserv`** to initiate a transfer.
`yppush`	Copies a new version of an NIS map from the NIS master server to its slaves. You run it on the master NIS server.
`ypset`	Tells a **`ypbind`** process to bind to a named NIS server. **`ypset`** is not for casual use.
`yppoll`	Tells which version of an NIS map is running on a server that you specify. It also lists the master server for the map.
`ypcat`	Displays the contents of an NIS map.
`ypmatch`	Prints the value for one or more specified keys in an NIS map. You cannot specify which NIS server's version of the map you are seeing.
`ypwhich`	Shows which NIS server a client is using at the moment for NIS services, or, if invoked with the **`-m`** *mapname* option, which NIS server is master of each of the maps.
`ypupdated`	Facilitates the updating of NIS information.
`yppasswd`	Utility that allows an NIS user on a client system to change his/her password.
`rpc.yppasswdd`	Daemon to allow password updating via yppasswd.

Fixing NIS Problems

This section explains how to clear problems encountered on networks running NIS. It has two parts, one covering problems seen on an NIS client and another covering problems seen on an NIS server.

Debugging an NIS Client

Before trying to debug an NIS client, review the first part of the chapter, which explains the NIS environment. Then look for the subheading in this section that best describes your problem.

Hanging Commands on the Client

The most common problem of NIS clients is for a command to hang and generate console messages such as:

```
yp: server not responding for domain <domainname>. Still
trying
```

Sometimes many commands begin to hang, even though the system as a whole seems normal and you can run new commands.

The message above indicates that **`ypbind`** on the local machine is unable to communicate with **`ypserv`** in the domain *domainname*. This happens when a machine running **`ypserv`** has crashed or is down or unavailable for any reason. It may also occur if the network or NIS server is so overloaded that **`ypserv`** cannot get a response back to the client's **`ypbind`** within the time-out period.

Under these circumstances, every client on the network will experience the same or similar problems. The condition is temporary in most cases. The messages will usually go away when the NIS server reboots and restarts **`ypserv,`** or when the load on the NIS server or network itself decreases.

However, commands may hang and require direct action to clear them. The following list describes the causes of such problems and gives suggestions for fixing them:

- The NIS client has not set, or has incorrectly set, the machine's domain name. Clients must use a domain name that the NIS servers know.
- On the client, type **`domainname`** to see which domain name is set. Compare that with the actual domain name in **`/var/yp`** on the NIS master server. If a machine's domain name is not the same as the server's, the machine's domain name entry in its installation scripts is incorrect. Log in as NIS administrator, edit the client's installation scripts, and correct the **`domainname`** entry. This assures the domain name is correct every time the machine boots. Then set **`domainname`** manually by typing the following:

  `domainname` *good_domain_name*

- If commands still hang, make sure the server is up and running. Check other machines on your local network. If several clients also have problems, suspect a server problem. Try to find a client machine behaving normally, and type the **`ypwhich`** command on it. If **`ypwhich`** does not respond, kill it and go to a terminal on the NIS server. Type the following:

  ```
  ypserver# ps -ef | grep yp
  ypserver# kill -9 pid
  ```

  where *pid* is the process-ID for **`ypbind`**. Look for **`ypserv`** and **`ypbind`** processes. If a **`ypserv`** process is running, type

  ```
  ypserver# ypwhich
  ```

  on the NIS server. If **`ypwhich`** does not respond, **`ypserv`** has probably hung, and you should restart it. Type the following while logged in as NIS administrator:

  ```
  ypserver# kill -9 [ypserv's pid # from ps]
  ypserver# /usr/lib/netsvc/yp/ypserv
  ```

  If **`ps`** shows no **`ypserv`** process running, start one up.

- If the server's **`ypbind`** daemon is not running, start it up by typing the following:

  ```
  ypserver# /usr/lib/netsvc/yp/ypbind
  ```

  Notice that if you run **`ypbind`** and you type **`ypwhich`** immediately, **`ypwhich`** will return the error message `not found` in all cases. Run **`ypwhich`** again; it should now return the name of a server.

- If commands still hang, you may try the following:

  1. Kill the existing **`ypbind:`**

     ```
     ps -ef | grep ypbind
     ```

  2. Restart **`ypbind`** with the **`ypset`** option that permits root to change the server:

     ```
     ypbind -ypsetme
     ```

  3. Reset the server to one you know is reliable:

     ```
     ypset servername
     ```

NIS Is Unavailable

When most machines on the network appear to be behaving normally, but one client cannot receive NIS service, that client may experience many different symptoms. For example, some commands appear to operate correctly while others terminate with an error message about the unavailability of NIS. Other commands limp along in a backup-strategy mode particular to the program involved. Still other commands or daemons crash with obscure messages or no message at all. Here are messages a client in this situation may receive:

```
ypcat myfile
ypcat: can't bind to NIS server for domain <domainname>.
    Reason: can't communicate with ypbind.
/usr/sbin/yppoll myfile
yppoll: Sorry, I can't communicate with ypbind. I give
up.
```

These symptoms usually indicate that the client's **`ypbind`** process is not running. Run **`ps -ef`** and check for **`ypbind.`** If it you do not find it, log in as NIS administrator and start it by typing the following:

```
/usr/lib/netsvc/yp/ypbind
```

NIS problems should disappear.

ypbind Crashes

If **`ypbind`** crashes almost immediately each time it is started, look for a problem in some other part of the system. Check for the presence of the **`rpcbind`** daemon by typing the following:

```
nfsping -o rpcbind
```

If the message returned states that **`rpcbind`** is not running, restart the **`rpcbind`** daemon or reboot. See **`nfsping(1M)`** for more information on **`nfsping`**.

If **`rpcbind`** itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software in the ways suggested in Chapter 16, "Trouble-shooting and Tuning TCP/IP".

You may be able to communicate with **`rpcbind`** on the problematic client from a machine operating normally. From the functioning machine, type:

```
rpcinfo client | grep ypbind
```

If **`rpcbind`** on the problematic machine is running normally, **`rpcinfo`** should produce an output similar to the following:

```
100007    3     tcp  0.0.0.0.12.169  ypbind  superuser
100007    3  ticlts  Q 00 00 00  ypbind  superuser
100007    3  ticots   07 00 00 00  ypbind  superuser
100007    3  ticotsord   07 00 00 00  ypbind  superuser
100007    3  starlandg   00 15sfsc.30719?00 20I 00 00 00 00 00 00 00 10 \
              00j 10 32v\376 02  ypbind  superuser
100007    3  starlan   00 15sfsc.;3719?00 20I 00 00 00 00 00 00 00 10 \
              00j 10 32v\376 01  ypbind  superuser
```

There should be one entry per transport; in the preceding example, the entry for `udp` is missing. Because **`ypbind`** was not registered for it in this case, **`ypbind`** cannot run on `udp.` As long as there are other transports to run on, **`ypbind`** should run but the omission may indicate some kind of a problem. Reboot the machine and run **`rpcinfo`** again. If the **`ypbind`** processes are there and they change each time you try to restart **`/usr/lib/netsvc/yp/ypbind,`** reboot the system, even if the **`rpcbind`** daemon is running.

ypwhich Displays Are Inconsistent

When you use **`ypwhich`** several times on the same client, the resulting display may vary because the NIS server changes. This is normal. The binding of NIS client to NIS server changes over time when the network or the NIS servers are busy. Whenever possible, the network stabilizes at a point where all clients get acceptable response time from the NIS servers. As long as your client machine gets NIS service, it does not matter where the service comes from. For example, one NIS server machine can get its own NIS services from another NIS server on the network.

Debugging an NIS Server

Before trying to debug your NIS server, read about the NIS environment at the beginning of this chapter. Then look in this subsection for the heading that most closely describes the server's problem.

Servers Have Different Versions of an NIS Map

Because NIS propagates maps among servers, occasionally you find different versions of the same map at NIS servers on the network. This version discrepancy is normal if transient, but abnormal otherwise.

Most commonly, normal map propagation is prevented if it occurs when an NIS server or router between NIS servers is down. When all NIS servers and the routers between them are running, **`ypxfr`** should succeed.

If a particular slave server has problems updating maps, log in to that server and run **`ypxfr`** interactively. If **`ypxfr`** fails, it will tell you why it failed, and you can fix the problem. If **`ypxfr`** succeeds, but you suspect it has occasionally failed, create a log file to enable logging of messages. As NIS administrator type the following:

```
ypslave# cd /var/yp
ypslave# touch ypxfr.log
```

This saves all output from **`ypxfr.`** The output resembles the output **`ypxfr`** displays when run interactively, but each line in the log file is time-stamped. You may see unusual orderings in the timestamps. This is normal - the time-stamp tells you when **`ypxfr`** started to run. If copies of **`ypxfr`** ran simultaneously but their work took different amounts of time, they may actually write their summary status line to the log files in an order different from that 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 root's **`crontab`** file and the **`ypxfr*`** shell scripts it invokes. Typos in these files cause propagation problems, as do failures to refer to a shell script within **`/var/spool/cron/crontabs/root,`** or 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 it is not, it still operates perfectly as a server, but **`yppush`** will not tell it when a new copy of a map exists.

If the NIS slave server's problem is not obvious, you can work around it while you debug it using **`rcp`** or **`tftp`** to copy a recent version of the inconsistent map from any healthy NIS server. You must not do this remote copy as root, but you can probably do it while logged in as `daemon.` For instance, here is how you might transfer the map **`busted`**:

```
ypslave# chmod go+w /var/yp/mydomain
ypslave# su daemon
rcp ypmaster:/var/yp/mydomain/busted.\* /var/yp/mydomain
exit
ypslave# chown root /var/yp/mydomain/busted.*
ypslave# chmod go-w /var/yp/mydomain
```

Here the `*` character has been escaped in the command line so that it will be expanded on `ypmaster,` instead of locally on `ypslave.` Notice that the map files should be owned by root, so you must change their ownership after the transfer.

ypserv Crashes

When the **`ypserv`** process crashes almost immediately and does not stay up even with repeated activations, the debug process is virtually identical to that previously described in “ypbind Crashes” on page 34-27. Check for the existence of the **`rpcbind`** daemon as follows:

```
ypserver# ps -ef | grep rpcbind
```

Reboot the server if you do not find the daemon. If it is there, type

`rpcinfo` *yp_server* **`| grep ypserv`**

and look for output similar to the following:

```
100004    2     tcp  0.0.0.0.12.168  ypserv  superuser
100004    2     udp  0.0.0.0.4.8  ypserv  superuser
100004    2  ticlts  B 00 00 00  ypserv  superuser
100004    2  ticots   06 00 00 00  ypserv  superuser
100004    2  ticotsord   06 00 00 00  ypserv  superuser
100004    2  starlandg   00 15sfsc.2>219?00 20I 00 00 00 00 00 00 00 10 \
             00j 10 32v\376 02  ypserv  superuser
100004    2  starlan   00 15sfsc.31319?00 20I 00 00 00 00 00 00 00 10 \
             00j 10 32v\376 01  ypserv  superuser
```

Your machine will have different port numbers. As in the case of **`ypbind,`** there should be one entry per transport. If a transport is missing, **`ypserv`** has been unable to register its services with it. Reboot the machine. If the **`ypserv`** processes are there, and they change each time you try to restart **`/usr/lib/netsvc/yp/ypserv,`** reboot the machine.

Turning off NIS

If **`ypserv`** on the master is disabled, you can no longer update any of the NIS maps. On the other hand, if there is no **`ypserv`** daemon running but clients have **`ypbind`** running, machines may hang indefinitely until they find a **`ypserv.`**

To turn off NIS services safely, make sure all the clients stop running **`ypbind`** before invoking **`ypinit -t`** to terminate the NIS daemons and make changes to NIS configuration files.

Glossary

address

A unique number that identifies a machine on a network (see *IP address*).

ARP

Address Resolution Protocol, a protocol that maps an IP address to its corresponding Ethernet address.

ARPANET

The Advanced Research Projects Agency funded network, for which TCP/IP was originally developed (see *DoD Internet*).

binding

The process by which a client locates the server that shares the information desired, and then sets up communication with that server.

bridge

A device used at the Data Link layer that selectively copies packets between networks of the same type.

caller process

A process that uses RPC to have another process execute a procedure call.

client

A machine that uses the resources of another machine; a machine can be both a *client*, utilizing resources that reside on other machines, and a *server*, making local resources available to other machines.

credentials

Information that is used to prove that something is as it claims to be—for example, an identification badge. (See also “verifiers”).

daemon

A program that runs autonomously, performing actions that facilitate more complex operations; for example, the mail service is run by several daemons, all of which work more or less without human oversight.

DARPA Internet

The Defense Advanced Research Projects Agency Internet (see *Internet*).

Data Encryption Standard (DES)

A standard cryptography algorithm used to ensure data security.

datagram

Transmission unit at the IP level.

diskless client

A machine that has no local disk and is, therefore, dependent upon the server for all its file storage. Diskless machines can act only as clients, never as servers.

DDN Internet

The Defense Data Network Internet (see *Internet*).

DoD Internet

The Department of Defense Internet, a wide area network to which the ARPANET belongs (see *Internet*).

domain

An administrative entity, typically consisting of several machines, that joins a larger network.

Domain Name Service

The name service of the Internet Protocol family.

file handle

A key obtained by a client from a server to facilitate all further requests between that client and server.

gateway

An IP router (see *router*).

group identification number (GID)

A number that refers to a specific group of users on a system. Each user can belong to one or more groups.

header

Information attached to the beginning of data. Headers usually contain information about the following data to aid in processing it.

hierarchy

A part or all of a file structure. The term “hierarchy” refers to both the directories of the file structure, and the files within the structure.

host

An individual machine on a network.

hung

When a process has been stopped abnormally, with no way to restart that process.

ICMP

Internet Control Message Protocol, which is responsible for handling errors and printing error messages.

Internet

A wide area network originally funded by the Department of Defense, which utilizes TCP/IP for data interchange. The term *Internet* is used to refer to any and all of ARPANET, DARPANET, DDN, or DoD Internets.

internetwork

A group of networks connected by routers.

IP

Internet Protocol, which allows host-to-host datagram delivery.

IP address

A unique number that identifies each host in a network.

IP network number

A unique number that identifies each IP network (See *net number*).

IP router

See *router.*

local machine

The machine you are currently logged in to, as opposed to a *remote machine*.

map

A file that contains a listing of mount points and their corresponding resources. The maps are used by the automounter program to locate where a file structure should be mounted when it is needed.

mount

The action a client performs to access files in a server's shared directories. When a client mounts a resource, it does not copy that resource, but rather accesses the resource across the network as if it were a local file system.

mount point

A location within the directory tree through which a machine accesses a mounted resource. You need to create and/or specify a mount point when you mount a remote resource. The mount point for a resource is usually an empty directory.

netname

A string of printable characters created by concatenating the name of the operating system, a user ID, and a domain name. Netnames are used in DES authentication.

Network File System (NFS)

A service that enables machines to share file resources across a network.

Network Information Service (NIS)

A distributed naming service used to identify and locate objects and resources accessible to a computing community. It provides a uniform, network-wide storage and retrieval method that is both protocol- and media-independent.

NIS Server

A machine with a disk that stores a set of NIS maps that it makes available to network hosts.

network mask

A number used by software to separate the local subnet address from the rest of a given IP address.

network protocols

Sets of rules that explain how software and hardware should interact within a network to transmit information.

NIC

Network Information Center, a service that administers IP network numbers and domain names.

NIC handle

A unique NIC database identifier assigned to a network's administrator and technical contact.

node name

The name you assign to your computer to use for communications needs. Networking software, such as Basic Networking Utilities, use this name to identify your machine.

packet switching

A concept which states that a network transmits packets over connections that last only for the duration of the transmission.

packets

A piece of data smaller than a datagram. A datagram is made up of one or more packets.

port numbers

Numbers used by UDP and TCP to identify the end points of communication.

privileged user

In a system with Enhanced Security installed and the Least Privileged Mechanism (LPM) running, a privileged user is a user in the `NET` (network administrator's) role. See **`adminuser(1M)`**. Otherwise, `root` is the privileged user.

process identification number (PID)

A number that identifies a process to the operating system. Every process has a PID by which it is referenced.

public key cryptography

A cipher system that uses a published public key and an encoded secret key.

RARP

Reverse Address Resolution Protocol, a protocol that is the reverse of ARP. RARP maps Ethernet addresses to its corresponding IP addresses.

Remote File Sharing (RFS)

A service that enables machines to share resources across a network. RFS is no longer supplied as part of the operating system.

remote machine

The machine to which you are connected across the network.

Remote Procedure Call (RPC)

Procedures that provide the means by which one process (the *caller*) can have another process (the *server*) execute a procedure call as if the caller had done so itself locally.

resource

A file system, a portion of a file system, a directory, or a single file shared by a server across a network.

router

A device that forwards (routes) packets of a protocol family, from one network to another.

run level

A user mode, also called a *run state* or *init state*.

server

A machine that shares file systems or portions of file systems; a machine can be both a *server*, making local resources available to other machines, and a *client*, utilizing resources that reside on other machines. Individual files may be shared if both the server and client are using NFS.

server process

A process that receives directives from a caller process to execute procedures locally.

share

The action a server machine performs to allow some or all of its resources to become available to other hosts.

subnet

An administrative division of a network into smaller networks.

subnet number

The part of an IP address that refers to the specific subnet desired.

TCP

Transmission Control Protocol.

TCP/IP

A term used to refer to the entire Internet family of protocols, consisting of the names of the two most important protocols.

transport provider

The software that provides a path through which network applications can communicate.

UDP

User Datagram Protocol, a protocol at the same layer as TCP, but without acknowledgment of transmission and therefore unreliable.

unsharing

Making a shared local resource unavailable to remote systems.

user/group ID number

Every user and group name has a corresponding number that is used by the UNIX operating system to handle permissions to files, directories, devices, and so on. These numbers are defined in the third field of the **`/etc/passwd`** or **`/etc/group`** files, respectively.

user/group name

The names associated with each local user and group that is allowed access to your computer. This information can be found in the first field of the **`/etc/passwd`** or **`/etc/group`** files, respectively.

user identification number (UID)

A number that identifies a user to the operating system. Every user has a unique number that identifies that user to the operating system.

verifiers

Information that is used to prove that credentials are valid.

virtual circuits

An apparent connection between processes which is facilitated by TCP. A virtual circuit allows applications to talk to each other as if they had a physical circuit.

zones

Administrative boundaries within a domain, often made up of one or more sub-domains (see *domain*).

Index

Spine for 2” Binder

Product Name: 0.5" from top of spine, Helvetica, 36 pt, Bold

Volume Number (if any): Helvetica, 24 pt, Bold

Volume Name (if any): Helvetica, 18 pt, Bold

Manual Title(s): Helvetica, 10 pt, Bold, centered vertically within space above bar, double space between each title

Bar: 1" x 1/8" beginning 1/4" in from either side

Part Number: Helvetica, 6 pt, centered, 1/8" up

PowerMAX OS

User/Administrator

Network Administration

0890432