CONVEX Network File System
System Manager's Guide

Document No. 710-001630-202

Second Edition, Rev. 1
April 1988

CONVEX Computer Corporation
Richardson, Texas

CONVEX Network File System
System Manager's Guide
Order No. DSW-113
Second Edition, Rev. 1

Printed in the United States of America

Revision Information for
CONVEX Network File System
System Manager's Guide

Edition	Document No.	Description
Second Rev. 1	710-001630-202	Released with CONVEX UNIX V6.2, April 1988. Includes the following changes: Chapter 2 corrected "Starting and Killing Daemons" section corrected Figure 2-1 added new information on */etc/rc.local* Appendix A, added "Updating */etc/rc.local*" Appendix B, updated sample contact session
2.0	710-001630-201	Released with CONVEX UNIX V6.1, October 1987. Includes the following changes: Chapter 2 added description of */etc/exports* file and use of over-the-net root access added description of advisory-record locking and lock manager added description of remote-execution utilities and daemon added description of named pipes revised subsections "File Operations Not Supported" and "Access to Remote Devices" added description new *inet* daemon Chapter 3 added description of *yp sendmail* support
1.0	710-000730-000	Initial release with CONVEX UNIX V6.0, April 1987.

Table of Contents

Appendices

List of Tables

List of Figures

Preface

Purpose and Audience

This document provides the information needed to install, maintain, and debug the Network File System (*nfs*), its related utilities, and the yellow pages (*yp*) database software.

This document addresses system managers or operators who have the following experience:

- Experience with the UNIX operating system
- Experience as a system manager or operator
- Experience as a manager of a CONVEX supercomputer
- Familiarity with the *CONVEX System Manager's Guide*

Previous experience with *nfs* and the yellow pages (*yp*) are helpful, but not required to use this document.

Organization

This document is organized into three chapters and two appendices, as follows:

- Chapter 1 introduces relevant terms and concepts.
- Chapter 2 describes how to install, maintain, and debug *nfs*. This chapter also discusses how to ensure the security of networked file systems.
- Chapter 3 describes how to install and maintain the yellow pages database software and how to add a new user to the *yp* environment.
- Appendix A illustrates a sample */etc/rc.local* file.
- Appendix B describes how to use the *contact* utility to report problems with software or documentation.

Notational Conventions

The following conventions are used in this document:

- Mnemonics enclosed in "less than" and "greater than" signs designate ASCII nonprintable characters. For example, **<CR>** stands for "carriage return."
- Within command sequences set off from regular text, **boldface** type indicates literals. Words appearing in **boldface** must be typed just as they appear. *Italics* within command sequences indicate generic commands or filenames. Substitute actual commands or filenames for the *italicized* words. For example, the command sequence

  ld [*switches*] [*object files*] [*libraries*]

instructs you to type the command *ld*, followed by your choice of switches, object files and libraries.

Italics within text indicate commands, filenames, or programs.

- Brackets [ ] designate optional entries.
- A horizontal ellipsis ... shows repetition of the preceding item(s).
- A vertical ellipsis shows continuation of a sequence where not all of the statements in an example are shown.
- Commands, utilities, and files that are documented in the *CONVEX UNIX Programmer's Manual* are italicized; occurrences that include a number enclosed in parentheses refer to the appropriate section of the manual (for example, *vmstat*(1) means that the command *vmstat* is located in Section 1 of the *Programmer's Manual*).
- The | symbol is used to denote command sequences in which you must pick no more than one alternative from a list of command options. In the following command sequence, for example:

  `(fp)>` **s[et] s[pu-selftest]** = [*d[isable* |*e[nable]*

  you must choose either *d[isable]* or *e[nable]*, but you cannot choose both.
- The pound sign (#) signifies the superuser prompt. The percent symbol (%) signifies the standard C shell user prompt.

Associated Documents

The following documents provide useful information as you learn about the Network File System and its related utilities.

- *CONVEX Network File System User's Guide.* This document describes how to use *nfs*, *rpc*, *xdr*, and *yp*.
- *CONVEX Network File System Reference Set.* This volume contains six reference manuals describing RPC programming and the *nfs*, *rpc*, and *xdr* protocols.
- *CONVEX System Manager's Guide.* This document provides the information needed to manage and maintain a previously installed and configured CONVEX supercomputer.
- *CONVEX Processor Operation Guide.* This document describes how to start up and shut down the system, boot the SPU UNIX operating system, and boot the CONVEX UNIX operating system.

Ordering Documentation

To order the current edition of this or any other CONVEX document, you need to know the exact title or the six-character order number. See the copyright page of this document for its six-character order number. To find the order numbers for other CONVEX documents, see the *CONVEX COMPUTER Price Book* or call the Technical Assistance Center or your local CONVEX office.

To order an edition other than the current edition, you need to know the 12-digit part number, which is also called the document number. See the title page of this document for its 12-digit document number. To find the document numbers for other CONVEX documents, call the Technical Assistance Center or your local CONVEX office.

To order CONVEX documentation, send requests to

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson, TX 75083-3851 USA

Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC). Use the phone numbers in the following table.

Location	Phone Number
In Texas	(214)952-0379
Other continental locations	1(800)952-0379
Outside continental U.S.	Contact local CONVEX office

1

Introduction and Terminology

This chapter provides conceptual background for system managers interested in learning about the Network File System (*nfs*) and the yellow pages (*yp*), two networking services provided as optional parts of the CONVEX UNIX operating system. The chapter consists of four parts:

- A discussion of the networking model used as a basis for the design of *nfs* and *yp*.
- Definitions of the terms used to discuss the network services throughout the manual.
- A discussion of how network services interact with the standard UNIX operating system.
- Some general hints on debugging networks in a UNIX environment.

While some of this material tends to be theoretical, its specific implications is seen again and again as you become familiar with system administration. For example, if you run the yellow pages, you must understand that some typical UNIX procedures have changed in the yellow pages environment. That is also true if you use the the network file system. This chapter covers only those aspects of network services necessary for performing the duties of system administration.

Networking Models

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

A distributed operating system allows the network software designer to make grand assumptions about the other machines on the network. Usually two of these assumptions are: that the remote piece of hardware is identical to the local hardware, and that the remote and local machines are running identical software. 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. That is to say, it is difficult to integrate new hardware or software into a closed network environment, 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, a system based on network services 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. Thus, the procedure (and therefore, the network service) is not tied to any particular operating system or hardware. The network services approach makes it possible for a variety of machines and software to talk to the network services, enabling the CONVEX supercomputer to talk to various types of computers.

As you might expect, a network services approach is more complex in design and implementation than a closed distributed operating system. It is also less efficient than a closed system. Since remote procedures are operating-system independent and hardware independent, multiple remote procedures must sometimes be called in the network services environment, where a single transaction might suffice in a closed system.

Terminology

The bulk of this chapter explains administration of two network services — the Network File System (*nfs*) and the yellow pages (*yp*). Before discussing these services, we must define some generic concepts and terms.

A *server* is any machine that provides a network service. A single machine may provide both *nfs* and *yp* services. A typical configuration would be for one machine to act as an *nfs*-server and a *yp*-server. In each of the network services, servers are entirely passive. The servers wait for clients to call them; they never call the clients.

A *client* is any entity that accesses a network service. We use the term entity because the thing doing the accessing may be either a machine or simply a UNIX process generated by a piece of software.

The degree to which clients are bound to their server varies with each of the network services. For example, a *yp* client binds randomly to a *yp* server by broadcasting a request. At any point, the *yp* client may decide to broadcast for a new server. Another example is *nfs*. An *nfs* client may choose to mount file systems from any server at any time.

The client, however, starts 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.

UNIX Meets Network Services

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

— The UNIX operating system 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 "over the net." For example, trusting user ID 0 (root) is not always a good idea.

— Some UNIX execution semantics are difficult. For example, the UNIX operating system 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.

— 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 all its bound neighbors down. The treatment of node failure on a network raises difficulties in any system and is especially difficult in the UNIX environment. A system of "stateless" protocols has been implemented, however, to circumvent the problem of a crashing server dragging down its bound clients. "Stateless" here 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 slow server.

In this implementation over the network, every attempt was made to remain compatible with the UNIX operating system. Certain incompatibilities have been introduced, however; they are typically of two kinds. First, those issues that would evolve a network UNIX operating system into a distributed operating system, rather than a collection of network services. And second,

those issues that would make crash recovery extremely difficult from the implementation and administration point of view. All incompatibilities are documented in the appropriate sections of this administration manual.

A Hint About Debugging UNIX in the Network Environment

When you cannot get something done that involves the network services, the problem probably lies in one of the following four areas. They are listed here with the most likely problem first.

1. The 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.

The following sections present specific instructions on how to check for these causes of failure in the *nfs* and *yp* environments.

2

Installing and Debugging *nfs*

This chapter discusses the basics of *nfs*, describes debugging procedures, and explains how to tune the system.

The early sections of the chapter define *nfs*, describe how it works, and explain how to install *nfs* on servers and clients and how to mount a file system remotely. Various aspects of system debugging are discussed next. The chapter closes with a discussion of tuning issues (how to enable asynchronous operation, various security measures, etc.) and a description of incompatibilities with the standard UNIX operating system.

For more information about how to use *nfs*, see the *CONVEX NFS User's Guide*. (The *User's Guide* also discusses use of *rpc*, *xdr*, and *yp*.)

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 starts the binding to a server's file system by using the *mount*(8) command. Typically, a client mounts one or more remote file systems at start-up time by placing lines like these in the file */etc/fstab*, that *mount* reads when the system comes up:

```
titan:/usr2 /usr2 nfs rw,hard 0 0
venus:/usr/man /usr/man nfs rw,soft 0 0
cyclops: /usr/docs /mnt nfs rw,intr,bg 0 0
```

See *fstab*(5) for a full description of the format.

Because clients start all remote mounts, *nfs* servers keep control over who may mount a file system by limiting named file systems to desired clients with an entry in the */etc/exports* file. For example:

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

is a self-explanatory */etc/exports* entry. Note that pathnames given in */etc/exports* must be the mount point of a local file system. See *exports*(5) for a full description of the format.

How *nfs* Works

Two remote programs implement *nfs* service—*mountd*(8) and *nfsd*(8). A client's *mount* request talks to *mountd*, that checks the access permission of the client and returns a pointer to a file system. After the *mount* completes, access to that mount point and below goes through the pointer to the server's *nfsd* daemon using *rpc*(3N). Client kernel file access requests (delayed-write and read-ahead) are handled by the *biod*(8) daemons on the client.

Although you can install and administer *nfs* just by considering the information given here, you can find more details in the *CONVEX Network File System Reference Set*.

How to Become an *nfs* Server

An *nfs* server is simply a machine that exports a file system or systems. To set up an *nfs* server, use the following procedure:

1. Install *nfs* software according to the directions included in the release notes distributed with the product.

2. Become superuser and place the mount-point pathname of the file system you want to export in the file */etc/exports*. See *exports*(5) for file format details. For example, to export */usr/src/mybin*, the export file would look like:

```
/usr/src/mybin
```

   Of course, an *nfs* server may only export file systems of its own.

3. Check */etc/rc.local* to make sure that */etc/portmap* has been included. Look for lines like these:

```
if [ -f /etc/portmap ]; then
        /etc/portmap & echo -n 'portmap'        >/dev/console
fi
```

   If you do not find these lines in the file, then add them immediately after you configure the network with *ifconfig*.

   NOTE

   You must edit the */etc/rc.local* startup file to become an *nfs* or yellow pages (*yp*) server. To verify the daemons that must be started and the order in which they must be started, refer to Appendix B, "Updating */etc/rc.local*."

4. As we saw above, *mountd* must be present for a remote mount to succeed. Make sure *mountd* is available for an *rpc* call by checking the file */etc/inetd.conf*, on the *nfs* server, for this line:

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

   If it isn't there, add it. For details, see *inetd.conf*(5). The configuration file is described in more detail in Chapter 4 of the *RPC Programmer's Manual*.

5. Check to make sure that */etc/portmap* is running. A good way to do this is to use *grep* as follows:

```
# ps ax | grep portmap
```

   If */etc/portmap* is not running, use the following command sequence to start *portmap* and reconfigure */etc/inetd*:

```
# /etc/portmap
# ps ax | grep inetd
# kill -1 [process number found using the previous instruction]
```

6. Remote mount also needs some number (typically 4) of *nfsd*'s to execute on *nfs* servers. Check */etc/rc.local* for lines like these:

```
if [ -f /etc/nfsd ]; then
        /etc/nfsd 4 & echo -n ' nfsd'                    >/dev/console
fi
```

   Add these lines, or your own version of them, if the new *nfs* server's */etc/rc.local* does not enable *nfsd*'s. You can enable *nfsd*'s without rebooting, by typing, while superuser:

```
# /etc/nfsd 4
```

After these steps, the *nfs* server should be able to export the named file system. Read the next section and try to remote mount.

How to Become an *nfs* Client

An *nfs* client is a machine that accesses networked file systems. To set up an *nfs* client, use the following procedure:

1. Install *nfs* software according to the instructions in the release notices distributed with the product.

2. Check */etc/rc.local* for the lines needed to invoke */etc/biod*. The necessary lines are:

```
if [ -f /etc/biod ]; then
        /etc/biod 4 & echo -n 'biod'                     >/dev/console
fi
```

   If you can't find these lines in */etc/rc.local*, add them. To enable *biod* without rebooting, use:

```
# /etc/biod 4
```

NOTE

> You must edit the */etc/rc.local* startup file to become an *nfs* or yellow pages (*yp*) client. To verify the daemons that must be started and the order in which they must be started, refer to Appendix B, "Updating */etc/rc.local*."

3. Add the following mount lines to */etc/rc.local*:

```
/etc/umount -at nfs
/etc/mount -vat nfs    >/dev/console
```

You should now be able to mount networked file systems on the server. See the next section for an explanation of how to mount file systems remotely.

How to Mount a File System Remotely

You can mount any exported file system onto your machine, as long as you can reach its server over the network, and you are included in its */etc/exports* list for that file system. When you mount directories, however, make sure to enable read and execute permission for "others" (i.e., *drwxrwxr-x*). (You must do this for both *nfs* **and** 4.2, or "local" partitions.) If you don't, attempts to determine the current working directory (with *pwd*(1)) in the mounted file system may fail with the message:

pwd: getwd: can't open ..

Once you've correctly set access permissions, log in as superuser on the machine where you want to mount the file system and enter the following:

**mount** *mount_options server_name:/file_system /mount_point*

Note that you can use the *mount_options* flag to create hard, interruptible hard, and soft mounted partitions. The differences in mount option cause programs to react in different ways when they encounter problems with networks or servers.

Hard mounts are the default and cause all operations (system calls) to succeed regardless of the load or a crash by the server. Hard mounts don't acknowledge interrupts, nor do they ever time out. Instead, they retransmit over and over again until an acknowledgement is received from the server. Because hard mounts don't time out, and because no data is lost even when a server crashes, they are the option of choice for critical tasks and for writing to remote file systems. Note, though, that there are some disadvantages to using hard mounts: operations may take a long time to complete (since it usually takes at least 15 minutes to reboot a machine), and user's tasks may hang repeatedly if hard mounts are used on an unreliable remote server. (If the server fails to respond, you'll receive an *NFS server not responding* message on the console and at your terminal.)

Suppose, for example, you run *df* across a file system mounted on a server that has just crashed. You'll receive output only until a *statfs*(2) mount request is sent to the downed server. Then, when output stops, you're stuck with a hung job that you can't break out of. The kernel continues to try to contact the downed server (and ignores interrupt signals!) until the server comes back up (when it kills the job with one of the pended interrupts it has stored).

Interruptible hard mounts solve many of the problems associated with hard mounts. They provide operation identical to hard mounts, but they enable you to "interrupt" operations from the keyboard if a server does not respond. If you are running a *df* that gets hung, for example, you can break out of it within 30 seconds by typing an interrupt from your keyboard. (Note that certain types of processes—*emacs* sessions, for example—ignore all types of signals, and so cannot be interrupted, no matter what type of mount option is used. You can, however, interrupt these processes by sending a *kill* signal from another keyboard.)

Soft-mounted partitions return the *ETIMEDOUT* error to a user's application after 3 or 4 attempts to contact a downed server. This allows the application to recover and retry the request or print an error and continue. Soft mounts work best with file systems mounted read-only with the *ro* option and for file systems intended primarily for perusal (sources, for example, or directories of manual pages).

For more information on mount options, see *mount*(8) and *fstab*(5). To make sure you have successfully mounted a file system (and that you've mounted it where you expected to) use either *df*(1) or *mount*(8), without an argument. Each of these displays the currently mounted file systems. (Note: Using the *-t* option with *df*(1) enables you to specify the type of file system—*nfs* or "4.2"—to be displayed. To display only the *nfs* partitions mounted, for example, use **df -t nfs**.)

Starting and Killing *nfs* Daemons

Like many system-level programs, both *nfs* and *yp* rely heavily on daemons. Under most circumstances, you won't notice the operation of these daemons. If they die, however, or are accidentally killed, you need to know how to restart them. The following listing contains instructions for restarting these daemons. Since you must often kill daemons to start others, instructions for killing *yp* daemons are also included here. (Note that *yp* daemons */etc/ypbind* and */usr/etc/ypserv* are discussed in the next chapter. Ignore instructions for the use of these daemons if you are not running *yp*.)

- */etc/portmap*—This daemon should always be the first daemon started in any sequence. There is no reason to ever kill it. If this daemon should for some reason die, kill (and then restart) */etc/inetd*, */etc/ypbind*, */usr/etc/ypserv*, and */etc/nfsd* according to the instructions included subsequently.

- */usr/etc/ypserv*—This daemon should only be started if your machine is a yellow pages slave or master server machine. Start it after the domain name is set and after */etc/portmap* is started. If it dies, restart it with the command sequence:

```
# ps ax | grep ypserv
# kill -9 [all processes found]
# /usr/etc/ypserv
```

- */etc/ypbind*—This daemon should only be started if your machine is running *yp*; that is, invoke this daemon only if your machine is a yellow pages server or client. When this daemon is running, system programs use *yp* services rather than reading the local files in the the */etc* directory. If this daemon dies, restart it with the command:

```
# ps ax | grep ypbind
# kill -9 [all processes found]
# /etc/ypbind
```

- */etc/biod*—This daemon should be started after */etc/portmap* but before you mount file systems with *nfs*. Typically, four *biod* daemons are started. To kill and restart them, use the following command sequence:

```
# ps ax | grep biod
# kill -9 [all processes found]
# /etc/biod 4
```

- */etc/nfsd*—This daemon should be started after the other daemons in */etc/rc.local*. Typically, four *nfsd* daemons are started. To kill and restart them, use the following command sequence:

```
# ps ax | grep nfsd
# kill -9 [all processes found]
# /etc/nfsd 4
```

- */etc/inetd*—This daemon is started from the */etc/rc* file. It should be sent a *SIGHUP* (*kill -1*) signal; this signal causes *inetd* to read its configuration file, */etc/inetd.conf*. (*inetd* reconfigures itself each time you kill it, sets up new sockets, and restarts itself.) To kill *inetd*, use the following command sequence:

```
# ps ax | grep inetd
# kill -1 [process number found]
```

Record Locking

Original versions of UNIX did not support record locking. In the past, typical UNIX customers have not maintained large databases or developed commercial database applications that required record locking mechanisms. To better serve the needs of commercial users, however, CONVEX now supports advisory record locking. This capability ensures System V compatibility and provides a mechanism for locking records in a network environment, making distributed databases using *nfs* more feasible and useful. Utilities used to support record locking include *fcntl*(2) (executes file and record locking requests), *lockf*(3) (a user-friendly front end to *fcntl*), *lockd*(8C) (the network lock daemon), and *statd*(8C) (used as a status monitor). Subsequent sections explain the use of these programs.

How Record Locking Works

Record-locking mechanisms enable users to control access to particular file regions, or *records*. Because UNIX has no concept of records, records are defined as arbitrary sequences of bytes located at the current file pointer. (Note that record-locking schemes differ from *file-locking* programs, which prevent processes from reading or writing entire files.) *Advisory* record locking enables cooperating processes to implement record locking, but the kernel does not in any way force other processes to respect the advisory locks. This means that errant processes may access previously locked records without using advisory locks, possibly resulting in database inconsistencies. Note, however, that in practice, application programs running on multiple *nfs* clients can cooperate to lock and update records of a master database effectively.

CONVEX supports two types of advisory locks: *shared* and *exclusive* locks. Shared locks enable cooperating processes to read locked records. Exclusive locks are placed when processes either write, or read and write, records. More than one process may hold a shared lock at any given time; but multiple exclusive, or shared and exclusive, locks may not exist simultaneously on the same record.

How *lockf* Works

The *lockf*(3) library routine is a front end, or user interface, to the *fcntl*(2) system call, which performs file and record locking. *lockf* is used to place advisory locks on *records*. In this respect, it as an improvement over *flock*, which provides a file locking mechanism only. *lockf* works by enabling you to specify the number of contiguous bytes within a particular file to be locked or unlocked. This data is passed in the *size* argument and may be positive (to indicate bytes extending forward from the current file offset) or negative (to indicate bytes preceding but not including the current offset). Logically, of course, these regions of contiguous bytes represent records. (Note that if you expand these regions of contiguous bytes far enough, you have, in effect, implemented file locking; *lockf* can be used for file locking using this method.)

lockf enables you to lock and unlock file regions; test a region for other locks; and to test and then lock a region. Test-and-lock operations that fail with a *-1* return the *No more processes* (EAGAIN) diagnostic if the section is already locked by another process. Lock operations sleep until locks are granted or a signal is caught.

Note that *lockf* does not offer all of the functionality of *fcntl*. In particular, *lockf* restricts you to:

- Use of exclusive locks (*lockf* does *F_WRLCK fcntl()* requests only)
- Starting at the current position in the file

Note further that *lockf* returns the*EDEADLK* diagnostic, not the *ENOLCK* returned by *fcntl*. (*flock* returns *EDEADLK* to ensure compatibility with */usr/group* standards.) Typically, complex tasks like specifying shared record locks or using a file offset different from the current position are best handled with *fcntl*.

How *lockd* Works

lockd is a user daemon process installed on *nfs* file servers. *lockd* processes and arbitrates lock requests from local processes and remote *nfs* client processes. *lockd* works as follows. When the user issues an *lockf*(3) or *fcntl*(2) call, the kernel contacts the local */etc/rpc.lockd* process. (This happens whether the file is local or remotely mounted via *nfs*.) If the *lockd* process has not been registered with the port mapper (*/etc/portmap*), *fcntl* exits with the *ENOLCK* diagnostic. If */etc/portmap* has not been started, *fcntl*(2) waits (theoretically forever) either for *portmap* to be started or until a signal is delivered to the process (user presses the interrupt key, for example).

After the local *lockd* process has been contacted, the kernel sends the particular locking request to the local *lockd* process (set lock, test lock, unlock, ...), and the local *lockd* processes the request as follows:

- *lockd* determines if the request is for a file on the local machine or for a partition remotely mounted with *nfs*.
- If the request is for a lock on a remote file, *lockd* contacts the local *statd* process to register for monitor server-crash- recovery services. Crash recovery is explained more completely in a subsequent section.
- If the request is for a remote file, *lockd* contacts the remote server's *lockd* process to register the request and return the result of the operation; if the request is for a local file, *lockd* simply registers the request and returns the results to the kernel.

Basically, the kernel forwards all record-locking requests to the local *lockd* process. The local *lockd* process contacts the local *statd* process for remote files. Then the local *lockd* process either contacts the remote *lockd* (if the request is for a remote file) or processes the request (if the request is for a local file). The results are then returned to the kernel. The kernel delivers the results to the user.

How to Install the Lock Manager System

To install the lock manager system, you must edit your */etc/rc.local* file to start up the two daemon programs—*/etc/rpc.statd* and */etc.rpc.lockd*— that make up the lock manager. Modify */etc/rc.local* as shown in Figure 2-1:

Figure 2–1: Starting Up Lock Manager Daemons

```
#
#  Start statd and lockd by adding the indicated lines
#  to /etc/rc.local
#
#
/usr/etc/quotaon -a
#
echo -n 'local daemons:'              >/dev/console
if [ -f /etc/nfsd ]; then
  /etc/nfsd 4 & echo -n ' nfsd'       >/dev/console
fi
#  Look for the previous lines
#  and add the next six lines
if [ -f /etc/rpc.statd ]; then
  /etc/rpc.statd & echo -n ' statd' >/dev/console
fi
if [ -f /etc/rpc.lockd ]; then
  /etc/rpc.lockd & echo -n ' lockd' >/dev/console
fi
```

Use this procedure whether you plan to use record locking only on local files or in the network environment with *nfs*.

The */etc/rpc.statd* program is a status monitor. It is informed of the recovery of *nfs* servers after they crash and communicates with */etc/rpc.lockd* to provide crash-and-recovery functions.

/etc/rpc.lockd processes record-locking requests sent either locally by the kernel or remotely by another lock daemon. Lock requests are forwarded to the server site's lock daemon using standard *rpc/xdr* routines. The lock daemon then requests the *stat* daemon (status monitor) for monitor services.

How to Use the Lock Manager

The lock manager system enables you to develop applications that perform transaction-type record locking on both CONVEX UNIX and System V implementations. Primary features of the system include the following:

- Compatibility with System V record locking
- Extension of the record-locking concept onto the *nfs* network environment by the use of the user processes *rpc.statd* and *rpc.lockd*.
- File location transparency. In general, applications do not need to know whether files are local (locked by local *lockd* processes) or remote (mounted via *nfs* and locked by remote *lockd* processes).

Note that your application may be sent a *SIGLOST* signal if a file lock on an *nfs* file cannot be reclaimed after a server crash. This signal is not standard to System V—it is an extension developed when record locking was extended into the network environment using *nfs*. If you want to use *SIGLOST* in your application, you can use the *#ifdef* C preprocessor feature as shown in Figure 2-2. Using *#ifdef* enables you to use *SIGLOST* while maintaining System V source code compatibility.

Figure 2–2: Adding *SIGLOST* Signal to Application Programs

```
#include <signal.h>
        ...
#ifdef SIGLOST
int lostlock();
#endif
        ...
#ifdef SIGLOST
        signal(SIGLOST, lostlock)
#endif
        ...
#ifdef SIGLOST
lostlock()
{
        /*
         * Code here to handle lost record lock; would perhaps
         * try to regain its lock again or exit with a message
         */
}
#endif SIGLOST
```

To use the lock manager system, you need to make the appropriate calls either to the *lockf*(3) library routine or, with appropriate arguments, to the actual *fcntl*(2) system call. As you recall, *lockf*(3) is simply a user-friendly interface to *fcntl*(2). *lockf*(3) and *fcntl*(2) are compatible with the System V versions. Note that you can use the system to lock an entire file (even if the file grows or shrinks), so your application can lock either pieces of the file or the entire file itself.

The lock manager is often used to control access to a file by competing processes. An example is the method used to update your mailbox when mail arrives. The mail system never updates your mailbox until it can gain exclusive access to it. The mail system then releases its lock on the file; then other processes can gain access. The cycle continues until all of your mail has been successfully delivered without being corrupted by having multiple processes updating the file at the same time.

Crash Recovery

The *statd* process maintains state information about which machines have outstanding record lock requests of this server. This information is contained in the */etc/sm* directory in the form of one directory entry for each host that has requested monitor services. When *statd* starts up, it notifies the *statd* process of each host listed in the */etc/sm* directory that it has recovered. When the remote *statd* receives this notification, it notifies its local *lockd* of the recovery. The local *lockd* tries to reclaim the lock, if possible. If it cannot, the process is sent a *SIGLOST* signal to note the lost lock.

Errors Returned

Basically, the errors returned are similar to the errors returned for most of the other system calls (*EBADF, EFAULT, EINVAL, EAGAIN, EINTR*). The new error code for *fcntl*(2) is *ENOLCK*, which is only returned by the record-locking code. *ENOLCK* basically means that no resources are available for this lock, or that the local *lockd* process cannot be contacted.

The *lockf*(3) library returns the same errors as the *fcntl*(2) system call does, except that it changes the *ENOLCK* error to be *EDEADLCK* for */usr/group* compatibility.

Remote Execution Utilities: *rex* and *rexd*

rex(1C) and its accompanying daemon, *rexd*(8C), enable you to execute commands remotely, providing you with the opportunity to off-load *nroff, make,* and other jobs onto lightly loaded machines. Unlike *rsh* and *rlogin*, *rex* neither starts a shell on the remote host nor performs remote logins. Instead, *rex* accomplishes the remote execution task by mounting local file partitions on remote machines via *nfs*. In an open environment, *rex* can also be used to improve load balancing between machines. *rex* does not require installation and uses a simple syntactical structure. Subsequent sections describe the use and administration of *rex* and *rexd*.

Like *rlogin* and *rsh*, *rex* operates quickly, without discernible start-up time. In fact, because *rex* doesn't perform remote logins or start up remote shells, it is faster than either *rsh* or *rlogin*. Note, however, that while *rex*'s quick start-up time is certainly an advantage, *rex* users are limited to relatively simple command sequences. Because *rex* does not start a remote shell, it cannot interpret the complex command sequences interpreted by *rsh*.

rex and *rexd* are included by default on standard *nfs* release distribution tapes. Because no installation is required, you should be able to use these utilities immediately. If you have trouble locating or using *rex* or *rexd*, contact the CONVEX Technical Assistance Center.

Using *rex*

The standard syntax for using *rex* is:

% rex -i -n -d *host command arguments*

where:

-i	Is used to invoke *interactive* mode (described subsequently).
-n	Is used to specify *no input*. This option is used to "close" standard input so that jobs can be run in the background with job control. (Ordinarily, remote standard input is passed from *rex*'s standard input.)
-d	Is used to invoke *debugging* mode. In debugging mode, *rex* prints diagnostic messages as work is being done.
host	Is the name of the remote host jobs are to be run on.
command	Is the command to be run.
arguments	Are standard command-line arguments (for example, the *-la* arguments commonly used with *ls*).

rex is commonly used to off-load high-overhead jobs (*nroff* and *make* jobs are prime examples). If, for example, you want to run an *nroff* job on a processor named *lotsacycles*, use *rex* as follows:

```
% rex lotsacycles nroff *.s
```

Although *rex* is convenient, you might have tried to use *rsh* to perform the same task. Unfortunately, *rsh* would not have worked because the two utilities work differently—*rsh* executes commands in your *home* directory on the remote host, while *rex* executes commands after mounting your current *local* working directory on the remote host. In the example above, *rsh* would have looked for in your home directory on the remote machine, and never would have found the local files you wanted to process. In fact, unless your remote home directory contained some *.s* files, *rsh* probably would have failed to do anything except return an error message. Table 2-1 illustrates a list of the differences between *rex* and *rsh*:

Table 2-1: Differences Between *rex* and *rsh*

rex	rsh
Executes command with arguments on host after mounting current local working directory and file system (if not already mounted).	Executes command with arguments in home directory on host.
Can't execute shell builtins (e.g., *alias*).	Can execute shell builtins because default shell is started on host.
Inherits current directory and ENVIRONMENT variables.	Does not inherit current directory or ENVIRONMENT variables.
Runs interactive commands.	Cannot run interactive commands.
Propagates terminal mode and window size to remote host via interactive commands.	Cannot perform these functions.

The important thing to remember is that while *rsh* has no concept of networked file systems, *rex* always uses them. (If a local file system is not mounted on the remote host, *rex* always tries to mount it.) *rex* always works from *local* files, *on* a remote machine. *rsh* always works from *remote* files, *on* a remote machine.

Because *rex* works from remotely mounted versions of the local file system, you should use relative pathnames that exist within the current file system. Note that *rex* mounts entire file systems, whether or not you are at a mount point. For example, the following command sequence:

```
% pwd
% /mnt/moe/src
% rex make all
```

mounts */mnt*, not */mnt/moe/src*. Once the file system is mounted, *rex* changes directories to the current working directory.

You can also use *rex* to run interactive jobs (like text editors) remotely. Command sequences similar to the following enable you to edit *local* files on a remote host interactively:

```
% rex -i convex vi *
```

To edit *remote* files, use a command sequence similar to:

```
% rex -i convex vi /etc/hosts
```

This command sequence enables you to interactively edit *convex*'s */etc/hosts* file. Using this approach, you can more easily complete tedious tasks like editing messages of the day around the network.

Examples of Advanced *rex* Usage

Figure 2-3 shows how *rex* can be used in a shell script to simplify day-to-day programming tasks. In this example, *rex* is used to run a job on the most lightly loaded machine available from the set including *fred, barney, pebbles*, and *bambam*. To use the script, add it to your */bin* directory as *qrex*, and invoke it via:

```
% qrex job_name
```

Figure 2–3: *rex* Application Example

```
#!/bin/sh
#
# qrex -- run a rex job on the lowest loaded machine that is listed as
#                                  a good candidate.
#
# The HOSTS variable contains an egrep format string of hosts that run the
# rex daemon and can compile sources to produce Convex binaries.
#
HOSTS="^fred |^barney |^wilma |^pebbles |^bambam "
REXHOST=`ruptime -l | egrep "$HOSTS" | grep " up " | tail -1 | awk '{print $1}'`

if [ -z "$REXHOST" ]; then
   echo $0: No host available 1>&2
   exit 1
fi

rex $REXHOST "$e"
```

rex can also be used with symbolic links to shorten command sequences that you use frequently. The following command sequence, for example:

```
% ln -s /usr/bin/rex ~/bin/convex
```

establishes a symbolic link between *rex* and a command name (*convex*) in your local */bin* directory. Having created the link, you can shorten command sequences referencing the host, *convex*. For example, once you've created the symbolic link, you can type:

```
% convex make
```

instead of:

```
% rex convex make
```

If you decide to use symbolic links, you'll obviously need symbolic links for every host you intend to use.

Administrative Issues

To administer the remote-execution system, you must understand not only *rex*, but also *rexd*, its accompanying daemon. The following sections describe how to use and administer *rexd*, and discuss how to avoid potential security problems.

Using *rexd*

rexd is the server-side daemon for remote program execution. It is started by *inetd*(8C) and so is included in the *inetd* configuration file, */etc/inetd.conf*. If you encounter problems with *rex*, check *inetd.conf* for the following line:

```
rexd   stream    tcpwait   root   1 /usr/etc/rexd     rexd
```

If you don't find this line, add it and restart *inetd* according to the instructions included in this chapter.

rexd is started up when a user invokes *rex*—it doesn't run continuously. When pended *rex* requests have been completed, *rexd* times itself out, and *inetd* takes control of the socket *rexd* was using.

There is no obvious way for *rexd* to break. If *rexd* doesn't seem to be working, however, kill it via the following command sequence:

```
# ps ax | grep inetd
# kill -1 [process ID found above]
```

Once *inetd* receives the signal, it reconfigures *inetd.conf* and restarts itself. The reconfiguration should eliminate problems with *rexd*.

Security Issues

Unlike other networking utilities, *rexd* enables users to mount directories. In theory, this flexible approach can lead to security problems. If, for example, a user with a home directory in */mnt* invokes the following command sequence:

```
% rex convex4 sleep 10000
```

/mnt is mounted on a */tmp* partition on the remote host, *convex4*. The mount point stays active for 10000 seconds, giving users on *convex4* access to */mnt*.

You can avoid this type of problem, of course, by protecting sensitive partitions in */etc/exports*. If, in the previous example, */mnt* had not been included in the *exports* file, users would be unable to mount the partition. Instead, they would receive the error message:

```
can't mount file system
```

Note that permission checking under *rex* is as strong as the permission checking used with *rsh* and *rlogin*. In all cases, the system checks not only for legitimate passwords, but also checks against the listings in */etc/host.equiv*. All things considered, *rex* is a reasonable security risk in open environments, but if you operate a secure environment, you may want to reconsider distributing *rex* to

your users. If you decide not to provide *rex*, you can turn it off by removing its entry from */etc/inetd.conf*, then reconfiguring *inetd* by sending a *SIGHUP* (*kill -1*) signal.

Using Named Pipes

Software supporting named pipes is distributed to CONVEX licensees. Named pipes are "ordinary" pipes that exist permanently in the file system with directory entries and pathnames. Because these pipes can be accessed by name, they can be used for applications for which unnamed pipes are not suited. Typically, named pipes are used to allow a number of processes to communicate with a daemon process.

Note that CONVEX named pipes can only be used locally. Even if the named pipe exists as an inode on a remote machine, data written to the pipe is only accessible to processes on the same client. Therefore, even though multiple clients mount the same file system, they will have separate streams associated with any named pipe in that file system.

For more information on the use of named pipes, see *mknod*(2), *mknod*(8), and *open*(2).

Debugging the Network File System

Before trying to debug *nfs*, read the section on how *nfs* works plus the man pages for *mount*(8), *nfsd*(8), *biod*(8), *showmount*(8), *rpcinfo*(8), *mountd*(8c), *inetd*(8c), *fstab*(5), *mtab*(5), and *exports*(5). You don't have to understand them fully, but you should be familiar with the names and functions of the various daemons and database files.

When tracking down an *nfs* problem keep in mind that, like all network services, the three main points of failure are: the server, the client, or the network itself. The debugging strategy outlined below tries to isolate each individual component to find the one that isn't working.

For example, let's look at a sample mount request made from an *nfs* client machine:

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

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

Now, obviously, if you are experiencing problems, things are not working as described in this example. So how do you determine where the problem is? First, look at the next section, that contains some general pointers. Then read the subsequent sections that describe problems and their likely causes.

General Hints

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

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

to see if the server is up at all. The server should print a list of program, version, protocol, and port numbers that resembles:

```
program vers proto   port
 100004    2   udp   1027  ypserv
 100004    2   tcp   1024  ypserv
 100004    1   udp   1027  ypserv
 100004    1   tcp   1024  ypserv
 100007    2   tcp   1025  ypbind
 100007    2   udp   1035  ypbind
 100007    1   tcp   1025  ypbind
 100007    1   udp   1035  ypbind
 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
 100002    2   udp   1095  rusersd
 100001    1   udp   1098  rstatd
 100001    2   udp   1098  rstatd
 100001    3   udp   1098  rstatd
```

If that works, you can also use *rpcinfo* to check if the *mountd* server is running:

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

This should come back with the response:

```
proc 100005 version 1 ready and waiting
```

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

If the server is alive but your machine can't reach it, you should check the Ethernet connections between your machine and the server.

If the server is okay and the network is okay, use *ps* to check your client daemons. You should have a *portmap*, *ypbind*, and several *biod* daemons running. (Note that *ypbind* is running only if you are running *yp*.) For example, running *ps* should result in output something like this:

```
% ps ax
32 ?  I     1:07 /etc/portmap
38 ?  I     0:42 /etc/ypbind
61 ?  S     0:45  (biod)
62 ?  S     0:36  (biod)
63 ?  S     0:30  (biod)
64 ?  S     0:27  (biod)
```

The four sections below deal with the most common types of failure. The first tells what to do if your remote mount fails; the next three describe servers not responding after you have file systems mounted.

/etc/rc.local Startup File

You must edit the */etc/rc.local* startup file to become an *nfs* or yellow pages (*yp*) client/server. To verify the daemons that must be started and the order in which they must be started, refer to Appendix B, "Updating */etc/rc.local*."

When Remote Mount Operations Fail

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

mount can get its parameters either from the command line or from the file */etc/fstab* (see *mount*(8)). The example below assumes command line arguments, but the same debugging techniques work if */etc/fstab* is used in the *mount -a* command.

Keep in mind the interaction of the various players in the mount request. If you understand this, the problem descriptions below make more sense.

Let's look again at the previous *mount* request example:

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

and see what 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. If you are running *yp*, *mount* calls *yp* binder daemon *ypbind* to determine which server machine to find *yp* server on. It then calls the *ypserv* daemon on that machine to get the Internet protocol (IP) address of *krypton*. (If you are not running *yp*, *mount* looks in the local version of */etc/hosts*.)
4. *mount* calls *krypton*'s portmapper (*/etc/portmap*) to get the port number of *mountd*.
5. *mount* calls *krypton*'s *mountd* and passes it */usr/src*.
6. *krypton*'s *mountd* reads */etc/exports* and looks for the file system containing */usr/src*.
7. *krypton*'s *mountd* calls *yp* server *ypserv* to expand the host names and netgroups in the export list for */usr/src*. (This happens only if you are running *yp*—otherwise, *mountd* uses the local version of */etc/hosts*.)
8. *krypton*'s *mountd* does a *getfh*(2) system call on */usr/src* to get the *fhandle*.
9. *krypton*'s *mountd* returns the *fhandle*.
10. *mount* supplies the *fhandle* and */krypton.src* via the *mount*(2) system call.
11. *mount* checks if the caller is superuser and if */krypton.src* is a directory.
12. The *mount* system call does a *statfs*(2) call to *krypton*'s *nfs* server (*nfsd*).
13. *mount* opens */etc/mtab* and adds an entry to the end.

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

- *mount: ... already mounted*

  The file system that you are trying to mount is already mounted or it has a bogus entry in */etc/mtab*.

- *mount: ... Block device required*

  You probably 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*.

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

  If *mount* is called with a directory or file-system name but not both, it looks in */etc/fstab* for an entry whose file system or directory field matched the argument. For example:

  ```
  # mount /krypton.src
  ```

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

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

  it does the mount as if you had typed:

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

  If you see this message, it means the argument you gave *mount* was 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*

  This means *yp* could not find the hostname you gave it in the */etc/hosts* database or that the *yp* daemon (*ypbind*) is dead on your machine. (Note that if you are not running *yp*, no hostname entry is in the local */etc/hosts* file.) First check the spelling and the placement of the colon in your mount call. If it looks okay, make sure that *ypbind* is running by typing:

  ```
  # ps ax | grep ypbind
  ```

  Try *rlogin* or *rcp* to some other machine. If this also fails, your *ypbind* is probably dead or hung. If you only get this message for one host name, it means that the */etc/hosts* entry on *yp* server needs to be checked. See the section on debugging *yp* later in this chapter.

- *mount: directory path must begin with '/'*

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

- *mount: ... server not responding: RPC_PMAP_FAILURE - RPC_TIMED_OUT*

  Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, try running:

```
# rpcinfo -p
```

You should get a list of registered program numbers. If you don't, restart the portmapper on the server according to the instructions in "Starting and Killing *nfs* Daemons". Note that restarting the portmapper requires that you then kill and restart both *inetd* and *ypbind*.

If you can't *rlogin* to the server but the server is up, check your Ethernet connection by trying *rlogin* to some other machine. Also check the server's Ethernet connection.

- *mount: ... server not responding: RPC_PROG_NOT_REGISTERED*

This means that mount got through to the portmapper but the *nfs* mount daemon (*rpc.mountd*) was not registered. Go to the server and be sure that */usr/etc/rpc.mountd* exists and that there is an entry in */etc/inetd.conf* exactly like:

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

Finally, use *ps* to be sure that the Internet daemon (*inetd*) is running. If you had to change */etc/inetd.conf*, you must kill *inetd* and restart it. Again, refer to the instructions in "Starting and Killing *nfs* Daemons." (A complete description of the *inetd.conf* file is contained in Chapter 4 of the *RPC Programmer's Manual.*)

- *mount: ...: No such file or directory*

Either the remote directory or the local directory doesn't exist. Check spelling. Try to *ls* both directories.

- *mount: not in export list for ...*

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

```
# showmount -e hostname
```

If the file system you want is not in the list, or your machine name or netgroup name is not in the user list for the file system, log in to the server and check the */etc/exports* file for the correct file system entry. A file system name that appears in the */etc/exports* file, but not in the output from *showmount*, indicates a failure in *mountd*. Either it could not parse that line in the file, or it could not find the file system, or the file system name was not a local-mounted file system. See *exports*(5) for more information. If *exports* seems okay, check the server's *ypbind* daemon: it may be dead or hung.

- *mount: ...: Permission denied*

This message is a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above), or the server couldn't figure out who you are (*ypbind* dead), or it could be that the server doesn't believe you are who you say you are. Check the server's */etc/exports* and *ypbind*. Here, you can just change your hostname with *hostname*(1) and retry the *mount*.

- *mount: ...: Not a directory*

Either the remote path or the local path is not a directory. Check spelling and try to *ls* both directories.

- *mount: ...: Not owner*

  You have to do the mount as root on your machine because it affects the file system for the whole machine, not just you.

- *mount: ...: Cannot mount a directory on top of itself*

  Self-explanatory.

When Programs Hang

If programs hang doing file-related work, your *nfs* server may be dead. If, for example, you are using machine *krypton* as a server, you may see the message *NFS server krypton not responding, still trying* on your terminal. If you see a message like this, there is probably a problem either with one of your *nfs* servers or with the Ethernet. You can figure out which *nfs* server it is by looking in */etc/hosts*, or by typing:

```
% ypcat hosts.byname | grep internet_address_#
```

Programs can also hang if a *yp* server dies (see *yp* below).

If your machine hangs completely, check the server(s) from which you have mounted. If one of them (or more) is down, don't worry; when the server comes back up, your programs continue automatically and they won't even know the server died. No files are destroyed.

If a soft-mounted server dies, other work should not be affected. Programs that time-out trying to access soft-mounted remote files fail with *errno ETIMEDOUT*, but you should still be able to get work done on your other file systems.

If all the servers are running, go ask someone else using the server or servers that you are using if they are having trouble. If more than one machine is having problems getting service, then it is probably a problem with the server's *nfs* daemon (*nfsd*(8)). Log in to the server and do a *ps* to see if *nfsd* is running and accumulating CPU time. If not, you may be able to kill and then restart *nfsd*. If this doesn't work, you must reboot the server.

If other people seem to be okay, you should check your Ethernet connection and the connection of the server.

When the System Hangs at Start-Up

If your machine comes part way up after a boot, but hangs where it would normally be doing remote mounts, probably one or more servers is down or your network connection is bad. See "Program Hung" and "Remote Mount Failed" above.

To avoid a hung system, add the *bg* flag to the *nfs* partitions listed in */etc/fstab*. For example:

```
# krypton:/usr/man /usr/man nfs rw,soft,bg 0 0
```

When Remote File Access Seems Slow

If access to remote files seems unusually slow, type:

```
# ps aux
```

on the server to be sure it is not being clobbered by a runaway daemon, bad *tty* line, etc. If the server seems okay and other people are getting good response, make sure your block I/O daemons are running; type *ps ax* and look for *biod*. If they are not running or are hung, you can find the process ID's by typing:

```
# ps ax | grep biod
```

and kill them with:

```
# kill -9 pid1 pid2 pid3 pid4
```

Restart them with:

```
# /etc/biod 4
```

To determine whether they are hung, do a *ps* as above, copy a large remote file and then do another *ps*. If the *biod*s don't accumulate CPU time, they are probably hung.

If *biod* is okay, check your Ethernet connection. The command *netstat -i* tells you if you are dropping packets. Also, *nfsstat -c* and *nfsstat -s* can be used to tell if the client or server retransmission rate is too high. A retransmission rate of 5% is considered high. Excessive retransmission usually means a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between your Ethernet board and the server's board.

Tuning *nfs*

The following topics are discussed in this section:

- Superuser access to networked files
- How to enable asynchronous *nfs* operation
- How to check privileged ports
- How to check IP source addresses
- How to patch SUN clients for compatibility with file systems with large block sizes
- Correcting clock skew in user programs

Superuser Access to Remote Files

Under *nfs*, a server exports file systems it owns so clients may remote mount them. When a client becomes superuser, it is denied permission on remote-mounted file systems. Let's look at the following example:

```
% cd
% touch test1 test2
% chmod 777 test1
% chmod 700 test2
% ls -l test*
-rwxrwxrwx  1 jsbach      0 Mar 24 16:12 test1
-rwx------  1 jsbach      0 Mar 24 16:12 test2
```

Now, retry it again as root.

```
% su
Password:
# touch test1
# touch test2
touch: test2: Permission denied
# ls -l test*
-rwxrwxrwx  1 jsbach      0 Mar 21 16:16 test1
-rwx------  1 jsbach      0 Mar 21 16:12 test2
```

The problem usually shows up during the execution of a set-uid root program. Programs that run as root cannot access files or directories unless the permission for *other* allows it.

There's another wrinkle to the problem. You cannot change ownership of remote-mounted files. Since users cannot do a *chown* command and root is treated as a normal user on remote access, no one but root on the server can change the ownership of remote files. For example, as yourself, you attempt to *chown* a new program, *a.out*, that must be set-uid root. It does not work, as demonstrated here:

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change the ownership, you must log in to the server as root and then make the change. Or, you can move the file to a file system owned by your machine (for example */usr/tmp* is always owned by the local machine) and make the change there.

If you are prepared to accept the security risks, you may also solve this problem by enabling over-the-net root access. Enabling over-the-net root access allows client root accounts superuser privileges on remotely-mounted partitions. Over-the-net root access can be enabled in either of two ways: on a machine-by-machine basis (via *sysgen*(8)) or on a file-system-by-file-system basis (via changes to the */etc/exports* file). Before you consider using either of these methods, however, note carefully:

CAUTION

CONVEX does not recommend enabling over-the-net root access as discussed in the following paragraphs.

To enable over-the-net root access on a particular machine, you must change the value of the kernel variable *nobody* in the *nfs* server's kernel. To make the change, reconfigure the kernel using *sysgen*(8). Instructions for using *sysgen* are included in the *CONVEX System Manager's Guide*.

A less drastic approach is to enable over-the-net root access for particular file systems. To do this, add one of three flags (*-0*, *--2*, *-r*) to the listings in */etc/exports*. These flags cause the kernel *nobody* variable to be mapped to *-0* (superuser privileges enabled), *-2* (superuser privileges disabled), and *-r*

(read-only privileges), respectively. (Note that an extra minus sign is used with UIDs that are negative; *e.g. --2.*) Although the three arguments discussed here are used most frequently, other arguments are available; see *exports*(5) for more information. Modify the server's */etc/exports* file as follows:

- To enable over-the-net root access: For each file system to be accessed, add the *-0* flag after the listing. In the following example, the */usr* file system has been modified to accept over-the-net root access by *convex2* and *convex3*:

```
/tmp
/sw
/mnt1
/usr        -0      convex2  convex3
/fonts              convex4
/usr/spool          convex2 convex3 convex4
```

- To disable over-the-net root access: You can disable over-the-net root access by adding the *--2* flag after the listing of each file system to be protected. In the following example, over-the-net root access is allowed on */tmp* and */usr/spool*, but prohibited *only* on */usr*:

```
/tmp                -0      convex2 convex3
/usr                --2     convex2 convex3
/usr/spool          -0      convex2 convex3
```

- To enable read-only access: For each file system to be accessed read-only, add the *-r* flag after the listing. In the following example, over-the-net root access is mapped to UID -2 for every file system listed (because the default value is *-2*). Any user, however, including root, can read the files in */usr/spool*. No user, including root, however, can *write* them. Note that although users can mount */usr/spool* read/write, attempted writes by any user (including root) fail with the *EROFS* (Read-Only File System) error:

```
/tmp
/sw
/mnt1
/usr
/fonts
/usr/spool          -r      convex2 convex3
```

Asynchronous *nfs* Operation

nfs servers normally operate in a synchronous fashion—data is written to permanent storage before an *nfs* transaction is acknowledged. Synchronous operation enables the server to maintain file integrity in spite of crashes. With synchronous operation, the client can't tell the difference between a server crash and slow server response. It just retries until the transaction is successfully completed.

The penalty paid for this reliability is the extra I/O performed on the server. Each time the client makes a write request, the server writes not only the data buffer, but "in-core" copies of the inode and indirect blocks as well. CONVEX systems software staff estimate that writing large files could proceed several times faster if the server operated asynchronously, making full use of the UNIX buffer cache.

The following paragraphs describe how to "patch" your system to enable asynchronous server operation. Note that the changes affect all the file systems served by a machine; there is no way to

specify that certain files or file systems be served asynchronously. Note that the patch is invisible to clients: except for the fact that data consistency may *not* be maintained across server crashes, the changes are not noticed by clients.

CAUTION

> Do not attempt the following procedures until you carefully consider the *nfs* environment to be changed and its ability to recover from inconsistent data if a server fails. Weigh carefully your need for the extra performance against the inconvenience and administrative effort required when the server crashes. Note that client machines may not be aware of the server's crash, and thus may not be able to detect that data has been lost.

To patch the CONVEX UNIX kernel for asynchronous server operation, use *adb* as follows:

```
# adb -w -k /vmunix /dev/mem
  enable_async_nfs_server/w              (display current value)
  enable_async_nfs_server/=w 1           (change memory copy to on)
  $q
%
```

Note that you can make these changes while the system is running since they take effect immediately.

Another way to enable asynchronous operation is to change the *defboot.cmd* script on the SPU, so that the change is made automatically whenever you boot the system. To do this, add the following line to *defboot.cmd* on the SPU:

```
options          _enable_async_nfs_server=0x1
```

If you already added other options to *defboot.cmd*, change the script as follows:

```
options          _enable_unique_core=0x0,_enable_async_nfs_server=0x1
```

Note that file systems served by asynchronous servers should be mounted "soft," so that *nfs* returns an error when the server malfunctions. You can set the time-out value and retry count values so that the operation times out and returns an error before the server can be rebooted. (The default values ensure this.)

Checking Privileged Ports

The Berkeley UNIX operating system includes some Internet domain source ports to which only privileged users can attach (these are known as "privileged ports"). Currently, *nfs* does not check to see if a client is bound to one of these. That is, an *nfs* server has no way of knowing whether a client's file request originated from the real client's kernel or from some miscreant's user-program. To turn on *nfs* server port checking, use the following procedure:

1. Add the following line to *defboot.cmd* on the SPU:

   ```
   options_nfs_portmon=0x1
   ```

2. Change */etc/inetd.conf* to the following line:

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

add the *-p* flag. When you finish, the line looks like this:

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

3. Restart *inetd* according to the instructions in "Starting and Killing nfs Daemons," that is, send *inetd* a *SIGHUP* (*kill -1*) signal. (If you don't restart *inetd*, privileged port checking does not start until the next reboot.)

Warning: Some non-UNIX systems do not enforce the privileged port convention (in particular, PCs with 3Com boards).

Checking IP Source Addresses

If you are in a situation where you cannot control all the root users on your net, then you can take some security steps. An *nfs* server has a list of machine names to which it exports file systems. This list is known as the export list and is defined in the file */etc/exports*. When a client makes a mount request to an *nfs* server, the server checks to see if the client's machine name appears in its export list.

This checking is not secure because one can use the *hostname* command to fake a machine-name to something in the server's export list. To improve security, you can have the server also check the client's source IP address. IP addresses can be faked by altering */etc/hosts*, but the machine being faked is warned repeatedly. The message *duplicate IP address! sent from Ethernet address: XXXXXX* appears on its console, displaying the faker's Ethernet address. To turn on source IP address checking, here's what you do:

1. Change */etc/inetd.conf* as follows. To the following line:

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

add the *-i* flag. When you finish, the line looks like this:

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

2. Restart *inetd* according to the instructions in "Starting and Killing *nfs* Daemons"; that is, send *inetd* a *SIGHUP* (*kill -1*) signal. (If you don't restart *inetd*, IP source address checking won't start until the next reboot.)

The next time the *rpc.mountd* process runs, it checks the source IP address.

Warning: This may not work for clients that are gateways because they have more than one IP address.

Client Bugs With Large File System Block Sizes

If you are using a Sun 3 workstation as a client to a CONVEX *nfs* server, note the following. Releases prior to and including Version 3.0 of the Sun operating system panic when they access remote file systems with block sizes larger than 8 Kbytes. (Panics are due to an *nfs* bug.) This bug is relevant to you because CONVEX servers can export file systems with block sizes as large as 64 Kbytes.

It is possible to fix this bug by patching the Sun 3 kernel. To do this, use *adb* to change nfs_mount+2b6. The previous instruction was bles, hex 6f06 (halfword). Replace it with a bra instruction, hex value 6006 (halfword). (Note that all these changes are to be made on the Sun machine.)

```
sun# adb -w /vmunix
      nfs_mount+2b6?x                (check for 6f06)
      nfs_mount+2b6?w 6006           (replace 6f06 with 6006)
      $q
%
```

Versions of *nfs* ported by other vendors may not be designed to support file systems with block sizes larger than 8 Kbytes. You may have difficulty using other types of hardware as a client for these file systems. Naturally, CONVEX *nfs* fully supports file systems block sizes from 4 Kbytes through 64 Kbytes.

Correcting Clock Skew in User Programs

Because the *nfs* architecture differs in some minor ways from earlier versions of the UNIX operating system, you should be mindful of those places where your own programs could run up against these incompatibilities.

Because each machine on the network keeps its own time, the clocks are out of sync between the *nfs* server and client. Obviously this might introduce a problem in certain situations. We have fixed all the clock skew problems we have seen. Here are examples of two problems and how they were fixed. Many programs make the (reasonable) assumption that an existing file could not have been created in the future. For example, *ls* does it. The command *ls -l* has two basic forms of output, depending on how old the file is:

```
# date
Jan 22 15:27:01 PST 1985
# touch file2
# ls -l file*
-rw-r--r--  1 root        0 Dec 27  1983 file
-rw-r--r--  1 root        0 Jan 22 15:27 file2
```

The first form of *ls* prints the year, month, and day of the last change to the file—if the file is more than six months old. The second form prints the month, day, and minute of the last change to the file is fewer than six months old.

Ls calculates the age of a file by simply subtracting the time of the last change to the file from the current time. If the results are greater than six months worth of seconds, the file is "old."

Now assume that the time on the server is Jan 22 15:30:31 (three minutes ahead of our local machine's time):

```
# date
Jan 22 15:27:31 PST 1985
# touch file3
# ls -l file*
-rw-r--r--  1 root        0 Dec 27  1983 file
-rw-r--r--  1 root        0 Jan 22 15:26 file2
-rw-r--r--  1 root        0 Jan 22  1985 file3
```

The problem is that the difference between the two times is huge:

```
(now) - (time_of_last_change) =
(now) - (now + 180 seconds) =
-180 seconds = huge unsigned number,
that is greater than six months.
```

Thus, *ls* believes the new file was created long ago in the past. *ls* has been modified to deal with files that are created a short time in the future.

ranlib(1) was also modified to deal with clock skew. *ranlib* timestamps a library when it is produced, and *ld* compares that timestamp with the last modified date of the library. If the last modified date occurred after the timestamp, then *ld* instructs the user to run *ranlib* again and reload the library.

If the library lives on a server whose clock is ahead of the client that runs *ranlib*, *ld* always complains. *ranlib* has been altered and now sets the timestamp to the maximum value of the current time and the library's modify time.

In general, remember that if your application depends on either local time or the file system timestamps, then it must deal with clock-skew problems when it uses remote files.

Incompatibilities With Earlier Versions

A few things work differently, or don't work at all, on remote *nfs* file systems. Here we discuss the incompatibilities and suggest how to work around them.

File Operations Not Supported

The *flock*(2) call fails to provide exclusive locks; that is, competing *nfs* processes on different *nfs* client machines can lock the same file simultaneously. For more information, see the sections on record locking in this chapter.

Also, append mode and atomic writes are not guaranteed to work on remote files accessed by more than one client simultaneously.

Access to Remote Devices

While using *nfs*, attempts to access a remote-mounted device or any other character or block special file (like named pipes) are treated as if a local device is being accessed.

3

Installing and Debugging *yp*

This chapter discusses the yellow pages at a conceptual level, describes how to install and administer the product, how to debug yellow pages clients and servers, and how the product affects system security. The chapter closes with a discussion of how to add a new user to the *yp* environment.

Note that *yp* is a completely optional product; *nfs* does not require it, nor does *yp* require *nfs*.

For more information on using *yp*, see the *Network File System User's Guide*.

What Is the Yellow Pages Service?

yp is a 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 doesn't matter which server process answers a client request; the answer is the same all over. This allows multiple servers per network and gives *yp* service a high degree of availability and reliability.
- *yp* is a lookup service. It maintains a set of databases that may be queried. A client may ask for the value associated with a particular key within a database, and may enumerate every key-value pair within 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.

yp can serve many databases. Typically these include some files that used to be found in */etc*. For example, before the release of the CONVEX UNIX V6.0 operating system, programs 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 want to look at the */etc/hosts* file now do a remote procedure call (*rpc*) to the servers to get the information. (A local file is used if *yp* is not running.)

Yellow Pages Map

The *yp* system serves information stored in *yp* "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* used by programs 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 usually identical to the format within the ASCII file. Maps are implemented by *dbm*(3X) files located in the subdirectories of the directory */etc/yp* on *yp* server machines.

Yellow Pages Domain

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

A domain name is required for retrieving data from a *yp* database. Each machine on the network belongs to a default domain set at boot time in */etc/rc.local* with the *domainname*(1) command. A domain name must be set on all machines, both servers and clients. Further, a single name should be used on all machines on a network.

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. A *yp* client machine runs *yp* processes and requests data from databases on other machines. 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 *yp* slave servers. If *yp* databases are created or changed on slave server machines instead of master server machines, the *yp* update algorithm breaks. Always create the database and change it on the master server machine.

A server may be a master of one map, and a slave to another. Random assignment of maps to server machines could introduce much confusion. We strongly urge you to make a single server the master for all the maps created by *ypinit*(8) within a single domain. This document assumes that one server is the master for all maps in the database.

Related Documentation

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, we list the man pages and an abstract of their contents here. For more information, see the *CONVEX Network File System Reference Set*, the *CONVEX Network File System User's Guide*, and the *CONVEX UNIX Programmer's Manual*.

ypserv(8) — describes the processes that constitute the *yp* system. These are *ypserv*, the *yp* database server daemon, and *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(5) — describes the database structure of the *yp* system.

ypinit(8) — many maps must be constructed from files located in */etc*, such as */etc/hosts*, */etc/passwd*, and others. The database initialization tool *ypinit*(8) does all such construction automatically. It also 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. Do not (generally) use it as an administrative tool for running systems.

ypmake(8) — describes the use of */etc/yp/Makefile*, that builds several commonly changed components of the *yp* database. These are the maps built from several ASCII files normally found in */etc*: *passwd, hosts, group, netgroup, networks, protocols, pwrestrict, rpc, ethers,* and *services*.

makedbm(8) — describes a low-level tool for building a *dbm* file that is a valid *yp* map. Databases not built from */etc/yp/Makefile* may be built or rebuilt using *makedbm*. *Makedbm* may also be used to "disassemble" a map so that you can see the key-value pairs that constitute it. The disassembled form may also be modified with standard tools (such as editors, *awk*, *grep*, and *cat*) and is in the form required for input back into *makedbm*.

ypxfr(8) — moves a *yp* map from one *yp* server to another, using the *yp* itself as the transport medium. It can be run interactively, or periodically from *crontab*. Also, *ypserv* uses *ypxfr* as its transfer agent when it is asked to transfer a map.

yppush(8) — describes a tool to administer a running *yp* system. It is run 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 the results for each server.

ypset(8) — 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(8) — asks any *ypserv* process for the information it holds internally about a single map.

ypcat(1) — dumps the contents of a *yp* map. Use it when you don't care which server's version you are seeing. If you need to see a particular server's map, *rlogin* to that server (or use *rsh*) and use *makedbm*.

ypmatch(1) — 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(1) — 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.

yppasswd(1) — changes (or installs) yellow pages passwords by passing requests to *yppasswdd*(8c). *yppasswdd*(8c) is a server that changes password and password-restriction entries. It should be run only from the machine used as a master server for the */etc/passwd* file.

Installing and Administering the Yellow Pages

Twelve installation and administration topics are covered:

- Setting up a master *yp* server
- Altering a *yp* client's database to use *yp* services
- Setting up a slave *yp* server
- Setting up a *yp* client
- Modifying individual *yp* maps after *yp* installation
- Propagating a *yp* map
- Making new *yp* maps after *yp* installation
- Adding a new *yp* server not in the original set of *yp* servers
- Changing the master server to a different machine
- Adding new users to the *yp* environment

- Installing password-restriction maps
- *yp sendmail* support
- What if you do not use *yp*?

How to Set Up a Master *yp* Server

To create a new master server, use the following procedure:

1. Become superuser and change your current directory to */etc/yp*.
2. Make sure the following files in */etc* are complete and reflect an up-to-date picture of your system: *passwd*, *hosts*, *ethers*, *group*, *networks*, *protocols*, *pwrestrict*, and *services*. Also, if you know how */etc/netgroup* is going to be set up, do that before running *ypinit*. If you don't know, *ypinit* creates an empty "netgroup" map.
3. Set the domain name by entering:

   # **/bin/domainname** *domain_name_here*

4. Run *ypinit*(8) with the *-m* switch. You are asked whether you want the procedure to die at the first non-fatal error (in that case, you can fix the problem and restart *ypinit* — this is recommended if you haven't done the procedure before), or to continue despite non-fatal errors. In this second case, you can try to fix all the problems by hand, or fix some, and then restart *ypinit*. *ypinit* prompts you for a list of other hosts that also are *yp* servers. (Initially, this is the set of *yp* slave servers, but at some future time any of them might become the *yp* master server.) You need not add any other hosts now, but if you know that you will be setting up some more *yp* servers, add them now. You will save yourself some work later, and there is no runtime penalty for doing it. (Don't name every host in the net, however.)
5. The line used to set the hostname in */etc/rc.local* is:

   ```
   /bin/hostname host_name_here
   ```

   Add the following line just *after* the hostname line:

   ```
   /bin/domainname domain_name_here
   ```

6. Check */etc/rc.local* to make sure that */etc/portmap* has been invoked. Look for lines like these:

   ```
   if [ -f /etc/portmap]; then
         /etc/portmap & echo -n 'portmap'        >/dev/console
   fi
   ```

   If you don't find these lines, add them immediately after the lines invoking *ifconfig*, as follows:

```
/etc/ifconfig ex0 convex arp trailers up              >/dev/console
if [ -f /etc/portmap]; then
        /etc/portmap & echo -n 'portmap'        >/dev/console
fi
```

7. Check to make sure that */etc/portmap* is running. A good way to do this is to use *grep* as follows:

   # ps ax | grep portmap

   If */etc/portmap* is not running, start it via:

   # /etc/portmap

   Next, reconfigure */etc/inetd* as follows:

   # ps ax | grep inetd
   # kill -1 [*process number found using the previous instruction*]

8. Add the following lines to */etc/rc.local* immediately after the lines used to start */etc/portmap* (see Step 6):

```
if [-f /usr/etc/ypserv -a -d /etc/yp/`domainname`]; then
  /usr/etc/ypserv & echo -n 'ypserv'   >/dev/console
fi

if [-f /etc/ypbind]; then
  /etc/ypbind & echo -n 'ypbind'>/dev/console
fi
```

   These lines start the *ypserv* and *ypbind* daemons.

For security reasons, you may 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 someplace other than */etc/passwd* and 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.

To start providing yellow pages services, invoke */usr/etc/ypserv* and */etc/ypbind*. They are started automatically from */etc/rc.local* on later reboots.

Altering a *yp* Client's Files to Use *yp* Services

Once the decision has been made to serve a database with the *yp*, it is desirable that all nodes in the net access the *yp*'s version of the information, rather than the potentially out-of-date information in their local files. That policy is enforced by running a *ypbind* process on the client node (including nodes that may be running *yp* servers) and by abbreviating or eliminating the files that traditionally start the database. The files in question are: */etc/passwd*, */etc/hosts*, */etc/ethers*, */etc/group*, */etc/networks*, */etc/protocols*, */etc/rpc*,/*etc/services*, */etc/netgroup*, */etc/hosts.equiv*, and */.rhosts*. The treatment of each file is discussed in this section.

/etc/networks, */etc/protocols*, */etc/ethers*, */etc/services*, */etc/rpc*, */etc/pwrestrict*, and */etc/netgroup* need not exist at any *yp* client node. If you are squeamish about removing them, move them to backup names; for instance, on a machine named *ypclient*:

```
ypclient# cd /etc
ypclient# mv networks networks-
ypclient# <The rest of the renames, similarly>
```

/etc/hosts.equiv

/etc/hosts.equiv is not normally served by the *yp*. You can, however, add escape sequences to start the *yp*. This reduces problems with *rlogin* or *rsh* sometimes caused by different */etc/hosts.equiv* files on the two machines.

To let anyone with an account log on to a machine, */etc/hosts.equiv* may be edited to contain a single line, with only the character "+" (plus) on it. A line with only a "+" means that all further entries are retrieved from the *yp* rather than the local file. Alternatively, more control may be exercised over logins by using lines of the form:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

Each of the names to the right of the "@" (at) character is assumed to be a netgroup name, defined in the global netgroup database. The netgroup database is served by *yp*. (Netgroups are discussed later in this chapter under the heading "Netgroups: Network-Wide Groups of Machines and Users.") Note that new entries are read from the top down. For this reason, make sure to add the *most* exclusionary (or inclusionary) entry *last*. If none of these escape sequences is used, only the entries in */etc/passwd* are used; *yp* is not used.

/.rhosts

/.rhosts is not normally served by the *yp*, either. Its format is identical to that of */etc/hosts.equiv*. Because this file controls remote root access to the local machine, however, unrestricted access is not recommended. Make the list of trusted hosts explicit, or use netgroup names for the same purpose.

/etc/hosts

/etc/hosts must contain entries for the local host's name, and the local loopback name. These are accessed at boot time when the *yp* service is not yet available. After the system is running, and after the *ypbind* process is up, the */etc/hosts* file is not accessed at all. An example of the hosts file for *yp* client *zippy* is:

```
127.1           localhost
100.0.0.1       zippy  # John Q.  Random
```

/etc/passwd

All */etc/passwd* **must** contain root entries. (Including root entries ensures that you can log in even if *ypbind* or some other *yp* utility breaks.) */Etc/passwd* entries should also contain both an escape entry to force the use of the *yp* service and entries for the primary users of the machine. CONVEX recommends a few additional entries: *daemon*, to allow file-transfer utilities to work; *sync*, to run *sync* on a screwed-up machine before rebooting; and *operator*, to let a dump operator login. A sample *yp* client's */etc/passwd* file looks like:

```
root:wAm0Y4lEnf6:0:10:God:/:/bin/csh
jrandom:uHP1gQ2:1429:10:J Random:/usr2/jrandom:/bin/csh
operator:VyZr6V9:333:20:sys op:/usr2/operator:/bin/csh
daemon:*:1:1::/:
sync::1:1::/:/bin/sync
+::0:0:::
```

The last line informs the library routines to include entries from the */etc/passwd* file on the master *yp* server. Entries that exist in */etc/passwd* mask analogous entries in the *yp* maps. (Knowing this, you can change encrypted passwords, home directories, or shells used by individuals on the local machine simply by including the appropriate entry in the local file. You cannot change group or user IDs, however.) Note also that earlier entries in the file mask later ones with the same user name, or the same uid. For this reason, the order of the `daemon` and `sync` entries (that use the same uid) is important. Duplicate this order in your own file.

Note that you can use the +/- notation to include or exclude individual logins. If, for example, you want to enable logins for user *curly*, but not his netgroup *stooges*, you add the following lines to */etc/hosts.equiv*:

```
+curly
-estooges
```

You can exclude *curly* from logging in on a machine used by his cohorts by using the opposite notation:

```
-curly
+estooges
```

Note that the order of the new entries is important. The following entry, for example:

```
+curly
-estooges
```

works fine, but its opposite:

```
-estooges
+curly
```

does not. As with */etc/hosts.equiv*, new entries are read from the top down. For this reason, make sure to add the *most* exclusionary (or inclusionary) entry *last*. If none of these escape sequences is used, only the entries in */etc/passwd* are used; *yp* is not used.

/etc/group

/etc/group may be reduced to a single line:

```
+:
```

that forces translation of group names and ids to be made via the *yp* service. This approach is recommended.

How to Set Up a Slave *yp* Server

The network must be working to set up a slave *yp* server — in particular, you must be able to *rcp* files from the master *yp* server to *yp* slaves (and *ypserv*(8) should be running too).

To create a new slave server, first work through the procedure listed in "How to Set Up a Master *yp* Server." Then, change directory to */etc/yp*. From there, run *ypinit*(8) with the **-s** switch and name a host already set up as a *yp* server, as the master. Ideally, the named host really is the master server, but it can be any host that has its *yp* database set up. The host must be reachable. You must be superuser when you run *ypinit*. The default domain name on the machine intended to be the *yp* slave server must be set up, and must be set to the same domain name as the default domain name on the machine named as the master. Also, an entry for "daemon" must exist in the */etc/passwd* files of both slave and master, and that entry must precede any other entries that have the same uid. You won't be prompted for a list of other servers, but you have the opportunity to choose whether the procedure gives up at the first non-fatal error.

After running *ypinit*, make copies of */etc/passwd*, */etc/hosts*, */etc/group*, */etc/networks*, */etc/protocols*, */etc/netgroup*, */etc/pwrestrict*, and */etc/services*. For instance on a machine named "ypslave":

```
ypslave# cp /etc/passwd /etc/passwd-
```

Edit the original files as per the section above on altering the client's database. This ensures that processes on the slave *yp* server make use of the *yp* services, rather than the local ASCII files. (That is, make sure the *yp* slave server is also a *yp* client). Make backup copies of the edited files, as well. For instance:

```
ypslave# cp /etc/passwd /etc/passwd+
```

After the *yp* database gets set up by *ypinit*, invoke */usr/etc/ypserv* and */etc/ypbind* to begin supplying *yp* services. On later reboots, it starts automatically from */etc/rc.local*.

How to Set Up a *yp* Client

To set up a *yp* client, use the procedure included here:

1. The line used to set the hostname in */etc/rc.local* is:

   ```
   /bin/hostname host_name_here
   ```

   Add the following line just *after* the hostname line:

   ```
   /bin/domainname domain_name_here
   ```

2. Set the domain name via:

   ```
   # /bin/domainname domain_name_here
   ```

3. Check */etc/rc.local* to make sure that */etc/portmap* has been invoked. Look for lines like these:

```
if [ -f /etc/portmap]; then
      /etc/portmap  & echo -n 'portmap'              >/dev/console
fi
```

4. Check to make sure that */etc/portmap* is running. Use *grep* as follows:

   # ps ax | grep portmap

   If */etc/portmap* is not running, start it via:

   # /etc/portmap

   Then, reconfigure */etc/inetd* as follows:

   # ps ax | grep inetd
   # kill -1 [*the process number found using the previous instruction*]

5. Finally, add the following lines to */etc/rc.local* (these lines automatically start *ypbind* after the next reboot). These lines should be added immediately after the lines used to start */etc/portmap* (see Step 2):

```
if [-f /etc/ypbind]; then
  /etc/ypbind & echo -n 'ypbind'>/dev/console
fi
```

   To start */etc/ypbind* immediately, without waiting for the next system reboot, enter:

   % /etc/ypbind

   Note that the *ypserv* daemon is *not* started when clients are set up.

With the ASCII databases of */etc* abbreviated and */etc/ypbind* running, the processes on the machine start running as clients of the *yp* services. At this point, a *yp* server must be available; all sorts of stuff hangs if no *yp* server is available while *ypbind* is running. Note the possible alterations to the client's */etc* database as discussed above in the section on altering the client. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force inclusion and exclusion of data from the *yp* databases are found in the following man pages: *passwd*(5), *hosts*(5), *netgroup*(5), *host.equiv*(5), *group*(5), *pwrestrict*(5). 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*(1).

How to Modify Existing *yp* Maps After *yp* Installation

Databases served by the *yp* must be changed **on the master server**. The databases expected to change most frequently, like */etc/passwd*, may be changed by first editing the ASCII file, and then running *make*(1) on */etc/yp/Makefile* — also see *ypmake*(8).

Non-standard databases (that is, databases that are specific to the applications of a particular vendor or site, but that are not part of this release), or databases that are expected to change rarely, or databases for which no ASCII form exists (for example, databases not around before the *yp*), may be modified "manually." The general procedure is to use *makedbm*(8) with the *-u* switch to disassemble them into a form that can be modified using standard tools (such as *awk*, *sed*, or *vi*). Then build a new version again using *makedbm*(8). This may be done by hand in two ways:

1. The output of *makedbm* can be redirected to a temporary file that can be modified and then fed back into *makedbm*, or

2. The output of *makedbm* can be operated on within a pipeline that feeds into *makedbm* again directly. This is appropriate if the disassembled map can be updated by modifying it with *awk*, *sed*, or a *cat* append, for instance.

Suppose we want to create a non-standard *yp* map, called *mymap*. We want it to consist of key-value pairs in which the keys are strings like al, bl, cl, etc. (the *l* is for "left"), and the values are ar, br, cr (the *r* is for "right"). There are two possible procedures to follow when creating new maps. In one we use an existing ASCII file as input; in the other we use standard input.

For example, suppose an ASCII file exists named */etc/yp/mymap.asc*, created with an editor or a shell script on our machine *ypmaster*. *home_domain* is the subdirectory where the map is located. We can create the *yp* map for this file by:

```
ypmaster# cd /etc/yp
ypmaster# makedbm mymap.asc home_domain/mymap
```

But at this point we notice the map really should have included another key-value pair: (dl, dr). We can make the change quite simply. In all situations like this, remember to change the map by first modifying the ASCII file. Changes made to the map, not also in the ASCII file, are lost. Make the change like this:

```
ypmaster# cd /etc/yp <if not already there>
ypmaster# <make editorial change to mymap.asc>
ypmaster# makedbm mymap.asc home_domain/mymap
```

When no original ASCII file exists, we can create the *yp* map from the keyboard by typing input like this (our machine name is *ypmaster*, and the default domain is *home_domain*):

```
ypmaster# cd /etc/yp
ypmaster# makedbm - home_domain/mymap
al ar
bl br
cl cr
^D
```

When you need to modify that map, you can use *makedbm* to create a temporary ASCII intermediate file that can be edited using standard tools. For instance:

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

At this point *mymap.temp* can be edited to contain the correct information. A new version of the database is created by the commands:

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

The preceding paragraphs explained how to use some tools, but in reality almost everything you have to do can be done by *ypinit*(8) and */etc/yp/Makefile*, 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 */etc/yp* or some other procedure—*Makefile* is one of many possible—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.

How to Propagate a *yp* Map

To propagate a map means to move it from place to place — in general, to move it from the master *yp* server to a slave *yp* server. Initially, *ypinit*(8) moves it, as described above in the section "How to Set Up a Slave *yp* Server." After a slave *yp* server has been initialized, updated maps are transferred from the master server by *ypxfr*(8). *ypxfr* may be run in three different ways: periodically by *cron*(8), by *ypserv*(8), and interactively by a user.

Using *cron* to Run *ypxfr*

Maps have differing rates of change; for instance *protocols.byname* may not change for months at a time, but *passwd.byname* may change several times a day in a large organization. You can set up *crontab*(5) entries to periodically run *ypxfr* at a rate appropriate for any map in your *yp* database. *ypxfr* contacts the master server and transfers the map only if the master's copy is more recent than the local copy.

To avoid a *crontab* entry for each map, several maps with about the same change characteristics can be grouped in a shell script, and the shell script can be run from */usr/lib/crontab*. Suggested groupings, mnemonically named, can be found in */etc/yp*: *ypxfr_1perhour*, *ypxfr_1perday*, and *ypxfr_2perday*. If the rates of change are inappropriate for your environment, these shell scripts can be easily modified or replaced.

These same shell scripts should be run at each *yp* slave server in the domain. Alter the exact time of execution from one server to another, to prevent the checking from bogging down the master. If you want the map transferred from some particular server, not the master, that can be specified (using *ypxfr*'s *-h* option) within the shell script. Finally, maps having uncommon change characteristics can be checked and transferred by explicit invocations of *ypxfr* within *crontab*.

Using *ypserv* to Run *ypxfr*

ypxfr also gets invoked by *ypserv*, responding to a "Transfer Map" request. That request is made as an *rpc* call from *yppush*(8). *yppush* is run on the master *yp* server. It enumerates the *yp* map "ypserver" to generate a list of *yp* servers in your domain. To each of the named *yp* servers, it sends a "Transfer Map" request. *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 *yppush* process with a summary status.

For CONVEX UNIX V6.0 and later releases, *ypserv* also kicks off *ypxfr* when it attempts to emulate the behavior of *ypserv* from early Sun releases; for instance, when a 2.x Sun master *ypserv* communicates directly with the CONVEX 6.0 *ypserv*.

In the cases mentioned above, *ypxfr*'s transfer attempts and the results can be captured in a log file. If */etc/yp/ypxfr.log* exists, results are appended 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.

Running *ypxfr* Interactively

In the third case, the user runs *ypxfr* as a command. Typically, one does this only in exceptional situations—for example when setting up a temporary *yp* server to create a test environment, or when trying to quickly get a *yp* server that has been out of service consistent with the other servers.

How to Make 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 mechanism has been described above in "Propagation of a *yp* Map." This section only describes the work required to get the proper mechanisms in place so the propagation works correctly. Things must be set up correctly on both the master and the slaves.

After deciding which *yp* server is the master of the map, you should modify */etc/yp/Makefile* on the master server so that the map can be conveniently rebuilt. The methods used to change this file are too varied to describe here, but typically a human-readable ASCII file is filtered through *awk*, *sed*, or *grep* to make it suitable for input to *makedbm*(8). Consult the existing *Makefile* as a source for programming examples. We urge you make use of the mechanisms already in place in */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 either in */usr/lib/crontab*, or in the *ypxfr* shell scripts mentioned in the previous section. To get an initial copy of the map, *ypxfr* can be run 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 *yp* servers, but not all, you see unpredictable behavior from client programs.

How to Add a New *yp* Server Not in the Original Set

To add a new *yp* slave server, start by modifying some maps on the master *yp* server. If the new server is a host that has not been a *yp* server before, the host's name must be added to the map *ypservers* in the default domain. The sequence for adding a server named *ypslave* to domain *home_domain* is:

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

Note that we display some commands above on two lines. You may type these as one long command (even if the line wraps on your screen), or you may escape the return and newline with a backslash, as shown here. You cannot, however, simply type half the command, press return, and type the second half.

The *yp* slave server's address should be in *hosts.byname*. If that's not true, edit */etc/hosts* and run *make*. Here, the commands should be:

```
ypmaster#  <edit /etc/hosts here>
ypmaster# cd /etc/yp
ypmaster# make hosts
```

The new slave *yp* server's databases should be set up by copying the databases from *yp* master server *ypmaster*. Remotely log in to the new *yp* slave and use *ypinit*(8) in the following way:

```
yps lave# cd /etc/yp
yps lave# ypinit -s ypmaster
```

Then complete the steps described above in the section "How to Set Up a Slave *yp* Server."

How to Change 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, it is not enough to use an existing copy at the new master, or to send a copy there with *ypxfr*. The key must be reassociated with the new master's name. If the map has an ASCII source file, it should be present in its current version at the new master. Remake the *yp* map (we'll call it *jokes.bypunchline*) locally with the sequence:

```
newmaster# cd /etc/yp
newmaster# make jokes.bypunchline
```

/etc/yp/Makefile must be set up correctly for this to work. If it isn't, do it now. This is also a good time to go back to the old master (if it will remain a *yp* server) and edit */etc/yp/Makefile* so that *jokes.bypunchline* is no longer made there—that is, comment out the section of *oldmaster:/etc/yp/Makefile* that made "jokes.bypunchline".

If the map only exists as a *dbm* data base, you can remake it 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:

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

After making the map on the new master, you must send a new copy of the map to the other (slave) *yp* servers. Don't use *yppush*, however; the other slaves try to get new copies from the old master, rather than the new one.

A typical method (you may find others) is to become superuser on the old master server and type:

```
oldmaster# /etc/yp/ypxfr -h newmaster jokes.bypunchline
```

Now you have a new copy on the old master, and now you may 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 get the new map, that names the new master as the current master.

If the method above fails, there is a cumbersome but sure-fire option. On each *yp* server machine, while superuser, execute the command shown just above. This certainly works, but should be considered the worst-case solution.

Adding a New User to the Yellow Pages Environment

To add a new user, use *nu*(8) according to the directions contained in the *CONVEX System Manager's Guide*. If you choose to use password restrictions, set up an */etc/pwrestrict* file on your machine. The *CONVEX System Manager's Guide* completely describes how to set and use password-restriction files. (Password files are also discussed.)

If you are adding the user to a master *yp* server, then move to the */etc/yp* directory and run *make* as follows:

```
# make
```

(Running *make* causes the changed password file to be propagated across *yp* network.) Note that if you add the user to to a slave server or to a client, the new entry is added only to the local */etc/passwd* file. (Instructions for changing password or password-restriction files *after* you have added a user are included in the next section.)

You can define the new user's environment on login in several ways. For example, you may give her a copy of such files as *.login* and *.cshrc* if she uses "/bin/csh", or *.profile* if she uses "/bin/sh". See the *csh*(1) and *sh*(1) pages in the *CONVEX UNIX Programmer's Manual* for discussion of these files; they can automatically set up the terminal and shell environment at each login.

If the new user is a member of any groups at your site, add her to */etc/group* as necessary—see *group*(5) and *groups*(1). Be sure to make the changes to the */etc/group* and */etc/netgroup* files on the master *yp* server if you run *yp*. You may also need to manually create home directories on other machines if your user's home directory is not networked. See *nu*(8) for more information.

Installing Password and Password-Restriction Maps

Obviously, to be effective a master yellow pages server must distribute its password and password-restriction files to other machines on the network. This distribution is accomplished through the use of maps created and propagated when you run *yppasswdd*(8c) according to the procedure outlined below. If you do not use *yppasswdd* to update and propagate the maps, */etc/passwd* and */etc/pwrestrict* are changed only locally. You should remember, therefore, to work through the procedure included here each time you change the password or password-restriction files on the master server.

Use the following procedure to update and propagate the maps.

1. Check to make sure that *ypbind* is running on the client. (The simplest way to do this is to use the command sequence **ps ax | grep yp**.)

2. Check the master server to make sure *ypserv*, *ypbind*, and *yppasswdd* are running. (You can use *ps* piped through *grep* just as you did on the client.)

3. Use the following command sequence to kill *yppasswdd* if it is running:

   ```
   # ps ax | grep yppasswdd
   # kill -9 [any processes found]
   ```

4. From */usr/etc*, execute the following command sequence:

   ```
   # rpc.yppasswdd /etc/passwd /etc/pwrestrict -m passwd pwrestrict
   ```

   where:

/etc/passwd and **/etc/pwrestrict**	are the local names of the password and password-restriction files.
passwd and **pwrestrict**	are the names of the password and password-restriction maps.

yp sendmail Support

By default, CONVEX computers use local */usr/lib/aliases* files, rather than a mapped version networked from a *yp* server. For most installations, this is the preferred alternative. You can, however, change the default by modifying the *sendmail.cf* file according to the following procedure:

1. Find and kill the *sendmail* deamons by executing the following command sequence:

   ```
   # ps ax | grep sendmail
   # kill -9 [any processes found]
   ```

2. Add the following lines to the "options" section of */usr/lib/sendmail.cf*:

   ```
   # Use yellow pages ''mail.aliases'' map file instead of /usr/lib/aliases file
   Oy
   ```

   After you have added these lines, the options section should resemble the following listing:

   ```
   ######################
   ###   Options   ###
   ######################

   # Use Yellow Pages "mail.aliases" map file instead of /usr/lib/aliases files
   Oy
   # location of alias file
   OA/usr/lib/aliases
   # default delivery mode (deliver in background)
   Odbackground
   # (don't) connect to "expensive" mailers
   #Oc
   # temporary file mode
   OF0644
   # default GID
   Og1
   # location of help file
   OH/usr/lib/sendmail.hf
   # log level
   OL9
   ```

3. Rebuild the *sendmail* database via the command sequence:

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

4. Start *sendmail* via the command sequence:

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

   Note that the system may print a diagnostic on the console after you complete this step. Messages are also printed when errors occur on peripherals or in the system.

These messages are collected on the SPU and written into the the *syslog*(8) file, typically found in */usr/spool/mqueue/syslog*.

Once you have made these changes, *sendmail* uses the master *yp* aliases file located on the master server. Note that you are not forced to run *yp* after you make this change; the system is smart enough to determine that *yp* is not running and use the local file. To resume use of the local */usr/lib/aliases* file, kill the *sendmail* deamon, remove the added lines, and rebuild the *sendmail* database.

What If You Do Not Use the Yellow Pages?

If you choose not to use *yp*, the procedure for bypassing the software implementation is simple. In the file */etc/rc.local* find the lines that look like:

```
if [ -f /etc/ypbind ]; then
        /etc/ypbind; echo -n ' ypbind'   >/dev/console
fi
```

And comment them out:

```
# if [ -f /etc/ypbind ]; then
#         /etc/ypbind; echo -n ' ypbind' >/dev/console
# fi
```

If you don't find these lines in your *etc/rc.local* file, don't worry. Just use *ps* to kill *ypbind* and its related processes. (Use **ps ax | grep ypbind.**)

Debugging a Yellow Pages Client

We divide this debugging section into two parts—first those problems seen on a *yp* client, and then those problems seen on a *yp* server.

Before trying to debug a yellow pages client, read the earlier section in this chapter on how *yp* works.

When Commands Hang

The most common problem at a *yp* client node is for a command to hang and generate console messages that say:

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

Sometimes many commands begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above shows that *ypbind* on the local machine is unable to communicate with *ypserv* in the domain *wigwam*. This often happens when machines that run *ypserv* have crashed. It may also occur if the net or the *yp* server machine is so overloaded that *ypserv* can't get a response back to your *ypbind* within the time-out period. Under these circumstances, all the other *yp* client nodes on your net show the same or similar problems. The condition is usually temporary, and the messages usually go away when the *yp* server machine reboots and *ypserv* gets back in business, or when the load on the *yp* server nodes or the Ethernet decreases.

In the circumstances described below, however, things never improve.

The *yp* client has not set, or has incorrectly set, *domainname* on the machine. Clients must use a domain name that the *yp* servers know. 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 */etc/rc.local*. When */etc/rc.local* fails to set, or incorrectly sets, *domainname*, you must do the following: become superuser on the machine in question, edit */etc/rc.local* to fix the *domainname* line with a proper domain name (this assures that the domain name is correct every time the machine boots), and set *domainname* "by hand" so it is fixed immediately:

```
# domainname good_domain_name
```

If your domain name is correct, make sure your local net has at least one *yp* server machine. You can only bind to a *ypserv* process on your local net, not on another accessible net. At least one *yp* server must be available 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 net has a *yp* server, make sure it is up and running. Check other machines on your local net. If several client machines have problems simultaneously, suspect a server problem. Find a client machine behaving normally and try the *ypwhich* command. If *ypwhich* never returns an answer, kill it and go to a terminal on the *yp* server machine. Type:

```
# ps ax | grep yp
```

and look for *ypserv* and *ypbind* processes. If the server's *ypbind* daemon is not running, start it by typing:

```
# /etc/ypbind
```

If a *ypserv* process is running, do a *ypwhich* on the *yp* server machine. If *ypwhich* returns no answer, *ypserv* has probably hung and should be restarted. Kill the existing *ypserv* process (you must be logged on as root) and start */usr/etc/ypserv*:

```
# kill -9 [some pid # from ps]
# /usr/etc/ypserv
```

If *ps* shows no *ypserv* process running, start one.

When *yp* Becomes Unavailable

When other machines on the network appear to be okay, but *yp* service becomes unavailable on your machine, many different symptoms may show up. Among them: some commands appear to operate correctly while others end, printing an error message about the unavailability of *yp*; some commands limp along in a backup-strategy-mode particular to the program involved; and some commands or daemons crash with obscure messages or no message at all. For example, things like the following may show up:

```
my_machine% ypcat myfile
ypcat: can't bind to yp server for domain <wigwam>.
        Reason: can't communicate with ypbind.

my_machine% /etc/yp/yppoll myfile
Sorry, I can't make use of the yellow pages.  I give up.
```

When symptoms like those above occur, try:

```
my_machine% ls -l
```

on a directory containing files owned by many users, including users not in the local machine's */etc/passwd* file—for example */usr*. If the *ls -l* reports file owners not in the local machine's */etc/passwd* file as numbers, rather than names, it is one more symptom that *yp* service is not working.

These symptoms usually mean that your *ypbind* process is not running. You can do a *ps ax* to check for one. If you do not find it, type:

```
my_machine# /etc/ypbind
```

to start it. *yp* problems should disappear.

When *ypbind* Crashes

If *ypbind* crashes almost immediately each time it is started, you should look for a problem in some other part of the system. Check for the presence of the *portmap* daemon by typing:

```
my_machine%  ps ax | grep portmap
```

If *portmap* itself does not stay up or behaves strangely, look for more fundamental problems. Check the network software in the ways suggested in the section on *Network Management* in the *CONVEX System Manager's Guide.*

You may be able to talk to the *portmap* on your machine from a machine operating normally. From such a machine, type:

```
flipper% rpcinfo -p your_machine_name
```

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

```
program      vers   proto  port
100007       2      tcp    1030  ypbind
100007       2      udp    1030  ypbind
100007       1      tcp    1030  ypbind
100004       2      udp    1033  ypserv
100007       1      udp    1030  ypbind
100004       2      tcp    1031  ypserv
100004       1      udp    1033  ypserv
100004       1      tcp    1031  ypserv
100003       2      udp    2049  nfs
100012       1      udp    1069  sprayd
100011       1      udp    1071  rquotad
100005       1      udp    1073  mountd
100008       1      udp    1075  walld
100002       1      udp    1077  rusersd
100002       2      udp    1077  rusersd
100001       1      udp    1080  rstatd
100001       2      udp    1080  rstatd
100001       3      udp    1080  rstatd
```

On your machine, the port numbers are different. The two entries that represent the *ypbind* process are:

```
[100007, 1, port_#]
[100007, 2, port_#]
```

If they are not there, *ypbind* has been unable to register its services. In this case, use the following procedure. (Be sure to check port numbers for evidence of *ypbind* after each step. If you find that *ypbind* is running, stop working through the procedure.)

1. Restart *portmap* and the various *rpc* services (*nfsd, biod, ypbind, rpc.mountd, rpc.rquotad, rpc.rstatd, rpc.ruserd, rpc.rwalld, rpc.ypasswd,* and *rpc.sprayd.*) If you're using *inetd* to start these daemons, you need to restart *inetd.*

2. Reboot the machine.

3. Call the CONVEX Technical Assistance Center.

When *ypwhich* Becomes Inconsistent

When you use *ypwhich* several times at the same client node, the answer you get back varies—the *yp* server changes. This is normal. The binding of *yp* client to *yp* server changes over time on a busy net, 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 doesn't matter where the service comes from. Often a *yp* server machine gets its own *yp* services from another *yp* server on the net.

Debugging a Yellow Pages Server

Before trying to debug a yellow pages server, read the earlier section in this chapter on how *yp* works.

When Different Versions of a *yp* Map Exist

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

Most commonly, normal update is prevented when some *yp* server or some gateway machine between *yp* servers is down during a map transfer attempt. (Normal update is described in the section above, "Propagation of a *yp* Map"). 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 understand and fix the problem. If *ypxfr* succeeds, but you believe it fails sometimes, create a log file to enable logging of messages:

```
ypslave# cd /etc/yp
ypslave# touch ypxfr.log
```

This saves all output from *ypxfr*. The output looks much like what *ypxfr* creates when run interactively, but each line in the log file is timestamped. (You may see funny orderings in the timestamps. That's OK—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. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it grows without limit.

While still logged in to the problem *yp* slave server, inspect */usr/lib/crontab* 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 that the *yp* slave server is in the map *ypservers* within the domain. If it's not, it still works fine as a server, but *yppush* does not tell it when a new copy of a map exists.

If the problem is not obvious, you can work around it while you debug by using *rcp*(1) or *ftp*(1) to copy a recent version from any healthy *yp* server. You may not be able to do this as root, but you can probably do it as daemon. For instance, to transfer map *busted*:

```
ypslave# chmod go+w /etc/yp/mydomain
ypslave# su daemon
$ rcp ypmaster:/etc/yp/mydomain/busted.\* /etc/yp/mydomain
$ ^D
ypslave# chown root /etc/yp/mydomain/busted.*
ypslave# chmod go-w /etc/yp/mydomain
```

Notice that the "*" character has been escaped in the command line, so that it is expanded on *ypmaster*, instead of locally on *ypslave*. Also notice that the map files should be owned by root, so you must change ownership of them after the transfer. Obviously, if you can do the *rcp* as root, it makes the whole thing easier.

When *ypserv* Crashes

When the *ypserv* process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in the section "When *ypbind* Crashes." Check for the *portmap* daemon:

```
ypserver% ps ax | grep portmap
```

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

```
ypserver% /usr/etc/rpcinfo -p
```

and look for output to the screen like:

```
program     vers   proto  port
100007      2      tcp    1030  ypbind
100007      2      udp    1030  ypbind
100007      1      tcp    1030  ypbind
100004      2      udp    1033  ypserv
100007      1      udp    1030  ypbind
100004      2      tcp    1031  ypserv
100004      1      udp    1033  ypserv
100004      1      tcp    1031  ypserv
100003      2      udp    2049  nfs
100012      1      udp    1069  sprayd
100011      1      udp    1071  rquotad
```

```
100005      1       udp    1073  mountd
100008      1       udp    1075  walld
100002      1       udp    1077  rusersd
100002      2       udp    1077  rusersd
100001      1       udp    1080  rstatd
100001      2       udp    1080  rstatd
100001      3       udp    1080  rstatd
```

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

```
[100004, 1, port_#]
[100004, 2, port_#]
```

If they are not there, *ypserv* has been unable to register its services. Reboot the machine. If they are there, and they change each time you try to restart */etc/ypserv*, reboot the machine. If the situation persists after reboot, call for help. See the section "When *ypbind* Crashes".

Yellow Pages Policies

Here are the policies set by the C library routines when they access the following files on a system running *yp*.

/etc/passwd — Always consulted. If there are + or - entries, the *yp* password map is consulted; otherwise *yp* is unused.

/etc/group — Always consulted. If there are + or - entries, the *yp* group map is consulted; otherwise *yp* is unused.

/etc/hosts.equiv — (And similarly for *.rhosts*) Always consulted, though neither of these files is in *yp* database. (See the section below "How Security Is Changed With the Yellow Pages" for a further 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/pwrestrict — Always consulted. If there are + or - entries, the *pwrestrict* 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/rpc — 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/rc.local* file). After that the *yp* is used instead.

How Security Is Changed With the Yellow Pages

Read the section above on *yp* accessing policies to better understand *yp* security issues.

For more details, see the following manual pages: *yppasswd*(1), *hosts.equiv*(5), *export*(5), *passwd*(5), *pwrestrict*(5), *group*(5), *netgroup*(5), and *yppasswdd*(8c).

Global and Local *yp* Database Files

Of the *yp* databases, six were formerly in */etc*: */etc/passwd*, */etc/group*, */etc/hosts*, */etc/networks*, */etc/services*, and */etc/protocols*. The four new files are: */etc/netgroup*, */etc/pwrestrict*, */etc/rpc*, and */etc/ethers*. (Note that a site may add database files of its own.) *yp* is divided into local and global file types. A local file is first checked for on your own machine, and then in the yellow pages. A global file is only checked for in *yp*. */etc/passwd*, */etc/group*, and */etc/pwrestrict* are the local files in the *yp* database. The other seven *yp* files are global.

For example, a program that calls */etc/passwd* (a local file) first looks in the password file on your machine; *yp* password file is only consulted if your machine's password file contains "+" (plus sign) entries. The */etc/passwd* file is local so that you can control the entries for your own machine. The two other local files are */etc/group* and */etc/pwrestrict*. To repeat, local files are consulted first on your own machine, before looking in *yp*.

The remaining *yp* files (*hosts*, *networks*, *ethers*, *rpc*, *services*, *protocols*, and *netgroup*) are global files. The information in these files is network-wide data and is accessed only from *yp*. When booting, however, each machine needs an entry in */etc/hosts* for itself. In summary, if *yp* is running, global files are only checked in *yp*; a file on your local machine is not consulted.

Two Other Files *yp* Consults

The files */etc/hosts.equiv* and */.rhosts* are not in the *yp* database. Each machine has its own copy. It is possible, however, to put entries in your */etc/hosts.equiv* file that refer to *yp*. For example, a line consisting of

```
+eengineering
```

includes all members of *engineering* as it is defined in the local file */etc/netgroup* or in the *yp* database. A line consisting only of "+" (a plus sign) includes everyone.

Security Implications

Recall that to be able to remotely log in to a machine without having a password(via *rlogin*), you need to be in both the */etc/hosts.equiv* file and the */etc/passwd* file. By having a "+" entry in */etc/hosts.equiv*, you effectively bypass this check, and anyone in your */etc/passwd* file is allowed to r*login* to your machine without restriction.

The */etc/passwd*, */etc/pwrestrict*, and */etc/group* files may also have "+" entries. A line in an */etc/passwd* file such as

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

pulls in an entry for *nb* from *yp*. It gets the *uid*, *gid*, and *password* from *yp*, and gets the Gecos, home directory, and default shell from the "+" entry itself. On the other hand, an */etc/passwd* entry such as

```
+nb:
```

gets all information from *yp*. Finally, notice that:

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

is different from

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

In the first of the two examples, the password field is obtained from *yp*; in the second, 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.

Netgroups: Network-Wide Groups of Machines and Users

Netgroups are network-wide groups of machines and users defined in the */etc/netgroup* file on the master *yp* server. These groups are used for permission checking during remote mount, login, remote login, and remote shell.

The master *yp* server uses */etc/netgroup* to generate three *yp* netgroup maps in the */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* netgroup maps are *login*(1), *mountd*(8C), *rlogin*(1C), and *rsh*(1C). *Login* consults them for user classifications if it encounters netgroup names in */etc/passwd*(5). *mountd* consults them for machine classifications if it encounters netgroup names in */etc/exports*(5). *Rlogin* and *rsh* consult the *netgroup* map for both machine and user classifications if they encounter netgroup names in */etc/hosts.equiv*(5) or *.rhosts*.

Here is a sample */etc/netgroup* file. See *netgroup*(5) 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 hardware.
#
engineering     hardware software
hardware        (mercury,alan,convex) (venus,beth,convex) (testing,-,convex)
software        (earth,chris,convex) (mars,deborah,convex) (-,eric,convex)
#
# Marketing: Time-sharing on jupiter
#
marketing       (jupiter,fran,convex) (jupiter,greg,convex) (jupiter,dan,convex)
#
# Others
#
allusers (-,,convex)
allhosts (,-,convex)
```

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)

And here is how the machines are 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)

A

Updating */etc/rc.local*

This appendix describes the daemons that must be started and the order in which they must be started in the */etc/rc.local* startup file. You must edit the */etc/rc.local* startup file to become an *nfs* or yellow pages (*yp*) client/server.

Prerequisite Information

If you have just installed *nfs* on your system, you will probably not want to start the *ypbind* or *ypserv* programs at this point because the installation for *yp* is more detailed than for *nfs*. You must, however, start the */etc/portmap* program at this point. The *rpc* daemons in */usr/etc* rely on the */etc/portmap* program, so you must start the */etc/portmap* program regardless of whether you intend to be an *nfs* or *yp* client/server.

Do not use your domain name if you do not intend to run yellow pages (*yp*).

The order in which the daemons are started is very important. Follow the order in which the daemons are started in the sample */etc/rc.local*.

Sample */etc/rc.local* File

In this sample */etc/rc.local* file, *nfs* additions are preceded by >>.

```
    /bin/hostname convexs
>>#
>># Only set the domainname if you start "ypbind" below.
>>#
>>/bin/domainname convex
    /etc/ifconfig ex0 'hostname' arp trailers up      >/dev/console
    #
    # Any manually added route entries should go here
    #
    /etc/route add dragon-net convexe 1               >/dev/console
>>echo -n 'starting rpc and net services:'             >/dev/console
>>#
>># Portmap must be started before any other network programs
>>#
>>if [ -f /etc/portmap ]; then
>>     /etc/portmap & echo -n ' portmap'              >/dev/console
>>fi
>>#
>># Only include the lines that start ypserv and ypbind if
>># this machine is to run yellow pages
```

```
>>#
>>if [ -f /usr/etc/ypserv -a -d /etc/yp/`domainname` ]; then
>>     /usr/etc/ypserv & echo -n ' ypserv'     >/dev/console
>>     /etc/ypbind & echo -n ' ypbind'         >/dev/console
>>fi
>>#
>># Biod is needed for this machine to be an NFS client
>># i.e., so that we can mount other machines' file systems
>>#
>>if [ -f /etc/biod ]; then
>>     /etc/biod 4 & echo -n ' biod'           >/dev/console
>>fi
>>echo '.'                                     >/dev/console
>>#
>># The "umount -at nfs" clears the rmtab on remote machines
>>#
>>/etc/umount -at nfs
>>/etc/mount -vat nfs                          >/dev/console
  echo -n 'check quotas: '                    >/dev/console
      /usr/etc/quotacheck -a
  echo 'done.'                                >/dev/console
  /usr/etc/quotaon -a
  echo -n 'local daemons:'                    >/dev/console
>>#
>># nfsd is needed for this machine to be an NFS server,
>># i.e., so that other machines can mount our file systems.
>>#
>># Be sure to create the /etc/exports file to list the
>># mountable partitions (see exports(5)).
>>#
>>if [ -f /etc/nfsd ]; then
>>     /etc/nfsd 4 & echo -n ' nfsd'           >/dev/console
>>fi
>>#
>># rpc.statd and rpc.lockd comprise the record lock manager.
>># If you wish to support System V compatible record locking,
>># you MUST start up these two daemons.
>>#
>>if [ -f /etc/rpc.statd ]; then
>>     /etc/rpc.statd & echo -n ' statd'       >/dev/console
>>fi
>>if [ -f /etc/rpc.lockd ]; then
>>     /etc/rpc.lockd & echo -n ' lockd'       >/dev/console
>>fi
  if [ -f /etc/stat ]; then
      /etc/stat & echo -n ' statistics'       >/dev/console
  fi
  if [ -f /etc/rwhod ]; then
      /etc/rwhod & echo -n ' rwhod'           >/dev/console
  fi
  if [ -f /usr/etc/queued ]; then
      /usr/etc/queued & echo -n ' queued'     >/dev/console
  fi
```

```
if [ -f /usr/etc/spd ]; then
    /usr/etc/spd & echo -n ' spd'          >/dev/console
fi
echo '.'                                   >/dev/console
```

B

Reporting Problems

Introduction

The *contact* utility is the recommended way to report software and documentation problems to the Technical Assistance Center (TAC). It is an interactive tool that prompts you for the information necessary to report a problem to the TAC.

You must have a UNIX-to-UNIX Communications Protocol (UUCP) connection to the TAC to use *contact*. A UUCP system allows communication between UNIX systems by either dial-up or hard-wired communication lines. See *uucp*(1) or the entry in *info*(1) (online information system) for more information.

You must know the name and version number of the product involved. If you do not know the version number of the program or utility you are having trouble with, use the *vers* command. The syntax for the command is

vers *filename*

where *filename* is the the full pathname of the program. If you don't know the full pathname of the program, type

which *program*

For more information on these commands, see *vers*(1) and *which*(1) in the *CONVEX UNIX Programmer's Manual*, Part I.

Information Required to Report a Problem

contact requires the following information:

1. Your name, title, phone number, and corporate name.
2. The name and version of the product involved. Use the *vers* command if you don't know the version number of the program or utility.
3. A short (1 line) summary of the problem.
4. A detailed description of the problem. Include source code and a stack backtrace whenever possible. (See *adb*(1) or *csd*(1) for information on obtaining stack backtraces.) The more information provided, the quicker your problem can be isolated and solved.
5. The priority of the problem. You are shown a list of six levels from which to select.
6. Instructions on how to reproduce the problem, including the command syntax used, any flags invoked, or anything else you attempted to make your program run.

7. Any other comments about the problem or files you wish to submit.

You will have a chance to review your report before you submit it. You can edit the report if you find an error in what you have typed. If you change your mind and don't want to submit the report, you can abort the *contact* session; the file is saved in your home directory in a file named *dead.report*.

The following figure is a sample *contact* session. User input is in bold lettering, and the system response is in constant-width lettering.

Figure B–1: Sample *contact* Session

```
%contact (RETURN)
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
> Margaret Atwood, systems programmer, 814-4444, University r
> of Chicago (RETURN)
> (CTRL-D)

Enter the name of the product involved
> CONVEX UNIX Programmer's Manual, Part I (RETURN)

Enter the version number (in the form X.X or X.X.X.X) of the product
> Revision 4.0 (RETURN)

Enter a short (1 line) summary of the problem
> The finger command manual page lists nonexistent bug (RETURN)

Enter a detailed description of the problem (^D to terminate)
> The finger(1) man page says, under the BUGS section, that "Only the first
line of the .project file is printed."  Happily, this is not true! (RETURN)
> (CTRL-D)

Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
> 4 (RETURN)

Enter the instructions by which the problem may be reproduced (^D to terminate)
> a) put more than one line in .project (RETURN)
> b) read the man page for finger(1) (RETURN)
> (CTRL-D)

Enter any comments that are applicable (^D to terminate) (RETURN)
> (CTRL-D)

Do you have any suggestions or comments on the documentation that you
referenced when you were trying to resolve your problem (for example,
additions, corrections organization, accessibility)? (^D to terminate)
> The man page should be updated. (RETURN)
> (CTRL-D)

Are there any files that should be included in this report (yes | no)?
> no (RETURN)

Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
> 3 (RETURN)

Problem report submitted.
%
```

Index

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CUSTOMER SERVICE
CONVEX Computer Corp.
P.O. Box 833851
Richardson, TX 75083-3851

(Fold Here Second)

(Tape or Staple)

Software Documentation

Index Enhancements

So that we can continue to provide better indexing in CONVEX documentation, please keep track of the words or phrases you look up in an index, but don't find. Then, list under which index entry you ultimately found the information you were seeking. You can mail one of these postage-paid forms to the CONVEX Software Documentation Department monthly, or you can submit the information to the Technical Assistance Center in the form of a bug report. You can get more forms by writing to CONVEX at the address below, or by calling us. You can also photocopy this form and mail it back in an envelope. Thank you for helping us to serve you better.

Name:_____ **Company:**_____

Phone:_____ **Date:**_____

Manual Title/Rev. No.	Looked Up This Word	Found Information Under This Word

CONVEX Computer Corporation — P.O. Box 833851, Richardson, TX 75083-3851 (214) 952-0200

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CUSTOMER SERVICE
CONVEX Computer Corp.
P.O. Box 833851
Richardson, TX 75083-3851

(Fold Here Second)

(Tape or Staple)

CONVEX Network File System
System Manager's Guide
Document No. 710-001630-202, Second Edition, Rev. 1

Reader's Forum

You are invited to submit your comments concerning the clarity and service of this manual. Constructive critical comments are most welcome and will help us continue in our efforts to generate quality customer documentation. (Please list the page number for questions and comments):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION/DOCUMENTATION:

Location	Phone Number
In Texas	(214)952-0379
Other continental locations	1(800)952-0379
Outside continental U.S.	Contact local CONVEX office

Direct Mail Orders to:

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