ChorusOS 4.0 File System Administration Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part Number 806-0612
November 1999

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227–14(g)(2)(6/87) and FAR 52.227–19(6/87), or DFAR 252.227–7015(b)(6/95) and DFAR 227.7202–3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Contents

Preface

ChorusOS 4.0 File System Administration Guide describes how to set up and use local and network file systems supported by ChorusOS™ 4.0. It details system image configuration, setting up devices attached to the ChorusOS system to support file systems, running NFS services on the ChorusOS system, mounting and unmounting file systems on the ChorusOS system, and related tasks.

This document does not describe how to run NFS services on your host workstation or boot server.

Who Should Use This Guide

This document is written for both ChorusOS 4.0 users and system administrators.

Before You Read This Guide

You must be familiar with the concepts explained in *ChorusOS 4.0 Introduction*.

You must also have an operational ChorusOS 4.0 system including at least a target and a host. For more information, see *ChorusOS 4.0 Installation Guide*.

How This Guide Is Organized

Chapter 1 presents the process of preparing file systems and discusses some important aspects of system initialization that concern file systems.

Chapter 2 explains how to configure a system image to include support for file systems and related hardware.

Chapter 3 details how to set up file systems on devices that are physically attached to the ChorusOS system.

Chapter 4 describes how to set up and run an NFS server on the ChorusOS system.

Chapter 5 explains how to add and remove file systems within the existing file system hierarchy.

Chapter 6 provides examples.

Related Reading

- *ChorusOS 4.0 Introduction* introduces the features and components of ChorusOS systems. It explains how to use the ChorusOS 4.0 product and how to create an application that runs on a ChorusOS system.
- The *ChorusOS 4.0 Device Driver Framework Guide* describes the device driver architecture of the ChorusOS system and explains how to add a new driver.
- The *ChorusOS 4.0 Hot Restart Guide* describes how to develop applications to use the hot restart functionality of the ChorusOS 4.0 product.
- The *ChorusOS 4.0 Installation Guide for Solaris Hosts* explains how to install the ChorusOS 4.0 product on hosts running the Solaris™ operating environment.
- The *ChorusOS 4.0 Installation Guide for Windows NT Hosts* explains how to install the ChorusOS 4.0 product on hosts running Windows NT 4.0.
- The *ChorusOS 4.0 Network Administration Guide* details how to use the networking capabilities of ChorusOS 4.0.
- The *ChorusOS 4.0 Porting Guide* explains how to port the ChorusOS system to another target board.
- The *ChorusOS 4.0 Reference Manual Collection* contains descriptions of the functionality available in ChorusOS 4.0.
- The *ChorusOS 4.0 Target Family Documentation Collection* includes documentation detailing functionality specific to the reference target family architectures supported for the ChorusOS 4.0 product.

- The XRAY Debugger documentation from Mentor Graphics explains how to debug a ChorusOS application. XRAY is the reference debugger for use with ChorusOS systems.

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at `http://www1.fatbrain.com/documentation/sun`.

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

What Typographic Conventions Mean

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	**Meaning**	**Example**
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls –a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`**
		`Password:`

TABLE P–1 Typographic Conventions *(continued)*

Typeface or Symbol	Meaning	Example
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **rm** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	machine_name%
C shell superuser prompt	machine_name#
Bourne shell and Korn shell prompt	$
Bourne shell and Korn shell superuser prompt	#

CHAPTER **1**

Introduction

This chapter addresses particularities of ChorusOS system initialization that concern file systems. It also presents the process of managing file systems on ChorusOS systems.

Note - In order for you to be able to include file system support in your ChorusOS system, you should install the OS component of the ChorusOS 4.0 product during installation on the host workstation, and you should `configure`(1CC) the host workstation directory where you build system images to include binaries from the OS component.

In this guide, we assume you build file IO support and the `C_INIT`(1M) actor into your system image.

1.1 System Initialization and File Systems

The `C_INIT`(1M) actor plays an important role in part of system initialization that concerns file systems. During target system initialization, it mounts a *pseudo* root node, `/`, on a `pdevfs` file system for internal use. It then creates `/dev` and `/image` directories. At this point, it creates a `/dev/console` device so it can write to the system console and special files in the `/dev` directory that correspond to the devices attached to the device tree. It also creates special files to access the memory banks and mounts the contents of the system image in `/image/sys_bank` as a FAT file system containing configuration files and bootable archives. It mounts other memory banks under `/image/`*bank_identifier* as well, if any others are available. Therefore, you should not be surprised to see several file systems mounted immediately after the system comes up:

```
$ rsh target mount
root_device on / (pdevfs)
devfs on /dev (pdevfs)
devfs on /image (pdevfs)
/dev/bd00 on /image/sys_bank (msdos)
```

Next, `C_INIT`(1M) looks for `sysadm.ini`(4CC), the system initialization file. Usually, it finds the system initialization file in `/image/sys_bank/sysadm.ini`, although it does this by probing the memory banks for the file and using the first instance it finds. (If it finds more than one `sysadm.ini` file, it displays a warning on the system console.) Finally, `C_INIT`(1M) executes the commands it finds in `sysadm.ini`, such as commands to set up network connections and commands to create special files for access to local media, such as hard disks, RAM disks and flash memory.

At this point, you will usually want to mount a root file system that contains actors you want to run on your target system. You do this *without unmounting the pseudo root node*. The ChorusOS system allows you to mount a new root while the *pseudo* root is still mounted.

Unless you build everything into your system image or mount a root directory on a device connected to the target system, you will probably mount *build_dir*`/root` on the host workstation as the root directory for your target system. You generate this directory on the host workstation using the `make root` command in the directory where you build system images.

Often, you include the command to mount the root file system in `sysadm.ini`(4CC). The following example however demonstrates what happens when you mount a new root file system exported through NFS from a host workstation:

```
$ rsh target mount
root_device on / (pdevfs)
devfs on /dev (pdevfs)
devfs on /image (pdevfs)
/dev/bd00 on /image/sys_bank (msdos)
$ rsh target mount host_IP_address:/export/target/root /
host_IP_address:/export/target/root on / (nfs)
$ rsh target mount
root_device on / (pdevfs)
devfs on /dev (pdevfs)
devfs on /image (pdevfs)
/dev/bd00 on /image/sys_bank (msdos)
host_IP_address:/export/target/root on / (nfs)
$ rsh target arun /bin/ls /
started aid = 2
Makefile        bin             etc             dev            image
lib
$ rsh target umount /
$ rsh target mount
root_device on / (pdevfs)
```

(continued)

(Continuation)

```
devfs on /dev (pdevfs)
devfs on /image (pdevfs)
/dev/bd00 on /image/sys_bank (msdos)
```

See the host workstation documentation for details about sharing this directory through NFS. If your host workstation is running the Solaris™ operating environment, you can probably use the `share`(1M) command to share the file system.

1.2 The Process of Managing File Systems

It also helps to understand beforehand what you must do to set up file systems on a ChorusOS system and in what order to perform the necessary steps. The following list summarizeds the stages of file system administration for ChorusOS systems.

1. Configuring the system image

   You must include support for file systems in the system image in order to use the functionality described in this guide. Configuring the system image for file system support involves setting features and tunables to support the media and file system types you plan to use, editing the built-in system initialization script to create special files for accessing the media, and then rebuilding the properly configured system image for use on the ChorusOS target system.

   This stage is generally completed *only once* for a given system.

2. Preparing local media

   Local media must be labelled in order to support file systems. If you have local flash memory, it must also be formatted before it is labelled.

   This stage is generally completed *only once* for each device.

3. Creating file systems

   You create new file systems on the available partitions on local media. This involves writing file system structures to devices attached to the ChorusOS system.

   This stage is generally completed *only once* for a given system.

4. Checking and mounting file systems

   Before mounting a local file system, you check it for errors. After the check is complete and any errors in the file system have been repaired, you mount the file system into the file system hierarchy. If the file system is being mounted over the network through NFS, you mount it without performing a check. Once mounted, the contents of the file system are available for use.

This stage is performed each time you use a file system.

5. (Optional) Exporting local UFS file systems through NFS

   If you have a local UFS file system that you want to make available over the network, you export it through NFS. This stage involves editing several configuration files and starting the daemon actors that make your target system an NFS server. This guide assumes you have a working network connection that makes exporting through NFS possible.

   File system export is often set up only once for a given system. Daemons may be started as part of system initialization.

6. (Optional) Unmounting file systems

   If you need to remove a file system from the hierarchy, you unmount it. This stage may be performed during normal system use, but is usually performed as part of system shutdown.

This document includes instructions to guide you through each stage of the process of managing file systems.

CHAPTER **2**

How to Configure the System Image

This chapter explains how to create a system image with support for file systems and related hardware.

Note - Before attempting to configure any of the features and tunables described below, check that they are supported by your ChorusOS system. Consult the appropriate document in the *ChorusOS 4.0 Target Family Documentation Collection* to determine which features and tunables your version supports.

For descriptions of available features and tunables, read the appropriate sections of the *ChorusOS 4.0 Introduction*. For more details concerning tunables, see *ChorusOS man pages section 5FEA: ChorusOS Features*.

In order to build a system image with file system support, you must have installed the OS component on the host workstation. If you did not install this component during initial installation, you may run the install wizard again to add the component to your existing ChorusOS 4.0 installation. See the *ChorusOS 4.0 Installation Guide for Solaris Hosts* or the *ChorusOS 4.0 Installation Guide for Windows NT Hosts* for installation instructions.

2.1 What ChorusOS Systems Support

This section reviews the file systems and related media supported for the ChorusOS 4.0 product. As stated in the note above, support for specific ChorusOS file systems and hardware devices depends on the target family.

2.1.1 Supported Media

The ChorusOS 4.0 product supports the following hardware, although not necessarily for all target platforms:

Flash memory
Flash device support is implemented using Flite 1.2. Flash support allows you to use only MS–DOS file systems on flash devices.

IDE hard disk drives
Supported hard disk drives must be connected to an IDE, ST506 or ESDI compatible disk controller.

RAM disk memory
RAM disk support allows you to create disk-like entities and use file systems in random-access memory.

SCSI CD-ROM drives
Selected SCSI CD-ROM drives and SCSI-PCI I/O processors of the NCR53C8xx family are supported.

SCSI hard disk drives
Selected SCSI hard disk drives and SCSI-PCI I/O processors of the NCR53C8xx family are supported.

ChorusOS 4.0 requires that you use special device driver files in order to read from and write to these devices. See Section 2.3 “Required Special Device Driver Files” on page 18 and `special`(7S) for details.

2.1.2 Supported File Systems

The ChorusOS 4.0 product supports the following file systems, although not necessarily for all target platforms:

Network File System (NFS)

NFS is the standard among UNIX operating systems for sharing file systems over the network. ChorusOS systems can support both NFS client access to shared file systems and NFS server capabilities to share local file systems with other systems on the network.

File Allocation Table (FAT) file system

Also known as the MS-DOS file system, this supports file allocation tables with 12, 16 or 32–bit entries, making it possible to support long file names.

UNIX File System (UFS)

Also known as the Fast File System, this supports long file names and links, and is the required type for file systems exported through NFS.

Swap file system

The ChorusOS 4.0 product also supports the use of a swap partition on supported local hardware devices.

The table below summarizes which file systems are supported for which media.

TABLE 2–1 File System Support By Media

	MS-DOS	UFS	Swap
Flash	X		
IDE	X	X	X
RAM disk	X	X	
SCSI	X	X	X

2.2 How to Add Support for File Systems and Related Hardware

▼ Adding Support Using the Graphical User Interface Tool

To configure your system image with support for file systems and related hardware through the `ews` graphical user interface, follow the procedure below.

1. **Open your system image configuration in** `ews`**:**

```
host% cd build_dir
host% ews conf/ChorusOS.xml &
```

2. **Use the hints in the following tables to set features and tunables for the file system support needed.**

   The following table lists the feature and tunable settings required for your ChorusOS system to support the media on which you use file systems.

TABLE 2–2 Media Support

To include support for...	**Set...**	**Comments**
Flash memory	FLASH=true	Flash memory supports only FAT (MS-DOS) file systems.
IDE hard disk drives	IDE_DISK=true	Enables support for hard disk drives connected to compatible IDE, ST506 and ESDI controllers.

RAM disk memory	RAM_DISK=true
	iom.ramdisk*X*.size= *size_in_hex*
	iom.ramdisk.sizeMax= *max_RAM_disk_size*
		iom.ramdisk*X*.size, where *X* is a hexadecimal digit `0`, `1`, `2`, ..., `d`, `e`, `f`, takes a size in the same hexadecimal format as the value for iom.ramdisk.sizeMax. For example, if you want to have one 4MB RAM disk use iom.ramdisk0.size= 0x40000.

SCSI hard disk drives	SCSI_NCR53C8xx=true
	SCSI_DISK=true
		Enables support for SCSI-PCI I/O processors of the NCR53C8xx family.

SCSI CD-ROM drives	SCSI_NCR53C8xx=true
	SCSI_CDROM=true
		Enables support for SCSI-PCI I/O processors of the NCR53C8xx family.

The following table summarizes the feature and tunable settings required for your ChorusOS system to support specific file systems.

TABLE 2–3 File System Support

To include support for…	Set…	Comments

FAT (MS-DOS) file systems	MSDOSFS=true
	iom.nbuf=32 (at least)	

NFS client capability (for mounting NFS file systems)	NFS_CLIENT=true
	iom.nbuf=8 (at least)	

NFS server capability (for exporting file systems)	NFS_CLIENT=true
	NFS_SERVER=true
	iom.kmemsize=0x80000 (at least)
	iom.nbuf=8 (at least)
		The ChorusOS system can export only local UFS file systems through NFS.
		MS-DOS and swap file systems *cannot be exported* through NFS.

Swap file system	FS_MAPPER=true
	ON_DEMAND_PAGING=true
		Swap may only be used on local media.
		Only one swap device may be used.
		Swap is only available in the VM memory model, where VIRTUAL_ADDRESS_SPACE=true.

UNIX (UFS) file systems	UFS=true
	iom.nbuf=8 (or more)
		UFS file systems cannot be used on flash media.

The `iom.nbuf` tunable sets the number of buffer cache entries; the standard buffered entries used for input and output. The larger the value, the larger the cache available. The `iom.nbuf` tunable influences the amount of RAM used by the system.

The `iom.kmemsize` tunable sets the amount of memory available to the kernel.

3. **Save your changes to the system image configuration.**

4. **Build the system image.**

▼ Adding Support Using Command-Line Tools

To configure your system image with support for file systems and related hardware through the `configurator`(1CC) command-line interface, follow the procedure below.

1. **Change to the directory where you build system images:**

   ```
   host% cd build_dir
   ```

2. **Use the hints in the table above, see Step 2 on page 16, to set features and tunables for the file system support needed.**

   `configurator` commands to set features and tunables take the form:

   ```
   host% configurator -c conf/ChorusOS.xml -set feature | tunable=value
   ```

3. **Build the system image to include the modifications you have made:**

   ```
   host% make system_image_name
   ```

2.3 Required Special Device Driver Files

This section reviews the special device driver files required for file system support.

If your target is an NFS client only (all its files are physically located on another system, such as the host workstation), you can omit this section.

2.3.1 What Special Files Are

The ChorusOS system requires you to use `special`(7S) device driver files to access the hardware devices where file systems reside. This means that disk labelling and other operations on uninitialized and unmounted file systems must be done using special files.

Each disk partition corresponds to at least one special file. Unless you plan to use a raw partition directly, without a file system, you must be able to access each partition in both block (buffered) mode and raw (character) mode, so you must create not just one special file per partition, but two. Each special file:

- Refers to either a block or a raw device. Block devices are used by file systems. Raw devices are used mainly for file system administration.
- Has a *major* number. Major numbers are used by the system to select the corresponding device driver when several devices are configured. Major numbers are the same for all devices managed by a given device driver and a given access method (raw or block). For example, all devices corresponding to hard disk partitions using raw mode have the same major number.
- Has a *minor* number. Minor numbers are not used directly by the system, but by the selected device driver. Minor numbers are different for each partition on a device. Their scope is limited to the device described by the major number, so special files with different major numbers may share the same minor number. One minor number corresponds to one partition.

2.3.2 Naming Conventions for Special Files

Special files normally reside in the `/dev` directory, which is mounted at boot time. By convention, special file names follow the form `/dev/r`*suffix* for raw (character) mode and `/dev/`*suffix* for buffered (block) mode.

The *suffix* is made up of:

- A string of letters referring to the device driver name, such as `sd` for a SCSI disk, `rd` for a RAM disk, `hd` for an IDE disk or `flash` for flash,
- Followed by a digit representing the disk unit number, such as `0`, `1`, `2` and so forth,
- Terminated by a single letter referring to the partition, such as `a`, `b`, ... `h`.

Caution - Special care must be taken with partition `c`. Partition `c` represents the whole disk and therefore *must not be used* to support a file system.

As file systems are based on BSD 4.4 as implemented in FreeBSD 2.2.7, the same limitations found in FreeBSD 2.2.7 apply to ChorusOS file system management. According to limitations imposed by FreeBSD, a disk can be divided into a maximum of eight different partitions for IDE and SCSI devices, two partitions for RAM and flash devices. Partitions can be left undefined. Partitions are named using a single character in the range from `a` to `h`, each letter corresponding to one of the eight partitions for IDE and SCSI devices. For RAM and flash devices, only partitions `a` and `c` are available.

2.4 How to Create Special Files

You create special files using `mknod`(1M) on the ChorusOS system. Generally, you create the special files you need at boot time by including commands in the system initialization file, `sysadm.ini`(4CC).

Note - Unlike earlier releases that used special device driver files created on the host, the ChorusOS 4.0 product only lets you create special files on the target. Previous releases allowed you to create special files on the host because no `/dev` directory was available at boot time. As the ChorusOS 4.0 product mounts a `/dev` directory at boot time, it is no longer necessary to create special files on the host.

▼ Creating Special Files at Boot Time

To create special files at boot time using the `sysadm.ini`(4CC) file embedded in the system image, follow the procedure below:

1. **Change to the directory containing** `sysadm.ini`**:**

```
host% cd build_dir/conf
```

2. **Include commands of the following form in** `sysadm.ini` **using the** `C_INIT`**(1M) built-in command** `mknod`**(1M):**

```
mknod /dev/name [b|c] maj_nbr min_nbr
```

where *name* follows the pattern described above in Section 2.3.2 “Naming Conventions for Special Files” on page 19, `b` represents a buffered (block) device, `c` represents a character (raw) device, *maj_nbr* is the major number of the device and *min_nbr* is the minor number of the partition on the device. The following table lists memory devices by major number:

Major Number	Device	Mode
3	ISA/IDE disk	character (raw)
4	ISA/IDE disk	block (buffered)
7	Flash device	character (raw)

Major Number	Device	Mode
8	Flash device	block (buffered)
9	SCSI disk	character (raw)
10	SCSI disk	block (buffered)
13	RAM disk	character (raw)
14	RAM disk	block (buffered)
15	SCSI CD-ROM	character (raw)
16	SCSI CD-ROM	block (buffered)

Note that RAM disk devices used for memory banks conventionally have major numbers `11` and `12` and are used internally by the system to make the contents of the memory banks, including the system image, available for use at boot time.

3. **Build the system image to include modifications to** `sysadm.ini`**:**

```
host% cd ..
host% make system_image_name
```

Creating Special Files Manually

To create special files manually on a running ChorusOS target system:

♦ **Run commands on the target of the type:**

```
rsh target mknod /dev/name [b|c] maj_nbr min_nbr
```

where *name* follows the pattern described above in Section 2.3.2 “Naming Conventions for Special Files” on page 19, `b` represents a buffered (block) device, `c` represents a character (raw) device, *maj_nbr* is the major number of the device and *min_nbr* is the minor number of the partition on the device.

CHAPTER **3**

How to Set Up File Systems On the Target

This chapter explains how to set up file systems on devices physically attached to the target system, such as Flash memory, IDE disks, RAM disks and SCSI disks, and CD-ROMs. If your target is an NFS client only (all its files are physically located on another system such as the host workstation), or if the only local file system devices you intend to access are SCSI CD-ROM drives, you can omit this chapter.

Before you can perform the procedures in this chapter, you must first configure the system image. See Chapter 2 for details.

You must also boot the new system image on the target, and mount the root file system where many of the useful actors reside. This usually involves mounting a file system located on the host workstation where you built the system image. See "Mounting an NFS File System" on page 33 for details.

Setting up file systems on the target involves:

1. Preparing the media on which you will use file systems.

   If the media used is flash memory, you must first `format`(1M) the flash device.

   You must label the media using the `disklabel`(1M) utility, which writes partition information onto the media based on entries in the `disktab`(4CC) file.

2. Creating file systems on the media.

   The `newfs_dos`(1M) utility lets you create an MS-DOS file system.

   The `newfs`(1M) utility lets you create a UFS file system.

3. Checking file system integrity.

   The `fsck_dos`(1M) utility lets you check an MS-DOS file system.

   `fsck`(1M) utility lets you check a UFS file system.

3.1 How to Format a Flash Memory Device

You must format Flash memory before you can label it. If your target does not have Flash memory, you can omit this section.

▼ Formatting a Flash Memory Device

1. **Reboot the target system with a system image that supports Flash memory and includes the special files needed to access it.**

2. **Format the Flash memory using the** `format` **command:**

```
host% rsh target arun /bin/format /dev/rflash0a
```

3.2 How to Label a Disk

This section describes how to label a disk using information in `/etc/disktab` and the `disklabel` utility.

Caution - If your disk has already been formatted, partitioned (using MS-DOS tools such as `fdisk` and `format`) and already contains valuable data you want to use, do *not* label the disk. Go to the next chapter instead.

The ChorusOS system recognizes the primary partitions in the existing DOS partition table, and uses the partition table rather than the disk labels described below. ChorusOS systems recognize only primary DOS partitions.

Labelling consists of writing specific information, such as disk geometry in terms of cylinders, heads, sectors per track and partition overlays, at particular locations on the disk. Disk drivers use labelling information to access different areas of the disk, called partitions.

3.2.1 How to Set Up Information About Disk Geometry

Information about disk geometry is found in `/etc/disktab`, which `disklabel` reads before writing a label to a drive. ChorusOS 4.0 provides a sample `/etc/disktab` that contains several useful examples. The following key to the abbreviations used in `/etc/disktab` is included at the top of the file:

```
#
# Disk geometry and partition layout tables.
# Key:
#
#   dt      controller type
#   ty      type of disk (fixed, removeable, simulated)
#   d[0-4]  drive-type-dependent parameters
#   ns      #sectors/track
#   nt      #tracks/cylinder
#   nc      #cylinders/disk
#   sc      #sectors/cylinder, ns*nt default
#   su      #sectors/unit, sc*nc default
#   se      sector size, DEV_BSIZE default
#   rm      rpm, 3600 default
#   sf      supports bad144-style bad sector forwarding
#   sk      sector skew per track, default 0
#   cs      sector skew per cylinder, default 0
#   hs      headswitch time, default 0
#   ts      one-cylinder seek time, default 0
#   il      sector interleave (n:1), 1 default
#   bs      boot block size, default BBSIZE
#   sb      superblock size, default SBSIZE
#   o[a-h]  partition offsets in sectors
#   p[a-h]  partition sizes in sectors
#   b[a-h]  partition block sizes in bytes
#   f[a-h]  partition fragment sizes in bytes
#   t[a-h]  partition types (file system, swap, etc)
#
# All partition sizes reserve space for bad sector tables.
# 5 cylinders are needed for maintenance including
# replacement sectors.
#
…
```

Entries in `/etc/disktab` consist of fields separated by colons (":") and follow the form:

label:*option*[=*value*|#*value*]…

where *label* is a string identifier up to eight characters long with no whitespace, *option* is an option from the list of abbreviations above, and *value* is the value assigned to an option.

For detailed examples, see the `root/etc/disktab` file generated using `make root` in the *build_dir* directory where you build system images. Note that *build_dir*`/root` is normally the directory exported for use as the target root directory. See "Mounting an NFS File System" on page 33 for details.

If you are unable to complete the disk geometry fields, boot the ChorusOS system image on the target system, and read the output concerning the disk driver displayed on the system console at boot time.

3.2.2 How to Use `disklabel`

Once you have correctly specified disk information in `/etc/disktab`, you are ready to use `disklabel`.

▼ Labelling a Disk

1. **Boot the target system with a system image that supports the hardware you want to label and that includes the special files needed to access that hardware.**

2. **Update the disk label on the device, using** `disklabel` **as follows:**

   ```
   host% rsh target arun /bin/disklabel -w -r device label name
   ```

   where *device* is an abbreviated form of the device name such as `sd0`, *label* is the label found in `/etc/disktab` and *name* is an optional string identifier up to sixteen characters long with no whitespace.

3. **Check that the disk label is correctly updated, using** `disklabel` **as follows:**

   ```
   host% rsh target arun /bin/disklabel -r device
   ```

   where *device* is an abbreviated form of the device name such as `sd0`.

3.3 How to Create a File System

This section explains how to create a file system on a disk. If you plan to use the disk partitions in raw mode and do not need a file system, you can omit the rest of this chapter.

Once the disk has been labelled, you can write a UFS or MS-DOS file system structure to any partition you have defined for the disk, except partition `c`.

▼ Creating a UFS File System

- ♦ **Create the file system using the `newfs` command as follows:**

```
host% rsh target arun /bin/newfs raw_device
```

where *raw_device* is a raw mode special file indicating a partition such as `/dev/rsd0a`.

▼ Creating an MS-DOS File System

- ♦ **Create the file system using the `newfs_dos` command as follows:**

```
host% rsh target arun /bin/newfs_dos raw_device
```

where *raw_device* is a raw mode special file indicating a partition such as `/dev/rhd0a`.

EXAMPLE 3–1 Creating a File System on a RAM Disk

You can pass extra parameters to `newfs` or `newfs_dos` when creating a file system on a RAM disk in order to maximize available space. The following command could be used to maximize space available on a 1 MB RAM disk:

```
host% rsh target arun /bin/newfs -o space -c 128 -m 0 /dev/rrd0a
```

3.4 How to Check File System Integrity

This section explains how to check existing file systems for errors.

Note - Before mounting a local file system with read-write access or as the root file system, and after any system crash, it is *strongly recommended* that you check the file systems you plan to mount.

▼ Checking a UFS File System

♦ **Check the file system using the** `fsck` **command as follows:**

```
host% rsh target arun /bin/fsck raw_device
```

where *raw_device* is a raw mode special file indicating a partition such as `/dev/rsd0a`.

▼ Checking an MS-DOS File System

♦ **Create the file system using the** `fsck_dos` **command as follows:**

```
host% rsh target arun /bin/fsck_dos raw_device
```

where *raw_device* is a raw mode special file indicating a partition such as `/dev/rhd0a`.

If no errors are found, both `fsck` and `fsck_dos` display information on the file system use. In case of errors, both `fsck` and `fsck_dos` attempt a recovery procedure, optionally requiring confirmation. If you want to run either command in non-interactive mode, use the `–y` option.

CHAPTER 4

How to Share Target File Systems Over NFS

This chapter explains how to set up an NFS server on the target system, allowing you to share files located on devices attached to the target with other systems on the network. If your target is an NFS client only (all its files are physically located on another system such as the host workstation), or if you do not intend to export target file systems, you can omit this chapter.

Before you can perform the procedures in this chapter, you must first configure the system image. See Chapter 2 for details.

The NFS server can only export UFS file systems. For details about creating a UFS file system on the target, see Chapter 3.

Setting up the target as an NFS server involves:

- Editing the network and NFS configuration files, `exports`(4CC), `hosts`(4CC), `networks`(4CC) and `netgroup`(4CC)
- Starting the `inetNShost`(1M) name server on the target
- Starting the NFS daemons, `portmap`(1M), `mountd`(1M) and `nfsd`(1M), on the target.

Setting Up the Configuration Files

Unless you build the files below into your system image, they probably reside in *build_dir*`/root/etc`, which you generate using the `make root` command in the directory where you build system images.

1. **Make sure** `/etc/exports` **specifies all the file systems located on the ChorusOS system that you plan to export through NFS. Entries in the** `/etc/exports` **file follow the form:**

```
export_dir options
```

   where *export_dir* indicates a file system to export, and *options* allows you to specify whether any directory under *export_dir* can be mounted by another system, which users, groups or netgroups have access to *export_dir*, whether the directory should be exported read-only and so forth. See `exports`(4CC) for details.

2. **Make sure** `/etc/hosts` **contains the IP addresses and hostnames of NFS clients. Entries in the** `/etc/hosts` **file follow the form:**

```
IP_address hostname [other_hostname ...]
```

   where *IP_address* is the IP address of the system, *hostname* is the hostname of the system and *other_hostname*s are alternate hostnames. See `hosts`(4CC) for details.

3. **Make sure** `/etc/netgroup` **is readable by everyone, and contains descriptions of all** *netgroups* **— sets of hosts, users and domains — used in** `/etc/exports`**. Entries in the** `/etc/netgroup` **file follow the form:**

```
netgroup (hosts,users,domains)
```

   where *netgroup* is the name of the group, *hosts* specifies the hostnames of the systems in the group, *users* specifies the users in the group, and *domains* specifies the domains that belong to the group. See `netgroup`(4CC) for details.

4. **Make sure** `/etc/networks` **contains correct information about available networks. Entries in the** `/etc/networks` **file follow the form:**

```
network_name network_number network_aliases
```

   where *network_name* is the primary name for the network, *network_number* is the IP network number prefix, such as `127` for loopback or `192.33.15` for a specific class-C network, and *network_aliases* are alternate names for the network. See `networks`(4CC) for details.

Starting the Basic Name Server (`inetNShost`*)*

The basic name server provides the `gethostbyname`(3STDC) service to NFS daemons. It runs on the target and requires access to `/etc/hosts` and `/etc/networks`.

1. **Make sure that `/etc` is visible from the target system:**

   ```
   host% rsh target arun /bin/ls /etc
   ```

   If you have not yet mounted a root file system, see Section 5.1 “How to Mount and Unmount File Systems” on page 33.

2. **Run the name server in the background:**

   ```
   host% rsh target arun /bin/inetNShost &
   ```

   See the *ChorusOS 4.0 Network Administration Guide* and `inetNS`(1M) for details on more advanced name servers.

Starting the NFS Daemons

The `portmap` daemon must be running to handle conversion of RPC calls, the `mountd` daemon listens for remote mount requests from other systems and the `nfsd` daemon provides remote NFS services to NFS client systems.

1. **Make sure that `/etc` is visible from the target system:**

   ```
   host% rsh target arun /bin/ls /etc
   ```

   If you have not yet mounted a root file system, such as *host*:*build_dir*/`root`, see Section 5.1 “How to Mount and Unmount File Systems” on page 33.

2. **Run the `portmap` daemon in background:**

   ```
   host% rsh target arun /etc/portmap &
   ```

3. **Run the `mountd` daemon in background:**

   ```
   host% rsh target arun /etc/mountd &
   ```

4. **Run the `nfsd` daemon in background:**

   ```
   host% rsh target arun /etc/nfsd -ut -n 3 &
   ```

   The above command creates three servers for TCP and UDP clients.

5. **Check that NFS is one of the TCP/UPD services available from the target. On a host workstation running the Solaris operating environment, the following example displays the registered RPC services available on the target:**

```
host% /bin/rpcinfo -p target
```

CHAPTER **5**

File System Commands

This chapter explains how to mount and unmount file systems. It also reviews useful file system utilities.

Before you can use most of the commands described in this chapter, you must first configure your hardware, system image and environment as described in previous chapters.

5.1 How to Mount and Unmount File Systems

The `mount` command—a `C_INIT`(1M) built-in command—allows you either to add a file system at a given point in an existing file system hierarchy, or to view all mounted file systems. Once mounted, a file system is fully operational. Applications can access it using the API provided.

The `umount` command, another `C_INIT` built-in command, allows you to remove one or more mounted file systems from the file system hierarchy.

▼ Mounting an NFS File System

♦ **Mount the file system under** *mount_dir* **on the host at** *mount_point* **on the target system:**

```
host% rsh target mount host:mount_dir mount_point
```

EXAMPLE 5–1 Mounting a Root Directory through NFS

For example:

```
host% rsh target mount host:/export/chorus/root /
```

Mounts the ChorusOS root directory, `/export/chorus/root`**, on** `host` **as the root directory on** `target`**.**

▼ Mounting a UFS File System

Note - When you mount a UFS file system as the **root** file system, `mount` first mounts the file system read-only. Next use `fsck` to check the file system. If the file system check finds and repairs errors, `fsck` then calls `mount` with the `update` option to mount the file system read-write.

---

1. **Mount the file system on the partition** *block_device* **on the host at** *mount_point*:

```
host% rsh target mount -t ufs block_device mount_point
```

2. **If you have mounted the file system as the root file system, check it using the** `fsck` **command:**

```
host% rsh target arun /bin/fsck raw_device
```

where *raw_device* is a raw mode special file indicating the partition you mounted as the root file system.

EXAMPLE 5–2 Mounting a UFS File System as the Root File System

For example:

```
host% rsh target mount -t ufs /dev/sd0a /
host% rsh target arun /bin/fsck /dev/rsd0a
host% rsh target mount -t ufs -o update /dev/sd0a /
```

mounts the UFS file system on the first partition on the first SCSI drive attached to to the ChorusOS system, `target`**, as the root file system.**

Mounting an MS-DOS File System

- **Mount the file system on the partition** *block_device* **on the host at** *mount_point*:

```
host% rsh target mount -t msdosfs block_device mount_point
```

EXAMPLE 5–3 Mounting an MS-DOS File System as the Root File System

For example:

```
host% rsh target mount -t msdosfs /dev/hd0a /
```

mounts the MS–DOS file system on the first partition on the first IDE drive attached to the ChorusOS system, `target`**, as the root file system.**

Viewing all Mounted File Systems

- **Use the** `mount` **command without any arguments to view all mounted file systems:**

```
host% rsh target mount
```

Unmounting a File System

- **Use the** `umount` **command to remove** *mounted_fs* **from the file system hierarchy:**

```
host% rsh target umount mounted_fs
```

5.2 How to Activate a Swap Partition

This section explains how to activate a swap partition *on a local disk* that has already been labelled. ChorusOS 4.0 supports a single swap partition on a local disk, unlike earlier releases that supported swap over NFS.

In order to use a swap partition on a ChorusOS system, you must first label the partition as a swap partition using `disklabel`. After the partition is labelled, you mount a swap directory using the `mount` command. Finally, you activate the swap partition using the `swapon` command.

Note - Swap cannot be disactivated.

After you have performed the procedure once, you can mount and activate the swap partition during system intilization by including the necessary commands in the `sysadm.ini` file that you build into the system image.

Preparing and Activating a Swap Partition

1. **Make a directory on the target to use as the mount point for the swap partition** *unless you have already done so***:**

```
$ rsh target arun /bin/ls swap_dir
started aid = 22
/swap_dir: No such file or directory
$ rsh target arun /bin/mkdir swap_dir
```

2. **Make sure the partition you intend to use for swap has been correctly labelled using `disklabel`.**

   The partition you intend to use for swap should be labelled as type `swap`.

3. **Mount the swap directory:**

```
$ rsh target mount -t swap block_special_file swap_dir
```

where *block_special_file* **represents the partition you labelled as type `swap`.**

4. **Activate the swap partition:**

```
$ rsh target swapon swap_dir
```

5.3 Useful File System Utilities

The following list reviews the file system utilities presented so far, and also includes a description of another utility, `df`(1M), useful for checking disk space.

`disklabel` Writes disk geometry and other information to a disk attached to the target system.

`df` Displays information about file system use such as total file system size, available space, inodes used and inodes available.

`flashdefrag` Defragements a Flash memory device attached to the target system.

`format` Formats a Flash memory device attached to the target system.

`fsck` Checks the integrity of a UFS file system and attempts to fix any errors it finds.

`fsck_dos` Checks the integrity of an MS-DOS file system and attempts to fix any errors it finds.

`mknod` Creates a special device driver file corresponding to a device attached to the ChorusOS system.

`mount` Mounts a file system at a specified mount point in the existing file system hierarchy.

`newfs` Writes a UFS file system structure to a labelled disk attached to the target system.

`newfs_dos` Writes an MS-DOS file system structure to a labelled disk attached to the target system.

`swapon` Activates a local swap partition that you have mounted.

`umount` Removes a mounted file system from the file system hierarchy.

CHAPTER **6**

Examples

This chapter offers concrete examples showing how to:

- Use a directory located on the host workstation as a root directory for the ChorusOS target system
- Prepare a RAM disk on the target system and populate it with all necessary files
- Create a file system on a target system local hard disk
- Set up a ChorusOS target system as an NFS server.

CODE EXAMPLE 6–1 Mounting a Root File System over NFS

The first thing to do before mounting the root file system over NFS is build a root file system for your target if you have not done so already.

Change to the directory where you build system images and `make root`:

```
$ cd build_dir
$ make root
```

If the root directory that is built under *build_dir* cannot be exported through NFS, you must move or copy it to a directory you can export:

```
$ cp -R build_dir/root exportable_dir
```

Next, you must export the directory so that it is readable by the ChorusOS target system. On a host workstation running the Solaris operating environment, you can export the directory by adding a line to `/etc/dfs/dfstab`, similar to the following:

```
share -F nfs -o ro -d "target root dir" /export/ChorusOS/root
```

In order for the directory to be shared, you must then restart the NFS server on the host workstation:

```
$ su
Password: root password for host
# /etc/init.d/nfs.server start
```

You can then verify that the directory is indeed available:

```
$ showmount -e host
/export/ChorusOS/root (everyone)
```

After you have verified that the target system root directory has been exported correctly on the host workstation, you may prepare a system image that mounts the directory during system initialization.

Make sure that the `mount` command is built into the `C_INIT` system actor by setting the `ADMIN_MOUNT` feature to `true`:

```
$ configurator -c conf/ChorusOS.xml -set ADMIN_MOUNT=true
```

Make sure that the ChorusOS system can act as an NFS client by setting the `NFS_CLIENT` feature to `true` and the `IOM` buffer tunables to appropriately large values:

```
$ configurator -c conf/ChorusOS.xml -set NFS_CLIENT=true
$ configurator -c conf/ChorusOS.xml -set iom.nbuf=8
```

Add the necessary commands to configure the network interface and mount the directory to the system initialization script, `sysadm.ini`(4CC), located in *build_dir*/`conf/sysadm.ini`:

```
#
# Set umask to 0 during system configuration.
#
umask 0

#
# Prepare Ethernet and loopback interfaces so that you can access the
# host workstation (129.157.197.144) where the root file system is
# located. Target system IP address here is 129.157.197.88.
#
```

```
# Note that you do not have to use Ethernet as your network interface,
# but Ethernet is simple to set up, so we have used it here to avoid
# cluttering the example.
#
mkdev ifeth 0
mkdev lo 0
ifconfig ifeth0 129.157.197.88 netmask 0xffffff00 broadcast 129.157.197.255
ifconfig lo0 127.0.0.1 up

#
# Reset umask to default.
#
umask 22

#
# During system initialization, the target mounts a pseudo / file
# system, but you can mount a new root over the old one. So, mount the
# root file system through NFS.
#
mount 129.157.197.144:/export/target/build/root /

#
# Display mount status to the console.
#
mount
```

Place the new system image where it can be downloaded onto the target and reboot the ChorusOS system. If you are using TFTP to download onto a PC target for example, then you might do so as follows:

```
$ cp chorus.bmon /tftpboot
$ rsh target reboot
```

After the system reboots, the mount information should appear on the console. You could also run one of the actors located under the root file system to verify that everything is functioning correctly:

```
$ rsh target arun /bin/ls /bin
started aid = 22
arp.r           domainname.r    ls.r            pax.r           tclsh.r
chorusNSinet.r  fsck.r          mkdir.r         pppstart.r      touch.r
chorusNSsite.r  fsck_dos.r      mkfifo.r        profctl.r       umount.r
chorusStat.r    ftp.r           mknod.r         profex.r        uname.r
cp.r            hostname.r      mount.r         rdbc.r          ypbind.r
cs.r            ifconfig.r      mv.r            rm.r            ypcat.r
date.r          inetNSdns.r     netstat.r       rmdir.r         ypmatch.r
dd.r            inetNShost.r    newfs.r         route.r         ypwhich.r
df.r            inetNSien116.r  newfs_dos.r     shutdown.r
disklabel.r     inetNSnis.r     nfsstat.r       sysctl.r
```

CODE EXAMPLE 6–2 Making a RAM Disk the Main System Disk

You can use RAM to house you root file system. This example shows you how. It assumes that you have already been able to mount a root directory through NFS. See Code Example 6–1 for details.

Before you get started, you must add RAM disk support to your system image. The following commands add that support, build the system image, copy the image to the download directory and reboot the ChorusOS system running on an x86 target:

```
$ cd build_dir
$ configurator -c conf/ChorusOS.xml -set RAM_DISK=true
$ configurator -c conf/ChorusOS.xml -set iom.ramdisk0.size=0x1600000
$ configurator -c conf/ChorusOS.xml -set iom.ramdisk.sizeMax=0x1600000
$ make chorus
$ cp chorus.bmon /tftpboot
$ rsh target reboot
```

In order to use RAM to house you root file system, you must first label part of the available space using `disklabel`(1M). `disklabel` uses entries in `disktab`(4CC) to label the disk. The following `disktab` entry serves to label a 16 megabyte area of RAM as a disk:

```
# This label can be used for a 16 megabyte RAM disk. The corresponding
# newfs command to be used with this RAM disk is:
# rsh target arun /bin/newfs -o space -c 2048 -m 0 /dev/rrd0a

rd16Meg:\
 :ns#4:nt#4:nc#32768 \
 :pa#32768:oa#0:ta=4.2BSD: \
 :pc#32768:oc#0:tc=unused:
```

The default system initialization file, `sysadm.ini`, creates special files to access the RAM, so you probably do not need to create them yourself. The entries to create special files may appear as follows in `sysadm.ini`:

```
# Create special files for RAM disk
mknod /dev/rrd0a c 13 0
mknod /dev/rrd0b c 13 1
mknod /dev/rrd0c c 13 2
mknod /dev/rd0a  b 13 0
mknod /dev/rd0b  b 13 1
mknod /dev/rd0c  b 13 2
```

The above example creates both raw and block mode special files for two user-accessible partitions, `a` and `b`, and one partition, `c`, for use by the system only.

Assuming the special files exist, you can proceed to label the disk and so forth. The following commands label the disk, read the label, create a file system on the disk and check the file system:

```
$ rsh target arun /bin/disklabel -w -r rd0 rd16Meg
started aid = 22
$ rsh target arun /bin/disklabel -r rd0
started aid = 22
# /dev/rrd0c:
type: unknown
disk: rd16Meg
label:
flags:
bytes/sector: 512
sectors/track: 4
tracks/cylinder: 4
sectors/cylinder: 16
cylinders: 32768
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0  # milliseconds
track-to-track seek: 0 # milliseconds
drivedata: 0

3 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:    32768        0    4.2BSD        0     0     0  # (Cyl.    0 - 2047)
  c:    32768        0    unused        0     0        # (Cyl.    0 - 2047)
$ rsh target arun /bin/newfs /dev/rrd0a
started aid = 22
/dev/rrd0a: 32768 sectors in 8 cylinders of 1 tracks, 4096 sectors
 16MB in 1 cyl groups (16 c/g, 32MB/g, 7680 i/g)
super-block backups (for fsck -b #) at:
 32,
$ rsh target arun /bin/fsck -y /dev/rrd0a
started aid = 22
** /dev/rrd0a
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
1 files, 1 used, 15390 free (14 frags, 1922 blocks, 0.1% fragmentation)
```

Once you have checked the new file system and it is ready to use, you need a place to mount it in the current file system hierarchy. As the current root file system is located on the host workstation, you create the mount point on the host and then mount the RAM file system there:

```
$ cd target_root_dir
$ mkdir ram
$ rsh target mount -t ufs /dev/rd0a /ram
/dev/rd0a on /ram
```

Next, we are going to populate the RAM file system with the contents of everything except the /ram directory:

```
$ rsh target arun /bin/cp -R bin dev etc image lib ram
```

The RAM file system, mounted under the /ram directory, now contains everything needed to serve as the main system disk. We are now going to take advantage of the fact that the mount command is built into the C_INIT system actor.

We are going to unmount the root file system.

The following commands unmount the root file system on the target and remount the RAM file system in its place:

```
$ rsh target umount /ram
$ rsh target mount /
$ rsh target mount -t ufs /dev/rd0a /
```

As you can see, the target is now independent of the host, using the RAM disk as the main disk:

```
$ rsh target arun /bin/ls
started aid = 22
bin             dev             etc             image           lib
```

Note that the ls actor used above is located in the RAM disk file system.

The following script is meant to run on the host workstation (or other system) and summarizes the above scenario:

```
#!/bin/sh

#
# Reboot the target and give it some time to come up.
#
rsh target reboot
sleep 60

#
# Label the RAM disk, then create the file system and check it.
#
rsh target arun /bin/disklabel -w -r rd0 rd16Meg MyRamDisk
rsh target arun /bin/newfs /dev/rrd0a
rsh target arun /bin/fsck -y /dev/rrd0a

#
# Mount the RAM file system and populate it with the rest of the root
# directory.
```

```
#
rsh target mount -t ufs /dev/rd0a /ram
rsh target arun /bin/cp -R bin dev etc image lib ram

#
# Unmount everything, then mount the RAM file system at the root.
#
rsh target umount /ram
rsh target umount /
rsh target mount -t ufs /dev/rd0a /
```

After adapting the above script for your environment, run it to make your ChorusOS target self-sufficient.

CODE EXAMPLE 6–3 Creating a File System on a Target System Hard Disk

If your ChorusOS system has a hard disk drive that you want to use to support a local file system, you can use the ChorusOS system utilities to prepare the disk, and create and populate a local file system. This example assumes that you have already been able to mount a root directory through NFS. See Code Example 6–1 for details.

Before you can prepare the disk, you must be able to access it. Set the appropriate feature or features, depending on whether the hard disk is an IDE or SCSI disk and depending on what file system type you plan to use (see Step 2 on page 16 for details), build the new system image that supports a hard disk and reboot the target system with the new system image. For example, if you are using TFTP to download onto a PC target with an IDE disk, then you might do so as follows:

```
$ cd build_dir
$ configurator -c conf/ChorusOS.xml -set IDE_DISK=true
$ configurator -c conf/ChorusOS.xml -set UFS=true
$ make chorus
$ cp chorus.bmon /tftpboot
$ rsh target reboot
```

The default `sysadm.ini` system initilization file contains commands that create special files for both IDE and SCSI disks, so you probably do not need to create any special files at this point.

If the target system disk already contains information, it is possible that you may be able to use the data on an existing partition. For the purposes of this example, we assume that the hard disk is bare and that you need to start by labelling the disk. In order to label the disk, you first need an appropriate entry in `disktab`(4CC). The following example `disktab` fragment could be used with a `ST34311A` IDE disk to create two partitions, one small and one large:

```
ST34311A:\
        :dt=ST506:ty=fixed:se#512:nt#15:ns#63:nc#8944:sf \
        :pa#60480:oa#0:ta=4.4LFS: \
        :pb#8391600:ob#60480:tb=4.4LFS: \
        :pc#8452080:oc#0:tc=unused:
```

Once the `disktab` file contains the entry describing the disk geometry, you can proceed to label the disk. For example:

```
$ rsh target arun /bin/disklabel -w -r hd0 ST34311A
```

After the disk is labelled, you can create file systems on the disk partitions:

```
$ rsh target arun /bin/newfs /dev/rhd0a
$ rsh target arun /bin/newfs /dev/rhd0b
```

Finally, before using the file systems, you should check them:

```
$ rsh target arun /bin/fsck -y /dev/rhd0a
$ rsh target arun /bin/fsck -y /dev/rhd0b
```

After you have checked the file systems you created, you can mount them into your existing file system hierarchy and use them.

CODE EXAMPLE 6–4 Setting Up an NFS Server on the Target System

If your ChorusOS system has a local file system, you may want to export part of it through NFS to share files with other systems on the network. The following example assumes that you want to do that and that you have already been able to create a local file system in RAM. See Code Example 6–2 for details.

Before you can use the ChorusOS system as an NFS server, you must set the appropriate features and tunables. For example, if you are using TFTP to download onto an x86 target, then you might do so as follows:

```
$ cd build_dir
$ configurator -c conf/ChorusOS.xml -set NFS_CLIENT=true
$ configurator -c conf/ChorusOS.xml -set UFS=true
$ configurator -c conf/ChorusOS.xml -set NFS_SERVER=true
$ configurator -c conf/ChorusOS.xml -set iom.kmemsize=0x80000
$ configurator -c conf/ChorusOS.xml -set iom.nbuf=8
$ make chorus
$ cp chorus.bmon /tftpboot
$ rsh target reboot
```

Note that you can export only UFS file systems through NFS.

Once the ChorusOS system has support for NFS, you must configure at least `/etc/exports` for the target system:

```
#
```

```
# Export everything to everyone.
#
/  -alldirs -maproot=0:1
```

The directory you export must be local to the ChorusOS system. For this example, we build on a previous example, Code Example 6–2, that leaves the target system with a full file system on a local RAM disk. From this point on, we assume you have gotten that example to work in your environment. Once you have gotten it to work, you can reboot the target, prepare the file system in RAM and run the daemons required to start NFS services. The following commands enable name services, portmapping, the `mountd` daemon and three NFS servers for UDP and TCP clients:

```
$ rsh target arun /bin/inetNShost&
$ rsh target arun /etc/portmap&
$ rsh target arun /etc/mountd&
$ rsh target arun /etc/nfsd -ut -n 3&
```

At this point, you can mount the target system root directory on another system, such as the host workstation or another target. The following commands mount the file system on the host workstation:

```
$ su
Password: root_password
# mount target:/ /mnt
```

You could also choose to mount the exported ChorusOS system directories from other target systems, for example.

CODE EXAMPLE 6–5 Setting up Swap

The following example sets up a swap partition on a local IDE disk. This example assumes you have already successfully completed Code Example 6–3 and that your system is properly configured to support UFS file systems on the hard disk. It also assumes that *no valuable data has been written to the hard disk*. If you have important data on the hard disk, you **must** back it up before trying this example.

Before you can activate swap on the disk, you must configure the system image to support the virtual memory. Following Code Example 6–3, you might do so as follows:

```
$ cd build_dir
$ configurator -c conf/ChorusOS.xml -set VIRTUAL_ADDRESS_SPACE=true
$ configurator -c conf/ChorusOS.xml -set FS_MAPPER=true
$ make chorus
$ cp chorus.bmon /tftpboot
$ rsh target reboot
```

For the purposes of this example, we assume that the hard disk is bare and that you need to start by labelling the disk. In order to label the disk, you first need an appropriate entry in `disktab`(4CC). The following example `disktab` fragment could be used with a `ST34311A_S` IDE disk to create two partitions, one small swap partition and one large partition to house a UFS file system:

```
ST34311A_S:\
        :dt=ST506:ty=fixed:se#512:nt#15:ns#63:nc#8944:sf \
        :pa#60480:oa#0:ta=swap: \
        :pb#8391600:ob#60480:tb=4.4LFS: \
        :pc#8452080:oc#0:tc=unused:
```

Once the `disktab` file contains the entry describing the disk geometry, you can proceed to label the disk. For example:

```
$ rsh target arun /bin/disklabel -w -r hd0 ST34311A_S
```

After the disk is labelled, you can create a mount point for the swap partition, mount the partition as type `swap` and activate it:

```
$ cd target_root_dir
$ mkdir swap
$ rsh target mount -t swap /dev/rhd0a /swap
$ rsh target swapon /swap
```

The swap partition is now active.

Index