File Systems [5]

5.1 UNICOS file systems

All files that are accessible from within the UNICOS system are organized into *file systems*. File systems store data in formats that the operating system can read and write. This chapter describes how to plan, configure, create, and monitor UNICOS file systems. As a system administrator, you must do the following:

- Plan the file systems
- Configure the file systems
- Create the file systems
- Monitor disk usage to ensure that your users have sufficient free space on their file systems to accomplish their work

No single configuration of available disk drives into file systems and logical devices will prove best for all purposes. Optimizing file system layout is usually an iterative process; make your best attempt, then run it for a while and monitor it for disk use monitoring information (see Section 5.4, page 56). You will adjust your configuration based on information you gather about your users' needs. As the needs of your users change, you will reconfigure your file systems to retain a well-balanced configuration. In the absence of a set of absolute rules, the facts and guidelines presented in this chapter will prove useful when you decide on a file system plan for your system.

Note: Although all UNICOS file systems have some common aspects, file system creation and organization varies on Cray Research systems. If you have Cray Research systems that are not CRAY J90 systems, see *General UNICOS System Administration*, Cray Research publication SG–2301, to determine differences in file systems and how to configure them.

5.2 Related file systems documentation

The following Cray Research publications contain information related to this section:

- *General UNICOS System Administration*, Cray Research publication SG–2301
- *UNICOS Resource Administration*, Cray Research publication SG–2302

- *UNICOS Installation Guide for CRAY J90 Model V based Systems*, Cray Research publication SG–5271
- *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011: `df`(1), `du`(1), `mkdir`(1), and `rm`(1) man pages
- *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014: `dir`(5), `dsk`(4), `fs`(5), `fstab`(5), `inode`(5), `ldd`(4), `mnttab`(5), and `pdd`(4) man pages
- *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022: `ddstat`(8), `diskusg`(8), `dmap`(8), `econfig`(8), `fsck`(8), `fsmap`(8), `fuser`(8), `labelit`(8), `mkfs`(8), `mknod`(8), `mount`(8), `stor`(8), and `umount`(8) man pages

5.3 An overview of file systems

A *file system* is a group of addressable disk blocks used to store UNICOS directories and files. A file system can either be mounted (accessible to users) or unmounted (unavailable to users). The system mount table records which file systems are currently mounted. The mount table is named in `/etc/mnttab`.

File systems have an inverted tree structure, with a file at each node of the tree. A base file system named `/` or `root` always exits. The `root` file system is always available for use and contains required files needed for booting the UNICOS system. When a file system is mounted, it is attached to a mount point (directory), which might be part of another file system. Mounting file systems on each other creates a series of cascading directories below the `root` file system.

To maintain data consistently and correctly, individual files are in **only** one file system. Each file system resides on unique physical locations on a physical disks, and the UNICOS system carefully controls the file systems. This isolation of data prevents security violations and data corruption.

Note: When you are in single-user mode, with only the `root` (`/`) file system available, you must do all editing by using the `ed` editor, because the `vi` editor is located in the `/usr` file system. If you want to use the `vi` editor before going to multiuser mode, you first must check (using `fsck`) and mount the `/usr` file system.

5.3.1 Terminology

This section provides terminology associated with file systems. Everything is viewed by the UNICOS system as a file, whether it is an ASCII file of user data or a physical disk device. The UNICOS system supports five types of files: regular, directory, block special (such as a disk drive), character special (such as a tape drive), and FIFO special. *Regular files* hold user data of various formats. *Directory files* contain the names of "regular" files and other directories, along with their corresponding inode numbers. When block or character special files are accessed, device drivers are invoked that communicate with peripheral devices, such as terminals, printers, and disk drives. FIFO special files, also called *named pipes*, allow unrelated programs to exchange information.

A *physical device* is a tape or disk device. Physical disk devices are read from and written to in units of 512-word (4096-byte) blocks . The smallest unit of I/O disk devices can perform is one block. UNICOS file systems are defined in regions of contiguous blocks called *slices*. File systems can be built on many different slices.

A *partition* is one slice on one physical device.

One or more slices create a *logical device*. Although a logical device appears to be one device, its slices can be located across several physical devices. Logical devices become file systems when the disk is initialized with a file system structure by using the `/etc/mkfs` command.

An *inode* contains information such as permissions and file size for all five types of files.

Regular files are composed of readable characters; these can include data, text, or program files that can be executed.

The following *special files* are not composed of readable data. Instead, they serve as a connection between a path name (such as `/dev/dsk/root`) and the device handling routines in the UNICOS kernel to control I/O to the device.

- *Block special files*: Block special files are used to communicate with file systems. The drivers for these files process data in blocks. Block devices have a minimum transfer unit size of one block (4096 bytes or 512 words). All I/O for CRAY J90 file systems use block special files. You can address block special files and their related devices by using various I/O techniques.
- *Character special files*: The drivers for these files process "raw" data, bypassing UNICOS kernel buffering. Data is transferred directly between the user's memory area and the physical device. UNICOS character special

files are used to support tape and tty connections, among others. You can use character special files and their related devices only for sequential I/O.

All special files have a major and a minor device number associated with it. A *major device number* refers to the type of device. Major device numbers are used as an index into a table of device drivers appropriate for that kind of physical device. These routines open, close, read, write, and control a physical device. A *minor device number* is used by the appropriate driver (determined by the major number) to specify a particular logical disk device, tape drive, or physical device. Minor device numbers range from 0 to 255 and must be unique within the same major number; however, numbers 250 through 255 are reserved for use by the operating system. For example, on CRAY J90 systems, minor number 253 is used for the `ce` partition. For additional information, see *General UNICOS System Administration*, Cray Research publication SG–2301.

All UNICOS special files are located in the `/dev` directory or one of its subdirectories. Your CRAY J90 system is initially supplied with sufficient UNICOS special files for most basic device configurations. You should create additional (block) special files to match your unique file system layout. All special files are created using the `mknod` command.

When a device special file is examined by using an `ls -l` command, the device special file's major and minor numbers, separated by a comma, are displayed where the number of bytes would appear for a regular or directory file.

The following are the directory paths of some UNICOS special files and scripts for file systems:

File	Description
`/dev`	Directory of special files and subdirectories of other special files.
`/dev/dsk`	Directory that contains all block special files that represent logical disk devices for current file system configuration. The major device number is 34 for disk devices. A `b` in the directory permissions field (`ls -l` output) indicates a block special file.

5.3.2 UNICOS file system structure

UNICOS file systems are often stored on several different physical devices. When you configure a file system, you first specify the physical locations that compose the file system. This information is stored in the `/sys/param` file,

and it is written using the menu system. You can store file systems on disk or in random-access memory (RAM), or a combination of both.

The definition of your system's logical and physical disk devices is defined in the `/sys/param` file on the IOS. You must initialize that area of disk, using the `mkfs` command.

The `mkfs` command structures the physical disk area with the following elements:

Element	Description
Super block	Used to store file system size and the number of inodes in the file system, as well as internal parameters such as allocation strategy. It is updated when the `mkfs` or `setfs` command is run. Several copies of the super block are kept for robustness (redundant copies make it easier to recover information if a catastrophic failure occurs). The super block is read into memory when the file system is mounted, and it is flushed to disk when it is modified or when the file system is unmounted.
Inode region	Each file in a mounted file system is identified with a unique pointer called an inode number. The *inode* itself contains file information such as permissions, file size, whether the file is a directory, and so on. The inode region contains a maximum of 32,768 inodes. You can have a maximum of four inode regions per partition.
Dynamic block	A block that contains the file system information that changes during system operation. The dynamic block contains block counts for a specific partition. This information is flushed to disk when the file system is modified, when the file system is unmounted, or when `sync`(2) is executed.

Block allocation bit map	A bit map that controls block allocation across the entire file system.
Map blocks	A bit map of the disk sectors.
Partition data blocks	The disk area for directories and user data.
`setfs`(8) command	This command changes dynamic information in the file system super block without remaking the file system.

5.4 Commands for examining files and file systems

One of the most important responsibilities of the system administrator is to monitor system disk usage and to ensure that the system's users have sufficient free space on their file systems to accomplish their work.

To display information about files and file systems, use the following commands:

Command	Description
`/usr/lib/acct/diskusg`	Summarizes the disk usage on the file system you specify by file ownership and identifies users who are using most of the space on a file system. The `/usr/lib/acct/diskusg -h` command is the preferred command for summarizing disk use; the `-h` option provides headings.
`/etc/econfig`	The `-d` option prints out `mknod` commands to generate file systems. You may want to do this when you first get your system in case you have to manually recreate these nodes.
`/etc/dmap`	Displays information about the configuration of a disk subsystem.
`/etc/bmap`	Displays which file is using the block on a given file system.
`/etc/fsmap`	Displays file system free block layout.
`/bin/df`	Displays the number and percentage of free blocks available for mounted file systems; the `-p` option is particularly useful.

`/bin/du`	Summarizes the disk usage on a file system, by directory structure. The `-s` option provides the total number of disk blocks used under each directory (or file) specified.
`/etc/errpt`	Processes errors report generated by `errdaemon`. This UNICOS command is for disk hardware errors; `errpt -a` produces a detailed list of errors; `errpt -d` *device-type* produces list of errors for the specified device type.
`/etc/mount`	Displays the list of all currently mounted disk files and their mount points when issued without arguments.
`/etc/stor`	Sorts special files by physical device numbers and writes to standard output information about starting and ending disk addresses and sizes.
`/ce/bin/olhpa`	Displays hardware errors by reading the `/usr/adm/errfile` file. The `-d` option lets you view disk errors.
`/bin/fck`	Displays information concerning the names files that are gathered from reading the inode and the address blocks from the block special device in the `/dev/dsk` directory.
`/etc/ddstat`	Displays configuration information about disk type character and block special devices.
`/etc/pddconf`	Controls the state of a disk drive.
`/etc/pddstat`	Displays information from the disk table, which controls disk I/O.

5.5 File system planning

When planning a file system, you must decide which parts of a disk will be used for each file system. This section provides file system minimum size requirements and device recommendations.

First, a UNICOS system administrator must plan which slices of a physical device will be used to make up each file system, as well as which file systems should be striped, if any, and which should be banded. (For information about disk striping, see Section 5.7, page 61, and for information about disk banding,

see Section 5.8, page 61.) You must consider disk capacity and transfer rate, as well as file system size and usage, along with the number of users and types of applications your Cray Research representative installed on your system.

The file systems listed in this are found on most UNICOS systems.

Note: The disk storage discussed is the **minimum** amount of storage required, not the **recommended** amount. The information is provided here to help you plan your file system space needs.

For up-to-date information regarding minimum file system and amount of free blocks needed to install other Cray Research software products, see the *UNICOS Installation Guide for CRAY J90 Model V based Systems*, Cray Research publication SG–5271.

5.5.1 The `root (/)` file system

Size recommendations: You should define a **minimum** region of 110,000 blocks to hold your `root` file system.

If you have them, DD-4, DD-5I, DD-5S, or DD-6S disk drives are the preferred type of drive on which to configure the `root` (`/`) file system, with enhanced serial driver interface (ESDI) drives a second choice.

If possible, the remaining blocks on the same physical device used for your `root` (`/`) file system are good locations for your smaller or lesser used file systems.

5.5.2 The `/usr` file system

Size recommendations: You should define a **minimum** region of 190,000 blocks. The contents of the `/usr/adm` subdirectory tend to grow very large because the UNICOS accounting data is kept here.

Device recommendations: To avoid contention, you should configure the `/usr` file system on a different controller, disk, and IOS than the one on which the `root` (`/`) file system resides.

If you have them, DD-4, DD-5I, DD-5S, or DD-6S disk drives are the preferred type of drive on which to configure these two file systems, with ESDI drives a second choice.

Be sure to size your `/usr` file system to meet the space requirements for any software to be installed later.

5.5.3 The /usr/src file system

Size recommendations: The recommended **minimum** value for CRAY J90 system is 120,000 blocks. This size is sufficient to hold all of the files necessary to relink the UNICOS kernel. You also must allow enough space in your default value to handle additional Cray Research asynchronous products you will load and use (for this information, see your UNICOS installation guide and related errata).

5.5.4 The /tmp file system

Size recommendations: You should define a **minimum** region of 50,000 blocks. You may want to allocate `/tmp` and `/home` in a 2 to 1 ratio (2 blocks `/tmp` per 1 block of `/home`).

Device recommendations: If two or more IOSs are present, to avoid contention, you should configure `/tmp` and `/home` on a different controller, disk, and IOS than the one on which the frequently accessed system file systems and logical devices reside. This file system is best handled by allocating slices from several different disks to compose the logical file system. This disk allocation strategy is called *banding*.

5.5.5 The swap device

Size recommendations: You should configure the `swap` device to be the **minimum** number of blocks, as follows:

Central memory size	Minimum blocks for swap device
256 Mbyte/32 Mwords	187,500 blocks
512 Mbyte/64 Mwords	375,000 blocks
1,024 Mbyte/128 Mwords or larger memory	750,000 blocks

Device recommendations: If possible, put the `swap` device on a separate drive from either the `root` (`/`) or `/usr` file system.

If your system's job mix swaps frequently, you may want to configure your swap device as a striped device. If possible, stripe the `swap` device across two DD-4, DD-5I, DD-5S, or DD-6S disks or across disks attached to separate ESDI controllers. For more information about striping, see Section 5.7, page 61, and *General UNICOS System Administration*, Cray Research publication SG–2301.

5.5.6 The dump device

Size recommendations: The **minimum** size of your `dump` device should be a little larger than the amount of memory you actually want to examine to allow an additional 1200 blocks for a `dump` header. You should start with a minimum of 50,000 blocks for the dump size. A dump entry must be in the logical device portion of the file system section of the `/sys/param` file. Take this into account when the range of memory you select to dump by using the `mfdump` command or the area you desired to dump will be truncated to preserve the dump header when the dump is written. The UNICOS system does not dump onto the swap device.

You cannot stripe the `dump` device because it is not a file system.

5.5.7 The back-up `root` (`/`) and back-up `/usr` file systems

Size recommendations: The backup `root` (`/`) (called `rootb`) and back-up `/usr` (called `usrb`) file systems are equal in size to the original `roota` (`/`) and `/usra` file systems.

Device recommendations: Keep `rootb` and `/usra` file systems on different disk drives, controllers, and IOSs, if possible, from `roota` (`/`) and `usrb` file systems. You also should keep the backup version of a file system on a different drive (and controller and IOS if possible) from your original file system.

To keep the `rootb` file system updated to match the `roota` file system, you can run the `dd` command as a `cron` job. For details, see the `dd` and `cron` man pages.

5.5.8 The `/home` file system

The size and location of your `/home` file system is site specific. A **minimum** of 50,000 blocks is recommended. The file system is used for login account home directories.

5.6 Disk device characteristics

You may define and mount one or more file systems on disk devices. For a table of disk device characteristics, see Appendix D, page 323.

5.7 Disk striping

A striped device can be made up of two or more of the same type of disk drives or can be logically the same type. The number of blocks must be the same in each slice. Several drives are combined together in one logical unit (known by the name of the first slice name), which makes simultaneous I/O operations possible. Slice members of a stripe group must be previously defined in the physical device statement of the configuration specification language (CSL). In addition to physical CSL definition statements in the IOS `/sys/param` file, a special stripe device definition statement also is required to configure a stripe group. For information about using CSL, see Section 5.9, page 61.

Disk striping allows an increase in the amount of data transferred with each I/O operation. In effect, the I/O rate is multiplied by the number of disk devices in the striped group. On baseline systems, however, only `swap` is recommended as a striped disk. Striping is best used only for large I/O moves, such as swapping.

Note: You should not run `ldcache` on a swap file.

5.8 Disk banding

Disk banding is the process of distributing a file system across several disk drives. The physical devices do not have to be of the same type or have their block ranges begin at the same block or be of the same length.

5.9 Configuring your devices and their file system allocation

The system configuration file that configures disks is the `/sys/param` file on the IOS disk drive. The configuration specification language (CSL) is used to define the configuration and parameter settings that are used at boot time. CSL defines the following:

- Number of IOSs
- Mapping IOS channels to specific CPUs
- Physical device attributes and slice layout
- Logical grouping of physical disk slices
- System-defined devices
- Network configuration

Note: If you use the menu system to configure these settings, it will automatically generate the CSL statements in the `/sys/param` file that describe your system configuration.

The remainder of this section provides information and procedures to help you do the following:

- Determine how to configure file systems by using CSL
- Determine the devices that are provided on your system when you receive it and how they are allocated to file systems
- Modify your system configuration
- Create file systems

5.9.1 Network disk array configuration

For information on configuring network disk arrays (HIPPI disks), see *Network Disk Array (HIPPI Disk) Configuration Options and Performance*, Cray Research publication SN–2185.

5.9.2 CSL syntax

Three classes of tokens make up the CSL: identifiers, constants, and operators/separators. White space (horizontal tabs, newlines, carriage returns, and spaces) separate individual tokens.

- An identifier is a sequence of digits and letters that specify either special keywords (such as `CONFIG`) or specific objects (such as a physical device). You can enclose the digits and characters in double quotation marks. The underscore (_) and dash (-) are interpreted as letters. Identifiers consist of letters, numbers, and the - and _ symbols. CSL identifiers are case sensitive. The first character may be any of the valid identifier characters.
  - Reserved disk type identifiers in CSL (descriptive comments are within brackets). `DD3`, `DDESDI`, `RD1`, and `DD4` are not supported disk type identifiers on CRAY J90 systems:

`DD3` (New ESDI)	`DD4` (IPI-2 Sabre-7)
`DDESDI` (Old ESDI)	`RD1` (Removable DD-3 ESDI)
`DDRAM` (RAM device)	`DD5I` (Buffered IPI)
`DDSTR` (Striped device)	`DD5S` (3-Gbyte SCSI drive)

`HD16` (HIPPI disk, 16-Kbyte sector)

`HD32` (HIPPI disk, 32-Kbyte sector)

`HD64` (HIPPI disk, 64-Kbyte sector)

`DD6S` (9-Gbyte SCSI drive)

`DD314` (4-Gbyte drive)

`DD318` (8-Gbyte SCSI drive)

- For a list of additional keyword identifiers that have special meaning, see the *UNICOS Configuration Administrator's Guide*, Cray Research publication SG–2303.

- Constants : All constants are positive integers. If a constant begins with a 0, octal format is assumed; otherwise, decimal format applies. The use of digits 8 or 9 in an octal constant causes an error.
- Operators/separators: { } ; ,
- Comments : Words within the paired /* */ symbols are comments.

5.9.3 Placement of CSL statements

All CSL statements that define the size, location, and other attributes of your UNICOS file systems are found in the `/sys/param` file. The placement and order of your configuration and CSL statements are important.

Note: If you use the menu system to configure these settings, it will automatically generate the CSL statements in the `/sys/param` file that describe your system configuration.

CSL statements must conform to the following requirements:

- All CSL statements must be terminated by a semicolon, and all section definitions must be placed within one pair of braces { } or ("curly brackets").
- The UNICOS system processes CSL statements in order of appearance in the IOS parameter file.
- Your IOS parameter file must begin with the keywords `revision sn` *xxxx*; *xxxx* is your machine's serial number.
- The first section of CSL statements defines IOS information.
- The second section of CSL statements defines the mainframe information.
- The third section of CSL statements defines the UNICOS configuration information.

- The fourth section of CSL statements defines the file systems information, which includes configuration statements for the physical devices, logical devices, and special system devices.
- The fifth section of CSL statements defines the network configuration for your system.
- The file must end with the closing brace.

The following sections describe the parameter file sections; a sample file is included.

5.9.3.1 Revision section

The revision section marks the configuration file with a site-defined name for identification purposes, particularly for programs and other Cray Research products. The revision section is specified in the parameter file by the following statement and is designated as `revision sn` *xxxx*; *xxxx* is your machine's serial number.

5.9.3.2 ios_v section

This statement section sets the number of IOSs configured. Using the UNICOS Installation / Configuration Menu System (ICMS), you specify the characteristics on the

```
Configure System
    ->IOS Configuration
```

submenu.

This section should include the following:

Statement	Description
`cluster` *n*	The *n* argument is the IOS; the first entry should be for cluster 0.
`miop`	Required keyword for the cluster.
`eiop 0`	Required keywords for the cluster.

`eiop` *xx*	Designates the controller numbers for the disks associated with the specific IOS.

5.9.3.3 Mainframe section

The mainframe section defines the number of CPUs, size of the mainframe memory, and channel information. Using the ICMS, you specify the characteristics on the

```
Configure System
    ->Mainframe Hardware Characteristics
```

submenu.

This section should include the following:

Statement	Description
value `cpus`	The number of CPUs.
value units `memory`	The *units* may be either words or Mwords; *value* is typically set to the physical amount of memory in the machine.
`channel` *value*	(See Table 1, page 65) The *value* of the master IOS is always 20 or 020 (octal). The channel numbers of the slave IOSs depend on the CPU to which they are connected. To define the channels that connect to the slave IOSs, use the `channel` keyword.

Table 1. CRAY J90 IOS Channel Values

Processor Module	IOS Channel Value
0	020 022 024 026
1	030 032 034 036
2	040 042 044 046
3	050 052 054 056
4	060 062 064 066
5	070 072 074 076

Processor Module	IOS Channel Value
6	100 102 104 106
7	110 112 114 116

5.9.3.4 UNICOS section

The UNICOS section sets certain tunable parameters in the `var` structure. You set these parameters in the

```
Configure System
    ->UNICOS Kernel Configuration
```

submenu. For more information on this topic, you should read *General UNICOS System Administration*, Cray Research publication SG–2301, to determine what parameters you might want to change.

5.9.3.5 File system section

The file system section includes the following:

- Description of physical devices in the system
- Description of logical devices (device nodes) in the system
- Identification of the root and swap devices

The following sections describe each portion of the file system section.

5.9.3.5.1 Physical device definition

The physical devices are defined in this portion of the file system section with the following syntax; lines that begin with `pdd` define the slices for the device:

```
disk name{type type; iopath { cluster number; eiop number;
   channel number;} unit number;pdd slice{minor number; sector
   measure; length number units;}}
```

You must define each field:

Field	Description
`disk` *name*	The *name* of each device is site-configurable, but it must be unique among all devices. By convention, the format is composed of the disk type, IOS number, controller number, and unit (for example, `S_0200`).
`type` *type*	Defines the disk type (see Table 2, page 67).
`cluster` *number*	Defines the IOS. Specifies the IOS number; no default.
`eiop` *number*	Defines the controller (see Table 2, page 67).
`channel` *number*	Defines the channel number.
`unit` *number*	Defines the unit attached to the controller (see Table 2, page 67).
`pdd` *slice*	Each *slice* name also is site-configurable, but it must be unique among all devices. Use a meaningful naming scheme for your file system slices. You may want to name the slices with a combination of a reference to the file system of which they are a part, and a unique number. Names of slices must be unique and consist of 8 characters or fewer.
`minor` *number*	Must be unique; you should use numbers in ascending order.
`sector` *measure*	For all disks used with CRAY J90 systems, *measure* is the starting block number of the slice.
`length` *number units*	For all disks used with CRAY J90 systems, *number units* is the number of blocks for the slice.

Table 2. Disk device types and their values

Disk type keyword	Disk device type number	`eiop` value range	Total blocks per disk unit	Maximum number of units
`DDESDI`	64	0 - 7	170,100	4 (0 - 3)
`DDLDAS`	66	8	1,269,114	-

Disk type keyword	Disk device type number	`eiop` value range	Total blocks per disk unit	Maximum number of units
`DD3`	65	0-7	334,200	4 (0 - 3)
`DDAS2`	67	8	2,502,000	-
`DD4`	68	10-17	653,000	2 (0 or 1)
`DD5S`	71	20-27	781,000	4 (0 - 3)
`DD5I`	72	30-37	723,000	4 (0 - 3)
`RD1`	69	0-7	334,200	-
`DDRAM`	104			
`DD6S`	74	20-27	2,389,000	0-3
`DD314`	75	20-27	1,102,000	0-3
`DD318`	76	20-27		0-3

Note: For HIPPI disk types `HD16`, `HD32`, and `HD64`, the capacity is not fixed according to the device type; the size depends on the specific disk device model.

5.9.3.5.2 Logical device definition

Device nodes (logical device groups) are defined using this portion of the file system section. A logical device groups one or more previously defined physical device slices. Each file system you configure has a corresponding logical device entry. Logical device entries always follow the complete set of physical device statements in the `/sys/param` file.

Each logical device is defined using the following syntax; lines that begin with `ldd` define the logical device:

`ldd` *name* `{minor` *number*`;` *device slice;* `}`

The fields are defined as follows:

Field	Description
`ldd` *name*	Consists of up to 8 characters. By convention, the name is lowercase and reflects the name of the special file that will be created automatically during UNICOS multiuser startup. Each `ldd` *name* shows up as a file name in the UNICOS

	`/dev/dsk` directory. The `ldd` *name* portion must be unique for each logical device.
`minor` *number*	Must be unique; you should use numbers in ascending order.
device slice	The previously defined physical slice name that describes this logical device.

5.9.3.5.3 Special system devices

This portion of the file system section defines the system devices. The `rootdev` and `swapdev` definitions are required. The `swapdev` is **not** a file system, it is an area of disk reserved for swapping activity.

The `rootdev` and `swapdev` definitions have the following syntax:

```
rootdev is ldd name; swapdev is ldd name;
```

Again, each slice (`ldd` *name*) must be the name of a logical device previously defined in the `/sys/param` file.

5.9.3.5.4 Network section

The network section defines network devices and network parameters. You can configure low- and high-speed network communication devices in the

```
Configure System
    ->UNICOS Kernel Configuration
        ->Communication Channel Configuration
```

menu. The network section includes the following information:

- Descriptions of network parameters
- Descriptions of each specific network device that uses standard templates or customized prototypes
- Customized network device prototypes

The network section is specified in the parameter file by the following statement syntax:

```
{ network parameters network number {            iopath {
  cluster number;                        eiop number;
  channel value;              } }
```

The *network* can be `endev` for Ethernet or `fddev` for FDDI. The channel *value* for Ethernet will always begin with 020, and it also can be 021, 022, and 023, depending on the number of channels. The channel *value* for FDDI will always begin with 040, and it also can be 041, depending on the number of channels.

Each Ethernet and each FDDI connection to your system should have one network statement.

5.10 Checking your disk configuration parameter file

To verify configurations, use either the menu system or the `/etc/econfig` command. If you are using the menu system, you can verify configurations by selecting the

```
Configure System
    ->Disk Configuration
        ->Verify the disk configuration ...
```

menu option.

To verify the configuration manually, check the syntax of CSL by using the following `/etc/econfig` command:

```
# /etc/econfig  your_param_file_name
```

The `/etc/econfig` program accepts only valid CSL statements as input. If you use the `/etc/econfig` command, you should use it before booting a new configuration to prevent receiving errors during CSL processing.

To generate the `mknod` commands from your parameter file, use the following syntax:

```
# /etc/econfig -d your_param_file_name > /dev/mkdev.sh
```

Remove the existing devices by using the following commands:

```
# cd /dev
# rm dsk/* pdd/* mdd/* sdd/* ldd/*
```

Generate the new device definitions by using the following commands:

```
# chmod 755 /dev/mkdev.sh
# cd /dev
# ./mkdev.sh
```

A sample CRAY J90 `/sys/param` configuration file follows.

```
revision "SN9003.20";

ios_e {
/* SN9003 - param.ios_e - Edition 3 [Wed Aug 20 14:37:31 CDT 1997] */
cluster 0 {
        miop; eiop 0; eiop 20; eiop 21; eiop 24; eiop 25; eiop 30;

}
cluster 1 {
        miop; eiop 0; eiop 20; eiop 21; eiop 30;

}
cluster 2 {
        miop; eiop 0; eiop 20; eiop 21; eiop 22; eiop 23;

}
cluster 3 {
        miop; eiop 0; eiop 20; eiop 21; eiop 22; eiop 23;

}
}

mainframe {
/* SN9003 - param.mf.hardware - Edition 11 [Tue Aug 26 18:36:41 CDT 1997] */
16 cpus;
512 Mwords memory;
channel 020 is lowspeed to cluster 0;
channel 030 is lowspeed to cluster 1;
channel 050 is lowspeed to cluster 2;
channel 052 is lowspeed to cluster 3;
channel 78 is lowspeed to pseudo TCP;
}

unicos {
/* SN9003 - param.unicos - Edition 6 [Thu Aug 21 11:31:44 CDT 1997] */
 5000 NBUF;
  200 NPBUF;
98280 LDCHCORE;
```

```
 5000 NLDCH;
  256 PDDMAX;
  256 LDDMAX;
   32 HDDMAX;
  300 PDDSLMAX;
    8 MDDSLMAX;
    8 SDDSLMAX;
    8 RDDSLMAX;
    4 SSDDSLMAX;
   64 HDDSLMAX;
    0 GUESTMAX;
294912 TAPE_MAX_PER_DEV;
    8 TAPE_MAX_CONF_UP;
   16 TAPE_MAX_DEV;
}

filesystem {
/* SN9003 - param.fs.disks - Edition 20 [Tue Sep 16 11:36:54 CDT 1997] */
/*
 *      Physical device configuration
 */
disk "02020.0" {
        type DD5S;
        iopath {
            cluster 0;
            eiop 20;
            channel 020;
        }
        unit 0;
        pdd 0s00_root_b {
            minor  3;
            sector  0;
            length 240000 sectors;
        }
        pdd 0s00_root_d {
            minor  4;
            sector  240000;
            length 240000 sectors;
        }
        pdd 0s00_OPEN {
            minor  5;
            sector  480000;
            length 41000 sectors;
```

```
        }
        pdd 0s00_usr_a {
            minor  6;
            sector  521000;
            length 260000 sectors;
        }
}
disk "02020.1" {
        type DD5S;
        iopath {
            cluster 0;
            eiop 20;
            channel 020;
        }
        unit 1;
        pdd 0s01_opt {
            minor  7;
            sector  0;
            length 781000 sectors;
        }
}
disk "02120.0" {
        type DD5S;
        iopath {
            cluster 0;
            eiop 21;
            channel 020;
        }
        unit 0;
        pdd 0s10_root_f {
            minor  8;
            sector  0;
            length 240000 sectors;
        }
        pdd 0s10_usr_b {
            minor  9;
            sector  240000;
            length 260000 sectors;
        }
        pdd 0s10_usr_d {
            minor  10;
            sector  500000;
            length 260000 sectors;
```

```
        }
        pdd 0s10_dmjrnl {
            minor  11;
            sector  760000;
            length 21000 sectors;
        }
}
disk "02120.1" {
        type DD5S;
        iopath {
            cluster 0;
            eiop 21;
            channel 020;
        }
        unit 1;
        pdd 0s11_root_e {
            minor  13;
            sector  0;
            length 240000 sectors;
        }
        pdd 0s11_root_a {
            minor  17;
            sector  240000;
            length 240000 sectors;
        }
        pdd 0s11_root_g {
            minor  18;
            sector  480000;
            length 240000 sectors;
        }
        pdd 0s11_OPEN {
            minor  20;
            sector  720000;
            length 61000 sectors;
        }
}
disk "02420.0" {
        type DD6S;
        iopath {
            cluster 0;
            eiop 24;
            channel 020;
        }
```

```
        unit 0;
        pdd 0s40_usr_e {
            minor  21;
            sector  0;
            length 260000 sectors;
        }
        pdd 0s40_OPEN {
            minor  22;
            sector  260000;
            length 203698 sectors;
        }
        pdd 0s40_u01 {
            minor  85;
            sector  463698;
            length 500013 sectors;
        }
        pdd 0s40_u23 {
            minor  74;
            sector  963711;
            length 500013 sectors;
        }
        pdd 0s40_ckpt {
            minor  53;
            sector  1463724;
            length 925276 sectors;
        }
}
disk "02420.1" {
        type DD6S;
        iopath {
            cluster 0;
            eiop 24;
            channel 020;
        }
        unit 1;
        pdd 0s41_root_utna {
            minor  23;
            sector  0;
            length 240000 sectors;
        }
        pdd 0s41_usr_ISV {
            minor  24;
            sector  240000;
```

```
            length 260000 sectors;
        }
        pdd 0s41_OPEN {
            minor  26;
            sector  500000;
            length 595858 sectors;
        }
        pdd 0s41_spool {
            minor  87;
            sector  1095858;
            length 367866 sectors;
        }
        pdd 0s41_root_h {
            minor  27;
            sector  1463724;
            length 240000 sectors;
        }
        pdd 0s41_dmspool {
            minor  29;
            sector  1703724;
            length 115276 sectors;
        }
        pdd 0s41_mail {
            minor  124;
            sector  1819000;
            length 70000 sectors;
        }
        pdd 0s41_u67 {
            minor  125;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "02520.0" {
        type DD6S;
        iopath {
            cluster 0;
            eiop 25;
            channel 020;
        }
        unit 0;
        pdd 0s50_root_j {
            minor  30;
```

```
            sector  0;
            length 240000 sectors;
        }
        pdd 0s50_dmfc1 {
            minor  142;
            sector  240000;
            length 37283 sectors;
        }
        pdd 0s50_dmfc2 {
            minor  143;
            sector  277283;
            length 37283 sectors;
        }
        pdd 0s50_dmfc3 {
            minor  144;
            sector  314566;
            length 37283 sectors;
        }
        pdd 0s50_dmfc4 {
            minor  145;
            sector  351849;
            length 37283 sectors;
        }
        pdd 0s50_dmfc5 {
            minor  146;
            sector  389132;
            length 37283 sectors;
        }
        pdd 0s50_dmfc6 {
            minor  147;
            sector  426415;
            length 37283 sectors;
        }
        pdd 0s50_ptmp {
            minor  89;
            sector  463698;
            length 1925302 sectors;
        }
}
disk "02520.1" {
        type DD6S;
        iopath {
            cluster 0;
```

```
    eiop 25;
    channel 020;
}
unit 1;
pdd 0s51_src_a {
    minor  32;
    sector  0;
    length 650000 sectors;
}
pdd 0s51_OPEN {
    minor  33;
    sector  650000;
    length 146104 sectors;
}
pdd 0s51_usr_adm {
    minor  92;
    sector  796104;
    length 299754 sectors;
}
pdd 0s51_usr_i {
    minor  34;
    sector  1095858;
    length 260000 sectors;
}
pdd 0s51_OPENa {
    minor  35;
    sector  1355858;
    length 107866 sectors;
}
pdd 0s51_core {
    minor  94;
    sector  1463724;
    length 301716 sectors;
}
pdd 0s51_OPENb {
    minor  141;
    sector  1765440;
    length 123560 sectors;
}
pdd 0s51_u89 {
    minor  129;
    sector  1889000;
    length 500000 sectors;
```

```
        }
}
disk "03020.0" {
        type DD5I;
        iopath {
            cluster 0;
            eiop 30;
            channel 020;
        }
        unit 0;
        pdd 0b00_swap {
            minor  41;
            sector  0;
            length 723000 sectors;
        }
}
disk "03020.1" {
        type DD5I;
        iopath {
            cluster 0;
            eiop 30;
            channel 020;
        }
        unit 1;
        pdd 0b01_tmp {
            minor  42;
            sector  0;
            length 723000 sectors;
        }
}
disk "03020.2" {
        type DD5I;
        iopath {
            cluster 0;
            eiop 30;
            channel 020;
        }
        unit 2;
        pdd 0b02_tmp {
            minor  51;
            sector  0;
            length 723000 sectors;
        }
```

```
}
disk "03020.3" {
        type DD5I;
        iopath {
            cluster 0;
            eiop 30;
            channel 020;
        }
        unit 3;
        pdd 0b03_tmp {
            minor  52;
            sector  0;
            length 723000 sectors;
        }
}
disk "12030.0" {
        type DD6S;
        iopath {
            cluster 1;
            eiop 20;
            channel 030;
        }
        unit 0;
        pdd 1s00_swap {
            minor  82;
            sector  0;
            length 463698 sectors;
        }
        pdd 1s00_OPEN {
            minor  36;
            sector  463698;
            length 632160 sectors;
        }
        pdd 1s00_mbin {
            minor  96;
            sector  1095858;
            length 469467 sectors;
        }
        pdd 1s00_admin {
            minor  97;
            sector  1565325;
            length 116865 sectors;
        }
```

```
        pdd 1s00_OPENa {
            minor  140;
            sector  1682190;
            length 206810 sectors;
        }
        pdd 1s00_u45 {
            minor  75;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "12030.1" {
        type DD6S;
        iopath {
            cluster 1;
            eiop 20;
            channel 030;
        }
        unit 1;
        pdd 1s01_root_upt {
            minor  37;
            sector  0;
            length 240000 sectors;
        }
        pdd 1s01_OPEN {
            minor  39;
            sector  240000;
            length 223698 sectors;
        }
        pdd 1s01_u45 {
            minor  76;
            sector  463698;
            length 500013 sectors;
        }
        pdd 1s01_u67 {
            minor  77;
            sector  963711;
            length 500013 sectors;
        }
        pdd 1s01_ckpt {
            minor  54;
            sector  1463724;
            length 925276 sectors;
```

```
        }
}
disk "12130.0" {
        type DD6S;
        iopath {
            cluster 1;
            eiop 21;
            channel 030;
        }
        unit 0;
        pdd 1s10_swap {
            minor  101;
            sector  0;
            length 463698 sectors;
        }
        pdd 1s10_usr_k {
            minor  40;
            sector  463698;
            length 260000 sectors;
        }
        pdd 1s10_usr_dm {
            minor  43;
            sector  723698;
            length 72406 sectors;
        }
        pdd 1s10_dump {
            minor  78;
            sector  796104;
            length 156114 sectors;
        }
        pdd 1s10_u89 {
            minor  79;
            sector  952218;
            length 500013 sectors;
        }
        pdd 1s10_OPENa {
            minor  44;
            sector  1452231;
            length 176769 sectors;
        }
        pdd 1s10_usr_utna {
            minor  45;
            sector  1629000;
```

```
            length 260000 sectors;
        }
        pdd 1s10_u67 {
            minor  126;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "12130.1" {
        type DD6S;
        iopath {
            cluster 1;
            eiop 21;
            channel 030;
        }
        unit 1;
        pdd 1s11_OPEN {
            minor  46;
            sector  0;
            length 463698 sectors;
        }
        pdd 1s11_ptmp {
            minor  104;
            sector  463698;
            length 1925302 sectors;
        }
}
disk "13030.0" {
        type DD5I;
        iopath {
            cluster 1;
            eiop 30;
            channel 030;
        }
        unit 0;
        pdd 1b00_swap {
            minor  61;
            sector  0;
            length 723000 sectors;
        }
}
disk "13030.1" {
        type DD5I;
```

```
        iopath {
            cluster 1;
            eiop 30;
            channel 030;
        }
        unit 1;
        pdd 1b01_tmp {
            minor  62;
            sector  0;
            length 723000 sectors;
        }
}
disk "13030.2" {
        type DD5I;
        iopath {
            cluster 1;
            eiop 30;
            channel 030;
        }
        unit 2;
        pdd 1b02_tmp {
            minor  71;
            sector  0;
            length 723000 sectors;
        }
}
disk "13030.3" {
        type DD5I;
        iopath {
            cluster 1;
            eiop 30;
            channel 030;
        }
        unit 3;
        pdd 1b03_tmp {
            minor  72;
            sector  0;
            length 723000 sectors;
        }
}
disk "22050.0" {
        type DD6S;
        iopath {
```

```
            cluster 2;
            eiop 20;
            channel 050;
        }
        unit 0;
        pdd 2s00_swap {
            minor  55;
            sector  0;
            length 463698 sectors;
        }
        pdd 2s00_utmp {
            minor  25;
            sector  463698;
            length 1301742 sectors;
        }
        pdd 2s00_usr_sl {
            minor  47;
            sector  1765440;
            length 123560 sectors;
        }
        pdd 2s00_u01 {
            minor  111;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "22050.1" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 20;
            channel 050;
        }
        unit 1;
        pdd 2s01_src_e {
            minor  48;
            sector  0;
            length 650000 sectors;
        }
        pdd 2s01_usr_upt {
            minor  49;
            sector  650000;
            length 260000 sectors;
```

```
        }
        pdd 2s01_root_k {
            minor  50;
            sector  910000;
            length 240000 sectors;
        }
        pdd 2s01_ram_root {
            minor  57;
            sector  1150000;
            length 14750 sectors;
        }
        pdd 2s01_root_c {
            minor  14;
            sector  1164750;
            length 240000 sectors;
        }
        pdd 2s01_OPEN {
            minor  58;
            sector  1404750;
            length 984250 sectors;
        }
}
disk "22150.0" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 21;
            channel 050;
        }
        unit 0;
        pdd 2s10_swap {
            minor  28;
            sector  0;
            length 463698 sectors;
        }
        pdd 2s10_src_f {
            minor  59;
            sector  463698;
            length 650000 sectors;
        }
        pdd 2s10_src_g {
            minor  60;
            sector  1113698;
```

```
            length 650000 sectors;
        }
        pdd 2s10_OPEN {
            minor  63;
            sector  1763698;
            length 125302 sectors;
        }
        pdd 2s10_u23 {
            minor  113;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "22150.1" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 21;
            channel 050;
        }
        unit 1;
        pdd 2s11_OPEN {
            minor  64;
            sector  0;
            length 463698 sectors;
        }
        pdd 2s11_ptmp {
            minor  65;
            sector  463698;
            length 1925302 sectors;
        }
}
disk "22250.0" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 22;
            channel 050;
        }
        unit 0;
        pdd 2s20_src_b {
            minor  66;
            sector  0;
```

```
            length 650000 sectors;
        }
        pdd 2s20_src_d {
            minor  67;
            sector  650000;
            length 650000 sectors;
        }
        pdd 2s20_OPEN {
            minor  68;
            sector  1300000;
            length 589000 sectors;
        }
        pdd 2s20_u89 {
            minor  115;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "22250.1" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 22;
            channel 050;
        }
        unit 1;
        pdd 2s21_src_h {
            minor  69;
            sector  0;
            length 650000 sectors;
        }
        pdd 2s21_src_i {
            minor  70;
            sector  650000;
            length 650000 sectors;
        }
        pdd 2s21_src_j {
            minor  73;
            sector  1300000;
            length 650000 sectors;
        }
        pdd 2s21_OPEN {
            minor  80;
```

```
            sector  1950000;
            length 439000 sectors;
        }
}
disk "22350.0" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 23;
            channel 050;
        }
        unit 0;
        pdd 2s30_OPEN {
            minor  88;
            sector  0;
            length 1889000 sectors;
        }
        pdd 2s30_u45 {
            minor  117;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "22350.1" {
        type DD6S;
        iopath {
            cluster 2;
            eiop 23;
            channel 050;
        }
        unit 1;
        pdd 2s31_src_k {
            minor  81;
            sector  0;
            length 650000 sectors;
        }
        pdd 2s31_src_upt {
            minor  83;
            sector  650000;
            length 650000 sectors;
        }
        pdd 2s31_src_ISV {
            minor  84;
```

```
            sector  1300000;
            length 650000 sectors;
        }
        pdd 2s31_OPEN {
            minor  86;
            sector  1950000;
            length 439000 sectors;
        }
}
disk "32052.0" {
        type DD6S;
        iopath {
            cluster 3;
            eiop 20;
            channel 052;
        }
        unit 0;
        pdd 3s00_swap {
            minor  90;
            sector  0;
            length 463698 sectors;
        }
        pdd 3s00_utmp {
            minor  38;
            sector  463698;
            length 1301742 sectors;
        }
        pdd 3s00_usr_sl {
            minor  91;
            sector  1765440;
            length 123560 sectors;
        }
        pdd 3s00_u01 {
            minor  119;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "32052.1" {
        type DD6S;
        iopath {
            cluster 3;
            eiop 20;
```

```
    channel 052;
}
unit 1;
pdd 3s01_root_i {
    minor  93;
    sector  0;
    length 240000 sectors;
}
pdd 3s01_usr_f {
    minor  95;
    sector  240000;
    length 260000 sectors;
}
pdd 3s01_usr_g {
    minor  98;
    sector  500000;
    length 260000 sectors;
}
pdd 3s01_usr_h {
    minor  99;
    sector  760000;
    length 260000 sectors;
}
pdd 3s01_usr_j {
    minor  100;
    sector  1020000;
    length 260000 sectors;
}
pdd 3s01_root_ISV {
    minor  102;
    sector  1280000;
    length 240000 sectors;
}
pdd 3s01_src_utna {
    minor  103;
    sector  1520000;
    length 650000 sectors;
}
pdd 3s01_OPEN {
    minor  105;
    sector  2170000;
    length 219000 sectors;
}
```

```
}
disk "32152.0" {
        type DD6S;
        iopath {
            cluster 3;
            eiop 21;
            channel 052;
        }
        unit 0;
        pdd 3s10_swap {
            minor  106;
            sector  0;
            length 463698 sectors;
        }
        pdd 3s10_utmp {
            minor  107;
            sector  463698;
            length 1301742 sectors;
        }
        pdd 3s10_OPEN {
            minor  108;
            sector  1765440;
            length 123560 sectors;
        }
        pdd 3s10_u23 {
            minor  121;
            sector  1889000;
            length 500000 sectors;
        }
}
disk "32152.1" {
        type DD6S;
        iopath {
            cluster 3;
            eiop 21;
            channel 052;
        }
        unit 1;
        pdd 3s11_usr_c {
            minor  15;
            sector  0;
            length 260000 sectors;
        }
```

```
        pdd 3s11_OPEN {
            minor  109;
            sector  260000;
            length 2129000 sectors;
        }
}
disk "32252.0" {
        type DD5S;
        iopath {
            cluster 3;
            eiop 22;
            channel 052;
        }
        unit 0;
        pdd 3s20_src_c {
            minor  16;
            sector  0;
            length 650000 sectors;
        }
        pdd 3s20_OPEN {
            minor  114;
            sector  650000;
            length 131000 sectors;
        }
}
disk "32252.1" {
        type DD5S;
        iopath {
            cluster 3;
            eiop 22;
            channel 052;
        }
        unit 1;
        pdd 3s21_CS_scr {
            minor  116;
            sector  0;
            length 771000 sectors;
        }
        pdd 3s21_tng_1 {
            minor  118;
            sector  771000;
            length 10000 sectors;
        }
```

```
}
disk "32352.0" {
        type DD5S;
        iopath {
            cluster 3;
            eiop 23;
            channel 052;
        }
        unit 0;
        pdd 3s30_CS_scr {
            minor  120;
            sector  0;
            length 771000 sectors;
        }
        pdd 3s30_tng_2 {
            minor  122;
            sector  771000;
            length 10000 sectors;
        }
}
disk "32352.1" {
        type DD5S;
        iopath {
            cluster 3;
            eiop 23;
            channel 052;
        }
        unit 1;
        pdd 3s31_OPEN {
            minor  123;
            sector  0;
            length 781000 sectors;
        }
}
/*
 * Logical devices
 */

sdd bswap {
        minor 2;
        pdd 0b00_swap;
        pdd 1b00_swap;
}
```

```
sdd vbswap {
        minor 3;
        pdd 1s00_swap;
        pdd 2s00_swap;
        pdd 3s00_swap;
        pdd 1s10_swap;
        pdd 2s10_swap;
        pdd 3s10_swap;
}
sdd usr_sl {
        minor 4;
        pdd 3s00_usr_sl;
        pdd 2s00_usr_sl;
}

ldd swap {
        minor 1;
        sdd bswap;
        sdd vbswap;
}
ldd admin {
        minor 46;
        pdd 1s00_admin;
}
ldd ckpt {
        minor 70;
        pdd 0s40_ckpt;
        pdd 1s01_ckpt;
}
ldd core {
        minor 39;
        pdd 0s51_core;
}
ldd dm_jrnl {
        minor 28;
        pdd 0s10_dmjrnl;
}
ldd dmspool {
        minor 29;
        pdd 0s41_dmspool;
}
ldd dump {
        minor 40;
```

```
        pdd 1s10_dump;
}
ldd mail {
        minor 55;
        pdd 0s41_mail;
}
ldd mbin {
        minor 48;
        pdd 1s00_mbin;
}
ldd opt {
        minor 96;
        pdd 0s01_opt;
}
ldd ptmp {
        minor 56;
        pdd 0s50_ptmp;
        pdd 1s11_ptmp;
        pdd 2s11_ptmp;
}
ldd ram_root {
        minor 57;
        pdd 2s01_ram_root;
}
ldd root_a {
        minor 17;
        pdd 0s11_root_a;
}
ldd root_b {
        minor 3;
        pdd 0s00_root_b;
}
ldd root_c {
        minor 14;
        pdd 2s01_root_c;
}
ldd root_d {
        minor 4;
        pdd 0s00_root_d;
}
ldd root_e {
        minor 13;
        pdd 0s11_root_e;
```

```
}
ldd root_f {
        minor 8;
        pdd 0s10_root_f;
}
ldd root_g {
        minor 18;
        pdd 0s11_root_g;
}
ldd root_h {
        minor 27;
        pdd 0s41_root_h;
}
ldd root_i {
        minor 93;
        pdd 3s01_root_i;
}
ldd root_j {
        minor 30;
        pdd 0s50_root_j;
}
ldd root_k {
        minor 50;
        pdd 2s01_root_k;
}
ldd root_upt {
        minor 37;
        pdd 1s01_root_upt;
}
ldd root_ISV {
        minor 102;
        pdd 3s01_root_ISV;
}
ldd root_utna {
        minor 23;
        pdd 0s41_root_utna;
}
ldd spool {
        minor 43;
        pdd 0s41_spool;
}
ldd src_a {
        minor 32;
```

```
        pdd 0s51_src_a;
}
ldd src_b {
        minor 66;
        pdd 2s20_src_b;
}
ldd src_c {
        minor 16;
        pdd 3s20_src_c;
}
ldd src_d {
        minor 67;
        pdd 2s20_src_d;
}
ldd src_e {
        minor 130;
        pdd 2s01_src_e;
}
ldd src_f {
        minor 59;
        pdd 2s10_src_f;
}
ldd src_g {
        minor 60;
        pdd 2s10_src_g;
}
ldd src_h {
        minor 69;
        pdd 2s21_src_h;
}
ldd src_i {
        minor 140;
        pdd 2s21_src_i;
}
ldd src_j {
        minor 73;
        pdd 2s21_src_j;
}
ldd src_k {
        minor 81;
        pdd 2s31_src_k;
}
ldd src_upt {
```

```
        minor 83;
        pdd 2s31_src_upt;
}
ldd src_ISV {
        minor 84;
        pdd 2s31_src_ISV;
}
ldd src_utna {
        minor 103;
        pdd 3s01_src_utna;
}
ldd tmp {
        minor 91;
        pdd 0b01_tmp;
        pdd 1b01_tmp;
        pdd 0b02_tmp;
        pdd 1b02_tmp;
        pdd 0b03_tmp;
        pdd 1b03_tmp;
}
ldd tng_1 {
        minor 118;
        pdd 3s21_tng_1;
}
ldd tng_2 {
        minor 122;
        pdd 3s30_tng_2;
}
ldd u01 {
        minor 123;
        pdd 2s00_u01;
        pdd 3s00_u01;
        pdd 0s40_u01;
}
ldd u23 {
        minor 124;
        pdd 2s10_u23;
        pdd 3s10_u23;
        pdd 0s40_u23;
}
ldd u45 {
        minor 125;
        pdd 1s00_u45;
```

```
        pdd 2s30_u45;
        pdd 1s01_u45;
}
ldd u67 {
        minor 126;
        pdd 0s41_u67;
        pdd 1s10_u67;
        pdd 1s01_u67;
}
ldd u89 {
        minor 127;
        pdd 0s51_u89;
        pdd 2s20_u89;
        pdd 1s10_u89;
}
ldd usr_adm {
        minor 45;
        pdd 0s51_usr_adm;
}
ldd usr_a {
        minor 6;
        pdd 0s00_usr_a;
}
ldd usr_b {
        minor 9;
        pdd 0s10_usr_b;
}
ldd usr_c {
        minor 15;
        pdd 3s11_usr_c;
}
ldd usr_dm {
        minor 53;
        pdd 1s10_usr_dm;
}
ldd usr_d {
        minor 10;
        pdd 0s10_usr_d;
}
ldd usr_e {
        minor 21;
        pdd 0s40_usr_e;
}
```

```
ldd usr_f {
        minor 132;
        pdd 3s01_usr_f;
}
ldd usr_g {
        minor 98;
        pdd 3s01_usr_g;
}
ldd usr_h {
        minor 99;
        pdd 3s01_usr_h;
}
ldd usr_i {
        minor 34;
        pdd 0s51_usr_i;
}
ldd usr_j {
        minor 100;
        pdd 3s01_usr_j;
}
ldd usr_k {
        minor 128;
        pdd 1s10_usr_k;
}
ldd usr_upt {
        minor 49;
        pdd 2s01_usr_upt;
}
ldd usr_ISV {
        minor 24;
        pdd 0s41_usr_ISV;
}
ldd usr_sl {
        minor 47;
        sdd usr_sl;
}
ldd usr_utna {
        minor 129;
        pdd 1s10_usr_utna;
}
ldd utmp {
        minor 95;
        pdd 3s00_utmp;
```

```
        pdd 2s00_utmp;
        pdd 3s10_utmp;
}
ldd dmfclass1 {
        minor 142;
        pdd 0s50_dmfc1;
}
ldd dmfclass2 {
        minor 143;
        pdd 0s50_dmfc2;
}
ldd dmfclass3 {
        minor 144;
        pdd 0s50_dmfc3;
}
ldd dmfclass4 {
        minor 145;
        pdd 0s50_dmfc4;
}
ldd dmfclass5 {
        minor 146;
        pdd 0s50_dmfc5;
}
ldd dmfclass6 {
        minor 147;
        pdd 0s50_dmfc6;
}
/* SN9003 - param.fs.special - Edition 3 [Wed Aug 20 14:37:53 CDT 1997] */
rootdev is ldd root_c;
swapdev is ldd swap;
dmpdev is ldd dump;
}

network {
/* SN9003 - param.network - Edition 6 [Thu Aug 21 11:31:52 CDT 1997] */
    8 nfs_static_clients;
    8 nfs_temp_clients;
    8 cnfs_static_clients;
    8 cnfs_temp_clients;
32768 nfs_maxdata;
  256 nfs_num_rnodes;
 1200 nfs_maxdupreqs;
    3 nfs_duptimeout;
```

```
    0 nfs_printinter;
16000 tcp_nmbspace;
    2 himaxdevs;
    4 himaxpaths;
    1 fdmaxdevs;
    0 npmaxdevs;
    1 enmaxdevs;
    2 atmmaxdevs;
131072 atmarp_recv;
65536 atmarp_send;
 1024 atmarp_entries;

0755 hidirmode;
0666 hifilemode;

endev   0 {
        iopath {
            cluster 1;
            eiop 0;
            channel 020;
        }
}

fddev   0 {
        iopath {
            cluster 2;
            eiop 0;
            channel 040;
        }
}
atmdev  0 {
        iopath {
            cluster 0;
            eiop 0;
            channel 020;
        }
}
atmdev  1 {
        iopath {
            cluster 3;
            eiop 0;
            channel 020;
        }
```

```
}
}
```

Procedure 4: Identifying devices defined on your system and their file system allocation

Note: To complete this procedure, you must be super user; you will see the `sn` *xxxx* # prompt.

To identify the devices provided on your system and their file system allocation, either use the menu system or execute commands.

If you are using the menu system, complete the following steps:

Enter the menu system:

Note: To eliminate the need to change to the `/etc/install` directory to enter the menu system, you can include `/etc/install` in your `PATH` statement in your `.profile` or `.cshrc` file.

```
snxxxx# cd /etc/install
snxxxx# ./install
```

1. Select the following menu:

```
UNICOS Installation / Configuration Menu System
    ->Configure system
        ->Disk configuration
```

Determine which devices and file systems are configured on your system by viewing the submenus.

Section 5.9.2, page 62, describes the sections of the `/sys/param` file.

A sample menu screen follows:

```
                        Disk Configuration

M-> Physical devices ==>
    Physical device slices ==>
    Logical devices (/dev/dsk entries) ==>
    Mirrored devices (/dev/mdd entries) ==>
    Striped devices (/dev/sdd entries) ==>
    Logical device cache ==>
    Verify the disk configuration ...
    Review the disk configuration verification ..
    Dry run the disk configuration ...
    Review the disk configuration dry run ...
    Update disk device nodes on activation?
    Import the disk configuration ...
    Activate the disk configuration ...
```

If you are not using the menu system, use the following commands to display information that you can use to identify the devices on your system and their file system allocation:

- The `/etc/pddstat` command displays the name of the device, its type, and whether it is up or down.
- The `/etc/ddstat /dev/dsk/*` command displays all disk devices and their file system allocation (or you can execute the command for individual devices). Logical devices are divided into their individual components and presented in a disk-specific format. The output is not formatted (headings are not provided), but the output provides comprehensive information. The following is an example of `ddstat` output from a CRAY J90 system. The fields are defined in the diagram that follows:

```
$ ddstat /dev/dsk/tmp

/dev/dsk/tmp b 34/69 0 0 /dev/ldd/tmp
       /dev/pdd/tmp_1 c 32/69 12 01036020 0 201600 00 0 0 0
       /dev/pdd/tmp_2 c 32/96 12 01036020 0 201600 00 0 0 1
       /dev/pdd/tmp_3 c 32/97 12 01036020 0 201600 00 0 0 2
```

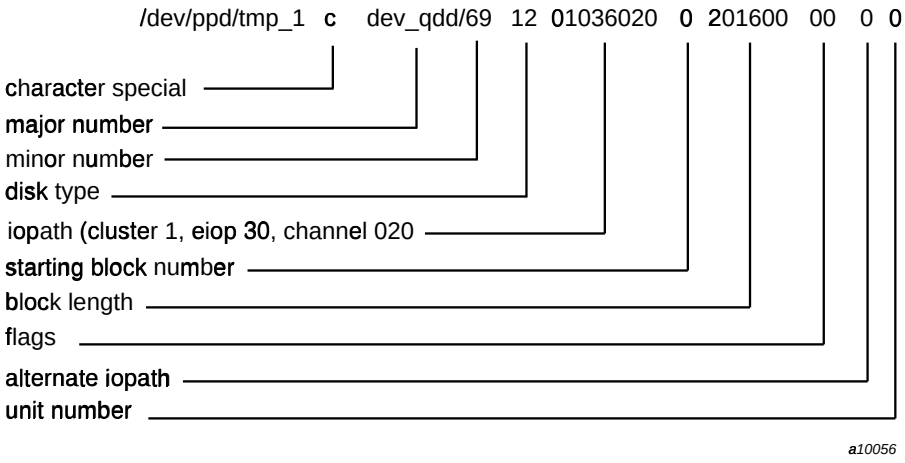

Figure 1. `ddstat` output

Procedure 5: Modifying your configuration file

Note: To do this procedure, you must be super user; you will see the `sn` *xxxx* # prompt.

To modify your configuration, either use the menu system or edit the parameter file.

If you are using the menu system to modify your configuration file, follow the procedure on the "Identifying devices defined on your system and their file system allocation" procedure. Then import the disk configuration, modify the menus as needed, and then activate your new configuration (`Activate the disk configuration ...` line of the `Disk Configuration` menu).

If you are not using the menu system, complete the following steps.

Note: A CRAY J90 IOS-V is case sensitive; enter all lowercase characters for IOS-V commands.

1. Create a backup copy of any file system that will be changed in your revised configuration file (`sys/param`) by using the `dump` command. See Chapter 6, page 133.
2. Create a backup copy of your current configuration file:

```
snxxxx# <CONTROL-A>

(toggles to the IOS)

snxxxx-iosx> cp /sys/param old.param
snxxxx-iosx> <CONTROL-A><RETURN>

(toggles to UNICOS)
```

3. Make sure that you are in `/etc/config` on the UNICOS system. Copy the IOS configuration file `sys/param` from the IOS0 disk drive to a UNICOS disk and a file name of your choice (`new.param` in the following example) by using the `/bin/exdf` command so that you can edit it. The following command specifies that the `/sys/param` file will be read from the IOS system disk (the `-i` input option) and be named `new.param` (the `>` redirection):

```
snxxxx# exdf -i /sys/param > new.param
```

4. Edit your copy of the parameter file on the UNICOS system (see Section 5.9.2, page 62).

```
snxxxx# vi new.param
```

5. Check for syntax errors by using the `/etc/econfig` command:

```
snxxxx# /etc/econfig new.param
```

6. When you are done making your configuration changes, copy your new version of the system configuration (`new.param`) on top of the old original version of the system configuration (`sys/param` on the IOS disk), using the `/bin/exdf`) command. The following command specifies that the `new.param` file will be written to the IOS system disk (the `-o` output option and `<` redirection) and be named `sys/param`:

Caution: If you use the `exdf -r` option as shown in the following example, the file will be overwritten; before you use the `-r` option, be sure that a back-up copy of your current configuration file exists.

```
snxxxx# exdf -ro sys/param < new.param
```

At this point, the next time the UNICOS system is booted, it will come up with the new system configuration that you specified, and the system will copy the IOS `sys/param` file to the UNICOS `/CONFIGURATION` file.

7. After the system is booted to single-user mode, you must make, label, check, and mount any file system (old or new) that differs in any way from the way it was previously defined in the original version of the IOS `sys/param` file you changed. (Section 5.13, page 120 describes these additional steps.) You then must restore altered file systems from the back-up tapes you created in step 1 by using the `restore` command.
8. You must create the new `mknod` information for any new disk devices you have defined. To do this, use the `econfig` command to create a file containing the `mknod` information:

```
snxxxx# econfig -d new_param_file > /dev/mkdev
```

Remove the existing devices:

```
snxxxx# rm dsk/* pdd/* ldd/* sdd/* mdd/*
```

Generate the new device definitions:

```
snxxxx# cd /dev
snxxxx# chmod 755 mkdev
snxxxx# ./mkdev
```

5.11 File system quotas

The file system quota system allows you to control the amount of file system space in blocks and numbers of files used by each account, group, and user on an individual basis. Control may be applied to some or all of the configured file systems, except for the root file system. Attempts to exceed these limits result in an error similar to the error that occurs if the file system is out of free space.

5.11.1 File system quota overview

File system quotas are implemented to control the amount of file system space consumed by users and are characterized as follows:

- You can set quotas for three different ID classes:
  - User ID (`uid`)
  - Group ID (`gid`)
  - Account ID (`acid`)
- You can set up two types of quotas, file and inode:
  - *File quotas* determine the amount of space an ID may consume in blocks (512 words=4096 bytes).
  - *Inode quotas* determine the number files an ID can create.
- You can apply controls to some or all of the configured file systems (except the `root` file system).
- You can create adjustable warning windows to inform the user when usage gets close to a quota.

5.11.2 Quota control structure

The quota control file, `.Quota60`, which by default resides on the file system it controls, contains all the information the quota system needs. A header in the quota control file contains the default information for IDs, such as default file and inode quotas, default warning window, and warning fractions. The default values are taken from a header file, `/usr/include/sys/quota.h`, and may be modified through the administrative command, `quadmin`(8).

Every ID number (user, group, and account) up to `MAXUID` has an offset into the quota control file. At that offset, control information exists for each ID class. Each of the following fields exists for each ID in the quota control file:

Field	Contents		
`Flags`	Only one flag is defined, which indicates that the entry has been preset by quadmin rather than the kernel.		
`File quotas` `Inode quotas`	Maximum number of file blocks or inodes allowed by the ID. The following special values are stored in this field:		
	#	Keyword	Description
	0		No value specified.

Field	Contents		
	2	default	Use the default file/inode quota that appears in the control file hearder.
	3	infinite	Infinite quota (no quota evaluatin is done).
	4	prevent	No blocks/inodes may be allocated by this ID.
`File warning window`	Number of blocks below the maximum number of file blocks when a warning should be issued.		
`Inode warning window`	Number of inodes below the maximum number of inodes when a warning should be issued.		
`File usage`	Current number of blocks used by the ID.		
`Inode usage`	Current number of inodes used by the ID.		
`Quota hit`	Time when the quota is reached.		

5.11.3 Commands

The following commands are used to administer file system quotas:

- `qudu`(8): Reports file system quota usage information
- `quadmin`(8): Administers file quotas
- `mount`(8): Mounts a file system (options for specifying quota control file)
- `quota`(1): Displays quota information

5.11.4 Quotas and the user

Every file system user on the Cray Research system can be controlled by a quota. As file system space is consumed, the user ID, group ID, and account ID, sorted in the file's inode, accumulate the file system usage information. When a user exceeds a quota, an error occurs that is similar to when the file system is out of free space. A `SIGINFO` signal is also sent when a warning is reached or a quota is exceeded. This signal, which is ignored by default, can be caught and interpreted by using a `getinfo`(2) request.

When data migration is turned on and a file is migrated, the space the file occupied is credited to the file owner's ID. When a file is brought back online, the number of blocks is added to the ID's file quota. If bringing a file back would violate an enforced quota limit, that file cannot be brought online.

If a new user is added to the system, the UNICOS kernel automatically creates a quota control entry with default values (taken from the header information in the `.Quota60` file) for any IDs that are not already defined in the quota control files.

If you need to adjust these values for that specific ID, run the `quadmin` command to set up the correct quota information for the ID on each file system.

5.11.5 Quota header file

The header file, `quota.h`, contains global information for file system quotas. It is recommended that you do not change the values in the header file. Use `quadmin` to adjust the value to better suite the needs of your site.

The following is an excerpt from a `quota.h` file:

```
# cat /usr/include/sys/quota.h...
#define  QFV_AFQ          5000     /* account default quota */
#define  QFV_AIQ          200      /* account default inodes */
#define  QFV_GFQ          5000     /* group default quota */
#define  QFV_GIQ          200      /* group default inodes */
#define  QFV_UFQ          5000     /* user default quota */
#define  QFV_UIQ          200      /* user default inodes */
#define  QFV_WARNAFQ      0.9      /* account file warning default */
#define  QFV_WARNAIQ      0.9      /* account inode warning default */
#define  QFV_WARNGFQ      0.9      /* group file warning default */
#define  QFV_WARNGIQ      0.9      /* group inode warning default */
#define  QFV_WARNUFQ      0.9      /* user file warning default */
#define  QFV_WARNUIQ      0.9      /* user inode warning default */
```

5.11.5.1 Soft quotas

Soft quotas is a mode of operation, also called oversubscription, that allows a user to exceed quotas by a controlled number of blocks for a limited period of time. It is selected by setting the algorithm selector in a header field. For more information about setting up oversubscription, see *General UNICOS System Administration*, Cray Research publication SG–2301.

Procedure 6: Setting up a quota control file

When your site decides to turn on the quota system, you must complete the following steps to ensure that a quota file exists:

1. To ensure that your system has a kernel built with the quota system turned on, look at the following UNICOS Installation / Configuration menu item:

```
Configure System
    ->Major Software Configuration
        ->S-> File Quotas on
```

Note: If you are implementing quotas on a newly created file system, skip step 2 and go on to step 3.

2. To implement quotas on an existing file system, collect current usage information for all user, group, and account IDs by using the `qudu` command. The output for `qudu` contains directives for the `quadmin` command, which will create or update the quota control file. Either redirect the output to a file, or pipe the output directory to `quadmin`.

```
# umount /dev/dsk/usa
# qudu /dev/dsk/usa > qudu.out
# cut -d' ' -f1-5 qudu.out| sort +0 -1 +4nr
```

(View IDs according to classes (uid, gid, and acid) with each class sorted so that the ID with the greatest inode usage is printed first.)

```
# cut -d' ' -f1,2,6-8 qudu.out| sort +0 -1 +4nr
```

(View IDs according to classes (uid, gid, and acid) with each class sorted so that the ID with the greatest file usage is printed first.)

3. Modify any quota entries in the `quadmin` source file (you can use any editor on the source file). All IDs take on the default values for file and inode quota limits unless they are updated by using the `quadmin` command. You may want to view what the current level of usage is for the file system (the `sort` and `cut` commands may be useful to accomplish this task). If you find that several IDs are already over the quota, you may want to consider raising the quota of those IDs or the overall default quota.

Note: Root and daemon IDs should not be under quota control. Put quota values of `infinite` in these ID fields. Setting values lower than 10 will result in a value of 10.

```
# vi qudu.out

(Append the following information:)

   enable uid 0-100
   user * file quota infinite
   user * inode quota infinite
   enable gid 0-100
   group * file quota infinite
   group * inode quota infinite
   enable acid 0-100
   account * file quota infinite
   account * inode quota infinite
   user sue file quota 35000
   user sue inode quota 500

# fsck /dev/dsk/usa
# mount /dev/dsk/usa /usa
```

4. Create the quota control file, `.Quota60`, for the file system. A quota control file is associated with a file system at the time the file system is mounted. Quotas are enforced when the file system is mounted with the `-q` or `-Q` option.

```
# quadmin -F -m qudu.out
# umount /dev/dsk/usa
# mount -Q /usa/.Quota60 /dev/dsk/usa /usa
```

5. Establish ownership of the quota file to be `root` and specify that others cannot access, modify, or delete the file.

```
# chown root /usa/.Quota60
```

6. If you mounted your file system using the `mount -q` or `-Q` option, you can activate file system quotas from one of the site-modified startup scripts (either `/etc/rc.mid` or `/etc/rc/pst`).

7. If you mounted your file system without specifying the `-q` or `-Q` options, you can activate quotas by using the `quadmin` command. The `quadmin` command has the following three activation levels, which can be changed at any time:

- *count* maintains counts for the quota system, but does not send a warning or quota limit signal and does not enforce quotas.
- *inform* maintains counts and informs users with a warning or quota limit signal, but does not enforce quotas.
- *enforce* maintains counts, issues warning and quota limit signals, and enforces quotas.

```
# quadmin -c enforce -s /usa
```

Note: Once quota control is activated, you can change its enforcement mode, but you cannot deactivate it. You must unmount the file system to deactivate quota controls.

5.11.6 Current usage information

When the `/etc/fsck` or `/etc/gencat` utilities find file system errors and try to correct the problem, they may remove an inode or modify information about a file.

Because these commands do not update the quota control file with current inode or file usage, you should run `/etc/qudu` after running `fsck` or `gencat`. Then run `quadmin` immediately after the device is mounted.

```
# fsck -u /dev/dsk/usa
# qudu /dev/dsk/usa > /qudu.out
# mount -q /dev/dsk/usa
# quadmin /qudu.out
```

5.11.7 Warning windows

As administrator, you are responsible for setting the warning window value. This is initially set through parameters in the `quota.h` file and can be adjusted by using `quadmin` directives.

Warning windows can be represented as fractions or absolute window values. A warning fraction, f, must be in the range of $0.0 < f < 1.0$. A number of 10 or greater is considered an absolute window value. A number ranging from 1 through 9 is interpreted as zero, meaning that there is no warning window and no warning will ever occur. An absolute window value is interpreted as the total number of blocks or inodes below the file or inode quota.

The following example causes a warning message to appear when a user has used up 90% of the allowed file quota or inode quota:

```
# quadmin -m infile1
# cat infile1
filesystem dsk/usa ; open dsk/usa
default acid file warning 0.9
default uid file warning 0.9
default gid file warning 0.9
default acid i-node warning 0.9
default uid i-node warning 0.9
default gid i-node warning 0.9
```

5.11.8 Sharing quota controls files between multiple file systems

You may have a single quota control file manage more than one file system. To set up and activate a shared quota control file, you should observe the following guidelines:

- When accumulating current usage information, you must run separate `qudu` commands for each file system. Then you must sum up the usage of all IDs involved in these file systems that are going to share the control file. There is currently no command to accomplish this task.
- You must ensure that the file system on which the quota control file resides is mounted before quotas can be activated on another file system that will share this quota control file.
- When the mount command is executed for the file systems that will share this quota control file, you must specify the `-Q` option.

Observe the following disadvantages of a shared control file:

- If there is too much quota control traffic, the impact on performance is uncertain.
- If a file system containing a quota control file is destroyed, quota control is lost on the file system that was sharing that quota control file.

5.11.9 Monitoring quotas

User warning and limit messages are automatically written to the standard error file, `stderr`, by the Korn, POSIX, and C login shells.

You can examine quotas by using the `quota` command. If you want to check all of a user's authorized account IDs and group IDs, enter the following command:

```
# quota -A -G -r wl

File system: /cyclone
 User: john, Id: 1846
              File blocks    Inodes
User Quota:   4000* ( 2.1%)  4000* ( 0.5%)
 Warning:     3600* ( 2.3%)  3600* ( 0.6%)
   Usage:     84             20
```

5.12 Planning file system change

You may reconfigure your file systems occasionally, usually to allow for growth in your file systems, to add new disks, and to meet new operational requirements. A well-thought-out and well-defined plan that is generated in advance can help smooth out this process.

5.12.1 Configuration objectives

To determine configuration objectives, understand your current configuration and determine your objectives for the final disk layout. Familiarize yourself with the form and syntax of the configuration specific language (CSL) that is used to describe file system layouts. Compare the output of the `ddstat /dev/dsk/*` command with the disk layout `param` file.

5.12.2 Plan preparation

To prepare a plan, separate the process of change into incremental steps, stating the objectives for each step. Do not try to make too many changes in one step, and try to combine changes that complement each other into one step.

If you can unmount a file system safely, you can change it in multiuser mode. While in multiuser mode, do **not** try to change the following:

- `root`, `usr`, `home`, `spool`, `tmp`, and `adm` file systems
- Any file system that may become active because of DMF, NQS, NFS, `cron`, MLS, or other activity
- Swap device area

You can change any file system in single-user mode except `root` and the swap device area, which require `param` file adjustments and a system reboot.

For each step, prepare a plan that details the following items:

- List each disk that changes.
- List each file system destroyed, created, or changed.
- For each file system destroyed:
  - If the data will be saved, verify that the contents from this file system were saved earlier in the plan.
  - Delete redundant `/dev` entries.
- For each file system created:
  - Format the file system by using the `mkfs`, `labelit`(8), and `fsck`(8) commands.
  - Populate (restore data to) the file system with the saved data (if any).
- For each file system changed:
  - Check that the data from the file system was saved earlier in the plan.
  - Format the file system by using the `mkfs`, `labelit`, and `fsck` commands.
  - Populate the file system with the saved data.

To ensure that any data required for the next stage is preserved, check your plan. Review your plan with a colleague or Cray Research service representative.

5.12.3 New disks

To bring a new disk online, add the disk to the disk `param` file and then reboot your system.

Your hardware installer will advise you where new disks have been attached to your system by providing channel, device, and unit numbers. Flaw tables, if applicable, will be initialized, but Redundant Arrays of Independent Disks (RAID) devices may require special initialization procedures.

5.12.4 Implementation

To implement the plan, follow these steps:

1. Back up all your data to tape. Verify the saved data (make sure that you verify the backup tapes).
2. Save a copy of the original production disk layout `param` files on the IOS.
3. Check for syntax and slice gaps or overlap by checking the `param` files by using the `econfig`(8) command. This can be done on the UNICOS system in single-user mode.
4. Allow ample time for the change; costly mistakes are more often made when working under pressure. To determine the amount of time you need, multiply the time it takes to shut down and reboot your system by the number of restarts in your plan. Then add the time needed to backup and restore the data to be moved.

5.12.5 Apply changes

You can use either of the following methods to apply the changes to the new disk layout `param` file.

Method 1:

1. Unmount the file systems that will change.
2. Load the changes into the UNICOS Installation / Configuration Menu system (the installation tool); that is, change variables and parameters in the installation tool to reflect the new, desired file system configuration.
3. Activate the changes.

Method 2:

1. Unmount the file systems that will change.
2. Generate a new `param` file (copy and modify `/etc/config/param` or `/CONFIGURATION`).
3. Generate the `mknod` commands for your new file system configuration by running the `econfig -d` command.
4. For the file systems that change, run the `mknod` commands generated by the `econfig -d` command.

5.12.6 As you proceed

Perform the following steps as you proceed with your file system change plan:

1. Check off items on your detailed plan as you proceed, noting any diversions.
2. Before you format a file system, carefully verify its placement by using the `dmap` command.
3. Verify that each newly completed file system contains what you expect it to contain.
4. Optimize file system usage by applying appropriate `mkfs`(8) options.

5.12.7 Helpful hints for implementing plan

The following information may be helpful as you implement your plan.

- To copy entire file systems, use the `dump`(8) and `restore`(8) commands; to make partial copies, use `find`(1) and `cpio`(1), or `tar`(1).
- Checkpointed jobs will not continue if the inode numbers of files used by that job change or the minor device number of the holding file system changes; any file system reconfiguration will cause restart failures for checkpointed jobs that have open files on any affected file system.
- The `dump` and `restore` commands change the inode numbers and defragment a file system. You can use the `dd`(1) command only between file systems of the same size and type.
- Moving or reordering slices or adding or removing striping or mirroring changes a file system.
- If you are running in single-user mode, the swap device must exist, but it does not have to be full production size.
- Because the swap device definition comes from the `param` file and not its `/dev` entry, you can move it arbitrarily across system reboots (that is, it needs no preparation before use).
- You can prepare and use the dump device area as a temporary file system; however, be sure that you reinitialize it after you have finished your file system changes.
- If you plan to destroy the `/tmp` file system, notify your users (users like to be warned about changes to the `/tmp` file system).

- When you change a file system that is subject to data migration, you must perform special steps. If you are unsure what is included in these steps, contact your Cray Research service representative.
- Although disk slice names do not have to include the disk number, a logical, ordered naming convention can be useful.
- To tag your data, place a file called `1.am.` *fsname* at the head of each file system (*fsname* represents the name of the file system).
- Do not reuse minor device numbers but keep the highest minor device number under the *type* `MAX` limit for your kernel (in which *type* represents the device type).
- You can use striping only on disk devices that have the same physical type; striping must be between slices of the same size and position on different disks.
- To speed data population, apply logical device cache (`ldcache`) to a destination file system.

5.13 Creating file systems

After you have planned the configuration of your physical and logical devices and defined them using CSL, you must follow the steps described in this section to create file systems on your logical devices. (To determine the devices provided with your system and how they are allocated to file systems, see Section 5.9.3.5, page 66.)

1. Build the file system by using the `/etc/mkfs` command.
2. (Optional) Label the file system by using the `/etc/labelit` command.
3. Check the file system structure integrity by using the `/etc/fsck` command.
4. If it does not already exist, create the mount point directory, using the `/etc/mkdir` command.
5. Mount the directory by using the `/etc/mount` command.

The remainder of this section describes the following:

- `/etc/mnttab` and `/etc/fstab` files
- Configuring a file system to be mounted automatically at the initialization of multiuser mode

- Unmounting a file system by using `/etc/umount`

  Note: *General UNICOS System Administration*, Cray Research publication SG–2301, and *UNICOS Resource Administration*, Cray Research publication SG–2302, include information on other aspects of file system maintenance. For example, *UNICOS Resource Administration*, Cray Research publication SG–2302 how the file system space monitoring capability can improve the usability and reliability of the system. Space monitoring observes the amount of free space on mounted file systems and takes remedial action if warning or critical thresholds are reached. *UNICOS Resource Administration*, Cray Research publication SG–2302 explains how the file system quota enforcement feature (also called *disk quotas*) lets you control the amount of file system space in blocks and the number of files used by each account, group, and user on an individual basis. You may apply controls to some or all of the configured file systems, except for the `root` file system. Attempts to exceed quota limits cause an error similar to the error that occurs if the file system is out of free space. Optional warning levels also are available for informing users when usage gets close to a quota limit.

Procedure 7: Create the file system

1. Building the file system

The `/etc/mkfs` command builds the file system structure in the areas of disk that make up the logical device for a given file system. This structure includes designating areas of the logical device to contain the boot block, super blocks, inode region, and so on. On CRAY J90 systems, you always should use the `-q` option when you run `mkfs` to build a structure, which will prevent the disk surfaces from being verified (the IOS `dsurf` and `dslip` commands do this). (When the UNICOS multilevel security (MLS) feature is enabled, `mkfs` provides the new file system with minimum and maximum security levels and authorized compartments.) The format of the `mkfs` command is as follows:

```
/etc/mkfs [-q] [-n blocks] [-a strategy] [-B bytes] [-A blocks]
  device
```

`-q`	Specifies quick mode; bypasses surface check.
`-n` *blocks*	Specifies number of blocks you want the file system to contain.
`-a` *strategy*	Specifies an allocation strategy. This option can take one of the following values:

	`rrf` Round-robin all files (default) `rrd1` Round-robin first-level directories `rrda` Round-robin all directories
`-B` *bytes*	Specifies the number of bytes after which a file is considered to be big. The default is 32,768 bytes (8 blocks) and is defined by the `BIGFILE` argument in `/usr/src/uts/sys/param.h`; you cannot change the definition. The default might be the value you want to use at your site.
`-A` *blocks*	Specifies the minimum number of 4-Kbyte blocks allocated for a file whose size is greater than or equal to `BIGFILE` (see the `-B` option). The default is 21 sectors (blocks) and is defined by the `BIGUNIT` argument in `/usr/src/uts/sys/param.h`; you cannot change the definition. The default might be the value you want to use at your site. For DD-60, DA-62, and DA-301 disk drives, for which the sector size is 16 Kbytes, the allocation unit is rounded up to the nearest multiple of four. For DA-60 disk drives, for which the sector size is 64 Kbytes, the allocation unit is rounded up to the nearest multiple of 16. The interaction of the `-A` and `-B` options is as follows. If a file creation request exceeds the size of `BIGFILE` (8 blocks), the system will allocate `BIGUNIT` (21) more blocks in an attempt to meet the request. The system then checks to see whether the request has been met. If the amount allocated so far is still less than the request, the system will allocate another `BIGUNIT` number of blocks and again check to see whether the request has been met. This cycle of allocation and checking will repeat until the request has been met. You must determine the best settings for the `-A` and `-B` options for your file systems and average allocation requests at your site.
device	Full path name of the block special file (`/dev/dsk/` *filename*). When the disk

configuration is activated at system startup, block special files are created for each logical device in your configuration. They are placed in the `/dev/dsk` directory and take on the same name as the logical device. You must know the full path name.

A basic example follows:

```
/etc/mkfs -q /dev/dsk/home
```

The following examples show the syntax and explain each of the three possible allocation strategies.

Example 1 uses a "round-robin, first-level" strategy (`rrd1`) to create a file system called `bob`. It tries to place all files, subdirectories, and directories of a file system on the same partition.

Example 1: round-robin, first-level

```
# /etc/mkfs -q -a rrd1 /dev/dsk/bob
```

Example 2 uses a "round-robin, all-directory" strategy (`rrda`) to create a file system named `jane`. Each directory and its files are allocated to the same partition, but each directory is allocated to a different partition than its subdirectories if possible.

Example 2: round-robin, all-directory

```
# /etc/mkfs -q -a rrda /dev/dsk/jane
```

Example 3 uses a "round-robin, all-files" strategy (`rrf`) to create a file system named `jones`. This strategy tries to place all inodes and directories on partition 0 if possible, and it allocates all files for a file system in a "round-robin" fashion. For example, on a three-partition file system, as files `a`, `b`, `c`, `d`, `e`, `f`, and `g` are created, `a` will be placed on partition 0, `b` on partition 1, `c` on partition 2, `d` on partition 0, `e` on partition 1, `f` on partition 2, `g` on partition 0, and so on.

Example 3: round-robin, all-files

```
# /etc/mkfs -q -a rrf /dev/dsk/jones
```

Continue with step 2.

2. Labeling the file system

To create a label on a newly created file system, use the `/etc/labelit` command. This step is optional, but when not done, a warning message is issued when the file system is mounted. The `mount:warning: <` *file-system-name* `> mounted as </` *mount-point-name* `>` message appears when the file system label does not match the mount point directory name. The syntax of `/etc/labelit` is as follows:

```
/etc/labelit device fsname volname
```

device — The name of the logical device that you want to label.

The actual label consists of the following two required fields:

fsname — The name you want to assign to the file system.

volname — The name you want to assign to the volume.

Note: If you do not specify a label, `labelit` displays current label information about a file system; see the following examples.

Example 4: assign file system name and volume name to umounted file system

The following command assigns a file system name of `usr01` and a volume name of `vol1` to the unmounted file system on `/dev/dsk/usr01`. Notice the new volume and new file system name as specified in the last command response line.

```
# /etc/labelit /dev/dsk/scr_esdi usr01 vol1
Current fsname: scr_esdi, Current volname: E000_scr, Blocks: 487800, Inodes: 121968
Date last mounted: Sun Sep 26 03:06:50 1993
NEW fsname = usr01, NEW volname = vol1
```

Example 5: `labelit` output

If you do not specify a label, `labelit` displays current label information about a file system, as shown in the following example, which specifies only the file system name:

```
# /etc/labelit /dev/dsk/scr_esdi
Current fsname: scr_esdi, Current volname: E000_scr, Blocks: 487800, I-nodes: 121968
Date last mounted: Sun Sep 26 10:52:53 1993
```

Continue with step 3.

3. Checking the file system

   Note: You must check a file system **before** it is mounted; otherwise, the file system will not be mounted. Before mounting a file system, always perform a consistency check on it to ensure that a reliable environment exists for file storage. When the system is brought to multiuser mode, the `/etc/bcheckrc` multiuser level start-up script automatically checks any file systems listed in the `/etc/fstab` file. The `/etc/fstab` file also has an option that can cause its files to be mounted automatically at multiuser start-up time (see Section 5.14.2, page 130). Because of the multipass nature of the `/etc/fsck` command, the file systems must be in an inactive state while being checked. You must ensure that all file systems to be checked are unmounted.

The `/etc/fsck` command is an interactive file system check and repair program that uses the redundant structural information in the file system to perform several consistency checks. The `fsck` process has six possible phases; a series of error messages may appear during each phase, and you are prompted to answer `YES` or `NO` to a series of questions about the errors encountered. To assess any potential problems, you may want to answer `NO` to all questions, then rerun `fsck` after you have decided on a plan for any needed repairs. If you use the `-n` option with `fsck`, the default answer to all questions is `NO`. For example, if the `/tmp` file system is truly used as a volatile scratch area, you may not want to bother repairing any errors that `fsck` finds, in which case, you may prefer the `-n` option.

When you are prompted to clear the inode, it is sometimes best to answer `NO` first. The `fsck` command also will display the inode number and size; you can make a note of the number, and then, if you do want to clear the inode, you can rerun `fsck` and clear it.

No matter how many error messages you receive from `fsck`, and no matter how serious the errors may seem, you always can reconstruct your file system

from the last version of your back-up media. Therefore, it is absolutely critical that you have a consistent method of doing backups and that you always follow that method. If you have the backups, you can always restore your file system from the backups if all else fails.

The `fsck` program always goes through the following five phases. Phase 6 sometimes occurs if an error occurred during phase 5. Generally, each phase is a "clean up" after the previous phase.

Phase	Description
1: Check blocks and sizes	Examines the file system's inode list for duplicate blocks, incorrect block numbers, or incorrect format.
2: Check path names	Removes directory entries that were modified in phase 1.
3: Check connectivity	Checks the connectivity of the file system, verifying that each inode has at least one directory entry and creating error messages for unreferenced directories.
4: Check reference counts	Lists errors from missing or lost directories, incorrect link counts, or unreferenced files.
5: Check free list	Checks the relationship between the number of allocated blocks in the file system, the number of blocks in use, and the difference between the two (the *free block list*). If the current free block count (immediately calculated) is not the same as the free block list, an error is reported.
6: Salvaging	Occurs only if an error occurred in phase 5 and you answered YES to the SALVAGE? prompt.

You must become familiar with using `fsck` and become comfortable replying to the `fsck` error messages.

If a file system was unmounted cleanly, `fsck` responds with the following message and does not perform the file system check:

```
/dev/dsk/usr01: Filesystem check bypassed
```

If an inconsistency is detected, fsck reports this in the same window in which the command was invoked and will ask whether the inconsistency should be fixed or ignored. The `/etc/fsck` command can often repair a corrupted file system.

The `/etc/fsck` command also checks for orphan files (files not connected to the `root` inode of the file system). A scan is done of all unaccounted blocks in the file system. Each block is checked for the inode magic number. If it is found, blocks that are claimed by the inode are checked to see whether they are valid and do not duplicate block numbers. If this step is accomplished safely, a prompt will appear that will ask whether you want the inode to be salvaged, which you probably will want to do.

Example:

```
/etc/fsck /dev/dsk/usr01
```

For a complete description of all parameters, see the fsck(8) man page.

> **Note:** A file system can become corrupted in a variety of ways, the most common of which are hardware failures and improper shutdown procedures. If you do not follow proper startup procedures, a corrupted file system will become further corrupted.

A hardware failure can occur because of the following:

- Disk pack error
- Controller failure
- Power failure

An improper system shutdown can occur because of the following:

- Forgetting to `sync` the system prior to halting the CPU
- Physically write protecting a mounted file system
- Taking a mounted file system offline

If you do not use `fsck` to check a file system for inconsistencies, an improper system startup can occur.

The `/etc/fsck` command primarily detects and corrects corruption of the following two types:

- **Improper file creation:** When a user creates a UNICOS file, the system goes through the following four basic steps:

  1. Allocates an inode from the inode region.
  2. Makes a directory entry, and places the new inode number and file name in the directory.

3. Allocates any data blocks as needed.
4. Increments the link count in the inode for the file. If this is a directory file, the system also increments the link count for the parent directory.

If the system cannot complete all four steps successfully, file system errors will occur.

- **Improper file removal:** When a file is removed using the `rm`(1) command, the system proceeds in reverse order, as follows:

  1. Decrements the link count in the inode for the file. If this is a directory file, the system also decrements the link count for the parent directory.
  2. Deallocates the data blocks (if the file's link count is 0).
  3. Removes the directory entry.
  4. Deallocates the inode (if the file's link count is 0).

  If the system cannot complete all four steps successfully, file system errors will occur.

Because a file might be linked to several different directory entries, the inode and data blocks are removed only when the last link is removed.

Continue with step 4.

4. Creating a mount point for the file system

If a mount point does not exist already for a file system, use the `/bin/mkdir` command to create one. Typically, the mount point is given the same name as the logical device name of the file system on which it will be mounted. For example, if a logical device named `/usr/home` has been configured in the IOS `/sys/param` file, the mount point also will be named `/usr/home`. You can create this mount point as shown in the following example.

Example:

```
# mkdir /usr/home
```

Note: The contents of the mount point directory are hidden when a file system is mounted on top of it.

Continue with step 5.

1. Mounting the file system

A file system is a sequential array of data until it is mounted. When the file system is mounted, the UNICOS kernel interprets the data as a UNICOS file system that is available as part of the system's complete directory structure. To be accessible to the UNICOS system, all file systems except `root` (`/`) must first be explicitly mounted by using the `mount`(8) command. The file system is mounted on an existing directory. The directory may have to be created, using the `mkdir`(1) command (see step 4). By convention, the name of the directory corresponds to the name of the logical device. The fourth field of the `/etc/fstab` file controls the automatic mounting of user file systems when going to multiuser mode. (For steps to configure a file system to be mounted automatically at initialization of multiuser mode, see Procedure 8, page 131.)

The system keeps a table of mounted file systems in memory and writes a copy of the table to `/etc/mnttab`. `root` is always available to the system and is entered into `/etc/mnttab` at boot time through `/etc/brc`. The `root` inode of the mounted file system replaces the mount-point inode in memory; therefore, any files in the mount-point directory are unavailable while the file system is mounted. That is, you should use only an empty directory as a mount point.

Note: You **must** check the file system by using the `fsck` command **before** it is mounted (see step 3).

Example:

```
# /etc/mount /dev/dsk/home /usr/home
```

For a complete description of all options, see the `mount`(8) man page.

Note: Check the permission of the mounted file system. To change the permission of the root directory of the mounted file system, if necessary, use the `chmod` command (see the `chmod`(1) man page).

5.14 `/etc/mnttab` and `/etc/fstab` files

The `/etc/mnttab` and `/etc/fstab` files are related to the condition of whether a file system is mounted or unmounted.

5.14.1 `/etc/mnttab`

The `/etc/mount` and `/etc/umount` commands maintain the `/etc/mnttab` file. Two tables keep track of mounted disk devices. The one maintained internally by the UNICOS kernel is always correct. The other, `/etc/mnttab`, is

maintained as a convenience for such scripts as `/etc/mount`, which, when issued without any arguments, will display the list of all currently mounted file systems.

When a file system is mounted (using the `/etc/mount` command), an entry is made in the `/etc/mnttab` file. When a file system is unmounted (using the `/etc/umount` command), the entry that corresponds to that file is removed from the `/etc/mnttab` file.

5.14.2 `/etc/fstab`

The system administrator maintains the `/etc/fstab` file. When you set up an `/etc/fstab` file, it has the following four primary purposes:

Note: The `/etc/fstab` file provides a way to mount user file systems automatically whenever the system is brought up to multiuser mode. For any file system that you want the `/etc/rc` script to mount automatically, set the fourth field of the `/etc/fstab` file for that entry to `CRI_RC=YES`.

- It contains a list of files that the start-up `/etc/bcheckrc` script checks by invoking the `/etc/mfsck` command, which does multiple synchronous file system checks (`/etc/fsck`).
- It allows a shortcut to be taken by using the `/etc/mount` command. When a mount command is invoked with only a special file name or only a mount point specified rather than both, the `/etc/fstab` file is searched for the missing arguments. For example, if you entered the `/dev/dsk/usr01` file system information in the `/etc/fstab` file, instead of typing the following command:

  ```
  /etc/mount /dev/dsk/usr01 /usr01
  ```

  you can type one of the following commands instead:

  ```
  /etc/mount /usr01
  ```

  ```
  /etc/mount /dev/dsk/usr01
  ```

- It provides a convenient way to mount file systems with file system quotas enforced.

For descriptions of the `fstab` fields, see the `fstab`(5) man page.

Procedure 8: Configuring a file system to be mounted automatically at the initialization of multiuser mode

If you want any file system to be mounted automatically when multiuser mode is initialized, you must edit the `/etc/fstab` file. Because the `/etc/fstab` file may have read-only permission, you must check the permissions on the file before you try to edit it to ensure that the file has write permission (see step 1). The system can be in either single-user or multiuser mode.

If the system is in single-user mode and the only file system available is `root` (`/`), the only available editor is the `ed` editor. The `vi` editor is located in the `/usr` file system, which is not mounted. If you check (using `fsck`) and mount (using `mount`) the `/usr` file system, the `vi` editor will be available to you even though you are in single-user mode. If the system is in multiuser mode, the `vi` editor is available and can be used to edit the `/etc/fstab` file.

1. Edit the `/etc/fstab` file by using either the `ed` editor or the `vi` editor.

When trying to edit a file, you may encounter a message that a file is "read only." One solution is to change the permissions of the file so that it can be edited, then return the permissions to their original settings when you are finished making changes. The example shown uses the `/etc/config/rcoptions` file.

```
#ls -la /etc/config/rcoptions
-r--r--r-- 1 root root 1914 Mar  8 11:29 /etc/config/rcoptions
# chmod 644 /etc/config/rcoptions
-rw-r--r-- 1 root root 1956 Mar  8 17:28 rcoptions
# vi rcoptions

(make changes)

# chmod 444 /etc/config/rcoptions
-r--r--r-- 1 root root 1914 Mar  8 11:29 /etc/config/rcoptions
```

If you are using the `vi` editor, you can accomplish the same effect by making your changes to the file and, from within the `vi` editor, typing the following command, which forces a write to the file:

```
<escape>:w!
```

2. Uncomment the line for any file system already mentioned in the `/etc/fstab` file that you want to be checked and mounted automatically when the system goes to multiuser mode, or add a line (with the appropriate format) for the desired file system if it is not already mentioned.

3. Edit the fourth field for the desired file system entry to read `CRI_RC=YES`.
4. Save the changes you have made to the `/etc/fstab` file.

Procedure 9: Unmounting file systems

At shutdown, the `/etc/shutdown` script unmounts all file systems; however, you may want to make a file system unavailable during normal operation for maintenance purposes. By convention, the `/mnt` directory is used to mount a file system that needs maintenance. To make a file system unavailable to users, unmount it by using the `umount` command. *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022.

Caution: The file system must be idle before you can unmount it. To determine whether the file system is idle, use the `/etc/fuser` utility.

The argument you specify on the `/etc/umount` command line can be either the name of the mount point or the special device name for the file system you want to unmount.

Examples:

```
# /etc/umount /usr01
```

```
# /etc/umount /dev/dsk/usr01
```

You also can use the `/etc/umountem` script to unmount all file systems quickly while in single-user mode. It executes the `/etc/mount` command to receive a list of the file systems that are currently mounted, edits the list to produce a script of `/etc/umount` commands, and then executes the script. The `umount` command flushes the file system cache to the disk before actually unmounting the file system.

```
# /etc/umountem
```