UNICOS Shared File System (SFS) Administration

SG–2114 1.5

New Features

UNICOS Shared File System (SFS) Administration SG–2114 1.5

This document includes additions throughout describing the administration of shared file systems on GigaRing-based I/O configurations.

Record of Revision

The date of printing or software version number is indicated in the footer.

Version	***Description***
9.0	August 1995 Original Printing.
1.5	April 1997 Printing for UNICOS/mk release 1.5 to support GigaRing-based I/O configurations.

Contents

Preface

This manual documents the UNICOS Shared File System (SFS) feature. This manual provides general information for system administrators who manage the operation of any Cray Research computer system running the UNICOS operating system.

Related publications

For detailed information on UNICOS installation procedures, see the following Cray Research, Inc. (CRI) publications:

- *UNICOS Installation Guide for the CRAY EL Series*, publication SG–5201
- *UNICOS Installation / Configuration Menu System User's Guide*, publication SG–2412

For detailed information on UNICOS system administration, see the following Cray Research, Inc. (CRI) publications:

- *General UNICOS System Administration*, publication SG–2301
- *UNICOS Resource Administration*, publication SG–2302
- *UNICOS Configuration Administrator's Guide*, publication SG–2303
- *UNICOS Networking Facilities Administrator's Guide*, publication SG–2304
- *UNICOS NQS and NQE Administrator's Guide*, publication SG–2305
- *Kerberos Administrator's Guide*, publication SG–2306
- *UNICOS Tape Subsystem Administrator's Guide*, publication SG–2307

Ordering Cray Research publications

The *User Publications Catalog*, publication CP–0099, describes the availability and content of all Cray Research hardware and software documents that are available to customers. Cray Research customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a document, either call the Distribution Center in Mendota Heights, Minnesota, at +1–612–683–5907, or send a facsimile of your request to fax number

+1–612–452–0141. Cray Research employees may send electronic mail to `orderdsk` (UNIX system users).

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
7D	DWB-related information
8	Administrator commands

Some internal routines (for example, the `ddcntl`() routine) do not have man pages associated with them.

variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.

[ ]	Brackets enclose optional portions of a command line.
...	Ellipses indicate that a preceding command-line element can be repeated.

The following machine naming conventions may be used throughout this document:

Term	Definition
Cray PVP systems	All configurations of Cray parallel vector processing (PVP) systems, including the following: CRAY C90 series CRAY C90D series CRAY EL series (including CRAY Y-MP EL systems) CRAY J90 series CRAY J90se series CRAY T90 series CRAY Y-MP E series CRAY Y-MP M90 series
Cray MPP systems	All configurations of the CRAY T3E series
All Cray Research systems	All configurations of Cray PVP and Cray MPP systems that support this release

The default shell in the UNICOS and UNICOS/mk operating systems, referred to in Cray Research documentation as the *standard shell*, is a version of the Korn shell that conforms to the following standards:

- Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) Standard 1003.2–1992
- X/Open Portability Guide, Issue 4 (XPG4)

The UNICOS and UNICOS/mk operating systems also support the optional use of the C shell.

Cray UNICOS Version 9.0 is an X/Open Base 95 branded product.

Reader comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. You can contact us in any of the following ways:

- Send us electronic mail from any system connected to the Internet, using the following Internet address:

  `publications@timbuk.cray.com`

- Contact your Cray Research representative and ask that a Software Problem Report (SPR) be filed. Use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.

- Call our Software Publications Group in Eagan, Minnesota, through the Customer Service Call Center, using either of the following numbers:

  1–800–950–2729 (toll free from the United States and Canada)

  +1–612–683–5600

- Send a facsimile of your comments to the attention of “Software Publications Group” in Eagan, Minnesota, at fax number +1–612–683–5599.

We value your comments and will respond to them promptly.

Introduction to Shared File Systems [1]

The UNICOS Shared File System (SFS) is a software feature available on Cray Research computer systems that allows multiple Cray Research systems to perform both read and write operations on one or more shared file systems. A shared file system provides most of the features present in the `NC1FS` file system, such as device striping, mirrored file systems, and asynchronous I/O.

Note: Mirroring is not supported on CRAY T3D or CRAY T3E systems.

To users, a UNICOS file system defined as a shared file system (file system type `SFS`) appears the same as an `NC1FS` file system, except that users can access the file system from any Cray Research system that is configured to use that shared file system. The Cray Research systems that share file systems and the media that contains the shared file systems constitute an *SFS cluster*.

Cray Research systems in an SFS cluster can mount any of the file systems defined as shared that are configured on the shared media accessible to the machines in that cluster.

All Cray Research systems that use a shared file system are peers. In an SFS configuration, there is no server system that accesses file systems on behalf of client systems. File system integrity is maintained through an external arbitration device that manages access to file data and file system structures.

1.1 SFS configurations

Different system configurations can take advantage of the UNICOS Shared File System feature. The following list gives some examples:

- **Fault tolerant configurations** of multiple Cray Research systems can be built where the tasks being performed by a system that fails (and the tasks are stored in data and command files on a shared file system) can be taken over and recovered by another machine in the SFS cluster when the system failure is detected.
- **Work Sharing SFS clusters** can be built where a queue of work to be done is maintained (either by Cray Research's Network Queuing Environment or by custom software) and the data and result files needed to perform the work reside on one or more shared file systems. Any machine in the SFS cluster can then work on any of the tasks because all the data and other programs needed to complete the items in the work queue are available to all the machines in the SFS cluster.

- **Division of Labor:** some machines in an SFS cluster may be producers of data while other systems in the SFS cluster may be consumers of that data. This would be typical of a configuration involving, for example, a CRAY C90 compute server which writes data files requiring a good deal of computation to produce to a shared file system also accessed by smaller CRAY J90 or CRAY Y-MP series systems, which might be well suited to further refining, rendering, archiving, or otherwise post-processing the data.

  In general, large file transfers are best suited for SFS operations.

1.2 Basic hardware requirements

SFS file systems are supported on CRAY T3E series, CRAY T3D series, CRAY T90 series, CRAY C90 series, CRAY EL series, CRAY J90 series, and CRAY Y-MP E series systems.

The primary hardware requirements for an SFS cluster are the following:

- *Shareable media*, such as Model E SSD storage devices (SSD-Es) or ND-12/14/30/40 High Performance Parallel Interface (HIPPI) disks. SSD storage devices may be shared, providing that the mainframes and SSD storage devices can be positioned close enough that VHISP channels may be used to interconnect the clustered mainframes and shared SSD storage devices.

  When ND-12/14/30/40 HIPPI disks are used as the shareable media, a set of switched HIPPI channels are used to allow the computers to access the SFS arbiter and the HIPPI disks.

- A *Shared File System arbiter*, used by UNICOS SFS to control access to the shared media. A HIPPI-SMP (H-SMP) can be used as the SFS arbiter. The H-SMP is a moderate performance, low-cost semaphore device that can coexist on a HIPPI switch with HIPPI disks and other devices. It is accessed by the Cray Research systems in the SFS cluster by using an extension to the IPI-3/HIPPI standard that provides semaphore functionality.

  The UNICOS operating system supports SFS configurations that contain more than one arbitration device associated with a particular Cray Research system. For example, the UNICOS operating system can support a configuration in which one arbitration device is required for a set of file systems shared by systems A and B, and a second arbitration device is required for a different set of file systems shared by systems B and C. In this configuration, system B accesses two different arbitration devices, each controlling a unique pool of file systems.

1.3 Licensing

To use the UNICOS SFS feature you must obtain a software license. Licenses for the UNICOS SFS feature are maintained and administered through the Flexible License Manager (FLEXlm) product. The license keys and installation instructions are sent to you by electronic mail.

Summary Procedure [2]

This chapter provides a summary of the steps you need to perform to configure, start up, and shut down an SFS environment. These steps are described in detail in later chapters.

2.1 Configuring the SFS environment

You configure an SFS environment and define `SFS` file systems with the following steps. To perform these steps with the aid of the UNICOS installation and configuration menu system, see Chapter 3, page 7. To perform these steps manually, see Chapter 4, page 25.

1. On each Cray Research system in the SFS cluster, define the interfaces to the semaphore device (`/dev/smp` and `/dev/sfs`).
2. On each system in the SFS cluster, define the Shared Mount Table.
3. Define and create the `SFS` file systems. You create an `SFS` file system on only one system in the SFS cluster.
4. Add the `SFS` file systems you have defined to the `/etc/fstab` file of each system in the cluster.
5. Create the `/etc/config/sfs` file, which includes an entry for each Cray Research system in the SFS cluster, indicating which SFS arbiters are accessible by that system, and an entry for each SFS arbiter, indicating the path names of the character special devices that support the arbiter.

2.2 SFS startup

After configuring the SFS environment and creating the `SFS` file systems, you initialize the SFS environment by executing the `/etc/sfs_start` command. This command initializes the SFS environment, checks and mounts the `SFS` file systems, and initiates the system lock recovery daemon. The `/etc/sfs_start` command is described in detail in Chapter 5, page 33.

2.3 Shutting down an SFS environment

To shut down an SFS environment, invoke the `/etc/sfs_stop` command. This command unmounts the shared file systems and kills the SFS daemon processes.

Configuring the SFS Environment with the Menu System [3]

This chapter describes how to configure Cray Research systems to use the SFS feature and how to create shared file systems by using the UNICOS installation and configuration menu system. If you want to configure your SFS software manually, see Chapter 4, page 25.

The menu system does not support GigaRing-based I/O configurations.

For this chapter, it is assumed that your Cray Research systems are up and running, and that they support the menu system.

When initializing the SFS environment with the menu system, you perform the following tasks:

1. Configure the SFS device nodes on one system in the SFS cluster
2. Indicate which hosts systems are connected to the SFS device nodes
3. Create the SFS device nodes and the SFS configuration file you have defined
4. Define the disk device nodes for the SFS cluster
5. Define the file systems for the SFS cluster
6. Copy the configuration information from the system where it has been defined to each of the systems you want to add to your SFS cluster
7. Correct machine-specific information for each of the systems that receives a copy of the configuration information

3.1 Before You Begin

Before you run the UNICOS installation and configuration menu system to configure your SFS environment, you should be sure that you have all the information at hand that the menu system will request.

For each SFS semaphore device that you configure, you will need to provide the following information:

```
/dev/hdd/smp I/O cluster (IOC) number         ___
/dev/hdd/smp IOP number                       ___
/dev/hdd/smp I/O channel number               ___
/dev/hdd/smp Ifield                           ___
/dev/smp port number                          ___
/dev/dsk/slr minor device number              ___
/dev/hdd/slr minor device number              ___
/dev/hdd/slr HIPPI disk device type           ___
/dev/hdd/slr I/O cluster (IOC) number         ___
/dev/hdd/slr IOP number                       ___
/dev/hdd/slr IOP Channel number               ___
/dev/hdd/slr starting sector number           ___
/dev/hdd/slr number of sectors                ___
/dev/hdd/slr HIPPI facility address           ___
/dev/hdd/slr HIPPI Ifield                     ___
SFS Arbiter                                   ___
```

You will also need to provide an associated host name for each SFS semaphore device that you define, on a screen in the following format:

```
Host name                                     ___ Connected Arbiter
```

For each disk that you will use in an SFS configuration, you will be asked for the following information:

```
Device name                                   ___
Device type                                   ___
I/O cluster (IOC) number                      ___
IOP number                                    ___
IOP Channel number
Alternate path?                               ___
Alternate I/O cluster (IOC) number
Alternate IOP number
Alternate IOP channel number
Unit                                          ___
Size in blocks
HIPPI Facility (Unit bits 0-8)                ___
HIPPI Raid Partition (Unit bits 9-15)         ___
HIPPI disk Ifield                             ___
SSD spare memory chip configuration file
```

The device names are limited to 14 characters. ND devices can be configured as HD-16, HD-32, and HD-64 devices. HD-16 is used for HIPPI disk devices with a 16 kbyte sector size, HD-32 is used for HIPPI disk devices with a 32 kbyte sector size, and HD-64 is used for HIPPI disk devices with a 64 kbyte sector size.

For each physical slice on a disk, you will be asked for the following information:

```
Slice name                                   ___
Physical device name                         ___
Minor device number                          ___
Starting number                              ___
Size of slice                                ___
Slice unit of measure is depicted in         ___
```

For each logical device you are configuring, you will be asked for the following information:

```
Logical device name                          ___
Logical minor number                         ___
Member device type                           ___
Member device name(s)                        ___
```

For each SFS file system you are defining, you will be asked for the following information:

```
Logical disk device name                     ___
File system mount point                      ___
File system type                             SFS
Backup frequency (in days)                   ___
File system check pass number
Read only option
Quota control file
User defined option
Mount at multi-user startup?
Mount point owner                            ___
Mount point group                            ___
Mount point mode                             ___
```

Note that you need to specify SFS as the file system type.

3.2 Initializing the SFS environment on the first system

To initialize the SFS environment on the first system in the SFS cluster, follow this procedure:

1. Verify that the existing disk and file system configuration information in the menu system is valid and complete. The SFS configuration information is appended to the existing system information.

2. Start the UNICOS installation and configuration menu on the first system in the cluster from the `/etc/install` directory using the special `./configsfs` program rather than the standard `./install` program.

   Note: You cannot run the `configsfs` program and the standard `install` program at the same time, unless you are running one of the programs in read-only mode. If you are running one of these utilities to configure the system, and you want to run the other to configure the system, you must first exit the utility you are currently running.

   When you run the `configsfs` program, the following screen appears. The default values of the SFS configuration screens are the values entered at the last invocation of the menu system.

```
                        Shared File System (SFS) Configuration

A-> Configure SFS semaphore device nodes ==>
    SFS hosts ==>
    Create SFS device nodes ...

    Disk configuration ==>
    File system (fstab) configuration ==>
    Encapsulate SFS device information ...

    Merge SFS devices to Disk Configuration ...
    Remove SFS devices from Disk Configuration ...

    Remove SFS device nodes ...
    Import root mount point



    Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI
```

3.2.1 Configuring SFS device nodes

To configure the SFS semaphore device nodes, perform the following steps:

1. Go to the `Configure SFS semaphore device nodes` menu.

```
Shared File System (SFS) Configuration
    ->Configure SFS semaphore Device Nodes
```

   A form screen appears indicating which SFS arbiters have been defined for this system.

```
                         Configure SFS Semaphore Device Nodes

      hdd/smp IOC  hdd/smp IOP  hdd/smp channel  hdd/smp Ifield  smp port  dsk/s
      -----------  -----------  ---------------  --------------  --------  --- >



     Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



         File "cfdb/sfsnode.cfg" is empty. Use the ’n’ key to add a record.
```

2. To add a new arbiter, press the `n Return` keys. The following screen appears, showing a new record which you must edit.

```
                         Configure SFS Semaphore Device Nodes

 S-> /dev/hdd/smp I/O cluster (IOC) number        0
     /dev/hdd/smp IOP number                      0
     /dev/hdd/smp I/O channel number              030
     /dev/hdd/smp Ifield                          0
     /dev/smp port number                         0
     /dev/dsk/slr minor device number             1
     /dev/hdd/slr minor device number             1
     /dev/hdd/slr HIPPI disk device type          HD16
     /dev/hdd/slr I/O cluster (IOC) number        0
     /dev/hdd/slr IOP number                      0
     /dev/hdd/slr IOP Channel number              030
     /dev/hdd/slr starting sector number          0
     /dev/hdd/slr number of sectors               0
     /dev/hdd/slr HIPPI facility address          2
     /dev/hdd/slr HIPPI Ifield                    0
     SFS Arbiter                                  __change__



     Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



                                    Record 1 of 1
```

This record view requests machine-specific information that is used by the system to define the `/dev/hdd/smp`, `/dev/smp`, and the `/dev/hdd/slr` device nodes. The screen first appears with default values, which you must modify for your system.

The `/dev/hdd/smp` node is the HIPPI-disk node that contains the low-level I/O path and the HIPPI I-field that are used for physically accessing the SFS arbiter.

The `/dev/smp` node is the low-level interface to the semaphore device.

The `/dev/hdd/slr` node defines the area of the shared media pool that you will use for the Shared Lock Region, an area that is used by UNICOS SFS to record common information such as semaphore assignment and system state.

`SFS Arbiter` indicates the user-provided name of the SFS arbitration device you are defining.

Note: The `SFS Arbiter` entry must be a number. For the first arbitration device, set this field to 0; for the second arbitration device, set this field to 1, and so forth.

3. Enter the requested information for the device node you are configuring.

   Note: You must modify any field that appears as `__change__` to create a functional SFS cluster.

4. Go to the `Shared File System Hosts` record screen.

   ```
   Shared File System (SFS) Configuration
       ->SFS Hosts
   ```

   The `Shared File System Hosts` form appears as follows:

```
                              Shared File System Hosts

       host name  arbiter name
       ---------  ------------



      Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



          File "cfdb/sfshost.cfg" is empty. Use the 'n' key to add a record.
```

5. To enter the host name, press the `n Return` keys, and the `Shared File System Hosts` record appears.

```
                              Shared File System Hosts

  S-> Host name                                         __change__
      Connected Arbiter                                 __change__



      Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



                                   Record 1 of 1
```

6. Enter the requested information. This information will be used to create the `/etc/config/sfs` file.

   Note: You must modify any field that appears as `__change__` to create a functional SFS cluster.

   For each SFS arbiter selection that you defined with the `Configure SFS Semaphore Device Nodes` menu, you must define at least one associated host with the `Shared File System Hosts` menu.

   The name of the `Connected Arbiter` must match one of the numbers you defined as the `SFS Arbiter` in the `Configure SFS Semaphore Device Nodes` menu. If the host is connected to more than one SFS arbiter, then the list of names or numbers should be comma-separated.

7. Return to the main `Shared File System (SFS) Configuration` menu and execute `Create SFS device nodes`. This creates the SFS nodes that you have configured and the `/etc/config/sfs` file that contains the names of the SFS hosts and arbiters.

   The following is a sample of the output that selecting this action yields.

```
Corrected output of action SFS device nodes actions:


Preparing to create the needed SFS device nodes

Creating /etc/config/sfs hostfile
Done creating the host section of /etc/config/sfs hostfile
Preparing to create arbiter #0 SFS device nodes

Verifying /dev/hdd/smp-0’s existence
Node name /dev/dsk/smp-0 being created
Verifying /dev/smp-0’s existence
Node name /dev/smp-0 being created
Verifying /dev/hdd/slr-0’s existence
Node name /dev/dsk/slr-0 being created
Verifying /dev/dsk/slr-0’s existence
Node name /dev/dsk/slr-0 being created
Verifying /dev/sfs-0’s existence
Node name /dev/sfs-0 being created
Verifying /dev/smnt-0’s existence
Node name /dev/smnt-0 being created

Done making the SFS device nodes
Finished creating /etc/config/sfs
```

8. To verify that you have configured you device nodes correctly, you can run the `sfs_start` command from a different window. This will yield errors if there are problems with the device nodes.

   For information on the `sfs_start` command, see Chapter 5, page 33.

3.2.2 Configuring disk devices

To configure the disk devices for an SFS cluster, perform the following steps. This procedure is identical to the procedure for configuring disk devices for a standard file system.

1. Go to the `Disk Configuration` menu.

```
Shared File System (SFS) Configuration
    ->Disk Configuration
```

   The `Disk Configuration` menu appears as follows:

```
                                Disk Configuration

M-> Physical devices ==>
    Physical device slices ==>
    Logical devices (/dev/dsk entries) ==>
    Mirrored devices (/dev/mdd entries) ==>
    Striped devices (/dev/sdd entries) ==>
    Logical device cache ==>
    Verify the disk configuration ...
    Review the disk configuration verification ...
    Dry run the disk configuration ...
    Review the disk configuration dry run ...
    Update disk device nodes on activation?      YES
    Allow active root node updates?              NO
    Import the disk configuration ...
    Activate the disk configuration ...



    Keys:   ^? Commands   H Help    Q Quit   V ViewDoc   W WhereAmI
```

   The process of defining disk configuration for shared file systems is identical to the process of defining disk configuration for file systems that will not be shared.

First define the physical devices, then the physical device slices, then the logical devices.

Caution: The device names you define for shared file systems must be the same for all of the systems in the SFS cluster.

2. To define physical devices, go to the `Physical Devices` menu.

```
Shared File System (SFS) Configuration
    ->Disk Configuration
        ->Physical Devices
```

3. To enter a new physical device, press the `n` `Return` keys and the following screen appears.

```
                                  Physical Devices

S-> Device name                                      0130
    Device type                                      DD60
    I/O cluster (IOC) number                         0
    IOP number                                       0
    IOP Channel number                               030
    Alternate path?                                  NO
    Alternate I/O cluster (IOC) number
    Alternate IOP number
    Alternate IOP channel number
    Unit                                             0
    Size in blocks
    HIPPI Facility (Unit bits 0-8)
    HIPPI Raid Partition (Unit bits 9-15)
    HIPPI disk Ifield
    SSD spare memory chip configuration file
    Note:  The device names are limited to 14 characters.

    Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



                                  Record 1 of 1
```

4. Enter the information for each device you are configuring.

   When defining physical devices on CRAY EL series systems or CRAY J90 series systems, you do not need to define the I/O cluster number (IOC) or I/O processor number (IOP).

5. To define the physical device slices, go to the `Physical Device Slices` menu.

```
Shared File System (SFS) Configuration
    ->Disk Configuration
        ->Physical Device Slices
```

6. To enter a new physical device slice, press the `n Return` keys and the following screen appears.

```
                                Physical Device Slices

S-> Slice name                                      slice1
    Physical device name                            0130
    Minor device number                             1
    Starting number                                 0
    Size of slice                                   0
    Slice unit of measure is depicted in            sectors

    Notes:  Any SSD or RAM slices must have blocks as the unit.
            HIPPI disks are recommended to use blocks.
            The slice names are limited to 14 characters.

    Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



                                   Record 1 of 1
```

7. Enter the information for each device slice you are configuring.

   When defining physical device slices for `/dev/hdd` devices, you must ensure that the minor numbers are unique.

8. To define the logical devices, go to the `Logical Device (/dev/dsk) Configuration` menu.

```
Shared File System (SFS) Configuration
    ->Disk Configuration
        ->Logical Device (/dev/dsk) Configuration
```

9. To enter a new logical device, press the `n Return` keys and the following screen appears.

```
                        Logical Device (/dev/dsk) Configuration

  S-> Logical device name                           ldd1
      Logical minor number                          1
      Member device type                            hdd
      Member device name                            slice1

      Note:  The logical device names are limited to 14 characters.



      Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI




                                   Record 1 of 1
```

10. Enter the information for each device you are configuring.

    Select the `Mirrored devices (/dev/mdd entries)`, `Striped devices (/dev/sdd entries)`, and `Logical device cache` entries, as necessary, and provide the requested information.

11. Before leaving the `Disk Configuration` menu, make sure that the `Update disk device nodes on activation` is set to `YES`; you will not be able to complete the process of defining your SFS configuration otherwise.

12. Select the menu options to perform the following tasks, reviewing them as desired:

    - `Verify the disk configuration`
    - `Dry-run the disk configuration`
    - `Activate the disk configuration`.

    When you perform any of these tasks, you may receive the warning `gap/overlap <before slice sfs>`. This could indicate that the menu system has reserved space on the disk because the SFS feature requires a portion of the HIPPI device for the Shared Lock Region. If you receive additional `gap/overlap` warnings, however, you may need to check your configuration.

13. After configuring the disk devices, you can run the `ddstat`(8) command to verify that the configuration was successful.

3.2.3 *Defining an* SFS *file system*

You define an SFS file system just as you define a file system that is not to be shared, using the same `Standard File Systems Configuration` menu.

1. Select the `File System (fstab) Configuration` menu from the main `Shared File System (SFS) Configuration` menu.

   ```
   Shared File System (SFS) Configuration
       ->File System (fstab) Configuration
   ```

   The `File System (fstab) Configuration` menu appears as follows:

```
                           File System (fstab) Configuration

 M-> Standard file systems ==>
     NFS file systems ==>
     Shared file system ==>
     Import fstab configuration ...
     Activate fstab configuration ...

     Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI
```

2. From this menu, select the `Standard file systems` entry and the following screen appears.

```
                          Standard File System Configuration

       Device Name          Mount Point                         FsType  Freq  Pas
       -------------------  ----------------------------------  ------  ----  - >



      Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI
```

3. To add a new file system, press the `n` `Return` keys and the following screen appears.

```
                         Standard File System Configuration

 S-> Logical disk device name                            /dev/dsk/fs
     File system mount point                             /mnt
     File system type                                    SFS
     Backup frequency (in days)                          1
     File system check pass number                       2
     Read only option
     Quota control file
     User defined option
     Mount at multi-user startup?                        NO
     Mount point owner                                   root
     Mount point group                                   root
     Mount point mode                                    0755


     Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI



                                   Record 1 of 1
```

4. Enter your file system information just as you would for a file system that is not to be shared, with the exception that you use `SFS` as the `File system type` instead of `NC1FS`. A file system name must be the same on all the systems in the SFS cluster.

5. Go to the `File System (fstab) Configuration` menu:

```
Shared File System (SFS) Configuration
    ->File System (fstab) Configuration
```

6. To create the `/etc/fstab` file, select `Activate fstab configuration`.

3.2.4 Checking the SFS configuration

To check the SFS configuration, move to another window or exit the menu system and perform the following procedure:

1. Run the `mkfs`(8) command on the file systems you have created, being sure to use `-q` and the `-s #` options. The `-s #` option indicates that this is an SFS file system and specifies the number of semaphores to assign to the file system. For information on using the `mkfs` command to create shared file systems, see Section 4.2.3, page 30.

Warning: Do not run the `mkfs`(8) command on a mounted file system.

2. Bring up the SFS system to see if the environment has been successfully initialized by executing the `sfs_start` command. For information on the `sfs_start` command, see Chapter 5, page 33.

3.2.5 Encapsulating SFS configuration information

In order to copy the SFS configuration information to the other systems in the cluster, you must encapsulate the configuration information into a file. Use the following procedure:

1. Return to the menu system, go to the main `Shared File System (SFS) Configuration` menu, and select `Encapsulate SFS device information`. This creates the `sfs_encap` file in the `/etc/install` directory. This file contains the encapsulated device information that the other systems in the SFS cluster will use.
2. Exit the menu system.

3.3 Initializing the SFS environment on additional systems

For each additional system in the SFS cluster, perform the following steps.

1. Copy the `sfs_encap` file from the system on which you created it to the `/etc/install` directory of the other systems in the SFS cluster with the `ftp` or `cp` command. Note that you must change the permissions on the `sfs_encap` file before copying it.
2. Log in to a different system in the cluster and start the `configsfs` program from the `/etc/install` directory, which contains `sfs_encap`.
3. From the main `Shared File System (SFS) Configuration` menu, perform the action `Merge SFS devices to Disk Configuration`.
4. Go to the `Configure SFS Semaphore Device Nodes` menu.

```
Shared File System (SFS) Configuration
    ->Configure SFS semaphore Device Nodes
```

5. Provide the machine-specific information for the `/dev/hdd/smp`, `/dev/smp`, and `/dev/hdd/slr` device nodes that need to be changed for the current system.

Caution: When defining the physical device driver port for `/dev/smp`, take care to ensure that the port number is unique for every machine in the cluster. The valid port number range is 0 to 63. The menu system itself does not ensure that each `/dev/smp` port number is unique for each machine in the cluster.

6. Go to the `Physical Devices` menu.

```
Shared File System (SFS) Configuration
    ->Disk Configuration
        ->Physical Devices
```

The menu appears as follows:

```
                                 Physical Devices

      Name          Type  Ioc         Iop         Chn         Alt?  AltIoc  AltI
      ------------  ----  ----------  ----------  ----------  ----  ------  -- >
      0230.0        DD62  0           2           030         NO
      0334.16.0.13  HD16  0           3           034         NO
      0334.16.0.21  HD64  0           3           034         NO
 E->  0334.18.0.13  HD16  __change__  __change__  __change__  NO


     Keys:   ^? Commands   H Help   Q Quit   V ViewDoc   W WhereAmI


            Use the + and - keys to access other pages within this menu.
                            Record 4 of 4 Page 1 of 1
```

7. Change the information in the categories indicated with the text string `__change__`. The following fields will need to be changed:

   Column numbers 3, 4, 5, 12, 13, and 14 must have the correct information for each system's access to the SFS device. The specifications to be changed for SFS devices are:

   - I/O cluster (IOC) number
   - IOP number
   - IOP Channel number
   - HIPPI Facility (Unit bits 0-8)
   - HIPPI Raid Partition (Unit bits 9-15)

(If you will not be using this partition, leave the default values as they first appear or the system will not allow activation.)

- HIPPI disk switch address

8. Go back to the `Disk Configuration` menu.

```
Shared File System (SFS) Configuration
    -> Disk Configuration
```

9. Make sure that the `Update disk device nodes on activation` is set to `YES`.

10. Select the menu options to perform the following tasks:

    - `Verify the disk configuration`
    - `Dry-run the disk configuration`
    - `Activate the disk configuration`.

    When you perform any of these tasks, you may again receive the warning `gap/overlap <before slice sfs>`. This indicates the space that the menu system has reserved on the HIPPI device for the Shared Lock Region.

11. After configuring the disk devices, you can run the `ddstat`(8) command to verify that the configuration was successful.

12. Go to the `File System (fstab) Configuration` menu.

```
Shared File System (SFS) Configuration
    ->File System (fstab) Configuration
```

13. Select `Activate fstab configuration`. This creates the `/etc/fstab` file.

Remember to repeat steps 1 through 13 for each additional system in the SFS cluster.

Configuring the SFS Environment Manually [4]

This chapter describes how to configure a Cray Research computer system to use the Shared File System (SFS) feature and how to create SFS file systems manually, as an alternative to using the UNICOS installation and configuration menu system. If you configured the SFS environment by using the menu system, you can ignore this chapter.

The examples in this chapter initialize an SFS environment using HIPPI or GigaRing I/O-basedIdisks.

4.1 Configuring SFS device nodes

Configuring a Cray Research system to use the SFS feature involves the creation of several special-purpose device nodes for each file system that you define.

The following is a list of device nodes you will create when configuring the SFS environment, along with a brief description of the purpose of each device.

Device	Description
`/dev/smp`	The low-level raw interface to the semaphore device.
`/dev/hdd/smp`, `/dev/xdd/smp`	The IPI-3 and GigaRing driver nodes, respectively. They contain the information for physically accessing the semaphore device and are used by `/dev/smp`. This is named `smp` by convention.
`/dev/sfs`	The interface to the SFS software administration programs. This device driver is responsible for the management of the Shared Lock Region. The Shared Lock Region is a small piece of the shared media pool used by the SFS feature to record common information such as semaphore assignment, system state, etc.
`/dev/dsk/slr`	The path to the Shared Lock Region slice on the shared media, used in defining `/dev/sfs`.

`/dev/dsk/smnt`	The path to the Shared Mount Table, which is contained within the Shared Lock Region of the shared media pool.

4.1.1 Configuring the semaphore device

Configuring the semaphore device for shared file systems is a two-step process, which you perform for each system in the SFS cluster. You must use the same major and minor numbers for each system in the SFS cluster.

The first step is to create the device node which describes the I/O path in order to reach the semaphore device from the Cray Research system you are configuring.

The following is an example of using the `mknod` command to create this node for a HIPPI-based semaphore device:

```
/etc/mknod /dev/hdd/smp c 60 1 1 20 0 0 34 0 255 6769
```

Parameter	Description
`/dev/hdd/smp`	This is the node name being created. For GigaRing devices, this would more properly be named `/dev/xdd/smp`.
60	Major number for IPI-3 (`hdd`) disk driver. The major number for GigaRing I/O disk driver (`xdd`) is 33.
1	Minor number. Do not use 0 as the minor number for a `/dev/hdd` node, as this causes a conflict with the `hddmon` command.
1	Type. An HD-16 device is type 1, which indicates a 16 kbyte sector device. An HD-32 device is type 2, which indicates a 32 kbyte sector device. An HD-64 device is type 1, which indicates a 64 kbyte sector device. For GigaRing devices, use 0.
20	I/O path
0	Start
0	Length
34	Flags
0	Reserved
255	Unit

6769	I-field (smp 's address on the HIPPI network when accessed from this system) or 0 for a GigaRing device

Define the physical device driver, /dev/smp, as in the following example:

```
/etc/mknod /dev/smp c 73 0 4 0 /dev/hdd/smp 0 0
```

Parameter	Description
73	Major number for smp device.
0	Minor number
4	smp device type (4 indicates HIPPI semaphore, 6 is used for xdd).
0	Port number. On HIPPI-based semaphore systems, you must put a unique number in this field for every machine in the cluster. This number is used to uniquely identify each machine in the cluster.
/dev/hdd/smp	Path name to the hdd node created in the previous step (/dev/xdd/smp if using GigaRing channel).
0	Reserved
0	Reserved

⚠ **Caution:** When defining the physical device driver, take care to ensure that the port number is a unique number for every machine in the cluster.

You can execute the /etc/sema command to determine whether the semaphore box you have defined is available to the system.

⚠ **Caution:** Do not run the sema command after you have started the system daemons by running sfs_start.

The following shows an example of the sema command. This command executes a TEST command against semaphore 1 in the /dev/smp device and prints the results.

```
/etc/sema TEST 1 /dev/smp
```

4.1.2 Defining the sfs *device*

The sfs device must be configured on each system of the SFS environment.

To define the sfs device, first define the area of the shared media that you will use for the Shared Lock Region. To do this, define a physical disk device and then define a logical disk device which contains a single slice consisting of the physical disk device you have just created.

The following example defines a physical disk device:

```
mknod /dev/hdd/slr c 60 3 1 20 0 1024 040 0 16 6769
```

Parameter	Description
`/dev/hdd/slr`	This is the device being created. For GigaRing devices, this would more appropriately be `/dev/xdd/slr`.
60	Major number for IPI-3 disk driver. For GigaRin devices, this would be 33 for the GigaRing (`xdd`) disk driver.
3	Minor number for device
1	Type (1 indicates 16 kbyte sector device, 2 indicates 32 kbyte sector device, 3 indicates 64 kbyte sector device)
20	I/O path
0	Starting sector
1024	Number of sectors
040	flags
0	Reserved (must be 0)
16	Unit (facility ID)
6769	I-field (disk's address on the HIPPI network when accessed from this system). Set to 0 for GigaRing devices.

The following example defines the logical disk device containing the physical disk device defined in the previous example (for HIPPI).

```
mknod /dev/dsk/slr b 34 200 0 0 /dev/hdd/slr
```

The following command shows an example of `sfs` node configuration:

```
/etc/mknod /dev/sfs c 48 0 0 0 /dev/dsk/slr 0 0
```

Parameter	Description
48	Major number for `sfs` device
0	Minor number
0	Type
0	Reserved
`/dev/dsk/slr`	Path name to Shared Lock Region slice, defined in the previous step
0	Reserved

0	Reserved

4.1.3 Defining the Shared Mount Table

The Shared Mount Table takes its space from the area you have defined as the Shared Lock Region. Use the `mknod` command to define the Shared Mount Table configuration. As the following example shows, 75 is the major device number for the Shared Mount Table and is the only significant parameter for this device type. The remainder of the arguments can be set to 0.

```
/etc/mknod /dev/smntent c 75 0 0 0 0 0 0 0 0 0
```

4.2 Creating SFS file systems

You create SFS file systems the same way as you create standard NC1FS file systems. An SFS file system is laid out in the same way as an ordinary NC1FS file system. It may be as simple as a single slice, or it may use the logical device constructs that incorporate mirroring and/or striping.

Note: Mirroring is not supported on CRAY T3D or CRAY T3E systems.

4.2.1 Describing slices on a HIPPI disk

You describe the slices on a HIPPI disk the same way as you define a slice for a standard NC1FS file system. The following example defines an `hdd` slice, which has a major device number of 60 and a minor device number of 101.

```
/etc/mknod /dev/hdd/h01 c 60 101 1 01230 0 10000 0 0 2 7
/etc/mknod /dev/dsk/hfs01 b 34 120 0 0 /dev/hdd/h01
```

Caution: Device names and major/minor numbers must be the same across all of the systems in an SFS cluster. Further, minor numbers must be less then 256 and multiple partitions from the same device should not be SFS exported (See "SFS Restrictions and Limitations," Chapter 9, for details).

Only the I/O path and the I-field information may be different across systems. The rest of the disk definition must be identical for a given SFS file system across all systems in an SFS cluster.

4.2.2 Describing slices on a GigaRing-based disk

You describe the slices on a GigaRing FCN, MPN or HPN disk the same way as you define a slice for a standard `NC1FS` file system. The following example defines an `xdd` slice, which has a major device number of 33 and a minor device number of 101.

```
/etc/mknod /dev/xdd/x01 c 33 101 1 01230 0 10000 0 0 2 7
/etc/mknod /dev/dsk/xfs01 b 34 120 0 0 /dev/xdd/h01
```

Caution: Device names and major/minor numbers must be the same across all of the systems in an SFS cluster. Further, minor numbers must be less then 256 and multiple partitions from the same device should not be SFS exported (See "SFS Restrictions and Limitations," Chapter 9, for details).

Only the I/O path may be different across systems. The rest of the disk definition must be identical for a given SFS file system across all systems in an SFS cluster.

4.2.3 Creating a shared file system

You create an `SFS` file system by using the `mkfs`(8) command. All standard `NC1FS` options of `mkfs` can be used when defining an `SFS` file system.

When creating an `SFS` file system, you must use the `-s` option of the `mkfs` command. The `-s` option requires an argument, which defines the number of semaphores to be assigned to the system at mount time. This number is recorded in the superblock of the `SFS` file system.

An `SFS` file system requires multiple semaphores to operate. The first assigned semaphore is used to protect global file system related metadata, specifically the superblock, dynamic block, allocation map, and Inode allocation maps. Remaining semaphores are then used, in a hashed manner, to protect individual inode sectors during inode updates.

The more semaphores you assign to a file system, the better the performance. This is because more semaphores result in less contention for inode semaphores, allowing more simultaneous inode sector updates.

You cannot assign more than 2048 semaphores in total to shared file systems. If you create more than one shared file system, you must allocate your semaphores according to your anticipated needs for each file system. When apportioning semaphores among shared file systems, you should assign a greater percentage of available semaphores to the more heavily used file systems.

If you attempt to mount a file system that will bring the number of assigned semaphores over the limit of available semaphores, the file system will not mount and an error message will result. If this should occur, you can use the `-s` option of the `setfs` command to change the number of semaphores assigned to a file system that is not mounted. You can also use this option to increase the number of semaphores assigned to a file system if you are removing existing shared file systems.

4.2.4 Changing file system types

You can change a file system created as a shared file system (file system type SFS) to an NC1FS file system and, conversely, you can change a file system created as an NC1FS file system to an SFS file system by using the -s option of the setfs(8) command. This allows more flexibility and speed in system administration, supporting such actions as making and restoring a file system in non-shared (fully cached) mode, and then marking the file system as shareable.

For more information about using the -s option of the setfs command, see the setfs(8) man page.

4.2.5 Adding SFS entries to the /etc/fstab file

After defining an SFS file system, you may choose to add the file system name to the /etc/fstab file of every system in the SFS cluster. SFS entries in /etc/fstab are in the same format as NC1FS entries, with SFS specified as the file system type. The following is an example of an fstab file that includes both NC1FS and SFS entries.

```
/dev/dsk/root                           NC1FS   rw,CRI_RC="YES" 1       2
/dev/dsk/usr            /usr            NC1FS   rw,CRI_RC="YES" 1       2
/dev/dsk/src            /usr/src        NC1FS   rw,CRI_RC="YES" 1       2
/dev/dsk/sfs_usr1       /sfs/usr1       SFS     rw,CRI_RC="YES" 1       2
```

If you list an SFS file system in /etc/fstab, the file system will be checked and mounted when sfs_start is run, as described in Chapter 5, page 33. This is the recommended way of mounting shared file systems because sfs_start performs valuable cross-system integrity checks on your shared file systems before mounting them.

4.3 The /etc/config/sfs file

A UNICOS SFS configuration supports multiple SFS arbitration devices. Each Cray Research system in the SFS cluster that will be sharing file systems must include an /etc/config/sfs file that contains the names of each Cray Research system in the cluster and which SFS arbiters are associated with each system. The

`/etc/config/sfs` file also describes the identity of the SFS arbiters and defines the character special devices that support each arbiter. For a description of the format of this file, see the `sfs`(4) man page.

You must maintain identical `/etc/config/sfs` files on all systems in an SFS cluster.

SFS Startup with the `sfs_start` Command [5]

After defining the UNICOS SFS environment and creating the file systems that the Cray Research systems will share, you can run the `sfs_start` command to initialize the UNICOS SFS environment. The `sfs_start` command runs a series of tests to ensure that the system is properly defined, and initiates the `sfsd` and `codeblue` system daemons. The `sfs_start` command also runs file system checks on the SFS file systems in the `fstab` file and mounts the shared file systems.

5.1 `sfs_start` program logic

The `sfs_start` daemon performs the following operations:

1. Ensures that the Cray Research system can read the `/etc/config/sfs` file, which indicates which systems are part of the SFS cluster.
2. Ensures that the current host is listed in the `/etc/config/sfs` file.
3. Ensures that the `/etc/services` file includes an entry for `sfs_config`.
4. Ensures that the various SFS commands and daemons are executable.
5. Ensures that the `/dev/smp` and `/dev/sfs` nodes have been defined as character special devices.
6. Ensures that the `slr` and `smnt` slices have been defined as a block special or character special device.
7. Compares the local configuration (`/etc/config/sfs`) with those of all other hosts in the SFS cluster.
8. Compares the device nodes of the local host with those of all other hosts in the SFS cluster.
9. Executes the `sema` command to test semaphore 0.
10. Sets semaphore 5, the SFS start semaphore, to ensure that only one system at a time is running `sfs_start`.
11. Executes `sfsping` for each member of the SFS cluster identified in `/etc/config/sfs` to determine which systems in the cluster are up (live) and which are not up (dead).

12. Prints a list of live systems.

13. Checks whether `/etc/sfsd` is currently running. The `/etc/sfsd` command initializes the UNICOS SFS environment. For information about this command, see Section 5.2, page 35.

14. If the `/etc/sfsd` command is not running, checks whether this is the first system up in the SFS cluster. If this is the first system up, then `sfs_start` executes `sfsd` with the `-F` option to force semaphore initialization. If this is not the first system up, it executes `sfsd` without the `-F` option, which would clear any semaphores already in use.

15. Checks whether recovery is currently taking place; if so, waits until recovery is finished before proceeding.

16. If other systems in the SFS cluster are up, checks whether the `/dev/smp`, `/dev/sfs`, `/dev/dsk/slr`, and `/dev/smnt` devices are compatible. For each of these devices, the major and minor device numbers must be identical on every system.

17. Retrieves the list of `SFS` file systems from the `/etc/fstab` file.

18. If this is the only live system, clears the recovery semaphore (semaphore 2), if it is set, and clears the Ports Needing Recovery (PNR) mask by executing `/etc/sfspnr -c` (clear PNR mask), if it is not zero.

19. Performs the following steps for each `SFS` file system:

    - Compares the `SFS` file system node on this host with the `SFS` file system node on all systems that are up. If the nodes do not match, the file system will not be mounted and an error is reported.

    - Checks whether the `SFS` file system is currently mounted on other up systems in the SFS cluster.

    - If the `SFS` file system is currently mounted, executes `/etc/sfsck -rv` on the file system. This command checks the file system without requiring that the file system be unmounted.

    - If the `SFS` file system is not currently mounted, executes `/etc/fsck -uS` on the file system.

    - Mounts the `SFS` file system, creating the mount point for the file system if necessary.

20. Starts `/etc/codeblue -d /etc/sfsprefix`. This monitors the status of other systems in the SFS cluster.

21. Clears semaphore 5, the SFS start semaphore.

22. Prints that SFS initialization is complete.

5.2 The sfsd command

For every system in the UNICOS SFS environment, the sfs_start program executes the sfsd(8) command to initialize the SFS environment. The first time sfsd is run, it uses the -F option to force the initialization of the semaphores. Using this option after the first Cray Research system has been brought up, however, will clear any semaphores in use, which will likely cause systems currently using the UNICOS SFS feature to panic.

The sfsd command provides multiple services as a system daemon. It validates the site license, then initializes the SFS devices. It also listens for queries from other SFS client programs such as sfsping, sfsstat, and sfsddstat, described on Section 5.3, page 36.

After sfs_start executes sfsd, messages appear on the system console indicating the status of the UNICOS SFS environment. The following example shows a typical message display.

```
00:55:43(GMT) uts/c1/io/smp-ipi3.c: INFO        IPI-3 SMP device initialized (1024 semas,
  64 ports)
00:55:43(GMT) uts/c1/io/smp.c: INFO     smpopen: SMP device supports 1024 semas, 1024 will
   be used
00:55:43(GMT) uts/c1/io/smp.c: INFO     smpopen: smp[0] initialized:
        type=04 SMP-IPI3  phys=1024  start=0  len=1024  port=2
00:55:45 uts/c1/io/slr.c-06: INFO       slr_open: lock region area is </dev/dsk/slr>
00:55:45 uts/c1/io/slr.c-36: INFO       slr_open: SLR is not mirrored
slr_open: slr_size_size = 128
00:55:45 uts/c1/io/slr.c-68: WARNING    slr_init: SLR area ‘init’ semaphore not set
esd_set_slr_mirror_mode: DEBUG: called with newmask 07
00:55:45 uts/c1/io/slr.c-73: WARNING    slr_init: SLR area initialized
00:55:45 uts/c1/io/slr.c-24: INFO       slr_open: SLR open completed
00:55:45 uts/c1/io/slr.c-86: INFO       slr_init: port 2 setting Validity sema 15
00:55:45 uts/c1/io/esd_hb.c-25: INFO    ESD-port 2, name <ice>, num <2> logged on
SMP Heartbeat started.
```

If the system detects an error, a message indicating the nature of the error appears on the window in which sfsd was executed. For example, if you are not licensed to run the UNICOS SFS feature, the following message appears:

```
craylm: checkout of "sfs" failed.
lm_checkout: no such feature exists
This system is not licensed to run SFS.
        Please contact your system administrator, or
        Cray Research Software Technical Support
        to obtain a Shared File System license.
```

Once sfsd is running, sfs_start can check and mount SFS file systems.

5.3 Additional SFS commands

In addition to `sfsd`, there are three SFS commands the `sfs_start` program uses when a Cray Research system is being booted and brought up to multiuser mode. These commands are as follows:

`/etc/sfsping`	Used to ping another system, to determine if it is running UNICOS SFS software.
`/etc/sfsstat`	Used to obtain `stat` information about a device node on another Cray Research system in an SFS cluster.
`/etc/sfsddstat`	Used to obtain `ddstat` information about a disk configuration on another Cray Research system in an SFS cluster.

5.4 Sample `sfs_start` output

The following shows typical output from the execution of the `sfs_start` command.

```
# /etc/sfs_start
sfs_start - INFO: Beginning Shared File System startup
sfs_start - INFO: Locking SFS init sema 5
sfs_start - INFO: SFS init sema 5 Locked.
sfs_start - INFO: Checking to see if cluster member hosta is running SFS...
sfs_start - INFO: Checking IPI-3 SMP port numbers... hostb
sfs_start - INFO: Clearing SMP device during SFS initialization
sfs_start - INFO: SFS daemon (sfsd) started.
sfs_start - INFO: Mounting Shared File Systems...
sfs_start - INFO: SFS Cluster monitor started
sfs_start - INFO: Clearing SFS init sema 5
sfs_start - INFO: SFS initialization complete.
```

System Lock Recovery [6]

If a Cray Research system in an SFS cluster crashes while it holds data locks on a shared file system, other systems in the cluster may be locked out of that file system. For this reason, failures of individual Cray Research systems in an SFS cluster need to be detected and recovered. The UNICOS SFS feature implements *heartbeat* mechanisms that allow properly functioning machines in an SFS cluster to detect failures of other machines in the cluster and to initiate recovery procedures to clean up any locks left behind by other systems in the cluster that have ceased to function.

The semaphore device provides support for system heartbeat monitoring. Each time the semaphore device services a semaphore operation for a client port, it records that fact as a bit in a row in a heartbeat matrix. Part of the response status for each semaphore operation is the client ports column out of the heartbeat matrix. In this manner, each client port can find the identity of the other client ports that have recently accessed the semaphore device. A special kernel thread monitors the heartbeat status of other client systems in an SFS Cluster, and insures that the system it represents keeps a current heartbeat within the semaphore device. If another client system fails to respond within several heartbeat intervals, that system is declared non-responsive.

The SFS heartbeat procedures and system lock recovery mechanisms are initiated automatically when the `sfs_start` command is executed.

6.1 System recovery commands

The commands that check the heartbeat status of the systems in an SFS cluster and initiate recovery when necessary are called automatically, in the following sequence:

- `sfs_start` initiates `codeblue`.
- `codeblue` executes `sfsprefix` (when necessary).
- `sfsprefix` executes `sfsrecover`.
- `sfsrecover` executes `sfsck`, `sfsclrlck`.

The actions of the system recovery commands are summarized as follows:

`codeblue`	`codeblue` registers with the kernel to receive heartbeat state information when it changes. `codeblue` is also responsible for creating and maintaining a log of system lock

	recovery related events. Upon detecting a system that transitions to a `DOWN` state, `codeblue` initiates a specified recovery action. The default action is to execute `/etc/sfsprefix`.
`/etc/sfsprefix`	`sfsprefix` first attempts to lock the SFS cluster global recovery semaphore. It is likely that all still-running systems will simultaneously notice when one system fails. The first system that successfully locks the recovery semaphore proceeds with the recovery; other systems wait their turn and typically find no recovery work to do. If the system that was conducting a recovery also fails, then the remaining systems have an opportunity to complete the recovery action for all failed systems. `sfsprefix` is a process that initiates recovery actions, and monitors the exit status of those actions. If it is found that a recovery action failed, `sfsprefix` can reinitiate the recovery. One reason that a recovery action would fail would be detection of the failure of yet another system. The default action is to execute `/etc/sfsrecover`.
`/etc/sfsrecover` *port* `#`	`sfsrecover` uses the port number as defined in `/dev/smp` on the machine that failed to indicate which system to recover. `sfsrecover` is an administrative shell script. The purpose of this script is to determine the list of shared file systems that were mounted on the failed system, and to execute the `/etc/sfsck` and `/etc/sfsclrlck` processes against those shared file systems.
`/etc/sfsck`	`sfsck` is the main lock recovery process. All the inodes in the subject shared file system are scanned, using the appropriate SFS semaphore

	locking protocols. File locks held by the failed system are released, and the inode updated.
`/etc/sfsclrlck`	`sfsclrlck` scans the semaphore device, clearing any semaphores held by the failed system.

System lock recovery actions perform their function while the shared file systems they are recovering are still mounted and used by the remaining Cray Research systems in the SFS cluster, allowing the work being done by other machines in the cluster to continue unaffected.

6.1.1 `codeblue` *program logic*

The following is a summary of the program logic of the `codeblue` program.

The `codeblue` command performs the following processes.

1. Parses the `codeblue` options:

`-d` *program*	Indicates *program* to run when a machine in the SFS cluster goes down. The default program is `/etc/sfsprefix`.
`-n` *script*	Indicates *script* to run when a machine in the SFS cluster becomes active for the first time. The default value of *script* is `NULL`.
`-u` *script*	Indicates *script* to run when a machine in the SFS cluster comes up again after a halt. The default value of *script* is `NULL`.
`-o` *pathname*	Indicates the path name of the file into which the output of the programs run by `codeblue` should be directed. The default is `/tmp/recovery.log`.
`-w` *sec*	The number of seconds to delay before checking the heartbeat table for state changes. The default is 5 seconds.

2. Ensures that this is the only copy of `codeblue` running.
3. Ensures that system is running the SFS feature.
4. Makes the process a daemon and opens a log file.
5. Opens the `/dev/sfs` file.
6. Checks the heartbeat table.

7. Reads the heartbeat table.
8. Executes a system call that awaits state changes in the SFS cluster. The process goes to sleep waiting for a change in the heartbeat table. When that occurs, it executes a system call to check on the nature of the state change. If the state change is from `ACTIVE` to `DEAD`, a mask is prepared to indicate that the port is down. If there are ports needing recovery, `/etc/sfsprefix` is spawned (unless a different script has been indicated with the `-d` option).

6.1.2 `sfsrecover` *program logic*

The following is a summary of the program logic of the `sfsrecover` program. The `sfsprefix` script executes the `sfsrecover` command, which performs the following operations.

1. Determines which ports are down and builds a port mask of those systems.
2. Prints the port mask of down systems.
3. Executes `/etc/sfsclrlck` with the `-p` (list of ports) option. This clears any semaphores held by the failed system.
4. Executes `mount -s` to get a list of mounted shared file systems.
5. For all mounted file systems, performs the following actions:
   - Validates file system parameters.
   - Checks if the file system is mounted on any of the systems that are to be recovered.
   - Executes `sfsck` with `-r`, `-P`, and `-p` options. The `-r` option indicates repair mode. The `-P` option monitors the Ports Needing Recovery (PNR) mask. The `-p` option indicates the ports with locks to be released.
   - Checks to see if `sfsck` exited with a return code of 99, which indicates that the PNR mask changed. If so, exits the program so that `codeblue` can be rescheduled.
   - If `sfsck` executed successfully, adds the file system to the list of file systems to be unmounted.
6. For all file systems to be unmounted, executes `umount` with `-p` (list of ports) option.

6.1.3 Sample recovery log

The following shows a sample log from an SFS recovery session.

```
Tue Oct 11 17:12:27 1994 [hosta (port 1)] /etc/codeblue: Awakened.
Tue Oct 11 17:12:27 1994 [hosta (port 1)] /etc/codeblue:   Port :  0 1 2 3 4 5 6 0
Tue Oct 11 17:12:27 1994 [hosta (port 1)] /etc/codeblue:   State:  . ^ ^ . . . . .
Tue Oct 11 17:12:27 1994 [hosta (port 1)] /etc/codeblue:   Resp :  0 0 0 0 0 0 0 0
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue: PNR after re-checking HB table: 0
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue: port 2 state transition
  (NEW -> ACTIVE)
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue: Awakened.
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue:   Port :  0 1 2 3 4 5 6 0
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue:   State:  . ^ ^ . . . . .
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue:   Resp :  0 0 0 0 0 0 0 0
Tue Oct 11 17:12:28 1994 [hosta (port 1)] /etc/codeblue: PNR after re-checking HB table: 0
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue: Awakened.
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue:   Port :  0 1 2 3 4 5 6 0
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue:   State:  . ^ v . . . . .
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue:   Resp :  0 0 14 0 0 0 0 0
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue: port 2 state transition
   (ACTIVE -> DEAD)
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue: PNR after re-checking HB table: 100
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue: down_path: /etc/sfsprefix
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue: Starting recovery for port(s): 2
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/codeblue: spawning '/etc/sfsprefix 2'
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: TSET'ing recovery sema 2
  (timeout = 6s)
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: Recovery sema 2 SET.
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: Recovery timeout for port(s) 2 = 0s.
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: Reading PNR information from ESD area
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: read OK - PNR mask = 100
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: Recovering for port(s) 2
  (port_mask: 100   PNR mask: 100)
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: Starting '/etc/sfsrecover' for
  port(s) 2.
Tue Oct 11 20:22:05 1994 [hosta (port 1)] /etc/sfsprefix: '/etc/sfsrecover' for port(s) 2
   started (pid 1160).
Tue Oct 11 20:22:07 1994 [hosta (port 1)] /etc/sfsrecover: INFO - recovery mask pattern =
  ports=??m?????

Tue Oct 11 20:22:07 1994 [hosta (port 1)] /etc/sfsrecover: INFO - calling /etc/sfsclrlck -p 2
Tue Oct 11 20:22:07 1994 [hosta (port 1)] /etc/sfsclrlck:  --- clearing semas for downed
  systems (ports 2) ---
Tue Oct 11 20:22:07 1994 [hosta (port 1)] /etc/sfsclrlck: FSREC Sema state: 1    (last changed
  by port 1)
```

```
Tue Oct 11 20:22:09 1994 [hosta (port 1)] /etc/codeblue: Recovery watchdog sez, 'woof! Checking
  PNR status...'
Tue Oct 11 20:22:09 1994 [hosta (port 1)] /etc/sfsclrlck: cleaned up 0 clear-on-exit semas
  for port 2
Tue Oct 11 20:22:09 1994 [hosta (port 1)] /etc/sfsclrlck: FSREC Sema state: 1    (last changed
  by port 1)
Tue Oct 11 20:22:10 1994 [hosta (port 1)] /etc/codeblue: Checking PNR status...
Tue Oct 11 20:22:10 1994 [hosta (port 1)] /etc/codeblue: PNR status:  orig: 100  cur: 100


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%
        %%         RECOVERING /dev/dsk/sfs_scsi for ports 2
        %%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Tue Oct 11 20:22:10 1994 [hosta (port 1)] /etc/sfsrecover: INFO - RECOVERING /dev/dsk/sfs_scsi
  for ports 2
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/sfsrecover: INFO - /dev/dsk/sfs_scsi isn't
  mounted on ports 2
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/sfsprefix: /etc/sfsrecover exited with
   status 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/sfsprefix: Recovery for port(s) 2 completed
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/sfsprefix: PNR bits cleared for port(s) 2. PNR
  mask now: 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/sfsprefix: Clearing sema 2 (timeout = 6s).
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/sfsprefix: sema 2 CLEARED.
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: Checking PNR status...
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: PNR status:  orig: 100  cur: 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: /etc/sfsprefix exited with status 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: recovery succeeded for port(s) 2
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: Awakened.
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue:   Port :  0 1 2 3 4 5 6 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue:   State:  . ^ v . . . . .
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue:   Resp :  0 0 14 0 0 0 0 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: port 2 state transition (ACTIVE -> DEAD)
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: PNR after re-checking HB table: 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: Awakened.
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue:   Port :  0 1 2 3 4 5 6 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue:   State:  . ^ v . . . . .
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue:   Resp :  0 0 14 0 0 0 0 0
Tue Oct 11 20:22:11 1994 [hosta (port 1)] /etc/codeblue: PNR after re-checking HB table: 0
```

6.2 The `sfsmddcp` command

You may sometimes need to rebuild a leg of a shared file system that includes mirrored legs. When rebuilding the mirrored legs of a shared file system, the operating system must be aware of SFS locking protocols and must be able to rebuild while the shared file system is still mounted and being used by other systems. The `sfsmddcp` command is a special version of the `mddcp`(8) utility that you can use for this.

Mounting Additional SFS File Systems [7]

After you have run the `sfs_start` command and your SFS cluster is up and running, you can still create and mount additional SFS file systems. In order to do this, you must manually check and mount the file system instead of relying on the `sfs_start` command to do this automatically.

7.1 Checking an SFS file system

After creating a file system with the `mkfs`(8) command, use the standard `fsck`(8) command to verify the file system structure. The `fsck` command needs to be executed on only one system in the SFS cluster before the file system is mounted anywhere in the cluster.

Caution: Once a file system is mounted on any system in the SFS cluster, you must not execute `fsck` on the file system. Mounted file systems may be checked with the `sfsck` command.

Following are examples of the output from `fsck` on a shared file system.

```
# /etc/fsck /dev/dsk/sfs_test0
/sfs_test0: file system opened
/sfs_test0: super block fname sfs_test, fpack sfsp_tes, (SFS enabled)
/sfs_test0: Phase 1 - Check Blocks and Sizes
/sfs_test0: Phase 2 - Visit Directories
/sfs_test0: Phase 3 - Checking Directories
/sfs_test0: Phase 4 - Checking Non-Directories and Link Counts
/sfs_test0:
/sfs_test0: i-node - 22080, name - ./hostb/lioAAAa00815/lio_815
/sfs_test0:   Clearing SFS (I) locks <0100000000000000000000, 040, 0, 040, 0>
/sfs_test0:         - 19 SFS locks cleared
/sfs_test0: Phase 5 - Verify Dynamic Information
/sfs_test0: dynamic block flags indicate mounted or not checked
/sfs_test0:            reflagged
/sfs_test0: Phase 6S - Rebuilding SFS Block Information
/sfs_test0: file system summary
/sfs_test0:     75776 total i-nodes (75448 free i-nodes)
/sfs_test0:     303104 total blocks  (296412 free blocks)
/sfs_test0:  ***** FILE SYSTEM WAS MODIFIED *****

# /etc/fsck /dev/dsk/sfs_test0
/sfs_test0: file system opened
```

```
/sfs_test0: super block fname sfs_test, fpack sfsp_tes, (SFS enabled)
/sfs_test0: clean exit for clean file system

# /etc/fsck /dev/dsk/sfs_test1
/sfs_test1: file system opened
/sfs_test1: super block fname sfs_test, fpack sfsp_tes, (SFS enabled)
/sfs_test1: Phase 1 - Check Blocks and Sizes
/sfs_test1: Phase 2 - Visit Directories
/sfs_test1: Phase 3 - Checking Directories
/sfs_test1: Phase 4 - Checking Non-Directories and Link Counts
/sfs_test1: i-node - 2816, name - ./dct/data/rtstest.gdb
/sfs_test1:   Clearing SFS (R) locks <0200000000000000000000, 0, 042, 042, 02>
/sfs_test1:         - 2 SFS locks cleared
/sfs_test1: Phase 5 - Verify Dynamic Information
/sfs_test1: dynamic block flags indicate mounted or not checked
/sfs_test1:             reflagged
/sfs_test1: Phase 6S - Rebuilding SFS Block Information
/sfs_test1: file system summary
/sfs_test1:     75776 total i-nodes (75366 free i-nodes)
/sfs_test1:     303104 total blocks  (271452 free blocks)
/sfs_test1:  ***** FILE SYSTEM WAS MODIFIED *****

# /etc/fsck /dev/dsk/sfs_test1
/sfs_test1: file system opened
/sfs_test1: super block fname sfs_test, fpack sfsp_tes, (SFS enabled)
/sfs_test1: clean exit for clean file system
```

7.2 Mounting SFS file systems

After successful execution of the `sfs_start` command, you can mount shared file systems. Attempting to mount a shared file system if you have not initialized the UNICOS SFS environment yields an error.

When a shared file system is mounted, the UNICOS kernel recognizes that the file system was made as a shared file system and an assignment of the specified number of semaphores is attempted. If the assignment was successful, then the assignment is recorded in the Shared Lock Region for all other systems to see.

You may mount a shared file system on every Cray Research system in the SFS cluster. To view the Shared Mount Table, which shows all current mounted SFS file systems, use the `mount`(8) command with the `-s` option.

Caution: After a file system is mounted, you should not execute `mkfs` on that file system again until it is unmounted from all Cray Research systems in the SFS cluster.

When you execute the `mount` command with the `-s` option, the display will indicate whether the file system is currently mounted on some other Cray Research system. The following example shows a display that results from executing `mount -s`.

```
# /etc/mount /dev/dsk/sfs_tst1 /sfs/tst1
# mount -s
/sfs/tst0 on /dev/dsk/sfs_tst0 SFS ports=--m-----,dev=34/205, semas=29-156,
  validity=0:2:303104:16:24576:10240,mode=rw
/sfs/tst1 on /dev/dsk/sfs_tst1 SFS ports=--m-----,dev=34/206,semas=29-156,
  validity=0:2:303104:16:34816:10240,mode=rw
```

The `ports=` display indicates which ports the file systems are currently mounted on, with one dash serving as a place holder for each port. The leftmost port is port 0. In this example, `ports=--m-----` indicates that the file system is mounted on port 2, because the place holders for the first two ports, port 0 and port 1, are indicated by dashes.

The `dev=` display indicates the major and minor number for the file system.

The `semas=` display indicates which semaphores have been assigned to the file system.

The `validity=` display is a "magic cookie" indicator to ensure that all the systems have compatible configurations.

The maximum number of Cray Research systems that can share a file system is 64. This is also the maximum number of systems that you can have in one SFS cluster.

Performance Issues [8]

With the exception that named pipes and mount points are not supported, an application program need not be aware that it is doing input or output to files in a shared file system.

The same system calls that are used for I/O in standard NC1FS file systems are used for files in a shared file system. Files are created with the open(2) system call, updated using read(2) and write(2), reada(2) and writea(2), or listio(2), and closed or unlinked with close(2) and unlink(2) when they are no longer needed. The fact that a file is part of an SFS configuration can be completely transparent at the user level.

It is possible, however, to take advantage of extensions to several system calls to control allocation of and access to files in an SFS configuration more closely, as well as to enhance input and output performance.

If an application is written that uses features that are exclusive to shared file systems, it should use the sysconf library routine to determine whether or not the UNICOS SFS feature is currently available on the host system. If it is, the sysconf call returns the SMP port number to which the host is connected. If not, it returns a negative number. The following code fragment demonstrates the use of sysconf.

```
if ( sysconf ( _SC_CRAY_SFS ) < 0 ) {
        printf ( "SFS not active on this machine\n" );
        exit;
} else {     /* continue processing */
        .
        .
        .
}
```

In the sections that follow, extensions to system calls that are exclusive to SFS file systems are specified as such. In addition, some features that are available to both regular NC1FS and SFS file systems, but which may profitably be used in conjunction with exclusive features, are described.

8.1 Data locking and SFS systems

A fundamental requirement of shared file systems is the preservation of data integrity. Considerable effort, and therefore system time, is expended to insure that a shared file is protected from inadvertent simultaneous events. For example, if two processes running on different Cray Research systems were simultaneously extending the same file, it is important that all parties involved maintain a common view of where the end of the file is located.

In the UNICOS SFS environment, any time a user process does anything to a file that changes any of the attributes of that file, such as size, modification date, ownership, and so on, the UNICOS SFS feature must update the on-media inode, under protection of an inode semaphore.

The notable exception to the above statement happens when the UNICOS SFS feature determines that the file update is occurring under control of one of the three supported data locks:

- Exclusive open lock
- Read lock
- Write lock

If a shared file is currently *open-locked*, *read-locked*, or *write-locked*, the UNICOS SFS feature defers inode updates, and therefore inode semaphore operations, until the lock is relinquished. In this *data-locked environment*, the UNICOS SFS feature is able to deliver device speed I/O performance.

8.1.1 Exclusive open lock

A UNICOS SFS exclusive open lock is granted if the user process used the `O_SFSXOP` flag on the `open`(2) request. Upon return from the `open`(2), the user process is guaranteed that no other process in the SFS cluster has this file open. While that process owns the open lock, the process may execute an `unlink`(2) call on the file, thus causing all other pending `open`(2) calls for this file to fail with an `ENOENT` error. The UNICOS SFS exclusive open lock can only be obtained through the `open`(2) call.

If a process tries to open a file without the `O_SFSXOP` flag, when the file is already open by another process with an exclusive open lock, the resulting behavior is determined by the presence or absence of either the `O_NDELAY` or the `O_NONBLOCK` flags. If either is set on the attempted open, the open will fail with an `EAGAIN` error. If neither flag is set, the process will sleep until the file has been closed by all other processes on all Cray Research systems.

If the same process opens a file with the exclusive open flag, and then attempts a subsequent non-exclusive open, the second open attempt will fail with an `EDEADLK` error.

Use of the `O_SFSXOP` flag makes opening and locking a file an atomic operation, which closes the timing window between what would otherwise require an `open` call followed by an `fcntl` system call to lock the file.

8.1.2 Read lock

A UNICOS SFS read lock is granted if no other process in the SFS cluster has a current open lock, or write lock. A read lock is requested by using the normal flock(2) or fcntl(2) UNICOS kernel interfaces. As many as 500 processes on each system in the SFS cluster may concurrently own a read lock on a shared file.

8.1.3 Write lock

A UNICOS SFS write lock is granted if no other process in the SFS cluster has a current open lock, read lock, or write lock. A write lock is requested using the normal flock(2) or fcntl(2) UNICOS kernel interface. Only one process in the entire SFS cluster may own a write lock on a shared file.

8.1.4 Setting write locks and read locks

An write lock or a read lock may be assigned to or released from an open file by calling the fcntl(2) UNICOS kernel interface with a pointer to an appropriately populated struct flock structure (shown below), which is defined in <sys/fcntl.h>. Note that because file data locking in a shared file system applies to the entire file, fields l_whence, l_start, and l_len are forced to zero by the data locking functions of the fcntl(2) system call.

```
struct flock {
        short l_type;
        short l_whence;
        off_t l_start;
        off_t l_len;
        short l_sysid;
        pid_t l_pid;
};
```

Both a blocking and a non-blocking version of file locking commands in fcntl exist. If F_SETLK is used and the lock cannot be set, the fcntl call returns immediately with a -1 value. If F_SETLKW is used and the read or write lock cannot be set, the process will sleep until it can be set.

8.1.5 Adding a write lock

If a process needs to update a file and wants to disallow both reading and writing by all other processes, it may set a write lock by using the fcntl(2) system call. While one process has a write lock set on an open file, no other process may set it, nor may a read lock be set, and, of course, no exclusive open operations of the file are allowed.

If a call is made to `fcntl` to set a write lock on a file that was opened with the `O_SFSXOP` flag, the exclusive open lock will be retained for the file. That is, it will have both an exclusive open and a write lock.

The following example demonstrates how to add a write lock.

```
#include <sys/fcntl.h>
struct flock flock, *fp = &flock;

main {
     int fd, rc;

     if ( ( fd = open ( "shared_file", O_CREAT | O_RDWR ) ) == -1 ) {
            exit ( 1 );
     }

     /*
      * Set the write lock. Note that another process
      * could obtain a write lock in this window
      */
      fp->l_type = F_WRLCK;
      if( ( rc = fcntl ( fd, F_SETLKW, fp ) ) == -1 ) {
             exit ( 1 );
      }
}
```

8.1.6 Adding a read lock

A file may have a read lock set. This lock is obtained by means of the `fcntl`(2) system call. The read lock is unique in that more than one process may hold the lock at the same time. Currently, 500 processes may hold read locks on the same file. If a file has one or more read locks, it will block write operations, exclusive open operations and the setting of a write lock until all processes holding the read lock release it.

If an `fcntl` call is made to set a read lock on a file that was opened with the `O_SFSXOP` flag, the exclusive open lock for the file will be released for the file. That is, a file cannot have both an exclusive open and a read lock set.

The following example shows how to set a read lock for a file.

```
#include <sys/fcntl.h>
struct flock flock, *fp = &flock;
```

```
main {
     int fd, rc;

     if ( ( fd = open ( "shared_file", O_CREAT | O_READ ) ) == -1 ) {
            exit ( 1 );
     }

     /*
      * Set the read lock. Note that another process
      * could obtain an write lock in this window
      */
     fp->l_type = F_RDLCK;
     if( ( rc = fcntl ( fd, F_SETLKW, fp ) ) == -1 ) {
            exit ( 1 );
     }
}
```

8.1.7 Removing data locks

A process may explicitly relinquish a data lock by using the `flock`(2) or `fcntl`(2) UNICOS kernel interface, or implicitly by simply closing the file. For open and write locks, where only one process in the SFS cluster can own the lock, the lock is relinquished immediately. Because read locks can be held by many processes, a reference counter mechanism is used. In this case, the lock is relinquished when all processes that own a read lock have relinquished their read locks.

When the `l_type` field of the `flock` structure is set to `F_UNLCK`, a lock currently associated with the file's data are removed, including the exclusive open lock. If this process is the last (or only) process holding a read lock on the file, other processes will be allowed to acquire a write lock. Otherwise, the file is still considered to be locked against such attempts due to other holders of the read lock. The following code example shows how to clear a file data lock.

```
#include <sys/fcntl.h>
struct flock flock, *fp = &flock;

main {
     int fd, rc;

     if ( ( fd = open ( "shared_file", O_CREAT | O_READ ) ) == -1 ) {
            exit ( 1 );
     }
```

```
                /*
                 * Set the read lock.
                 */
                fp->l_type = F_RDLCK;
                if( ( rc = fcntl ( fd, F_SETLKW, fp ) ) == -1 ) {
                       exit ( 1 );
                }

                /*
                 * Clear all file data locks held by this process
                 */
                fp->l_type = F_UNLCK;
                if ( ( rc = fcntl ( fd, F_SETLK, fp ) ) == -1 ) {
                        exit ( 1 );
                }
        }
```

8.2 File allocation issues

NC1FS file systems, as well as SFS file systems, may be configured to have a *primary allocation area* and a *secondary allocation area*. In general, each of these two allocation areas will have distinctive characteristics. For example, the primary area might be mirrored to provide redundancy for file system structures such as inodes, while the secondary area could be striped to provide high bandwidth I/O. In cases such as this, it becomes desirable to be able to direct certain types or sizes of files to one or the other allocation area.

Note: Mirroring is not supported on CRAY T3D or CRAY T3E systems.

In the absence of explicit direction from the process that opened the file, where a file's data is initially allocated is a function of the allocation unit sizes of the areas and the quantity of data written to the file at a time. This assumes that neither allocation area is full at the time the file is opened.

Another aspect of file allocation that might be important is whether or not the file's data should be allowed to reside partially in both allocation areas. Again, in the absence of explicit direction from the user, a file that is initially allocated in the primary allocation area will have subsequent allocations in the secondary area if the file grows beyond a system defined size (BIGFILE, <sys/param.h>), or if the primary allocation area becomes full.

Yet another consideration is whether a file should be allowed to increase in size once it has reached a user-specified size. For example, if a process uses the ialloc routine to preallocate space for a file, the user may wish that space to be the maximum size that the file can become. By default, simply preallocating a file does not keep it from growing beyond the preallocated size.

All of these issues (initial placement of data, residence in both allocation areas, and limiting file size) can be controlled by options on various system calls, which are discussed in the following sections.

8.2.1 The `open`(2) system call

The `O_BIG` flag may be used at file open time to control allocation of a file in file system configurations with both a primary and secondary allocation area.

By including the `O_BIG` flag in the `open` system call, an application program forces initial allocation of the file's data to the secondary allocation area.

8.2.2 The `ialloc`(2) system call

The `ialloc` system call allows a user to preallocate data space for a previously opened file.

The form of the `ialloc` system call is:

```
long ialloc ( int fildes, long nb, int flag, int part, long *avl );
```

Note: Although the user requests space allocation in *bytes*, the kernel actually allocates in multiples of *allocation units*, so it is possible that the total amount of space available for a file is somewhat larger than what the user requested. This is discussed in the `fcntl` section.

Several flags may be used to precisely control the characteristics of the preallocated file space.

- The `IA_CONT` flag specifies that the space is to be allocated contiguously in the file system. If the amount of space requested is unavailable, no space is allocated, and *ialloc* returns a -1, unless the `IA_BEST` flag also is set.
- If the `IA_BEST` flag is set, then either the amount of space requested or the amount of space available is allocated, whichever is smaller, and the amount allocated is returned in the system call return value.
- If the `IA_RAVL` flag is set, then either the requested amount is allocated, or nothing is allocated and an error is returned, and the amount of space available is returned in the *avl* parameter.

- The IA_PART flag restricts allocation to the current allocation area. If the O_BIG flag was set when the file was opened, the initial allocation area is the secondary allocation area.

All of the flags can be used in combination. For example, to allocate the largest contiguous chunk of space in the allocation area, the IA_CONT, IA_BEST, and IA_PART flags should all be set in the ialloc call.

8.2.3 The fcntl(2) system call

It may be desirable to restrict a file to a maximum size. This can be accomplished by setting the nogrow flag for a file.

The nogrow flag can be set for a file with a call similar to the following:

```
prev_flags = fcntl ( fd, F_SETALF0, S_ALF_NOGROW ) ;
```

The return value is the state of the flags prior to the call.

This flag is intended for situations where a file is preallocated with ialloc, and its size is not allowed to either increase or decrease. Note that calls to ialloc supply the number of *bytes* to be allocated for the file, but file space is actually allocated in multiples of *allocation units*. Thus, the actual size of the file might be somewhat larger than was specified in the ialloc call. This notwithstanding, if a file has the nogrow flag set, additional space will not be allocated for it beyond what it has at the time the fcntl call is made.

Even if a file has the nogrow flag set, the trunc system call may still be used to decrease the logical size of the file, but the physical space allocated to it is not released.

If, at some point during processing, it becomes desirable to allow file size changes to a file that has the nogrow flag set, it may be cleared with a call similar to the following:

```
prev_flags = fcntl ( fd, F_CLRALF, S_ALF_NOGROW ) ;
```

8.2.4 The join(2) and fjoin(2) system calls

The join and fjoin system calls are supported for both NC1FS and SFS file systems. There forms are:

```
int fjoin ( int fildes1, int fildes2 );
```

These calls are used to concatenate two files without moving file data. Instead, the extent descriptors in *file2*'s inode are appended to *file1*'s extent descriptors. If any allocated but unused blocks exist at the end of *file1*, they are deallocated before *file2*'s extents are appended. At the completion of the system call, *file2* has been truncated to zero blocks.

> **Note:** With the `join` system call, the file being joined to (first argument) must end on an allocation unit boundary. For example, if the allocation unit was one megabyte, then the first file would have to be an integral number of megabytes in size.

8.2.5 The open*(2) system call*

In addition to the exclusive open data lock described above, the `open`(2) system call supports the `O_SFS_DEFER_TM` flag, which may be useful when writing applications that use shared files.

The `O_SFS_DEFER_TM` flag may be useful in reducing file system overhead by declaring to the UNICOS kernel that updates to file inode time stamps may be done less frequently than they otherwise would. If it is not critical to the application that the time stamps returned by the `stat`(2) or `fstat`(2) system calls be highly accurate, then setting this flag on very active files is advisable. The number of inode updates to media will be reduced, which implies a reduction in overhead and a corresponding increase in performance.

The `O_SFS_DEFER_TM` flag is only valid for files using the UNICOS SFS feature.

SFS Restrictions and Limitations [9]

There are some UNICOS features that require support from the file system level code, but are not fully supported in the current SFS implementation.

Disk quotas	The implementation of disk quotas is dependent upon the utilization of in-core data structures for the enforcement of appropriate quota levels. The current implementation of SFS does not support disk quotas.	
Mount points	A directory in a shared file system cannot be used as a mount point.	
Block and character special devices	The current UNICOS SFS implementation does not support the utilization of block and character special device nodes. Since the principal location of nodes of this type is within the `/dev` directory structure, the main impact of this limitation is to tape devices. The character special node that would be created as the result of a `tpmnt` command cannot reside in a shared file system.	
Named pipes	The current SFS implementation does not support the utilization of a named pipe that has been created within SFS name space.	
Security	The UNICOS Multilevel Security (MLS) feature is not currently supported with UNICOS SFS.	
Data Migration	Client/Server operations for Data Migration Facility (DMF) for shared file systems are not yet available.	
MPP Restrictions	The following restrictions apply to MPP systems.	
	Mirroring	Mirrored disk partitions are not currently supported on CRAY T3D and CRAY T3E systems.
	Input/Output Node (ION) configuration for FCNs and MPNs	The FCN and MPN Input/Output Nodes provide the semaphore operations for the disks they support. Some restrictions apply to their use. No other semaphore device can be used with disks connected through these IONs. Further, only a single FCN or MPN may be used to access the disks when

	SFS is being used (a second ION would not have access to the semaphore information).
Input/Output Node configuration for HPNs	The HPN Input/Output Node does not contain the semaphore code. It passes semaphore information on to a semaphore device on the HIPPI network. As such, it does not have the same limitations as the FCNs and MPNs. The HPN does not support both IPI-3 and TCP/IP packets simultaneously. The HPN disk driver will not supported until UNICOS/mk 1.5.1.

Symbolic Semaphore Representation

The SFS symbolic semaphore implementation has a design limitation.

Devices that use symbolic semaphores (ND-40 disks, H-SMP semaphore devices, GigaRing based Fiber Channel Nodes (FCNs) and GigaRing-based Multiple-Purpose I/O Nodes (MPNs)) select semaphores to use for a given file based on the file's device (minor) number and inode value placed into a 32-bit field. The upper 8-bits store the lower portion of the device number while the lower 24-bits store the lower portion of the inode value (removing the partition component).

The potential hazard is that when multiple devices are using the same semaphore server, it is possible for the value derived from the truncated device and inode values to match that of a file on another device. This makes it feasible for a single semaphore to be used for access to two different files and can cause file corruption and system panics.

The problem can be avoided by doing two things:

- Limit the minor device values for the file systems to values less then 256. This will ensure that the top 8-bits of the semaphore selection field are unique for a given device.

- Do not mount SFS file systems with multiple partitions

Questions should be directed to your Cray Research Service Representative.

Topics in SFS Internals [10]

The data locking capabilities of the `NC1FS` file system are not always sufficient to allow the access control that is necessary when multiple processes on different Cray Research systems require the same degree of controlled, shared access to a file. For this reason, `SFS` file systems support the following extensions to the `NC1FS` file system which make the sharing of file systems between cooperating Cray Research systems possible:

- File system meta-data cache coherency
- Mandatory locking
- Lock ownership by system
- Media (sector) update protection

These features are described in the following sections.

10.1 File system meta-data cache coherency

Traditional (non-shared) high-performance file systems, including the `NC1FS` file systems, derive some degree of their high performance by keeping often-referenced data structures in cache memory. By avoiding the overhead of reading and writing often-used file system data structures from and to the file system media, significant performance gains can be realized.

The problem that this technique poses for `SFS` file systems is that only one Cray Research system in an SFS cluster can read the cached data: the Cray Research system whose main memory is acting as the cache. If the cached data is ever modified, as it will be if the file is deleted or changes size, the rest of the systems in the cluster can only read old information (the file system information that resides on-media), which is out-of-date with respect to the cached information visible only to the Cray Research system that owns the cache.

The UNICOS SFS feature solves this problem using the straightforward approach of keeping file system data up-to-date on the file system media all the time. No cacheing of file system information by any of the Cray Research systems sharing a file system is allowed, with one notable exception: if a file is locked, the UNICOS system that owns the file lock may cache file system information related to the locked file after the file lock has been recorded on the file system media so that the other Cray Research systems sharing that file system can determine that the file is locked.

10.2 Mandatory locking

In the NC1FS file system, data locking is considered discretionary; that is, it only works if the programs that will be using a shared file cooperate and agree on how they will utilize and react to data locks. With the UNICOS SFS feature, data locking is automatically mandatory in nature; that is, the synchronization of contending programs is handled automatically by the UNICOS operating system.

10.3 Lock ownership by system

A lock on a file is said to be owned by a particular mainframe in an SFS cluster. The concept of file locks being owned by computer systems, and not simply by processes running on those systems, is the second key extension to the NC1FS file system that makes the sharing of file systems possible.

When a file is locked on an NC1FS file system, or on another non-shared file system that supports file locking, some information about which process on the system locked the file is recorded as part of the lock. This information is used for purposes such as making sure that only the process that locked a file may unlock it.

On NC1FS file systems, no information about which computer system the process that locked the file is running on is recorded; this information is not needed because it is assumed that only one computer system can be accessing the file system. In a UNICOS SFS environment, this single-system assumption is no longer valid, so the concept of lock ownership by systems as well as processes has been introduced.

From a shared file system point of view, "which system has this file locked?" is at least as important a question as "which process has this file locked?" Changes have been made to the NC1FS file system inode data structure to be able to record lock ownership by UNICOS systems as well as by processes to support this concept.

10.4 Media (sector) update protection

NC1FS file system inodes are stored on file system media with many inodes packed into a single media sector, where a sector is the smallest unit of data that may be transferred to or from the media. Typically, on a network disk, several hundred inodes are stored in a physical sector of the media holding the file system.

Separate and distinct inodes, whose only relationship may be that they happen to reside in the same physical media sector, may be independently accessed and updated in a non-shared environment because the media sectors holding these inodes are only being accessed by a single system.

In a shared environment, updates to inodes must involve an extra locking operation, which protects the sector containing the inode being updated from simultaneous updates from other systems.

Index