File System Planning [2]

All of the files that are accessible from within the UNICOS operating system are organized into *file systems*. File systems store data in formats that the operating system can read and write.

Warning: This chapter contains warnings and information critical to the use of a Cray ML-Safe configuration.

This chapter discusses several aspects of administering file systems. This chapter is organized as follows:

- Introduction to UNICOS file systems
- File system concepts
- Using the `mkspice`(8) command (IOS-E and IPN-1 systems only)
- Creating file system nodes
- Configuring disk arrays
- File system initialization
- Inode allocation strategies
- Inode region allocation
- Labeling a file system
- Mirrored file systems
- Performance considerations

2.1 Introduction to UNICOS file systems

The following sections introduce some general characteristics of UNICOS file systems. The following topics are discussed:

- File system overview
- File system types
- File system strategies

2.1.1 File system overview

The file is the logical unit of data storage within the UNICOS operating system. Files are grouped into structures called *file systems*. The root file system contains the base or root of the file system tree. Other file systems are logically attached to (*mounted*) and detached from (*unmounted*) the root file system by the super user.

A file system is typically made up of slices of one or more disk devices. However, a file system can also reside totally or in part in memory.

The solid-state storage device (SSD) is not supported on CRAY J90 systems.

2.1.2 File system types

The exact format of the file system data is determined by the hardware architecture, the `mkfs`(8) options used to make the file system, and the version of the UNICOS operating system under which the file system was made. The following list summarizes the types of file systems:

Type	Description
`NC1FS`	UNICOS file system on Cray PVP systems
`NFS`	Network file system (NFS)
`SFS`	Shared file system
`PROC`	`/proc` file system
`INODE`	`/inode` file system

`NC1FS` file systems are the standard UNICOS file system on Cray PVP systems.

File systems of the type `NFS` reside on a remote server, have been mounted under the UNICOS operating system, and can be accessed through NFS. For information on administering NFS file systems, see the *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

The `SFS` file system is a file system that can be shared among multiple Cray Research systems. To use the UNICOS Shared File System (SFS) feature you must obtain a software license, which is maintained and administered through the Flexible License Manager (FLEXlm) product. For information on the UNICOS SFS feature, see *Shared File System (SFS) Administrator's Guide*, Cray Research publication SG–2114.

The `/proc` file system is intended to be used by the debugging utilities to debug running processes and, to a lesser extent, as an interprocess

communication mechanism. The /proc file system is a special file system that consists of a directory in which all of the processes present in the system appear as regular files. Processes can then be read and written as though they were simple disk files.

The chown(1), chmod(1), and link(8) operations are prohibited in the /proc file system. Users may modify the regular files (representing processes) that they own. The super user may modify all files.

The /proc file system does not consume any disk space or kernel buffers. Instead, it is produced directly from information that is maintained in the kernel process table and is constantly changing as processes are created and destroyed in the system. The /proc file system never needs to be checked for damage by any of the file system repair utilities (for instance, fsck(8)) because such damage is impossible.

The output of the df(1) command presents different information for the /proc file system than for other file systems. The number listed under % disk space used refers to the percentage of memory in use. The number listed under % free refers to the percentage of memory available at the present time.

For more information about the /proc file system, see proc(4).

The /inode file system allows privileged processes access to a file or directory when the process knows the device and inode number of a file system. For more information about the /inode file system, see inode(4).

2.1.3 File system strategies

No one configuration of available disk drives into file systems will prove best for all purposes. In addition, as the needs of users change, the file system layout will most likely need to be reconfigured occasionally. In the absence of a set of absolute rules, the following facts and guidelines should prove useful when you choose the file system layout for your system.

- When organizing disks into file systems, you should first consider attributes of the user population. These attributes can provide a logical division, such as the following:
  - Location
  - Project
  - Applications
  - File-storage requirements

- Security considerations

By considering these factors, you can significantly enhance system performance.

- A large file system allows the maximum amount of resource sharing. However, a large file system takes far longer to dump and restore than several small file systems. In some cases, smaller file systems are desirable to separate users and reduce disk contention. In addition, a file system that uses a small number of disks has a lower risk of losing data because of disk failures.

  There is no universal solution to deciding on the best file system size. In most cases, a compromise is needed to maximize performance while maintaining file system recoverability.

- Because each file is contained in a single file system, a file can be no larger than the size of its file system.

- A user can take disk space from a file system only if it contains a directory (or file) with write permissions for the user. This usually means that a user can take space only in the file system that contains the user's login directory and in the file systems of shared temporary directories `/tmp` and `/usr/tmp`. Therefore, two users can avoid competition over disk space if they have login directories on different file systems and if they do not use the shared temporary directories. Many UNICOS commands use temporary directory space, but the location of the space can be controlled by the `TMPDIR` shell variable.

- You should also consider *throughput*, or the file transfer rate, when dividing disks into file systems. The two relevant factors are hardware transfer rate and head-positioning overhead.

  The hardware transfer rate is fixed and sets a limit on data transfer speed, but several disks can be combined into a single file system to provide a higher aggregate transfer rate.

  Head-positioning overhead is a problem when a single disk is split into several partitions and those partitions are active simultaneously. The time wasted in moving the head between the two partitions can decrease the effective transfer rate.

- To maintain optimum performance, you should configure your system with no more than one slice per physical device for each file system. This reduces head movement and channel contention.

2.2 File system concepts

The following sections provide an overview of the basic file system concepts. The information is presented in the following order:

- Disk organization
- Disk flawing (IOS-E and IPN-1 only)
- Disk striping
- Disk mirroring
- Physical devices
- Simple logical devices
- Striped logical devices
- Mirrored logical devices
- Logical device descriptor files

2.2.1 Disk organization

Disk drives are divided into sectors. The sectors are numbered, starting with 0. Each sector can be identified by its sector number. See the `diskspec`(7) man page for information on the sector size of disk drives used with I/O subsystem model E (IOS-E) based systems and disk drives connected to the IPI-2 I/O Node (IPN-1). See the `disksfcn`(7) man page for information on the sector size of disk drives connected to the FCN-1. See the `disksmpn`(7) man page for information on the sector size of disk drives connected to the MPN-1.

A *track* is a circle on the disk that the disk head can read in a single revolution of the disk without moving. The density of the disk determines how many sectors are in a track. The tracks are also numbered, starting from 0.

A *cylinder* is the group of tracks, one from each platter surface, with the same track number. The number of tracks per cylinder is determined by the number of surfaces within the disk device.

> **Note:** Disk devices connected to the multipurpose node 1 (MPN-1) and disk devices connected to the Fibre Channel I/O Node (FCN-1) do not organize disks into tracks or cylinders, but only into sectors.

See the `diskspec`(7) man page for a summary of the physical characteristics of the disk drives used with IOS-E based systems and disk drives connected to the

IPI-2 I/O Node (IPN-1). See the `disksfcn`(7) man page for a summary of the physical characteristics of disk drives connected to the FCN-1. See the `disksmpn`(7) man page for a summary of the physical characteristics of disk drives connected to the MPN-1.

Disk drives are divided into *slices*, which are contiguous groups of cylinders. This division is specified during disk configuration. For each slice there is a corresponding physical device node.

A *partition* is part of a file system that consists of a single slice of a disk device.

A collection of slices from one or more physical drives make up a *logical device*. Logical devices are also specified at disk configuration. For each logical device there is a corresponding logical device node.

Disk drives on IOS-E based systems and disk drives on the IPN-1 must have a slice of cylinders reserved for use by the customer engineer (*CE cylinders*) and a slice reserved as *spares cylinders* to be used in place of other cylinders of the disk that become unusable or *flawed*.

2.2.2 Disk flawing (IOS-E and IPN-1 only)

Disk drives have minute defects, or *flaws*, on the recording surface that may interfere with reading and writing data. The number of flaws and their locations vary from drive to drive. Each disk device is shipped with a *factory flaw table* that lists known flawed blocks on the disk. Disk flawing under the UNICOS system is done on a physical device basis, by replacement of bad blocks with alternative blocks from the spares cylinders.

Note: On CRAY J90 systems, disk flawing is handled by the IOS, rather than by the UNICOS system.

The information for bad blocks is kept separate from the file system on each physical device and is known only to the driver. The advantage of this mechanism is that the logical device always appears contiguous, which allows file systems to be transferred to various logical devices without regard for bad blocks.

Flawing is done twice; once before a physical device is placed in the system and again if blocks go bad during online use. Typically, flawing is done either before a new drive is placed online for the first time or once for each device when an initial system install is performed.

For information on creating physical disk device inodes that describe the spare sector map, factory flaw map, and diagnostic and customer engineering slices, see Section 2.3, page 15.

2.2.3 Disk striping

Striping is designed for moving large amounts of data at very high bandwidths, higher than one can normally achieve with existing disk drives. With this technique, several drives are combined into one logical unit. Data in *n*-sector size pieces is written to and read in from the drives in a round-robin fashion, allowing I/O to the individual disks to be overlapped. The drives of a stripe group must all be the same type; for example, you cannot have a stripe group of two DD-49 disk drives and one DD-40 disk drive.

Note: CRAY J90 systems do not support IOS disk striping.

The overlapping of I/O operations causes the increased bandwidth. Each drive in the *stripe group* (group of striped disks) can have an I/O operation active simultaneously at any time. For example, consider a four-drive stripe group to which you want to write several tracks of data. A write request for a track is started for the first drive then a write request for the second track to second drive is started without waiting for the first write to complete, and so on. The I/O operations are asynchronous between drives in the stripe group.

The disadvantage to striping is that sequential data is scattered across several disks. Losing any one of the disks of the stripe group ruins the striped files (for a four-drive stripe group with a striping factor of one track, you would miss every fourth track).

In most cases, the only device that you should use for striping is `SWAPDEV`, as it is usually the only device with I/O requests large enough for striping to be advantageous. You should not use striping for `/root` or `/usr` file systems.

2.2.4 Disk mirroring

Mirroring is used to provide data redundancy when data integrity is important. It is implemented by using two to eight slices, usually on as many different physical disks, each of the same size. A write operation to a mirrored device causes separate write operations to be performed on each of the components. A read operation may be performed on any of the component devices.

For instructions on creating mirrored devices, see Section 2.4.4.3, page 26. For a description of mirrored file systems, see Section 2.10, page 47.

2.2.5 Physical devices

There are four types of physical-level devices, each with its own device driver, as follows:

- Disk drive
- Solid-state disk
- RAM disk
- Network or HIPPI disk device

By convention, disk drive, solid-state disks, and RAM disk device files are kept in the `/dev/pdd` directory and HIPPI disk device files are kept in the `/dev/hdd` directory. Disk drives connected to the MPN-1 or HPN-1 on a GigaRing channel are kept in the `/dev/xdd` directory.

For instructions on creating physical devices, see Section 2.4.1, page 17.

2.2.6 Simple logical devices

A *logical disk device* is a collection of blocks on one or more physical disks or other logical disk devices. A *logical direct device* indicates that the logical disk includes exactly one partition or physical slice. A *logical indirect device* indicates that the logical disk includes more than one partition or physical slice.

Logical drivers call the physical drivers by using the device switch mechanism. By convention, logical device files are kept in the `/dev/dsk` directory.

The simple logical device is the highest level driver. Any mountable file system will interface with the driver at this level. Logical device cache is supported only at this level.

For information on creating logical devices, see Section 2.4.4, page 23.

2.2.7 Striped logical devices

A striped logical device is a means of combining two or more slices of two different physical devices together to increase bandwidth. This is best for heavily used partitions like the swap device where very large chunks of data are being transferred.

Striped logical devices consist of physical device names lists, and can be combined into simple logical devices. Striped logical device routines can be

called directly with the `open`(2), `close`(2), `read`(2), `write`(2), and `ioctl`(2) system calls or from another logical device driver.

By convention, striped logical device files are kept in the `/dev/sdd` directory.

For information on creating striped logical devices, see Section 2.4.4.2, page 25.

2.2.8 Mirrored logical devices

A mirrored logical device is used to provide data redundancy where data integrity is important. It consists of two or more slices, typically on as many different physical devices, each of the same size. A mirrored logical device is similar in configuration to a striped device. A write operation to a mirrored device causes separate write operations to be performed on each of the components. A read operation may be performed on any of the component devices.

Mirrored logical devices are made up of lists of physical device names, and can be combined into simple logical devices. Mirrored logical device routines can be called directly with the `open`, `close`, `read`, `write`, and `ioctl` system calls, or from another logical device driver, but not concurrently.

By convention mirrored logical device files are kept in the `/dev/mdd` directory.

For more information on `mdd` files, see the `mdd`(4) man page. For instructions on creating mirrored devices, see Section 2.4.4.3, page 26. For a description of mirrored file systems, see Section 2.10, page 47.

2.2.9 Logical device descriptor files

A logical device descriptor file is used to combine one or more character special disk files to form a single logical disk device. A logical device descriptor file is a list of absolute path names of striped, mirrored, physical, and HIPPI devices. By convention, logical descriptor files are kept in the `/dev/ldd` directory.

For instructions on creating logical descriptor files, see Section 2.4.4, page 23.

2.3 Using the `mkspice`(8) command (IOS-E and IPN-1)

When configuring a disk for the first time, use the `mkspice`(8) command to create physical disk device inodes that describe the spare sector map, factory flaw map, and diagnostic and customer engineering slices. You name the physical disk devices according to their I/O paths. The following example

creates the spare, ift, diagnostic, and ce slices for DD-49s on cluster 0, IOP 1, channel 30; cluster 1, IOP 2, channel 32; and cluster 1, IOP 3, channel 34:

```
/etc/mkspice -t dd49 0130 01232 01334
```

The mkspice(8) command is not supported on CRAY J90 systems.

The -i option of the mkspice(8) command initializes the spare maps from the ift nodes. It also initializes the /etc/aft files (ASCII flaw files) which are used with the bb(8) (bad block) command. See aft(5) for information on the aft files and bb for information on creating the bad block files from the aft files.

The -i option of the mkspice(8) command invokes the ift(8) command to read the Factory Flaw table from the /dev/ift node. The following is an example of ift output.

```
*
*  engineering flaw table for DD49
*
*  factory flaw map date: 10-08-86
*
*       S/N     C2236

#       0 0 0 0

*       count   head    sector  cylinder

        1       3       0       1333
        1       3       1       1333
        1       3       24      1333
```

The -i option of the mkspice(8) command also invokes the spmap(8) command, which generates and writes a physical disk spare map. The spmap command reads flaw information from standard input in the same format written by the ift(8) command. The output from ift can be piped directly into spmap, or the output from ift can be written to an ASCII Flaw table and then piped into spmap.

It is recommended that you create the ASCII Flaw tables. If a new flaw develops, it must be added to the end of the ASCII Flaw table with a text editor. The spmap command then needs to be rerun. If the ASCII Flaw table is

lost, you should re-create it, using`spmap` to ensure that the ordering of the alternate blocks is preserved. For example:

```
/etc/spmap -r /dev/spare/0130 > /etc/aft/0130
```

2.4 Creating file system nodes

Disk partitions and logical devices are defined with the `mknod`(8) command. Only the `root` and `swap` partitions are defined in the parameter file during startup.

This section provides information on the following areas of creating file systems:

- Creating physical devices
- Examples of physical device creation
- Creating physical devices (GigaRing systems)
- Creating logical devices
- Creating logical descriptor files
- Defining alternate disk paths
- Shared dump and swap configuration

2.4.1 Creating physical devices

> **Note:** For information on creating physical devices on GigaRing based systems, see Section 2.4.3.

A disk slice is defined by an I/O path, a unit number, a starting sector number, and a length in sectors. Slices must be aligned on track boundaries, and in practice they are often aligned on cylinder boundaries.

To create a partition, you must first use the `mknod`(8) command to create a character special file or node and specify a major and minor device number for each slice you want to create. The major device number should be expressed symbolically.

Symbol	Device
`dev_pdd`	Disk devices.
`dev_rdd`	RAM disk devices.

`dev_ssdd` SSD devices. (SSD devices are not supported on CRAY J90 systems.)

The minor device number ranges from `0` to `PDDSLMAX` for disk devices, `0` to `RDDSLMAX` for RAM disk devices, and `0` to `SSDDSLMAX` for SSD devices.

The following minor device numbers have specific designations:

Number	Designation
`250`	The `root` partition on the `fsload` tape.
`251`	The `swap` partition in the initial UNICOS kernel.
`252`	The `diagnostic` partitions.
`253`	The `ce` partitions (customer engineering).
`254`	The `spare` partitions (Factory Flaw table).
`255`	The `ift` partitions. The number `255` is special in that only nodes with this minor device number can be used on the `ioctl`(8) command to bring a disk device up after it has gone down.

Only one node of a minor device number can be open at any given time. With the exception of the `diagnostic`, `ce`, and `ift` nodes, do not create two disk nodes with the same minor device number.

The physical device driver fills in internal driver structures and checks for conflicts in the device open routine. An array of slice structures are indexed by the minor device number within a given physical device driver.

The following partitions can be opened by the system without using the nodes in `/dev` (but may still be accessed through nodes):

- `root`
- `swap`

The index, offset, and length of these partitions is passed into the UNICOS kernel through the parameter file. The parameter file index, offset, and length must match the node information. A node for `root` is needed to run the `fsck`(8) command; a node for `swap` is required for defining flaws; and a node for `dump` is required for snatching dumps.

The following example shows the `mknod` command needed to create the diagnostic physical device node for a DD-60 disk drive:

```
/etc/mknod /dev/ddd/2230.0 c dev_pdd 252 10 02230 0 120106 0137 0 0
```

Component	Definition
`/dev/ddd/2230.0`	Device path name
`c`	Character special
`dev_pdd`	Major device number
`252`	Minor device number
`10`	Disk type
`02230`	I/O path in octal
`0`	Starting sector number
`120106`	Length of slice in sectors
`0137`	Special purpose flags
`0`	Alternate I/O path for redundancy
`0`	Disk unit number

2.4.2 Examples of physical device creation

Note: For information on creating physical devices on GigaRing based systems, see Section 2.4.3.

The following sections give detailed examples of creating the following types of physical devices:

- Disk
- RAM
- SSD (Not supported on CRAY J90 systems)

Note: When working with I/O paths and flags, remember that octal numbers must have a leading 0. Because I/O paths are usually expressed in octal, take care when working with a multicluster system to express I/O paths in the proper form to a command. For example, I/O path `0130` works as expected because of the leading 0, but I/O path `1232` needs a leading 0 when expressed to a UNICOS command.

2.4.2.1 Creating a physical disk device

To create a physical disk device, use the `mknod`(8) command. For information about the physical disk device interface, including the supported disk types, see the `pdd`(4) man page.

Slices normally begin and end on cylinder boundaries.

The I/O path is a concatenation of the cluster, EIOP, and channel numbers. The major number is `dev_pdd`, and the minor number should be less than `PDDSLMAX`, as defined in the parameter file.

The available flags are defined in the `/usr/include/sys/eslice.h` file as follows:

```
/*
 *  flags for physical slice control
 */
#define S_CONTROL       0001    /* control device */
#define S_NOBBF         0002    /* no bad block forwarding */
#define S_NOERREC       0004    /* no error recovery */
#define S_NOLOG         0010    /* no error logging */
#define S_NOWRITEB      0020    /* no write behind */
#define S_CWE           0040    /* control device write enable */
#define S_NOSPIRAL      0100    /* no spiraling */
#define S_NOSORT        0200    /* no disk sorting */
```

Example 1: The following commands define two slices on a DD-49 on cluster 1, EIOP 2, channel 32. The slice `dd49_0` starts at cylinder 0 and spans 732 (01334) cylinders; its flags field is 0, there is no alternate I/O path, and its unit field is 0. `dd49_1` starts at cylinder 732 (01334) and spans 150 (0226) cylinders; its flags field is 0, there is no alternate I/O path, and its unit field is 0.

```
/etc/mknod /dev/pdd/dd49_0 c dev_pdd 50 3 01232      0 245952 0 0 0
/etc/mknod /dev/pdd/dd49_1 c dev_pdd 51 3 01232 245952  50400 0 0 0
```

Example 2: The following commands define two slices spanning an entire DD-60 on cluster 0, EIOP 1, channel 30 (the DD-60s are daisy-chained and are on units 0 and 1):

```
/etc/mknod /dev/pdd/dd60_0 c dev_pdd 200 10 0130 0 119692 0 0 0
/etc/mknod /dev/pdd/dd60_1 c dev_pdd 201 10 0130 0 119692 0 0 1
```

Use the `stor`(8) command to examine your partitions. The following example shows the `stor` output that you would receive if you typed in the preceding `mknod`(8) commands:

```
DD60 0130.0

ID                      START               END         LENGTH
----------------------  ------------------  ---------  ----------------
name             minor    block     cyl.hd     cyl.hd    blocks  mbytes

dd60_0             200        0       0.00   05052.00    119692   1961.0


DD60 0130.1

ID                      START               END         LENGTH
----------------------  ------------------  ---------  ----------------
name             minor    block     cyl.hd     cyl.hd    blocks  mbytes

dd60_1             201        0       0.00   05052.00    119692   1961.0


DD49 1232

ID                      START               END         LENGTH
----------------------  ------------------  ---------  ----------------
name             minor    block     cyl.hd     cyl.hd    blocks  mbytes

dd49_0              50        0       0.00   01334.00    245952  1007.4
dd49_1              51   245952   01334.00   01562.00     50400   206.4
```

2.4.2.2 Creating RAM disks

The amount of core memory available for creating RAM disks is specified in the parameter file, as shown in the following example:

```
RAM ramdev { length 10240 blocks;
        pdd ram        {minor  3; block 0      ; length 10240 blocks;}
}
```

To create a RAM disk physical device, use the `mknod`(8) command. The device type and I/O path parameters are not applicable and should be 0. The major number is `dev_rdd`, and the minor number should be less than `RDDSLMAX`, as defined in the parameter file. The following example shows the creation of a RAM physical device:

```
/etc/mknod /dev/pdd/ram0 c dev_rdd 0 0 0    0 1024
/etc/mknod /dev/pdd/ram1 c dev_rdd 1 0 0 1024 9216
```

2.4.2.3 Creating SSD slices

The amount of SSD memory available for creating SSD slices is specified in the parameter file, as shown in the following example. (SSD devices are not supported on CRAY J90 systems.)

```
SSD ssddev { length 10240 blocks;
        pdd ssd        {minor  3; block 0      ; length 10240 blocks;}
}
```

To create an SSD physical device, use the `mknod`(8) command. The device type and I/O path parameters are not applicable and should be 0. The major number is `dev_ssdd`, and the minor number should be less than `SSDDSLMAX`, as defined in the parameter file. The following example shows the creation of an SSD physical device:

```
/etc/mknod /dev/pdd/ssd0 c dev_ssdd 0 0 0    0 1024
/etc/mknod /dev/pdd/ssd1 c dev_ssdd 1 0 0 1024 9216
```

For information on using the SSD as a logical device, see Section 2.11.3, page 56.

2.4.3 Creating physical devices (GigaRing systems)

You define a disk slice on a GigaRing based system just as you define a disk slice on an IOS-E based system, with the following considerations:

- The major device number for a disk device connected to the IPN-1 should be expressed symbolically as `dev_qdd`. See the `qdd`(4) man page.
- The major device number for a disk device connected to the MPN-1 or the FCN-1 should be expressed symbolically as `dev_xdd`. See the `xdd`(4) man page.
- The I/O path for a physical device on a GigaRing based system is a concatenation of the GigaRing number, the node number, and the controller number of the device.
- By convention, `xdd` device files are kept in the `/dev/xdd` directory. `qdd` device files, on the other hand, are kept in the `/dev/pdd` directory.

The following example shows the `mknod` (8) command needed to create a physical device node on a DD-318 disk drive connected to the MPN-1:

```
/etc/mknod /dev/xdd/s100a c dev_xdd 2 0 03021  0     100000 0 0 1 0
```

Component	Definition
`/dev/xdd/s1000a`	Device path name
`c`	Character special
`dev_xdd`	Major device number
`2`	Minor device number
`0`	Disk type
`03021`	I/O path in octal
`0`	Starting sector number
`100000`	Length of slice in sectors
`0`	Special purpose flags
`0`	Alternate I/O path for redundancy
`1`	Disk unit number
`0`	Disk subunit number

2.4.4 Creating logical devices

Logical devices are categorized into the following types:

- Simple logical
- Striped logical
- Mirrored logical

Each type has its own major number and associated device driver.

Logical devices are created by using the `mknod`(8) command, which has the following format:

```
/etc/mknod  name  b  major  minor  0  0  path
  /etc/mknod  name  c  major  minor  0  0  path
```

name	Name of the logical device.
`b`	Block special device.
`c`	Character special device.
major	Major device number.

minor	Minor device number in the range `1` through `LDDMAX`. You cannot use `minor 0` for a logical device.
`0 0`	Placeholders for future use.
path	Absolute path name to a physical device, logical device, or a logical descriptor file.

2.4.4.1 Creating simple logical devices

Simple logical devices can point to a physical device or a logical descriptor file and use major number `dev_ldd`. The minor number must be less than `LDDSLMAX`, as defined in the parameter file.

The following example command sequence creates a simple logical direct device and its associated physical slice. The logical direct device has a minor device number of 100, it is on a DD-49, on IOC 0, IOP 1, channel 30, starting at 0 with a length of 32,768 sectors.

```
/etc/mknod /dev/pdd/x0 c dev_pdd 100 3 0130 0 32768 0 0 0 0
/etc/mknod /dev/dsk/x0 b dev_ldd 100 0 0 /dev/pdd/x0
```

To examine logical device nodes, use the `ddstat`(8) command. The following is an example of `ddstat` output for the preceding `mknod`(8) commands:

```
# /etc/ddstat /dev/dsk/x0
x0 b 34/100 /dev/pdd/x0
        /dev/pdd/x0 c 32/100 3 0130 0 32768 00 0 0 0
```

The following example command sequence creates a simple logical indirect device and its associated physical slices:

```
/etc/mknod /dev/pdd/x0 c dev_pdd 100 3 0130 0 32768 0 0 0 0
/etc/mknod /dev/pdd/y0 c dev_pdd 110 3 0132 0 32768 0 0 0 0
/etc/mknod /dev/ldd/cluster0 L /dev/pdd/x0 /dev/pdd/y0
/etc/mknod /dev/dsk/cluster0 b dev_ldd 30 0 0 /dev/ldd/cluster0
```

The following is an example of the associated `ddstat` output:

```
# /etc/ddstat /dev/dsk/cluster0
cluster0 b 34/30 /dev/ldd/cluster0
        /dev/pdd/x0 c 32/100 3 0130 0 32768 00 0 0 0
        /dev/pdd/y0 c 32/110 3 0132 0 32768 00 0 0 0
```

2.4.4.2 Creating striped logical devices

Striped logical devices must point to a logical descriptor file. Striped logical devices use major number `dev_sdd`, as defined in the parameter file.

The following example command sequence creates a striped logical device, its associated physical slices, logical descriptor file, and direct logical device:

```
/etc/mknod /dev/pdd/x0 c dev_pdd 100 3 0130 0 32768
/etc/mknod /dev/pdd/y0 c dev_pdd 110 3 0132 0 32768
/etc/mknod /dev/ldd/stripe0 L /dev/pdd/x0 /dev/pdd/y0
/etc/mknod /dev/sdd/stripe0 b dev_sdd 30 0 0 /dev/ldd/stripe0
/etc/mknod /dev/dsk/stripe0 b dev_ldd 30 0 0 /dev/sdd/stripe0
```

The associated `ddstat` output is as follows:

```
# /etc/ddstat /dev/dsk/stripe0
stripe0 b 34/30 /dev/sdd/stripe0
       /dev/sdd/stripe0 b 39/30 /dev/ldd/stripe0
              /dev/pdd/x0 c 32/100 3 0130 0 32768 00 0 0 0
              /dev/pdd/y0 c 32/110 3 0132 0 32768 00 0 0 0
```

The following procedure provides an example of how to define devices that include SSD devices.

1. Define physical device slices of equal size where each device is on a unique I/O path (argument 6), each is of the same length (argument 8) and each has the same stripe I/O unit (definable on SSD only).

```
/etc/mknod /dev/pdd/ssd0_s1 c dev_ssdd 10 3 001 0 1000000 0 0 0 1024
/etc/mknod /dev/pdd/ssd1_s1 c dev_ssdd 20 3 002 0 1000000 0 0 0 1024
/etc/mknod /dev/pdd/ssd2_s1 c dev_ssdd 30 3 004 0 1000000 0 0 0 1024
/etc/mknod /dev/pdd/ssd3_s1 c dev_ssdd 40 3 010 0 1000000 0 0 0 1024
```

2. Define a logical node to reference the four stripe components.

```
/etc/mknod /dev/ldd/s1 L /dev/pdd/ssd0_s1
                         /dev/pdd/ssd1_s1
                         /dev/pdd/ssd2_s1
                         /dev/pdd/ssd3_s1
```

3. Define a striped device.

```
/etc/mknod /dev/sdd/s1 c dev_sdd 10 0 0 /dev/ldd/s1
```

4. Define a logical disk device to reference the striped device.

```
/etc/mknod /dev/dsk/s1 b dev_ldd 10 0 0 /dev/sdd/s1
```

You can use the `ddstat`(8) or the `stor`(8) command to verify the device structure as defined. This verification is done by reading the appropriate inodes and files. The device is not opened when `ddstat` or `stor` performs a read operation. Executing `/etc/ddstat` on the configuration example would show the following:

```
/dev/dsk/s1 b 34/10 0 0 /dev/sdd/s1
      /dev/sdd/s1 c 39/10 0 0 /dev/lss/s1
          /dev/pdd/ssd0_s1 c 37/10 3 001 0 1000000 0 0 0 1024
          /dev/pdd/ssd1_s1 c 37/20 3 002 0 1000000 0 0 0 1024
          /dev/pdd/ssd2_s1 c 37/30 3 004 0 1000000 0 0 0 1024
          /dev/pdd/ssd3_s1 c 37/40 3 010 0 1000000 0 0 0 1024
```

Errors in configuration are generally detected by the `ddstat` or `stor` command or by the device drivers when the logical disk device is first opened. Errors at open time are issued to the console and the kernel log.

2.4.4.3 Creating mirrored logical devices

Mirrored logical devices must point to a logical descriptor file and use major number `dev_mdd`. The minor number must be less than `LDDSLMAX`, as defined in the parameter file. The following command sequence creates a mirrored logical device, its associated physical slices, logical descriptor file, and direct logical device:

```
/etc/mknod /dev/pdd/x0 c dev_pdd 100 3 0130 0 32768
/etc/mknod /dev/pdd/y0 c dev_pdd 110 3 0132 0 32768
/etc/mknod /dev/ldd/mirror0 L /dev/pdd/x0 /dev/pdd/y0
/etc/mknod /dev/mdd/mirror0 c dev_mdd 30 0 077 /dev/ldd/mirror0
/etc/mknod /dev/dsk/mirror0 b dev_ldd 30 0 0 /dev/mdd/mirror0
```

The associated `ddstat` output is as follows:

```
# /etc/ddstat /dev/dsk/mirror0
mirror0 b 34/30 /dev/mdd/mirror0
       /dev/mdd/mirror0 b 40/30 0 077 /dev/ldd/mirror0
              /dev/pdd/x0 c 32/100 3 0130 0 32768 00 0 0 0
              /dev/pdd/y0 c 32/110 3 0132 0 32768 00 0 0 0
```

Each member of a mirrored group (in the previous example `pdd/x0` and `pdd/y0`) contains a *rwmode* parameter. The bits of this parameter control read,

write, and initialize privilege for each member; each octet represents (from right to left) r-w-x (read/write/x(init)). For a *rwmode* of `077`, as shown in the preceding example, both `x0` and `y0` are read/write/x(init). A *rwmode* of `073` would indicate initialize `x0` and `y0`, read only from `y0`, and write to both `x0` and `y0`. A *rwmode* of `037` would indicate that `x0` is read-enabled and `y0` is not.

See the `mdd`(4) man page for more information on `mdd` files. For a more complete overview of mirrored file systems, see Section 2.10, page 47.

2.4.4.4 Restrictions on striped and mirrored logical devices

The following restrictions and guidelines apply to the configuration of striped and mirrored groups:

- All slices must be of the same length.
- All devices on a striped group must be up. When configuring a mirror group, only one member needs to be up.
- The starting sector of each slice must be a multiple of the stripe factor.
- The length of each slice must be a multiple of the stripe factor.
- When a stripe group is opened, the driver allocates a number of `PBUF` headers equal to nine times the number of slices. For mirror groups, the number is three times the number of slices. If there are not enough `PBUF` headers free, the driver waits. If not enough `PBUF` headers are configured, the system may hang.

2.4.5 Creating logical descriptor files

A logical descriptor file can contain up to 64 absolute path names, each which can be up to 48 characters in length. Each absolute path name is a *member* of the logical disk device. The members are combined in a manner prescribed by the character or block special device referencing the logical descriptor file. The members can be physical or logical devices.

You can use the `mknod` command to create a logical descriptor file, using the following format:

```
/etc/mknod name L member0 [member1 member2] ...
```

Example 1: To create a logical descriptor file `x` containing two physical slices of type `x0` and `x2`, specify the following:

```
/etc/mknod /dev/ldd/x L /dev/pdd/x0 /dev/pdd/x1
```

Example 2: To create a logical descriptor file `y` containing two logical striped devices of type `y0` and `y2`, specify the following:

```
/etc/mknod /dev/ldd/y L /dev/sdd/y0 /dev/sdd/y1
```

2.4.6 Defining alternate disk paths

The UNICOS operating system allows you to define an alternate path to a disk when you define the device node by using the `mknod`(8) command. The alternate path provides a backup for the primary path to a device. If you configure a device with an alternate path, it is initialized when you open the device. The system then can attempt error recovery by means of the alternate path when it detects a hard failure on the active disk path.

A *path* is defined as the hardware resources between the CPU and the disk device. On IOS-E based systems, these consist of the low-speed channel to the mainframe, the multiplexing I/O processor (MUXIOP), the low-speed channel from the MUXIOP to a given model E I/O processor (EIOP), and the EIOP itself with its channel adapter.

You can access a disk through the alternate path if any element of the primary path is not functional and hardware elements exist to connect to a second port of the disk device. Optimally, on an IOS-E based system, the alternate path should include a different I/O cluster and EIOP.

You create an alternate path when you define a physical device interface with the `mknod`(8) command. The following example defines an alternate I/O path of 1130 to device 0236.7.

```
mknod /dev/pdd/device.7 c dev_pdd 15 10 0236 1472 32768 0 01130 7 0
```

A path of 1130 defines IOC 1, IOP 1, and channel 30.

The alternate path is used as part of error recovery. If the standard five retries and micro-sequencing error recovery fails, the disk driver attempts recovery on the alternate path. This can recover errors such as high-speed channel (HISP) errors and errors on the channel adaptors or in the EIOP. Errors that are on the disk itself remain as errors, just as on the primary path.

A device running on the alternate path does not switch back to the primary path if normal error recovery fails. Errors that occur during write-behind, when

the CPU no longer has the data, cannot be recovered by switching to a different path, because data cannot be retrieved from the IOS.

If a device is configured with an alternate path, the system, by default, switches to the alternate path during error recovery. You can disable and enable the error recovery switch with the `autoswitch` option of the `sdconf`(8) command, as follows.

`sdconf` *device* `autoswitch on | off`

You can control both the primary and alternate path by using the `sdconf` command. Executing `sdconf` *device pripath* and `sdconf` *device altpath* resets the primary or alternate path.

You can also change the path to a device with the `inform`(8) command on the OWS-E. If the `pddinform` function in the driver is notified that a given EIOP has died, any device running on that EIOP that has an alternate path switches to the alternate path to complete any outstanding I/O.

2.4.7 Configuring alternate paths on FCN devices

FCN devices can be dual-ported. If these devices are dual-ported, the `XDD` driver has the capability to dynamically switch to the alternate path when the primary path fails. If the alternate path fails, the `XDD` driver will not switch back to the primary path.

An FCN device can be dual-ported in any of three configurations:

- 2 Fibre Channel Loops on the same FCN device
- 2 FCN devices on the same GigaRing channel
- 2 FCN devices, each on a different GigaRing channel.

These configurations are illustrated in "Configuring alternate paths," Section 2.4.7.1.

2.4.7.1 Configuring alternate paths on FCN devices

The following diagram illustrates a system in which no devices are dual-ported and no alternate paths are defined.

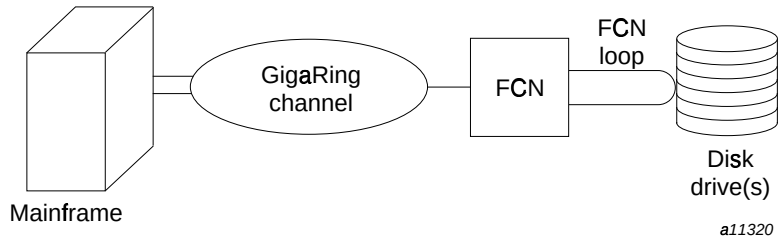

Figure 1. Configuration with no alternate path

The following diagram illustrates a system configured with two Fibre Channel Loops on a single FCN device.

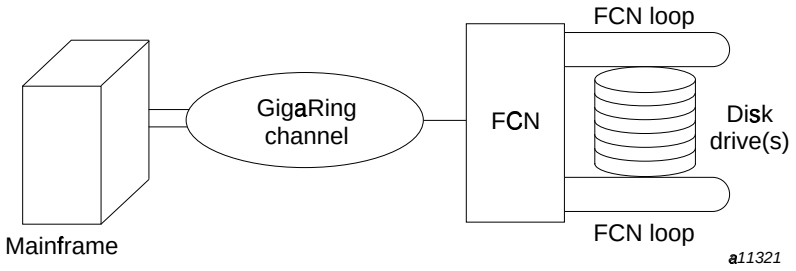

Figure 2. Configuration 1: One FCN device, two Fibre Channel Loops

Configuration 1 shows the following path configuration:

primary path	Ring 1 Node 1 Channel 1 Unit 1
alternate path	Ring 1 Node 1 Channel 2 Unit 1

The following command creates a device node with the primary and alternate path definitions of Configuration 1:

```
/etc/mknod dd308 c 33 40 0 01011 0 2340000 0 01012 01 0
```

The following diagram illustrates a system configured with two FCN devices on the same GigaRing channel.

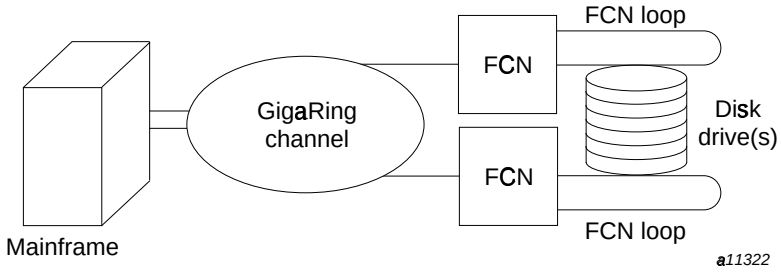

Figure 3. Configuration 2: Two FCN devices, one GigaRing channel

Configuration 2 shows the following path configuration:

primary path Ring 1 Node 1 Channel 1 Unit 1

alternate path Ring 1 Node 2 Channel 1 Unit 1

The following command creates a device node with the primary and alternate path definitions of Configuration 2:

```
/etc/mknod dd308 c 33 40 0 01011 0 2340000 0 01021 01 0
```

The following command creates a device node with the primary and alternate path definitions of Configuration 1:

```
/etc/mknod dd308 c 33 40 0 01011 0 2340000 0 01012 01 0
```

The following diagram illustrates a system configured with two FCN devices on different GigaRing channels.

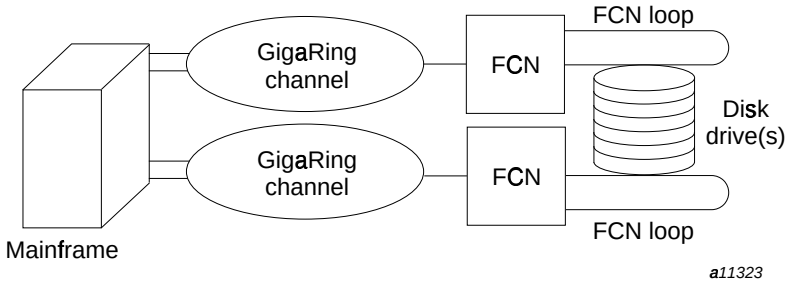

Figure 4. Configuration 3: Two FCN devices, two GigaRing channels

Configuration 3 shows the following path configuration:

primary path	Ring 1 Node 1 Channel 1 Unit 1
alternate path	Ring 2 Node 2 Channel 1 Unit 1

The following command creates a device node with the primary and alternate path definitions of Configuration 3:

```
/etc/mknod dd308 c 33 40 0 01011 0 2340000 0 02021 01 0
```

2.4.7.2 Failure modes

Dynamic switching to the alternate path takes place under the following conditions:

- Fibre Channel Loop failure
- FCN failure
- GigaRing channel failure

The following sections describe the system response to each of these conditions.

2.4.7.2.1 Fibre Channel Loop failure

When the mainframe detects a failure of a request to the FCN that is associated with the Fibre Channel Loop, and if there is an alternate path to the device through another Channel Loop on the same FCN, another FCN on the same GigaRing channel, or another GigaRing channel, the dynamic path switch algorithm will be executed.

The FCN return a status indicating that this is a failure of the Fibre Channel Loop.

You can force this type of failure by disconnecting the cable to the port or disconnecting the Fibre Channel Loop cable.

2.4.7.2.2 FCN failure

Failure of the FCN is detected by the PEER to PEER message protocol layer. This condition is detected after failure of any communication from the FCN with the mainframe after 30 seconds. At this time, the `XDD` driver is informed of a PEER down message condition.

If there is an alternate path for these devices through another FCN on the same GigaRing channel or another GigaRing channel, the XDD driver will execute the alternate path switching algorithm.

You can force this type of failure by disconnecting the FCN from the GigaRing channel.

2.4.7.2.3 GigaRing channel failure

Failure of the GigaRing channel is detected by the PEER to PEER message protocol layer. This condition is detected after failure of any communication from the FCN with the mainframe after 30 seconds. At this time, the XDD driver is informed of a PEER down message condition. The system cannot return a status that distinguishes between a GigaRing channel failure and an FCN failure.

If there is an alternate path for these devices either through another FCN on the same GigaRing channel or another GigaRing channel, the XDD driver will execute the alternate path switching algorithm.

If the alternate path is on another FCN on the same GigaRing channel, the path switch will fail. If the alternate path is on a different GigaRing channel, the path switching will be successful if the FCN is functional.

You can force this type of failure by disconnecting the GigaRing cable.

2.4.7.2.4 Alternate path switching restrictions

The following restrictions apply to the use of the alternate path:

- On first open, both paths must be valid and functional; that is, the XDD driver must be able to open both paths on first open.
- Once the device is being used on the alternate path, failure of the alternate path will not cause the XDD driver to switch to the primary path.

2.4.8 Shared dump and swap configuration

Under the UNICOS operating system, you can configure the same physical slice that you use for a dump partition as part of the swap device. The dump header is preserved permanently and space used by a system dump will be allocated as needed for system dumps. The space is released back to the swap device when the dump is processed by cpdmp(8). A single slice swap device may be shared with the dump device.

The following restrictions apply when configuring a shared slice:

- The slice must be disk-based.
- SSD or RAM slices may not be used as the shared slice.
- The slice cannot be a member of a striped or mirrored device.
- The slice must be defined as a `pdd` type member of the swap device.

In the following swap configuration, slice dump may be chosen as the shared slice, but slices `ssd_swap`, `diskswap1`, and `diskswap2` may not.

```
SSD  ssd      {
     length 1024 Mwords;
     pdd ssd_sds    {minor    4; block         0; length  1835008 blocks;}
     pdd ssd_swap   {minor    6; block   1835008; length  262144  blocks;}

disk d0130.1  {type DD60; iopath{cluster 0; eiop 1; channel 030;}
     unit 1;
     pdd dump      {minor   22; sector       0; length  119692 sectors;}

disk d0236.0  {type DD60; iopath{cluster 0; eiop 2; channel 036;}
     unit 0;
     pdd diskswap1  {minor   67; sector       0; length  119692 sectors;}
}
disk d1236.0  {type DD60; iopath{cluster 1; eiop 2; channel 036;}
     unit 0;
     pdd diskswap2  {minor   68; sector       0; length  119692 sectors;}


   sdd stripe_swap { minor 10; pdd diskswap1;
                               pdd diskswap2;

   ldd swap      { minor  10; pdd ssd_swap;
                              pdd dump;
                              sdd stripe_swap;
   swapdev is ldd swap;
```

Both a swap logical device and a dump logical device must be defined with the shared slice as a component. The minor number of the dump logical device is different than the minor number of the swap logical device, but they are not

required to be the same as the examples shown. For the above example the dump device definition would be as follows:

```
ldd dump      { minor  19; pdd dump    ;
              }
```

The `dumpdev` statement is provided below.

To initialize and use the shared partition, 3 different boot parameter files must be used.

For dump device initialization, the UNICOS system must be booted one time in a nonshared configuration. The `mkdmp` command should be run as shown in the `mkdmp`(8) man page. To initialize the dump device in the above example, use the following parameter file definitions:

```
SSD  ssd      {
     length 1024 Mwords;
     pdd ssd_sds    {minor    4; block         0; length  1835008 blocks;}
     pdd ssd_swap   {minor    6; block   1835008; length  262144  blocks;}

disk d0130.1  {type DD60; iopath{cluster 0; eiop 1; channel 030;}
     unit 1;
     pdd dump      {minor   22; sector       0; length  119692 sectors;}

disk d0236.0  {type DD60; iopath{cluster 0; eiop 2; channel 036;}
     unit 0;
     pdd diskswap1  {minor   67; sector       0; length  119692 sectors;}
}
disk d1236.0  {type DD60; iopath{cluster 1; eiop 2; channel 036;}
     unit 0;
     pdd diskswap2  {minor   68; sector       0; length  119692 sectors;}


   sdd stripe_swap { minor 10; pdd diskswap1;
                               pdd diskswap2;

   ldd swap      { minor  10; pdd ssd_swap;
                              sdd stripe_swap;

   ldd dump      { minor  19; pdd dump    ;
                 }
```

```
swapdev is ldd swap;
dmpdev  is ldd dump;
```

After the dump device is initialized, the shared configuration should be used. Reinitialization of the dump device is only necessary if the dump device is changed or if flaws are added or removed on the dump device. If reinitialization is necessary, the nonshared configuration must booted for just the `mkdmp`(8) processing.

To boot the UNICOS system in a shared dump/swap configuration, the `dmpdev` statement must indicate that the dump device is a `pdd` device:

```
dmpdev is pdd dump;
```

To perform a system dump, the `dumpsys`(8) command requires that the `dmpdev` statement indicate that the dump device is an `ldd` device, so the boot parameter file cannot be used:

```
dmpdev is ldd dump;
```

It is suggested that the shared slice be named `dump`. This allows the `cpdmp`(8) command to use defaults and will be easier to implement onsite.

2.5 Configuring disk arrays

This section contains information on how to configure DD-308/FCNs in RAID-3.

2.5.1 Installing an array

Perform the following steps to install and configure a disk array:

- Copy any data to be saved to another media
- Initialize the device as a RAID-3 array
- Write data to the parity drive
- Modify configuration information

After you perform these steps, the array is ready for I/O. These steps are described in detail below.

1. Copy any data to be saved to another media.

   It is not possible to move a filesystem on 4 single drives onto an array of 4+1 drives without first dumping, then restoring the data.

There are two factors that may affect the restore of the data:

- If there are many small files, the restored data could take more space. This is because the minimum sector size is four blocks, so small files may take more space.
- For extended files, the space allocation may be more efficient after the restore and take less space.

Unless the existing file system is nearly full, it is unlikely that it will not restore.

2. Initialize the device as a RAID-3 array:

   `xdms -a init -m 35 /dev/xdd/`*name*

   or

   `xdms -a init -m 35 xdms -a init -m 35 02010.11 # 011 (Octal)`

   See the `xdms`(8) man page for details.

   The syntax `-m 35` means raid 3, 5 units. The 35 does not indicate a mask, but a hex code identifying the raid type (mode) to be initialized.

3. Write data to parity drive.

   `xdms -a scrub /dev/xdd/`*name*

   or

   `xdms -a scrub 02010.11`

   A scrub of a DA-308 takes approximately thirty minutes.

   If the scrub fails or is interrupted with Control-C, you will see errors when you execute `mkfs`(8) on the file system

4. Modify configuration information:

   `/etc/mknod /dev/xdd/`*name* `c 33` *minor* `4` *pripath* `0 2340000 0` *altpath unit*

   The *dtype* field needs to be 4 (to indicate a RAID-3 device); the 4 specifies that there are four 512-word blocks per sector. *Loop_ID* is sometimes used instead of *unit* number. Only one `/dev/xdd/`*XXXX* node is needed to represent an entire array.

The following example shows the output from a `ddstat -m disk*` command on an array:

```
/etc/mknod disk1 c 33 39 4 02010 0 2340000 0 0 01 0
/etc/mknod disk2 c 33 40 4 02010 0 2340000 0 0 011 0
/etc/mknod disk3 c 33 41 4 02020 0 2340000 0 0 01 0
/etc/mknod disk4 c 33 42 4 02030 0 2340000 0 0 01 0
/etc/mknod disk5 c 33 43 4 02030 0 2340000 0 0 011 0
```

The last line in the above example represents a RAID-3 slice on ring 2, node 3, FCN channel/loop 0, involving Loop-IDs/units 011-015.

Configure the `/dev/dsk` entries:

```
/etc/mknod disk1 b 34 39 0 0 /dev/xdd/disk1
/etc/mknod disk2 b 34 40 0 0 /dev/xdd/disk2
/etc/mknod disk3 b 34 41 0 0 /dev/xdd/disk3
/etc/mknod disk4 b 34 42 0 0 /dev/xdd/disk4
/etc/mknod disk5 b 34 43 0 0 /dev/xdd/disk5
```

The following example shows the output from a `ddstat -m *` command:

```
/etc/mknod disk1 b 34 39 0 0 /dev/xdd/disk1
        /etc/mknod /dev/xdd/disk1 c 33 39 4 02010 0 2340000 0 0 01 0
/etc/mknod disk2 b 34 40 0 0 /dev/xdd/disk2
        /etc/mknod /dev/xdd/disk2 c 33 40 4 02010 0 2340000 0 0 011 0
/etc/mknod disk3 b 34 41 0 0 /dev/xdd/disk3
        /etc/mknod /dev/xdd/disk3 c 33 41 4 02020 0 2340000 0 0 01 0
/etc/mknod disk4 b 34 42 0 0 /dev/xdd/disk4
        /etc/mknod /dev/xdd/disk4 c 33 42 4 02030 0 2340000 0 0 01 0
/etc/mknod disk5 b 34 43 0 0 /dev/xdd/disk5
        /etc/mknod /dev/xdd/disk5 c 33 43 4 02030 0 2340000 0 0 011 0
```

After performing the above steps, the array is ready for I/O:

```
# /etc/mkfs -q -A96 /dev/dsk/disk1

/etc/mkfs: *** NC1FS filesystem initialized on /dev/dsk/disk1 ***
        *** Lower security level = 0   Upper security level = 0
        *** Valid security compartments = 0
                none
        *** Big file: 32768 bytes  big allocation unit: 96 blocks
        *** Allocation strategy: Round robin all files(rrf)
        *** 1 partitions / 9360000 total blocks / 9351704 free
        *** 131072 total inodes / 131068 free
```

```
*** 1 primary partitions / 4 blocks per alloc. unit
*** File system partitions:
        part 0: primary blocks 0 - 9359999  on device disk1
*** Panic on error option selected
```

2.5.2 Replacing a failing spindle

Use the following procedure to replace a failing spindle when unrecovered errors are occurring.

1. Determine the failing spindle from the failing spindle mask provided in the `errpt` output or system console log. See "Spindle to unit number mapping", Table 1.

Table 1. Spindle to unit number mapping

Unit	Spindle	Spindle mask
0?1	4	020 (parity)
0?2	3	010
0?3	2	004
0?4	1	002
0?5	0	001

2. Disable the spindle:

   `xdms -a disable` *iopath.unit-spindle*

   For example:

   ```
   xdms -a disable 02010.1-3 # disable spindle 3
   ```

   This step, disabling the spindle, may take place automatically, depending on the type of errors encountered.

3. Spin down the spindle:

   `xdms -a spindown` *iopath.unit-spindle*

If you try to spin down a drive that is not disabled, the next I/O to that array/spindle will cause the array to spin up as part of normal error recovery.

4. Replace the failing spindle.

   Note: Pulling or insertion of a spindle will cause the FCN software to re-initialize the DSF. This DSF initialization may take a few minutes and will affect not just the DSF containing the pulled/pushed spindle (which will suspend I/O to other drives in the DSF), but will affect other DSFs on the same daisy-chained channel. Some error/console message may appear at this time.

   Before reconstructing the array, check that the serial number of the drive/unit (on the I/O path) is correct:

   `xdms -a info` *iopath.unit-spindle*

5. Reconstruct the spindle.

   The replacement drive that is to be used to reconstruct the full 4+1 array does not need to be initialized.

   Execute the following command:

   `xdms -a reconstruct` *iopath.unit*

   A reconstruct can take 1.5 hours, minimum, depending on other I/O to the same disk array or other I/O to the same DSF. Some reconstructs have taken over 6 hours to complete, when running heavy I/O to the same array (for example, by using `fstest`(8)).

   A message appears on the console (and in the `ion_syslog.info`) when the reconstruct is complete:

```
10/16/97 17:39:17 NOTICE sdisk_admin_r3.c line 774
Array Reconstruction is complete on FC Loop 0 Target 1
```

2.5.3 Converting RAID members to single spindles

The spindle can be initialized in a slot that would normally be part of an array, although this is not required to perform a reconstruct.

The following example initializes one member of a RAID to a single spindle:

```
# ./xdms -ainit -m 1 4044.21

4044.21 appears to be a member of a RAID-3S 4+1    .
To init JUST THIS MEMBER to Single Spindle enter "y"
To init ALL RAID MEMBERS to Single Spindle enter "a"

Please be sure all activity to device is idled before continuing with init.
Continue with Init to Single Spindle now? (y, a, or n)
y
xdms: Initialized iopath 4044.21 as Single Spindle
```

Note: This command is different from the `xdms`(8) command to init the full array, which uses `-m 35`.

To turn a disk array back into 5 single disks initialize each spindle of the array or follow the example below.

The following example initializes all RAID members to a single spindle:

```
# ./xdms -ainit -m 1 4044.21

4044.21 appears to be a member of a RAID-3S 4+1    .
To init JUST THIS MEMBER to Single Spindle enter "y"
To init ALL RAID MEMBERS to Single Spindle enter "a"

Please be sure all activity to device is idled before continuing with init.
Continue with Init to Single Spindle now? (y, a, or n)
a
xdms: Initialized iopath 4044 unit 21
xdms: Initialized iopath 4044 unit 22
xdms: Initialized iopath 4044 unit 23
xdms: Initialized iopath 4044 unit 24
xdms: Initialized iopath 4044 unit 25
```

Note: If the `-z` option is used with the `-a init` action and the device mode is a `-m 1` only the RAID member specified will be initialized. The net affect of the `-z` option is the same as in previous example.

2.5.4 Software Limitations

The following limitations are in effect when configuring and restoring disk arrays.

- You cannot mix arrays with single drives on the same channel/loop. The primary path and the alternate path are considered to be the same channel.

  For example you can configure multiple DA-308 arrays on a single FCN channel, but you cannot configure a DA-308 and a DD-308 on the same channel. Similarly, you can configure a primary and alternate path with all DA-308 or all DD-308 drives, but you cannot configure a primary and alternate path that mix DA-308s with DD-308s.

- Executing a Control-C after issuing a `xdms -a reconstruct` command does not halt the reconstruct. After the initial reconstruct request, the FCN performs the reconstruct. However, `xdms`(8) can resume monitoring status of the reconstruct by reissuing the reconstruct or issuing an info request on the array device. The info request will indicate the percent of reconstruct complete if there is a reconstruct currently in process.

- The addresses of array members cannot be changed. If the array was initialized as targets/units 1–5, it cannot be moved to 9–13 (011–015), 17–21 (021–025), etc. It can be moved to another channel as long as the target addresses are the same. When you want to change the addresses, you must execute `xdms -a init` and remake the `/dev/xdd` nodes.

2.6 File system initialization

Use the `mkfs`(8) command to initialize file systems. The `mkfs` command builds the file system with a boot block, a super block, a root inode and a bit map of free blocks. By default, it also performs a surface check and zeroes the disk data blocks before initialization. When the UNICOS multilevel security (MLS) feature is enabled, `mkfs` provides the new file system with minimum and maximum security levels and authorized compartments. See Chapter 8, page 145, for more information on using `mkfs` on a UNICOS MLS system.

`NC1FS` file systems include a secondary allocation area. The secondary allocation area is a means of segmenting file data by usage. The secondary allocation area contains only user file data and may be allocated in different allocation units than primary allocation areas. If a secondary allocation area exists, default allocation of user data will occur there once a file has grown to "big file" size, as defined in the `sys/param.h` file or with the `mkfs`(8) or `setfs`(8) commands.

For a complete list of the options available with `mkfs`, see the `mkfs`(8) man page. For further information on allocation of inode regions, see Section 2.8, page 45.

2.7 Inode allocation strategies

You can specify inode allocation strategies at system configuration time by using the `mkfs`(8) command with the `-a` option, as follows:

```
mkfs -a strategy
```

You can also change the allocation style after configuration is complete by using the `setfs`(8) command.

The allocation strategies are as follows:

Strategy	Description
`rrf`	Round-robin files (default)
`rrd1`	Round-robin first-level directories
`rrda`	Round-robin all directories

Round-robin allocation is a process of allocating files and directories to partitions in sequence. When a file or directory is created, it is assigned to the next partition in sequence after the partition to which the previous file or directory was assigned. When a file or directory is created in the last partition, the next partition in sequence is partition 0.

Each allocation strategy specifies the preferred location for data blocks, directory blocks, and inodes; however, if the preferred locations are full, the kernel places these items wherever possible in the file system.

If you use the logical device cache with your file systems, you can reduce system overhead by using the `mkfs` command to make the allocation unit equal to the logical device cache block size or a multiple of the same. Alternatively, you can use partition cache with some or all of the partitions of a file system to reduce system overhead.

You can improve performance on individual jobs and by pre-allocating space using the `setf`(1) command, the `assign`(1) command, or the `ialloc`(2) system call. These commands and system call perform all the allocation in one step and allocate storage in contiguous or nearly contiguous areas. Additionally, you can use `setf`, `assign`, and `ialloc` to force allocation from particular file system slices, so that different files can be placed on different physical disk devices.

2.7.1 `rrf` allocation

The default allocation strategy, `rrf`, is the recommended strategy. The inodes and directories are assigned to the first 25% of partition 0 whenever possible. When new files are created, the data blocks are round-robined. As data blocks are added to a file, they remain on the same partition as the preceding data blocks whenever possible. This allocation style provides good recovery and a good distribution of file blocks for performance.

2.7.2 `rrd1` allocation

The `rrd1` allocation strategy round-robins all first-level directories among the available partitions. (*First level directories* are directories defined in the root (top-level) directory of the file system.) The inode and the directory data blocks for each directory are in the assigned partition. Within a first-level directory, all files and subdirectories remain in the same partition as the first-level directory whenever possible.

This allocation scheme allows for better recovery and performance by limiting the impact of system crashes. For example, if a first-level directory corresponds to a single user, as in the typical home directory case, then all the files for one user are in the same partition and on the same disk. If one disk of a multidisk crashes, only a subset of all the users are affected. However, the top-level directory data blocks and inode are on partition 0. If the first disk is lost, the entire file system must be restored as before.

The disadvantage to this allocation strategy is that there can be severe performance penalty for assigning all of the files for one user on the same disk. If, for example, you use the `cp`(1) command to copy a file within the directory to another file within that directory, you would be reading from and writing to files on the same disk drive.

2.7.3 `rrda` allocation

The `rrda` allocation strategy round-robins each new directory that is created. The disk files within a directory have their inodes and data blocks in the same partition as the parent directory.

This strategy produces bad performance and recovery. If one disk of a three disk site were lost, even though the root directory is in partition 0, one third of its directory entries will have been on the bad disk. Of the remaining two thirds, the files within those directories will be recoverable, but one third of the subdirectories will be unrecoverable. The more complex the directory tree is,

the worse this situation is. Using the `fsck`(8) command on this file system is not always productive.

The same performance problems of the `rrd1` allocation also affect the `rrda` strategy. If you copy one file to another in one directory, the command is always doing I/O on one drive only. Also, the data blocks for files within a directory are not distributed among the disks in the system.

2.8 Inode region allocation

Before creating a file system, you must understand how inode regions are allocated and how the `mkfs -i` option can be used. Up to 64 partitions can exist in an `NC1FS` file system. Each partition can have up to four inode regions. The size of an inode region is limited by the number of bits contained in the first block that is used as a bit map for free inodes. For disks with 512-word sectors, each inode region can contain at most 32,768 inodes. The size of an inode region is defined when the region is created and cannot later expand or contract.

When the `mkfs`(8) command creates a file system, exactly one inode region is created in each partition. When an inode region in a partition is full, a new one is created in that partition unless there are already four regions. Then, an inode region in the next partition is tried. If all four regions in all partitions are full, no more files can be created until some files (and inodes) are released.

The `mkfs -i` option specifies the desired ratio of blocks to inodes; the default value is 4.

Consider some examples in which the file systems have been created with the `mkfs -i 2` command and they contain 100,000 blocks. The `-i 2` option and argument indicate that 50,000 inodes (100,000/2) should be created. It is assumed that the default `rrf` allocation strategy is used in the following examples.

File system 1 contains one partition. Because one inode region is created and an inode region can contain at most 32,768 inodes, only 32,768 inodes are created for this file system. If all 32,768 are used, the file system will create another inode region if there is room on the device. A maximum of four inode regions can be created per partition. Therefore, at most, four inode regions can exist on file system 1.

File system 2 contains two partitions. The inode region on the first partition contains 32,768 inodes. The inode region on the second partition contains 17,232 inodes, so there are 50,000 inodes available in the file system. If the inode region in the first partition fills, a second, then a third, and finally a fourth are

created before the inode region in the second partition is used. An exception to this rule would occur if there were not enough contiguous blocks to create additional regions in the first partition. In that case, the kernel would switch immediately to the inode region in the second partition.

File system 3 contains three partitions. The inode region on the first partition contains 32,768 inodes, the region on the second partition contains 17,232 inodes, and the region on the third partition contains 16 inodes. (One block contains 16 inodes; this is the smallest region that is created.) File system 3 contains 50,016 inodes. As with file system 2, the kernel will create more regions in the first partition (up to four total regions) before using the inode region in the second partition. The kernel attempts to create all four regions in the second partition before using the inode region in the third partition.

The size of the inode regions created in these example file systems when the first region is full do not depend on what had been specified with the `mkfs -i` option. Rather, the size depends on the ratio of the number of blocks actually in use to the number of inodes (used and unused) in all of the existing inode regions.

For example, the first inode region fills up in file system 2 and 80,000 blocks are used in the file system. Because there is already room for 50,000 inodes (in the regions in both partitions), the blocks-to-inode ratio is 80,000/50,000 = 1 (integer division). Therefore, a blocks-to-inode ratio of 1 is used for the next region. Because there are 20,000 free blocks in the file system, room is allocated for 20,000/1 = 20,000 inodes in the second inode region in the first partition.

2.9 Labeling a file system

A label on a newly created file system should be created through the `labelit`(8) command. It is optional, but when a label is not given to a file system, a warning message is issued in the following format when the file system is mounted; *mntpt* is the mount point of the file system:

`mount: warning: <> mounted as </`*mntpt*`>`

The basic format of the `labelit` command is as follows:

```
/etc/labelit device [fsname volname]
```

The *device* is the name of the device file that you want to label. The variables *fsname* and *volname* specify the file system name and volume name to be written

in the label. The `labelit` command takes several options. See the `labelit`(8) man page for a description of the options to `labelit`.

The following example shows how to label the device `/dev/dsk/usr` as the file system `usr` with a volume name of `usr-6.1`. After the file system is labeled, a `sync`(1) command is issued.

```
/etc/labelit /dev/dsk/usr usr usr-6.1sync
```

2.10 Mirrored file systems

A mirrored file system resides in two or more component devices, each of which contains a full copy of the mirrored information. A write operation to the mirrored device causes separate write operations to be performed on each of the components. A read operation may be performed on any of the component devices.

The multiple write operations provide for redundancy and for recovery in the case of a failure by a single component. The multiple read paths provide more options for scheduling the read operation, leading to faster completion

2.10.1 Creating a mirrored file system

Example 1: The following example creates a mirrored file system consisting of identical areas on two different physical disks.

```
# for each component
/etc/mknod /dev/pdd/m0 c dev_pdd 100 10 0130 0 119692 0 0 0
/etc/mknod /dev/pdd/m1 c dev_pdd 101 10 0132 0 119692 0 0 0
# grouping the physical devices into a logical device
/etc/mknod /dev/ldd/log_mir L /dev/pdd/m0 /dev/pdd/m1
# making the mirror
/etc/mknod /dev/mdd/mir c dev_mdd 100 0 07777 /dev/ldd/log_mir
# making the fs
/etc/mknod /dev/dsk/fs_mir b dev_ldd 100 0 0 /dev/mdd/mir
/etc/mkfs  /dev/dsk/fs_mir
```

Example 2: The following example creates a mirrored file system consisting of a slice of the SSD paired with a slice of disk. This file system is intended to create an "all-cached-spindle" in which you perform read operations from the SSD and write operations to both the SSD and the non-volatile disk; this combines the speed of SSD with the permanence of a non-volatile disk. Because of the nature

of the default path through system startup, it is important that the SSD component be declared second in the mirror.

```
# for each component
/etc/mknod /dev/pdd/acs_disk c dev_pdd 100 10 0130 0 4600 0 0 0
/etc/mknod /dev/pdd/acs_ssd c dev_ssdd 10 10 0 4600
# for the logical device
/etc/mknod /dev/ldd/acs_ldd L /dev/pdd/acs_disk /dev/pdd/acs_ssd
# for the mirror
/etc/mknod /dev/mdd/acs_mir c dev_mdd 100 0 037 /dev/ldd/acs_ldd
# for the file system
/etc/mknod /dev/dsk/acs b 100 0 0 /dev/mdd/acs_mir
/etc/mkfs /dev/dsk/acs
```

2.10.2 Configuring a mirrored device

While the mountable file systems are usually found in the `/dev/dsk` directory, the mirrored devices are usually in the `/dev/mdd` directory. At this level, before it is a file system, the mirrored device may be configured to enable reading and/or writing on selected components of the device.

For an open mirrored device that is known to the `mdd` driver in the kernel, the configuration is carried in a kernel table. For a closed mirrored device, the configuration is in one of the device-dependent fields of the inode.

Just as a standard file is characterized by a read/write/execute mode, each component of a mirrored device can be summarized by `r-w-x` bits, or by a single octal digit. The rightmost field describes the state of the first component.

The meaning of the bits are as follows:

Bit	Meaning
04 (r-bit)	This component is enabled for reading
02 (w-bit)	This component is enabled for writing
01 (x-bit)	This component is undamaged and physically available

You can use the `/etc/mddconf` program to display or change the configuration. If you specify the `-p` option with a new configuration, the value is written to the permanent disk-resident inode. Following are four sample configurations.

Example 1: This example shows a two-component mirrored file system that is enabled in all components for reading and for writing:

```
# /etc/mddconf /dev/mdd/mir
      device name                        r/w mode
-------------------------      ----------------------
/dev/mdd/mir                   ------------------rwxrwx
```

Example 2: This example shows a two-component mirrored file system that reads from a single device but writes to both:

```
# /etc/mddconf /dev/mdd/acs_mir
      device name                        r/w mode
-------------------------      ----------------------
/dev/mdd/mir                   ------------------rwx-wx
```

No matter what is written to the following file system, the data that is read will not change:

```
# /etc/mddconf /dev/mdd/hard_head
      device name                        r/w mode
-------------------------      ----------------------
/dev/mdd/mir                   -------------------wxr-x
```

Example 3: This example shows a two-component mirror; both components are being synchronized with the data in the read-enabled component:

```
# /etc/mddconf /dev/mdd/new
      device name                        r/w mode
-------------------------      ----------------------
/dev/mdd/mir                   ------------------wx-rwx
```

Though the previous examples show a file system as composed of a single logical device that is a mirror of physical devices, other configurations are possible.

2.10.3 Default configuration

In software provided by Cray Research, the default configuration for a mirrored device is `rwx` in each component. For file systems such as the `acs` file system described in one of the previous examples, the default configuration may be changed by a `mirror=` entry in the options field of the `/etc/fstab`

description for the file system. For the `acs` example with an SSD in the second mirrored component, the `/etc/fstab` description would be:

```
/dev/dsk/acs  /mount_point NC1FS  rw,mirror=(acs_mir:073)
```

The mirror configurations are specified within the parentheses. The first field is the base name of the mirrored device. The second field, separated from the first by a colon, is the configuration specification given numerically. A leading zero is recommended so that the configuration will be interpreted in octal.

If there is more than one mirrored device in a file system, the semicolon separates entries, as in the following example:

```
mirror=(zero:0337;one:0373;two:0733)
```

If the existing configuration of a mirror does not contain an exercise (`x`) bit in each component, any configuration found in `/etc/fstab` is ignored. The tuned configuration used by Cray Research software will enable read and write operations in every component that is marked with the `x` bit.

2.10.4 Mirrored devices during startup

If a file system containing a mirrored device has been cleanly dismounted, it may be remounted without examination. If a mirrored file system is in an unknown state as the result of a system crash, it must be handled with special processing to prevent different components from being matched in a mirror. There are three programs that help in bringing up a mirrored device.

The first program, `/etc/mdd_pre`, runs before the `fsck`(8) utility. If the file system needs to be checked, it configures all mirrors to a single component read/write configuration; for example, `0117`, `0171`, or `0711`. The configuration permissions ensure that `fsck` sees a consistent set of data.

The second program, `/etc/fsck`, is used by all file systems, mirrored or not.

The third program, `/etc/mdd_post`, is used after `fsck`. Though the file system may be mounted and used as soon as `fsck` completes, the `mdd_post` program performs the following steps to finish bringing up the mirrored parts of the file system:

1. Reconfigures the mirror to a wide-write state. In this state, all read operations are completed by the mirrored component used during the `fsck` step, but the write operations go to every component.
2. Executes `/etc/mddcopy`. This program uses `ioctl` calls at the `mdd` driver level to copy identical information to all components of the mirror.

3. Reconfigures the mirror to the tuned state.

The `mdd_pre` and `mdd_post` programs should be placed in the start-up scripts before and after the `mfsck`(8) command. Experience in the field suggests that `mdd_pre` fits in `/etc/inittab` and `mdd_post` fits in `rc.pst`.

2.10.5 Manual startup of mirrored file systems

It is possible to bring up a mirrored file system without using the `/etc/mdd_pre` and `/etc/mdd_post` programs. This technique might be valuable when there are more recently updated mirror components than the one chosen by `mdd_pre`.

First, use the `/etc/mddconf` command to restrict the I/O to a single component of each mirrored device in the file system. Then run the `fsck`(8) utility. You can repeat the `mddconf` and `fsck` step using the `-n` option on `fsck` to survey the status of the file system.

When `fsck` has run to completion and performed the necessary corrections, the file system may be mounted. You can perform the `mdd_post` processing manually, as outlined above, or you can start the `/etc/mdd_post` program with a single file system as a parameter.

2.11 Performance considerations

The following sections discuss device and file system configuration that can affect the performance of the system:

- Logical device cache
- System buffer cache
- Using SSD as a file system
- Secondary data segments (SDS)
- File system placement

2.11.1 Logical device cache

The logical device cache provides an excellent means of utilizing the SSD, by providing the capability to assign SSD cache to specified logical devices. The benefit of this type of cache is that predictable and high-speed I/O can be assigned to specific file systems, based on the needs of a particular group or

individual and at the discretion of the administrator. It is suggested that you assign cache to `/tmp` and other heavily used file systems.

Note: CRAY J90 systems only support RAM-based logical device cache.

You can also allocate cache at the partition level, with the `pcache`(8) command. You cannot use partition cache and logical device cache on the same file system.

Cache for logical devices is assigned by the `ldcache`(8) command or with the installation and configuration menu system. Cache may be changed dynamically.

The cache for a logical device is specified as a number of units and a count of 4096-byte blocks per unit. The ability to dynamically alter the cache for logical devices allows you to easily tailor the cache for differing job mixes.

The relationship between number of cache units and the size of each cache unit can dramatically affect the throughput of the cache. For extremely sparse, random I/O, a greater number of units reduces the wait time for a cache unit to become available and increases the probability that data will remain in the cache for a reasonable amount of time. Larger cache units provide higher I/O rates for sequential data access and lower system overhead.

There are some applications for which you might want to bypass the logical device cache when performing I/O on particular files. For example, applications that the perform a large amount of I/O and that access more data than will fit in a cache can significantly decrease system performance because of thrashing in the logical device cache. To bypass the logical device cache, set the `O_LDRAW` flag and the `O_RAW` flag with the `open` system call, as described on the `open`(2) man page.

Each cache unit configured requires a header, which is maintained in main memory. At boot time, you can change the number of logical device cache headers allocated by editing the `NLDCH` parameter in the configuration specification language (CSL) parameter file. If `NLDCH` is not specified in the CSL parameter file, the value specified in `config.h` is used.

The `ldcache`(8) command lets you set logical device cache configuration and display cache statistics. Logical device cache configuration use is restricted to the super user, but users can display cache information.

2.11.1.1 Setting cache configuration

To set cache configuration, use the following command line:

```
ldcache -l dev -n units [-s size] [-t type]
```

The options perform the following functions:

`-l` *dev* — Specifies a file system name, or the name or number of a logical device. If *dev* is a device name, it must begin with `/`.

`-n` *units* — Specifies the number of cache *units* to be assigned. If 0, all cache for the device is released.

`-s` *size* — Specifies the *size* of each cache unit in 4096 byte blocks. This option is meaningful only when the `-n` option is nonzero. *size* is typically chosen to be a multiple of the track size of the disk on which the file system resides.

`-t` *type* — Specifies the memory *type* for cache; *type* can be `SSD`, or `MEM` (main memory). The default is `SSD`. This option is meaningful only when specified with the `-n` option.

2.11.1.2 Displaying cache statistics

To display cache statistics, you can use either of the following command lines:

```
ldcache -a
   ldcache -l dev -r rate
```

The `-a` option displays devices that have any read or write operations, even though no cache is attached. The `-l` option functions as it does in the configuration command line. The `-r` option specifies the refresh rate for detailed display; the default refresh rate is 1 second.

If you specify no options or arguments, `ldcache` displays information about all devices with cache in the format of the following example:

```
T  units    size       hits     misses     hit rate      name

S  200       252       66196      361     99.457608     /dev/dsk/drop
S  800        84     1186345      897     99.924447     /dev/dsk/tmp
M  300        42     2250878     4248     99.811629     /dev/dsk/root
S  300        42      618508     1789     99.711590     /dev/dsk/usr
S  100        42      289529     4267     98.547632     /dev/dsk/slash_a
S  100        42      117462     6167     95.011688     /dev/dsk/slash_b
S  100        42      163790     6207     96.348759     /dev/dsk/slash_c
```

In the `T` (memory type) column, `S` stands for SSD and `M` stands for main memory. If you use the `-l` option to specify a specific device, `ldcache` displays information about that device at the refresh rate specified (with the `-r` option) or at the default refresh rate of 1 second.

For example, to receive information on `/dev/dsk/root`, you would specify the following command line and get the output shown:

```
cray$ ldcache -l /dev/dsk/root

/dev/dsk/root                  Tue May 24 09:26:51 1988

                             Read Data            Write Data

Blocks transferred:          23549                4864
Average request size:        2 blks               1 blks
Lst transfer rate:           1.067367 Mbs         0.044813 Mbs
Max transfer rate:           10.751468 Mbs        1.870442 Mbs
Cache hits:                  12025                4864
Cache misses:                35                   0
Cache hit rate:              99.709784            100.000000
```

You can use the following commands when viewing the display produced by the `-l` option of `ldcache`:

Command	Description
`n`	Goes to next device with cache attached
`+`	Increases refresh interval by 1 second
`-`	Decreases refresh interval by 1 second
`c`	Clears counters to 0

2.11.1.3 Aging and threshold parameters of `ldcache`

For large applications that perform frequent write requests, you may want to consider using the following options of the `ldcache`(8) command:

Option	Description
`-x` *max*, *min*	Specifies, in seconds, aging parameters for units in the logical device cache. When the age of any dirty cache unit exceeds the *max* value, the kernel

automatically flushes all dirty units older than the *min* value.

`-h` *high*, *low*
: Specifies threshold values for the dirty units in the logical device cache. The *high* value specifies the maximum number of dirty units that can be in cache at one time. If the number of dirty units equals the *high* value, requests to dirty more cache units are put to sleep until the number of dirty units falls below this threshold value. When the *low* threshold value is exceeded, `ldcache` starts flushing the oldest dirty units until the *low* threshold is no longer exceeded.

These options implement a *trickle sync* mechanism, which lessens periods of intense disk activity caused by an `ldsync`(8) command.

The `-h` parameter of `ldcache` provides a relatively stable read cache within a larger read/write cache. The `-h` *max* parameter limits the number of dirty units that can be in the cache. The difference between this value and the size of the cache is the size of the read cache. These extra units are, in effect, reserved for read requests, which are typically more likely to be reused.

To pick effective `-h` parameter values for `ldcache`, you need to determine the relative amounts of the different types of I/O requests that are made to each `ldcache` file system. For example, if most of the requests are read requests, the `-h` parameter is unnecessary. If most of the requests are write requests and the amount of data written over a relatively short period of time is larger than the cache, set the `-h` parameter so that there is enough read cache left over to handle the read I/O requests of the applications.

When you use the trickle sync option, you may want to disable the `LDSYNCTM` parameter. To do this, manually set `ldsynctm` in the `/etc/inittab` file or change `LDSYNCTM` in the `/usr/src/cmd/init/conf.c` file to a value greater than 1000000 and rebuild `/etc/init`.

2.11.2 System buffer cache

The amount of system buffer cache configured affects the performance of a system. A cache that is too small degrades system performance; too large a cache wastes memory that could be of use elsewhere. For I/O-intensive jobs that do not use raw I/O, a larger system buffer cache can be used to increase throughput. Experimentation can help determine the optimal number of

buffers. System buffers are allocated at boot time in 512-word blocks and use up part of main memory.

You can change the system buffer allocation by using the following menu of the installation and configuration menu system:

```
Configure System
    ->Kernel configuration
        ->Table size parameters
```

At boot time, you can change the number of system buffers allocated by editing the `NBUF` parameter in the CSL parameter file.

The effectiveness of your system buffer cache and I/O in general can be monitored by using the `sar`(8) command.

2.11.3 Using SSD as a file system

SSD can be used as a logical partition. The SSD can be configured as a logical device and mounted individually, or grouped with other logical slices and mounted as a logical device. System performance can be improved by using the SSD as a logical device where access time is critical to system performance (for example, for file systems such as `/tmp`, `/bin`, or `/lib`). File systems that are used heavily can be mounted on the SSD to increase throughput and reduce I/O wait time.

As a file system, the SSD can be configured in the same manner as disk devices; that is, it is configured as one or more slices each having a starting block number and number of blocks.

One or more of these slices may be used as logical devices upon which file systems can be built. In addition, SSD slices can be combined with disk slices to form logical devices.

Note: SSD data can be lost across system power failures. This must be taken into account when deciding whether or not a file system should span both disks and SSD. It is recommended that file systems reside completely on disk and that logical device cache blocks be assigned to the SSD (see Section 2.11.1, page 51.

2.11.4 Secondary data segments (SDS)

Secondary data segments (SDS) is a feature that allows a part of the SSD to be used as extended memory. This area must be defined in the parameter file.

A user can specify that a file resides on the SDS by using the `assign`(1) command. Users then make requests to either expand or contract their SDS field length. SDS is automatically released when the owning process terminates.

The system maintains a base and limit address for the SDS area of each process. All I/O requests to SDS are relative to address `0`. The simple mapping scheme allows use of a short-circuited path to process I/O requests to SDS, providing transfer rates up to 10 times higher than that of SSD file systems. The system calls `ssbreak`(2), `ssread`, and `sswrite` (see `ssread`(2)) are discussed in *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012.

The status of processes using SDS space can be determined with the `sdss`(1) command.

The UNICOS operating system also supports direct data transfers between SDS and disk files through the backdoor or sidedoor channel.

Allocation of a file on SDS is accomplished by opening a disk file with the `O_SSD` flag set (see `open`(2)). All read/write addresses are then treated as relative to SDS.

Fortran I/O library support allows particular files to be assigned to SDS in a program-transparent manner. See `assign`(1) and `env`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011.

For a discussion of SDS management in the batch environment, refer to the *NQE Administration*, Cray Research publication SG–2150.

2.11.5 File system placement

The `/usr` partitions should be on a different disk drive than the `root` (`/`) partition(s) to reduce disk I/O contention. In addition, system core dumps should be copied to another partition so the root partition is not filled.

The `/usr/spool` and `/usr/adm` directories should be on separate file systems from the `/usr` file system so that Network Queuing System (NQS), and accounting are not interrupted if `/usr` becomes filled.

User directories should not be located on either the `root` (`/`) or `/usr` partition to prevent users from filling these partitions.