Network File System (NFS) [3]

The UNICOS network file system (UNICOS NFS) is a Cray Research software product that lets users share directories and files across a network of machines. Users of UNICOS NFS can use standard UNICOS I/O system calls, commands, and permission controls to access files from any file system. Similarly, other users of NFS can make use of UNICOS file systems from anywhere in the local network environment. UNICOS NFS can be used in diverse administrative environments through the use of the ID mapping facility. This facility is on by default in the UNICOS kernel. The user interface to UNICOS NFS is transparent.

UNICOS NFS uses a server/client system to provide access to files on the network. A *server* is any machine that allows a part of its local disk space to be exported (made available for mounting on a host machine). A *client* is any machine that makes a request for an exported file system. When a user of UNICOS issues an I/O call (such as `read`(2), `write`(2), or `create`(2)) for a file that resides on a file system mounted by NFS, the call is transmitted to the server machine. When the server receives the request, it performs the indicated operation. For read or write requests, the indicated data is returned to the client or written to disk, respectively. This processing is transparent to users, and it appears that the file resides on a disk drive that is local to the UNICOS operating system.

The following sections explain various aspects of UNICOS NFS:

- Section 3.1, page 244, provides system administrators with necessary information on activating and configuring NFS, setting up servers and clients, ID mapping, and security.
- Section 3.2, page 283, describes some common problems facing system administrators and suggests solutions.
- Section 3.3, page 297, describes the test suite that provides for early detection of UNICOS NFS problems.
- Section 3.4, page 303, describes factors that affect NFS performance, and methods for obtaining performance figures.

3.1 Administering UNICOS NFS

UNICOS NFS supports both NFS server and client capabilities. UNICOS NFS servers allow remote systems to mount local UNICOS file systems or directories; UNICOS NFS clients allow remote file systems or directories to be mounted locally. Users can then access and manipulate files in the usual way, subject to usual permission checks. The fact that parts of a file system might reside on various machines around the network is transparent to users. As system administrator, you control the use of these file systems. The following sections provide the information you need to activate, configure, and maintain NFS.

3.1.1 Activating NFS

If you are upgrading from UNICOS 9.0 and using the conversion utility, the NFS feature is on or off, depending on whether the feature was turned on or off in your UNICOS 9.0 configuration. Otherwise, the NFS feature is off by default.

If you are using the UNICOS Installation Configuration Menu System (ICMS) for your configuration, consult the `Configure System -> Major software configuration` menu for the menu item that turns on the NFS feature.

If you are not using the UNICOS ICMS for your configuration, you can turn on the NFS feature by modifying the `/etc/config/config.mh` file. Change the line that reads

```
#define CONFIG_NFS 0
```

to read

```
#define CONFIG_NFS 1
```

After you make this change, follow the rest of the system build procedures outlined in the *UNICOS System Configuration Using ICMS*, publication SG–2412.

3.1.2 Choosing a Configuration Method

The following sections describe three methods you can use to configure UNICOS NFS. Details of NFS server and client configuration are described in Section 3.1.3, page 246, and Section 3.1.4, page 249, respectively. Details of ID map configuration are described in Section 3.1.6, page 254.

3.1.2.1 UNICOS ICMS Configuration Method

You can use the UNICOS ICMS to configure NFS servers and clients.

Note: The UNICOS ICMS does not support configuration of NFS ID mapping. Refer to Section 3.1.6, page 254, for details of ID mapping.

As you use the UNICOS ICMS, the scripts and files that the UNICOS system supplies are updated for you.

Following is a description of the submenus that you can use to configure an NFS server:

- `Configure system -> Network configuration -> NFS configuration`

  This submenu lets you configure your system as an NFS server and create an `/etc/exports` file.

- `Configure system -> System daemons configuration -> System daemons table`

  This submenu configures daemons that are required for an NFS server by updating the `/etc/config/daemons` file.

Following is a description of the submenus that you can use to configure an NFS client:

- `Configure system -> Network configuration -> NFS configuration`

  This submenu lets you configure your system as an NFS client and create automount maps.

- `Configure system -> File System (fstab) Configuration -> NFS file systems`

  This submenu lets you configure the `/etc/fstab` file with a list of static NFS mounts. The `/etc/mountnfs` file that is called within the `/etc/nfsstart` script can mount these NFS file systems or directories at system startup.

- `Configure system -> System daemons configuration`

  This submenu lets you configure daemons required for an NFS client by updating the `/etc/config/daemons` file.

The help menus provide further assistance for using the UNICOS ICMS to configure NFS.

3.1.2.2 Manual Configuration Method

You can manually configure NFS by using the scripts and files that are supplied with the UNICOS operating system. The `/etc/nfsstart` script, which is called from the `/etc/netstart` script, is the script that allows manual configuration. After you have activated UNICOS NFS (see Section 3.1.1, page 244, for details of activating NFS), the `/etc/nfsstart` script performs the following actions:

1. Executes the `/etc/uidmaps/Set.domains` script, which either enables or disables ID mapping. See Section 3.1.6, page 254, for details on creating this and other ID mapping scripts.

2. Calls the `/etc/sdaemon` script to start the necessary NFS daemons in the `/etc/config/daemons` file. You must manually update this file. All input required for this file is described in Section 3.1.3, page 246, and Section 3.1.4, page 249.

3. Mounts selected remote NFS file systems or directories by calling the `/etc/mountnfs` script. You must manually update this script. All input required for this script is described in Section 3.1.4, page 249.

3.1.2.3 Local Script and File Configuration

You can configure UNICOS NFS by using local scripts and files. Details of this method of configuration are given in Section 3.1.3, page 246, Section 3.1.4, page 249, and Section 3.1.6, page 254.

3.1.3 Setting up a UNICOS NFS Server

A UNICOS NFS server is a machine that can export its own file systems and directories to another machine (an NFS client). Following are the steps required to configure your system as an NFS server:

1. Export file systems and directories

   As super user, enter the mount-point path name of the file systems and directory hierarchies that you want to export in the `/etc/exports` file. (See `exports`(5) for the file format details.)

   For example, to export `/usr/src/mybin` to `machine7` and `machine9`, and to export `/usr/man` to all machines, add the following lines to the `/etc/exports` file (or use the UNICOS ICMS):

```
/usr/src/mybin  -access=machine7:machine9
/usr/man
```

As shown, if no machines are specified for a file system, the file system is exported globally (that is, any machine can mount it).

The `exportfs`(8) command activates export entries in the `/etc/exports` files. By default, the `exportfs` command reads the `/etc/exports` file and puts an entry for each valid export in the `/etc/xtab` file. The `mountd`(8) command reads the `/etc/xtab` file to determine access rights.

The `exportfs` command is usually run during system startup from an entry within the NFS group of the `/etc/config/daemons` file. However, if a change is made to the `/etc/exports` file while the system is running, the `exportfs` command must be executed to make the changes effective. For example, the following command activates or changes a single export in the `/etc/exports` file:

`exportfs` *pathname*

The *pathname* variable specifies the full path name of the file system or directory to be exported. Any options associated with this export are read from the `/etc/exports` file.

The `/etc/exports` file must have unique entries for each file system; if entries are repeated, only the last entry that is read is valid. For example, consider the following commands:

```
/tmp orion, stardust
/tmp starship
```

In this example, only `starship` has access to `/tmp`.

2. Set up NFS server daemons

The following daemons are required for an NFS server (this list assumes that the `/etc/portmap` daemon for RPC was already started through the TCP group in the `/etc/config/daemons` file):

Daemon	Description
`mountd`	The NFS mount daemon handles incoming NFS mount requests from NFS clients. When the NFS mount request is received, `mountd` reads the `/etc/xtab` file to determine which file systems and directories are available to

	export and to determine the NFS client systems to which these files can be exported (see the preceding step regarding `/etc/xtab`). The `mountd` daemon is usually run during system startup from an entry in the NFS group of the `/etc/config/daemons` file.
`nfsd`	The NFS daemons handle NFS client file access requests after an NFS file system is mounted successfully. Typically, four `nfsd` daemons are run; these daemons are usually started during system startup from an entry within the NFS group of the `/etc/config/daemons` file, and it has no options. Following is a sample command that shows a request for four processes: `/etc/nfsd 4`
`cnfsd`	The `cnfsd` (Cray Research `nfsd`) daemon starts NFS server daemons, which use a modified NFS protocol that allows protocol extensions and eliminates most eXternal Data Representation (XDR) processing. One advantage of using `cnfsd` is that it allows Cray Research systems to communicate with other Cray Research systems with 64-bit file offsets; `nfsd` uses the NFS protocol standard of 32-bit file offsets. `cnfsd` does not provide a significant performance enhancement over `nfsd`; `cnfsd` is intended as an NFS functionality enhancement between Cray Research systems. The `cnfsd` daemon is intended for use only between Cray Research systems. For additional NFS communication to systems that are not Cray Research systems, `nfsd` must also be started in conjunction with `cnfsd`.
`pcnfsd`	This daemon is required on an NFS server if any NFS clients are personal computers (PCs). Because PC users do not have user

	IDs, it is necessary to perform special user authentication when an NFS request comes from them. The `pcnfsd` daemon runs continuously on an NFS server system to service PC NFS requests for user authentication and print spooling.
`kerbd`	The `kerbd` daemon is required if `AUTH_KERB` NFS is configured. The daemon handles requests from the kernel NFS and sends requests to and from the Kerberos key distribution center (KDC). `kerbd` also maps Kerberos user names into local user and group IDs.

Note: Remote Procedure Call (RPC) server applications (for example, `mountd` and `nfsd`), communicate with `portmapper` on the last local Internet interface that is up and running (usually `loopback`). With UNICOS security, the application must be allowed to connect with a socket on this interface. This is done by adding an explicit NAL entry in the `spnet.conf` file for the corresponding host name or a default entry. You can use the `netstat -iv` command to determine the IP address of the last up and running interface connection.

3.1.4 Setting up a UNICOS NFS Client

A UNICOS NFS client is a machine that can mount remote file systems and directories from another machine (an NFS server). To configure your system as an NFS client, the system must gain access to an exported file system and files must be set up to perform desired mounts during system startup. Automounter maps can also be set up for dynamic mounting. The following sections describe these procedures.

3.1.4.1 Mounting a Remote File System

To gain access to an exported file system, an NFS client simply mounts the file system as if it were located on a local disk. By using the `mount`(8) command, you can mount any exported file system or directory on your machine if you can reach its NFS server over the network and if your machine is included in the `/etc/exports` list for that file system, or the file system is exported globally. Using the `automount`(8) command, you can cause specified file systems to be automatically and transparently mounted whenever a file or directory within that file system is opened (see Section 3.1.4.2, page 251, for more details on automatic mounting).

After a file system is mounted, it is accessible to users as if it were a local subdirectory.

To mount a file system or directory from an NFS server, become the super user or activate one of the administrative categories with UNICOS security, and type the `mount`(8) command with the desired options. For example, to mount the man pages from remote machine `elvis` on the local directory `/usr/man`, you can type the following:

```
mount -t NFS -o bg,soft,rsize=8192,wsize=8192,nreadah=2

elvis:/usr/pubs/man /usr/man
```

The `bg` argument indicates that if the NFS mount fails, the NFS mount request should be tried repeatedly in the background. Without either the `bg` or the `soft` arguments, failed NFS mount requests are retried up to the maximum number of retries specified (or set by default) on the `mount` command; these retries occur in the foreground.

The `soft` argument indicates that if the server does not respond to either an NFS mount request or an NFS request to an already mounted file system, the requested operation fails with an error. This argument prevents processes on the client from hanging while waiting for the NFS server to respond. It is strongly recommended that all NFS mounts from a Cray Research system be soft mounts because hard mounts, indicated by the `hard` argument of the `mount` command, continue to try either mounting the NFS file system or accessing that mounted system in the foreground until the NFS server responds to the NFS request. This can cause processes and especially system startups to hang until the NFS server responds. The `intr` argument can be used with the `hard` argument at mount time, which lets you interrupt a process that is hung while waiting for the NFS server to respond.

The `rsize` and `wsize` arguments set the read and write buffer sizes, respectively, to the specified number of bytes. The default value for both `rsize` and `wsize` is 8192 (8 Kbytes). This value is adequate for most NFS servers. However, when the server is also a Cray Research system running the UNICOS system, setting `rsize` and `wsize` to 32768 improves NFS performance. The `nreadah` argument sets the number of `rsize` asynchronous read-aheads. The default value of this argument is 1. See ''Section 3.4, page 303, for more details on `rsize` and `wsize`.

At system startup, you can mount frequently used file systems and directories by placing mount entries for them in the `/etc/fstab` file (see `fstab`(5)) and invoking the `/etc/mountnfs` script from within `/etc/nfsstart`. The `/etc/nfsstart` script is called from the `/etc/netstart` script. The `/etc/mountnfs` file could contain, for example, the following lines:

```
mount /usr2 &
mount /usr/man &
```

The corresponding entries for the NFS file systems in the `/etc/fstab` file might be as follows:

```
titan:/usr2     /usr2     NFS bg,soft,rw,rsize=8192,wsize=8192,nosuid
venus:/usr/man  /usr/man  NFS soft,nosuid
```

If no `fstab` entry exists, the `/etc/mountnfs` file could contain the following lines:

```
mount -t NFS -o bg,soft,rw,rsize=8192,wsize=8192,nosuid titan:/usr2 /usr2 &
mount -t NFS -o soft,nosuid venus:/usr/man /usr/man &
```

Note: Performing NFS mounts in the background (this is done by starting `/etc/nfsstart` in the background or by using the `&` shown in the preceding example) ensures that the remainder of system startup completes in a timely manner, even if the remote NFS server systems do not respond to the mount requests.

The `biod` client daemon handles asynchronous block I/O. The `biod` daemon attempts to collect contiguous data from system buffers and write them to the network in `wsize` length sections.

On Cray Research systems, the `biod`(8) daemon enables asynchronous write-behind and read-ahead processes that can significantly improve read and write performance. The `biod` daemon is usually run during system startup from an entry within the NFS group of the `/etc/config/daemons` file, and it has no options. For optimal performance, the number of `biod` daemons should never exceed the total number of static client handles. Following is a sample command that shows a request for four processes:

```
/etc/biod  4
```

3.1.4.2 Automount Facility

Note: The use of the automount facility is not supported with the Cray ML-Safe configuration of the UNICOS system.

The automount facility automatically and transparently mounts and unmounts an NFS file system as it becomes necessary. When a user on an NFS client machine running the automount facility enters a command that accesses a file or directory that belongs to a remote file system, the remote file system is automatically mounted. When the automatically mounted remote file system has not been accessed within a period of time, the file system is unmounted automatically.

The `automount`(8) command does not consult the `/etc/fstab` file for a list of remote file systems or directories to mount, but instead, has its own set of configuration files known as *maps*. Therefore, to enable the automount facility, you must first create map files. See `automount` for the format of these files and a description of the command.

The `automount` daemon is required if the NFS client will be running the automounter. The `automount` command is usually run during system startup from an entry within the NFS group of the `/etc/config/daemons` file. Following is a sample `automount` command:

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

This is a typical `automount` command because the UNICOS system requires the `-m` option and the use of an automount master file.

Note: Any automount options listed within the indirect map entries override all options listed in the master map for that entry. If you are using `automount` when running ID mapping on an NFS client, you must define `loopback` or `localhost` as an ID mapping domain.

3.1.4.3 Protocol between Cray Research Systems

When processing is between two Cray Research systems, you can use a modified NFS protocol (which the `cnfsd`(8) daemon uses) to reduce the CPU time required to process an NFS request. The removal of XDR processing makes this reduction possible. You can access much larger files across NFS with this protocol than with the standard NFS protocol because all file size and file offset fields within the modified protocol are a full 64 bits. Another advantage of using `cnfsd` is that it uses 32 Kbytes read and write sizes; `nfsd` defaults to 8 Kbytes. You can use the modified NFS protocol between Cray Research systems by specifying the `-o` option and `cray` operand with the `mount`(8) command on the Cray Research NFS client when mounting a Cray Research NFS server. Also, you must start at least one `cnfsd` process on the Cray Research NFS server. See `mount`(8) and `cnfsd` (see `nfsd`(8)) for more information.

3.1.5 Typical UNICOS NFS Layout

The following output from two `mount`(8) commands demonstrates the layout of UNICOS NFS in a typical environment, showing both the Cray Research client and a client that is not a Cray Research client. The first example is from a Cray Research system called `cray2`. It shows the local file systems and the NFS mounted file system `/nfs/titan` (exported from server `titan`). The output from the mount operation is followed by a listing of the mounted file system.

Example 1: `cray2` as client, `titan` as server, as seen from the `cray2` system:

```
cray2%  /etc/mount
/ on /dev/dsk/root  read/write on Mon Apr 4 06:55:07 1988
/usr on /dev/dsk/usr read/write on Mon Apr 4 06:55:35 1988
/u on /dev/dsk/u  read/write on Mon Apr 4 06:55:35 1988
/nfs/titan on titan:/usr/titan read/write,rsize=8192,wsize=8192 on Mon
   Apr 4  06:55:45 1988
cray2% ls -l /nfs/titan
total 39
drwxr-x---     22  btk          network 1536 Mar 31 10:16 btk
drwxr-xr-x      9  common       cray2    512 Feb  1 08:32 X
drwxr-xr-x     37  mer          netqa   2560 Mar 31 15:50 mer
drwxr-xr-x      5  prb          cray2   3072 Apr  1 22:51 prb
drwxr-xr-x     23  wtg          appl    1024 Mar 23 13:32 wtg
```

Example 2: `titan` as client, `cray2` as server, as seen from the `titan` system:

```
titan% /etc/mount
/dev/xy0a on / type 4.2 (rw)
/dev/xy1c on /usr.MC68020/titan type 4.2 (rw)
cray2:/u on /usr/cray2/u type nfs (rw,soft,bg)
titan% ls -lg /usr/cray2/u
total 32
drwxr-x--x     26  btk          secure  1715 Jun 10  1984 btk
drwxrwxr-x     59  common       os      1089 Jan  7  1987 X
drwxr-xr-x     41  mer          netqa   3134 Apr 13 16:45 mer
drwxr-xr-x     12  pfh          starter  557 Jun  2  1986 pfh
drwxr-x---      2  slevy        msc      512 Feb 18  2:11 slevy
drwxr-xr-x     35  wtg          network  544 Apr  4 10:06 wtg
```

3.1.6 ID Mapping

UNICOS NFS includes an ID mapping facility that allows the use of NFS in diverse administrative environments. Traditional NFS environments make use of the Sun Microsystems network information service (NIS) distributed look-up service to provide for various network management functions. The user space that NIS provides is flat; that is, a given ID number always refers to the same user or group. This flat user space is necessary because NFS transmits user and group identifiers in binary form, and it provides no translation services for these values.

Cray Research systems, however, are often shared by many different administrative environments, making the creation of a single administrative space for user and group identification technically or organizationally difficult, if not impossible. A given ID number can refer to different users or groups in different administrative environments. To meet the needs of these environments, ID mapping was developed.

ID maps contain an equivalent remote ID for each local ID in a map. There are two types of ID maps: user ID maps and group ID maps. For every user ID map, a corresponding group ID map exists.

ID mapping domains associate Internet addresses with a particular pair of user and group ID maps. When an NFS request is sent to or received from an address within an ID mapping domain, the pair of ID maps associated with that ID mapping domain can be used to replace the IDs in the request.

ID mapping can also be used to control access to the local Cray Research system through NFS by allowing requests only from certain Internet addresses, or by restricting permissions for certain users at these addresses.

Figure 17 is a diagram of the ID mapping function as it relates to UNICOS NFS system interfaces.

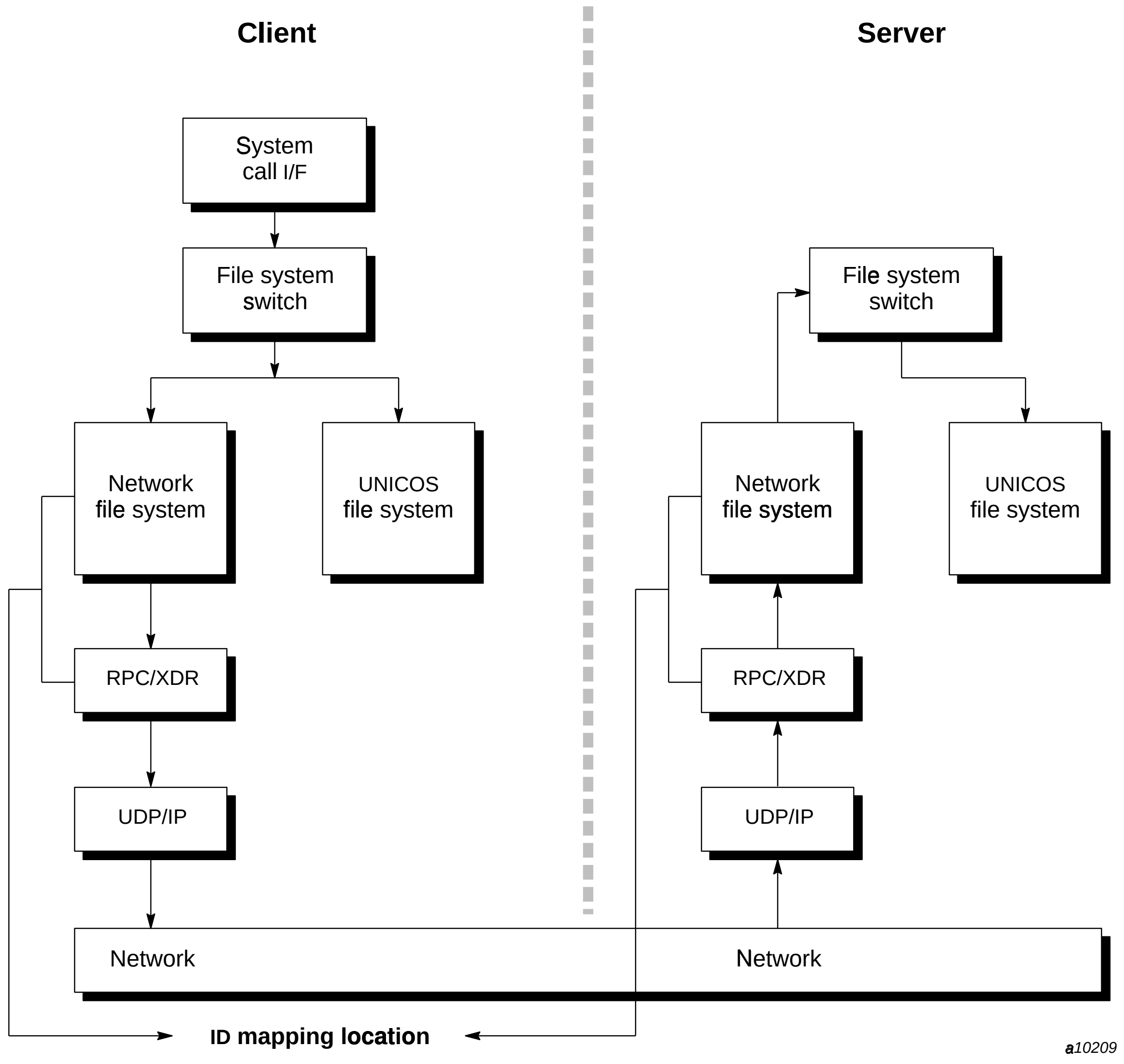

Figure 17. System interfaces and ID mapping

3.1.6.1 Disabling ID Mapping

ID mapping in the UNICOS kernel is on by default. The `nfsidmap`(8) command disables the use of ID mapping at run time. NFS ID mapping is not required when the Cray Research system and all other systems using NFS use the same user ID space.

To disable NFS ID mapping, create an executable `/etc/uidmaps/Set.domains` script that contains the following line:

```
/etc/uidmaps/nfsidmap -d
```

3.1.6.2 Configuring and Using ID Mapping

Configuring and using NFS ID mapping is a site-dependent function. However, you should always use the following basic steps to configure NFS ID mapping:

1. Obtain the `passwd`(5) and `group`(5) files from each remote administrative environment for which IDs will be mapped.
2. Use the `passwd` and `group` files from the local Cray Research system, along with those obtained from the remote systems, to create the user and group ID map files between the remote domains and the local Cray Research system.
3. Load the ID maps into the kernel, and define the ID mapping domains.

If a remote administrative environment is the same as that on the local Cray Research machine, the creation of an ID map for use between these machines (steps 1 and 2) is not necessary unless you are running with UNICOS security. With UNICOS security, ID maps are required for all remote environments.

The `/etc/uidmaps` directory exists to store ID mapping commands and associated files for ID mapping. In an environment without UNICOS security, the password and group files can be collected into this directory and processed. This directory also contains the ID mapping commands, the administrative shell scripts, and files that contain the constructed ID maps that will be loaded into the kernel. In a UNICOS security environment, the `/etc/uidmaps` directory is created with a `syshigh` label to protect the commands and maps after they are created. A directory with a `syshigh` label is unsuitable for remote copies from the network, so a separate directory must be created for collecting password and group files from remote systems.

There are two `/etc/uidmaps` subdirectories. The `/etc/uidmaps/users` directory should contain all `passwd` files from the remote administrative domains that are being mapped, and any exceptions files (explained in "Exceptions file," page Section 3.1.6.4.4, page 261). The `/etc/uidmaps/groups` directory should contain all `group` files from the remote administrative domains that are being mapped, and any exceptions files.

3.1.6.3 Network Description Example

The example used throughout this section shows how to configure and use the UNICOS NFS ID mapping facility. The procedures (shell scripts) shown can be used on all Cray Research machines in the example. Although the sequence of the commands shown is common to any Cray Research system using ID mapping, the specific contents of the procedures might vary according to site standards.

Assume the following network consisting of three Cray Research machines configured to perform various types of ID mapping, a single machine (not a Cray Research machine) with its own administrative domain, and several workstation networks using different NIS domains:

Name	Description
`groucho`	A Cray Research machine with its own administrative domain.
`chico`	A Cray Research machine with its own administrative domain.
`harpo`	A Cray Research machine with its own administrative domain.
`zeppo`	A different type of machine with its own administrative domain; that is, the assigning of login names and group names (and binary values associated with those names) occurs separately from those operations on other machines on the network. This machine is also running a version of UNIX, rather than another operating system.
`nfl`	A network of workstations that share a single NIS domain for both the `/etc/passwd` and `/etc/group` files. The name `nfl` was chosen because `nfl` is a fileserver that is the NIS master for the password and group NIS databases.
`disney`	A network of workstations similar to `nfl`, but with separate administration using NIS.

Each machine name is the host name of a representative machine for its administrative domain. It is also the name that is chosen for each ID mapping domain.

3.1.6.4 Setup, Creation, and Maintenance of ID Map Files Example

User and group ID map files are built from the `passwd` and `group` files of the local Cray Research system and of each remote administrative domain. Mappings are created for each user or group on the Cray Research system that matches a user or group on a remote administrative domain. Each mapping is placed in either a user ID map file or a group ID map file.

ID map files are built by using the `nfsmerge`(8) utility. The `Get.domains` and `Merge.domains` scripts control the creation of user maps. These scripts, which you must configure for local systems, correspond to steps 1 and 2 of configuring ID mapping domains, as described in Section 3.1.6.2, page 256. For these two steps, it is not necessary to run a kernel with ID mapping configured.

Note: You must manually supervise the running of `Get.domains` and `Merge.domains` to ensure that they actually work, because the scripts contain no error handling of their own. If they are run automatically and they encounter errors, the resulting user ID maps might not be valid or secure.

3.1.6.4.1 ID Map File Setup

The `Get.domains` script copies `passwd` and `group` files from the local system and from a representative of each remote administrative domain to the local system. You are free to use any utilities or mechanisms to create these files. Usually, `rsh`(1), `rcp`(1), or `ftp`(1B) is used to copy files from the remote systems. (If `rsh`(1) or `rcp`(1) is used, remote execution from the local Cray Research machine must be allowed on all remote machines.) If a remote machine has an operating system other than UNIX or one of its derivatives, the administrator of that machine must construct the equivalent `passwd` and `group` files to enable the `nfsmerge`(8) utility to create an ID map file.

The following `Get.domains` script should be run whenever you are notified that users or groups were added to any of the remote domains:

Note: If your site uses ID mapping, `root` (`uid` equals 0) must be contained in the `passwd`(5) files to be used for the mapping.

```
% cat /etc/uidmaps/Get.domains
```

```
#
# Script to collect and sort password and
# group files from the various domains referenced.
#
#    This for loop collects information from non-NIS hosts
#

USERS=/etc/uidmaps/users
GROUPS=/etc/uidmaps/groups

for sv in groucho chico harpo zeppo
do
   echo "Getting passwd from $sv"
   remsh $sv cat /etc/passwd | sort -t: +0 -1 -o $USERS/passwd.$sv
   echo "Getting group from $sv"
   remsh $sv cat /etc/group | sort -t: +0 -1 -o $GROUPS/group.$sv
done
#
#  This loop collects information from NIS hosts
#
for yp in nfl disney
do
   echo "Getting passwd from $yp"
   remsh $yp ypcat passwd | sort -t: +0 -1 -o $USERS/passwd.$yp
   echo "Getting group from $yp"
   remsh $yp ypcat group | sort -t: +0 -1 -o $GROUPS/group.$yp
done
```

The created files are called `passwd`. *domain* and `group`. *domain*. (As with the copying of files, this procedure requires that each remote machine allow remote execution from the local Cray Research machine.) Each of these files is sorted on the user or group name. It is not necessary to sort the `passwd` or `group` files before making the ID maps; however, sorting generally speeds the creation of the maps.

3.1.6.4.2 ID Map File Creation

The `Merge.domains` script creates the ID map files. This script calls the `nfsmerge`(8) utility to create the ID map files between the local Cray Research machine and the remote administrative domains. The `nfsmerge`(8) utility uses the `passwd` and `group` files (usually a copy of each) from the local Cray Research machine and from a remote administrative domain to create a mapping between the numerical user and group ID values on the two domains, using login and group names for comparison. It expects, for example, that the login name `grumpy` on the local Cray Research machine and on the remote administrative domain refers to the same user. The same is true for groups.

3.1.6.4.3 ID Map File Maintenance

Rerun `Get.domains` periodically to update the map files (see Section 3.1.6.4.1, page 258). The following `Merge.domains` script should be run after any update of the local copies of the `passwd` and `group` files by the `Get.domains` script. It should also be run when an exceptions file changes (see Section 3.1.6.4.4, page 261).

> **Note:** With UNICOS security, NFS ID maps contain mandatory access control (MAC) configuration information. Because these maps contribute to enforcement of MAC policy, you must protect these maps and the scripts that produce them by labeling them with the `syslow` label. To safeguard against security risks, the maps must also be manually inspected each time they are created. This is because the maps are constructed from password and group files that might not be protected by the `syslow` label when they are collected from the other systems on the network. The method of inspection and the degree to which that process can be automated is site-dependent.

To meet this labeling requirement, every script or program used in ID map generation must have a `syslow` label; they must be executed at `syslow`, and the resulting map files must have a `syslow` label. The NFS commands in the `/etc/uidmaps` directory and the directory itself are automatically installed with this label. These commands are privileged to access the `syslow` -labeled ID map files and to load the maps into the kernel. To use these commands, you must have the `secadm` category active. The `nfsmerge`(8) utility, which creates ID map files, labels them with its process-execution label. However, existing ID map files are overwritten without their label being changed. Therefore, you should remove all existing ID map files at the start of the ID map generation process.

```
% cat /etc/uidmaps/Merge.domains

#
#  Script to create ID maps from the sorted passwd and group files
#  for each administrative domain.
#

HOST=‘hostname‘
USERS=/etc/uidmaps/users
GROUPS=/etc/uidmaps/groups
CMD=/etc/uidmaps/nfsmerge

for cray in groucho chico harpo
do

    for domain in groucho chico harpo zeppo nfl disney
    do
      if ‘test $cray != $domain‘      then
        echo "Creating user and group ID maps between $cray and $domain"
                                                     | tee l.$cray.$domain
        $CMD    -l $USERS/e.$cray.$domain -u u.$cray.$domain
                -e $GROUP/e.$cray.$domain -g g.$cray.$domain
                $USERS/passwd.$cray $USERS/passwd.$domain
                $GROUPS/group.$cray $GROUPS/group.$domain >> l.$cray.domain

    done
  fi
done
```

3.1.6.4.4 Exceptions File

Users are likely to have the same login name wherever possible, even though they might use several machines from different administrative environments. However, if a user has a login name in the remote administrative environment that differs from that on the local Cray Research machine, that user ID can be mapped into an exceptions file. The same applies to group ID mapping.

The exceptions file should contain a list of name pairs, one pair per line, the names separated by white space. The name pairs are of the following form:

local_name   *equivalent_remote_name*

For example, user Big Bad Wolf has the login name `bbw` on machine `groucho` and the login name `wolf` in the `disney` NIS domain. The exceptions file on groucho could be called /etc/uidmaps/users/e.groucho.disney and could contain the following entry:

```
bbw  wolf
```

Use of this exceptions file when making the map file between groucho and the disney NIS domain would ensure that the user ID mapping for Big Bad Wolf is placed in the map file.

If *equivalent_remote_name* is not specified in the exceptions file, it is considered the same as *local_name*. This feature is useful for restricting maps to use only the names in the exceptions files (see the -E or -L option of the nfsmerge(8) command for details).

An exceptions file can be used to prevent user names from being mapped. You can use a name that is not present in the remote passwd or group file as an exception for each name that is to be restricted. For example, assume that NFS access to a Cray Research system is to be restricted for login names maleficent and stepmother from the disney NIS domain. Entries for these users could be put into the exceptions file, as follows:

```
maleficent      BogusUser
stepmother      BogusUser
```

The login name BogusUser is not a valid login name in the disney NIS domain or on the Cray Research system. Therefore, user IDs for these users are not mapped between groucho and any machines using the disney NIS domain.

Examine the passwd and group files for exceptions before running the Merge.domains script, which is written to expect an exceptions file for all mappings. If an exceptions file is not present, but specified on the command line, the Merge.domains script issues a warning message.

3.1.6.4.5 Map Files

The previous Get.domains and Merge.domains example scripts assume that ID map files are maintained in the /etc/uidmaps directory.

The Merge.domains script creates map files only between the local Cray Research machine and all remote administrative domains. A log file, user ID file, and group ID file are created each time nfsmerge is called in the script. These files are placed in the /etc/uidmaps directory.

The log files contain a line identifying the type of ID map, the names of the local and remote passwd files, and a list of all names for which IDs were mapped. The example script is written so that it is obvious from the name of the map file which user or group ID map was created.

The files involved in mapping user IDs between local machine `harpo` and the NIS domain `nfl` in the example are as follows:

File	Description
`/etc/uidmaps/users/passwd.harpo`	A copy of the local `passwd` file sorted on login name
`/etc/uidmaps/users/passwd.nfl`	A copy of the remote `passwd` file sorted on login name
`/etc/uidmaps/users/e.harpo.nfl`	A list of login name exceptions between the administrative domains `harpo` and `nfl`
`/etc/uidmaps/l.harpo.nfl`	A log file from the creation of the user and group ID map between `harpo` and the `nfl` NIS domain
`/etc/uidmaps/u.harpo.nfl`	The user ID map file between `harpo` and the `nfl` NIS domain

Similarly, the files involved in mapping group IDs between `harpo` and the `nfl` NIS domain are as follows:

File	Description
`/etc/uidmaps/groups/group.harpo`	A copy of the local `group` file sorted on group name
`/etc/uidmaps/groups/group.nfl`	A copy of the remote `group` file sorted on group name
`/etc/uidmaps/groups/e.harpo.nfl`	A list of group name exceptions between the administrative domains `harpo` and `nfl`

`/etc/uidmaps/l.harpo.nfs`

A log file from the creation of the group ID map between `harpo` and the `nfl` NIS domain

3.1.6.5 Kernel Map Manipulation Example

Each ID map is given a name to use for display purposes and to use with some commands related to ID mapping. A separate map should be created for each autonomously administered system. For example, a map might be created for each stand-alone mainframe system on the network; one map would be required for a network of workstations in a single NIS domain.

Mapping in the kernel is a two-step process. The first step involves determining the particular ID mapping domain, given an Internet address. The second step uses the ID mapping domain, the type of map operation (user or group), and the direction (into or out of the Cray Research system) to determine the effective user or group identifier. Both of these operations occur in the UNICOS kernel with NFS configured and are based on information inserted into the kernel by the `nfsaddmap`(8) and `nfsaddhost`(8) utilities. See the *UNICOS Administrator Commands Reference Manual*, publication SR–2022, for a complete description of these commands.

Hosts are grouped into ID mapping domains based on sets of *address, mask* pairs on `nfsaddhost` calls. Addresses can be specified in standard form (for example, 128.1.0.1), as network names (names are found in `/etc/networks`), or as host names (from `/etc/hosts`). Default masks for hosts and standard forms are all 1's; for network names, the default masks are 1's covering the network part of the address (see the example of masks in the `nfsaddhost`(8) command description).

User ID mapping is straightforward. Each user ID map entry contains the local user ID and groups list and the remote user ID and groups list. When an NFS request is sent to the Cray Research system and the Cray Research system is mapping IDs for client-side requests, the Internet address to which the request is sent determines the ID map to be used. The map is searched for the local user ID; if it is found, the remote user ID and groups list is used in that request. If the local user ID is not found in the ID map, the ID mapping domain is checked to see what should be done (the `nfsaddhost`(8) command has options that determine the action associated with a particular ID mapping domain). There are three choices:

- The value for a bad user ID (-1) can be returned. In this case, the request should be denied with an RPC authentication error. This is the default.

- The value for the user "nobody" (-2) can be returned. In this case, the user has access only to other-accessible files and directories on the server.
- The local IDs can remain unmapped and can be put into the request. Group ID mapping is even simpler. The group ID map tables are used only to map file attributes (that is, to map IDs associated with a file that is accessed through NFS). Each group map entry contains a local group ID to correspond to a remote group ID.

Kernel ID mapping tables and ID mapping domains are inserted by the `/etc/uidmaps/Set.domains` script, called out of the `/etc/nfsstart` script. The `Set.domains` script example corresponds to step 3 of configuring ID mapping domains, as described in Section 3.1.6.2, page 256. This script primarily uses three ID mapping commands, `nfsclear`(8), `nfsaddmap`(8), and `nfsaddhost`(8). The `nfsclear`(8) command is called first to ensure that any previous ID mapping information in the kernel is cleared out. The `nfsaddmap`(8) utility is called to read a map file, previously created with the `nfsmerge`(8) command, into the kernel. The `nfsaddhost`(8) utility defines the ID mapping domains; it associates Internet addresses with the kernel ID maps to use for mapping.

A special domain name, `MAP_THRU`, is defined for systems known to share the local Cray Research system's user and name space. The `MAP_THRU` domain simply allows user and group identifiers to pass through, without modifying them in any way.

Note: Do not use `MAP_THRU` with UNICOS security because the security information that NFS requires with UNICOS security will be missing.

The `Set.domains` example performs configurations, as follows:

Note: Your `Set.domains` file should contain the loopback entry as shown in the example; it is required if you are running the automounter.

1. Cray Research hosts `groucho` and `chico` perform server ID mapping between them.
2. Cray Research host `chico` performs all ID mapping between Cray Research host `harpo` and itself.
3. Cray Research hosts `harpo` and `groucho` perform client ID mapping between them.
4. Cray Research hosts `groucho`, `chico`, and `harpo` perform both client and server mapping to all other hosts on the network. The `Set.domains` script is as follows:

```
% cat /etc/uidmaps/Set.domains

#
#       Set
#

HOST='hostname'MAPS=/etc/uidmaps
CMDS=/etc/uidmaps

#
#       Reinitialize kernel ID mapping information.
#

$CMDS/nfsidmap -d
sleep 3

$CMDS/nfsclear
$CMDS/nfsaddhost -l loopback
#
#       Set ID maps and ID mapping domains between all of
#       the Cray machines on the network.
#
if 'test $HOST = groucho'then
#       Groucho does server ID mapping only for chico
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.chico
                         -g $MAPS/g.$HOST.chico  chico
        $CMDS/nfsaddhost -d chico -s -l chico-inet
        $CMDS/nfsaddhost -d chico -s -l chico-prod
        $CMDS/nfsaddhost -d chico -s -l chico-lsp
        $CMDS/nfsaddhost -d chico -s -l chico-hsx -u chico-hsx2

#       Groucho does client mapping only for harpo.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.harpo
                         -g $MAPS/g.$HOST.harpo  harpo
        $CMDS/nfsaddhost -d harpo -c -l harpo-inet
        $CMDS/nfsaddhost -d harpo -c -l harpo-vme24
        $CMDS/nfsaddhost -d harpo -c -l harpo-vme26
        $CMDS/nfsaddhost -d harpo -c -l harpo-vme32
        $CMDS/nfsaddhost -d harpo -c -l harpo-lsp
```

```
elif ‘test $HOST = chico‘then
#       Chico does server mapping only for groucho.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.groucho
                         -g $MAPS/g.$HOST.groucho  groucho
        $CMDS/nfsaddhost -d groucho -s -l groucho-inet
        $CMDS/nfsaddhost -d groucho -s -l groucho-lsp
        $CMDS/nfsaddhost -d groucho -s -l groucho-hsx -u groucho-hsx2

#       Chico does both client and server mapping to harpo.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.harpo
                         -g $MAPS/g.$HOST.harpo  harpo
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-inet
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-vme24
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-vme26
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-vme32
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-lsp
        $CMDS/nfsaddhost -d harpo -c -s -l harpo-hsx

elif ‘test $HOST = harpo‘then
#       Harpo does client mapping only to groucho.
        $CMDS/nfsaddmap  -u $MAPS/u.$HOST.groucho
                         -g $MAPS/g.$HOST.groucho  groucho
        $CMDS/nfsaddhost -d groucho -c -l groucho-inet
        $CMDS/nfsaddhost -d groucho -c -l groucho-lsp
        $CMDS/nfsaddhost -d groucho -c -l groucho-hsx -u groucho-hsx2

#       Harpo maps through to chico (MAP_THRU facility).
        $CMDS/nfsaddhost -l chico-inet
        $CMDS/nfsaddhost -l chico-prod
        $CMDS/nfsaddhost -l chico-lsp
        $CMDS/nfsaddhost -l chico-hsx -u chico-hsx2

else
        echo "don’t know how to set domains for $HOST"
        exit
fi
```

```
# The Cray machines necessarily do both client and server mapping for
# all of the rest of the machines in the network.
#
#       zeppo
#
$CMDS/nfsaddmap -u $MAPS/u.$HOST.zeppo -g $MAPS/g.$HOST.zeppo zeppo

$CMDS/nfsaddhost -d zeppo -c -s -l zeppo-inet

#
#       Workstation network (nfl YP domain)
#

$CMDS/nfsaddmap -u $MAPS/u.$HOST.nfl -g $MAPS/g.$HOST.nfl nfl

#
#  Networks in the nfl YP domain
#

$CMDS/nfsaddhost -d nfl -c -s -l afc-eastnet -u afc-westnet
$CMDS/nfsaddhost -d nfl -c -s -l nfc-eastnet
$CMDS/nfsaddhost -d nfl -c -s -l nfc-centralnet
$CMDS/nfsaddhost -d nfl -c -s -l nfc-westnet

#
#  Explicit hosts in the nfl YP domain
#

$CMDS/nfsaddhost -d nfl -c -s -l nfl-gate
$CMDS/nfsaddhost -d nfl -c -s -l afc-gate
$CMDS/nfsaddhost -d nfl -c -s -l nfc-gate
$CMDS/nfsaddhost -d nfl -c -s -l nfc-east-server
$CMDS/nfsaddhost -d nfl -c -s -l nfc-central-server
$CMDS/nfsaddhost -d nfl -c -s -l nfc-west-server
$CMDS/nfsaddhost -d nfl -c -s -l afc-east-server
$CMDS/nfsaddhost -d nfl -c -s -l afc-central-server
$CMDS/nfsaddhost -d nfl -c -s -l afc-west-server
$CMDS/nfsaddhost -d nfl -c -s -l superbowl-server
$CMDS/nfsaddhost -d nfl -c -s -l bears-inet
$CMDS/nfsaddhost -d nfl -c -s -l colts-prod
$CMDS/nfsaddhost -d nfl -c -s -l giants-inet
$CMDS/nfsaddhost -d nfl -c -s -l redskins-prod
```

```
$CMDS/nfsaddhost -d nfl -c -s -l broncos-inet
$CMDS/nfsaddhost -d nfl -c -s -l vikings-inet
$CMDS/nfsaddhost -d nfl -c -s -l niners-prod
$CMDS/nfsaddhost -d nfl -c -s -l raiders-inet
$CMDS/nfsaddhost -d nfl -c -s -l patriots-prod
$CMDS/nfsaddhost -d nfl -c -s -l saints-prod#
#       Workstation network (disney YP domain)
#

#$CMDS/nfsaddmap -u $MAPS/u.$HOST.disney -g $MAPS/g.$HOST.disney disney

#
#  Networks in the disney YP domain
#

$CMDS/nfsaddhost -d disney -c -s -l snowwhitenet
$CMDS/nfsaddhost -d disney -c -s -l junglebooknet
$CMDS/nfsaddhost -d disney -c -s -l bambinet

#
#  Hosts in the disney YP domain
#

$CMDS/nfsaddhost -d disney -c -s -l disney-gate
$CMDS/nfsaddhost -d disney -c -s -l disney-land
$CMDS/nfsaddhost -d disney -c -s -l disney-world

$CMDS/nfsaddhost -d disney -c -s -l snowwhite-server
$CMDS/nfsaddhost -d disney -c -s -l snowwhite-inet
$CMDS/nfsaddhost -d disney -c -s -l bashful-inet
$CMDS/nfsaddhost -d disney -c -s -l sleepy-inet
$CMDS/nfsaddhost -d disney -c -s -l sneezy-inet
$CMDS/nfsaddhost -d disney -c -s -l dopey-prod
$CMDS/nfsaddhost -d disney -c -s -l dopey-inet
$CMDS/nfsaddhost -d disney -c -s -l happy-inet
$CMDS/nfsaddhost -d disney -c -s -l grumpy-prod
$CMDS/nfsaddhost -d disney -c -s -l doc-inet
$CMDS/nfsaddhost -d disney -c -s -l doc-prod
```

```
$CMDS/nfsaddhost -d disney -c -s -l junglebook-server
$CMDS/nfsaddhost -d disney -c -s -l junglebook-prod
$CMDS/nfsaddhost -d disney -c -s -l mowgli-inet
$CMDS/nfsaddhost -d disney -c -s -l hista-prod
$CMDS/nfsaddhost -d disney -c -s -l sherkan-inet
$CMDS/nfsaddhost -d disney -c -s -l baloo-inet
$CMDS/nfsaddhost -d disney -c -s -l kinglouie-inet
$CMDS/nfsaddhost -d disney -c -s -l bakkera-prod

$CMDS/nfsaddhost -d disney -c -s -l bambi-server
$CMDS/nfsaddhost -d disney -c -s -l bambi-inet
$CMDS/nfsaddhost -d disney -c -s -l thumper-inet
$CMDS/nfsaddhost -d disney -c -s -l flower-inet

$CMDS/nfsidmap   -e
```

3.1.6.6 Other Administrative Considerations

When ID mapping is configured, all server activity makes use of it. Most NFS client systems pass the `root` user ID (0) for user identification on their mount requests; these values are also subject to ID mapping.

You can remove kernel ID maps and ID mapping domains through the use of the `nfsrmmap`(8) and `nfsrmhost`(8) commands. These commands, along with `nfsaddmap`(8), `nfsaddhost`(8), `nfsadduser`(8), and `nfsrmuser`(8), give you the ability to modify ID mapping at any time; it is not necessary to recompile anything to modify the ID maps. To view the currently defined ID mapping domains, use the `nfslist`(8) command.

The following is an abbreviated sample of the use of `nfslist`(8) on machine `groucho`:

```
groucho% nfslist
NFS ID Mapping is : ENABLED

NFS ID    NFS ID                     Lower        Upper
Map       Mapping                    Bound        Bound        Address
Name      Flags                      Address      Address      Mask

nfs       CLIENT  SERVER  BAD_ID     c0.09.23.00  c0.09.26.00  [ff.ff.ff.00]
                      Addr(dec)      Addr(hex)    Host Name
                      128.1.35.0     c0.09.23.00  afc-eastnet
                      128.1.36.0     c0.09.24.00  afc-centralnet
                      128.1.37.0     c0.09.25.00  afc-westnet

zeppo     CLIENT  SERVER  BAD_ID     c0.09.0e.5a  c0.09.26.5a  [ff.ff.ff.ff]
                      Addr(dec)      Addr(hex)    Host Name
                      128.1.14.90    c0.09.0e.5a  zeppo-inet

chico             SERVER  BAD_ID     54.00.cf.05  54.00.cf.05  [ff.ff.ff.ff]
                      Addr(dec)      Addr(hex)    Host Name
                      84.0.207.5     54.00.cf.05  chico

chico             SERVER  BAD_ID     5c.00.00.05  5c.00.00.05  [ff.ff.ff.ff]
                      Addr(dec)      Addr(hex)    Host Name
                      92.0.0.5       5c.00.00.05  chico-lsp

harpo     CLIENT          BAD_ID     54.00.cf.06  54.00.cf.06  [ff.ff.ff.ff]
                      Addr(dec)      Addr(hex)    Host Name
                      84.0.207.6     54.00.cf.06  chico-lsp

harpo     CLIENT          BAD_ID     5c.00.00.06  5c.00.00.06  [ff.ff.ff.ff]
                      Addr(dec)      Addr(hex)    Host Name
                      92.0.0.6       5c.00.00.06  chico-lsp
```

To remove an ID mapping domain, the options of `nfsrmhost`(8) must exactly match the definition of the domain. You cannot remove part of a domain. Also, to remove a kernel ID map, you must remove all ID mapping domains that reference that map.

You must also be aware of hosts that are running NFS but are not UNIX systems. For these hosts, the `Get.domains` script used in the example must be modified according to specific characteristics of the site's network. For example, the script run in the example in this section assumes that `passwd` and `group` files exist on the remote system. This is not necessarily true for hosts that are not UNIX systems. To create the map files necessary for mapping IDs through NFS, you must construct files in passwd and group format for any administrative domains that do not already have these files (that is, you must create synthetic passwd and group files). If you create synthetic passwd and group files for use with ID mapping, you must ensure that entries are present for user root and group sys, or whatever these entries are called on the local Cray Research system.

3.1.6.7 Running pcnfsd with NFS ID Mapping Control

Note: The use of pcnfsd is not supported with the Cray ML-Safe configuration of the UNICOS system.

If you have a PC NFS client, the pcnfsd(8) daemon runs on an NFS server. When a PC NFS client connects to a pcnfsd, the client prompts for a login name and password. After verifying the password for the given login name, the pcnfsd daemon passes a user ID and a groups list back to the PC NFS client. The PC uses the IDs it receives from pcnfsd for subsequent NFS requests to that NFS server.

The pcnfsd daemon on a Cray Research system can use NFS ID mapping, which makes PC NFS access more secure. After the password validation that pcnfsd performs is complete, the user ID map entry for that user is added to an ID map. Therefore, if pcnfsd on a Cray Research system is configured to use an ID map, only users whose passwords were actually validated through pcnfsd can access a Cray Research NFS server.

To use pcnfsd with NFS ID mapping, ensure that the following steps have been taken:

1. Create an ID map file for a Cray Research system to the same Cray Research system in the Merge.domains script, as follows:

   ```
   nfsmerge -u /etc/uidmaps/u.cray.cray -g /etc/uidmaps/g.cray.cray
   /etc/passwd /etc/passwd /etc/group /etc/group
   ```

   Note: If you are also setting up a special MAP_THRU NFS ID map (see Section 3.1.6.9, page 275), the nfsmerge command needs to be executed only once because pcnfsd ID mapping and special MAP_THRU ID mapping use the same ID map file.

2. Add that ID map to the kernel with an appropriate name in the Set.domains script. The following command adds an empty user ID map and a group ID map called pcidmap to the kernel:

```
nfsaddmap -g /etc/uidmaps/g.cray.cray pcidmap
```

The user ID map is empty so that `pcnfsd` can add user entries when the user's password validation succeeds.

3. Ensure that the network addresses of the PCs that will be accessing the NFS server on the Cray Research system are in an ID mapping domain that uses the ID map (called `pcidmap` in the previous examples), as follows:

```
nfsaddhost -d pcidmap -c -s -l pc_addr1
nfsaddhost -d pcidmap -c -s -l pc_addr2
          .  .  .  .
nfsaddhost -d pcidmap -c -s -l pc_addrN
```

4. Start `pcnfsd` with the name of the user ID map file and kernel map name, as follows:

```
pcnfsd -u /etc/uidmaps/u.cray.cray -m pcidmap
```

The `pcnfsd` daemon does not remove entries that `pcnfsd` has added to this map. Therefore, until the system administrator resets the ID maps in the kernel by running the `Set.domains` script or until the system reboots, any user validated through `pcnfsd` has NFS access to a Cray Research system from the PC network addresses in the ID mapping domains that reference `pcidmap`.

3.1.6.8 Deciding When to Use ID Mapping

Kernel ID maps contain the following information for each local user ID:

- Default account ID (acid)
- Security information (minimum and maximum security level and valid security compartments)
- Pointer to a list of optional Kerberos authenticated Internet addresses
- Pointer to a list of client side `auth_kerb` validated structures
- Pointer to a list of server side `auth_kerb` validated structures

Following is a description of circumstances in which it is desirable and circumstances in which it is necessary for the Cray Research NFS server to access this information:

- When acids rather than user IDs are being used for disk accounting and/or file quotas.

In this case, NFS ID mapping is desirable, but not necessary. Acids are unique to UNICOS and therefore are not passed across the network as part of the NFS protocol. When files are created on the NFS server through NFS, the acid given to the file is the acid in the user structure of the `nfsd` process that does the first write operation to the file. Because acids are not part of the credentials in the NFS request, the acid attached to any file created across NFS is the acid of the running `nfsd` process (`root` 's default acid). This defeats disk quotas and disk accounting based on acids. Because ID maps contain the user's default acid, the NFS server can use this information when ID mapping occurs. A user cannot change the acid in the ID maps.

- When UNICOS security is enabled (with or without the IP security option (IPSO) enabled).

  In this case, NFS ID mapping is necessary. When UNICOS security information is required on the Cray Research system, and is passed across the network (through IPSO), the NFS server must validate the NFS requests based on the security information for the user making the request. The ID maps contain such security information.

  > **Note:** `MAP_THRU` ID mapping domains do not contain the required security information for UNICOS security, and cannot be used.

- When file systems or directories have been exported with the `krb` (Kerberos authentication required) export option in the `/etc/exports` file.

  In this case, NFS ID mapping is necessary. Users are required to run the `nfsid` command from the NFS client machine to the Cray Research NFS server machine to gain access to those file systems that have been exported with the `krb` option. The information from which users have been validated through Kerberos from certain Internet addresses is kept in the ID maps.

  > **Note:** The `kerberos` and `krb` operands used with the `exportfs` command are not supported on the Cray ML-Safe configuration of the UNICOS system.

- When file systems or directories in the `/etc/exports` file have been exported by using the `exportfs` command with the `-o` option and the `kerberos` operand (`auth_kerb` RPC authentication required). In this case, NFS ID mapping is necessary.

Following is a description of circumstances in which it is necessary for a Cray Research NFS client to access information contained in ID maps.

- When the `mount`(8) command with the `-o` option and `kerberos` operand (`auth_kerb` RPC authentication required) is used to mount an NFS file system. In this case, NFS ID mapping is necessary.

3.1.6.9 Special `MAP_THRU` NFS ID Map

If you need access to the information kept in ID maps (see the circumstances listed in the previous section), you must create a special ID map called a `MAP_THRU` map if one of the following situations is true:

- You have been running with NFS ID mapping and you have `MAP_THRU` ID mapping domains.
- You were not previously running with NFS ID mapping. In this case, you must also set up `MAP_THRU` ID mapping domains for all hosts and networks that are using the Cray Research system as an NFS server or client.

Typically, `MAP_THRU` ID mapping domains do not use a kernel ID map. However, if the `MAP_THRU` ID map is defined in the kernel, all `MAP_THRU` ID mapping domains use it. The special `MAP_THRU` ID map is built from the `/etc/passwd` file and `/etc/group` file from the local machine only. Following is a sample `nfsmerge` command to be added to the `Merge.domains` script that builds the ID map file:

```
nfsmerge -u /etc/uidmaps/u.cray.cray -g /etc/uidmaps/g.cray.cray /etc/passwd
/etc/passwd /etc/group /etc/group
```

The ID map file built by this command is the same ID map file that can be used with `pcnfsd` as described in Section 3.1.6.7, page 272). Therefore, if you are running `pcnfsd` with NFS ID mapping and you are using the special `MAP_THRU` NFS ID map, you need to execute the `nfsmerge` command only once.

To add the special `MAP_THRU` ID map to the kernel, add the following to the `Set.domains` file:

```
/etc/uidmaps/nfsaddmap -M /etc/uidmaps/u.cray.cray
```

To determine whether the `MAP_THRU` ID map is defined in the kernel, use the following command:

```
nfsidmem -v | grep MAP_THRU
```

If there are `MAP_THRU` ID mapping domains defined when the `MAP_THRU` ID map is loaded into the kernel, those ID mapping domains are also converted to use the `MAP_THRU` ID map. Conversely, if the `MAP_THRU` ID map is removed from the kernel, all `MAP_THRU` ID mapping domains (which point to the

MAP_THRU ID map) are converted back to standard MAP_THRU ID mapping domains (which do not point to an ID map). The special MAP_THRU ID map is the only ID map that can be removed from the kernel with the nfsrmmap command while ID mapping domains are referencing it. The following command adds the host address to the ID mapping domain:

```
/etc/uidmaps/nfsaddhost -l hostname
```

3.1.7 Configuring NFS Parameters

You can change NFS configuration parameters, such as the size of the rnode table, in several ways. All configurable NFS parameters appear in the /usr/src/uts/cf. *xxxx* /config.h file; you can change them by editing this file and building a new kernel. You can change the NFS parameters at boot time by entering appropriate entries in the network section of the system parameter file. You can use the UNICOS ICMS to make these changes, or you can make them manually.

3.1.7.1 Changing the config.h File

Table 3, Configurable NFS parameters, lists and describes the configurable parameters in the config.h file:

Table 3. Configurable NFS parameters

Name	Default value	Description
NFS_MAXDATA	32768	Maximum number of bytes of user data read or written.
NFS_NUM_RNODES	256	Number of NFS rnodes. Each active NFS file or directory requires an rnode.
NFS_PRINTINTER	0	Time interval in tenths of seconds between service not responding messages appearing on the console.
NFS_STATIC_CLIENTS	8	Number of permanently allocated client handles. Each NFS request to a server requires a client handle.
NFS_TEMP_CLIENTS	8	Number of temporarily allocated client handles. These client handles are destroyed when freed.

Name	Default value	Description
`CNFS_STATIC_CLIENTS`	8	Number of permanently allocated client handles for sending requests to a server whose file system is mounted with the `cray` mount option.
`CNFS_TEMP_CLIENTS`	8	Number of temporarily allocated Cray NFS client handles.
`NFS_MAXDUPREQS`	1200	Number of entries in the duplicate request cache. This value should be large enough so that the request entry is still present when the first retry of that request arrives.
`NFS_DUPTIMEOUT`	3	Time interval in seconds after the original request, during which duplicate requests received by the server are not reprocessed.

3.1.7.2 Changing the NFS Parameter File

You can change the configurable NFS parameters at boot time by placing entries in the network section of the system parameter file, `/etc/config/param`. The parameter names are the same as in the `config.h` file, except that they appear in lowercase.

Following is an example of the network section of a system parameter file:

```
network {
        .
        .
        .
        .
        512 nfs_num_rnodes;
        16  nfs_static_clients;
        16  nfs_temp_clients;
        .
        .
        .
        .
}
```

3.1.8 General Security Concerns

Although UNICOS NFS is an excellent tool for sharing files between computer systems, it also makes the files on a server vulnerable to unauthorized access. A user with `root` access on a workstation and a knowledge of how UNICOS NFS works can pretend to be any user on the network and thereby gain access to server files that would not otherwise be accessible to that user. For example, the `/etc/exports` file and the `/etc/mountd` process are convenient mechanisms for providing the information needed by legitimate NFS clients. However, this information can usually be obtained by other means, making it possible to bypass the access controls that the `/etc/exports` file provides.

ID mapping provides some additional security by restricting access to NFS to those network addresses specified in the ID maps.

Because the standard NFS protocol was not designed with the use of access control lists (ACLs) in mind, access across NFS to files that use ACLs can be denied unexpectedly. (This does not occur if you use the `cray` option of the `mount`(8) command when processing between two Cray Research systems.) The NFS client checks the UNICOS permissions and sends the request to the server based on those permissions. However, the NFS server checks the ACL entries and grants or denies the request according to the procedures provided in the description of the UNICOS security feature in *General UNICOS System Administration*, publication SG–2301.

An additional security concern is the execution of `setuid` programs. An individual with `root` permission on a workstation or fileserver can create a `setuid root` program that can then be executed on a UNICOS NFS client. The `nosuid` argument to the `-o` option on the UNICOS `mount`(8) command prevents this operation.

The basic security mechanisms in UNICOS NFS are as follows:

Security mechanism	Description
Export control	Administrators can choose to restrict the list of hosts allowed to mount Cray Research file systems through the `exports`(5) file.
Mount control	Administrators control both the remote systems from which they import file systems and the permissions used on the UNICOS directories on which mounting is done through `mount`(8) options.
Standard UNICOS file permission checking	User and group ownerships and read, write, and execute-search permissions operate the same way on UNICOS NFS file systems as they do on UNICOS file systems. Users and administrators

	concerned about security should fully understand these mechanisms.
Kerberized NFS	Each NFS request is sent using the `AUTH_KERB` flavor of RPC. This RPC flavor protects packets by including an encrypted time stamp and other information on each packet. Users must have a valid Kerberos granting ticket prior to making NFS `AUTH_KERB` transactions.

Another level of security can be implemented through the ID mapping facility of UNICOS NFS. Administrators who elect to make exported file systems globally accessible (in the `/etc/exports` file) can impose restrictions through selective inclusion of remote addresses in the ID mapping domains, and they can further restrict access on those systems by the inclusion or exclusion of users in a particular domain’s user or group ID map. (See Section 3.1.6.4.4, page 261, for more details.)

For information regarding avoiding mandatory access control (MAC) violation risks in the creation and use of NFS ID maps, see ‘‘ID map file maintenance,’’ Section 3.1.6.4.3, page 260.

3.1.8.1 NFS and UNICOS Security

Note: This section describes NFS in a UNICOS security environment; however, no additional considerations exist for NFS with a secure environment, except that if you are running the Cray ML-Safe configuration of the UNICOS system, you must enable the Internet Protocol Security Option (IPSO).

When you are running UNICOS with UNICOS security, information about the sensitivity label of client users and server files must be communicated between the client and server hosts. No provision exists for this communication in the NFS protocol that UNICOS uses. To solve this problem, UNICOS places an interpretation on the labels of the datagrams that contain the NFS requests and responses.

When an NFS request is required, the client sends it at the label of the process that is attempting to access the file. The server uses the label of the request to perform mandatory access checks. The label must be a valid label for both the client host (through the NAL), and the user making the request. Valid user label ranges are stored in the kernel NFS ID maps. For this reason, ID mapping is required on UNICOS systems that run UNICOS security. See Section 3.1.6, page 254, for information on setting up ID maps.

After the server processes the request, the response packet is sent labeled with the sensitivity label of the file being accessed. If this label is invalid for the client host, the response is dropped.

When the client receives a response from the server, the label on the datagram is used as the sensitivity label of the file for any mandatory access checks that the client NFS software performs.

It is important to note that this scheme works only if both the client and the server can unambiguously determine the labels on datagrams passed between them. They can do this only if the systems use IP security labeling or the hosts are single-label hosts. If you are running with the `NETW_STRICT_B1` kernel option, NFS access is supported with any system that uses IP security options.

Warning: If you are running UNICOS security without the `NETW_STRICT_B1` kernel option (thus allowing multilabel ranges for hosts that do not use IP security options), do not export file systems to or mount file systems from any host that does not use IP security options and has a multilabel range in the NAL. UNICOS NFS is unable to correctly enforce MACs with such a configuration.

Note: Do not hard-mount file systems if one of the following is true:

- The Cray Research system is a system with UNICOS security and an NFS client.
- The NFS client system is mounting from a Cray Research NFS server system with UNICOS security.

With UNICOS security, requests that fail MAC do not receive responses; therefore, NFS requests on hard-mounted file systems hang.

Parameters in the `config.h` file that are supported with UNICOS security and NFS are as follows:

`NFS_SECURE_EXPORT_OK`	When set to a nonzero value, this parameter allows labeled file systems to be exported (used by the NFS server).
`NFS_REMOTE_RW_OK`	When set to a nonzero value, this parameter allows a Cray Research NFS client to mount a remote file system in read-write mode. When set

to 0, NFS mounts the file system in read-only mode.

Note: If you are running NFS with `SECURE_MAC` enabled, the address associated with the `localhost` interface must be defined as `syslow` to `syshigh` levels and all compartments are in the Network Access list (NAL) (see `spnet`(8)). If `SECURE_MAC` is not enabled, the address associated with the `localhost` interface must be defined as 0 through 16 and all compartments are in the NAL. Under the Cray ML-Safe configuration of the UNICOS system, `SECURE_MAC` is enabled.

3.1.8.2 Kerberos Authentication

Note: Kerberos authentication is not supported with the Cray ML-Safe configuration of the UNICOS system.

Kerberos authentication can be required for NFS access to exported UNICOS file systems through the `krb` operand of the `exports`(5) command. This export option requires users to register with `mountd`, using the `nfsid`(1) command on the client machine from which they want NFS access to exported UNICOS file systems. ID maps are required to support the `krb` export option and are described in Section 3.1.6, page 254. For more information, see the `nfsid`(1) command.

The following examples show the mapping option (`-m`), which registers the user with `mountd` and the unmapping option (`-u`), which removes the user's registration:

```
nfsid -m remote_host_name
nfsid -u remote_host_name
```

Users who have not executed the `nfsid` command are granted only `others` access to files in file systems exported with the Kerberos option.

3.1.8.3 Kerberized NFS

Kerberized NFS uses the `AUTH_KERB` kernel RPC for NFS requests. Each NFS request contains additional information, which is validated by the NFS server's kernel.

UNICOS file systems may be exported by using the `exportfs`(8) command with the `-o` option and the `kerberos` operand. These exported file systems are mounted by using the `mount`(8) command with the `-o` option and `kerberos` operand. The super user, or user as root, must have an unexpired Kerberos

Ticket Granting Ticket (TGT) before executing the `mount` command. A TGT is obtained by using the `kinit`(1) command.

Note: The `krb` and `kerberos` operands of the `exportfs` command may not be used at the same time.

Machines running Kerberized NFS must have an NFS principal entry in the `/etc/srvtab` file. Your Kerberos database must contain a principal of `nfs` and an instance of `hostname`. You must install a new `/etc/srvtab` file with the `nfs` principal prior to running Kerberized NFS.

The following guidelines must be understood in order to run Kerberized NFS:

- Your Kerberos database must contain specific information. For example, if your host name is `harpo`, you must have a principal of `nfs` and an instance of `harpo` in your Kerberos database. The `srvtab` entry would list `nfs` as the service and an instance of `harpo`.
- The `kerbd`(8) daemon must be running. `kerbd` handles requests from kernel level NFS and sends the requests to and from the Kerberos key distribution center (KDC).
- The user must have an unexpired TGT prior to attempting access to a Kerberos NFS mounted file system.
- A root user ticket must be regenerated every 21 hours. This is due to a limitation in the current implementation. All file systems mounted with the automounter and Kerberos NFS mount options are affected by this limitation.

3.1.9 UDP Checksum

The standard NFS implementation does not calculate user datagram protocol (UDP) checksums for the packets exchanged between NFS clients and servers. However, situations occur in which checksumming might be desirable, such as when the network is suspected to be error prone. Therefore, on UNICOS systems, checksumming for NFS is implemented in two parts: client and server.

To enable client-side checksumming, use the `cksum` argument on the `mount`(8) command. This argument causes the client to calculate and verify the checksums for all UDP packets sent to the server of this file system. However, this does not ensure that the server will also calculate and verify the checksum. You should confirm that the server in question verifies incoming checksummed packets.

The UNICOS NFS server automatically calculates and verifies the checksum for incoming checksummed packets. To enable server-side checksumming for outgoing packets, use the `cksum export` option within the `/etc/exports` file. However, this argument does not ensure that the NFS client that receives the packet will calculate and verify the checksum.

3.2 Troubleshooting

When a network service is not performing properly, the trouble usually lies in one of the following areas (listed from most likely to least likely):

- The network access control policies do not allow the operation, or architectural constraints prevent the operation.
- The NFS client software or environment is malfunctioning.
- The NFS server software or environment is malfunctioning.
- The network between the NFS server and client is malfunctioning.

The following sections offer a checklist for determining the location of the problem, some common problems, and a list of `mount`(8) command error messages.

Before trying to debug UNICOS NFS, read the man pages from the following lists that are relevant to your NFS environment:

Table 4. NFS man pages

Server	Client	ID mapping
`mountd`(8)	`mount`(8)	`nfslist`(8)
`nfsd`(8)	`automount`(8)	`nfsidmem`(8)
`exports`(5)	`biod`(8)	`nfsuid`(8)
`exportfs`(8)	`fstab`(5)	`nfsckhash`(8)
`portmap`(8)	`portmap`(8)	`nfsmerge`(8)
`rpcinfo`(8)	`rpcinfo`(8)	`nfsidmap`(8)
`pcnfsd`(8)	`nfsid`(1)	`nfsaddmap`(8)
		`nfsaddhost`(8)

Server	Client	ID mapping
		`nfsclear`(8)
		`nfsadduser`(8)
		`nfsgid`(8)
		`nfsrmhost`(8)
		`nfsrmmap`(8)

3.2.1 Isolating the Problem

The following sections contain a checklist to help you either resolve the problem or isolate the problem (that is, help identify the environment in which the problem occurred). This checklist is sequential, and it verifies whether all of the basic functions required for NFS are working. Use this checklist as a starting point if you have no idea where the NFS problem is occurring.

The checklist is grouped into the following topics:

- NFS mounting problems
- Problems accessing NFS mounted files
- Problems with ID mapping

If you are having problems configuring NFS for the first time, use the complete checklist. Individual items within the checklist can be used at any time to help isolate or resolve problems when NFS is already up and running.

This checklist assumes that an NFS client is having problems mounting or accessing NFS files from an NFS server.

3.2.1.1 NFS Mounting Problems

If the `mount`(8) command times out, perform the following steps:

1. Ensure that the NFS server machine is up and that you can access the NFS server from the NFS client. On the NFS client, use the `ping`(8) command, as follows:

   `ping` *server_hostname*

2. Ensure that the NFS server machine is at least running either `portmap`(8) or `rpcbind`(8), `mountd`(8), and `nfsd`(8) daemons. On the NFS server machine, use the `ps`(1) command, as follows:

   ```
   ps -ae | egrep 'portmap|rpcbind|mountd|nfsd'
   ```

   Following is sample output from the `ps` command:

   ```
   668   -    0:02 portmap
   1347  -    0:00 mountd
   1343  -    0:00 nfsd
   1341  -    0:00 nfsd
   1342  -    0:00 nfsd
   1344  -    0:00 nfsd
   ```

   Only one `portmap` and `mountd` daemon should be running at any one time, but one or more `nfsd` daemons can be running at one time (a typical number is 4).

   If more than one `mountd` daemon is running, conflicts regarding the `mountd` requests can occur and the client mount requests will not be serviced.

   If `portmap` or `rpcbind`(8) is not running, stop the `mountd`, `nfsd`, and `pcnfsd`(8) daemons. You must also stop any other RPC registered servers that are running. Then start the `portmap` or `rpcbind` daemon. After this daemon is started (the `ps -ae` command shows `portmap` or `rpcbind` running), start `mountd` and the `nfsd` daemons.

   Note: If the NFS server is a Silicon Graphics (SGI) system and the Cray Research system is the client, the `mountd` daemon on the SGI system must be started with the `-n` option. Sun Microsystems has the `-n` option set by default within their startup scripts; other systems may vary.

3. Ensure that `portmap`(8) or `rpcbind`(8), `mountd`(8), and `nfs`(4P) are registered RPC services on the NFS server machine. On the client, use the `rpcinfo`(8) command, as follows:

   `rpcinfo -p` *server_hostname*

   Following is partial sample output from the `rpcinfo` command:

   ```
   program vers proto   port
   100005    1   tcp    678  mountd
   100005    1   udp    676  mountd
   100003    2   udp   2049  nfs
   ```

```
          :
          :
          :
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
```

There are two entries for `mountd` and `portmapper`, one for `udp`, and the other for `tcp`. NFS has only one entry, because it uses only `udp` as the transport protocol.

If one or more of these programs are not registered RPC programs, either step 2 has failed, or you should kill the `portmap` or `rpcbind`, `mountd`, and `nfsd` daemons (along with any other RPC registered programs) and restart them. Start `portmap` or `rpcbind` first; after it has begun, start the `mountd` and `nfsd` daemons, along with any other RPC daemons.

The `-u` option of the `rpcinfo` command can also be run on the client to determine whether these servers are registered and responding through `portmap`. The `-u` option uses user datagram protocol (UDP) to the specified server on the specified program, as in the following example:

```
rpcinfo -u server_hostname 100005 1
```

The program number for `mountd` is 100005.

Following is sample output from the `rpcinfo` command:

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

4. Ensure that the file system or directory you are trying to mount is exported on the NFS server giving the client permission to mount. Use the following command only if you are running as `root`:

   ```
   /etc/exportfs
   ```

   The `exportfs` command without options prints the currently exported file systems or directories.

   If you are not running as `root`, use the following command:

   ```
   cat /etc/xtab
   ```

   This file is updated by `exportfs`; thus, it also shows the current list of exported file systems or directories.

> **Note:** Viewing the `/etc/exports` file does not necessarily show the currently exported file systems or directories; therefore, it should not be used to determine whether something is exported.

5. Ensure that the `/etc/hosts` files are accurate on both the NFS server and client. Check to verify that the entry in the `/etc/hosts` file for the server is the same on both systems (that is, the server's host name used on both systems points to the same Internet address), and perform the same check with the entry for the client on both systems.

   > **Note:** If you are running the domain name server, using the `named`(8) daemon, the `/etc/hosts` file is not accessed and you should use either the `nslookup`(1) or the `host`(1B) command to identify the host entry. See the man pages for details. If you are using NIS, use the `ypcat HOSTS | grep` *host* command, where *host* is the name of the host or machine you want to access.

   See the troubleshooting steps in Section 3.2.1.3, page 290, if you are using ID mapping and either the NFS server or client has multiple network interfaces (therefore, multiple `/etc/hosts` file entries).

3.2.1.2 Problems Accessing NFS Mounted Files

After the NFS system is mounted, do the following if you cannot access these files:

1. Ensure that the file system or directory is still mounted on the NFS client. Use the following `mount`(8) command without options to list all currently mounted local and NFS file systems:

   ```
   /etc/mount
   ```

   The `rsize` and `wsize` options of the `mount` command specify the number of bytes in the read buffer and the write buffer, respectively (they are typically set to the same value). Ensure that these options are correct.

   To determine these values, check the exact `mount` command used (you may need to look in `/etc/fstab` to determine the exact options used). If an `rsize` or `wsize` option is not used with the `mount` command, these values are set to a default size. On UNICOS systems, the default size is 8 Kbytes. However, the maximum size, 32 Kbytes, is set by the `NFS_MAXDATA` kernel variable in the `config.h` file. A similar variable should exist on systems that are not UNICOS systems. Contact the appropriate vendor for this information.

Following are some suggestions for setting `rsize` and `wsize` on the `mount` command under various configurations:

- Cray-to-Cray NFS environment

  For UNICOS systems, the default value for read and write buffers is 8 Kbytes, which is the maximum buffer size for many other systems. However, if you are running UNICOS NFS between two Cray Research systems, you should set `rsize` and `wsize` to the maximum value of 32 Kbytes. With Cray Research systems running earlier releases of UNICOS, it is not necessary to specify `rsize` and `wsize`, because the default is 32 Kbytes.

- Cray Research NFS client and other vendor's NFS server

  The `rsize` and `wsize` values set on the Cray NFS client, or the default value set if these options were not specified, must not exceed the maximum read or write buffer size of the NFS server. A typical maximum value for other vendors is 8 Kbytes. If you are running UNICOS 9.0 or later, the client default value is 8 Kbytes; therefore, it is not necessary to specify the `rsize` and `wsize` options, unless the NFS server's limit is less than 8 Kbytes. However, if you are running an earlier UNICOS release, the default value is 32 Kbytes; therefore, you must specify appropriate values for `rsize` and `wsize`.

  For example, if a UNICOS 6.0 system is the NFS client and a Sun system is the NFS server, you could use the following `mount` command:

  ```
  /etc/mount -t NFS -o bg,soft,rsize=8192,wsize=8192 sun:/usr /usr/sun
  ```

  Contact your vendor for the maximum buffer size (`NFS_MAXDATA`) supported. See Section 3.4, page 303, for more details.

- Cray Research NFS server and other vendor's NFS client

  In this configuration, the limiting factor is again the other vendor's maximum buffer size (`NFS_MAXDATA`). See Section 3.4, page 303, for more details on `rsize` and `wsize`.

2. Ensure that the access options and the directory and file permission settings on the NFS server and client are not the problem.

On the NFS server, check the following `exportfs`(8) or `exports`(5) options to see whether the file system or directory was exported with any options that affect permissions:

Option	Description
`ro`	Export read-only.
`rw=` *hostname*	Export read-write to *hostname* only.
`anon=` *uid*	User ID for unknown user.
`root=` *hostname*	Give `root` access to *hostname* only.
`krb`	Kerberos authentication required for access.

See the `exportfs`(8) and `exports`(5) man pages for more detail.

On the NFS client, check the permissions of the following:

- The mount point. If the file system or directory is already mounted, you must unmount it to obtain the actual mount point directory permission settings. At a minimum, the mount point directory permissions should be set to 555.
- The mounted files or directories that are having trouble being accessed. Check the owner and group name and permission settings to see whether this might be the problem.

  Ensure that you have a flat administrative environment in which all user IDs are the same across all systems, or that you are running NIS, or that you have UNICOS ID mapping set up and running. See the following section for more information on ID mapping troubleshooting). If you are not using UNICOS ID mapping (that is, you do not have map-through ID mapping domains), ensure that ID mapping is disabled by using the `/etc/uidmaps/nfsidmap -d` command. You must have `root` permission to use this command.

3. Ensure that enough memory buffers (mbufs) are available. Use the following command to obtain an mbuf count:

   ```
   /etc/netstat -m
   ```

   See Chapter 2, page 3, and `netstat`(1) for more details on mbufs and their use.

3.2.1.3 Problems with ID Mapping

If the Cray Research system is an NFS server or NFS client, UNICOS ID mapping can occur. It is recommended that if both server and client are Cray Research systems, ID mapping should occur on the NFS server. You can perform the following troubleshooting steps on either an NFS server or client, depending on where the ID mapping is occurring.

1. Ensure that ID mapping is enabled. Output from the following command (without options) specifies whether ID mapping is enabled or disabled:

   ```
   /etc/uidmaps/nfsidmap
   ```

2. Ensure that an ID mapping domain exists for the system experiencing problems. Use the following command to list the ID mapping domains that contain the *hostname* address:

   ```
   /etc/uidmaps/nfslist -a hostname
   ```

   The host name specified in this command should be the same host name specified on the `mount`(8) command.

   If multiple network interfaces (that is, multiple network paths) are between the NFS server and client systems, ensure that an ID mapping domain is set up for the correct interface or for all such interfaces, if routing is not static.

3. Ensure that ID maps and corresponding hash tables are accurate. Use the following command (without options) to check the consistency of the user ID and group ID maps in the kernel:

   ```
   /etc/uidmaps/nfsckhash
   ```

   If this command indicates any problems, follow the procedure to rebuild your ID maps, as previously described.

4. Ensure that the user(s) are set up correctly. During the actual `mount` command operation, `root` is the user. If the Cray Research system is experiencing access problems when trying to mount an NFS file system, `root` is the affected user. You should not have any exceptions listed in an exceptions file for user `root`. For example, although `root` must be present in an exceptions file for mounting, you should not attempt to map `root` to the bad user ID.

   After the file system is mounted, any user can try to access that file system. Ensure that the user entries, such as group lists, are in the required ID map and that the entries contain the most current information. Use the following command to print the user entries:

```
/etc/uidmaps/nfsuid -m map_name list_of_user_names
```

The specified map name should be the map listed as output from the `nfslist` command described in step 2.

If a map-through map was created for the domain in question and you are not running UNICOS security, you do not need to perform this command.

5. Because the UDB allows user entries with no group ID, it creates password files with empty group ID fields. Because the `nfsmerge`(8) command does not accept an entry without a group ID field, you might receive the following message if you execute `nfsmerge`:

   ```
   nfsmerge: WARNING!! Could not read entire password file.
   There is probably an invalid entry (no gid?).
   The resulting ID map file may be incomplete.
   ```

   For mapping to work, a user must have a group ID. If this error occurs, assign the user a group ID.

3.2.2 NFS Mount Failure

This section consists of an example of an NFS mount, followed by a list of error messages with explanations. If your `mount`(8) command fails for any reason, check the generated error messages for information about possible solutions.

3.2.2.1 NFS Mount Example

The `mount`(8) command can get its parameters from the command line or from the `/etc/fstab` file. The following example assumes command-line arguments, but the same debugging techniques work if `/etc/fstab` is used. Look at a sample mount request made from a client machine:

```
mount -t NFS krypton:/usr/src /krypton.src
```

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

The following is a list of steps the `mount` command takes to mount a remote file system, as in the previous example:

1. The `mount`(8) command parses the first argument into host `krypton` and remote directory `/usr/src`.
2. The `mount` command determines the Internet Protocol (IP) address of `krypton`.
3. The `mount` command calls the `krypton mountd` (8) program and passes `/usr/src` to it.
4. The `krypton` server's `mountd` command reads `/etc/exports` and looks for the exported file system that contains `/usr/src`.
5. The `krypton` server's `mountd` command expands the host names and network groups in the export list for `/usr/src`.
6. The `krypton` server's `mountd` command gets a file handle for `/usr/src` from the operating system.
7. The `krypton` server's `mountd` command returns *fhandle*.
8. The `mount`(8) command performs an NFS `mount`(2) system call with the file handle and the `/krypton.src` directory.
9. The NFS `mount`(2) system call determines whether the caller is a super user and whether `/krypton.src` is a directory.
10. The NFS `mount`(2) system call does a `statfs`(2) system call to `krypton` 's UNICOS NFS server (`nfsd`).
11. The `mount` command opens the `/etc/rmtab` file and appends an entry to it.

3.2.2.2 NFS Mount Failure Error Messages

Any one of the steps in the previous section can fail, some of them in more than one way. Following are specific error messages, along with descriptions of the failures associated with each.

```
/etc/fstab:  No such file or directory
```

The `mount`(8) command tried to search for the name in `/etc/fstab`, but `/etc/fstab` did not exist.

```
mount:  ...  Block device required
```

You probably omitted the `krypton`: part of the following request:

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

The `mount`(8) command assumes that you are doing a local mount, unless there is a colon in the file system name or the file system type is `NFS` in `/etc/fstab`.

```
mount:  directory path must begin with /
```

The second parameter to `mount` identifies the path of the specified directory. This must be a full path name beginning with `/`.

```
mount:  ...:  No such file or directory
```

Either the remote directory or the local directory does not exist. Check the spelling, and use `ls`(1) to request a listing of each directory.

```
mount:  ...:  Not a directory
```

Either the remote path or the local path specified is not a directory. Check the spelling, and use `ls`(1) to request a listing of each directory.

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

If `mount` is called with either a directory or a file system name, but not both, it looks in `/etc/fstab` for an entry whose file system or directory name field matches the argument on the command line. For example, the following entry results in a search of `/etc/fstab` for a line that has a directory name field of `/krypton.src`:

```
mount /krypton.src
```

Assume that an entry such as the following is found:

```
krypton:/usr/src /krypton.src NFS rw,soft,rsize=8192,wsize=8192
```

The mount is then performed as though you had typed the following:

```
mount -t NFS -o rw,soft,rsize=8192,wsize=8192 krypton:/usr/src /krypton.src
```

If you see this message, it means that a match for the argument you gave to `mount` was not found in any of the entries in `/etc/fstab`.

```
mount:  not in export list for ...
```

In the `/etc/exports` file, your machine name is not included in the export list for the file system you want to mount. You can look at the file either by logging in to the server system or by using `remsh`(1B). If you are to mount the system, add your machine names to the relevant exports list, or the list of machine names should be null, indicating that any machine can mount the file system.

```
mount:  ...:  Not owner
```

You must do the mount as super user on your machine because it affects the file system configuration for the whole machine, not just for you.

`mount: ...: Permission denied`

This message generally indicates that some authentication failed on the server. It could simply be that your machine name is not included in the appropriate `/etc/exports` list (see the preceding message) or that the server could not determine who you are. Possibly, the server does not acknowledge that you are who you say you are. Check the server's `/etc/exports` file.

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

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

```
rpcinfo -p hostname
```

You should see a list of registered program numbers. If you do not, you might have to restart the `portmap`(8) daemon. If you cannot perform a remote login to the server, but the server is up, you should check your network connection by trying a remote login to some other machine. You should also check the server's network connection.

`mount: ... server not responding: RPC_PROG_NOT_REGISTERED`

This message indicates that `mount` got through to the portmapper, but the NFS mount daemon (`mountd`) was not registered.

`amp;... unknown host`

The host name you supplied could not be found in `/etc/hosts`. First check the spelling and the placement of the colon in your `mount` call. If the spelling and syntax are correct, try `ping`(8) on the local machine (client) and on another machine to determine whether the remote host is responding.

For UNICOS, `mountd`(8) is typically started from `/etc/netstart`. Use `sdaemon`(8) to start it manually. Generally, it can be restarted by simply entering the following command:

```
/etc/mount mountd
```

3.2.3 Hanging Programs

If programs hang while they are performing file-related work, your server might be dead. In this case, you might see the following message on your console:

`NFS server` *Internet address* `not responding, still trying`

This problem originates with either one of your servers or the network. If the problem is with a server, you can determine which server is malfunctioning by locating the address provided in the message in `/etc/hosts`. If your machine hangs completely, check the server(s) from which you have mounted. If one (or more) of them is down, your programs will continue automatically when the server comes back up. There will be no indication that the server died, and no files will be destroyed.

If a soft-mounted server dies, other work should not be affected. Programs that time out while trying to access soft-mounted remote files will fail with the `errno` external variable set to `ETIMEDOUT,` but you should still be able to work on your other file systems.

If all servers are running, check to see whether anyone else using the server or servers in question is having trouble. If more than one machine is having problems getting service, it is probably a problem with the server's daemon (`nfsd`(8)). Log in to the server and execute the `ps`(1) command to see whether `nfsd` is running and accumulating CPU time. If not, you might be able to terminate and then restart `nfsd`. If this does not work, you must reboot the server.

If no one else is having problems, check the network connection and the connection of the server.

If your machine comes up partially after a boot, but it hangs where it would usually be doing NFS mounts, one or more servers is probably down or a problem exists with your network connection.

3.2.4 No Super-user Access over the Network

Under UNICOS NFS, a server exports the file systems it owns so that clients can mount them remotely. When you become a super user on a client, you are denied access on remotely mounted file systems. Consider the following example:

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

Now, retry it as super user.

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

The problem usually appears during the execution of a `setuid root` program. Programs that run as `root` cannot access files or directories, unless the permission for `other` allows it.

Also, if the server is not an NFS server and the `export` (5) file or the `export` option on the `exportfs`(8) command does not allow your workstation `root` access, you cannot change ownership of remotely mounted files. Because, in this case, users cannot perform a `chown`(1) command, and the super user is treated as a standard user on remote access, no one but the super user on the server can change the ownership of remote files. For example, if you try to execute `chown` as yourself on new program `a.out`, which must be `setuid root`, it will not work, as demonstrated in the following example:

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

To change the file ownership, you must log in to the server as a super user and then make the change. Alternatively, 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.

3.2.5 File Operations Not Supported

Remote file systems support file locking if the daemons `lockd`(8) and `statd`(8) are running. By default, file locking is supported.

If you do not want file locking, mount the file system using the `nolock` option on the `mount`(8) command.

Append mode and atomic writes are also not ensured to work on remote files that are accessed simultaneously by multiple clients.

3.2.6 Remote Device Access Not Supported

Under UNICOS NFS, you cannot access a remotely mounted device or any other character or block special file, such as a named pipe.

3.3 Confidence Testing

The UNICOS NFS confidence test suite is provided to ensure the proper installation of UNICOS NFS. These tests are used by system analysts and administrators. Two types of test groups are included in the test suite: functional and performance.

Functional tests verify that specific features are working as expected. If the functional tests fail, you should look for errors in installation or configuration. The functional test group includes the following groups of tests:

Test group	Description
Basic	Determines whether UNICOS NFS is providing basic functionality
General	Checks some commonly used applications such as compiles, `nroff`(1), and `tbl`(1)
Special	Analyzes specific facilities that have required special attention in the past

Cray Research client	Analyzes facilities that have required special attention on Cray Research systems as NFS clients

After the functional tests pass, you can look at performance to determine whether the system can be fine-tuned to run faster. The performance test provides file transfer rate measurements.

All confidence tests are self-checking. Error messages are provided to help you determine the source of a problem. The tests described in this section are designed to be compiled and executed on client machines running the UNICOS operating system. No test suite software runs on the server.

During the execution of the test suite on a client machine, directories and files are created along a separate path called the *test directory*. The test directory usually consists of the path to a remote directory that was mounted on the client from a server. Optionally, the test suite can mount a remote directory on the local machine.

3.3.1 Installation

The `/usr/src/net/nfs/tests` directory contains the confidence test suite source code. This source code should be copied to a user directory before it is compiled. The `/usr/src/net/nfs/tests/Makefile` makefile can be used to distribute the test source code to a new location, as follows:

```
su root
make dist DESTDIR=destination_path_for_test_source
```

After placing the source tree in the desired location, use the `Makefile` makefile to compile the tests. Execute the following command in the destination directory:

```
cd destination_path_for_test_source
make all
```

To move the compiled tests to a new location, use the following makefile:

```
su root
make copy DESTDIR=destination_path_for_compiled_tests
```

When the test suite is created successfully and is in its execution location, the permission mode and ownership of binary file `domount`, which exists in the root of the test suite tree, must be changed. The super user must own `domount`, and the `setuid` bit must be set. As super user, enter the following commands:

```
chown root domount
chmod 4555 domount
```

3.3.2 Test Execution

Execute the test suite as follows:

1. As super user, mount the desired NFS directory.

   Example:

```
mount -t NFS -o rw,soft,wsize=8192,rsize=8192 cray2:/tmp /usr/tmp/mount
```

2. As a standard user, set the `NFSTESTDIR` environment variable to a subdirectory of the mounted directory. For example, use the mount point in the previous example, as follows:

   ```
   setenv NFSTESTDIR /usr/tmp/mount/craytest
   ```

   If the test directory already exists, the test suite deletes it and its contents (`craytest`, in this example).

3. Modify the `TESTS` and `TESTARG` variables in the `tests.init` file to indicate the type of functional test you want to run, as follows:

   - Set the `TESTS` variable to indicate which of the following tests you want to run:

Variable	Test type
`TESTS="-b"`	Runs the basic tests
`TESTS="-g"`	Runs the general tests
`TESTS="-s"`	Runs the special tests
`TESTS="-c"`	Runs the Cray Research client tests
`TESTS="-a"`	Runs all of the preceding tests (default)

   - Set the `TESTARG` variable (which affects only the basic tests) to run an abbreviated functional test, as follows:

Variable	Test type
`TESTARG="-f"`	Causes the basic tests to run a shorter functional test. Use this variable in

	conjunction with `TESTS="b"` to run a quick-look test of NFS.
`TESTARG="-t"`	Causes the basic tests to run a longer test with timing functions (default).

- Run the functional tests with the `runtests` command, as follows:

  ```
  runtests
  ```

  The `runtests` script (located in the root directory of the test tree) is invoked on the client machine. If the test directory exists, you are given the following message and choice:

  ```
  The NFSTESTDIR directory /usr/tmp/mount/craytest exists.
  The NFS tests expect to create this directory.
  Remove the existing directory (and its contents!)
  and continue with the NFS tests (n[y])?
  ```

  To delete the directory and continue, type **y**; to abort the test and leave the directory intact, type **n**. `n` is the default.

- Run the performance test as a separate test. For meaningful performance numbers, the client machine, the network, and the server machine must be dedicated to the test. Run the test as follows:

  ```
  nfsperf  [-l num_times_to_loop_thro_test]
           [-d results_directory]
           [-n num_of_concurrent_tests]
           [-s file_size]
           [-f num_of_files]
           testdir
  ```

Option	Description
`-l`	Determines the number of times to repeat each test; default is 1.
`-d`	Indicates that the `results_directory` directory is to be used to store the results files.
`-n`	Indicates the number of concurrent tests; default is 1.
`-s`	Indicates the size of files that will be created during the tests in multiples of 32 Kbyte blocks; default is 50.

`-f`	Indicates the number of files to create for each test; default is 5.
testdir	Name of directory that contains the tests. This parameter is optional if the `NFSTESTDIR` environment variable has been set.

3.3.3 Test Configurations

The following test configurations are suggested for UNICOS NFS confidence testing in a Cray Research environment:

1. Run the test suite on a Cray Research machine, specifying `NFSTESTDIR` as a local (not remotely mounted) directory. This provides a baseline set of results (and timings if the `-t` option is selected for the basic tests).

   With this configuration, the tests run against standard file space, providing baseline timings (speed of the Cray Research file system).

2. Make the Cray Research machine both the server and the client machine by mounting a local file system onto another local directory, using `localhost` as the *server_name* argument on the `mount`(8) command.

3. Mount a file system from a remote server machine onto the Cray Research system. The test directory then exists on a remote machine.

3.3.4 Executing Individual Tests

You can execute any individual test by setting the `NFSTESTDIR` environment variable to the name of a directory within the mounted directory. For example, if a remote directory is mounted on local test directory `/usr/tmp/mount`, specify a new subdirectory, such as `/usr/tmp/mount/craytest`. Then execute the following command:

```
setenv NFSTESTDIR /usr/tmp/mount/craytest
```

To run a basic test, change to the basic test directory, check the specific test for required arguments, and then run the test, as follows:

test_name arguments

For example, to run functionality test `test3` from the basic section, execute the following commands:

```
cd basic
test3 -f
```

To run any of the tests other than the basic tests, you must first copy the tests to the mounted test directory.

Example:

```
mkdir $NFSTESTDIR
make copy DESTDIR=$NFSTESTDIR
cd $NFSTESTDIR
rename 100
```

3.3.5 Cleaning up

If the `runtests` script is used to run the test suite, all files and directories created in the test directory are removed at the completion of testing. However, the tester must clean up the test directory after running an individual test.

The executable files in the test suite can be removed by using the following command:

```
rm -rf $NFSTESTDIR
```

The super user must remove executable file `domount`.

3.3.6 Test Contents

Basic tests create and remove files and directories, obtain and set file attributes, perform look-up functions, read and write files, read directory entries, and obtain file system statistics.

Special test functions include checking access to open files that have had their modes changed, checking replies lost on nonidempotent requests, performing exclusive create functions, performing seeking functions to a negative offset, repeatedly renaming files, creating and accessing files with holes (data blocks not allocated), and taking proper `umask`(1) action on remote files.

Performance tests provide a file transfer rate benchmark. Several files are created from the mounted file system, and then data is written to and read from those files.

General file system tests include compiles, simultaneous compiles, doing a `makefile`, nroffing a file, or using `tbl`. They also provide timing information for performance measures.

Cray Research client tests include tests of special areas that are unique to UNICOS, including asynchronous I/O, truncation tests, and additional `umask` tests.

Please send comments and suggestions concerning the UNICOS NFS test suite to the following:

Cray Research, Inc. Software Division Network and Communications Quality Group 655F Lone Oak Road Eagan, MN 55121

Additional tests to be added to the suite are welcome.

3.4 Performance and Tuning

NFS is a synchronous protocol designed for reliable remote access. The synchronous characteristic means that for each NFS request sent, a response must be received (indicating the completion of the request) before another NFS request can be sent. This characteristic is one of the primary reasons NFS is not as fast as other TCP/IP applications. NFS uses UDP/IP, which is a connectionless, unreliable transport protocol (NFS does not use TCP). Therefore, NFS runs best over reliable local area networks.

Many NFS performance factors are related to the manner in which UNICOS TCP/IP networking parameters were tuned or optimized. Particularly affecting performance are maximum transmission units (mtus) (see Section 2.3.1, page 128); memory buffers (mbufs) (see Section 2.3.2, page 133); and network routing (see Section 2.3.3, page 154).

The following sections describe factors that affect NFS performance and methods for obtaining performance figures.

3.4.1 Factors That Affect NFS Performance

The following sections describe factors that can affect NFS performance and offer guidelines for increasing performance.

3.4.1.1 `NFS_MAXDATA` Parameter

The `NFS_MAXDATA` parameter (defined in the `config.h` file) defines the maximum size of the data part of a remote NFS request. By default, `NFS_MAXDATA` is set at 32 Kbytes (32,768 bytes) on UNICOS systems. This parameter has the greatest effect on NFS servers, but it also affects NFS clients. For example, a Cray Research system acting as an NFS server can receive a maximum of 32 Kbytes of data from an NFS client request. However, the maximum amount of data an NFS client can send is determined by the value of its `NFS_MAXDATA` variable. This makes the limiting factor the system with the smallest `NFS_MAXDATA` size. Generally, the more data that can be sent and received at one time, the better the performance. Therefore, it may be advantageous to increase `NFS_MAXDATA` on systems that are not Cray Research systems.

> **Note:** It is not known how all other systems define this parameter, or whether the parameter can be changed on all of these systems. Contact the appropriate vendor for such information.

3.4.1.2 `mount` Command Arguments

The `mount`(8) arguments `rsize` and `wsize` can have a direct effect on NFS performance. You can set these parameters when issuing the `mount` command on the NFS client. However, `rsize` and `wsize` cannot be set above the client's `NFS_MAXDATA` size, and they should not be set greater than the server's `NFS_MAXDATA` value.

The `rsize` parameter specifies the read buffer size in bytes (the maximum amount of data the NFS client can accept from an NFS read request); the `wsize` parameter specifies the write buffer size in bytes (the maximum amount of data the NFS client will send in an NFS write request). Generally, `rsize` and `wsize` are set to the same value. A guideline to use in setting these parameters is to set them as large as possible, but not to exceed the smaller of the `NFS_MAXDATA` size of the client or server. If these mount parameters are not specified, the default used is based on the client's default buffer size.

For Cray Research systems running UNICOS 8.0, the default buffer size (default `rsize` and `wsize`) is 8 Kbytes, with a maximum of 32 Kbytes (`NFS_MAXDATA`). Most systems that are not Cray Research systems have both their default and maximum buffer sizes set to 8 Kbytes. Therefore, generally, NFS client systems do not need to specify `rsize` and `wsize`; instead, the default of 8 Kbytes is used. One exception to this is when NFS is being run between two Cray Research systems; for better performance, you should set `rsize` and `wsize` on the `mount` command to 32 Kbytes instead of using the default of 8 Kbytes.

3.4.1.3 NFS Daemons

Other factors that affect NFS performance are the number of NFS daemons (`nfsd`(8)) running on the NFS server and the number of block I/O daemons (`biod`(8)), if any, running on the NFS client. The `nfsd` s daemons are used on the NFS server to respond to requests from NFS clients for access to exported file systems. The `biod` daemons are used on the NFS client to allow for applications to use asynchronous block I/O. The `biod` daemons on the UNICOS system provide write-behind capabilities on behalf of the application.

The typical number of `nfsd` daemons running on a UNICOS NFS server system is 4; the typical number of `biod` daemons running on a UNICOS NFS client system is 4. If your system, acting as an NFS server, is going to be used as a fileserver for the rest of the network, more `nfsd` daemons may be required. However, it is not clear how to determine whether you are running enough of these daemons. You must use a trial-and-error method in which you determine whether adding more daemons makes any improvements to NFS performance.

Note: Adding more `nfsd` or `biod` daemons does not improve performance of a single stream, but it does affect the overall performance of multiple users. However, you should run some `biod` daemons, because running `biod` daemons increases the performance of a single write stream. You should also note that `biod` daemons do use mbufs (see Chapter 2, page 3, for details on setting the number of mbufs).

3.4.1.4 File System Configuration and `ldcache`(8)

One of the major limiting factors for any NFS server is its performance in accessing data (disk I/O performance). Many times this is the only limiting factor and therefore, performance never improves above the NFS server's disk I/O performance. For Cray Research systems acting as NFS servers, disk I/O performance is affected by whether `ldcache`(8) is configured and how it is used. If a Cray Research system is acting as an NFS server with a heavy NFS access load, using `ldcache` can greatly increase performance.

However, you should be aware of some factors. Ensure that using `ldcache` for NFS data access does not negatively affect local I/O performance (see the *UNICOS Configuration Administrator's Guide*, publication SG–2303, for more information on `ldcache` and configuring disks). Also, `ldcache` violates the NFS stateless protocol. For example, if the NFS server goes down, it should not affect the NFS client, except for a delay in processing until the NFS server comes back up. However, if the NFS server uses `ldcache` and the NFS server goes down, it may affect the NFS client, in that any NFS data that was changed or added by the client may not have been written to disk before the crash; the NFS client will not be aware that the changes were not made to the disk. Of course, similar concerns also affect local I/O requests using `ldcache`.

3.4.1.5 Network Speed

The network speed of channel devices such as the 10 Mbit/s Ethernet, 100 Mbit/s FDDI, 50 Mbit/s HYPERchannel, or 800 Mbit/s HIPPI affects performance. Generally, the faster the network, the better the performance. However, there can be exceptions because network speed is not the only factor that affects performance. One network can be faster than another network, but it can have higher overhead that decreases performance.

3.4.1.6 Network Configuration and Load

Performance is poor for any network application if the network is overloaded or poorly configured. You should check your networks for signs of overload (for example, a high number of collisions) by using such utilities as `netstat`(1B) to evaluate the state of your network, and `netperf`(8) and `nfsstat`(8), which display NFS/RPC statistics.

You should also try to configure your networks to reduce the number of gateway hops that are required to get from an NFS client to an NFS server.

3.4.1.7 NFS Server/client Configuration and Load

If a system will be an NFS server for several NFS clients with heavy access, it should be configured as a dedicated NFS server system and be used only minimally for other purposes. If an overloaded system is acting as either an NFS server or client, performance will be poor.

3.4.2 Obtaining NFS Performance Figures

NFS performance figures can be obtained by running the NFS performance test, `fileperf`, included in the NFS source directory, `/usr/src/net/nfs/tests/nfsperf`. Tests should be run with the Cray Research system acting as an NFS server and run again with the Cray Research system acting as an NFS client. See Section 3.3.6, page 302, for more details.

If possible, you should run the same tests through a similar network configuration, between two systems that are not Cray Research systems, to obtain figures that can be compared to the Cray Research system performance figures.

To obtain peak performance numbers for a particular configuration, follow these guidelines:

- Use the fastest networks available for NFS access (see the previous section for exceptions to this guideline).
- Ensure that the client machine, the network, and the server machine are dedicated for the test.
- Eliminate (or at least reduce) gateway hops.
- Set the value of `NFS_MAXDATA` on the NFS server and client to the optimal size. By default, Cray Research systems set `NFS_MAXDATA` to 32 Kbytes; it is desirable, but not always possible, to match this value on the other system. See Section 3.4.1.1, page 303, for more information on setting this value.
- Set the mtu for the interface to the optimal size (see Chapter 2, page 3, for details on setting the mtu size).
- Use `ldcache`(8) on Cray Research systems, or comparable caching methods on other systems, for the file systems on the NFS server.
- Run `biod`(8) daemons on the NFS client system.

NFS peak performance figures are usually no better than 50% of peak performance rates obtained with `ftp` (in some cases, NFS figures may be considerably less than 50%). One of the major factors for these statistics is that NFS is a synchronous protocol.