CONVEX Network File System
System Manager's Guide

Document No. 710-001630-203

Second Edition, Rev. 2
November 1988

CONVEX Computer Corporation
Richardson, Texas

CONVEX Network File System
System Manager's Guide
Order No. DSW-113
Second Edition, Rev. 2

Printed in the United States of America

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

Edition	Document No.	Description
Second Rev. 2	710-001630-203	Released with CONVEX UNIX V7.0, November 1988. Includes the following changes: Updated Preface Chapter 2 corrected entries for */etc/exports*, */etc/rc.local*, and */etc/biod* corrected entry to include the *nfs* mount daemon in the */etc/inetd.conf* file corrected entries to enable over-the-net root access, read-only access, read-write access, and asynchronous operation Chapter 3 corrected entry to */etc/rc.local* to start */etc/portmap* updated sample output for */etc/portmap* updated sample output for */usr/etc/rpcinfo* updated "Yellow Pages Policies" section Chapter 4 added this chapter on installing and debugging NETdisk
Second Rev. 1	710-001630-202	Released with CONVEX UNIX V6.2, April 1988.
2.0	710-001630-201	Released with CONVEX UNIX V6.1, October 1987.
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
Within continental locations	1(800)952-0379
Outside continental U.S.	Contact local CONVEX office

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 -access=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:

```
mount     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 -a -f /etc/exports -a -f /usr/etc/exportfs ] ; then
        /etc/nfsd 4 & echo -n ' nfsd'                    >/dev/console
        >/etc/xtab; /usr/etc/exportfs -a &
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
# /usr/etc/exportfs -a
```

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
# /usr/etc/exportfs -a
```

- */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
 100000    2   tcp    111  portmapper
 100000    2   udp    111  portmapper
 100004    2   udp   1026  ypserv
 100004    2   tcp   1027  ypserv
 100004    1   udp   1026  ypserv
 100004    1   tcp   1027  ypserv
 100007    2   tcp   1028  ypbind
 100007    2   udp   1034  ypbind
 100007    1   tcp   1028  ypbind
 100007    1   udp   1034  ypbind
 100009    1   udp   1023  yppasswdd
 100003    2   udp   2049  nfs
 100002    1   udp   1120  rusersd
 100002    2   udp   1120  rusersd
 100001    1   udp   1126  rstatd
 100001    2   udp   1126  rstatd
 100001    3   udp   1126  rstatd
 100008    1   udp   1131  walld
 100005    1   udp   1134  mountd
 100011    1   udp   1137  rquotad
 100012    1   udp   1140  sprayd
 100017    1   tcp   1055  rexd
 100026    1   udp   1145  bootparam
```

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:

  ```
  mount     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 on a file-system-by-file-system basis (via changes to the */etc/exports* file). Before you consider using this method, 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 for particular file systems, add one of the flags (*-anon=0* or *-ro*) to the listings in */etc/exports*. Although the 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 by root, add the *-anon=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        -anon=0,access=convex2:convex3
/fonts              -access=convex4
/usr/spool          -access=convex2:convex3:convex4
```

- To enable over-the-net root access to particular hosts on the access list, use the *-root=hostname* command. The command systax is **-root**=hostname [:hostname] . In the following example, root access is given to convex2, but not to convex3.

```
/tmp
/sw
/mnt1
/usr        -root=convex2,access=convex2:convex3
```

- To enable read-only access: For each file system to be accessed read-only, add the *-ro* 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/src*. No user, including root, however, can *write* them. Note that although users can mount */usr/src* read/write, attempted writes by any user (including root) fail with the *EROFS* (Read-Only File System) error:

```
/tmp
/sw
/mnt1
/usr
/fonts
/usr/src  -ro,access=convex2:convex3
```

- To enable read-write access only on selected machines, use the *-rw=hostname [:hostname]* option. For example, the following lines specify that convex3 has read-write privileges, but not convex2.

```
/tmp
/sw
/mnt1
/usr
/fonts
/usr/src  -rw=convex3,access=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.

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 enable asynchronous operation on an export filesystem basis, add the *async* option in the */etc/exports* file. For example, adding the following lines to */etc/exports* enables asynchronous operation for */tmp* and */usr/spool*:

```
/tmp      -async
/usr/spool      -access=convex2:convex3:convex4,async
```

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 *bootcmd* on the SPU:

   ```
   tune cpu nfs_portmon = 1
   ```

2. Be sure that the */etc/inetd.conf* file includes the following line:

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

   This line ensures that *-n* option is not set in the *rpc.mountd* daemon.

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.)

CAUTION

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

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
sun#
```

Versions of *nfs* ported by other vendors may not be designed to support file systems with block sizes larger than 8 Kbytes. You may have difficulty using other types of hardware as a client for these file systems. 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.

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

```
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.

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
 100000    2   tcp    111  portmapper
 100000    2   udp    111  portmapper
 100004    2   udp   1026  ypserv
 100004    2   tcp   1027  ypserv
 100004    1   udp   1026  ypserv
 100004    1   tcp   1027  ypserv
 100007    2   tcp   1028  ypbind
 100007    2   udp   1034  ypbind
 100007    1   tcp   1028  ypbind
 100007    1   udp   1034  ypbind
 100009    1   udp   1023  yppasswdd
 100003    2   udp   2049  nfs
 100002    1   udp   1120  rusersd
 100002    2   udp   1120  rusersd
 100001    1   udp   1126  rstatd
 100001    2   udp   1126  rstatd
 100001    3   udp   1126  rstatd
 100008    1   udp   1131  walld
 100005    1   udp   1134  mountd
 100011    1   udp   1137  rquotad
 100012    1   udp   1140  sprayd
 100017    1   tcp   1055  rexd
 100026    1   udp   1145  bootparam
```

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
 100000    2   tcp    111  portmapper
 100000    2   udp    111  portmapper
 100004    2   udp   1026  ypserv
 100004    2   tcp   1027  ypserv
 100004    1   udp   1026  ypserv
 100004    1   tcp   1027  ypserv
 100007    2   tcp   1028  ypbind
 100007    2   udp   1034  ypbind
 100007    1   tcp   1028  ypbind
 100007    1   udp   1034  ypbind
 100009    1   udp   1023  yppasswdd
 100003    2   udp   2049  nfs
 100002    1   udp   1120  rusersd
 100002    2   udp   1120  rusersd
 100001    1   udp   1126  rstatd
 100001    2   udp   1126  rstatd
 100001    3   udp   1126  rstatd
 100008    1   udp   1131  walld
 100005    1   udp   1134  mountd
 100011    1   udp   1137  rquotad
 100012    1   udp   1140  sprayd
 100017    1   tcp   1055  rexd
 100026    1   udp   1145  bootparam
```

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 map.

/etc/protocols — Never consulted. The data that was formerly read from this file now comes from the *yp* protocols map.

/etc/networks — Never consulted. The data that was formerly read from this file now comes from the *yp* networks map.

/etc/netgroup — Never consulted. The data that was formerly read from this file now comes from the *yp* netgroup map.

/etc/rpc — Never consulted. The data that was formerly read from this file now comes from the *yp* rpc map.

/etc/ethers — Never consulted. The data read from this file comes from the *yp* rpc map.

/etc/hosts — Consulted only when booting (by the *ifconfig* command in the */etc/rc.local* file). After that the *yp* host map is used instead.

/etc/bootparams — Never consulted. The data from this file comes from the *yp* bootparams map.

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 new files are */etc/netgroup*, */etc/pwrestrict*, */etc/rpc*, */etc/ethers*, and */etc/bootparams*. (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 remaining *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

```
+@engineering
```

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)

4

Installing and Debugging NETdisk

The NETdisk product is included with all standard shipments of CONVEX *nfs*. NETdisk is a collection of administrative programs, combined with *nfs*. NETdisk allows a CONVEX server machine to support booting diskless workstations using *nfs*.

This chapter discusses NETdisk at a conceptual level and provides sample installation procedures. Specifically, this chapter

- addresses disk space requirements
- describes how to use the *INSTALL* program to install the client software
- describes how to boot a diskless client workstation
- describes how to add and remove clients
- describes how to install optional software after running *INSTALL*
- lists important files used by the NETdisk system
- describes the steps taken by a diskless client workstation when booting from a CONVEX *nfs* server

For further reference on NETdisk, see the *INSTALL*(8), *setup_client*(8), and *sunboot*(8s) manual pages.

The only clients currently supported are Sun-3 and Sun-4 workstations running SunOS 4.0 or later. Other workstations that use the same boot procedure as the Sun-3 and Sun-4 may be used (see the "Theory of Operation" later in this chapter for further information).

NOTE

Booting workstations other than Sun-3 and Sun-4 workstations from your CONVEX server is not supported by the CONVEX Technical Assistance Center. Contact your CONVEX sales representative if you wish to boot a Sun-2 system running SunOS 4.0 or later from your CONVEX server.

What Is NETdisk?

NETdisk is a collection of administrative programs loaded into the */usr/etc/install* directory when *nfs* is installed. NETdisk is used to boot diskless workstations (called clients) from a CONVEX *nfs* server.

NETdisk makes it possible to use relatively inexpensive diskless workstations without the need to purchase expensive boot or file servers. With NETdisk, you can use your CONVEX *nfs* server to boot a diskless client and to serve exclusively the files that it needs.

To boot a diskless workstation it is necessary to load the workstation manufacturer's software distribution tape on your CONVEX server. These tapes must be ordered from the workstation vendor. CONVEX recommends purchasing software distributions on 1600-bpi or 6250-bpi ½ inch magnetic tape. Using software distribution tapes lets you load the software without having to use the network to access a remote tape drive.

For example, the following software distribution formats are available from Sun Microsystems:

- ½ inch 1600-bpi magnetic tape
- ¼ inch QIC-24 format cartridge tape

These tapes can be extracted either remotely or locally, though a remote Sun computer must be used to read the ¼ inch cartridge tape format.

NETdisk Disk Space Requirements

The amount of disk space consumed per workstation depends on the following factors:

- The number of architectures loaded—for example, one CONVEX server can boot Sun-3 and Sun-4 workstations, even though the Sun-3 and Sun-4 use different binary formats. The *INSTALL* program loads the different software distributions into different directories, depending on the architecture type.

- The number of optional software packages loaded per architecture—each loaded architecture may be loaded with optional software packages; the more packages loaded, the larger the disk space consumption.

- The number of clients supported—each client workstation requires its own root directory.

- The size of the clients' swap files—each client workstation requires one or more files that it uses exclusively as a swap device. Disk space must be statically allocated for this swap file on the CONVEX server to be certain that a swap write request does not fail due to a disk partition filling up. The size of the swap file varies depending on the amount of memory and type of application the client workstation intends to run.

Following are approximate disk space values for a SunOS 4.0 release. Use these values to decide how much disk space to allocate for NETdisk.

architecture-shared executables (*/usr*):	58 megabytes
root directory per client workstation:	2 megabytes
swap file per client workstation:	12 megabytes

Thus, to support one Sun-3 workstation with all standard optional software loaded requires about 72 megabytes of disk space. Each additional client, however, only requires about 14 Megabytes. Supporting 10 diskless Sun-3 workstations would therefore require 198 Megabytes of disk storage (58 megabytes for */usr* and 140 megabytes for the root directories and swap files of the 10 diskless clients).

NOTE

CONVEX recommends using a separate file system to hold the NETdisk files.

CONVEX suggests that you create a new file system called */export* to hold the NETdisk files. Then use the following full pathnames:

/export/root/client-name root directory for client *client-name*

/export/swap/client-name swap file for client *client-name*

/export/exec/arch-name architecture (*/usr*) directory for architecture *arch-name*

Thus, the root directory for a Sun-4 SPARC architecture workstation named jogger would be */export/root/jogger*, the swap file would */export/swap/jogger*, and the architecture-shared SPARC binaries would live in the */export/exec/sun4* directory.

Remember that these disk space values are estimates. Your disk space requirements may differ depending on the size of swap files used and the amount of optional software installed.

Loading Client Software and Booting

Loading a Sun Microsystem's distribution tape and booting a Sun-3 client is discussed in this section.

Before Loading Software

Follow these steps before loading the software and booting a diskless workstation.

1. Identify the Ethernet address of the client workstation by powering up the monitor.

2. Use your favorite editor to add the Ethernet address to the */etc/ethers* file.

3. Execute the following *arp* command:

   /etc/arp -e /etc/ethers

   This command takes the client Ethernet address in the */etc/ethers* file and makes a permanent address entry in the kernel.

4. Add the following lines to the */etc/rc.local* file:

```
if [ -d /tftpboot ] ; then
    if [ -f /etc/ethers -a -f /etc/arp ] ; then
      /etc/arp -e /etc/ethers & echo -n ' rarp'      >/dev/console
    fi
fi
```

   In subsequent boots of the diskless client workstation, these lines in the */etc/rc.local* file will automatically map the client's Ethernet address to a permanent address entry in the kernel.

5. Include the host name and internet protocol (IP) address of the new client workstation (the one you are going to boot) to the */etc/hosts* file. For information on determining IP addresses, see the *CONVEX Networking Utilities System Manager's Guide.*

6. Set up the *exports* file system. See the *newfs*(8) man page for details on setting up a new file system.

Using *INSTALL* to Load Software

This section guides the superuser through a sample installation of the client architecture software. This sample installation assumes the following:

- You are using the local ½ inch magnetic tape drive to install a Sun-3 architecture.
- A diskless Sun-3 workstation named jogger is also configured and set up at the same time.
- The CONVEX machine is running in multiuser mode and the *nfs* system is working correctly.
- The SunOS 4.0 software distribution tape for the Sun-3 (68020 SUNBIN) has been loaded on the local CONVEX machine's tape unit 0.
- The CONVEX server machine is running the yellow pages (*yp*) system.
- The */export* file system has been set up and is large enough to hold the executables, root, and swap files.
- The */etc/hosts* file (or the *hosts* database if *yp* is running) lists the client as a known host.
- The */etc/ethers* file (or the *ether* map) has the client Ethernet address.
- The current directory " . " is in the PATH environment variable.
- You have superuser (root) privileges.

This procedure takes approximately two hours to complete. The program, however, does not have to be constantly monitored once you have selected the optional software you want installed. You may need to change tapes. The *INSTALL* program prompts you to do so, if necessary.

1. Log in as superuser.
2. Change your working directory to the */usr/etc/install* directory, as shown in the next screen.

```
# cd /usr/etc/install
```

   This directory contains all the administrative scripts needed to extract software distributions from tapes and to add and remove clients.

3. To invoke the *INSTALL* program, enter **INSTALL**, as shown in the next screen.

```
# INSTALL
```

4. The *INSTALL* program prompts you for the type of installation. Answer **local** as shown in the next screen.

```
        >>>     Sun Diskless Client Installation Tool   <<<
        >>>             (for non-Sun Servers)           <<<
        >>>                  Release 1.0                <<<

Enter tape drive type ? [local | remote]: local
```

5. The *INSTALL* program then asks which tape drive to use. For a local installation on ½ inch magnetic tape, use **/dev/rmt12** because it is a non-rewind tape device. For tape unit 1, use /dev/rmt13.

```
Enter tape type ? [ar[08] | st[08] | mt[08] | xt[08] | /dev/???]: /dev/rmt12
```

When using a local magnetic tape, you must specify the full pathname to a non-rewind tape drive. Do not use the shortcut **mt8** specification because the *INSTALL* program assumes it to be a Sun cartridge tape drive, which is unavailable on CONVEX machines.

If you had selected **remote** in the previous step, *INSTALL* would have prompted for the host name to which the remote tape is connected. Although not the case in this example, note that remote installations require that the remote host's */.rhosts* file contain the local CONVEX host name on a line by itself.

Also, when doing a remote installation using a ¼ inch QIC-24 cartridge tape from an already configured Sun system, the correct response at this point is **st8** to select the Sun QIC-24 cartridge tape drive.

6. Next enter the architecture type to load. Since this sample procedure assumes a Sun-3 architecture, the appropriate response is **sun3**, as illustrated in the next screen.

```
Enter next architecture type to load
    [ sun3 | sun4 | ... | continue | done ]: sun3
```

7. The *INSTALL* program asks where to load the executables. CONVEX strongly recommends loading these in the */export/exec* directory, as shown in the next screen.

```
Enter pathname for sun3 executables ? /export/exec
```

The directory */export/exec/sun3* is automatically created to hold the executables. If this had been a Sun-4 installation, the directory */export/exec/sun4* would be created instead.

8. The *INSTALL* program now tries to access the tape. Be sure the tape is mounted and online. Press (RETURN) when ready.

```
Reading the table of contents for sun3 architecture...

Mount a sun3 release tape and hit RETURN, or enter "exit" :
```

9. At this point, the *INSTALL* program reads a standard format-header file on the tape that describes the entire contents of the tapes comprising the release. The *INSTALL* program then prompts for confirmation to load each file on the tapes. No files are loaded at this time. Rather, an extraction list is created for automatic extraction at a later time.

   In this sample installation, all optional software is extracted except for the **Games** file, which will be used in a later exercise to add optional software after an initial installation.

```
Select optional software for the sun3 architecture:
Install "User File System" [20971520 bytes; required] ? [y/n]: y
Install "Sys" [2721792 bytes; desirable] ? [y/n]: y
Install "Networking tools and programs" [953344 bytes; desirable] ? [y/n]: y
Install "Debugging tools" [3383296 bytes; desirable] ? [y/n]: y
Install "SunView_Users Programs" [1453056 bytes; common] ? [y/n]: y
Install "SunView_Programmers Programs" [2064384 bytes; optional] ? [y/n]: y
Install "SunView_Demo Program source" [564224 bytes; optional] ? [y/n]: y
Install "Text Processing tools" [690176 bytes; optional] ? [y/n]: y
Install "Install tools" [1004544 bytes; optional] ? [y/n]: y
Install "User Level Diagnostics tools" [1491968 bytes; optional] ? [y/n]: y
Install "SunCore Libraries" [2991104 bytes; optional] ? [y/n]: y
Install "uucp programs" [270336 bytes; optional] ? [y/n]: y
Install "System V programs and libraries" [4481024 bytes; optional] ? [y/n]: y
Install "Manual Pages" [6072320 bytes; optional] ? [y/n]: y
Install "Demonstration Programs" [2790400 bytes; optional] ? [y/n]: y
Install "Games" [2468864 bytes; optional] ? [y/n]: n
Install "Versatec" [6117376 bytes; optional] ? [y/n]: y
Install "SunOs Security Features" [146432 bytes; optional] ? [y/n]: y
```

10. The *INSTALL* program asks you if you want to load shared objects in shared locations. This is only useful when loading more than one architecture type, such as loading both Sun-3 and Sun-4 architectures on the same CONVEX server. In this case, answer **n**, as shown in the next screen.

```
Load sharable objects in shared location ? [y/n] : n
```

    If you answer **y** to the previous prompt, the *INSTALL* program automatically loads shared object filess (e.g. man pages, nroff macro sets, etc) into a directory called */export/exec/share*. Then the directories */export/exec/sun3* and */export/exec/sun4* (for Sun-3 and Sun-4 architectures, respectively) are automatically created as links to */export/exec/share*.

By answering **n** to the prompt, you direct the *INSTALL* program to load the object files directly into the */export/exec/sun3/share* or */usr/export/exec/sun4/share* directories.

11. This completes the first phase of the procedure—loading the Sun-3 architecture. The *INSTALL* program supports loading more than one architecture at a time, so it returns to ask for the next architecture type to load. To proceed, respond with **done**, as shown in the next screen.

```
Enter next architecture type to load
    [ sun3 | sun4 | ... | continue | done ]: done
```

12. The next phase of the *INSTALL* procedure is to set up the root and swap files for any clients that are ready to be installed. You can skip this phase by specifying **done**. For the sake of illustration, however, specify the host name **jogger**, as shown in next screen.

```
Enter a sun3 client name ? [ name | done ]: jogger

Verifying ip address...
  192.18.44.130 jogger

Verifying ethernet address...
  8:0:20:0:f:ad jogger
```

The *INSTALL* program checks the */etc/hosts* file (or the *hosts* database if *yp* is running) to verify that jogger is a known host. *INSTALL* also checks the */etc/ethers* file (or the *ether* map) to determine the Ethernet address of jogger.

CAUTION

The *etc/hosts* and */etc/ethers* files must be updated prior to invoking the *INSTALL* program. If they are not, the program fails to find the needed information and prompts for another client host name. You can suspend the procedure or login from another terminal to fix the files and try again. Or you can simply enter **done** and add clients using the *setup_client* command after the *INSTALL* program has finished running.

13. The *INSTALL* program asks for the yellow pages type for the client. This sample procedure assumes that yellow pages is running on the CONVEX server, so the appropriate response is **client**.

```
Enter yp type of jogger ? [ master | slave | client | none ]: client
```

14. The *INSTALL* program next requests for information about jogger. You are prompted for the swap size, root directory, swap and dump file, and home directory. CONVEX suggests using the values listed in the next screen as initial default values. The swap size of 12 Megabytes can be easily increased later if necessary.

```
Enter swap size of jogger ? 12m

Enter root pathname of jogger ? /export/root

Enter swap pathname of jogger ? /export/swap

Enter dump pathname of jogger (or "none") ? none

Enter home pathname of jogger (or "none") ? none
```

In the previous screen, the "dump pathname" and "home pathname" queries are answered **none** because this sample installation does not use the */home* partition and the dump pathname defaults to the swap partition */export/swap*. The pathnames **/export/dump** and **/export/home** may also be used.

15. Next, you are asked to confirm that the information you've specified is correct and to enter a new client name (if you want to add another client). More clients may be added at this time simply by specifying another client host name and repeating the last three steps. In this sample procedure, no other clients are added.

```
Information for jogger ok ? [y/n] : y

Enter a sun3 client name ? [ name | done ]: done
```

If you answer **n** to the "Information for jogger ok" request, the program prompts you again to specify the file swap size, root pathname, swap pathname, and so on.

16. The *INSTALL* program prompts you to start installing the software. Enter **y** to begin the installation, as shown in the next screen. Installing all of the optional software takes approximately two hours—about one hour for each tape. The next prompt you see will be to load the second release tape.

```
Are you ready to start the installation ? [y/n] : y

Beginning Installation for the sun3 architecture.

Installation of sun3 executable files begins :

[Loading version 4.0 of sun3 architecture.]

Loading prototype root tree...

Extracting "usr" files from "/dev/rmt12" release tape.

Extracting "Sys" files from "/dev/rmt12" release tape.

Extracting "Networking" files from "/dev/rmt12" release tape.

Extracting "Debugging" files from "/dev/rmt12" release tape.

Extracting "SunView_Users" files from "/dev/rmt12" release tape.
```

17. After approximately an hour, the *INSTALL* procedure extracts the entire first tape. You are then prompted to load the second tape and press RETURN.

```
Tape loaded is #1

Load release tape #2 for architecture sun3 and hit <RETURN>:
```

The output appears as follows.

```
Extracting "SunView_Programmers" files from "/dev/rmt12" release tape.

Extracting "SunView_Demo" files from "/dev/rmt12" release tape.

Extracting "Text" files from "/dev/rmt12" release tape.

Extracting "Install" files from "/dev/rmt12" release tape.

Extracting "User_Diag" files from "/dev/rmt12" release tape.

Extracting "SunCore" files from "/dev/rmt12" release tape.

Extracting "uucp" files from "/dev/rmt12" release tape.

Extracting "System_V" files from "/dev/rmt12" release tape.

Extracting "Manual" files from "/dev/rmt12" release tape.

Extracting "Demo" files from "/dev/rmt12" release tape.

Extracting "Versatec" files from "/dev/rmt12" release tape.

Extracting "Security" files from "/dev/rmt12" release tape.

Installation of sun3 executable files completed.
```

18. The *INSTALL* program sets up the clients and updates the associated administrative files. The output appears as follows.

```
Starting installation of sun3 clients...


Start creating sun3 client "jogger" :

Updating bootparams ...

Creating root for client "jogger".

Creating 10m bytes of swap for client "jogger".

Setting up /tftpboot directory.

Completed creating sun3 client "jogger".

Updating bootparams YP map...
updated bootparams
pushed bootparams

Diskless Client Installation Completed.
```

19. The system prompt returns.

```
#
```

Booting a Sun

This section describes the steps required to boot jogger, the Sun workstation configured in the previous section. In general, booting is very simple and can be achieved by either cycling the power to the Sun workstation or by simply typing **b** to the Sun PROM monitor.

This sample boot procedure assumes the following:

- The Sun-3 has been connected to the Ethernet and is functional.
- The Sun-3 is turned on and the Sun PROM monitor prompt (>) is displayed.
- The Sun-3 architectural support programs and root directory for jogger have been loaded as described in the previous section.
- The Sun-3 is on the same subnet as the CONVEX server where the Sun-3 architectural support programs are loaded.
- The name of the CONVEX server is **conserve** and it is in the yellow pages domain **convex**.
- *nfs* on the CONVEX machine is working correctly, and the *rpc.bootparamd* server is running or installed in the */etc/inetd.conf* file.

The booting procedure is simple and requires only minimal input. Once the Sun has been instructed to boot, booting multiuser mode is completely automatic and quick. It should take less than five minutes to boot diskless Sun workstations.

1. Ensure that the Sun-3 has power and the following Sun PROM monitor prompt is displayed.

```
>
```

2. Enter **b** to boot the Sun directly to multiuser operation, as shown in the next screen.

```
> b
```

3. The boot process continues automatically. The Sun determines its internet protocol (IP) address and downloads a boot program from the CONVEX server **conserve** using *tftp*. The output appears as follows.

```
Using IP Address 192.18.44.130 = C0122C82
Boot: le(0,0,0)
Booting from tftp server at 192.18.44.122 = C0122C7A
Downloaded 126056 bytes from tftp server.
```

In the previous screen, the tftp server address 192.18.44.122 is the IP address of the Sun server responding to the tftp request to download the boot program. The address of 192.18.44.130 is the booting Sun's IP address determined from the server using Reverse ARP (RARP) protocols.

The C0122C82 address is the uppercase hex value of this IP address, where 192=C0, 18=12, 44=2C, and 130=82: thus, 192.18.44.130=C0122C82. The tftp protocol requests this uppercase hex string, which serves as the filename for the boot program, to download from the server.

4. The boot program is then executed. It again determines the correct IP address to use, connects to the *rpc.bootparamd* server on **conserve**, and uses the *nfs* protocols to download the *vmunix* kernel. The output appears as follows.

```
Using IP Address 192.18.44.130 = C0122C82
hostname: jogger
domainname: convex
server name 'conserve'
root pathname '/export/root/jogger'
root on conserve:/export/root/jogger fstype nfs
Boot: vmunix
Size: 632768+115120+16076 bytes
```

5. The next screen shows the output as the Sun kernel is executed and the system comes up in multiuser mode.

```
SunOS Release 4.0 (GENERIC) #5: Thu Apr 14 19:24:56 PDT 1988
Copyright (c) 1988 by Sun Microsystems, Inc.
mem = 4096K (0x400000)
avail mem = 2949120
Ethernet address = 8:0:20:0:f:ad
si0 at obio 0x140000 pri 2
sd0 at si0 slave 0
sd1 at si0 slave 1
sd2 at si0 slave 8
sd3 at si0 slave 9
st0 at si0 slave 32
st1 at si0 slave 40
zs0 at obio 0x20000 pri 3
zs1 at obio 0x0 pri 3
le0 at obio 0x120000 pri 3
bwtwo0 at obmem 0x100000 pri 4
bwtwo0: resolution 1152 x 900
hostname: jogger
domainname: convex
root on conserve:/export/root/jogger fstype nfs
swap on conserve:/export/swap/jogger fstype nfs size 10240K
dump on conserve:/export/swap/jogger fstype nfs
checking filesystems
Automatic reboot in progress...
Thu Sep  1 07:40:26 PDT 1988
checking quotas: done
starting rpc and net services: portmap ypbind keyserv routed.
mount: conserve:/export/exec/sun3 already mounted
rdate conserve
starting additional services: biod
starting system logger
starting local daemons: auditd sendmail statd lockd.
link-editor directory cache
preserving editor files
clearing /tmp
standard daemons: update cron uucp.
starting network daemons: inetd printer.
Thu Sep  1 07:41:22 PDT 1988
```

6. Finally, the system displays the login prompt for jogger.

```
jogger login:
```

Adding and Removing NETdisk Clients

This section describes using the *setup_client* program to add and remove NETdisk clients.

The *INSTALL* program may be used interactively to load client architecture support onto a CONVEX server. Once the client architecture support has been loaded, however, it may be more convenient to use the *setup_client* program to add new clients or remove existing ones. The *setup_client* program is also part of the */usr/etc/install* directory. For more information on the *setup_client* program, see the *setup_client*(8) man page.

The *setup_client* program is not an interactive one like *INSTALL*. Instead all information is specified on the command line and the process of adding and removing the client happens without confirmation. You can display the expected options to *setup_client* by entering **setup_client** with no arguments.

To remove the client, jogger, that is added in the previous section, use the following procedure:

1. Log in as superuser.

2. Change your current working directory to */usr/etc/install*, as shown in the next screen.

```
# cd /usr/etc/install
```

3. Enter **setup_client** to determine the arguments it expects. When you enter this command with no arguments, the program responds with a list of options it expects to see specified on the command line.

```
# setup_client
setup_client: incorrect number of arguments.
usage:
setup_client op name yp size rootpath swappath dumppath
        homepath execpath sharepath arch
where:
        op          = "add" or "remove"
        name        = name of the client machine
        yp          = "master" or "slave" or "client" or "none"
        size        = size for swap
                      (e.g. 16M or 16m ==> 16 * 1048576 bytes
                            16K or 16k ==> 16 * 1024 bytes
                            16B or 16b ==> 16 * 512 bytes
                            16         ==> 16 bytes )
        rootpath    = pathname of root (e.g. /export/root )
        swappath    = pathname of swap (e.g. /export/swap )
        dumppath    = pathname of dump (e.g. /export/dump ) or "none"
        homepath    = pathname of home (e.g. /home or homeserver:/home)
                      or "none"
        execpath    = full pathname of exec directory (e.g. /export/exec/sun3 )
        sharepath   = full pathname of shared files (e.g. /export/exec/share )
                      or "none"
        arch        = "sun3" or "sun4" etc
```

4. To execute *setup_client,* include the appropriate arguments in the order specified in the previous screen.

   To remove the client, jogger, which is added in the section "Using *INSTALL* to Load Software," use the *setup_client* program as shown in the next screen.

```
# setup_client remove jogger client 10m /export/root /export/swap none none \
        /export/exec/sun3 none sun3
```

   The output appears as follows.

```
Start removing sun3 client "jogger" :

Updating bootparams ...
updated bootparams
pushed bootparams

Removing root for client "jogger".

Removing swap for client "jogger".

Completed removing sun3 client "jogger".
```

   To add the client, jogger, use the *setup_client* program as shown in the next screen.

```
# setup_client add jogger client 10m /export/root /export/swap none none \
        /export/exec/sun3 none sun3
```

   The following screen illustrates sample output from the previous command.

```
Start creating sun3 client "jogger" :

Updating bootparams ...
updated bootparams
pushed bootparams

Creating root for client "jogger".

Creating 10m bytes of swap for client "jogger".

Setting up /tftpboot directory.

Updating /etc/exports to export "jogger" info.

Completed creating sun3 client "jogger".
```

Installing Optional Software After Running *INSTALL*

Page 5 illustrates installation of a Sun-3 client architectural support. All of the optional software packages are loaded at that time with the exception of the **Games** file. This section shows how to invoke the *INSTALL* program once again to extract optional software after the basic architectural support has been loaded.

All the assumptions stated on page 4 apply to this installation session.

1. Log in as superuser.
2. Change your working directory to */usr/etc/install*, as shown in the next screen.

```
# cd /usr/etc/install
```

3. To invoke the *INSTALL* program, enter **INSTALL**, as shown in the next screen.

```
# INSTALL
```

4. The *INSTALL* program prompts you for the type of installation. Answer **local** as shown in the next screen.

```
        >>>     Sun Diskless Client Installation Tool   <<<
        >>>             (for non-Sun Servers)           <<<
        >>>                  Release 1.0                <<<

Enter tape drive type ? [local | remote]: local
```

5. The *INSTALL* program then asks which tape to use. For a local installation on ½ inch magnetic tape, use **/dev/rmt12** since that is a non-rewind tape device. For tape unit 1, you would use /dev/rmt13.

```
Enter tape type ? [ar[08] | st[08] | mt[08] | xt[08] | /dev/???]: /dev/rmt12
```

6. Next enter the architecture type to load. Since this exercise assumes a Sun-3 architecture, the appropriate response is **sun3**, as shown in the next screen.

```
Enter next architecture type to load
    [ sun3 | sun4 | ... | continue | done ]: sun3
```

7. The *INSTALL* program asks where to load the executables. Since the sample installation on page 5 loads the executables in */export/exec*, this pathname is used in the next screen.

```
Enter pathname for sun3 executables ? /export/exec
```

8. The *INSTALL* program notices that the */export/exec* directory already exists and prompts you to confirm what is about to be done. Since this exercise it to install optional software, answer **use** to the program prompt, as shown in the next screen.

```
INSTALL: exec tree /export/exec/sun3 appears to already exist.
  You may select one of the following options:
        ignore  continue as if this architecture had not been specified
        remove  remove existing exec tree, then continue
        use     continue, loading any new optional software specified
        clients continue with sun3 clients, but use existing exec tree
                        ? use
```

9. The *INSTALL* procedure tries to access the tape. (Be sure the tape is mounted and online.) Press RETURN to proceed.

```
Reading the table of contents for sun3 architecture...

Mount a sun3 release tape and hit RETURN, or enter "exit" :
```

10. The next screen shows input to extract only the **Games** file.

```
Select optional software for the sun3 architecture:
Install "User File System" [20971520 bytes; required] ? [y/n]: n
Install "Sys" [2721792 bytes; desirable] ? [y/n]: n
Install "Networking tools and programs" [953344 bytes; desirable] ? [y/n]: n
Install "Debugging tools" [3383296 bytes; desirable] ? [y/n]: n
Install "SunView_Users Programs" [1453056 bytes; common] ? [y/n]: n
Install "SunView_Programmers Programs" [2064384 bytes; optional] ? [y/n]: n
Install "SunView_Demo Program source" [564224 bytes; optional] ? [y/n]: n
Install "Text Processing tools" [690176 bytes; optional] ? [y/n]: n
Install "Install tools" [1004544 bytes; optional] ? [y/n]: n
Install "User Level Diagnostics tools" [1491968 bytes; optional] ? [y/n]: n
Install "SunCore Libraries" [2991104 bytes; optional] ? [y/n]: n
Install "uucp programs" [270336 bytes; optional] ? [y/n]: n
Install "System V programs and libraries" [4481024 bytes; optional] ? [y/n]: n
Install "Manual Pages" [6072320 bytes; optional] ? [y/n]: n
Install "Demonstration Programs" [2790400 bytes; optional] ? [y/n]: n
Install "Games" [2468864 bytes; optional] ? [y/n]: y
Install "Versatec" [6117376 bytes; optional] ? [y/n]: n
Install "SunOs Security Features" [146432 bytes; optional] ? [y/n]: n

Load sharable objects in shared location ? [y/n] : n
```

11. This procedure completes the loading of the **Games** file. Respond with **done** to the next two prompts to indicate that the no other architecture support is to be loaded and that no clients are to be loaded.

```
Enter next architecture type to load
    [ sun3 | sun4 | ... | continue | done ]: done

Enter a sun3 client name ? [ name | done ]: done
```

12. The *INSTALL* program asks you if you're ready to start the installation. If so, answer **y**, as shown in the next screen.

```
Are you ready to start the installation ? [y/n] : y
```

13. The *INSTALL* program asks for confirmation once again before proceeding. In this case, choose **ignore** to load the optional software directly on top of the existing files.

```
Beginning Installation for the sun3 architecture.

/usr/etc/install/setup_exec: /export/exec/sun3 already exists.
  You may select one of the following options:
        abort   exit with error status
        ok      exit with ok status
        remove  remove existing tree, then continue
        ignore  continue (load on top of existing tree)
                        ? ignore
```

14. The *INSTALL* program realizes that the **Games** file is not on the first tape and prompts for the second tape to be loaded. Load the tape and then press (RETURN).

```
Installation of sun3 executable files begins :

[Loading version 4.0 of sun3 architecture.]
Tape loaded is #1

Load release tape #2 for architecture sun3 and hit <RETURN>:
```

The output from the previous command appears as follows.

```
Extracting "Games" files from "/dev/rmt12" release tape.

Installation of sun3 executable files completed.
```

15. Because no new clients are to be loaded, the procedure to install **Games** is complete, and the system prompt returns.

```
Starting installation of sun3 clients...


Diskless Client Installation Completed.
#
```

Important NETdisk Files and Directories

The following files and directories are significant to the NETdisk release.

/etc/exports — Automatically updated by the *INSTALL* program, this file is used to specify the special export options used by the client root directory, swap files, and executables.

/etc/bootparams — Automatically updated by the diskless client installation scripts *INSTALL* and *setup_client*. This file is used by the *rpc.bootparamd* program, which supplies information about a client's host name and location of its root, swap, and dump files. A new yellow pages map is also produced for this file.

/usr/etc/install — Directory that holds the *INSTALL* and *setup_client* programs. When executing these programs, be sure that your current working directory is */usr/etc/install* and that "." is in your PATH environment variable.

/tftpboot — Directory containing the boot programs that are downloaded using the *tftp* protocol when a diskless client is booting. For example, the boot program for a Sun-3 machine is named */tftpboot/boot.sun3*. The diskless clients download the file that is their internet address converted into uppercase hex numbers. This file is typically a symbolic link to the boot.*sun?* file (where *sun?* is the name of the sun client architecture). For example, the client named jogger has an internet address of 192.18.44.130. This number converted to hex is C0122C82 (i.e., 192=C0, 18=12, 44=2C, and 130=82). The client downloads the file */tftpboot/C0122C82*, which is a symbolic link to *tftpboot/boot.sun3*, and executes it.

/export/exec — This directory contains the architecture specific programs and libraries (essentially, the */usr* file system) that are shared among similar clients. For example, the programs for a Sun-3 machine are stored in the */export/exec/sun3* directory, while those for a Sun-4 are stored in the */export/exec/sun4* directory.

/export/root	This directory contains the root file system for each client that is served from this server. The client, jogger, finds its root file system in the */export/root/jogger* directory.
/export/swap	This directory contains the files that clients use for swapping. The swap file for the client, jogger, is */export/swap/jogger.*
/export/dump	This directory is typically unused, but can be specified to hold the crashdump image of a crashed client. Normally, the image is simply written to the swap file.

Theory of Operation

This section explains, in general terms, the mechanics of booting a diskless workstation. The general booting procedure is further discussed in the boot man page *sunboot*(8s), which supplements this description.

Booting consists of several interacting procedures:

- Reverse ARP
- tftp "get" of the boot program
- bootparams query to discover important identity information
- nfs mount and access for loading operating system

The server daemons handling these requests in NETdisk servers are, respectively: kernel Reverse ARP handler, *in.tftpd*(8c), *rpc.bootparamd*(8), and *rpc.mountd*(8c) and *nfsd*(8).

These operations work as follows: The client, when booting, knows only its Ethernet address. This address is used in a broadcast Reverse Arp request to discover its internet address. The NETdisk server responds to this request if and only if the arp mapping has been loaded into the kernel arp table (as a permanent entry) with the *arp*(8c) command. The next screen shows the command to add a permanent entry into the arp table.

```
# /etc/arp -e /etc/ethers
```

Once the client has its IP address, it sends a *tftp* "get" request of a file named the same as its IP address in uppercase hex characters. Some clients also append a dot and uppercase architecture name (such as .SUN4) to the IP address in this request, but the *in.tftpd* daemon handles this peculiarity.

This initial *tftp* request is first directed to the server that answers the Reverse ARP query, but the request is broadcast on the local network if no response is received from that server.

The file to be downloaded via *tftp* is the actual boot program. So that there is no need for information passed between booting phases, the boot program also does a Reverse ARP request as above. Boot then sends a bootparams "whoami" request (directed first to the responding server, then broadcast, as with *tftp*. The result of this *rpc* tells the client its host name, domain name, and IP address of an acceptable IP router. A second bootparam procedure, "getfile," is then used to determine the *nfs* directory of the operating system image (i.e., the client's root pathname). This pathname is used in an *nfs* mount request to get the directory file handle, which is then used in normal *nfs* lookup/read procedures to load the kernel (vmunix by default).

Once the kernel is loaded and running, it does Reverse ARP and bootparam requests. In addition to getting the pathname for the root, the kernel also gets pathnames for swap and dump. Default keys used in the "getfile" requests are *root*, *swap*, and *dump*. To override these keys, use the *-a* option with the Sun PROM monitor boot command. The next screen illustrates this command.

```
>b le() -a
```

Index

O

P

R

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. 710-001630-203, Second Edition, Rev. 2

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
Within 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

(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)