NFS User's Guide

IRIS-4D Series

SiliconGraphics
Computer Systems

Document number: 007-0850-030

NFS User's Guide

Document Version 3.0

Document Number 007-0850-030

9/90

Technical Publications:

Seth Katz
Amy B. W. Smith
Melissa Heinrich
Lorrie Williams

Engineering:

Andrew Cherenson
Brendan Eich
Dana Treadwell

NFS User's Guide
Document Version 3.0
Document Number 007-0850-030

Silicon Graphics, Inc.
Mountain View, California

Contents

List of Tables

1. Introduction

The Network File System is a facility for sharing files in a heterogeneous environment of machines, operating systems, and networks. Sharing is accomplished by mounting a remote filesystem, then reading or writing files in place as though they were on your own machine. The NFS is open-ended, and users are encouraged to interface it with other systems.

Operating system independence was taken as an NFS design goal, along with machine independence, crash recovery, transparent access, and high performance. The NFS was thus designed as a network service, and not as a distributed operating system. As such, it is able to support distributed applications without restricting the network to a single operating system.

The goal of NFS was to make all resources available as needed. Individual workstations have access to all information residing anywhere on the network, as well as resources such as printers and supercomputers.

Silicon Graphics, Inc. has adapted Sun Microsystems, Inc.'s 4.0 release of NFS for use on the IRIS-4D Series workstation. Performance on the IRIS-4D Series workstation has been optimized. NFS has also been integrated with the IRIS WorkSpace™ environment and System toolchest.

NFS is an option on the IRIS-4D Series workstation. You can run NFS on an IRIS-4D Series workstation only with Silicon Graphics, Inc.'s implementation of NFS.

1.1 Getting Started

This document makes some assumptions about you and your IRIS-4D Series workstation.

- Your system is up and running and you know how to boot it. If not, consult your owner's guide.
- You know a little about the IRIX™ operating system. If not, see "Using IRIX" in your owner's guide.
- You are familiar with a text editor that runs on the IRIS. If not, see "Editing Text Files with *jot*" in your owner's guide, or the *vi*(1) man page.
- You are familiar with TCP/IP commands. If not, read the *Network Communications Guide* published by Silicon Graphics, Inc.

The sections below summarize the terms, key files, concepts, and commands used by NFS.

1.2 Terminology

NFS uses a number of terms that have specific meaning in the NFS environment. These terms are listed below with their meanings.

mount	When you use the *mount* command, it announces to your machine that there is a file system that is to be attached to the file tree at a specified point in your directory. An NFS *mount* allows you to put another machine's file system on yours. Mounting a file system is analogous to grafting a branch on a tree.
export	To export a file system means to allow other machines to mount your file system on their machines. You can specify which file systems you want to export in the */etc/exports* file. A machine can export only its own file systems.

server	A *server* is any machine that provides a network service. A single machine can provide more than one service. In fact, a typical configuration is one machine acting as an NFS server and a YP server. In each of the NFS network services, servers are entirely passive. The servers wait for clients to call them; they never call the clients.
client	A *client* is any entity that accesses a network service. A client can be anything from an actual machine to a IRIX process generated by a piece of software.

A machine can be both a server and a client. The degree to which clients are bound to their server varies with each of the NFS network services. The YP client binds randomly to one of the YP servers by broadcasting a request. At any point, the YP client may decide to broadcast for a new server. An NFS client may choose to mount file systems from any number of servers at any time.

In all cases, however, the client initiates the binding. The server completes the binding, subject to access control rules specific to each service. Since most network administration problems occur at bind time, a system administrator should know how a client binds to a server and what (if any) access control policy each server uses.

1.2.1 NFS Permissions

The configuration files listed below are used by NFS to give remote machines access to local file systems.

/etc/hosts	This file contains a list of Internet address and machine name pairs such as 42.0.0.5 *elvis* for use with TCP/IP. This file must be kept current and must include all the machines with which you want to communicate.

/etc/exports NFS uses this file to grant mount permissions to other machines. Each line of the file contains a directory plus a list of options which might include a list of machines (see *exports*(4)). The example below allows only users on *abbott* and *costello* to mount any directory on the local machine's root file system and allows users on any machine to mount any directory on the local machine's */usr* file system.

```
/                    abbott  costello
/usr
```

Note: NFS has been enhanced to allow exporting of arbitrary files and directories.

1.2.2 NFS Commands

NFS uses two basic commands:

mount This command allows access to a directory exported by a remote machine through a local directory. The remote directory is mounted on a directory that serves as a mount point and accessed as a directory, like a local IRIX file system. The *mount* command checks for appropriate permissions before an exported file system is mounted. After you run the *mount* command with arguments, type it without arguments, to report the remote file systems as well as the local ones. Below is an example of the syntax for *mount* with arguments:

```
/etc/mount  host:/pathname /pathname
```

umount This command unmounts the remote directory. You can use either the remote name of the directory (*host:/pathname*) or the local name (*/pathname*) as the argument. Below is an example of the syntax for *umount*:

```
/etc/umount  /pathname
```

You must be the superuser to run these commands. See the *mount*(1M) man page for full option information.

1.3 Managing NFS with the System Manager and WorkSpace

The IRIX *System Manager Tools* and *WorkSpace* provide an alternative to traditional NFS management. IRIS-4D Series workstation users who prefer the visual interface may not have to use any other tools when managing and using NFS. If you are not yet familiar with the System Manager and WorkSpace, read these sections in your *IRIS-4D Series Owner's Guide* or *Personal IRIS Owner's Guide*:

- ''Choosing an Administrator''
- ''The System Manager Tools''
- ''Getting the Most Out of the WorkSpace''

1.3.1 Using the System Manager

The System Manager provides a visual interface and menus for the administration of NFS and other networking products. You can find information about the System Manager and NFS in the *IRIS-4D Series Owner's Guide* or *Personal IRIS Owner's Guide* in these sections:

- ''Adding Your IRIS to a Distributed Network''
- ''Turning on Network Software -- Centralized''
- ''Mounting a Remote Filesystem''

1.3.2 Using WorkSpace

Directories mounted by NFS appear identical to directories on the client in the WorkSpace Directory View window. You can use these directories and the files they contain in WorkSpace as you would use any other file or directory.

Files in NFS mounted file systems can not be displayed with the correct icon if that icon does not exist on the client or is not associated with the same file or file type. To represent the files in a filesystem shared with NFS, you must have the same icons associated with the same files or file types on each workstation. For more information about icons and file typing rules, see *Programming the IRIS WorkSpace.*

1.4 Hints about Debugging IRIX in the Network Environment

When you cannot perform a task that involves NFS network services, the problem probably lies in one of the following four areas, listed below in descending order of frequency.

1. The NFS network access control policies do not allow the operation or architectural constraints prevent the operation.
2. The client software or environment is broken.
3. The server software or environment is broken.
4. The network is broken.

For more information on these four areas, consult the appropriate section of this document.

1.5 Relevant Documentation

You may find useful these documents in planning and setting up your network.

- *Network Communications Guide*, published by Silicon Graphics, Inc.
- *Defense Data Network Protocol Handbook*, which is available from the Network Information Center, Defense Data Network, SRI International, 333 Ravenswood Ave., Menlo Park, California 94025, telephone: 800-235-3155. This three-volume set contains information on TCP/IP and UDP/IP.
- *IRIS-4D Series Owner's Guide* or *Personal IRIS Owner's Guide*, which provide information on hardware installation for TCP/IP and administration of NFS through WorkSpace. This is published by Silicon Graphics, Inc.

- *IRIS-4D Programmer's Reference Manual*, published by Silicon Graphics, Inc.

1.6 Conventions

To refer to entries in IRIX™ documentation, this manual cites the entry name, followed by the section number in parentheses. For example, *cc*(1) refers to the *cc* manual entry in Section 1 of the *IRIS-4D Programmer's Reference Manual.*

In command syntax descriptions and examples, square brackets ([ ]) surrounding an argument indicate that the argument is optional. Words in *italics* represent variable parameters, which you should replace with the string or value appropriate for the application.

In text descriptions, filenames and IRIX commands are written in *italics*. Command syntax descriptions and examples are written in `typewriter font`.

1.7 Product Support

Silicon Graphics, Inc., provides a comprehensive product support and maintenance program for IRIS products. For further information, contact your service organization.

2. Introduction to Network Concepts

This chapter introduces and describes network services. It describes the services currently available and defines some useful terms for describing the network environment.

This manual introduces and explains each of the services that are provided by the NFS service. The NFS service is introduced and explained in Chapter 3, and the Yellow Pages service is introduced and explained in Chapter 5. Each chapter contains information on troubleshooting for the service under discussion.

2.1 Networking Models

There are many ways to make computers and networks interface transparently. The two major methods are the distributed operating system approach and the network services approach.

A distributed operating system allows the network software designer to make assumptions about the other machines on the network. Usually two of these assumptions are that the remote and local hardware are identical, and that the remote and local software are identical. These assumptions allow a quick and simple implementation of a network system in an environment limited to specific hardware and software.

This type of distributed operating system is, by design, closed. A closed environment is one in which it is very difficult to integrate new hardware or software, unless it comes from the vendor of that network system. A closed network system forces a customer to return to one vendor for solutions to all computing needs.

On the other hand, the network services approach is not closed. Network services are made up of remote programs composed of remote procedures called from the network. Optimally, a remote procedure computes results based entirely on its own parameters. This procedure and the network service are not tied to any particular operating system or hardware. The design of the network service makes it possible for a variety of machines and software to talk to the network services. This enables the IRIS workstation to talk to various types of computers.

The network services approach is more complex in design and implementation than a closed distributed operating system. Since remote procedures are independent of operating systems and hardware, multiple remote procedures must sometimes be called in the NFS environment, where a single transaction might suffice in a closed system.

2.2 Introduction to Major Services

This overview presents each of the major NFS services followed by a short description of their functionality. The NFS services are discussed in greater detail in this manual.

The *Remote Procedure Call (RPC)* facility is a library of procedures that provide a means whereby one process (the caller process) can have another process (the server process) execute a procedure call, as if the caller process had executed the procedure call in its own address space (as in the local model of a procedure call). Because the caller and the server are now two separate processes, they no longer have to live on the same physical machine. RPC is documented in the *Network Communications Guide.*

The *External Data Representation (XDR)* is a specification for the portable data transmission standard. Together with RPC, it provides a kind of standard I/O library for interprocess communication. Thus programmers now have a standardized access to sockets without having to be concerned about the low-level details of socket-based IPC.

NFS is an operating system-independent service which allows users to mount directories, even root directories, across the network, and then to treat those directories as if they were local.

The *portmapper* is a utility service that all other services use. It's a kind of registrar that keeps track of the correspondence between ports (logical communications channels) and services on a machine, and provides a standard way for a client to look up the port number of any remote program supported by the server.

Yellow Pages (YP) is a network service designed to ease the job of administering the large networks that NFS encourages. The YP is a replicated, read-only, database service. Network file system clients use it to access network-wide data in a manner that is entirely independent of the relative locations of the client and the server. The YP database typically provides password, group, network, and host information.

Network Lock Manager supports advisory file and record locking for both local and NFS mounted files. User programs simply issue *lockf()* and *fcntl()* system calls to set and test file locks — these calls are then processed by *Lock Manager* daemons, which maintain order at the network level.

The *automount* utility service is used to dynamically mount filesystems. Filesystems can be mounted automatically when they are used, and unmounted silently following a predetermined idle period. The *automount* utility can be used for a variety of purposes, including automatically mounting a user's home directory from any workstation connected by NFS.

There are other network services, such as *sprayd*, *rstatd*, and *rwalld*. This section, however, is intended as an introduction, and it covers only the fundamental services noted above.

2.3 NFS Abstract

This section offers a discussion of some of the principles behind Silicon Graphics, Inc.'s implementation of NFS.

2.3.1 Transparent Information Access

Users are able to get directly to the files they want without knowing the network address of the data. To the user, all NFS-mounted filesystems look just like private disks. There's no apparent difference between reading or writing a file on a local disk, and reading or writing a file on a disk in the next building. Information on the network is truly distributed.

2.3.2 Different Machines and Operating Systems

No single vendor can supply tools for all the work that needs to get done, so appropriate services must be integrated on a network. NFS provides a flexible, operating system-independent platform for such integration.

2.3.3 Easily Extensible

A distributed system must have an architecture that allows integration of new software technologies without disturbing the extant software environment. Since the NFS network-services approach does not depend on pushing the operating system onto the network, but instead offers an extensible set of protocols for data exchange, it supports the flexible integration of new software.

2.3.4 Ease of Network Administration

The administration of large networks can be complicated and time-consuming, yet they should (ideally) be at least as easy to administer as a set of local filesystems on a timesharing system. The UNIX system has a convenient set of maintenance commands developed over the years, and the Yellow Pages (YP), an NFS-based network database service, has allowed them to be adapted and extended for the purpose of administering a network of machines. The YP also allows certain aspects of network administration to be centralized onto a small number of file servers, e.g. only server disks must be backed up in networks of diskless clients.

2.3.5 Reliability

NFS's reliability derives from the robustness of EFS (or *Extent File System*), from the stateless NFS protocol, and from the daemon-based methodology by which network services like file and record locking are provided. See the section *The Lock Manager* for more details on locking. In addition, the file server protocol is designed so that client workstations can continue to operate even when the server crashes and reboots.

The major advantage of a stateless server is robustness in the face of client, server, or network failures. Should a client fail, it is not necessary for a server (or human administrator) to take any action to continue normal operation. Should a server or the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network gets fixed. This robustness is especially important in a complex network of heterogeneous systems, many of which are not under the control of a disciplined operations staff, and which may be running untested systems that are often rebooted without warning.

2.3.6 High Performance

The flexibility of the NFS allows configuration for a variety of cost and performance trade-offs. For example, configuring servers with large, high-performance disks, and clients with no disks, may yield better performance at lower cost than having many machines with small, inexpensive disks. Furthermore, it is possible to distribute the filesystem data across many servers and get the added benefit of multiprocessing without losing transparency. In the case of read-only files, copies can be kept on several servers to avoid bottlenecks.

Several performance enhancements have been added to SGI's NFS, such as fast paths for key operations, asynchronous service of multiple requests, disk-block caching, and asynchronous read-ahead and write-behind. The fact that caching and read-ahead occur on both client and server effectively increases the cache size and read-ahead distance. Caching and read-ahead do not add state to the server; nothing (except performance) is lost if cached information is thrown away. In the case of write-behind, both the client and server attempt to flush critical information to disk whenever necessary, to reduce the impact of an unanticipated failure; clients do not free write-behind blocks until the server confirms that the data is written.

2.4 UNIX and NFS

Unlike many recently marketed distributed operating systems, UNIX was originally designed without the knowledge that networks existed. This networking ignorance presents three impediments to linking UNIX with currently available high-performance networks.

1. UNIX was never designed to yield to a higher authority (like a network authentication server) for critical information or services. As a result, some UNIX semantics are hard to maintain across a network. For example, trusting remote users to log in as *root* is not always a good idea.

2. Some UNIX execution semantics are difficult. For example, UNIX allows a user to remove an open file, yet the file does not disappear until closed by everyone. In a network environment, a client UNIX machine may not own an open file. Therefore, a server may remove a client's open file.

3. When a UNIX machine crashes, it takes all its applications down with it. When a network node crashes (whether client or server), it should not drag down all of its bound neighbors. The treatment of node failure on a network raises difficulties in any system and is especially difficult in the UNIX environment.

   NFS has implemented a system of *stateless* protocols to circumvent the problem of a crashing server dragging down its bound clients. Stateless means that a client is independently responsible for completing work, and that a server need not remember anything from one call to the next. In other words, the server keeps no state. With no state left on the server, there is no state to recover when the server crashes and comes back up. So, from the client's point of view, a crashed server appears no different from a very slow server.

In implementing UNIX over the network, NFS attempted to remain compatible with UNIX whenever possible. However, two kinds of incompatibilities have been introduced. First, there are issues that would make a networked UNIX evolve into a distributed operating system, rather than a collection of network services. Second, there are issues that would make crash recovery extremely difficult from both the implementation and administration point of view.

3. NFS Service Overview

This chapter discusses Silicon Graphics, Inc.'s implementation of Sun Microsystem, Inc.'s Network File System (NFS). This chapter begins with an explanation of some NFS terms and concepts. It then describes how to create an NFS server that exports file systems, how to mount and utilize remote file systems, how to debug NFS when problems occur, and how to work with incompatibilities between NFS files and normal UNIX files. The NFS services are also discussed in the following chapter. Some references to the Yellow Pages (YP) are included in this chapter. Section 4.2 contains a description of YP commands.

3.1 What is NFS?

NFS enables users to share file systems over the network. A client may mount or unmount file systems from an NFS server machine. The client always initiates the binding to a server's file system by using the *mount*(1M) command. Typically, a client mounts one or more remote file systems at startup by placing lines like those shown in the example below in the file */etc/fstab*, which *mount* reads when the system comes up.

```
titan:/usr2   /usr2   nfs   rw,hard,bg 0 0
venus:/usr/man   /usr/man   nfs   rw,hard,bg 0 0
```

See *fstab*(4) for a full description of the format.

Since clients initiate all remote mounts, NFS servers control who may mount a file system. NFS servers do this by limiting named file systems to desired clients with an entry in the */etc/exports* file. In the example below, the file system */usr* is exported to the world, while the file system */usr2* is limited to specific machines.

```
/usr    # export to the world
/usr2   nixon ford reagan # export to only these machines
```

See *exports*(4) and *exportfs*(1M) for a full description of the format.

3.2 How NFS Works

Two remote programs implement NFS service: *mountd*(1M) and *nfsd*(1M). The *mountd* service is run by an NFS client. The *mountd* service checks permission to verify access by the client. If the client does have permission to mount the file system requested, a pointer to that file system is returned. When access to the mount point and the directories below it are requested, this pointer leads you to the server's *nfs* daemon, through a Remote Procedure Call (RPC). The *nfsd* service starts the *nfs*(4) daemons on the server that handle client file system requests. For more information on Remote Procedure Calls, see the *Network Communications Guide*.

3.3 How to Become an NFS Server

An NFS server is a machine that exports one or more file systems. To enable any machine to export a file system, become the superuser (*root*) and follow these steps:

1. Add the pathname from the mount point of the file system you want to export in the file */etc/exports*. See *exports*(4) for file format details. For example, to export */usr*, the export file would look like this:

   ```
   /usr
   ```

   An NFS server can export only its own file systems.

2. Make sure *mountd* is available for an RPC call by checking */usr/etc/inetd.conf* on the NFS server for these lines:

   ```
   mountd/1 dgram rcp/udp wait root /usr/etc/rpc.mountd mountd
   ```

   If the lines above are not present, add them. For details, see *inetd*(1M).

3. Make sure that NFS is enabled at system initialization by setting the *nfs* configuration flag on:

   ```
   /etc/chkconfig  nfs on
   ```

   The initialization script, */etc/init.d/network*, starts NFS automatically, every time you boot the server, if the flag is on.

   To enable NFS servers without rebooting, type:

   ```
   /usr/etc/nfsd 4
   /usr/etc/exportfs -va
   ```

After these steps, the NFS server should be able to export the file system named in */etc/exports*. The next section describes how to mount a remote file system.

3.4 How to Mount a Remote File System

This section contains information on mounting a remote file system. You can mount any exported file system onto your machine as long as you can reach the server over the network and your machine is included in its */etc/exports* list for that file system. The terms *hard* and *soft mount* are defined as follows:

hard mount	A *hard mount* attaches a remote file system to a local machine in a way that causes the client to continue to call the server until the server responds. A hard mount will always wait until it gets a response. If the server is down or slow, the hard mount causes the client to wait indefinitely for any operation on that file system. A hard mount is the default kind of mount.

soft mount	A *soft mount* also attaches a remote file system to a local machine. However, the client will not continue to call indefinitely. Rather, it will call a number of times and then give up. If a client is not able to execute any command on a soft-mounted file system (because the server fails to respond), it will print an error on the console. A soft mount is an option that you must specify when using the *mount* command.

On the machine on which you want to mount the file system, become the superuser and type:

```
mount server_name:/pathname /directory
```

For example, to soft mount the */usr* file system from the remote machine *elvis* onto the local empty directory */usr/elvis*, type:

```
mount -o soft,bg elvis:/usr /usr/elvis
```

You can mount file systems with the *soft* option, so that if *elvis* goes down, the local machine does not keep trying to mount */usr/elvis* indefinitely. This command fails if the server goes down, rather than waiting for the server to come back up.

You may wish to keep trying to mount */usr/elvis*. For example, it may be important to mount the filesystem as soon as elvis recovers from a crash, or you may want the filesystem mounted automatically if it is available. If this is the case, you can use the *hard* option, along with the background option *bg*, causing minimal impact on system performance:

```
mount -o hard,bg elvis:/usr /usr/elvis
```

Use the *hard* option when performing operations that write across the NFS connection, or for any operation that must complete.

Note that it is advisable to use the background option with any mounts to avoid hanging while waiting for the server to respond or recover.

To make sure you have mounted a file system where you expected, use either *df*(1) or *mount*(1M), without an argument. Each of these commands displays the currently mounted file systems.

Typically, you mount frequently used file systems at startup by placing entries for them in the file */etc/fstab*. See *fstab*(4).

3.4.1 *automount*

Rather than mount file systems and leave them mounted at all times, the *automount* command mounts file systems only when they are in use, and unmounts them when they are not in use. Because file systems are unmounted when not in use, *automount* can be used to mount any or all filesystems available on your network, using only one mount point.

The *automount* command starts a daemon that mounts and unmounts file systems when they are used. The remote file systems are accessible to the user in the same way they would be if they were always mounted. To access a file or directory in a remote file system use the same commands that access files on local file systems. The file system is mounted when it is accessed, in a manner transparent to the user. For more information on *automount*, see Chapter 4, *Using the NFS Automounter* and the *automount*(1M) manual page.

3.5 Port Registration

Every portmapper on every host is associated with port number 111. The portmapper is the only network service that must have such a well-known (dedicated) port. Other network services can be assigned port numbers statically or dynamically so long as they register their ports with their host's portmapper. For example, a server program based on the RPC library typically gets a port number at run time by calling an RPC library procedure. Note that a given network service can be associated with port number 256 on one server and with port number 885 on another; on a given host, a service can be associated with a different port every time its server program is started. Delegating port-to-remote program mapping to portmappers also automates port number administration. Statically mapping ports and remote programs in a file duplicated on each client would require updating all mapping files whenever a new remote program was introduced to a network. (The alternative of placing the port-to-program mappings in a shared NFS file would be too centralized, and if the fileserver went down the whole network would go down with it).

The port-to-program mappings which are maintained by the portmapper server are called a *portmap*. The portmapper is started automatically whenever a machine is booted. Both server programs and client programs call portmapper procedures. As part of its initialization, a server program calls its host's portmapper to create a portmap entry. Whereas server programs call portmappers to update portmap entries, clients call portmappers to query portmap entries. To find a remote program's port, a client sends an RPC call message to a server's portmapper; if the remote program is supported on the server, the portmapper returns the relevant port number in an RPC reply message. The client program can then send RPC call messages to the remote program's port. A client program can minimize its portmapper calls by caching the port numbers of recently called remote programs.

3.6 The Network Lock Manager

SGI's NFS includes a NFS-compatible Network Lock Manager (see the *lockd(8C)* man page for more details) that supports the *lockf()/fcntl()*, System V style of advisory file and record locking over the network. System V locks are generally considered superior to 4.3BSD locks, implemented with the *flock()* system call, for they provide record level, and not merely file level, locking. Record level locking is essential for database systems. *flock()* is supported for use on individual machines, but *flock()* is not intended to be used across the network. *flock()* locks exclude only other processes on the same machine. There is no interaction between *flock()* and *lockf()*.

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 locking service by way of the standard *lockf()* system-call interface, and rarely requires any detailed knowledge of how it works. The kernel maps user calls to *flock()* and *fcntl()* into RPC-based messages to the local lock manager (or, if the files in question are on EFS-mounted filesystems, into calls to EFS). 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. Furthermore, locking is *inherently stateful*. If a server crashes, clients with locked files must be able to recover their locks.

If a client crashes, its servers must have the sense to hold the client's locks while it recovers. And, 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 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 per se. They are separate services that use the status monitor (see the section *The Network 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 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's 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 and the status monitor 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-execution services *rexd* and 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 (like *lockf ()*) 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 section *The Locking Protocol* for details of 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's possible, for example, for users to alter the lock manager to provide locking in a different style.

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 is 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 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 manager retransmits all lock requests previously granted by the recovered server. This retransmitted information is used by the server to reconstruct its locking state. See below for more details.

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 be notified of the crash and send it the information that 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.

3.6.1 The Locking Protocol

The lock style implemented by the network lock manager is that specified in the *AT&T System V Interface Definition*, (see the *lockf(2)* and *fcntl(2)* man pages for details). There is no interaction between the lock manager's locks and *flock() -style* locks, which remain supported, but which should be used for non-network applications only.

Locks are presently advisory only, on the (well supported) assumption that cooperating processes can do whatever they wish without mandatory locks. Besides, mandatory locks pose serious security problems — if */etc/passwd* is locked against reading, the whole system freezes. (See the *fcntl(2)* man page for more information about advisory locks.)

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 managers adheres to the *lockf()/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 of these 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 it will request them again when it wants them. When a server crashes, however, matters are different: the clients will wait for it to come back up, and when it does, its lock manager will give the client lock managers a grace period to submit lock reclaim requests, and during this period 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.

3.6.2 The Network Status Monitor

The Network Status Monitor (see the *statd(8C)* man page for more details) was introduced with the lock manager, which relies heavily on it to maintain the inherently stateful locking service within the stateless NFS environment. However, the status monitor 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 makes it more or less routine.

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 implements a simple protocol which allows applications to easily monitor 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 it notifies the interested applications to that effect, and they then take whatever actions are necessary to reestablish a consistent state.

There are several major advantages to this approach:

- 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.

3.7 Setting the Time in User Programs

Since NFS architecture differs in some minor ways from earlier versions of UNIX, be aware of those places where your own programs could run up against these incompatibilities.

Because each workstation keeps its own time, the clocks will be out of sync between the NFS server and client. You can use *timed*(1M) and *timeslave*(1M) to synchronize the clocks of the workstations on your network.

3.8 Debugging NFS

If you experience difficulties with NFS, review documentation regarding NFS functionality before trying to debug NFS. Relevant material can be found in Section 3.2, "How NFS Works" in this manual and the manual pages for *mount*(1M), *nfsd*(1M), *showmount*(1M), *exportfs*(1M), *rpcinfo*(1M), *mountd*(1M), *inetd*(1M), *fstab*(4), *mtab*(4), and *exports*(4). You do not have to understand them fully, but be familiar with the names and functions of the various daemons and database files.

When analyzing an NFS problem, keep in mind that, like all network services, there are three main points of failure: the server, the client, and the network itself. The debugging strategy outlined below tries to isolate each individual component to find the one that is not working.

For example, here is a sample mount request made from an NFS client machine:

```
mount  krypton:/usr/src  /krypton.src
```

The example asks the server machine *krypton* to return a file handle (a unique identifier) for the file system */usr*. This file handle is then passed to the kernel in the *mount*(2) system call. The kernel looks up the directory */krypton.src* and, if everything is working, it ties the file handle to the file system in a mount record. From now on, all file system requests to that file system and below will go through the file handle to the server *krypton*.

The example above shows how a remote mount should work. Section 3.9 contains some general information and lists the possible errors and their causes.

When there are network or server problems, programs that access hard-mounted remote files fail differently from those that access soft-mounted remote files. Hard-mounted remote file systems cause programs to continue to try until the server responds again. Soft-mounted remote file systems return an error message after trying for a specified number of intervals. See *mount*(1) for more information.

Programs that access hard-mounted file systems will not respond until the server responds. In this case, NFS displays the message:

```
server  not  responding
```

On a soft-mounted file system, programs that access a file whose server is inactive get the message:

```
Connection  timed  out
```

Unfortunately, many IRIX programs do not check return conditions on file system operations, so this error message may not be displayed when accessing soft-mounted files. Nevertheless, an NFS error message is displayed on the console.

If a client is having NFS trouble, check first to make sure the server is up and running. From a client, type:

```
/usr/etc/rpcinfo  -p  server_name
```

This checks whether the server is running. If the server is running, this command displays a list of program, version, protocol, and port numbers similar to the following:

```
program          vers            proto           port
100003           2               udp             2049  nfs
100012           1               udp             1087  sprayd
100011           1               udp             1089  rquotad
100005           1               udp             1091  mountd
100008           1               udp             1093  walld
100002           1               udp             1095  rusersd
100003           2               udp             1095  rstatd
```

If the server you want to use is running, also use *rpcinfo* to check if the *mountd* server is running by using its program number as an argument followed by the version number. Type:

`/usr/etc/rpcinfo -u` *server_name* 100005 1

The system responds:

```
program  100005  version  1  ready  and  waiting
```

If these fail, log in to the server's console to see if it is working.

If the server is operative but your machine cannot reach it, check the ethernet connections between your machine and the server. You can also check other systems on your network to see if they can reach the server.

If the server and the network are working, type *ps -de* to check your client daemons. *portmap* and several *biod* daemons should be running. For example, typing *ps -de* produces output similar to the following:

```
PID   TTY  TIME   COMMAND
 97   ?    0:00   routed
102   ?    0:00   portmap
103   ?    0:00   inetd
113   ?    0:00   nfsd
116   ?    0:00   nfsd
117   ?    0:00   nfsd
118   ?    0:00   nfsd
119   ?    0:00   biod
121   ?    0:00   biod
123   ?    0:00   biod
125   ?    0:00   biod
```

3.9 Types of Failures

The four sections below describe the most common types of failure. The first section tells what to do if your remote mount fails; the next three sections discuss servers that do not respond once you have mounted file systems.

3.9.1 Remote Mount Failed

This section describes problems related to mounting. If for any reason *mount* fails, check the sections below for specific details about what to do. The sections are arranged according to where the problems occur in the mounting sequence. Each section is labeled with the error message you are likely to see.

The *mount* utility gets its parameters from either the command line or the file */etc/fstab*. See *mount*(1M). The example below assumes command line arguments, but the same debugging techniques described below also work if the *mount* command uses */etc/fstab*.

This section explains the interaction of the various players in the *mount* request. If you understand this interaction, the problem descriptions below will make more sense. Here is the example *mount* request previously given in Section 3.4:

```
mount   krypton:/usr/src   /krypton.src
```

Below are the steps *mount* goes through to mount a remote file system.

1. *mount* opens */etc/mtab* and checks that this mount has not already been done.

2. *mount* parses the first argument into host *krypton* and remote directory */usr/src*.

3. *mount* uses */etc/hosts* to translate the host name into its Internet Protocol (IP) address for *krypton*. If you are using the YP service, *mount* uses library routines that call *ypbind* to determine which server machine is a YP server. It then calls the *ypserv* daemon on that machine to get the IP address of krypton.

4. *mount* calls *krypton*'s *portmap* daemon to get the port number of *mountd*. See *portmap*(1M).

5. *mount* calls *krypton*'s *mountd* and passes it */usr/src*.

6. *krypton*'s *mountd* reads */etc/exports* and looks for the exported file system that contains */usr.*

7. *krypton*'s *mountd* calls YP to expand the host names and network groups in the export list for */usr.*

8. *krypton*'s *mountd* performs a system call on */usr/src* to get the file handle.

9. *krypton*'s *mountd* returns the file handle.

10. *mount* does a *mount*(2) system call with the file handle and *krypton.src*.

11. *mount* checks to see if the caller is the superuser and if */krypton.src* is a directory.

12. *mount* does a *statfs*(2) call to *krypton*'s NFS server (*nfsd*).

13. *mount* opens */etc/mtab* and adds an entry to the end.

Any of these steps can fail, some of them in more than one way. The section below gives detailed descriptions of the failures associated with specific error messages.

/etc/mtab: No such file or directory

The mounted file system table is kept in the file */etc/mtab*(4). This file must exist before *mount* can succeed.

mount: ... already mounted

The file system that you are trying to mount is already mounted or there is an incorrect entry for it in */etc/mtab.*

mount: ... Block device required

You probably left off the *krypton:* part of

```
#    mount krypton:/usr/src    /krypton.src
```

The *mount* command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is **nfs** in */etc/fstab*. See *fstab*(1).

mount: ... not found in /etc/fstab

If you use *mount* with only a directory or file system name, but not both, it looks in */etc/fstab* for an entry with file system or directory field matching the argument. For example,

```
mount /krypton.src
```

searches */etc/fstab* for a line that has a directory name field of */krypton.src*. If it finds an entry, such as:

```
krypton:/usr/src    /krypton.src    nfs    rw,hard 0 0
```

it mounts as if you had typed:

```
mount   krypton:/usr/src    /krypton.src
```

If you see this message, it means the argument you gave *mount* is not in any of the entries in */etc/fstab*.

/etc/fstab: No such file or directory

mount tried to look up the name in */etc/fstab* but there was no */etc/fstab*.

... not in hosts database

The host name you gave is not in the */etc/hosts* database. First, check the spelling and the placement of the colon in your *mount* call. Try to *rlogin* or *rcp* to some other machine.

mount: directory path must begin with a slash (/).

The second argument to *mount* is the path of the directory to be covered. This must be an absolute path starting at /.

mount: ... server not responding: RCP: Port mapper failure

Either the server from which you are trying to mount is inactive, or its *portmap* daemon is inactive or hung. Try logging in to that machine. If you can log in, type:

```
/usr/etc/rpcinfo  -p  hostname
```

This should produce a list of registered program numbers. If it does not, start the *portmap* daemon again. Note that starting the *portmap* daemon again requires that you kill and restart *inetd*, *ypbind*, and *ypserv*. *ypbind* is only active if you are using the YP service. See *network*(1M) for information about how to stop and restart daemons.

There are two methods for dealing with a server that is inactive or whose *portmap* daemon is not responding. You could reboot the server or you could do the following:

1. On the server, become the superuser and kill the daemons. Type:

   ```
   su
   killall  portmap  inetd  ypbind
   ```

2. Start new daemons. Type:

   ```
   /usr/etc/portmap
   /usr/etc/inetd
   /usr/etc/ypbind
   ```

   If you cannot *rlogin* to the server, but the server is operational, check your ethernet connection by trying *rlogin* to some other machine. Also check the server's ethernet connection.

mount: ... server not responding: RCP: Program not registered

This means *mount* reached the *portmap* daemon but the NFS mount daemon (*rpc.mountd*) was not registered.

Go to the server and be sure that */usr/etc/rpc.mountd* exists and that there is an entry in */usr/etc/inetd.conf* exactly like the following:

```
mountd/1 dgram rcp/udp wait root /usr/etc/rpc.mountd mountd
```

Type ps -de to be sure that the internet daemon (*inetd*) is running. If you had to change */usr/etc/inetd.conf*, type:

```
killall 1 inetd
```

This command informs *inetd* that you have changed */usr/etc/inetd.conf*.

mount: ... No such file or directory

Either the remote directory or the local directory does not exist. Check your spelling. Use the *ls* command for the local and remote directories.

mount: not in export list for ...

Your machine name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems by running:

```
showmount -e hostname
```

If the file system you want is not in the list, or your machine name or network group name is not in the user list for the file system, log in to the server and check the */etc/exports* file for the correct file system entry. A file system name that appears in the */etc/exports* file but not in the output from *showmount*, indicates that you need to run *exportfs*(1M).

mount: ... Permission denied

This message is a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above), the server could not figure out who you are, or the server does not believe you are who you say you are. Check the server's */etc/exports*. In the last case, just change your hostname in */etc/sys_id*, reboot, and retry the *mount*.

mount: ... Not a directory

Either the remote path or the local path is not a directory. Check your spelling and use the *ls* command for both the local and remote directories.

mount: ... You must be root to use mount

You must do the mount as *root* on your machine because it affects the file system for the whole machine, not just your directories.

3.9.2 Programs Do Not Respond

If programs stop responding while doing file related work, your NFS server may be inactive. You may see the message:

```
NFS server host_name not responding, still trying
```

The message includes the host name of the NFS server that is down. This is probably a problem either with one of your NFS servers or with the ethernet hardware. Attempt to *rlogin* to the server to determine whether the server is down. If you can successfully *rlogin* to it, its server function is probably disabled. See Section 3.3. If *rlogin* fails, you may need to reboot the server.

Programs can also hang if a YP server becomes inactive. See Chapter 4 for YP information and an explanation of YP commands.

If your machine hangs completely, check the servers from which you have mounted. If one or more of them is down, it is not cause for concern. When the server comes back up, your programs will continue automatically, as if the server had not become inactive. No files will be destroyed. This procedure assumes a hard mount.

If a soft-mounted server is inactive, other work should not be affected. Programs that timeout trying to access soft-mounted remote files will fail, but you should still be able to use your other file systems.

If all of the servers are running, ask someone else who is using the same NFS server or servers if they are having trouble. If more than one machine is having difficulty getting service, then it is probably a problem with the server's NFS daemon *nfsd*(1M). Log in to the server and type **ps -de** to see if *nfsd* is running and accumulating CPU time. If not, you may be able to kill, and then restart *nfsd*. If this does not work, reboot the server.

If other people seem to be able to use the server, check your ethernet connection and the connection of the server.

3.9.3 Hangs Part Way through Boot

If your workstation comes part way up after a boot, but hangs where it would normally be doing remote mounts, one or more servers are probably down or your network connection may be bad. See Sections 3.9.1 and 3.9.2.

This problem can be avoided entirely by using the background option to *mount*, *bg*. For more information on the background option, see Section 3.4.

3.9.4 Everything Works Slowly

If access to remote files seems unusually slow, go to the server and type:

```
ps  -de
```

Check whether the server is being slowed by a runaway daemon, bad *tty* line, etc. If the server seems to be working and other people are getting good response, make sure your block I/O daemons are running; type **ps -de** on your client workstation and look for *biod*. To determine whether the processes are hung, type **ps -de** as shown below, then copy a large remote file and type **ps -de** again. If the *biod*s do not accumulate CPU time, they are probably hung. To find out if they are hung, type:

```
ps -de | grep biod
```

Kill the processes by using the *killall* command. Type:

```
killall  biod
```

Restart the processes by typing:

```
/usr/etc/biod  4
```

If *biod* is working, check your ethernet connection. The command *netstat* –i tells you if packets are being dropped. A packet is a unit of transmission sent across the ethernet. Also, you can use */usr/etc/nfsstat* –c and */usr/etc/nfsstat* –s to tell if the client or server is retransmitting a lot. A retransmission rate of 5% is considered high. Excessive retransmission usually indicates a bad ethernet board, a bad ethernet tap, a mismatch between board and tap, or a mismatch between your ethernet board and the server's board.

3.9.5 Cannot Access Remote Devices

In NFS, you cannot access a remote mounted character or block device (i.e., a remote tape drive or similar peripheral).

4. Using the NFS Automounter

The NFS *automount* utility provides a new way to mount file hierarchies. File hierarchies are mounted when used and unmounted when they are no longer in use. The high level of convenience provided by *automount* allows NFS users to execute relatively complex implementations, such as mounting one's home directory from any workstation on the network, by simply using the desired files or directories.

4.1 The Automounter

The *automount* program enables users to mount and unmount remote directories on an as-needed basis. Whenever a user on a client machine running the automounter invokes a command that needs to access a remote file or directory, such as opening a file with an editor, the hierarchy to which that file or directory belongs is mounted and remains mounted for as long as it is needed. When a certain amount of time has elapsed during which the hierarchy is not accessed, it is automatically unmounted. No mounting is done at boot-time, and the user no longer has to know the superuser password to mount a directory or even use the *mount* and *umount* commands. It is all done automatically and transparently.

Mounting some file hierarchies with *automount* does not exclude the possibility of mounting others with *mount*. A diskless machine must mount / and */usr* through the *mount* command and the */etc/fstab* file. In all cases, the automounter should not be used to mount */usr/share*.

This chapter explains how the automounter works, how to write the files (maps) the automounter uses, how to invoke it, and which error messages are related to it.

4.1.1 How the Automounter Works

Unlike *mount*, *automount* does not consult the file */etc/fstab* for a list of hierarchies to mount. Rather, it consults a series of maps, which can be either direct or indirect. The names of the maps can be passed to *automount* from the command line, or from another (master) map.

The following is a simplified bird's eye view of how the automounter works:

When *automount* is started, either from the command line or from *rc.local*, it forks a daemon to serve the mount points specified 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 file hierarchy. At that time the daemon does the following:

1. Intercepts the request
2. Mounts the remote file hierarchy
3. Creates a symbolic link between the requested mount point and the actual mount point under */tmp_mnt*
4. Passes the symbolic link to the kernel, and steps aside
5. Unmounts the file hierarchy when a predetermined amount of time has passed in which the link has not been touched (generally five minutes)

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. Note that, as with any other kind of mount, a mount affected through the automounter on a non-empty mount point will hide the original contents of the mount point for as long as the mount is in effect.

The */tmp_mnt* directory is created automatically by the automounter. Its default name can be changed, as explained later in this chapter.

4.2 Preparing the Maps

A server never knows, nor cares, whether the files it exports are accessed through *mount* or *automount*. Therefore, you need not do anything different on the server for *automount* than for *mount*.

A client, however, needs special files for the automounter. As mentioned previously, *automount* does not consult */etc/fstab*; rather, it consults the map file(s) specified on the command line. If no maps are specified, it looks for an YP map called *auto.master*. If no YP *auto.master* exists, *automount* exits silently. Later on, in this chapter, we explain how to invoke *automount* so that it consults local maps.

By convention, all automounter maps are located in the directory */etc* and their names all begin with the prefix *auto*.

There are three kinds of automount maps: master, indirect, and direct.

The Master Map

The *master* map lists (as if from the command line) all other maps, applicable options, and mount points.

Each line in a master map, by convention called */etc/auto.master*, has the syntax:

```
mount-point map-name [mount-options]
```

where: *mount-point* is the full pathname of a directory. If the directory does not exist, the automounter will create it if possible. If the directory exists, and is not empty, mounting on it will hide its contents. The automounter will issue a warning message in this case. *map-name* is the map the automounter should use to find the mount points and locations. *mount-options* is an optional, comma separated, list of options that regulate the mounting of the entries mentioned in *map-name*, unless the entries in *map-name* list other options.

A line whose first character is # is treated as a comment, and everything that follows until the end of line is ignored. A backslash (\) at the end of line permits splitting long lines into shorter ones. The notation /- as a mount point indicates that the map in question is a direct map, and no particular mount point is associated with the map as a whole.

Direct and Indirect Maps

Lines in direct and indirect maps have the syntax:

```
key [mount-options] location
```

where *key* is the pathname of the mount point. The *mount-options* are the options you want to apply to this particular mount. *location* is the location of the resource, specified as *server:pathname.*

As in the master map, a line whose first character is # is treated as a comment and everything that follows until the end of line is ignored. A backslash at the end of line permits splitting long lines into shorter ones.

The only formal difference between a direct and an indirect map is that the key in a direct map is a full pathname, whereas in an indirect pathname it is a simple name (no slashes). For instance, the following would be an entry in a direct map:

```
/usr/man -ro,intr goofy:/usr/man
```

and the following would be an entry in an indirect map:

```
parsley -ro,intr veggies:/usr/greens
```

As you can see, the *key* in the indirect map is begging for more information: where is the mount point *parsley* really located? That is why you must either provide that information at the command line or through another map. For instance, if the above line is part of a map called */etc/auto.veggies*, you would have to invoke it either as:

```
automount /veggies /etc/auto.veggies
```

or specify, in the master map:

```
/veggies /etc/auto.veggies -ro,soft
```

In either case, you are associating a mount directory (*/veggies*) with the entries (*parsley* in this case) mentioned in the indirect map */etc/auto.veggies*. The end result is that the hierarchy */usr/greens* from the machine *veggies* will be mounted on */veggies/parsley* when needed.

4.2.1 Writing a Master Map

As stated above, the syntax for each line in the master map is

```
mount-point map-name [mount-options]
```

A typical *auto.master* file contains

```
#Mount-point   Map                 Mount-options
/-             /etc/auto.direct    -ro,intr
/home          /etc/auto.home      -rw,intr,secure
/net           -hosts
```

The automounter recognizes some special mount points and maps, which are explained below.

Mount point /-

In the example above, the mount point /- is a filler that the automounter recognizes as a directive not to associate the entries in */etc/auto.direct* with any directory. Rather, the mount points are to be the ones mentioned in the map. (Remember, in a direct map the key is a full pathname.)

Mount point */home*

The mount point */home* is to be the directory under which the entries listed in */etc/auto.home* (an indirect map) are to be mounted. That is, they will be mounted under */tmp_mnt/home*, and a symbolic link will be provided between */home/directory* and */tmp_mnt/home/directory*.

Mount point */net*

Finally, the automounter will mount under the directory */net* all the entries under the special map *-hosts*. This is a built-in map that does not use any external files except the hosts database (*/etc/hosts* or the YP map *hosts.byname* if YP is running). Notice that since the automounter does not mount the entries until needed, the specific order is not important. Once the automount daemon is in place, a user entering the command

```
example % cd /net/gumbo
```

will change directory to the top of the hierarchy of files (i.e., the root file system) of the machine *gumbo* as long as the machine is in the hosts database and it exports any of its file systems. However, the user may not see under */net/gumbo* all the files and directories. This is because the automounter can mount only the *exported* file systems of host *gumbo*, in accordance with the restrictions placed on the exporting.

The actions of the automounter when the command in the example above is issued are as follows:

1. *ping* the null procedure of the server's mount service to see if it's alive.

2. Request the list of exported hierarchies from the server.

3. Sort the exported list according to length of pathname.

```
/usr/src
/export/home
/usr/src/sccs
/export/root/blah
```

This sorting ensures that the mounting is done in the proper order, that is, */usr/src* is done before */usr/src/sccs*.

4. Proceed down the list, mounting all the file systems at mount points in */tmp_mnt* (creating the mount points as needed).

5. Return a symbolic link that points to the top of the recently mounted hierarchy.

Note that, unfortunately, the automounter has to mount all the file systems that the server in question exports. Even if the request is as follows:

```
example % ls /net/gumbo/usr/include
```

the automounter mounts all of *gumbo*'s exported systems, not just */usr*.

In addition, unmounting that occurs after a certain amount of time has passed is from the bottom up. This means if one of the directories at the top is busy, the automounter has to remount the hierarchy and try again later.

Nevertheless, the *-hosts* special map provides a very convenient way for users to access directories in many different hosts without having to use *rlogin* or *rsh*. (These remote commands have to establish communication through the network every time they are invoked.) Furthermore, they no longer have to modify their */etc/fstab* files or mount the directories by hand as superuser.

Notice that both */net* and */home* are arbitrary names. The automounter will create them if they do not exist already.

4.2.2 Writing an Indirect Map

The syntax for an indirect map is:

```
key [mount-options] location
```

where *key* is the basename (not the full pathname) of the directory that will be used as mount point. Once the key is obtained by the automounter, it is suffixed to the mount point associated with it either by the command line or by the master map that invokes the indirect map in question.

For instance, one of the entries in the master map presented above as an example, reads:

```
/home /etc/auto.home -rw,intr,secure
```

Here */etc/auto.home* is the name of the indirect map that will contain the entries to be mounted under */home*.

A typical *auto.home* map might contain:

```
#key       mount-options    location
willow                      willow:/home/willow
cypress                     cypress:/home/cypress
poplar                      poplar:/home/poplar
pine                        pine:/export/pine
apple                       apple:/export/home
ivy                         ivy:/home/ivy
peach      -rw,hard         peach:/export/home
```

As an example, assume that the map above is on host *oak*. If user *laura* has an entry in the password database specifying her home directory as */home/willow/laura*, then whenever she logs into machine oak, the automounter will mount (as */tmp_mnt/home/willow*) the directory */home/willow* residing in machine *willow*. If one of the directories is indeed laura, she will be in her home directory, which is mounted read/write, interruptible and secure, as specified by the options field in the master map entry.

Suppose, however, that *laura*'s home directory is specified as */home/peach/laura*. Whenever she logs into oak, the automounter mounts the directory */export/home* from *peach* under */tmp_mnt/home/peach*. Her home directory will be mounted read/write, *hard*. Any option in the file entry overrides all options in the master map or the command line.

Now, assume the following conditions occur: user laura's home directory is listed in the password database as */home/willow/laura*. Machine *willow* exports its *home* hierarchy to the machines mentioned in *auto.home*. All those machines have a copy of the same *auto.home* and the same password database.

Under these conditions, user laura can run *login* or *rlogin* on any of these machines and have her home directory mounted in place for her.

Furthermore, now laura can also enter the command

```
% cd ~brent
```

and the automounter will mount brent's home directory for her (if all permissions apply).

On a network without YP, you have to change all the relevant databases (such as */etc/passwd*) on all systems on the network in order to accomplish this. On a network running YP, make the changes on the YP master server and propagate the relevant databases to the slave servers.

4.2.3 Writing a Direct Map

The syntax for a direct map (like that for an indirect map) is:

```
key [mount-options] location
```

where: *key* is the *full* pathname of the mount point. (Remember that in an indirect map this is not a full pathname.) *mount-options* are optional but, if present, override — for the entry in question — the options of the calling line, if any, or the defaults. *location* is the location of the resource, specified as *server:pathname.*

Of all the maps, the entries in a direct map most closely resemble, in their simplest form, what their corresponding entries in */etc/fstab* might look like. An entry that appears in */etc/fstab* as:

```
dancer:/usr/local /usr/local/tmp nfs  ro 0 0
```

appears in a direct map as:

```
/usr/local/tmp -ro dancer:/usr/local
```

The following is a typical */etc/auto.direct* map:

```
/usr/local \
                  /bin      -ro,soft   ivy:/export/local/PIris \
                  /share    -ro,soft   ivy:/export/local/share \
                  /src      -ro,soft   ivy:/export/local/src
/usr/man                    -ro,soft   oak:/usr/man \
                                       rose:/usr/man \
                                       willow:/usr/man
/usr/games                  -ro,soft   peach:/usr/games
/usr/spool/news             -ro,soft   pine:/usr/spool/news
/usr/frame                  -ro,soft   redwood:/usr/frame2.0 \
                                       balsa:/export/frame
```

There are a couple of unusual features in this map:
multiple mounts and multiple locations.
These are the subject of the next two subsections.

Multiple Mounts

A map entry can describe a multiplicity of mounts, where the mounts can be from different locations and with different mount options. Consider the first entry in the previous example:

```
/usr/local \
              /bin      -ro,soft   ivy:/export/local/PIris \
              /share    -ro,soft   ivy:/export/local/share \
              /src      -ro,soft   ivy:/export/local/src
```

This is, in fact, one long entry whose readability has been improved by splitting it into four lines by using the backslash and indenting the continuation lines with blank spaces or tabs. This entry mounts

/usr/local/bin, */usr/local/share* and */usr/local/src* from the server *ivy*, with the options read-only and soft. The entry could also read:

```
/usr/local \
               /bin      -ro,soft      ivy:/export/local/PIris \
               /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. The difference between the above and three separate entries, for example:

```
/usr/local/bin      -ro,soft      ivy:/export/local/PIris
/usr/local/share    -rw,secure    willow:/usr/local/share
/usr/local/src      -ro,intr      oak:/home/jones/src
```

is that the first case, multiple mount, guarantees that all three directories will be mounted when you reference one of them. In the case of the separate entries, if you, for instance, enter:

```
% cd /usr/local/bin
```

you cannot *cd* to one of the other directories using a relative path, because it is not mounted yet:

```
% cd ../src
../src: No such file or directory
```

A multiple mount obviates this problem. In multiple mounts, each file hierarchy is mounted on a subdirectory within another file hierarchy. When the root of the hierarchy is referenced, the automounter mounts the whole hierarchy.

The concept of *root* here is very important. The symbolic link returned by the automounter to the kernel request is a path to the mount root. This is the root of the hierarchy that is mounted under /tmp_mnt. This mount point should theoretically be specified:

```
parsley  /  -ro,intr  veg:/usr/greens
```

In practice, it is not specified because in a trivial case of a single mount as above, it is assumed that the location of the mount point is *at* the mount root or "/." So instead of the above it is perfectly acceptable, indeed preferable, to enter:

```
parsley  -ro,intr  veg:/usr/greens
```

The mount point specification, however, becomes important when mounting a hierarchy: here the automounter must have a mount point for each mount within the hierarchy. The example above is a good illustration of multiple, non-hierarchical mounts under */usr/local* when the latter is already mounted (or is a subdirectory of another mounted system).

When the root of the hierarchy has to be mounted also, it has to be specified in the map, and the entry becomes a "hierarchical" mount, which is a special case of multiple mounts.

The following illustration shows a true hierarchical mounting:

```
/usr/local \
                /          -rw,intr     peach:/export/local \
                /bin       -ro,soft     ivy:/export/local/PIris \
                /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.

A true hierarchical mount can be problematic if the server for the root of the hierarchy goes down. Any attempt to unmount the lower branches will fail, since the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down.

Finally, a word about mount options. In one of the examples above,

```
/usr/local \
              /bin      -ro,soft   ivy:/export/local/PIris \
              /share    -ro,soft   willow:/usr/local/share \
              /src      -ro,soft   oak:/home/jones/src
```

all three mounts share the same options. This could be modified to:

```
/usr/local   -ro,soft \
             /bin         ivy:/export/local/PIris \
             /share       willow:/usr/local/share \
             /src         oak:/home/jones/src
```

If one of the mount points needed a different specification, you could then write:

```
/usr/local   -ro,soft \
/bin                      ivy:/export/local/PIris \
/share       -rw,secure   willow:/usr/local/share \
/src                      oak:/home/jones/src
```

4.2.4 Multiple Locations

In the example for a direct map, which was:

```
/usr/local \
                       /bin     -ro,soft  ivy:/export/local/PIris \
                       /share   -ro,soft  ivy:/export/local/share \
                       /src     -ro,soft  ivy:/export/local/src
/usr/man                        -ro,soft  oak:/usr/man \
                                          rose:/usr/man \
                                          willow:/usr/man
/usr/games                      -ro,soft  peach:/usr/games
/usr/spool/news                 -ro,soft  pine:/usr/spool/news
/usr/frame                      -ro,soft  redwood:/usr/frame2.0 \
                                          balsa:/export/frame
```

the mount points */usr/man* and */usr/frame* list more than one location (three for the first, two for the second). This means that the mounting can be done from any of the replicated locations.

This procedure makes sense only when you are mounting a hierarchy read-only, since (at least theoretically) you must have some control over the locations of files you write or modify (that is, you don't want to modify files on one server on one occasion and, minutes later, modify the "same" file on another server). A good example is the man pages.

In a large network, more than one server may export the current set of manual pages. It does not matter which server you mount them from, as long as the server is up and running and exporting its file systems. In the example above, multiple mount locations are expressed as a list of mount locations in the map entry:

```
/usr/man -ro,soft oak:/usr/man rose:/usr/man willow:/usr/man
```

This could also be expressed as a comma separated list of servers, followed by a colon and the pathname (as long as the pathname is the same for all the replicated servers):

```
/usr/man -ro,soft oak,rose,willow:/usr/man
```

Here you can mount the man pages from the servers *oak*, *rose*, or *willow*. From this list of servers the automounter first selects those that are on the local network and *pings* these servers. This launches a series of RPC requests to each server. The first server to respond is selected, and an attempt is made to mount from it. Note that the list does not imply an ordering.

This redundancy, very useful in an environment where individual servers may or may not be exporting their file systems, is enjoyed only at mount time. There is no status checking of the mounted-from server by the automounter once the mount occurs. If the server goes down while the mount is in effect, the file system becomes unavailable. An option here is to wait five minutes until the auto-unmount takes place and try again. Next time around the automounter will choose one of the other, available servers. Another option is to use the *umount* command, inform the automounter of the change in the mount table and retry the mount.

4.2.5 Specifying Subdirectories

The earlier subsection, ‘‘Writing an Indirect Map’’ showed the following typical *auto.home* file:

```
#key      mount-options   location
willow                    willow:/home/willow
cypress                   cypress:/home/cypress
poplar                    poplar:/home/poplar
pine                      pine:/export/pine
apple                     apple:/export/home
ivy                       ivy:/home/ivy
peach     -rw,hard        peach:/export/home
```

Given this *auto.home* indirect file, every time a user wants to access a home directory in, say, */home/willow*, all the directories under it will be mounted. Another way to organize an *auto.home* file is by user name, as in:

```
#key   mount-options   location
john                   willow:/home/willow/john
mary                   willow:/home/willow/mary
joe                    willow:/home/willow/joe
```

The above example assumes that home directories are of the form */home/user* rather than */home/server/user*. If a user now enters the following command:

```
% ls   john   mary
```

the automounter has to perform the *equivalent* of the following actions:

```
mkdir /tmp_mnt/home/john
mount willow:/home/willow/john /tmp_mnt/home/john
ln -s /tmp_mnt/home/john /home/john

mkdir /tmp_mnt/home/mary
mount willow:/home/willow/mary /tmp_mnt/home/mary
ln -s /tmp_mnt/home/mary /home/mary
```

However, the complete syntax of a line in a direct or indirect map is actually:

```
key  [mount-option]  server:pathname[:subdirectory]
```

Until now you used the form *server:pathname* to indicate the location. This is also an ideal place for you to indicate the subdirectory, like this:

```
#key   mount-options   location
john                   willow:/home/willow:john
mary                   willow:/home/willow:mary
joe                    willow:/home/willow:joe
```

Here *john*, *mary*, and *joe* are entries in the *subdirectory* field. Now when a user refers to *john*'s home directory, the automounter mounts *willow:/home/willow*. It then places a symbolic link between */tmp_mnt/home/willow/john* and */home/john*.

If the user then requests access to *mary*'s home directory, the automounter sees that *willow:/home/willow* is already mounted, so all it has to do is return the link between */tmp_mnt/home/willow/mary* and */home/mary*. In other words, the automounter now only does:

```
mkdir /tmp_mnt/home/john
mount willow:/home/willow /tmp_mnt/home
ln -s /tmp_mnt/home/john /home/john

ln -s /tmp_mnt/home/mary /home/mary
```

In general, it is a good idea to provide a *subdirectory* entry in the *location* when different map entries refer to the same mounted file system from the same server.

4.2.6 Metacharacters

The automounter recognizes some characters as having a special meaning. Some are used for substitutions, some to escape other characters.

Ampersand (&)

If you have a map with a lot of subdirectories specified, for example:

```
#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
        [. . .]
```

consider using string substitutions. You can use the ampersand character (&) to substitute the key wherever it appears. Using the ampersand, the above map now looks as follows:

```
#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 results in:

```
#key      mount-options   location
willow                    &:/home/&
peach                     &:/home/&
pine                      &:/home/&
oak                       &:/home/&
poplar                    &:/home/&
          [. . .]
```

Asterisk (*)

Notice that all the above entries have the same format. This permits you to use the catch-all substitute character, the asterisk (*). The asterisk reduces the whole thing to:

```
*                   &:/home/&
```

where each ampersand is substituted by the value of any given key. Notice that once the automounter reads the catch-all key it does not continue reading the map, so that the following map would be viable:

```
#key      mount-options   location
oak                       &:/export/&
poplar                    &:/export/&
*                         &:/home/&
```

whereas in the next map the last two entries would always be ignored:

```
#key      mount-options   location
*                         &:/home/&
oak                       &:/export/&
poplar                    &:/export/&
```

You can also use key substitutions in a direct map, in situations like the following:

```
/usr/man                             willow,cedar,poplar:/usr/man
```

which is a good candidate to be written as:

```
/usr/man                             willow,cedar,poplar:&
```

Notice that the ampersand substitution uses the whole key string, so if the key in a direct map starts with a / (as it should), that slash is carried over, and you could not do something like

```
/progs   &1,&2,&3:/export/src/progs
```

because the automounter would interpret it as:

```
/progs   /progs1,/progs2,/progs3:/export/src/progs
```

Backslash (\)

Under certain circumstances you may have to mount directories whose names may confuse the automounter's map parser. An example might be a directory called *rc0:dk1*; this could result in an entry like:

```
/junk   -ro   vmsserver:rc0:dk1
```

The presence of the two colons in the location field will confuse the automounter's parser. To avoid this confusion, use a backslash to escape the second colon and remove its special meaning of separator:

```
/junk   -ro   vmsserver:rc0\:dk1
```

Double quotes (")

You can also use double quotes, as in the following example, where they are used to hide the blank space in the name:

```
/smile          dentist:/"front teeth"/smile
```

4.2.7 Environment Variables

You can use the value of an environmental variable by prefixing a dollar sign ($) to its name. You can also use braces to delimit the name of the variable from appended letters or digits. You can use environmental variables anywhere in an entry line, except as a *key*.

The environmental variables can be inherited from the environment or can be defined explicitly with the **-D** command line option. For instance, if you want each client to mount client-specific 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 it would be:

```
/mystuff        cypress,ivy,balsa:/export/hostfiles/willow
```

This scheme is viable within a small network, but maintaining this kind of host-specific maps across a large network would soon become unfeasible. The solution in this case would be to start the automounter with a command line similar to the following:

```
automount -D HOST=`hostname` ......
```

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.

4.2.8 Including Other Maps

A line of the form *+mapname* causes the automounter to consult the mentioned map as if it were included in the current map. If *mapname* is a relative pathname (no slashes), the automounter assumes it is an YP map. If the pathname is an absolute pathname the automounter looks for a local map of that name. If the mapname starts with a dash (–), the automounter consults the appropriate built-in map.

For instance, you can have a few entries in your local *auto.home* map for the most commonly accessed home directories, and follow them with the included YP map:

```
ivy   -rw,intr,noquota                &:/home/&
oak   -rw,intr,noquota                &:/export/home
+auto.home
```

After consulting the included map, the automounter continues scanning the current map if no match is found, so you can add more entries, for instance:

```
ivy      -rw,intr,noquota    &:/home/&
oak      -rw,intr,noquota    &:/export/home
+auto.home
*   -rw    &:/home/&
```

Finally, as mentioned before, the map included can be a local file, or even a built-in map:

```
            +auto.home.financeo     # YP map
            +auto.home.sales        # YP map
            +auto.home.engineering  # YP map
            +/etc/auto.mystuff      # local map
            +auto.home              # YP map
            +-hosts                 # built-in hosts map
            *        &:/export/&    # wild card
```

Notice that in all cases the wild card should be the last entry, because the automounter does not continue consulting the map after it, on the assumption that the wild card will have found a match.

4.3 Starting *automount*

Once the maps are written, you should make sure that there are no equivalent entries in */etc/fstab*, and that all the entries in the maps refer to NFS exported files. The *automount*(8) manual page contains a complete description of all *automount* options.

The *mount-options* that you can specify at either the command line or in the maps are the same as those for a standard NFS *mount*, excepting *bg* (background) and *fg* (foreground), which do not apply.

By default, the file */etc/init.d/network* starts the automounter at boot time through the lines:

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

That is, if the file */usr/etc/automount* exists, start it. When started like this, with no option, the automounter looks for a YP map called *auto.master*. If it finds it, it follows the instructions contained there. If YP is not running, or the map is not to be found, the automounter exits silently. The **-m** option instructs the automounter not to look for the YP map. The **-f** option instructs it to look for the file named immediately after the option. If no **-m** or **-f** options are specified, the automounter expects a series of mount points and maps (and optional mount options) specified on the command line.

Given the following set of three maps:

auto.master

```
#Mount-point   Map                 Mount-options
/net           -hosts
/home          /etc/auto.home      -rw,intr,secure
/-             /etc/auto.direct    -ro,intr
```

auto.home

```
#key      mount-options   location
willow                    willow:/home/willow
cypress                   cypress:/home/cypress
poplar                    poplar:/home/poplar
pine                      pine:/export/pine
apple                     apple:/export/home
ivy                       ivy:/home/ivy
peach     -rw,hard        peach:/export/home
```

auto.direct

```
/usr/local \
                  /bin     -ro,soft   ivy:/export/local/PIris \
                  /share   -ro,soft   ivy:/export/local/share \
                  /src     -ro,soft   ivy:/export/local/src
/usr/man                   -ro,soft   oak:/usr/man \
                                      rose:/usr/man \
                                      willow:/usr/man
/usr/games                 -ro,soft   peach:/usr/games
/usr/spool/news            -ro,soft   pine:/usr/spool/news
/usr/frame                 -ro,soft   redwood:/usr/frame1.3 \
                                      balsa:/export/frame
```

you can invoke the automounter (either from the command line or, preferably, from *rc.local*) in one of the following ways:

1. You can specify all arguments to the automounter without reference to the master map (either YP or local), as in:

```
automount /net -hosts /home /etc/auto.home \
-rw,intr,secure /- /etc/auto.direct -ro,intr
```

2. You can specify the same in the *auto.master* file, and instruct the automounter to look in it for instructions:

```
automount -f /etc/auto.master
```

3. You can specify more mount points and maps in addition to those mentioned in the master map, as follows:

   ```
   automount -f /etc/auto.master /src /etc/auto.src -ro,soft
   ```

   or

   ```
   automount /src /etc/auto.src -ro,soft
   ```

   The first example specifies additions to the local master map, the second additions to the YP master map.

4. You can nullify one of the entries in the master map. (This is particularly useful if you use a map that you cannot modify and does not meet the needs of your machine):

   ```
   automount -f /usr/lib/auto.master /home -null
   ```

   This cancels the entry for */home* in the master map (it could also be the YP master map).

5. You can replace one of the entries with your own:

```
automount -f /usr/lib/auto.master /home /myown/auto.home -rw,intr
```

In the example above, the automounter first mounts all items in the map */myown/auto.home* under the directory */home*. Then, when it consults the master file */usr/lib/auto.master* and reaches the line corresponding to */home* it simply ignores it, since it has already mounted on it.

Given the *auto.master* file of the previous example, commands (1) and (2) are equivalent *as long as your network does not have a distributed auto.master file*. This file is only available on networks running YP. If your network includes a distributed YP *auto.master* file, the second example would have to be modified in the following way to be equivalent to example 1:

```
automount -m -f /etc/auto.master
```

The -m option instructs the automounter not to consult the master file distributed by YP. However, if you do not run YP, you do not have to specify the -m option. The automounter is completely silent when it does not find a distributed master file.

You can log in as superuser and type any of the above commands at shell level to start the automounter. Ideally, you should edit *rc.local* and include your preferred command there.

4.3.1 The Temporary Mount Point

The default name for all mounts is */tmp_mnt*. Like the other names, this is arbitrary. It can be changed at startup time by use of the -M option. For instance:

```
automount -M /auto ...
```

causes all mounts to happen under the directory */auto*, which the automounter will create if it does not exist. You should not designate a directory in a read only file system, as the automounter would not be able to modify anything then.

4.3.2 The Mount Table

Every time the automounter mounts or unmounts a file hierarchy, it modifies */etc/mtab* to reflect the current situation. The automounter keeps an image in memory of */etc/mtab*, and refreshes this image every time it performs a mounting or an automatic unmounting. If you use the *umount* command to unmount one of the automounted hierarchies (a directory under */tmp_mnt*), the automounter should be forced to re-read the */etc/mtab* file. To do that, enter the command:

```
example % ps -ef | grep automount | egrep -v grep
```

This gives you the process ID of the automounter. The automounter is designed so that on receiving a SIGHUP signal it re-reads */etc/mtab*. So, to send it that signal, enter:

```
% kill -1 PID
```

where *PID* stands for the process ID you obtained from the previous *ps* command.

4.3.3 Modifying the Maps

You can modify the automounter maps at any time, but that does not guarantee that all your modifications will take effect the next time the automounter mounts a hierarchy. It depends on what map you modify and what kind of modification you introduce. You may have to reboot the machine. This is generally the simplest way of restarting the automounter, although, if it is used sparingly, you could theoretically kill it and restart it from the command line.

Modifying the Master Map

The automounter consults the master map only at startup time. A modification to the master map will take effect only the next time you reboot the machine.

Modifying Indirect Maps

You can modify, delete, or add to indirect maps. The change takes effect the next time the map is used, which is next time a mount is done.

Modifying Direct Maps

Each entry in a direct map is an automount mount point, and the entry mounts itself at these mount points at startup. Therefore, adding or deleting an entry in a direct map will take effect only the next time you reboot the machine. However, existing entries can be modified (changing mount options or server names, and so on, but not name of mount points) while the automounter is running, and will take effect when the entry is next mounted, because the automounter consults the direct maps whenever a mount has to be done.

For instance, suppose you modify the file */etc/auto.direct* so that the directory */usr/src* is now mounted from a different server. The new entry takes effect immediately (if */usr/src* is not mounted at this time) when you try to access it. If it is mounted now, you can wait until the auto unmounting takes place, and then access it. If this is not satisfactory, you can unmount with the *umount* command, notify *automount* that the mount table has changed, and then access it. The mounting should now be done from the new server. However, if you wanted to delete the entry, you would have to reboot the machine for the deletion to take effect.

For this reason, and because they do not clutter the mount table like direct maps do, indirect maps are preferable, and should be used whenever possible.

4.3.4 Mount Point Conflicts

You can cause a mount conflict by mounting one home directory on top of another. The conflict occurs when you have a home partition, on a local disk, that is mounted on */home* and you want to use the automounter to mount other home directories on the same mount point. If you give it the mount point */home* then the automounter will hide the local home partition whenever you try to reach it.

The solution is to mount the partition somewhere else, for instance on */export/home*. You would then need, for instance, an entry in */etc/fstab* that says:

```
/dev/z              ...                 /export/home      ...
```

(where z stands for the name of the partition) and, assuming that the master file contains a line similar to:

```
/home /etc/auto.home
```

an entry in *auto.home* that says:

```
terra               terra:/export/home
```

where *terra* is the name of the machine.

If the partition is set up such that home directories are found as */home/machine/user*, move all the directories at the */user* level one level up, to eliminate the */machine* level:

```
# cd /home
# mv machine/* .
# rmdir machine
```

There is no need to change the */etc/passwd* entry for the user. His or her home directory will still be accessible through */home/machine/user*, as before. Instead of doing a mount, the automounter will recognize that the file system is on the same machine and will establish a symbolic link from */home/machine* to */export/home*.

4.4 Problem Solving

From time to time, you may encounter problems with the automounter. This section is aimed at making the problem solving process easier. It is divided into two subsections.

The first subsection expands on the bird's eye view explanation of how the automounter works presented at the beginning of this chapter. This explanation is written especially for advanced system administrators and programmers, though users may want to read it to get an idea of the issues involved.

The second subsection presents a list of the error messages the automounter generates. The list is divided in two parts: error messages generated by the verbose (–v) option of *automount*, and error messages that may appear at any time.

In general, it is recommended that you start the automounter with the verbose option, since otherwise you may experience problems without knowing why.

4.4.1 Automount Sequence of Events

There are two distinct stages in the automounter's actions: The initial stage, boot time, when *rc.local* boots the automounter. The mounting stage, when a user tries to access a file or directory in a remote machine.

At the initial stage, when *rc.local* invokes *automount*, it opens a UDP socket and registers it with the portmapper service as an NFS server port. It then forks off a server daemon that listens for NFS requests on the socket. The parent process proceeds to mount the daemon at its mount points within the file system (as specified by the maps). Through the *mount*(2) system call, it passes the server daemon's port number and an NFS *file handle* that is unique to each mount point. The arguments to the *mount*(2) system call vary according to the kind of file system; for NFS file systems, the call is

```
mount ( "nfs", "/usr", &nfs_args );
```

where *&nfs_args* contains the network address for the NFS server. By having the network address in *&nfs_args* refer to the local process (the automount daemon), *automount* in fact deceives the kernel into treating it as if it were an NFS server. Instead, once the parent process completes its calls to *mount*(2), it exits, leaving the daemon to serve its mount points.

In the second stage, when the user actually requests access to a remote file hierarchy, the daemon intercepts the kernel NFS request and looks up the name in the map associated with the directory.

Taking the location (*server:pathname*) of the remote file system from the map, the daemon then mounts the remote file system under the directory */tmp_mnt*. It answers the kernel, saying it is a symbolic link. The kernel sends an NFS READLINK request, and the automounter returns a symbolic link to the real mount point under */tmp_mnt*.

The behavior of the automounter is affected by whether the name is found in a direct or an indirect map. If the name is found in a direct map, the automounter emulates a symbolic link, as stated above. It responds as if a symbolic link exists at its mount point. In response to a GETATTR, it describes itself as a symbolic link. When the kernel follows up with a READLINK it returns a path to the *real* mount point for the remote hierarchy in */tmp_mnt*.

If, on the other hand, the name is found in an indirect map, the automounter emulates a directory of symbolic links. It describes itself as a directory. In response to a READLINK, it returns a path to the mount point in */tmp_mnt*, and a *readdir*(3) of the automounter's mount point returns a list of the entries that are currently mounted.

Whether the map is direct or indirect, if the file hierarchy is already mounted and the symbolic link has been read recently, the cached symbolic link is returned immediately. Since the automounter is on the same host, the response is much faster than a READLINK to a remote NFS server. On the other hand, if the file hierarchy is not mounted, a small delay will occur while the mounting takes place.

4.4.2 Error Messages Related to *automount*

The following paragraphs are labeled with the error message you are likely to see if the automounter fails, and an indication of what the problem may be.

Error Messages Generated by the Verbose Option

no mount maps specified

The automounter was invoked with no maps to serve, and it cannot find the YP *auto.master* map. It exits. Recheck the command, or restart YP if that was the intention.

mapname: Not found

The required map cannot be located.
This message is produced only when the **-v** option is given.
Check the spelling and pathname of the map name.

leading space in map entry entry text in mapname

The automounter has discovered an entry in an automount map that contains leading spaces. This is usually an indication of an improperly continued map entry, e.g.

```
foo
    /bar                             frobz:/usr/frotz
```

In the example above, the warning is generated when the automounter encounters the second line, because the first line should be terminated with a backslash (\).

bad key <key> *in indirect map mapname*

While scanning an indirect map the automounter has found an entry key containing a "/". Indirect map keys must be simple names — not pathnames.

bad key <key> *in direct map mapname*

While scanning a direct map the automounter has found an entry key without a prepended "/". Keys in direct maps must be full pathnames.

YP bind failed

The automounter was unable to communicate with the ypbind daemon. This is information only — the automounter will continue to function correctly provided it requires no explicit YP support. If you need YP, check to see if there is a *ypbind* daemon running.

Couldn't create mountpoint <mountpoint>*: reason*

The automounter was unable to create a mountpoint required for a mount. This most frequently occurs when attempting to hierarchically mount all of a server's exported file systems. A required mountpoint may exist only in a file system that cannot be mounted (it may not be exported) and it cannot be created because the exported parent file system is exported read only.

WARNING: mountpoint already mounted on

The automounter is attempting to mount over an existing mountpoint. This is indicative of an internal error in the automounter (a bug).

server:pathname already mounted on mountpoint

The automounter is attempting to mount over a previous mount of the same file system. This could happen if an entry appears both in */etc/fstab* and in an automounter map (either by accident or because the output of *mount* -p was redirected to *fstab*). Delete one of the redundant entries.

can't mount server:pathname: reason

The mount daemon on the server refuses to provide a file handle for *server:pathname*. Check the export table on server.

remount server:pathname on mountpoint: server not responding

The automounter has failed to remount a file system it previously unmounted. This message may appear at intervals until the file system is successfully remounted.

WARNING: mountpoint not empty!

The mount point is not an empty directory. The directory mountpoint contains entries that will be hidden while the automounter is mounted there. This is advisory only.

General Error Messages

pathok: couldn't find devid <device id>

An internal automounter error (bug).

WARNING: default option "option" ignored for map mapname

where option is an unrecognized default mount option for the map mapname.

option ignored for key in mapname

The automounter has detected an unknown mount option. This is advisory only. Correct the entry in the appropriate map.

bad entry in map mapname "key"
map mapname, key <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.

Can't get my address

The automounter cannot find an entry for its host in */etc/hosts* (or YP *hosts.byname*).

Cannot create UDP service

Automounter cannot establish a UDP connection.

svc_register failed

Automounter cannot register itself as an NFS server. Check the kernel configuration file.

couldn't create pathname: reason

Where pathname is */tmp_mnt* or the argument to the -M command line option.

Can't mount mountpoint: reason

The automounter couldn't mount its daemon at mountpoint.

Can't update pathname

Where pathname is */etc/mtab* it means that the automounter was not able to update the mount table. Check the permissions of the file.

exiting

This is an advisory message only. The automounter has received a SIGTERM (has been killed) and is exiting.

WARNING: pathname: line line number: bad entry

Where pathname is */etc/mtab* it means that the automounter has detected a malformed entry in the */etc/mtab* file.

server:pathname no longer mounted

The automounter is acknowledging that server:pathname which it mounted earlier has been unmounted by the umount command. The automounter will notice this within 1 minute of the unmount or immediately if it receives a SIGHUP.

trymany: servers not responding: reason

No server in a replicated list is responding. This may indicate a network problem.

server:pathname - linkname : dangerous symbolic link

The automounter is trying to use *server:pathname* as a mountpoint but it is a symbolic link that resolves to a pathname referencing a mount point outside of */tmp_mnt* (or the mount point set with the **-M** option). The automounter refuses to do this mount because it could cause problems in the host's file system, e.g. mounting on */usr* rather than in */tmp_mnt*.

host server not responding

The automounter attempted to contact server but received no response.

Mount of server:pathname on mountpoint: reason

The automounter failed to do a mount. This may indicate a server or network problem.

pathconf: server: server not responding

The automounter is unable to contact the mount daemon on server that provides (POSIX) pathconf information.

pathconf: no info for server:pathname

The automounter failed to get pathconf information for pathname.

hierarchical mountpoints: pathname1 and pathname2

The automounter does not allow its mountpoints to have a hierarchical relationship, i.e., an automounter mountpoint must not be contained within another automounted file system.

mountpoint: Not a directory

The automounter cannot mount itself on mountpoint because it's not a directory. Check the spelling and pathname of the mount point.

dir mountpoint must start with '/'

Automounter mount point must be given as full pathname. Check the spelling and pathname of the mount point.

mapname: yp_err

Error in looking up an entry in an YP map. May indicate YP problems.

hostname: exports: rpc_err

Error getting export list from hostname. This indicates a server or network problem.

nfscast: cannot send packet: reason

The automounter cannot send a query packet to a server in a list of replicated file system locations.

nfscast: cannot receive reply: reason

The automounter cannot receive replies from any of the servers in a list of replicated file system locations.

nfscast:select: reason
Cannot create socket for nfs: rpc_err

These error messages indicate problems attempting to *ping* servers for a replicated file system. This may indicate a network problem.

NFS server (pid@mountpoint) not responding still trying

An NFS request made to the automount daemon with PID serving mountpoint has timed out. The automounter may be temporarily overloaded or dead. Wait a few minutes; if the condition persists, the easiest solution is to reboot the client. If not, use *fuser*(1M) to find and kill all processes that use automounted directories (or, change to a non automounted directory in the case of a shell), kill the current automount process and restart it again from the command line. If this does not work, reboot.

5. The YP Service

This chapter introduces YP-related terms, provides an overview of YP, explains the installation and administration of the YP system, tells how to debug YP when problems occur, and discusses file access policies and special security issues raised by the YP environment.

5.1 The YP Service

The YP is NFS's distributed network lookup service.

- YP is a distributed system. The database is fully replicated at several sites, each of which runs a server process for the database. These sites are known as YP servers. At steady state, it does not matter which server process answers a client request; the answer is the same throughout the network. This allows multiple servers per network, and gives the YP service a high degree of availability and reliability. Servers propagate updated databases among themselves, ensuring consistency.
- YP is a lookup service. It maintains a set of databases that you can query. A client can ask for the value associated with a particular key within a database, and can enumerate every key-value pair within a database. A key is the argument you use when you search a database.
- YP is a network service. It uses a standard set of access procedures to hide the details of where and how data is stored.

The basic function of YP is to synchronize the databases that many machines in a network use. The YP service eases the burden of network administration by allowing networkwide databases to be maintained and updated in a central place. The YP service provides a convenient, automated means of propagating network changes and information. YP scales to about 1000 workstations. Networks larger than 1000 workstations can use *named*(1M) to extend network synchronization functionality.

It is not necessary to use YP with NFS. The YP service is turned on by the person functioning as NFS site or system administrator and is transparent to the user. Changes made to the */etc/hosts* file in a NFS network that does not use YP must be made to each workstation manually.

The procedure for turning on YP in NFS is described in Section 4.3.

If you choose not to use the YP, no specific action is required. You must take specific action to enable the YP, as described in Section 4.3.

5.1.1 The YP Map

The YP system uses information stored in YP databases that are referred to as maps. Each map contains a set of keys and associated values. For example, in a map called *hosts.byname*, all the host names within a network are the keys, and the Internet addresses of these host names are the values. Each YP map has a *mapname* that programs use to access it. Programs must know the format of the data in the map.

Many of the current maps are derived from ASCII files traditionally found in */etc*: *hosts*, *group*, *passwd*, and a few others. The format of the data within the YP map is identical (in most cases) to the format within the ASCII file. Maps are implemented by *dbm*(3B) files located in the subdirectories of the directory */usr/etc/yp* on YP server machines.

Maps sometimes have nicknames. Although the *ypcat* command is a general YP database print program, it knows about the standard files in the YP. Thus *ypcat hosts* is translated into *ypcat hosts.byaddr* since there is no file called *hosts* in the YP. The command *ypcat* –x furnishes a list of expanded nicknames.

5.1.2 The YP Domain

A YP domain is a named set of YP maps. Use the *domainname*(1) command to determine and set your YP domain. Note that YP domains are different from both Internet domains and *sendmail* domains. A YP domain is simply a directory in */usr/etc/yp* where a YP server holds all of the YP maps. The name of the subdirectory is the name of the domain. For example, maps for the *literature* domain would be in */usr/etc/yp/literature.*

You must specify a domain name for retrieving data from a YP database. Each machine on the network belongs to a default domain set at boot time in */etc/init.d/network* with the *domainname*(1) command, using the contents of the file */usr/etc/yp/ypdomain* as the domain name. A domain name must be set on all machines, both servers and clients. Use only one domain name for all machines on a network.

5.1.3 Servers and Clients

A YP client runs YP processes and requests data from databases on a YP server. Servers provide resources, while clients consume them. The terms *server* and *client* do not necessarily indicate machines. Consider both the NFS (network file system), and the YP.

NFS

The NFS allows client machines to mount remote filesystems and access files in place, provided a server machine has exported the filesystem. However, a server that exports filesystems may also mount remote filesystems exported by other machines, thus becoming a client. So a given machine may be both server and client, or client only, or server only.

YP

The YP server, by contrast, is a process rather than a machine, A process can request information out of the YP database, obviating the need to have such information on every machine. All processes that make use of YP services are YP clients. Sometimes clients are served by YP servers on the same machine, but other times by YP servers running on another machine. If a remote machine running a YP server process crashes, client processes can obtain YP services from another machine. Thus, the network YP service will remain available even if a single YP host machine goes down.

5.1.4 Masters and Slaves

In the YP environment, only a few machines have a set of YP databases. The YP service makes the database set available over the network. Two kinds of machines have databases: a YP slave server and a YP master server. The master server updates the databases of the slave servers. Make changes to databases only on the YP master server. The changes propagate from the master server to the slave servers. If you create or change YP databases on slave server machines instead of master server machines, the YP's update algorithm will be broken. Always do all the database creation and modification on the master server machine.

A server may be a master with regard to one map, and a slave with regard to another. Random assignment of maps to server machines could cause a great deal of confusion. Make a single server the master for all the maps created by *ypinit*(1M) within a single domain. This document assumes that one server is the master for all maps in the database.

5.2 YP Overview

The YP can serve any number of databases. Typically, these include some files that used to be found in */etc*. For example, before NFS included YP, programs would read the */etc/hosts* file to find an Internet address. When a new machine was added to the network, a new entry had to be added to every machine's */etc/hosts* file. With YP, programs that use the */etc/hosts* file now do a Remote Procedure Call (*rpc*) to a YP server to get the information. A client machine makes an RPC call to a YP server each time it needs information from a YP database. The *ypbind* daemon remembers the name of a server. When a client boots, *ypbind* broadcasts asking for the name of the YP server. Similarly, *ypbind* broadcasts asking for the name of a new YP server if the old server crashes. The *ypwhich* command gives the name of the server that *ypbind* currently points at.

To become a server, a machine must contain the YP databases, and must also be running the YP daemon *ypserv*. The *ypinit* command invokes this daemon automatically. It also takes a flag indicating the creation of a master or a slave. When updating the master copy of a database, you can force the change to be propagated to all the slaves with the *yppush* command. This pushes the information out to all the slaves. Conversely, from a slave, the *ypxfr* command gets the latest information from the master.

The makefile in */etc/yp* first executes *makedbm* to make a new database, and then calls *yppush* to propagate the change throughout the network.

By default, IRIS workstations have a number of files in their YP: */etc/bootparams /etc/rpc /etc/passwd, /etc/group, /etc/hosts, /etc/networks, /etc/services, /etc/protocols, /etc/ethers,* and */usr/lib/aliases*. In addition, there is the *netgroup*(5) file, which defines network wide groups, and is used for permission checking when doing remote mounts, remote logins, and remote shells.

Most of the information describing the structure of the YP system and the commands available for that system is contained in manual pages and is not repeated here. For quick reference, this section lists the manual pages and an abstract of their contents. For more information, see the manual pages at the end of this manual.

ypserv(1M)	describes the processes that comprise the YP system. These processes are */usr/etc/ypserv*, the YP database server daemon, and */usr/etc/ypbind*, the YP binder daemon. *ypserv* must run on each YP server machine. *ypbind* must run on all machines that use YP services, both servers and clients.
ypfiles(1M)	describes the database structure of the YP system.
ypinit(1M)	constructs many maps from files located in */etc*, such as */etc/hosts*, */etc/passwd*, and others. The database initialization tool *ypinit*(1M) does all such construction automatically. Also, it constructs initial versions of maps required by the system but not built from files in */etc*; an example is the map *ypservers*. Use this tool to set up the master YP server and the slave YP servers for the first time. Use *ypinit* to construct initial versions of maps rather than as an administrative tool for running systems.

ypmake(1M)	describes the use of */usr/etc/yp/Makefile*, the file that builds several commonly changed components of the YP's database. *ypmake* builds these maps from several ASCII files normally found in */etc*: *bootparams*, *passwd*, *hosts*, *group*, *netgroup*, *networks*, *protocols*, *rpc*, and *services*, as well as */usr/lib/aliases*.
makedbm(1M)	describes a low-level tool for building a *dbm* file that is a valid YP map. You can use *makedbm* to build or rebuild databases not built from */etc/yp/Makefile*. You can also use *makedbm* to disassemble a map so that you can see the key-value pairs that comprise it. You can also modify the disassembled form with standard tools (such as editors, *awk*, *grep*, and *cat*). The disassembled form is in the form required for input back into *makedbm*.
ypxfr(1M)	moves a YP map from one YP server to another, using the YP itself as the transport medium. You can run it interactively, or periodically from *crontab*. Also, *ypserv* uses *ypxfr* as its transfer agent when it is asked to transfer a map.
yppush(1M)	describes a tool to administer a running YP system. It runs on the master YP server. It requests each of the *ypserv* processes within a domain to transfer a particular map, waits for a summary response from the transfer agent, and prints out the results for each server.
ypset(1M)	tells a *ypbind* process (the local one, by default) to get YP services for a domain from a named YP server. This is not for casual use.
yppoll(1M)	asks any *ypserv* for the information it holds internally about a single map.
ypcat(1)	lists the contents of a YP map. Use it when you do not care which server's map version you see. If you need to see a particular server's map, *rlogin* or *rsh* to that server and use *makedbm*.
ypmatch(1M)	prints the value for one or more specified keys in a YP map. Again, you have no control over which server's version of the map you are seeing.

ypwhich(1M)	tells you which YP server a node is using at the moment for YP services, or which YP server is master of some named map.

5.3 YP Installation and Administration

Nine installation and administration topics are covered in this section.

Topic	Section
Setting Up a Master YP Server	4.3.1
Altering a YP Client's Database to Use YP Services	4.3.2
Setting Up a Slave YP Server	4.3.3
Setting Up a YP Client	4.3.4
Modifying Individual YP Maps after YP Installation	4.3.5
Propagating a YP Map	4.3.6
Making New YP Maps after YP Installation	4.3.7
Adding a New YP Server Not in the Original Set of YP Servers	4.3.8
Changing the Master Server	4.3.9

Table 5-1. YP Installation Topics

5.3.1 Setting Up a Master YP Server

To create a new master server, follow these steps:

1. Become the superuser and change your current directory to */usr/etc/yp*. Type:

   ```
   su
   cd /usr/etc/yp
   ```

2. Check these files in */etc*: *passwd*, *hosts*, *ethers*, *group*, *networks*, *protocols*, *services*, and *rpc*. Make sure they are complete and reflect a current picture of your system. Also check */usr/lib/aliases*.

3. If you know how */etc/netgroup* is going to be set up, set it up before running *ypinit*. If you do not know, *ypinit* can make an empty *netgroup* map.

4. Run *ypinit*(1M) with the –m switch. Set up the default domain name and the hostname. Define the domain name in */usr/etc/yp/ypdomain* and the hostname in */etc/sys_id*. The usual case is that *domainname* and *hostname* have been set up from */etc/init.d/network*.

   The system queries whether you want the procedure to terminate at the first nonfatal error, in which case you can fix the problem and restart *ypinit*. Do this if you have not done this procedure before. You can also choose to continue despite nonfatal errors. In this second case, try to fix all the problems by hand, or fix some, then restart *ypinit*. *ypinit* prompts for a list of other hosts that also are YP servers. Initially, this is the set of YP slave servers for this domain. You need not add any other hosts at this time, but if you know that you will be setting up some more YP servers, add them now. This saves some work later, and there is no performance penalty for doing it.

5. For security reasons, you can restrict access to the master YP machine to a smaller set of users than that defined by the complete */etc/passwd*. To do this, copy the complete file to a file name other than */etc/passwd*, for example */etc/passwd.yp*. Then edit out undesired users from the remaining */etc/passwd*. For a security-conscious system, this smaller file should not include the YP escape entry discussed in the next section.

   Next, edit the startup script, */etc/init.d/network*, to change the pathname */etc/passwd* to the new pathname of the YP file (*/etc/passwd.yp*, if you use the name given as an example above). This enables the slave machines to obtain all the user IDs and passwords from the larger file, */etc/passwd.yp*.

   You can restrict use of mail aliases in a similar manner. Just as you copied your */etc/passwd* file, copy */usr/lib/aliases* to another file, such as */usr/lib/aliases.yp*. Edit this file to reflect the security considerations you used in editing the */etc/passwd.yp* file.

   Note that for some systems */etc/passwd* and */usr/lib/aliases* files might be larger than their YP counterparts.

   To make YP aware of the */usr/lib/aliases.yp* and */etc/passwd.yp* files, you must make note of them in a file read by YP. Create the file */etc/config/ypmaster.options*. In this file, include the following lines:

```
PWFILE =/etc/passwd.yp
ALIASES = /usr/lib/aliases.yp
```

If you did not use the file names */usr/lib/aliases.yp* and */etc/passwd.yp*, substitute your file names and paths in the */etc/config/ypmaster.options* file.

6. To start providing YP services, type:

   ```
   /usr/etc/ypserv
   ```

   To make sure that the YP server process is started automatically on subsequent reboots, type the command:

   ```
   /etc/chkconfig ypserv on
   ```

7. Starting the *ypbind* process automatically also allows processes running on the master server to use YP services. Type:

   ```
   /etc/chkconfig yp on
   ```

   ```
   /usr/etc/ypbind
   ```

8. Start *rpc.passwd* only on the master server. To start *rpc.passwd*, type:

   ```
   /etc/chkconfig ypmaster on
   ```

5.3.2 Altering a YP Client's Files to Use YP Services

Once you have decided to serve a database with the YP, it is best if all nodes in the network access the YP's version of the information, rather than the potentially out-of-date information in their local files. *ypbind* implements this policy by running a process on the client node. *ypbind* supplements or effectively replaces the contents of the following files: */etc/rpc*, */etc/bootparams*, */etc/passwd*, */etc/hosts*, */etc/ethers*, */etc/group*, */etc/networks*, */etc/protocols*, */etc/services*, */etc/netgroup*, and */usr/lib/aliases*. See *ypserv*(1M) for more information on *ypbind*.

/etc/hosts.equiv is a local file not served by the YP. However, entries in */etc/hosts.equiv* can reference information in YP database, for example the *netgroup* database. See *netgroups*(4).

You can edit */etc/hosts.equiv* to contain a single line, with only the plus character (+) on it. This allows anyone who has a user identification (ID) on any machine on the network to gain access to your machine.

Alternatively, you can exercise more control over logins by using lines of the form:

```
+@can_login_group1
+@can_login_group2
-@can't_login_group
```

Each of the names to the right of the at (@) character must be a *netgroup* name, defined in the global *netgroup* database. YP uses the *netgroup* database. The names to the right of the at (@) characters are called escape entries because they refer the system to another database, in this case, the *netgroup* database. If *host.equiv* does not use any of the escape entries, YP is not used.

YP does not normally use */.rhosts*. Its format is identical to that of */etc/hosts.equiv*. Because */.rhosts* controls remote *root* access to the local machine, you must list only those hosts that can log in as *root*, or use *netgroup* names for the same purpose. See *netgroup*(4).

/etc/hosts must contain the *localhost* name and its network number, 127.1. Your machine uses this entry to test the network by first attempting to communicate with itself. This entry is tested at boot time when the YP service is not yet available. If the system is using only YP to resolve addresses, after the system is running, and after the *ypbind* process has been started, the */etc/hosts* file is not accessed at all. See *resolver*(4). The example below shows the hosts */etc/file* for YP client *zippy*.

```
127.1           localhost       # special loopback address
192.9.1.87      zippy           # John Q. Random
```

/etc/passwd should contain entries for *root* and the primary users of the machine, and an escape entry to force the use of the YP service. Add an entry for *daemon*, shown in the example below, to the file to allow file-transfer utilities to work. A sample YP client's */etc/passwd* file looks like:

```
root:wAm0Y41Enf6:0:10:superuser:/:/bin/csh
jrandom:uHP1gQ2:1429:10:J Random:/usr2/jrandom:/bin/csh
daemon:*:1:1::/:
+::0:0:::
```

The last line informs the library routines to insert all entries from the YP password database into */etc/passwd* at that location. Entries that exist in */etc/passwd* mask analogous entries in the YP maps. Also, earlier entries in the file mask later ones with the same user name, or the same user ID.

You can limit the users who can use your file systems by selectively including individual YP entries by inserting password file entries with *+user* in the login name field. For example, to include just the entries for users *fred* and *barney* from YP, insert an entry like the one below into the client's *passwd* file.

```
+fred
+barney
```

You can also override fields of password records that are included from YP with values that differ on the local system. For example, the name of a user's home directory may be different on the local system. If user *fred* is to be included from YP, but given a different home directory, simply replace the home directory field of the passwd file entry as follows:

```
+fred:::::/d/people/fred:
```

In the above entry, YP will not allow the group and user IDs to be overridden with local values.

The local */etc/group* file interacts with YP in the same way that the */etc/passwd* file does, i.e., you can have both local and YP entries. To include the entire YP group database, put the following line in */etc/group*:

```
+:::
```

You can include individual YP group records in the same fashion as individual password file records. The following example contains both locally defined groups and records from the YP group database:

```
sys:*:0:
daemon:*:1:
staff:*:10:
+bin:::
+demos:::
+guest:::
```

You can */etc/group* to a single line:

```
+:
```

This entry forces all translation of group names and group IDs to be made via the YP service. This is the recommended procedure.

5.3.3 Setting Up a Slave YP Server

The network must be working to set up a slave YP server; in particular, you must be able to copy files using *rcp* from the master YP server to YP slaves. If the master server is running NFS 3.0 or later release, */usr/etc/ypserv* should also be running.

To create a new slave server, follow these steps:

1. Change directories to */usr/etc/yp*.

   ```
   cd  /usr/etc/yp
   ```

2. Become the superuser.

   ```
   su
   ```

3. Run *ypinit*(1M) with the –s switch, and name the master, a host already set up as a YP server. Ideally, the named host really is the master server, but it can be any host that has its YP database set up. You must be able to reach the host. Type:

   ypinit -s *master_server's_name*

4. Set the default domain name on the machine you intend to be the YP slave server. See *domainname*(1). You must set it to the same domain name as the default domain name on the machine named as the master. Also, you must have an entry for *daemon* in the */etc/passwd* files of both slave and master. That entry must precede any other entries that have the same user ID. Note the example shown in Section 4.3.2. You are not prompted for a list of other servers, but you can choose whether or not the procedure gives up at the first nonfatal error.

5. After running *ypinit*, make copies of */etc/passwd*, */etc/hosts*, */etc/group*, */etc/networks*, */etc/protocols*, */etc/netgroup*, and */etc/services*. For example, on the slave machine, type:

   ```
   cp  /etc/passwd  /etc/passwd-
   ```

   Edit the original files in accordance with Section 4.3.2, to ensure that processes on the slave YP server will actually make use of the YP services, rather than the local ASCII files, i.e., make sure the YP slave server is also a YP client. Make backup copies of the edited files, as well. Type:

   ```
   cp  /etc/passwd  /etc/passwd+
   ```

6. After *ypinit* sets up the YP database, use */usr/etc/ypserv* to begin supplying YP services. Type:

   ```
   /usr/etc/ypserv
   ```

7. If you want the *ypserv* process to be started automatically on subsequent reboots, type the command:

   ```
   /etc/chkconfig ypserv on
   ```

8. If you want the *ypbind* process to also start automatically on slave servers, type:

   ```
   /etc/chkconfig yp on
   ```

9. Run */usr/etc/rpc.passwd* only on the server that is the master server for the password database.

5.3.4 Setting Up a YP Client

To set up a YP client, edit the local files described in Sections 4.3.2 and 4.1.2. If */usr/etc/ypbind* is not running already, start it. Type:

```
/usr/etc/ypbind
```

With the ASCII databases of */etc* abbreviated and *ypbind* running, the processes on the machine are served by YP. At this point, there must be a YP server available. The programs that request YP services will hang if no YP server is available while *ypbind* is running.

Note the possible alterations to the client's */etc* database as discussed above in Section 4.3.2. Because some files may not be there, or some may be specially altered, it is not always obvious how YP uses the ASCII databases. The escape conventions used within those files to force inclusion and exclusion of data from the YP databases are found in the following manual pages: *passwd*(4), *hosts*(4), *netgroup*(4), *hosts.equiv*(4), *group*(4). In particular, notice that changing passwords in */etc/passwd* by editing the file, or by running *passwd*(1), only affects the local client's environment. Change the YP password database by running *yppasswd*. See *yppasswd*(1).

In order for the *ypbind* process to start automatically on subsequent reboots, type the following command on a YP client machine:

```
/etc/chkconfig yp on
```

As mentioned above, do not do this until there is a YP server serving the default domain of the client machines. See *resolver*(4) and the *TCP/IP User's Guide* for information on specifying the order of lookup services for hostname resolution.

5.3.5 Modifying Existing YP Maps after YP Installation

You must change databases served by YP on the master server. Look at the databases expected to change most frequently, like */etc/passwd*. You can change these files by first editing the ASCII file, and then running *make*(1) in */usr/etc/yp/Makefile* on the master server. Also see *ypmake*(1M) for more information.

You can manually modify nonstandard databases, i.e., databases that are specific to the applications of a particular vendor or site but are not part of the NFS release, databases that are expected to change rarely, or databases for which no ASCII form exists (for example, databases invented to use the YP). The general procedure is to use *makedbm*(1M) with the **–u** switch to disassemble them into a form that you can modify using standard tools (such as *awk*, *sed*, or *vi*), then build a new version again using *makedbm*(1M). There are two manual ways to modify nonstandard databases.

1. Redirect the output of *makedbm* to a temporary file which you can modify, then feed back into *makedbm*.
2. Operate on the output of *makedbm* within a pipeline that feeds into *makedbm* again directly. This is appropriate if you can update the disassembled map by modifying it with *awk*, *sed*, or a *cat* append, for instance.

Read the rest of this section only if you want to create nonstandard YP maps or add new categories of information to the YP.

Suppose you want to create a non-standard YP map, called *mymap*. To simplify this example, *mymap* will consist of key-value pairs in which the keys are strings like **al**, **bl**, **cl**, etc., and the values are **ar**, **br**, **cr**. (The **l** is for left and the **r** is for right.) There are two possible procedures to follow when creating new maps. In the first, you use an existing ASCII file as input; in the second, you use standard input.

For example, suppose there is an existing ASCII file named */usr/etc/yp/mymap.asc*, created with an editor or a shell script on your machine. In the example below, *home_domain* is the subdirectory where the map is located. Create the YP map for this file by typing:

```
cd  /usr/etc/yp
makedbm  mymap.asc  home_domain/mymap
```

But, at this point, you notice the map really should have included another record with fields **dl**, **dr**. In all situations like this, remember to modify the map by first modifying the ASCII file. Modifications that you make to the map that are not also made in the ASCII file will be lost. Modify the file by typing:

```
cd  /usr/etc/yp
<make  editorial  change  to  mymap.asc>
makedbm  mymap.asc  home_domain/mymap
```

When there is no original ASCII file, you can create the YP map from the keyboard. In this example, your machine is *ypmaster* and the default domain is *home_domain*. At *ypmaster*, type:

```
cd  /usr/etc/yp
makedbm  -  home_domain/mymap
al ar
bl br
cl cr
<ctrl-d>
```

When you need to modify that map, you can use *makedbm*(1M)
to create a temporary intermediate ASCII file
that you can edit using standard tools. Here is an example:

```
cd /usr/etc/yp
makedbm  -u  home_domain/mymap  >  mymap.temp
```

At this point, you can edit *mymap.temp* to contain the correct information. Create a new version of the database by typing:

```
makedbm  mymap.temp  home_domain/mymap
rm  mymap.temp
```

The preceding paragraphs explain how to use some tools to modify existing maps. You can use *ypinit*(1M) and */usr/etc/yp/Makefile* (see *ypmake*(1M)) to do almost everything necessary to modify a map, unless you add non-standard maps to the database, or change the set of YP servers after the system is already up and running.

Whether you use the Makefile in */usr/etc/yp* or some other procedure, the goal is the same: a new pair of well-formed *dbm* files must end up in the domain directory on the master YP server.

5.3.6 Propagation of a YP Map

To propagate a map means to move it from place to place. In this context, propagate means to move it from the master YP server to a slave YP server. Initially, *ypinit*(1M) moves it as described in Section 4.3.3. After you have initialized a slave YP server, *ypxfr*(1M) transfers updated maps from the master server. *ypxfr* contacts the master server and transfers the map only if the master's copy is more recent than the local copy. You can run *ypxfr* three different ways: periodically by *cron*(1M); by *ypserv*(1M); and interactively by a user. Below is an example of each. Note that maps have differing rates of change; for instance, *protocols.byname* may not change for months at a time, but *passwd.byname* may change several times a day in a large organization.

The first example of map propagation uses *cron*. The standard */usr/spool/cron/crontabs/root* has entries to periodically run *ypxfr* from shell scripts at a suggested rate for the various mappings in your YP database. You can find these shell scripts in */usr/etc/yp*: *ypxfr_1phr*, *ypxfr_1pd*, and *ypxfr_2pd*. These groupings transfer once per hour, once per day, and twice per day, respectively. These shell scripts are run at each YP server in the domain. If the rates of change are inappropriate for your environment, you can easily modify */usr/spool/cron/crontabs/root*. Also, you should alter the crontab to change the exact time of execution from one server to another to prevent the transfers from slowing down the master. Modify the shell script if the suggested groupings of the maps are inappropriate for your site. For more information on how to use *crontab*, see *crontab(1)*

If you want to transfer the map from some particular server, not the master, specify that (using *ypxfr*'s –h option) within the shell script. Also, you can check and transfer maps having unique change characteristics by explicitly invoking *ypxfr* within the *root* crontab.

Another example of map propagation uses *ypxfr*. Run *yppush* on the master YP server. It enumerates the YP map *ypserver* to generate a list of YP servers in your domain. *yppush* sends a transfer map request to each of the named YP servers. *ypserv* forks off a copy of *ypxfr*, invoking it with the –C flag, and passing it the information it needs to identify the map and to call back the initiating *yppush* process with a summary status.

The *ypserv* also invokes *ypxfr* when it attempts to emulate the behavior of *ypserv* from an earlier release, for instance, when an NFS 2.0 master *ypserv* communicates directly with the 4.0 *ypserv*.

In the cases mentioned above, you can capture *ypxfr*'s transfer attempts and the results in a log file. If */usr/etc/yp/ypxfr.log* exists, *ypxfr* appends results to it. No attempt to limit the log file is made; you are in charge of that. To turn off logging, remove the log file.

In a third case of map propagation, you can run *ypxfr* as a command. Typically, you run *ypxfr* only in exceptional situations. For example, *ypxfr* is used when setting up a temporary YP server to create a test environment, or when a YP server that has been out of service must be made consistent with the other servers quickly.

5.3.7 Making New YP Maps after YP Installation

Adding a new YP map entails getting copies of the map's *dbm* files into the domain directory on each of the YP servers in the domain. The actual mechanism has been described above in Section 4.3.6. This section describes only the work required to get the proper mechanisms in place so the propagation works correctly. You must set up files correctly on both the master and the slaves.

After deciding which YP server is the master of the map, modify */usr/etc/yp/Makefile* on the master server so that you can conveniently rebuild the map. Actual case-by-case modification is too varied to describe here, but typically *Makefile* filters each readable ASCII file for which a map is to be built, such as */etc/hosts*, through *awk*, *sed*, and/or *grep* to make two databases suitable for input to *makedbm*(1M). These databases are stored as */usr/etc/yp/domainname/mapname.pag* and */usr/etc/yp/domainname/mapname.dir*. Consult the existing *Makefile* as a source for programming examples. Make use of the mechanisms already in place in */usr/etc/yp/Makefile* when deciding how to create dependencies that *make*(1) recognizes; specifically, the use of *.time* files allows you to see when the *Makefile* was last run for the map.

Support on the YP slave servers for propagation of the new maps consists of appropriate entries in one of the *ypxfr* shell scripts mentioned in Section 4.3.6. To get an initial copy of the map, run *ypxfr* by hand on each of the slave servers. The map must be globally available before clients begin to access it. If the map is available from some but not all YP servers, client programs behave unpredictably.

5.3.8 Adding a New YP Server Not in the Original Set

To add a new YP slave server, start by modifying some of the YP server maps on the master YP server. If the new server is a host that has not been a YP server before, you must add the host's name to the map *ypservers* in the default domain. To add a new YP server not in the original set, follow these steps:

1. To add a server named *ypslave* to domain *home_domain*, go to *ypslave* and type:

   ```
   cd  /usr/etc/yp
   (makedbm  -u  home_domain/ypservers;\
   echo  ypslave  ypslave)|makedbm  -  tmpmap
   mv  tmpmap.dir  home_domain/ypservers.dir
   mv  tmpmap.pag  home_domain/ypservers.pag
   yppush  ypservers
   ```

   Note that some commands are displayed on two lines. You can type these as one long command (even if the line wraps on your screen), or you can escape the return and go to a new line with a backslash (\), as shown above. However, you cannot simply type in half the command, press return, and type the second half.

2. The host's address should be in *hosts.byname*, a map that contains all the host names within a network and the Internet addresses of these host names. Each YP map has a *mapname* that programs use. If it is not, edit */etc/hosts* and run *make*. In this case, first edit */etc/hosts*, then type:

   ```
   cd  /usr/etc/yp
   make  hosts
   ```

3. Set up the new slave YP server's databases by copying the databases from YP master server *ypmaster*. Remote log in to the new YP slave. Use *ypinit*(1M) by typing:

   ```
   cd  /usr/etc/yp
   ypinit  -s  ypmaster
   ```

4. Complete the steps described in Section 5.3.3.

5.3.9 Changing the Master Server

To change a map's master, first build the map at the new master. Because the old YP master's name occurs as a key-value pair in the existing map, you cannot use an existing copy at the new master or send a copy there with *ypxfr*. You must reassociate the key with the new master's name. If the map has an ASCII source file, it should be present in its current version at the new master. To remake the YP map (called *jokes.bypunchline* in this example) locally, follow these steps:

1. Go to the machine you want to use as the new master server and type:

```
cd  /usr/etc/yp
make  jokes.bypunchline
```

2. You must have set up */usr/etc/yp/Makefile* correctly for this to work; if it is not correctly set up, set it up now. See *ypmake*(1M). Also, this is a good time to go back to the old master (if it will remain a YP server) and edit */usr/etc/yp/Makefile* so that *jokes.bypunchline* is no longer made there, i.e., comment out the section of *oldmaster:/usr/etc/yp/Makefile* that made *jokes.bypunchline.*

3. If the map exists only as a *dbm* database, you can make it over on the new master by disassembling an existing copy (one from any YP server will do) and running the disassembled version back through *makedbm.* For example:

```
cd  /usr/etc/yp
ypcat  -k  jokes.bypunchline | makedbm\
- mydomain/jokes.bypunchline
```

4. After making the map on the new master, send a new copy of the map to the other slave YP servers. However, do not use *yppush* directly at this time. The other slaves will try to get new copies from the old master, rather than the new one.

   A typical method (you may find others) to send a new copy of the map to the other slave YP servers is to become the superuser on the old master server. Type:

```
su
/usr/etc/yp/ypxfr -h newmaster jokes.bypunchline
```

5. Now you have a new copy on the old master, and can run *yppush*, which forces the propagation of a changed YP map. Until you run *yppush*, the remaining slave servers still believe that the old master is the current master, and 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.

6. If the method above fails, there is another option. Log in to each YP server machine as the superuser and execute the command shown above. This certainly works, but should be considered the worst case solution.

5.4 Debugging a YP Client

This debugging section is divided into two parts: problems seen on a YP client and problems seen on a YP server.

Before trying to debug a YP client, read Sections 4.1 and 3.2 that discuss how YP and NFS servers work.

5.4.1 On Client: Commands Hang

The most common problem at a YP client node is for a command to hang and generate console messages such as:

```
yp: server not responding for domain <domain_name>.
Still trying
```

Sometimes many commands begin to hang, even though the system as a whole seems to be working 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 *domain_name*. This often happens when machines that run *ypserv* have crashed. It may also occur if the network or the YP server machine is so overloaded that *ypserv* cannot get a response back to your *ypbind* within the timeout period. Under these circumstances, all the other YP client nodes on your network show the same or similar problems. The condition is temporary in most cases, and the messages usually go away when the YP server machine reboots and *ypserv* is restarted, or when the load on the YP server nodes and/or the Ethernet decreases. The YP servers are set up to dynamically switch users to another server if your initial server is either overloaded or hung.

However, in some circumstances, the situation will not improve without intervention. The circumstances described below are coupled with their solutions.

- The YP client has not set, or has incorrectly set, *domainname* on the machine. Clients must use a domain name that the YP servers recognize. Use *domainname*(1) to see the client domain name. Compare that with the domain name set on the YP servers. The domain name should be set in */usr/etc/yp/ypdomain*. If this file contains the wrong domain name, follow these steps:

  1. Become the superuser on the machine in question.
  2. Edit */usr/etc/yp/ypdomain* to fix the domain name. This assures that domain name is correct every time the machine boots.
  3. Set *domainname* by hand so it is fixed immediately. Type:

     domainname *good_domain_name*

- If your domain name is correct, make sure your local net has at least one YP server machine. You can bind to a *ypserv* process only on your local network, not on another accessible network. There must be at least one YP server for your machine's domain running on your local net. Two or more YP servers improve availability and response characteristics for YP services.

- If your local network has a YP server, make sure it is up and running. Check other machines on your local network. If several client machines have problems simultaneously, suspect a server problem. Follow these steps:

  1. Find a client machine that is behaving normally, and try the *ypwhich* command. If *ypwhich* does not return a response, type an interrupt character such as **<ctrl-c>**.
  2. Go to a terminal on the YP server machine. Look for *ypserv* and *ypbind* processes. Type:

     ```
     ps -de | grep yp
     ```

If the server's *ypbind* daemon is not running, start it up by typing:

```
/usr/etc/ypbind
```

If there is a */usr/etc/ypserv* process running, type *ypwhich* on the YP server machine. If *ypwhich* returns no answer, *ypserv* is probably not working. You must restart *ypserv*. Log on as *root*. Follow these steps:

1. Kill the existing *ypserv* process. Type:

   ```
   killall -v ypserv
   ```

2. If *killall* confirms that it killed *ypserv*, type:

   ```
   /usr/etc/ypserv
   ```

5.4.2 On Client: YP Service Unavailable

When other machines on the network appear to be functioning normally, but YP service is unavailable on your machine, your machine can display many symptoms. These are some of the symptoms:

- Some commands appear to operate correctly while others terminate, printing an error message about the unavailability of YP.
- Some commands work slowly in a backup-strategy mode particular to the program involved.
- Some commands or daemons crash with obscure messages or no message at all.

Here are two examples of commands and daemons crashing.

1. If you type this command:

   ```
   ypcat  myfile
   ```

The system displays this message:

```
ypcat:  can't  bind  to  YP  server  for  domain  domain_name.
                 Reason: can't communicate with ypbind.
```

2. If you type this command:

```
/usr/etc/yp/yppoll  myfile
```

   The system displays this message:

```
Sorry, I can't make use of the Yellow  Pages. I give up.
```

These symptoms usually indicate that your *ypbind* process is not running. Follow these steps:

1. Type the following to check for a *ypbind* process.

```
ps -de
```

2. If you do not find it, start the *ypbind* process. Type:

```
/usr/etc/ypbind
```

Typing */usr/etc/ypbind* will make these YP problems disappear.

5.4.3 On Client: *ypbind* Crashes

If */usr/etc/ypbind* crashes almost immediately each time it is started, look for a problem in some other part of your machine. Check for the presence of the *portmap* daemon by typing:

```
ps -de | grep portmap
```

If you do not find it running, reboot your machine.

If the *portmap* daemon itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software.

You can start up the *portmap* daemon on your machine from a machine that is operating normally. From such a machine, type:

```
rpcinfo -p your_machine_name
```

If your *portmap* is functional, the output should look like:

```
program version  proto  port

100005    1       udp    1026  mountd
100001    1       udp    1028  rstatd
100001    2       udp    1028  rstatd
100003    2       udp    2049  nfs
100008    1       udp    1033  walld
100002    1       udp    1035  rusersd
100012    1       udp    1037  sprayd
100007    2       tcp    1026  ypbind
100007    2       udp    1046  ypbind
100007    1       tcp    1026  ypbind
100004    2       udp    1051  ypserv
100007    1       udp    1046  ypbind
100004    2       tcp    1027  ypserv
100004    1       udp    1051  ypserv
100004    1       tcp    1027  ypserv
```

The port numbers will be different on your machine. The four entries that represent the *ypbind* process are:

```
100007    2       tcp    1026  ypbind
100007    2       udp    1046  ypbind
100007    1       tcp    1026  ypbind
100007    1       udp    1046  ypbind
```

If these entries are not there, *ypbind* has been unable to register its services with the *portmap* daemon. Reboot the machine. If they are there and they change each time you try to restart */usr/etc/ypbind*, reboot the system, even if the *portmap* daemon is up. If the situation persists after reboot, call your service organization.

5.4.4 On Client: *ypwhich* Inconsistent

When you use *ypwhich* several times at the same client node, the answer you receive varies because the YP server has changed. This is normal. The binding of YP client to YP server changes over time on a busy network and when the YP servers are busy. Whenever possible, the system stabilizes at a point where all clients get acceptable response time from the YP servers. As long as your client machine gets YP service, it does not matter where the service comes from. Often a YP server machine gets its own YP services from another YP server on the network.

5.5 Debugging a YP Server

Before trying to debug a YP server, read Sections 5.4.1 and 5.4.2 in this manual.

5.5.1 On Server: Different Versions of a YP Map

Since YP works by propagating maps among servers, you may find different versions of a map at servers on the network. This version skew is normal if it is transient, and abnormal otherwise.

The normal update of YP maps is prevented when some YP server or some gateway machine between YP servers is down during a map transfer attempt. This is the most frequent cause of different versions of a map on servers on a network. Normal update procedures are described in Section 4.3.6. When all the YP servers and all the gateways between them are up and running, *ypxfr* should succeed.

If a particular slave server has problems updating, log in to that server and run *ypxfr* interactively. If *ypxfr* fails, it tells you why it failed, and you can fix the problem. If *ypxfr* appears to succeed, but you believe it fails intermittently, follow these steps:

1. Log in to the slave server in question.

2. Create a log file to enable message logging. Type:

   ```
   cd  /usr/etc/yp
   touch  ypxfr.log
   ```

   This saves all output from *ypxfr*. The output looks much like the output from *ypxfr* when run interactively, but each line in the log file is timestamped. You may see unusual orderings in the timestamps. This is normal; the timestamp tells you when *ypxfr* began its work. If copies of *ypxfr* ran simultaneously, but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from the order of invocation.

Any pattern of intermittent failure shows up in the log. Look at the messages and try to determine what is needed to fix the failure. You know that you have fixed it when you no longer receive failure messages.

3. When you have fixed the problem, turn off message logging by removing the log file. Type:

   ```
   rm  ypxfr.log
   ```

   If you forget to remove the log file, the log file grows without limit.

4. Inspect */usr/spool/cron/crontabs/root* and the *ypxfr* shell scripts it invokes. Typos in these files cause propagation problems, as do failures to refer to a shell script within *crontab*, or failures to refer to a map within any shell script. Also make sure the *yp* and *ypserv* configuration flags are on.

5. Check that the YP slave server is in the map *ypservers* within the domain. If the slave server is not in the map, it still works fine as a server, but *yppush* will not tell *ypserv* when a new copy of a map exists.

6. If the problem is not obvious, you can work around it while you debug by using the file copy command, *rcp*(1), to copy a recent version from any healthy YP server. You may not be able to do this as *root*, but you probably can to do it by using the guest account on the master server. For instance, to transfer the map *busted* from the master server to the slave server, type the following from the slave server machine, *ypslave*.

   ```
   rcp  guest@ypmaster:/usr/etc/yp/mydomain/busted.\* \
   /usr/etc/yp/mydomain
   ```

   Notice that the asterisk (*) has been escaped in the command line, so that it will be expanded on *ypmaster*, instead of locally on *ypslave*.

5.5.2 On Server: *ypserv* Crashes

When the *ypserv* process crashes almost immediately, and does not stay up even with repeated activations, you must debug like you did in Section 4.4.3. Follow these steps:

1. Check for the *portmap* daemon:

```
ps -de | grep portmap
```

2. Reboot the server if you do not find the *portmap* daemon. If it is there, type:

```
/usr/etc/rpcinfo -p
```

Look for output to the screen similar to:

```
program version  proto  port
100005    1      udp    1026  mountd
100001    1      udp    1028  rstatd
100001    2      udp    1028  rstatd
100003    2      udp    2049  nfs
100008    1      udp    1033  walld
100002    1      udp    1035  rusersd
100012    1      udp    1037  sprayd
100007    2      tcp    1026  ypbind
100007    2      udp    1046  ypbind
100007    1      tcp    1026  ypbind
100004    2      udp    1051  ypserv
100007    1      udp    1046  ypbind
100004    2      tcp    1027  ypserv
100004    1      udp    1051  ypserv
100004    1      tcp    1027  ypserv
```

On your machine, the port numbers will be different. The four entries that represent the *ypserv* process are:

```
100004    2      udp    1051  ypserv
100004    2      tcp    1027  ypserv
100004    1      udp    1051  ypserv
100004    1      tcp    1027  ypserv
```

3. If these four entries are not present, *ypserv* has been unable to register its services with the *portmap* daemon. Reboot the machine.

4. If these entries present, and they change each time you try to restart */usr/etc/ypserv*, reboot the machine. If the situation persists after you reboot, call the Geometry Hotline for help.

5.6 YP Policies

When the C library routines access the following files on a system running the YP, they set some policies:

/etc/passwd	Always consulted. If there are + or – entries, the C library routines consult the YP password map. Otherwise, YP is unused.
/usr/lib/aliases	Always consulted. If there is a +:+ entry, the C library routines consult the YP aliases map. Otherwise, YP is unused.
/etc/group	Always consulted. If there are + or – entries, the C library routines consult the YP group map. Otherwise YP is unused.
/etc/hosts.equiv	(and similarly for *.rhosts*) Always consulted, though neither of these files is in the YP database. (See Section 4.7 for a more complete explanation of these two files.) If there are + or – entries, whose arguments are netgroups, the YP netgroup map is consulted; otherwise YP is unused.
/etc/services	Never consulted. The data that was formerly read from this file now comes from the YP *services* database.
/etc/protocols	Never consulted. The data that was formerly read from this file now comes from the YP *protocols* database.
/etc/networks	Never consulted. The data that was formerly read from this file now comes from the YP *networks* database.
/etc/netgroup	Never consulted. The data that was formerly read from this file now comes from the YP *netgroup* database.
/etc/ethers	Never consulted. The data read from this file comes from the YP *netgroup* database.

/etc/hosts	Consulted only when booting (by the *ifconfig* command in the */etc/init.d/tcp* file). After that the YP is used instead.
/etc/rpc	Never consulted. The data that was formerly read from this file now comes from the YP rpc.bynumber database.
/etc/bootparams	Consulted only when the lookup in the YP *bootparams* database fails.

More information on these configuration files is included in the *IRIS-4D Owner's Guide* and the *TCP/IP User's Guide.*

5.7 Changing Security with the YP

Read Section 4.6 on YP accessing policies to better understand YP security issues. The sections below describe specific issues in network security using the YP.

5.7.1 Global and Local YP Database Files

Of the YP databases, ten originate from */etc*: */etc/passwd, /etc/group, /etc/hosts, /etc/networks, /etc/services, /etc/protocols, /etc/rpc, /etc/bootparams, /etc/netgroup*, and */etc/ethers*. One more resides in */usr/lib*: */usr/lib/aliases*. Note that a site may add database files of its own. YP consists of local and global file types. Local files are those files that YP first checks for on your own machine, then in the YP. Global files are those files that programs using YP check for only in the YP database. */etc/passwd* and */etc/group* are the local files in the YP database. The other five YP files are global.

For example, a program that calls *getpwent* to access */etc/passwd* (a local file) first looks in the password file on your machine; the YP password file is only consulted if your machine's password file contains a plus sign (+) entry. The */etc/passwd* file is local, so that you can control the entries for your own machine. The only other local file is */etc/group*. To repeat, YP consults local files first on your own machine, and only consults global files if these files contain plus (+) entries referring to YP.

The remaining YP files (*hosts*, *networks*, *ethers*, *services*, *rpc*, *bootparams*, *protocols*, and *netgroup*) are global files. The information in these files is network-wide data, and only YP accesses it. However, you must make sure each machine has an entry in */etc/hosts* for itself when booting. In summary, if YP is running, YP only checks global files; YP does not consult a file on your local machine.

5.7.2 Two Other Files YP Consults

The files */etc/hosts.equiv* and */.rhosts* are not in the YP database. Each machine has its own unique copy. However, you can put entries in your */etc/hosts.equiv* file that refer to the YP. For example, the following line includes all members of *engineering* as it is defined in the local file */etc/netgroup* or in the YP database.

```
+@engineering
```

A line consisting only of a plus sign (+) includes everyone in your */etc/hosts.equiv* file.

5.7.3 Security Implications

To be able to log in to a machine without having a password, you need to be in both the */etc/hosts.equiv* file and the */etc/passwd* file. A plus (+) entry in */etc/hosts.equiv* allows you to effectively bypass this check. Anyone in your */etc/passwd* file is allowed to *rlogin* to your machine without restriction.

The */etc/passwd* file and */etc/group* file may also have plus (+) entries. The following line in an */etc/passwd* file pulls in an entry for *nb* from the YP.

```
+nb:::Napoleon Bonaparte:/usr2/nb:/bin/csh
```

It gets the user ID, group ID, and password from the YP, and gets the user's name, home directory, and default shell from the plus entry (+) itself. On the other hand, the following */etc/passwd* entry gets all information from the YP.

```
+nb:
```

Here are two types of entries in */etc/passwd.*

```
+nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

```
nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

In the first entry, YP obtains the password field, which is stored by the plus (+) entry. In the second entry, user *nb* has no password. Also, if there is no entry for *nb* in YP, then the effect of the first example is as if no entry for *nb* was present at all, i.e., the entry points to something that does not exist.

5.7.4 Special YP Password Change

When you change your password with the *passwd*(1) command, you change the entry explicitly given in your own */etc/passwd* file. If your password is not given explicitly, but rather is pulled in from the YP with a plus (+) entry, then the *passwd* command prints the error message:

```
Not in passwd file
```

To change your *passwd* in the YP, use the *yppasswd*(1) command. To enable this service, the system administrator must start up the daemon *rpc.passwd*(1M) server on the machine serving as the master for the YP password file.

5.7.5 Netgroups: Networkwide Groups of Machines and Users

The */etc/netgroup* file on the master YP server defines the *netgroup*(4). A *netgroup* is a networkwide group of machines and users. Login, remote mount, remote login, and remote shell use these groups for permission checking.

The master YP server uses */etc/netgroup* to generate three YP maps in the */usr/etc/yp/domainname* directory: *netgroup*, *netgroup.byuser*, and *netgroup.byhost.* The YP map *netgroup* contains the basic information in */etc/netgroup*. The two other YP maps contain a more specific form of the information to speed the lookup of *netgroups* given the host or user.

The programs that consult the YP maps are *login*(1), *mountd*(1M), *rlogin*(1), and *rsh*(1). *login* consults them for user classifications if it encounters *netgroup* names in */etc/passwd*(4). *mountd* consults them for machine classifications if it encounters *netgroup* names in */etc/exports*(1M). *rlogin* and *rsh* consult the *netgroup* map for both machine and user classifications if they encounter *netgroup* names in */etc/hosts.equiv*(4) or *.rhosts*(4).

Here is a sample */etc/netgroup* file. See *netgroup*(4) for a description of file format and definition of lines and fields.

```
#  Engineering: Everyone but eric has a machine;
#  he has no machine.
#  The machine 'testing' is used by all of hardware.
#
engineering  hardware  software
hardware    (mercury,alan,sgi)  (venus,beth,sgi)  (testing,-,sgi)
software    (earth,chris,sgi)  (mars,deborah,sgi)  (-,eric,sgi)
#
#  Marketing:  Time-sharing  on  jupiter
#
marketing   (jupiter,fran,sgi)  (jupiter,greg,sgi)  (jupiter,dan,sgi)
#
#  Others
#
allusers   (-,,sgi)
allhosts   (,-,sgi)
```

Based on the above, the users are classified into groups as follows:

Group	Users
hardware	alan, beth
software	chris, deborah, eric
engineering	alan, beth, chris, deborah, eric
marketing	fran, greg, dan
allusers	(every user in the YP map passwd)
allhosts	(no users)

Table 5-2. Example User Groups

Here is how the machines would be classified:

Group	Hosts
hardware	mercury, venus, testing
software	earth, mars
engineering	mercury, venus, earth, mars, testing
marketing	jupiter
allusers	(no hosts)
allhosts	(all hosts in the YP map hosts)

Table 5-3. Example Machine Groups

For more details, see the following manual pages: *yppasswd*(1), *hosts.equiv*(4), *export*(4), *passwd*(4), *group*(4), *netgroup*(4), and *rpc.passwd*(1M).

5.8 Adding a New User to a Machine

To add a new user to a machine using YP, add an entry to the password file and create a home directory on the new user's machine as described in the steps below.

5.8.1 Edit the Master YP Server's */etc/passwd* File

Add a password file entry to every machine on the local network for a new user. If all machines are YP clients, then all you need to do is add an entry for the new server in the YP password maps. To add a new entry, follow these steps:

1. Become the superuser.

```
su
```

2. Edit the master YP server's */etc/passwd* file. Add a new line to the password file with the text editor of your choice. */etc/passwd* is a readable ASCII file with a one line entry for each valid user on the system. Separate each entry into fields by colons (:). There are seven fields on each line. You can leave some fields blank by placing two colons (::) back to back. Avoid using the characters for single and double quotes (' ' "), backslashes (\), and parentheses ( ) in the password file. See *passwd*(4) for more information about the file format.

   For example, suppose that your new user's name is Mr. Chimp and his account is going to be *bonzo*. Add a line similar to the one shown below.

   ```
   bonzo::1947:10:Mr. Chimp:/usr2/bonzo:/bin/csh
   ```

   Notice that the two colons cause the second field to read as blank in the example. This field, when filled, contains an encrypted version of the user's password. However, when the field is blank, anyone can log in simply by typing the user name — no password is required. You cannot create a password by making an entry in the */etc/passwd* file. You must use *passwd*(1) while logged in as the user or as the superuser. Since anyone can log in when a user has no password, provide a password for the new user and let him know it so he can log in and change it to whatever he prefers. When Mr. Chimp logs in for the first time, he can use the *passwd* utility to change his password, or *yppasswd*(1) to change it in the YP database.

   After Mr. Chimp has a password, the entry for *bonzo* in the password file will look something like:

   ```
   bonzo:3u0mRdrJ4tEVs:1947:10:Mr.  Chimp:/usr2/bonzo:/bin/csh
   ```

   Fields in the password file have the following meanings:

Login name	This is synonymous with the user name.
Encrypted password	*passwd*(1) creates this entry. The system administrator tells all new users how to add, or change, their password with the *passwd* command and the *yppasswd* command. The system administrator can clear this field when a user has forgotten his or her password, thereby enabling login without a

	password until the user creates a new one. Note that an asterisk (*) in this field matches no password. The user can log in only if the machine name of the machine he is logging in from is in the */etc/hosts.equiv* file on the local machine.
User ID	This entry is a number unique to this user. In the example above, this number is 1947. A system knows the user by ID number associated with login name, therefore a login name must have the same user ID number on all password files of machines that are networked in a local domain.
	Failure to keep IDs unique prevents moving files between directories on different machines because the system will respond as if the directories are owned by two different users. Also, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with user's IDs that match those of different users on the NFS server.
Group ID	Use this entry to group together users who are working on similar projects. In this example, Mr. Chimp is in group 10, the system staff group. If you are not sure into which group you should put a new client, see *group*(4) and look in the file */etc/group*.
Information about the user	This entry is usually the real name, phone number, etc. An ampersand (&) here is shorthand for the user's login name.
User's home directory	This entry is the directory into which the user logs in.

Initial shell to use on login	If this field is blank, the default */bin/sh* is used. It is recommended that you place */bin/csh* here, as in the example above. It gives a C shell as the user's initial shell.

After you update the password file and create a password for the new user, update the YP database by running */usr/etc/yp/Makefile* for */etc/passwd*. Type:

```
cd  /usr/etc/yp
make  passwd
```

Now the new entry you created for Mr. Chimp in */etc/passwd* has been added to the YP *passwd* maps.

5.8.2 Make a Home Directory

After adding a new entry to the password file, create a home directory for the new user. This is the same as the directory given in the sixth field of the password file entry. In the */usr2* directory in the example below, make a directory for the new user, and change ownership using the *chown* command to the user's login name and change group to the user's group. Type:

```
cd /usr2
mkdir bonzo
chown bonzo bonzo
chgrp 10 bonzo
```

Note that if the YP databases for the password file have not yet been updated on the machine's YP server, you get the following error message when you attempt to use *chown*:

```
unknown user id: username
```

In that case, you can use the following set of commands:

```
cd  /usr2
mkdir bonzo
chown userid# bonzo
chgrp 10 bonzo
```

Use Mr. Chimp's user ID number (from the password file entry) instead of login name to change the ownership of his home directory.

5.8.3 The New User's Environment

Finally, you can define the new user's environment on login in several ways. For example, you can give the new user a copy of such files as *.login* and *.cshrc* if they use */bin/csh*, or *.profile* if they use *bin/sh*.

If the new user is a member of any groups at your site, add them to */etc/group* as necessary. See *group*(4). Make the changes to the */etc/group* and */etc/netgroup* files on the master YP server if you run the YP.

Index