LynxOS User's Guide

LynxOS Version 2.5

DOC-0165

Printed in the United States of America.

Contents

Preface

The *LynxOS User's Guide* contains information about basic system administration and kernel-level specifics of LynxOS. It in covers a range of topics, including tuning system performance, booting LynxOS over a network, and creating kernel images for embedded applications.

For More Information

For more information on the features of LynxOS, refer to the following printed and online documentation.

- *Release Notes*

  This printed document contains late-breaking information about the release.
- *LynxOS Installation Guide*

  This manual supports the initial installation and configuration of LynxOS and the X Windows System.
- *Quick Start Guide*

  This guide provides an overview of basic LynxOS concepts.

- *Writing Device Drivers for LynxOS*

  This printed document provides a description of LynxOS device drivers and gives numerous examples to help you write custom device drivers for many different situations.
- *GNU C/C++ Tools Set*

  This set of five volumes describes the GNU C/C++ compilers and tools.
- *LynxOS Reference Manuals Set*

  These manuals are organized into UNIX-like functional sections. The sections are:
  - Section 1, Utility Programs (three volumes)
  - Section 2, System Calls (one volume)
  - Section 3, Library Functions (two volumes)
  - Section 4, Device Facilities; section 5, File Formats; and section 9, Driver Service Calls (combined into one volume)
- *Online information*

  The commands and utilities described in the *LynxOS Reference Manuals* are also provided online in text format. You can access this information with the man command. For example:

  ```
  man gcc
  ```

Conventions Used in This Guide

This Preface shows the typographical conventions used in this guide. All references to file names and commands are case sensitive and should be typed exactly as shown.

Fonts

The typefaces in this guide are used to emphasize important concepts. These are described below.

Description	Example
`Courier` font denotes commands, programs, file names, and words that would appear on your computer screen.	`myprog.c` Enter `ls` on the command line
`Bold Courier` font indicates words that you have typed on your computer.	`login:` **`myname`** **`cd /usr/home`**
Words that you must replace with a real name or value appear in *italics*.	`cat` *filename* `mv` *file1 file2*
New terms, book titles, and emphasized words appear in *italics*.	Refer to the *LynxOS Reference Manuals*.

Special Notes

The following notations highlight key points and cautionary notes in this guide.

NOTE: *These callouts note important points in the text.*

CAUTION! *This symbol stresses information that is critical to the integrity or continuity of an operation.*

Technical Support

Lynx Technical Support is available to answer your questions about LynxOS. Please feel free to contact us Monday–Friday (holidays excluded) between 8:00 AM and 5:00 PM Pacific Time (Lynx U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Lynx Europe).

In addition, browse our World Wide Web home page for additional information about Lynx products, Frequently Asked Questions (FAQs), and Lynx newsgroups.

World Wide Web

http://www.lynx.com

Lynx U.S. Headquarters

Internet: support@lynx.com
Phone: (408) 879-3940
Fax: (408) 879-3945

Lynx Europe

Internet: tech_europe@lynx.com
Phone: (+33) 1 30 85 06 00
Fax: (+33) 1 30 85 06 06

CHAPTER 1

Administration Basics

When LynxOS is used heavily by several users developing software or operating some multi-user application system, various maintenance tasks are necessary. These tasks include disk space maintenance and backup, user support, spooling system configuration, adding and removing system devices and terminals, and system shutdown and reboot.

User Privilege

Many of the programs described in this chapter are only available to the root user. The root user is that user whose user ID number is `0`. The root users are often called the “super user” because they have more capabilities than other users.

Once initial system setup is complete, the password should be set. Securing the root account is important to protect the system from unwanted intrusion over dial-up lines and from uninformed or irresponsible use by ordinarily non-privileged users.

See “Understanding Security Issues” on page 21 for details on passwords and security issues.

Files in /etc

As a system administrator, you will need to understand the set of files in the `/etc` directory. They are listed below with a brief explanation.

`devices` — Contains a list of devices which are to be installed by the `devinstall` command with the `-a` option. Each line of the file describes a single device via four fields, separated by colons:

```
info:driver:type:extra
```

See the `devices(5)` man page in the *LynxOS Reference Manuals* for more information.

The `devices` file need exist only if the automatic installation features of the `drinstall` and `devinstall` programs are to be used.

`drivers` — Contains a list of device drivers which are to be installed by the `drinstall` command with the `-a` option. Each line of the file describes a single driver via two fields, separated by a colon:

```
driverfile:type
```

See the `drivers(5)` man page in the *LynxOS Reference Manuals* for more information.

The `drivers` file need exist only if the automatic installation features of the `drinstall` and `devinstall` programs are to be used.

`fstab` — Contains a list of devices with file systems which are normally mounted while the system is running. Each line gives a single device and the point at which it is mounted via five fields as follows:

`device:mountdir:type:freq:passno`

Further information can be found in the `fstab` file format documentation and the `mount` and `umount` utilities documentation.

The `fstab` file must exist for the benefit of the `rc` system start-up command file and for the `df` command.

`mtab`	Contains information about currently mounted file systems. The format of `mtab` is not human readable; the file is entirely maintained by the `mount` and `umount` programs. See the `fstab(5)` man page in the *LynxOS Reference Manuals* for more information. The `mtab` file is created by `mount` if necessary. The file contains the only record of mounted devices; however, its existence is not critical for proper system operation.
`nodetab`	Contains a list of standard nodes to be created by `mknod` with the `-a` option. Each line describes a single node via four fields separated by colons: `name:type:major:minor` This file will probably be used only once per file system. It can be created or modified as necessary, or even deleted. For more information, see the `nodetab(5)` and the `mknod(1)` man pages in the *LynxOS Reference Manuals*.

`passwd`	Contains entries describing each user known to the system. A “user” is an entity which can own files or possess an active process. Users are identified internally be numbers, which through the passwd file are mapped to user names. LynxOS internally allows any user numbers, but the normal set of users on a given system is in this file. Each line describes a user as follows: `name:passwd:uid:gid:info:dir:shll` For more information, see the `passwd(5)` man page in the *LynxOS Reference Manuals*. The `passwd` file must exist. It is used by several system programs to relate one item of information about a user to another. It is also consulted by the login program to verify a user name.
`printcap`	Contains information about the various printers available on the system. The printers can be either local or remote. The file is used by `lpr` and `lpd` whenever a print job is queued. For more information, see the `printcap(5)` man page in the *LynxOS Reference Manuals*.
`starttab`	Contains information read by `init` at system boot time. The `starttab` file gives the default `umask` value, the program to use for the single user shell at boot time, the name of the system initialization file (usually `/bin/rc`), and the default data, stack, and core limits for all tasks. See the `starttab(5)` man page in the *LynxOS Reference Manuals*.

	The `starttab` file is only read by `init`. If it does not exist, default values are used.
`tconfig`	Contains descriptions of serial port configurations. It serves the same purpose as `/etc/gettytab` on BSD-based systems. The `tconfig` file is referenced by `stty`, `tset`, and other programs to prepare a terminal for some special use. Each entry consists of one or more lines containing assignments of values to various parameters. For more information, see the `tconfig(5)` man page in the *LynxOS Reference Manuals*. The `tconfig` file is very important. Various utility programs (`tset` in particular) will refuse to operate if the file cannot be found.
`termcap`	Contains descriptions of software features for various specific terminal devices. Each entry consists of one or more lines containing assignments of values to various parameters. The descriptions are read by the text editor `vi`, the shell, and other programs that must manipulate the terminal using software command sequences. The `termcap` file should exist, and it should contain descriptions of all terminals likely to be used with the system.
`ttys`	Contains a list of serial ports to be recognized by `init` as valid for login. Each line of the file describes one

serial port via five fields separated by colons:

`device:flag:config:terminal:login`

For more information, see the `ttys(5)` man page in the *LynxOS Reference Manuals*. The `ttys` file must exist for `init` to control multi-user operations.

`utmp` Contains entries describing active login sessions. The file consists of records (not human readable) containing the user name, host of origin, and login time for each active session. Each record corresponds to an entry in `ttys`.

The `utmp` file must exist for `who` to function. It will be created if possible by `login` if for some reason it has been removed.

Device Names under LynxOS

Like UNIX, LynxOS uses “special files” for device I/O. These files normally reside in the directory `/dev`. All peripherals, such as disks, tapes, and CD-ROM devices need to have a corresponding special file in `/dev`. These files follow a naming convention that is described in more detail below.

Floppy Disks

Floppy device drives are supported on LynxOS for x86, SPARC, and PowerPC. On the SPARC, the device name for a floppy drive is simply `fd0`. On the x86 and PowerPC, however, floppy device names are derived as shown in Figure 1-1.

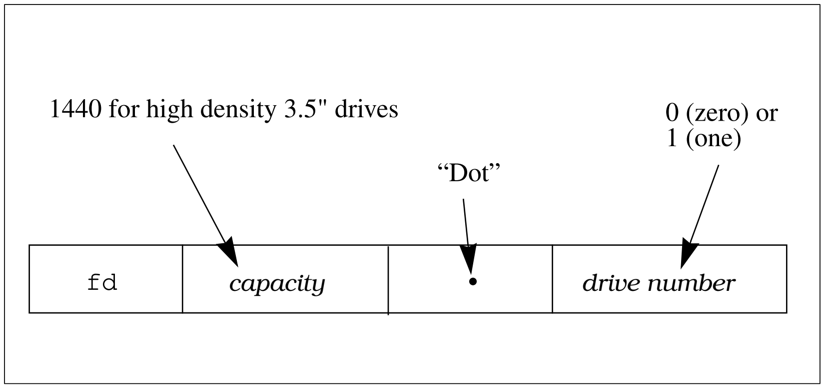

Figure 1-1: Deriving LynxOS Floppy Drive Names on x86 and PowerPC

The *capacity* field is the disk capacity and the *drive number* refers to the physical connection of the drive to the system. If you're familiar with MS-DOS, Drive 0 corresponds to drive letter `A:` and drive number 1 corresponds to drive letter `B:`. If you have only one floppy drive, it will always be Drive 0.

As an example, the device name for a 3.5 inch high density (1.44 MB) floppy drive on Drive 0 (the boot drive) is `fd1440.0`.

On PowerPC systems only, LynxOS provides support for the TEAC SCSI floppy disk drive. The syntax for the device name of a SCSI floppy is fdscsi<*capacity*>.0 For example, a 1.44 MB SCSI floppy would be `fdscsi1440.0`.

Disk Drive(s)

LynxOS supports systems that have one or more disk drives. CD-ROM drives are treated as read-only disk drives. Disk drive names contain important information about the device. In general, a prefix identifies the disk type and controller combination. This is followed by the disk ID number and is completed with the partition number. Refer to Figure 1-2.

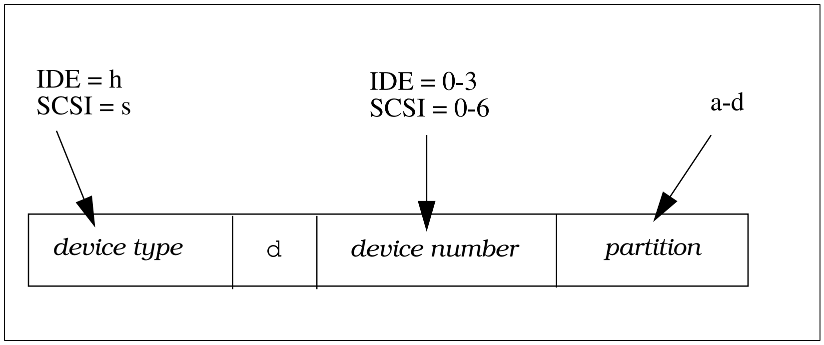

Figure 1-2: Deriving LynxOS Disk Names

However, there are some platform-specific issues associated with device names. These issues are discussed in the following sections.

Device Types By Platform

For x86

On the x86, two disk controller types are currently supported by LynxOS— the standard AT-controller for IDE drives or the SCSI controller for SCSI drives. If a SCSI drive is the only drive being used, or if you want to boot off the SCSI drive, the system BIOS configuration must have “Hard Disk Type” set to “Not Installed”. Refer to your system documentation to determine the type and capacity of your disk.

On the x86, “h” denotes an IDE device. LynxOS supports up to two IDE controllers on a single x86 system. There are four possible devices that been be accessed.

`hd0`	primary master
`hd1`	primary slave
`hd2`	secondary master

`hd3`	secondary slave

NOTE: *IDE CD-ROM drives must be either* `hd0` *or* `hd2`.

For example, the device name for partition A on the primary master IDE disk is: `hd0a`.

However, SCSI device names for the x86 differ from the above figure. SCSI devices follow this convention:

`sd` + *controller name* + `.` + *device number* + *partition*

For example, the name for an Adaptec 1542 SCSI device is: `sd1542.0a`

For 68k, SPARC, and PowerPC

On the 68k, SPARC, and PowerPC, the device type is “s” for a SCSI device. For example: `sd0a`

Device number

Most systems only have one hard disk and are set up as device number 0 (i.e. the first device of this type). Device number 1 would refer to the second device of this type. The drive number is often set by jumpers on the hard disk. Refer to the manufacturer’s documentation for the hard disk to determine or change the drive number.

IDE devices (x86 and PowerPC) are limited to two device numbers per controller. For this type of disk, the device number can be either 0 or 1. Some EIDE controllers allow support of up to four disks; however, LynxOS only supports two disks per EIDE controller.

SCSI devices, in contrast, can have up to seven device numbers per controller. Valid device numbers for a SCSI device are 0-6. This number maps to the SCSI ID number for that device.

Partitions

Each hard disk on your system can have up to four partitions. The partition that contains the operating system that boots the machine is known as the *root partition*.

Partitions are indicated by the letters “a”-”d”. The letter “a” refers to the first partition and “d” to the last. The partition value may be left blank if you have an unpartitioned disk.

Examples

Table 1-1 gives examples of device names for some disk/controller combinations.

Table 1-1: Examples of Disk Drive Device Names

System	Device Type	Controller	Drive Number	Partition Number	Device Name
x86	IDE		0	a	`hd0a`
x86	SCSI	Adaptec 1542	0		`sd1542.0`
x86	SCSI	Adaptec 2940	1	c	`sd2940.1c`
68k	SCSI		0	b	`sd0b`
SPARC	SCSI		3		`sd3`
PowerPC	SCSI		6	a	`sd6a`

Major and Minor Numbers

Although users and applications use device names to refer to a particular device (as discussed in section on page 6), the operating system (LynxOS) uses a unique numeric identification. This identification consists of a major and minor number. The major and minor numbers of the devices on your system depend on your implementation of LynxOS. A long listing of the `/dev` directory shows the major and minor numbers of each device special file. These are located in column four as two digits separated by a comma (*major, minor*).

```
$ ls -l /dev
brw-rw-rw-   1 root       2,13   Apr  7 17:15 fd1440.1
brw-------   1 root       1,0    Apr  7 17:15 hd0
brw-------   1 root       0,32   Apr  7 17:15 sd0b
brw-------   1 root       3,0    Apr  7 17:15 sd1542.0
brw-------   1 root       3,16   Apr  7 17:15 sd1542.0a
```

Figure 1-3: Example Listing of /dev

In the above example, `/dev/fd1440.1` has a major number of 2 and a minor number of 13. To view the major and minor numbers of the devices on your system, use the command `ls -l /dev`.

Creating Device Nodes

Under LynxOS, all system resources are accessed through the directory structure of the file system. Each device must therefore be named in a directory. Such directory entries (which do not actually describe data files but rather are interfaces to a device driver) are called nodes.

All devices included in a standard system are installed into the file system and are ready to use initially. However, if new devices are added to a system, or if a new file system is created on another disk or diskette, new nodes will need to be created. The `mknod` program creates nodes for any character device, block device, or named pipe (FIFO). The syntax for `mknod` is:

`mknod` *name type major minor*

The *name* is a LynxOS pathname for the new node. Nodes can be made in any directory, but are usually placed in `/dev`. *type* gives the device type and is either `b` for a block device, `c` for a character device, or `f` for a named pipe. The *major* and *minor* arguments give the major and minor numbers for the device (see “Major and Minor Numbers” on page 10 for more information).

These numbers are dependent upon the configuration of the operating system and any devices and drivers which have been dynamically installed. For example, to create a node for the second floppy disk that has a major number of `0`:

`mknod /dev/fd1 b 0 16`

It is often necessary to create new file systems on floppy disks, for backup or testing. Because device nodes must exist for many utility and system programs to work, they must be installed in a `/dev` directory on each new file system which might be used as the root file system. To simplify the process of creating nodes on new file systems, a list of important devices for a particular system should be kept in the file `/etc/nodetab`. The nodes are listed one per line, much as when described as arguments directly to `mknod`:

```
name:type:major:minor
```

The `mknod` program will read this file and create all the nodes described if invoked as shown below:

`mknod -a [`*filename***`]`**

If *filename* follows the **`-a`** switch, it will be read instead of `/etc/nodetab`.

Formatting Media

New disks must be formatted before they are used. The process of formatting varies from device to device, but in general it involves writing any necessary control information on the media such as sector marks, initial data, and CRC information.

For floppy disks in particular, formatting overwrites all old data. Two utilities are supplied with LynxOS for formatting media, `fmtflop` for floppy disks and `fmtscsi` for SCSI disks. These programs issue `ioctl` calls to the appropriate drivers requesting a format operation. For floppy disks, the interleave factor and an initial filler value can be specified.

`fmtflop [-v][-f` *filler*`][-i` *interleave*`]` *device*

The *interleave* factor dictates the physical spacing of logically sequential blocks. Optimally, logically sequential blocks are also physically sequential. The *filler* value, any non-zero value, is written into each block. Default values for *interleave* and *filler* are 1 and 229. `fmtflop` normally operates silently. The **-v** (verbose) option instructs the program to print each track number as it is formatted.

Many SCSI disks do not do anything when requested to format. The `fmtscsi` command is provided nevertheless and is invoked as follows:

`fmtscsi -F [-i` *interleave*`]` *device*

As with floppy disks, *interleave* should almost always be `1` (the default). `fmtscsi` requires two confirmations before it will actually issue the format command to the driver.

Making File Systems

The `mkfs` command creates a new file system on a disk or diskette by making a new super block, an inode table, a free block list, and a root directory.

CAUTION! *The command* `mkfs` *is destructive. All existing data on a disk or partition will be lost after executing* `mkfs`*.*

The syntax for `mkfs` is:

mkfs -b *value* *device* [[/]inodes]

The **-b** option is the block size of the new file system (default 512). Accepted values are 512, 1024, 2048, 4096 , 8192, 16384, and 32768; these values describe how many bytes comprise a disk block. The installation utility, `installit`, uses **-b 2048** when making the file system. You should choose a large number of bytes per block when the majority of your files are large. Chose 512 or 1024 when if your disk contains mostly small files. This will optimize file storage on the disk. The values 2048 and 4096 are good general use values.

For example, the following command will make a 2048 byte block size file system on partition c of SCSI device 0:

```
mkfs -b 2048 /dev/sd0c
```

The option **inodes** allows you to modify total number of inodes. If you expect that your system will have many small files, you may want to specify a larger value for **inodes**. For example, to create a file system with 8000 inodes on partition b of SCSI device 2, type the following command:

```
mkfs /dev/sd2b 8000
```

For floppy diskettes, however, the space used for unnecessary inodes can be important, so a larger ratio may be chosen. For example:

```
mkfs /dev/fd1440.0 4
```

After it has been created, you can examine the file system parameters with the command `df`. For example:

df -i *device*

`df` will show the number of free blocks available for data, after taking into account the blocks used for the inode table and the super block. Once a file system has been created, the device containing it can be mounted with the `mount` command.

Making Directories

The primary means of organizing files in the LynxOS file structure is the hierarchical directory system. A directory is a file which records pairs of file names and inode numbers. Among the files recorded in a directory may be other directories.

When a file system is created, only a single directory, the root directory, is created. All others, including those necessary for a root file system, must be created. (It is worth noting at this point that the term root is somewhat over-exercised. It denotes the privileged user, the main file system, and the top of the directory tree—three unrelated entities. The terminology has been inherited from UNIX.)

The `mkdir` command creates directories. It must be used when creating a new root file system to make `/bin`, `/cc`, `/dev`, `/etc`, and `/lib`. The syntax is:

`mkdir` *name* [*name ...*]

Each directory named is created with the two special entries "`.`" and "`..`", which name the inode for the directory itself and the inode of its parent directory. For example, to create the important directories listed above, assuming a new file system has been mounted at `/mnt`, type the following command:

```
mkdir /mnt/bin /mnt/cc /mnt/dev /mnt/etc
/mnt/lib
```

In addition to the directories needed for system operation, directories for software development or applications must also be created. Generally, each individual who is to have an "account" on the system will be given a directory, either directly beneath the root or in some subdirectory (`/usr` is commonly created for this purpose).

Such additional directories can be freely created by the super user or by any other users as the system is used. If a directory is no longer needed, it can be removed with the `rmdir` command:

`rmdir` *directory*

The directory must be empty, except for "dot" (`.`) and "dot dot" (`..`). If a non-empty directory and its contents are to be removed, the `rm` command can be used:

`rm -r` *directory*

The **`-r`** option instructs `rm` to remove the entire sub-tree beneath the directory. Once again, these commands are available to all users, not just the super user.

Establishing User Accounts

Each user on a LynxOS system should be given a distinct, separate account. An account consists simply of an entry in the `/etc/passwd` file and a directory sub-tree controlled by the user. The root of the user's sub-tree is called the home directory of that user. Normally, file protections (see "File Protection" on page 21) are established and maintained such that only the super user can perform the work of creating accounts.

There are two unique attributes of a user: the user name (an arbitrary character string) and the user ID number (an integer between 0 and 65534). The user name is customarily some adaptation of a person's name. The user ID can be chosen arbitrarily; it is rarely manipulated or even seen. However, since file ownership is recorded in terms of user ID number, not user name, different users *must* have distinct user ID numbers.

Note that user ID `0` is the super user. An entry in the password file `/etc/passwd` associates the chosen user name with the user ID number. The entry also contains space for an encrypted password, a group ID number, a brief description of the user (usually the user's real-world name), the name of the user's home directory, and the name of the user's shell. As an example, consider this entry for a user named "bob":

```
bob::17:3:Bob Jones:/usr/bob:/bin/bash
```

The user has user ID `17`, group ID `3`. His home directory is `/usr/bob`. No password has been set for this account; if one had, the second field would contain a printable, but unintelligible, string.

LynxOS Administration Utilities

setup

The **`setup`** utility guides you through the initial configuration and later reconfiguration of the system. To run the **`setup`** utility, log into the LynxOS system as the user "setup"; this automatically starts the **`setup`** utility from which the following procedures are performed:

1. Add a password for the `setup` and `root` user.
2. Set up symbolic links for the floppy drives (x86 only).
3. Enable multiple consoles (x86 only).

4. Enable `ttys` for login.
5. Add/Delete users to/from the system.
6. Enable Virtual Memory.
7. Set Time, Time Zone, and Daylight Savings support.
8. Enable Write-Through Cache.

The **`setup`** utility is a self-documented, menu-driven program. Read all instructions displayed by **`setup`** before making selections.

The first question the program will ask is if you want to add a password to the `setup` and `root` accounts.

```
Do you want to give the setup account a password?
< yes [no] >
```

Both the `setup` and the `root` accounts are typically reserved for special usage and have “super-user” power; i.e., the ability to manipulate all other users’ accounts and data.

It is recommended that these accounts have passwords.

When prompted, type in the password. The characters will not echo on the screen. This is to ensure that passwords remain secret. The system will ask you to repeat the password to ensure you did not miss-type it.

If the password you enter is too short, the system will prompt you for a longer, less obvious password. The password-length safety mechanism can be overridden, allowing entry of a shorter password, by repeatedly entering the desired, shorter password; after two such efforts, the system will accept the shorter password.

adduser

User accounts can be added to the system with the **adduser** utility, which is automatically invoked by the **setup** utility. **adduser** will create an entry in the `/etc/passwd` file and a home directory for each user.

To add each user to the system, enter "yes" to this prompt:

```
Do you want to add a user to the system?
< yes [no] >
```

For each user added, the following information must be supplied:

1. User Name
2. User ID Number

   The default is the first available number.
3. User's Group ID Number

   The default group ID is 2 (the staff group ID). A new group can be created by modifying the `/etc/group` file.
4. Comment Field

   A string of any length can be entered in this filed. Typically, the full name of the user is entered into the comment field.
5. User's Home Directory

   This will be the user's initial working directory when logging into the system; it is owned by the user.
6. User's Login Shell

   The login shell is the interface between the user and the LynxOS kernel. Neither login nor shell configuration files are copied to the user directory.

The system administrator will need to copy those files.

There are five available shells:

`/bin/csh`	A symbolic link to `/bin/tcsh`
`/bin/tcsh`	Enhanced Berkeley C shell
`/bin/sh`	A symbolic link to `/bin/bash`
`/bin/bash`	Free Software Foundation Korn-compatible shell
`/bin/dlsh`	Lynx Real-Time Systems' proprietary shell

The user's password can also be assigned at this time.

deluser

User accounts can be deleted from the system with the **`deluser`** utility. **`deluser`** is automatically invoked by the **`setup`** utility. It deletes the specified user's entry from the `/etc/passwd` file and removes the user's home directory and subdirectories, if specified.

The first time **`setup`** runs, the following prompt does not appear. On subsequent invocations, to remove a user, enter `yes` when prompted:

```
Do you want to delete a user from the system?
< yes [no] >
```

If you remove the user's home directory, make sure all files that may be needed in the future are backed up.

Understanding Security Issues

In order to protect data from accidental removal, and to protect applications from inadvertent or malicious tampering, LynxOS provides a means to protect files and execute programs. The protection scheme revolves around user and group ID numbers, which are associated with every file and (in the case of the user ID) with every process. In general, the owner of a file or process is able to manipulate it freely.

File Protection

Each file is associated with a set of access control flags which define how the file can be used by its owner, members of the owning group, and all others. For each category, the flags, called the *mode* of the file, may permit reading, writing, or (if the file is an executable program) execution.

Note that by this method even the owner can be prevented from accessing the file in some ways; however, the owner can freely change the mode of the file to suit his needs. The mode of a file is stored in its inode as an integer. The lower nine bits of the integer specify access privileges for the owner, the owning group, and others as follows:

8	7	6	5	4	3	2	1	0
r	w	x	r	w	x	r	w	x
owner			**group**			**other**		

If a bit is enabled, then the corresponding access permission (**`r`** for read, **`w`** for write, or **`x`** for execute) is granted to one of the classes of user. When a file is created, the effective user ID number of the creating process is assigned to the file. The group ID number of the file is inherited from the directory in which it is created.

The mode of the new file is derived by combining the mode requested by the process when it called open with the current `umask` of the process. The `umask` is a number which, when expressed in binary, tells which access bits of a requested mode are to be disabled.

For example, if a process attempts to create a file with mode 0666 (expressed in octal; `110110110` in binary) and the `umask` is currently `0022` (also octal), then the file will be given mode `0644`:

requested:	1	1	0	1	1	0	1	1	0
umask:	0	0	0	0	1	0	0	1	0
result:	1	1	0	1	0	0	1	0	0

The mode of a file can be changed with the `chmod` command. In its simplest form, `chmod` accepts an octal representation of the mode and applies it to a list of files:

chmod *mode file file ...*

The `chmod(1)` man page contains a description of more facilities and options.

Changing Effective User ID

Sometimes, applications must access files which would normally be protected from access by a user ID under which the application is running. For example, a program that maintains a simple database might allow all users to add records to the database, even though the data files are protected against access with ordinary utility programs.

In addition to the basic protection code bits, the mode of an executable file contains flags which can be used set the effective user and group ID numbers upon execution. With this facility a program can be configured so that while it is running the effective user or group ID, which are used to determine access privileges, is changed to that of the owner of any protected data.

The `set-user-ID` and `set-group-ID` flags are referred to by constants defined in the include file `<stat.h>`, `S_ISUID` and `S_ISGID`. These bits can be set or cleared with the `chmod` command:

`chmod u+s` *file*
`chmod g+s` *file*

Note that programs which are owned by user ID `0` that have the `set-user-ID` flag set assume most super-user privileges while running. Such programs must be carefully written to prevent unwanted side effects or security violations.

Process Protection

Processes, like files, are also protected against interference from processes running under different user ID numbers. Only the owner of a process and the super user can send signals or set the priority of that process.

When a process starts, it is assigned the user ID of the parent process. When a program is loaded, the user ID of the process will change if the `S_ISUID` bit of the program is set. Although a process is protected from direct interference from other processes, it is possible nevertheless for a non-privileged process to set its priority high enough to monopolize the processor.

In the context of a real-time application, this feature is expected of a real-time operating system. However, in a development environment, one user could monopolize system resources. The `setpriority` system call allows the root user to set the maximum priority of all non-root users. Thus, non-root users cannot set their priority above that value.

Managing Terminals

When a LynxOS computer is used for software development, terminals are an important resource. The LynxOS utilities provide several methods to simplify and organize the job of installing new or different terminals, or adding extra serial communications lines.

Most of the work necessary to manage the terminals on a system involves updating some tables kept in the `/etc` directory.

Enabling Ports for login

The serial ports on a LynxOS system will be used by both terminals and other RS-232 devices, such as printers or machinery. Therefore only some of the serial ports should be recognized by `init` as being available for interactive use. On these ports, `init` will maintain an active process, usually `login`. Serial ports should all be listed in the `/etc/ttys` file. Each line of this file describes one port as shown below:

```
device:flag:config:terminal:login
```

The `device` field names a node for a serial port. The `port` is enabled for login (actually, enabled for consideration by `init`) if flag is non-zero. The `config` and `terminal` fields select a configuration and terminal type from the files `/etc/tconfig` and `/etc/termcap`. Both are described below.

Note that it is convenient to include all serial ports in this file, even those which are unlikely to ever be connected to an interactive terminal. As long as the `flag` field for these ports is `0`, they will never be bothered by `init`.

For each port enabled in `/etc/ttys`, `init` will start a program named by the login field of the description line for the port. The standard input, standard output, and standard error file descriptors (0, 1, and 2) will be connected to the port. For interactive software development, the `login` program will be `/bin/login`, although any program can be substituted (for example, a simpler version of login that ignores passwords).

`init` monitors activity of the child process it creates, and when it terminates another is started in its place. Pseudo-tty devices used by network and window-management software are also listed in the `/etc/ttys` file. They are always disabled. The entries in the file are used to “fool” `login` and provide terminal type information when `login` is started by a network daemon.

Describing Terminals

In order for the text editor and some other utilities to perform, the actual terminal device used with the system must be thoroughly described. Terminal characteristics and capabilities are stored in the `/etc/termcap` database.

Each different make or model of terminal used on a system should be included in the database. The details of a terminal description are rather involved; suffice to say that a list of common terminal capabilities is associated with corresponding strings and numeric values which pertain to the terminal. Terminal descriptions are fully described in the `termcap` file format reference pages.

The terminal type can be given in such relatively static situations as the terminal field in the `/etc/ttys` description of the port. If the terminal type for the port varies, as it possibly would on a modem port, the terminal field should be “dialup” or “unknown”.

The value of this field is placed in the environment by `login` as the value of the `TERM` environment variable. The `TERM` variable can be examined and possibly changed by the shell initialization scripts and is ultimately used by the editor or other programs to determine the terminal type.

Serial Port Configurations

The serial port driver, called the tty driver, includes facilities to control the details of input and output through a serial port. Aspects of communications which can be controlled include the baud rate, input character echoing, and the mapping of certain characters to driver-supported editing and process management functions.

All communication with the driver is (as with all drivers) through the `ioctl` system call. Because there are so many tty driver features, a program might have to call `ioctl` several times to configure the serial port for its needs. The LynxOS library routine `ttyconfig` simplifies this process by allowing the details of the configuration to be kept in the `/etc/tconfig` file.

Using `ttyconfig`, a program simply requests a configuration by name and the `ioctl` calls are performed automatically. At least one entry, called default, should be in the `/etc/tconfig` file. Others may be added, or the file can be forever ignored, but because `login`, `stty`, and `tset` use the information the file should exist. See the `ttyconfig(3)` man page in the *LynxOS Reference Manuals* for information on creating configuration entries.

CHAPTER 2

System Maintenance

This chapter describes the procedures for backing up and restoring data and for managing use of disk space.

Disk Space Management

Although LynxOS provides no disk space quota mechanism, the system manager can exercise arbitrarily tight control over resource consumption by periodically checking each user's space use. The `du` program reports space consumed under a particular directory. Assuming user accounts are all established with home directories in the `/usr` directory, the following `du` command reports consumption for each user:

```
du -s /usr
```

The values reported by `du` are in terms of kilobytes (KB). Note that `du` can be applied to any portion of the directory hierarchy, not just user-owned directories. The total amount of space available in a file system is stored in the super block. The `df` program reads the super block and reports the free disk space and optionally the number of free inodes.

df reports free space for each file system listed recursively therein. Specific file systems can also be specified on the command line. If disk space use is over 90%, any unused files should be eliminated or backed-up. Files which have not been accessed in a long while can be found with the command find as shown below:

```
find / (-atime +30 -o -mtime +30)
```

The expressions -atime +30 and -mtime +30 cause find to report all files which have not been read or written in more than 30 days.

With a similar command, files can be removed using find:

```
find / (-atime +30 -o -mtime +30) -print -exec
rm -f {} \;
```

The above is an example of a dangerous operation that will probably remove your entire operating system.

Backing Up Your System

Regardless of the physical reliability of hard disks, many opportunities exist for data loss due to software errors. For example, executing the previous find command will globally delete potentially critical files.

To backup data on your system, use the tar command. This command gathers sub-trees of the directory structure (or even the entire directory tree) into one file, usually on some removable device. Once the tar backup file has been created, tar can examine the file and selectively extract portions of the saved directory tree. As an example, to copy the entire contents of the /usr directory sub-tree to the tape drive, use the following command:

```
tar -cv /usr
```

The **cv** argument is called the key. In this case, the key directs `tar` to create (`c`) a new file on the output rather than skipping to the end of an existing file, and to give a file-by-file listing of files copied (`v`).

No output file was given in this case; normally, a system is configured such that the file `/dev/tar.default` is linked to the device name of the tape drive. If `tar` detects that it has run out of space on the output device, it will request that another storage unit be inserted. In order for `tar` to be able to detect end of media, the raw device interface must be used.

In the examples above, the `/usr` directory was referred to with an absolute pathname. The file names in the headers in the `tar` file will reflect the absolute pathnames. This implies that when files are later extracted, they will be forced into the exact position in the overall directory tree they originally occupied. If, on the other hand, the original `tar` had been performed as

```
cd /usr
tar -cv .
```

By first changing the working directory to `/usr` and then giving `tar` the pathname `.`, the files would be written with relative instead of absolute pathnames. They could then be extracted from the backup file and placed anywhere in the directory structure.

Recovery from Backups

Once a `tar` backup is made onto a tape, another disk drive, or to floppy disk, it can be used to restore the saved information in case of data loss. For example:

```
tar -xv
```

The **x** in the key means extract. All files in the backup volume will be copied to the main file system based on the pathnames in the headers. As each file is extracted, its name and other statistics will be printed. Individual files can be extracted by listing the desired pathnames after the key

```
tar -xv /usr/m5/mainprog.c /usr/m5/files.c
```

Each file whose name in its header matches any requested name will be restored.

Back-Up Policy and Procedure

All computer systems need the data on their disks to be backed-up However, the frequency of backups will depend on your particular system requirements. A back up should also be done whenever a new version of LynxOS is to be installed on the system. Possible media for backups include floppy disks and QIC 1/4 inch cartridge tapes. Tape drives must be SCSI devices.

The utilities `find` and `tar` can be used to locate the files and write to backup media. For example, all the files that have been modified in the last week can be backed up with the following commands:

```
find / -mtime -7 > /tmp/backup.list
tar -cvf /dev/rdevice `cat /tmp/backup.list`
```

Possible values for *rdevice* (raw device) are:

`rst1542.`*n*	1/4 inch cartridge tape drive attached to an Adaptec SCSI controller where *n* is the SCSI ID of the tape. (x86)
`rtape`*n*	1/4 inch cartridge tape drive attached to a 68k system SCSI controller where *n* is the SCSI ID of the tape device. (68k and SPARC)

`rfd1440.0`	3.5 inch 1.44 MB floppy drive. (x86)

System Files to Back-Up

The following is a list of system configuration files that should be backed up to tape whenever a backup is done. This list is not meant to be complete. For any individual system, some of the files listed will not have to be saved, while others that are not listed may need to be saved.

/.bashrc	/.cshrc
/.dlshrc	/.Xsession
/.exrc	/bin/rc
/.lesskey	/etc/exports
/.less	/etc/fstab
/.login	/etc/group
/.logout	/etc/hosts
/.motifbind	/etc/hosts.equiv
/.mwmrc	/etc/magic
/.profile	/etc/passwd
/.rhosts	/etc/printcap
/.subroutines	/etc/profile
/.twmrc	/etc/starttab
/.Xdefaults	/etc/ttys
/.xdm-config	/net/rc.network
/.xinitrc	/sys/lynx.os/CONFIG.TBL
/.Xresources	/sys/lynx.os/uparam.h
/.Xservers	/sys/cfg

CHAPTER 3

Customizing LynxOS

This chapter explains how to customize the LynxOS kernel for specific application requirements.

Why Customize the Kernel?

As you use LynxOS in your environment, and, as you develop your applications, you may need to customize the LynxOS kernel to:

- tune the performance of a LynxOS native development system
- remove functionality—such as unneeded device drivers—to conserve system resources
- add functionality—such as networking support or a custom device driver—to support a specific application

By default, the LynxOS kernel on the bootable installation media contains all supported disk and terminal drivers for a given hardware architecture. This allows a single bootable media to load LynxOS for initial installation on a range of hardware configurations.

The default boot kernel also has been configured to run on minimally configured systems. During the initial installation, you already encountered some basic “customizing” choices, such as whether to add networking support, that were used to configure the LynxOS kernel that you are running now.

This chapter explains how to customize LynxOS after initial installation. The LynxOS kernel files, located in the `/sys` directory, have been organized so that you can quickly and easily build customized LynxOS kernels.

Customizing for Performance

If you use LynxOS as your primary development system, you may want to customize the LynxOS kernel to improve performance. For example, you probably want your development system to handle a large number of processes at any one time. This is a configurable parameter that can be increased to meet your requirements.

This chapter will show you how to do this and other common performance tuning tasks such as increasing or decreasing the:

- total number of simultaneously-mountable file systems
- size of the disk cache
- maximum number of available processes

Customizing for Size

Your application environment may have size and memory constraints that make it critical for you to customize the LynxOS kernel. This chapter discusses what components can be removed from the kernel to reduce its size. It shows you how to:

- remove excess device drivers
- tune performance characteristics for a more minimal environment

For example, here at Lynx, we make a bootable LynxOS kernel image that is put onto a high-density floppy for product installation on x86 systems. We customize this installation boot kernel so that the kernel image, certain installation utilities (our application), and the RAM disk file system all fit in 1.44 MB. We also made sure that it can execute in at least 4 MB of memory.

Customizing for Functionality

The third common scenario involves customizing the functionality of the LynxOS kernel. This includes adding or removing Lynx supported drivers such as TCP/IP, NFS, STREAMS and the Simple Kernel Debugger (SKDB). You also may need to add your own custom device drivers for a particular application.

This chapter describes the structure of the LynxOS kernel directory and the process for adding in custom drivers to the build environment. For detailed information on developing custom device drivers, please see the guide *Writing Device Drivers for LynxOS*.

Overview of /sys

Read this section before you start modifying kernels to learn more about the organization of the files in the main kernel directory `/sys`.

The LynxOS `/sys` directory contains all the scripts, library archives, and platform-specific files needed to customize and rebuild a LynxOS kernel. It is organized as follows:

`Makefile`	Top level Makefile
`OBJ_RULES`	Makefile rules
`README`	File describing the `/sys` structure

`bsp.xxx/`	Default Board Support Package (BSP) for the main hardware platforms currently supported by LynxOS (`x86`, `ppc`, `usparc`, `030_M147`, `040_MVME`). This directory contains the parts of the LynxOS kernel that are particular to a given target board. Also, the actual kernel image is modified and linked in this directory.
`cfg/`	Specifies driver entry points for each device driver
`devices/`	Device driver information files
`dheaders/`	Device driver header files
`drivers/`	Device driver sources
`lib/`	Contains the following kernel and driver library archives:

`aux.a`	Auxiliary support routines
`hybrid.a`	Hybrid driver
`nfs_server.a`	NFS server code
`syscalls.a`	System call interface
`devices.a`	Devices
`kernel.a`	Kernel routines
`nullnfs.a`	Null NFS driver
`tcpip.a`	TCP/IP code
`drivers.a`	Device drivers
`nfs_client.a`	NFS Client code
`streams.a`	STREAMS code

`lynx.os@`	A symbolic link to the supported BSP

`romkit/`	Examples for building bootable kernel images
`stubs/`	"Stub files" for particular functionality modules that can override objects in `kernel.a`

Of these directories, you will most likely modify only the files in the `lynx.os` directory (a symbolic link to `bsp.xxx`). You can also modify the files in:

```
cfg/
devices/
dheaders/
drivers/
```

However, these files primarily are modified only when new drivers are added, or when the functionality of a specific driver needs to be changed.

Using the Main Kernel Directory—/sys/lynx.os

The main directory for customizing and rebuilding a LynxOS kernel is `/sys/lynx.os`. It is a symbolic link to the directory `bsp.`*xxx*, where *xxx* represents the board support package (BSP) that supports your hardware target (`x86`, `ppc`, `usparc`, `030_M147`, `040_MVME`).

The symbolic link from `lynx.os` to `bsp.`*xxx* provides the flexibility to add support for target hardware variants while maintaining a consistent build environment and access to supporting scripts via the `/sys/lynx.os` path.

Structure Overview of /sys/lynx.os

This directory contains the following files and subdirectories.

`CONFIG.TBL`	Control file for adding/removing device drivers
`CONFIG.h`	Automatically generated header file
`Makefile`	Target dependent `Makefile`

`Makefile.common`	Used by all LynxOS targets; included by `Makefile`
`*.o`	Pre-built object files
`info.c`	Contains the device information table
`conf.c`	Configuration file
`uparam.h`	User definable kernel parameters; the user-modifiable subset of parameters from the standard `/usr/include/param.h`
`*.h`	Supporting header files
`version.c`	LynxOS version file
`scripts/`	Shell functions that are used during a `make install.`*xxx*, `make uninstall.`*xxx*, or `make all.`*xxx*

Makefile Rules

To rebuild a kernel, change to the /sys/lynx.os directory and run the make utility with a specified rule.

`cd /sys/lynx.os`
`make` *rule*

The following `Makefile` rules are defined in `/sys/lynx.os/Makefile`:

`all`	Rebuilds `a.out` (LynxOS kernel) in the current directory.
`a.out`	Same as `make all`.
`clean`	Removes `a.out`, `CONFIG.h`, `sysdevices.h`, `nodetab`, and `timestamp.o`.
`install`	Rebuilds `a.out`, copies `/lynx.os` to `/lynx.os.old`, copies `a.out` to

	`/lynx.os`, and copies `nodetab` to `/etc/nodetab`.
`install.`*xxx*	Adds the *xxx* functionality (`tcpip`, `nfs`, `streams`, `skdb` or `all`) to `a.out`, then installs it (see `make install`).
`uninstall.`*xxx*	Removes the *xxx* functionality (`tcpip`, `nfs`, `streams`, `skdb` or `all`) from `a.out` then installs it (see `make install`).
`all.`*xxx*	Adds the *xxx* functionality (`tcpip`, `nfs`, `streams`, `skdb` or `all`) to `a.out`.

NOTE: *You should understand the implication of the "install" rule in the kernel Makefile. The process of "installing" a kernel involves copying the kernel to the root directory as /lynx.os and copying the device nodetab file to /etc/nodetab. The "installed" kernel, /lynx.os, is usually the default bootable kernel at system startup.*

Functionality Scripts

You can also use the scripts in `/usr/bin` to install or remove the major functional modules TCP/IP networking, network file system (NFS), STREAMS, and Simple Kernel Debugger (SKDB) support.

For example, to remove SKDB support from the kernel, enter:

```
/usr/bin/Uninstall.skdb
```

Adding/Removing Device Drivers with CONFIG.TBL

This file controls what device drivers are put into the kernel at build time. Each driver has a one line entry in this file. The comment character # is used to deactivate the device driver on that line. For example, this code fragment shows two lines from a `/sys/lynx.os/CONFIG.TBL` on a LynxOS x86 system:

```
# Secondary ide controller
I:hds.cfg
```

This means that the LynxOS kernel built with this `CONFIG.TBL` contains a device driver for the secondary IDE controller. To remove this device driver, you would modify the lines in `CONFIG.TBL` as shown below:

```
# Secondary ide controller
#I:hds.cfg
```

To reflect this change in the kernel, you will have to rebuild it.

CAUTION! *When a new kernel is built after changes have been made to* `CONFIG.TBL`, *the device nodes for the new kernel may not match the previous nodes. You should rebuild the device nodes after making the new kernel. See* `mknod(1)` *for more information.*

Whenever `CONFIG.TBL` is modified to build a new kernel, you must also rebuild the device nodes for the new kernel. Since the major and minor device numbers may change when edits are made to `CONFIG.TBL`, you may also have to modify the root device settings to boot LynxOS. This is shown in “Example—Removing Unused Device Drivers” on page 54.

Here are some guidelines for handling changes to `CONFIG.TBL`.

- If you add or remove a disk driver (usually specified at the beginning of `CONFIG.TBL`), you must determine the new major and minor number of your root disk device to boot the new kernel. See "Example—Removing Unused Device Drivers" on page 54.

- Since functionality modules, like TCP/IP drivers, are specified after the disk drivers, the major and minor number of the root device does not need to be changed to boot the new kernel.
- If you are creating a custom device driver, you should add it to the bottom of `CONFIG.TBL` so that the device node information for the preceding devices stays the same.

Making a Personal Kernel Build Directory

When working in a team project, you and the other developers on your team may need to maintain individual kernel build environments. The `/sys/lynx.os` directory was designed with this activity in mind.

The `lynx.os` directory (`bsp.`*xxx*) is "position independent." That is, it can be copied to any location on the file system and then used to build a "local" kernel (`a.out`) from the copy.

The `Makefile` in `lynx.os` uses the environment variable, `$ENV_PREFIX`, to find the library archives and object rules, but it is otherwise ignorant of its location on disk. `$ENV_PREFIX` defaults to the root directory (`/`) on native development systems. On cross development systems, the shell script `SETUP.`*shell* defines `$ENV_PREFIX` when executed. See "Customizing from a Cross Development Host" on page 42 for more information.

For example, try the following:

```
cp /sys/lynx.os /tmp/my_lynx.os
cd /tmp/my_lynx.os
touch CONFIG.TBL
make a.out
```

In this way, you (and other developers on your team) can have multiple kernel build directories on a given system, each having their own unique `CONFIG.TBL`, `nodetab`, and `uparam.h`.

CAUTION! *When building local LynxOS kernels, do not invoke the command* `make install`*. This option causes the resulting local kernel to overwrite* `/lynx.os` *and for the new node table to overwrite* `/etc/nodetab`*.*

The options `install.`*xxx*, `uninstall.`*xxx*, and `all.`*xxx*, can be used to create local copies of important system files such as `etc/hosts` and `net/rc.network`. These options prompt you to choose whether you want to modify the local or root build directory.

If you select local copies, the scripts copy files to your local build directory and make the appropriate modifications. You are then responsible for updating the appropriate files in the root directory if you want to boot the local kernel.

Customizing from a Cross Development Host

The LynxOS cross development distribution provides shell scripts to initialize key environment variables, namely `$PATH` and `$ENV_PREFIX`. These scripts, `SETUP.bash` and `SETUP.csh`, are located in `/usr/lynx/2.5.0/`*platform*, where platform is `x86`, `ppc`, `68k`, or `usparc`. Either script must be executed before developing with the LynxOS tools. For example:

```
cd /usr/lynx/2.5.0/x86
source SETUP.bash
```

With the exception of invoking a `SETUP` script first on a cross development system, *there is no discernible difference in developing LynxOS kernels from cross or native machines*. The same Makefiles and directory structure are used.

The design of the `/sys` directory and `Makefile` system provides this functionality. The following example demonstrates the similarities of developing kernels on a cross development and native LynxOS system.

Example—Adding TCP/IP to a LynxOS Kernel

On a Native LynxOS System

```
cd /sys/lynx.os
make install.tcpip
```

In this example, the LynxOS kernel (`a.out`) and the devices `nodetab` are copied to `$ENV_PREFIX` which is /on a native LynxOS system.

On a Cross Development System

```
cd /usr/lynx/2.5.0/x86
source SETUP.bash
```

This sets up the `$PATH` variable to point to the cross development tools, and sets `$ENV_PREFIX` to point to your installed LynxOS directory tree.

```
cd $ENV_PREFIX/sys/lynx.os
make install.tcpip
```

In this example, the LynxOS kernel (`a.out`) and the devices `nodetab` are copied to `$ENV_PREFIX`.

Tuning a Kernel for Performance

Changing the Parameters in uparam.h

The parameters defined in `/sys/lynx.os/uparam.h` tell the kernel how much space (dynamic memory) to set aside for various data structures at run-time. You can modify this file to increase (or decrease) the default values.

Table 3-1 lists some of the configurable parameters in `uparam.h`. You also can view the `uparam.h` file online using the `more` command or a text editor such as `vi`.

Table 3-1: Example Parameters in uparam.h

Parameter	Maximum Number of <Parameter>
CACHEBLKS	disk cache blocks
MAX_FAST_SEMS	fast semaphores
MAXSYMLINKS	symbolic links in a path
NBDEVS	dynamically-loaded block devices
NCDEVS	dynamically-loaded character devices
NDRIVS	dynamically-loaded drivers
NFILES	open files in system
NINODES	in cache inodes
NMOUNTS	mounted file systems
NPIPES	pipes
NPROC	processes
NRLOCKS	file locks
NSHLIB	shared libraries
NSOCKETS	sockets
NSTASKS	kernel threads

Table 3-1: Example Parameters in uparam.h

Parameter	Maximum Number of <Parameter>
NTHREADS	user threads
NUMIOINTS	registerable interrupt vectors via iointset()
QUANTUM	clock ticks until preemption
SMEMS	Lynx IPC named shared memory segments
USR_NFDS	open files per process
USR_PPSMEMS	shared memory segments per process
USEMS	Lynx IPC user semaphores
USYNCH	Lynx IPC pages user synchronization objects

The following tables show the default values, by platform, for the parameters in `uparam.h` based on the standard development system installation. Modifications to the values of these parameters not only influences system performance but also increases or decreases the dynamic size of the kernel. For this reason, the tables also show the sizes of each parameter.

NOTE: *Before making changes to* `uparam.h`, *read the comments in the files for the parameter that you want to change. Some parameters depend on others. For example, the value of* `NFILES` *must be greater than the sum of all the file parameters, including message queue, shared memory, and semaphore special files.*

Table 3-2: Default Values in uparam.h (LynxOS x86)

Parameter	Value	Size (bytes)	Total (bytes)
CACHEBLKS	256	564	144384
MAX_DATA_POOL	50	4096	204800
MAX_FAST_SEMS	512	16	8192
MAX_MSERVERS	5	20	100

Table 3-2: Default Values in uparam.h (LynxOS x86)

Parameter	Value	Size (bytes)	Total (bytes)
MAX_MSG_POOL	50	4096	204800
NAIOBUF	10	116	1160
NAIOSVC	5	52	260
NBDEVS	10	72	720
NCDEVS	20	84	1680
NDRIVS	15	88	1320
NFILES	256	220	56320
NMOUNTS	10	52	520
NMSGS	100	92	9200
NPIPES	50	80	4000
NPMSGS	5	120	600
NPROC	50	1133	56650
NPSHMS	5	76	380
NRLOCKS	140	24	3360
NSHLIB	10	96	960
NSOCKETS	100	180	18000
NSTASKS	20	681	13620
NTHREADS	50	681	34050
NVSEMS	20	48	960
NVSHMS	25	76	1900
SMEMS	10	60	600
USEMS	20	44	880
USYNCH	50	4	200

Table 3-3: Default Values in uparam.h (LynxOS PowerPC)

Parameter	Value	Size (bytes)	Total (bytes)
CACHEBLKS	256	564	144384
MAX_DATA_POOL	50	4096	204800
MAX_FAST_SEMS	512	16	8192
MAX_MSERVERS	5	20	100
MAX_MSG_POOL	50	4096	204800
NAIOBUF	10	116	1160
NAIOSVC	5	52	260
NBDEVS	10	72	720
NCDEVS	20	84	1680
NDRIVS	15	88	1320
NFILES	256	220	56320
NMOUNTS	10	52	520
NMSGS	100	92	9200
NPIPES	50	80	4000
NPMSGS	5	120	600
NPROC	50	1776	88800
NPSHMS	5	76	380
NRLOCKS	140	24	3360
NSHLIB	10	100	1000
NSOCKETS	100	180	18000
NSTASKS	20	1256	25120
NTHREADS	50	1256	62800
NVSEMS	20	48	960
NVSHMS	25	76	1900

Table 3-3: Default Values in uparam.h (LynxOS PowerPC)

Parameter	Value	Size (bytes)	Total (bytes)
SMEMS	10	60	600
USEMS	20	44	880
USYNCH	50	4	200

Table 3-4: Default Values in uparam.h (LynxOS 68k)

Parameter	Value	Size (bytes)	Total (bytes)
CACHEBLKS	256	564	144384
MAX_DATA_POOL	50	4096	204800
MAX_FAST_SEMS	512	16	8192
MAX_MSERVERS	5	20	100
MAX_MSG_POOL	50	4096	204800
NAIOBUF	10	116	1160
NAIOSVC	5	52	260
NBDEVS	10	72	720
NCDEVS	20	84	1680
NDRIVS	15	88	1320
NFILES	256	220	56320
NMOUNTS	10	52	520
NMSGS	100	92	9200
NPIPES	50	78	3900
NPMSGS	5	120	600
NPROC	50	1564	78200
NPSHMS	5	76	380
NRLOCKS	140	22	3080
NSHLIB	10	96	960

Table 3-4: Default Values in uparam.h (LynxOS 68k)

Parameter	Value	Size (bytes)	Total (bytes)
NSOCKETS	100	178	17800
NSTASKS	20	1004	20080
NTHREADS	50	1004	50200
NVSEMS	20	48	960
NVSHMS	25	76	1900
SMEMS	10	60	600
USEMS	20	44	880
USYNCH	50	4	200

Table 3-5: Default Values in uparam.h (LynxOS SPARC)

Parameter	Value	Size (bytes)	Total (bytes)
CACHEBLKS	1024	564	577536
MAX_DATA_POOL	50	4096	204800
MAX_FAST_SEMS	512	16	8192
MAX_MSERVERS	5	20	100
MAX_MSG_POOL	50	4096	204800
NAIOBUF	10	116	1160
NAIOSVC	5	52	260
NBDEVS	10	72	720
NCDEVS	20	84	1680
NDRIVS	15	88	1320
NFILES	450	220	99000
NMOUNTS	10	52	520
NMSGS	100	92	9200
NPIPES	50	80	4000
NPMSGS	5	120	600

Table 3-5: Default Values in uparam.h (LynxOS SPARC)

Parameter	Value	Size (bytes)	Total (bytes)
NPROC	150	1249	187350
NPSHMS	5	76	380
NRLOCKS	140	24	3360
NSHLIB	10	96	960
NSOCKETS	100	180	18000
NSTASKS	20	789	15780
NTHREADS	50	789	39450
NVSEMS	20	48	960
NVSHMS	25	76	1900
SMEMS	10	60	600
USEMS	20	44	880
USYNCH	50	4	200

Example—Increasing Max Processes

This example illustrates how to customize a LynxOS kernel to handle a greater number of processes. It assumes that you want to modify the default bootable kernel `/lynx.os`.

In this example, you will:

- Modify `/sys/lynx.os/uparam.h`
- Make a new kernel
- Reboot the system

Before you begin, make a copy of `uparam.h` that will be your backup in case you want to return back to your original settings.

```
cd /sys/lynx.os
cp uparam.h uparam.h.old
```

To increase the maximum number of processes, you need to increase the parameter `NPROC`. Use a text editor, such as `vi`, to open `uparam.h`. Find the parameter `NPROC` and increase the number.

```
#define NPROC     250     /* max number of processes */
```

To activate the changes, rebuild the kernel and reboot the system.

```
make install
reboot –a
```

Changing a Kernel's Size

Usually, you change the size of a kernel to decrease the amount of storage space it needs and/or the amount of memory it needs for execution. This section shows the components of the kernel that affect its size, both in memory and in storage.

In general, there are four basic components that you can modify to change the size of the LynxOS kernel:

- system parameters in `/sys/lynx.os/uparam.h`
- functionality modules (TCP/IP, NFS, etc.)
- device drivers
- symbol table information

Of these, `uparam.h` is the only component that is *dynamic*—changes to `uparam.h` will affect the kernel size in memory. The other components mainly affect the *static* size of the kernel—the amount of space it takes to store on a device. However, if you reduce the static size of the kernel, you usually reduce some of the dynamic size as well. The only exception is the symbol table information—removing this only affects the static size of the kernel.

After you determine what components to modify, you must make a new kernel to see the size changes take place. This process is similar to tuning a kernel for performance.

Determining the Size

As you experiment with changing the kernel, you'll want to know how to determine its size.

How much disk space?

An easy way to get the disk space size of the kernel is by using the `ls` command. For example:

```
ls -l /lynx.os
-rwxr-xr-x   1 root       877060 Oct 29 12:01 /lynx.os*
```

The kernel in this example is 877060 bytes, or about 856 KB.

You might also want to use the `size` command to look more closely at the sizes of the `TEXT`, `DATA`, and `BSS` segments of the kernel. For example:

```
size /lynx.os
text    data    bss     dec     hex     filename
688096  45056   142300  875452  d5bbc   /lynx.os
```

This utility reports the size (in bytes) of each segment and the sum of the segments. The total size of the file `a.out` in this example is 875452 bytes (or `0xD5BBC`).

Why does `size` report a different total than `ls`? In addition to the `TEXT` and `DATA` segments, the kernel contains header and symbol table information that fill up the extra 1608 bytes shown by `ls`.

How much memory?

Figuring out the memory usage of the kernel is more difficult. You can use the ps command to look at a currently running LynxOS kernel. Or, you can use the Simple Kernel Debugger (SKDB) to look at a either a running kernel or one that has been made.

The following example shows how to use ps to look at a currently running LynxOS kernel. On a LynxOS system, enter:

ps -axonT

```
pid tid ppid pgrp pri text stk data time dev   user S name
0   0   0    0    0   0    0   0    25:49      root R nullpr
1   7   1    1    16  20   8   12   2.44       root W /init
20  12  1    20   18  12   8   8    0.08 atc0  root W /bin/syncer
23  48  26   23   17  56   8   36   0.10       root R /net/rlogind
26  17  1    26   17  52   8   40   0.01       root W /net/inetd
28  18  1    28   17  40   8   52   0.00       root W /net/unfsio
31  19  1    31   17  44   12  32   0.03       root W /net/portmap
33  20  1    33   17  56   8   52   0.00       root W /net/mountd
35  21  1    35   17  44   8   40   0.00       root W /net/nfsd
37  22  1    37   17  64   8   44   0.00       root W /net/rpc.statd
39  23  1    39   17  60   8   60   0.00       root W /net/rpc.lockd.svc
41  24  1    41   17  60   8   68   0.02       root W /net/rpc.lockd.clnt
43  11  1    43   17  72   8   40   0.00       root W /bin/lpd
45  14  1    45   17  204  28  128  0.26 atc0  root W /bin/bash
46  27  1    46   17  204* 28  128  0.27 atc1  root W /bin/bash
47  30  1    47   17  204* 28  136  0.74 atc2  root W /bin/bash
48  33  1    48   17  204* 28  124  0.19 atc3  root W /bin/bash
63  45  46   63   17  340  8   72   0.02 atc1  root S /bin/vi
78  41  23   78   17  204* 28  124  0.08 ttyp0root W /bin/bash
119 54  45   119  17  40   8   36   0.08 atc0  root W /bin/rlogin
121 55  78   121  17  40   12  40   0.07 ttyp0root C /bin/ps
137 46  48   137  17  340* 12  72   0.04 atc3  root S /bin/vi
147 40  119  119  17  40*  8   36   0.06 atc0  root W /bin/rlogin

8352K/0K free physical/virtual, 2872K used (in this display)
```

This command displays all executing processes, including threads and unattached processes. It reports the total in kilobytes (KB) at the end of the listing. To calculate the size of the kernel:

1. Add the amount of free physical memory (8352 KB) to the amount used (2872 KB).

   8352 + 2872 = 11224 KB

2. Subtract this number from the total amount of memory (RAM) on your system. The machine in this example has 16 MB of memory, which translates to 16384 KB.

   16384 – 11224 = 5160 KB

Example—Removing Unused Device Drivers

One way to change to the size of a kernel is to remove unused device drivers, as described in this example.

The example uses a LynxOS x86 system that has only one Adaptec 2940 SCSI adapter to control its disk and CD-ROM drives. It doesn't need device entries (i.e. space used by the kernel) for other LynxOS supported SCSI devices drivers.

Before You Begin

For the purposes of this example, copy `/sys/lynx.os` to `/tmp/test`. (These commands *can* be used directly on the files in `/sys/lynx.os`.)

```
cp -r /sys/lynx.os /tmp/test
cd /tmp/test
```

Next, record some information about the current kernel that you are running. You should note the:

- size of `/lynx.os` (just to see the change)
- device name of the root device
- major and minor number of the root device

Use the `ls -l` (or `size`) command to determine the current size of `/lynx.os`.

```
ls -l /lynx.os
-rwxr-xr-x   1 root       698924 Oct 29 12:01 /lynx.os*
```

Use the `df` command to determine the device name of the root device. In this example, the root device is `/dev/sd2940.0a`.

```
df /
/dev/sd2940.0a   2048  127992  36979   91013  28%
```

Use the `ls -l` command to determine the major and minor number of the root device. In this example, the major number is 6 and the minor number is 16.

```
ls -l /dev/sd2940.0a*
brw-------   1 root  6,16  Oct  29 02:33 /dev/sd2940.0a
```

Modifying CONFIG.TBL

Use a text editor (such as `vi`) to modify `/test/CONFIG.TBL`. The edited fragment is shown in Figure 3-1.

```
# Supported LynxOS Adaptec Scsi drivers
#       for SCSI Manager-based Adaptec 152x driver
I:sim1522.cfg
#       for SCSI Manager-based Adaptec 154x driver
I:sim1542.cfg
#       for SCSI Manager-based Adaptec 174x driver
I:sim1742.cfg
#       for SCSI Manager-based Adaptec 274x driver
I:sim2742.cfg
#       for SCSI Manager-based Adaptec 294x driver
I:sim2940.cfg

#I:scsi810.cfg
#       for high-level-low-level based ncr 810 driver
```

Figure 3-1: Excerpt of Disk Driver Section in CONFIG.TBL

Building a New Kernel

Enter the following command (within `/tmp/test`) to create and install a new LynxOS kernel.

```
make install
```

After the new kernel (`/tmp/test/a.out`) is built, it is copied to `/lynx.os` and the new nodetab (`/tmp/test/nodetab`) is copied to `/etc/nodetab`. Notice that the former versions of these files are copied to `/lynx.os.old` and `/etc/nodetab.old`.

Look at the size of the new kernel. It is smaller than the previous one by about 36 KB.

```
ls -l /lynx.os*
-rwxr-xr-x   1 root    661669 Oct 29 12:01 /lynx.os*
-rwxr-xr-x  1 root   698924 Oct 29 12:01 /lynx.os.old*
```

Making New Device Nodes

First, make a copy of the old devices directory.

```
mv /dev /dev.old
mkdir /dev
```

Now build new device nodes that correspond with the new `/etc/nodetab`.

```
cd /dev
mknod -a /etc/nodetab
```

Loading the New Kernel

You *must* know the major and minor number of the root device of the new kernel before rebooting the machine. Notice that the major number changed to 2.

```
ls -l /dev/sd2940.0a
brw-------   1 root  2,16  Oct  29 02:33 /dev/sd2940.0a
```

CAUTION! *If you specify the wrong major and minor number as the root device, you may see the error message "Main file system failed to install (no dev)." or the kernel may crash.*

Reboot the system.

```
reboot -a
```

At the `preboot` command prompt (the bootstrap kernel loader program) specify the new major and minor number of the root device. See `preboot(1)` for a description of the commands available on your hardware platform.

```
Command? R 2 16
Setting root device to 2 16.

Command? b
```

Updating the Default Root Device

Now that the new kernel is up and running, you can update the default root device with the `makeboot` command. This allows you to directly boot LynxOS without using the **R** command in `preboot`.

```
makeboot preboot
```

Adding Functionality to a Kernel

Example—Adding a Custom Device Driver

Many LynxOS customers have developed device drivers that are specific to their target hardware. With LynxOS 2.5, we have modified the `/sys/lynx.os/Makefile` in an attempt to better support this development activity. In this `Makefile`, you will see several `Makefile` environment variables you can modify to explicitly specify custom device drivers, patches, or other objects that you want to include in the kernel.

We recommend the following approach for adding a custom driver to a LynxOS kernel. Substitute file and directory names where appropriate.

1. Create a custom driver directory in `drivers`.

   mkdir drivers/my_driver

2. Use an existing device driver `Makefile` as a template.

   cp drivers/mem/Makefile drivers/my_driver

3. Modify the `Makefile` so that during a kernel build, your driver will be compiled into a unique archive. For example:

   FILES = *file1*.x *file2*.x *file3*.x
   LIBRARY=my_driver

   These lines instruct the `Makefile` to compile files `file*.c`, and to insert the compiled `.o` files in `sys/lib/my_driver.a`. You can look at the `OBJ_RULES` file in `/sys` to determine the substitutions that are appropriate for your compilation of the `.c` file. For example, `.x` will compile an optimized ANSI `.c` file, `.u.x` will not optimize the `.c` file.

4. Modify the `Makefile` in `lynx.os` so that the archive for your device driver will be linked into the kernel.

   ```
   CUSTOMER_DRIVERS=$(LIB)/my_driver.a
   ```

5. Add the configuaration file `my_driver.cfg` file for your driver in `/sys/cfg`. (See *Writing Device Drivers for LynxOS* for more information about device configuration files.)

6. Edit `CONFIG.TBL` so that your new driver is represented. You should always add a new driver to the end `CONFIG.TBL` so that the major, minor numbers for existing device drivers remains the same.

```
I:my_driver.cfg   # for instance
```

7. Build the LynxOS kernel with the new device driver.

```
cd /sys/lynx.os
make a.out
```

CHAPTER 4

TCP/IP—Networking Protocols

LynxOS supports Transmission Control Protocol/Internet Protocol (TCP/IP) networking. During your initial installation of LynxOS, you may have chosen to install TCP/IP support from the menus within the `installit` utility. However, you also can configure, install, or remove TCP/IP support at any time. This chapter describes some basics about TCP/IP, and some advanced TCP/IP configuration options.

Overview

LynxOS TCP/IP features standard TCP/IP data transfer services. It provides the BSD (Berkeley Software Distribution) socket interface system and networking library functions needed to access the underlying network protocols, TCP, IP, and UDP (User Datagram Protocol).

Installing/Removing TCP/IP Support

You can install and/or remove TCP/IP support after initial installation of LynxOS. First check with your Network Administrator to get a unique hostname and IP address for your system.

To install:

```
cd /sys/lynx.os
make install.tcpip
```

To remove:

```
cd /sys/lynx.os
make uninstall.tcpip
```

For more advanced information on these steps, see Chapter 3, “Customizing LynxOS”.

Some Common TCP/IP Utilities

This section is an overview of some of the most common TCP/IP utilities. These pages are intended to introduce TCP/IP utilities to the novice user, discussing the rudiments of remote computer access and file transfer between computers. For a complete discussion of each utility, refer to the appropriate man pages in the *LynxOS Reference Manuals*.

For the sake of clarity in these examples, the prompt will be given showing the user name, the host name, and a trailing `$`. The following example shows the prompt for user `jones` on host computer `shark`:

```
jones@shark$
```

Testing TCP/IP (ping)

The simplest way to test the TCP/IP setup of your system is to use the `ping` utility. `ping` can send a test message to any addressable host. This does not exercise application software at the addressed host, but it does verify the correct operation of the hardware and the TCP/IP software connecting the two hosts. `ping` will continue to send packets to the addressed host once every second until its execution is terminated with a `^C`.

In the following example, the host `shark` will verify that its TCP/IP software and hardware are configured correctly by `ping`ing itself as shown in Figure 4-1.

```
$ ping shark
PING shark (193.1.1.1): 56 data bytes
64 bytes from 193.1.1.1: icmp_seq=0.time=10. ms
64 bytes from 193.1.1.1: icmp_seq=1.time=0. ms
64 bytes from 193.1.1.1: icmp_seq=2.time=0. ms
64 bytes from 193.1.1.1: icmp_seq=3.time=0. ms
^C
----shark PING Statistics----
4 packets transmitted, 4 packets received, 0%
packet loss round-trip (ms)  min/avg/max = 0/2/10
$
```

Figure 4-1: Verifying TCP/IP Setup with ping

Once assured of the correctness of its own setup, each host in the network can ping the other members of the network as shown in Figure 4-2.

```
$ ping orca
PING orca (193.1.1.2): 56 data bytes
64 bytes from 193.1.1.2: icmp_seq=0.time=10. ms
64 bytes from 193.1.1.2: icmp_seq=1.time=0. ms
64 bytes from 193.1.1.2: icmp_seq=2.time=0. ms
64 bytes from 193.1.1.2: icmp_seq=3.time=0. ms
^C
----orca PING Statistics----
4 packets transmitted, 4 packets received, 0%
packet loss round-trip (ms)  min/avg/max = 0/2/10
$
```

Figure 4-2: Pinging Other Hosts on a Network

If the host being pinged is not properly declared in the `/etc/hosts` file, the `ping` will fail as shown in Figure 4-3. Edit the `/etc/hosts` and add `fish`.

```
$ ping fish
ping: fish: Host name lookup failure
$
```

Figure 4-3: Host Not Defined in /etc/hosts

If the addressed host is not connected or is incorrectly configured, the system will not respond. You should terminate the program (by pressing **Ctrl-C**) after a couple of seconds if no output is generated by the `ping` utility as shown in Figure 4-4.

```
$  ping orca
PING orca (193.1.1.209): 56 data bytes
^C
----orca PING Statistics----
4 packets transmitted, 0 packets received, 100%
packet loss
$
```

Figure 4-4: Host Not Available on Network

The three common reasons for this to happen are:

1. The machine `orca` is not connected to the network.
2. The machine `orca` is down.
3. The `/etc/hosts` file on `shark` has the wrong Internet address for `orca`.

Logging Into a Remote Computer (telnet, rlogin)

The user can login into another host on the network using either of the two utilities supplied with TCP/IP: `telnet` and `rlogin`.

telnet

`telnet` allows a user to log into any type of computer that supports TCP/IP. The computer can run any operating system, UNIX-compatible or not. This utility allows a LynxOS user to access a mainframe or a PC as if the user were on that computer's local console.

`telnet` is invoked with a single parameter—the hostname of a remote computer, as shown in Figure 4-5.

```
jones@orca$ telnet shark
Trying...
Connected to shark.
Escape character is '^]'.

LynxOS (shark)

user name: jones
password:

jones@shark$
```

Figure 4-5: Using telnet to Remotely Login

Like a locally-attached terminal, access to the remote computer is guarded by the `login` program.

A `telnet` session is terminated by logging out of the system the same way as logging out from a local session, as shown in Figure 4-6.

```
jones@shark$ exit
Connection closed by foreign host.

jones@orca$
```

Figure 4-6: Terminating a telnet Session

rlogin

`rlogin`, like `telnet`, is used to remotely log into another computer. Unlike `telnet`, `rlogin` requires the host computer to have a UNIX-compatible operating system. Additionally, `rlogin` contains greater functionality than `telnet`.

`rlogin` can be invoked with a single parameter—the host name of the remote computer, as shown in Figure 4-7.

```
jones@orca$ rlogin shark
login shark vt100
password:

jones@shark$
```

Figure 4-7: Using rlogin to Remotely Login

In the previous example, no user name was passed to `rlogin`. In this case, the user name on the local machine is used to log into the remote machine; i.e., user `jones` on host `shark` logged onto remote host `orca` as user `jones`.

If the user wants to login into an account which does not have an identical login account, an additional parameter—the login account—must be passed, as shown in Figure 4-8.

```
jones@orca$ rlogin shark -l doc
login shark vt100
password:

doc@shark$
```

Figure 4-8: Remote Login as Another User

Unlike `telnet`, the `rlogin` utility can take advantage of the information in `/etc/hosts.equiv` and `.rhosts` files. Users on machines that have been set up to be considered local will not be prompted for a password.

Executing Commands Remotely (rsh)

Another utility included with TCP/IP is the `rsh` command—the remote shell command. This utility allows the user to access remote hosts, to execute a command on that host, and to have the output redirected to the local machine. `rsh` only works when the user is considered equivalent on the local and remote machine.

The syntax for this command is:

`rsh host[-l` *user_name*`]` *command*

An optional user name is given, if it is desired, to execute the command as a specific user. This is useful if your current user account is not considered equivalent on the remote machine. `rsh` redirects standard input, standard output, and standard error from the remote machine to the local host.

You can use the `who` utility to find out who is currently logged into a remote host, as shown in Figure 4-9.

```
davis@shark$ rsh orca who
root   atc0          Mon Dec 23 10:40:54
root   ttyp0  :0.0   Mon Dec 23 10:41:19
root   ttyp1  :0.0   Mon Dec 23 10:49:02
root   ttyp2  :0.0   Mon Dec 23 11:05:45
davis@shark$
```

Figure 4-9: Using who to Query Logins on a Remote System

Also, you can invoke a command on a remote host as another user. In Figure 4-10, user `jones` on host `orca` will invoke the `whoami` command. This utility reports the current login account.

```
davis@shark$ rsh orca -l jones whoami
jones

davis@shark$
```

Figure 4-10: Using rsh to Remotely Execute a Utility

Transferring Files Between Machines (ftp, rcp)

File transfer using TCP/IP is fast and efficient. Like the remote login utilities previously discussed, there are two ways to copy files between hosts, `ftp` for hosts of differing operating systems, and `rcp` for UNIX compatible operating systems.

ftp

`ftp` is an acronym for File Transfer Protocol. This program is very powerful and provides many ways to configure options for your application. After reading this brief introduction, explore the man page for the `ftp` utility and experiment with some of the more advanced options.

In its simplest invocation, `ftp` is called with a single parameter—the host name of the remote machine. `ftp` prompts for a login ID and password. In Figure 4-11, user `davis` on host `orca` will connect to host `shark` and login as user `jones`.

```
davis@shark$ ftp orca
Connected to orca.
220 orca FTP server (Version 4.162 Tue Nov 1
10:50:37 PST 1988) ready.
Name (orca:davis): jones
331 Password required for jones.
Password:
230 User jones logged in.
ftp>
```

Figure 4-11: Connect to a Remote System with ftp

Once connected, a user can retrieve files from the remote host using the `get` command, as shown in Figure 4-12.

```
ftp> get .login
200 PORT command successful.
150 Opening ASCII mode data connection for .Login
(209 bytes).
226 Transfer complete.
218 bytes received in 0.01 seconds (21.29KB/s)
ftp>
```

Figure 4-12: Downloading Remote Files with ftp

The user can send files to the remote host using the `put` command, as shown in Figure 4-13.

```
ftp> put hosts.equiv
200 PORT command successful.
150 Opening ASCII mode data connection for
hosts.equiv.
226 Transfer complete.
12 bytes sent in 0.07 seconds (0.17KB/s)
ftp>
```

Figure 4-13: Uploading Remote Files with ftp

NOTE: *The default transfer mode is ASCII. If binary (executable) files are being transferred,* `ftp` *must be explicitly configured using the* `binary` *command.*

To transfer binary (executable) files, you must change the transfer mode of `ftp` using the `binary` command. For example:

```
ftp> binary
200 Type set to I.
ftp>
```

A binary file transferred as an ASCII file will have the same number of bytes, but the byte order will be corrupted. To preserve the integrity of a binary file, you should set `ftp` to binary mode. A drawback of this transfer mode is that binary file transfers are more time-consuming than ASCII file transfers.

A list of commands can be produced by entering a ? at the prompt, as shown in Figure 4-14.

```
ftp> ?
Commands may be abbreviated. Commands are:
!         delete    mdelete    proxy    runique
$         debug     mdir       send     portsend
account   dir       mget       put      size
append    disconnect mkdir     pwd      status
ascii     form      mls        quit     struct
bell      get       mode       quote    system
binary    glob      modtime    recvs    unique
bye       hash      mput     remotehelp tenex
case      help      nmap       rstatus  trace
cd        image     nlist      rhelp    type
cdup      lcd       ntrans     rename   user
close     ls        open       reset    verbose
cr        macdef    prompt     rmdir    ?
ftp>
```

Figure 4-14: Listing of ftp Commands

rcp

For quick exchange of data between UNIX-compatible hosts, `rcp` is a very fast and efficient method. To be able to access files using `rcp`, the `/etc/hosts.equiv` and `.rhosts` files must be set up correctly. Assuming this has been done, a user can copy files between hosts using a syntax similar to the UNIX `cp` command.

The only difference is that the remote host's file name must be indicated properly. In the following example, file `/etc/hosts` will be copied from host `orca` to host `shark`:

```
jones@shark$ rcp orca:/etc/hosts /etc
```

Like the `cp` command, multiple files can be transferred to a directory:

```
jones@shark$ rcp /etc/passwd /etc/printcap orca:/etc
jones@shark$
```

Finally, the `rcp` command can be used to transfer files between two hosts that are different than the host you are currently logged into (assuming proper configuration). In the following example, the `/etc/passwd` file will be copied from host `orca` to host `fish` from a user on host `shark`:

```
jones@shark$ rcp orca:/etc/passwd fish:/etc/passwd
jones@shark$
```

Installing and Configuring Ethernet Cards on x86

The following sections describe how to install and configure Ethernet cards on x86 systems.

WD and SMC Ethernet Cards

Follow these steps to set up the configurable parameters for the Ethernet card to work under LynxOS:

- Put the jumper `W1` in `SOFT` setting.
- Place the Ethernet card in a free slot and boot LynxOS.
- Invoke `ezsetup`.
- Choose the following values for the parameters:

IOBASE	0x240
RAMBase	0xCC000
IRQ Level	5

The RAMBase address can be changed if there is a conflict with any other board address; in which case, edit the `/sys/devices/if_wd3einfo.c` file to reflect the new value and remake the kernel. Make sure that the selected RAM area lies within the supported address range by the Ethernet card (ideally, within a `0xC0000 - 0xDFFFF` range). Also, be sure to use `BIOS-SETUP` (where applicable) to disable the "Adaptor ROM shadow" for the RAM address range used by the Ethernet card.

Note that if your node is configured as a diskless LynxOS node, the address range `0xD0000` to `0xDFFFF` is used up by the `TFTPBOOT` PROM.

The field `enable_16bit` in the `/sys/devices/if_wd3einfo.c` file applies only to 16-bit adapters in the 16-bit slot. Setting this field to `1` enables 16-bit access to the card; but this disables any other 8-bit card in the 128KB address segment which overlaps the Ethernet card's RAMbase address (segment `0xC0000 - 0xDFFFF` for default value). Only 16-bit cards will be supported in this address range. This is due to the inherent limitation of the AT bus.

3COM Cards for ISA/EISA Bus

In case of 3C509 adapter in an ISA Bus, please follow the instructions given in the adapter manual chapter, *Configuring the 3C509 or 3C509-TP Adapter for an ISA Computer*. You will need to boot DOS to be able to software configure the adapter. If you are using this adapter in an EISA machine, follow the instructions given in, *Configuring the ISA (3C509 and 3C509-TP) Adapter for an EISA Computer* to do the setup.

In case of the 3C579 EISA adapter for EISA bus, please setup the card following instructions given in the adapter manual chapter, *Configuring an EISA (3C579 and 3C579-TP) Adapter in an EISA Computer*. You will need the EISA configuration diskette of the computer to configure this adapter.

Driver Defaults

The following list shows the default values within the driver info files. The files are located in `/sys/devices`. To change the defaults, you will need to edit the file, re-compile it, and install TCP/IP support again. For more information, see “Installing/Removing TCP/IP Support” on page 61.

Info File	Target	Defaults	Notes
`hbtcpip_info`	all	Mbufs (multiple of 4): 512 clusters: 128 disable checksum for UDP packets: 1	This file contains the amount of memory to be allocated for TCP/IP driver usage. Note that the recommended ratio of Mbufs to clusters is 4:1.
`if_3c509`	x86	iobase: 0x240 interrupt: 5 slot number 0 TP = 0, AUI = 1, BNC = 3 (default 3)	The TCP/IP driver for the 3COM 3C509 adapter.
`if_3c579.c`	x86	iobase: 0x3f0 interrupt: 5 slot number 7 TP = 0, AUI = 1, BNC = 3 (default 3)	The TCP/IP driver for the 3COM 3C579 adapter.
`if_wd3e.c`	x86	iobase: 0x240 on board ram address: 0xcc000 interrupt: 5 16 bit access 1	The TCP/IP driver for Western Digital and SMC cards.
`if_dec21040info.c`	x86	AUI/BNC = 0, TP=1 (default 0), 100Mbit = 2[1]	The TCP/IP driver for the SMC EtherPower (PCI) card.

1. The 100Mbit interface is supported by the SMC EtherPower 10/100 LAN controller.

Notes

- On the x86, if you change the RAMBase, make sure that the selected RAM area lies within the supported address range by the Ethernet card (ideally, within a `0xC0000 - 0xDFFFF` range). Also, be sure to use BIOS-SETUP (where applicable) to disable the "Adaptor ROM shadow" for the RAM address range used by the Ethernet card.

- In the hbtcpip_info.c file, you can enable/disable the checksum calculation for UDP packets by setting the udpcksum filed to 1 (enable) or 0 (disable). Disabling the checksum will increase the throughput for UDP packets.
- The Ethernet drivers available on LynxOS for SPARC, both oblan and hybrid, perform automatic switching between the twisted-pair (TPE) and AUI interfaces. In order to sense the presence of an AUI, SQE needs to be enabled (see your network administer or the transceiver documentation). The TPE has link status built in so nothing special needs to be done.

Hybrid Ethernet Driver

The Hybrid Ethernet Driver is an Ethernet driver facility that supports both the STREAMS-based Data Link Provider Interface (DLPI) and the BSD Network Interface for TCP/IP support. It is provided on the x86, SPARC, and PowerPC. The Hybrid Ethernet Driver consists of three components:

- A STREAMS-based Ethernet driver
- A BSD-compatible Pseudo Network Interface Driver
- An utility called dlbsd

The STREAMS-based Ethernet driver implements Version 2 of the DLPI specification and supports the Connectionless Mode Service Interface. It also has hooks to interface with a BSD-compatible Pseudo Network Interface driver for TCP/IP, ARP and RARP protocol messages.

The BSD-compatible Pseudo-Network Interface driver performs the function of interfacing the BSD-based TCP/IP protocol code with the STREAMS-based driver for TCP/IP protocol network traffic. This driver is accessible to all the BSD networking utilities like `ifconfig`, `netstat` and others. When the system is booted, this device is detached from the STREAMS-based Ethernet driver.

The `dlbsd` utility provides the capability to attach the Pseudo BSD Network Interface device to the STREAMS Ethernet device. Once this is done, the `ifconfig` utility is used to activate the Pseudo BSD Network Interface device for use by the BSD networking applications.

The `dlbsd` utility can also be used to detach the Pseudo BSD Network Interface from the STREAMS Ethernet device, and also to check the status of this “attachment” of the STREAMS Ethernet device.

x86 and PowerPC

The Hybrid Ethernet Interface is supported with the SMC EtherPower Ethernet controller for the PCI bus on the x86. On the PowerPC, it is supported with the on-board DEC21040 Ethernet controller.

On the x86 and PowerPC, the driver code is contained in two object modules— `dlpi.o` and `dlpi_21040.o`. In addition, `dlpi_21040info.o` contains the user-configurable parameters for this driver. Note that the driver configuration file `dlpi_21040.cfg` is set up to associate the node `/dev/dlpi_21040` with this device and the name `dlpi_21040` with this driver and device.

The BSD-compatible Pseudo Network Interface driver is contained in the object module `bdlan.o`. The BSD network interface device supported by this driver is called `bdlan`. Note that the driver configuration file `bdlan.cfg` is set up to associate the interface name as `bdlan` and the driver name as `bdlan`.

SPARC

The Hybrid Ethernet Interface is supported with the on-board Ethernet controller on the SPARC. The driver code is contained in two object modules— `dlpi.n.o` and `dlpilan.n.o`. In addition, `dlpi_laninfo.n.o` contains the user-configurable parameters for this driver. Note that the driver configuration file `dlpilan.cfg` is set up to associate the node `/dev/dlpilan` with this device and the name `dlpilan` with this driver and device.

The BSD-compatible Pseudo-Network Interface driver is contained in the object module `if_bsdlan.n.o`. The BSD network interface device supported by this driver is called `bdlan`. Note that the driver configuration file `if_bsdlan.cfg` is set up to associate the interface name as `bdlan` and the driver name as `bsdlan`.

When the `dlpilan` driver is installed to manage the Hybrid Ethernet Interface, be sure to remove the `oblan` driver as described in “Installing/Removing TCP/IP Support” on page 61.

Tuning the Hybrid Ethernet Driver Configuration

STREAMS Ethernet Driver

On the x86 and PowerPC, the `dlpi_21040info.h` file in `/sys/dheaders` defines a structure `dlpi_dcinfo0` to specify the tunable parameters for the STREAMS Ethernet Driver. The first four parameters specify the resources for the DLPI operations. The last parameter specifies the network connection for the Ethernet controller. The DLPI parameters can be modified depending on the system requirements.

On the SPARC, these parameters are defined by the `dlpi_laninfo` structure in `/sys/dheaders/dlpi_laninfo.h`. The first three parameters in this structure specify the hardware configuration and the last four specify the resources for DLPI operations.

The actual values of these parameters are defined in:

`/sys/devices/dlpi_21040info.c` (x86, PowerPC)

`/sys/devices/dlpi_laninfo.c` (SPARC)

The following is a description of the parameters defined in the above structures.

`XMITQ_LIMIT` Specifies the maximum number of packets which can be queued for transmission when the Ethernet Controller's transmit buffers are all busy. The packets are dropped when this limit is exceeded.

Default value is 50.

`STREAMS_LIMIT` Specifies the maximum number of streams that can be opened to the Ethernet driver at the same time. The amount of memory required for each stream depends on the size of the `dlpidev` structure defined in the `dlpidev.h` file and the values of the `SUBSAP_LIMIT` and `MCAST_LIMIT` parameters described below.

Default value of `STREAMS_LIMIT` is 16.

`SUBSAP_LIMIT` The DLPI specifications allow for binding multiple SAP addresses to the same stream. Each address takes up 4-bytes of memory. This parameter specifies the maximum number of addresses that can be bound to a stream using the `DL_SUBS_BIND_REQ` primitive.

Default value of `SUBSAP_LIMIT` is 8.

`MCAST_LIMIT` The DLPI specifications allow for enabling multiple multicast addresses for each stream. Each

	address take up 6-bytes of memory. This parameter specifies the maximum number of multicast addresses that can be enabled for a stream using the DL_ENABMULTI_REQ primitive. Default value of MCAST_LIMIT is 8.
XCVR_TYPE	(x86, PowerPC) Specifies the network connection for the Ethernet controller. The default value is 0 for AUI/BNC connection.
LAN_PORT	(SPARC) The Ethernet controller port address.
LAN_VECTOR	(SPARC) The Ethernet controller interrupt vector.
DMA_ADDRESS	(SPARC) The Ethernet controller DMA address.

Pseudo BSD Network Interface

In /sys/devices, the files if_bdlaninfo.c (x86, PowerPC) and bsdlan_info.c (SPARC) define the structure if_bdlan_info to specify user-configurable parameters for the Pseudo BSD Network Interface. This structure definition contains just one parameter that is a pointer to the name of the STREAMS Ethernet device to which it can be attached for TCP/IP packet transmission and reception.

CHAPTER 5

NFS—Network File System

During your initial installation of LynxOS, you may have chosen to install NFS support from the menus within the `installit` utility. You can also configure, install, or remove NFS support at any time. This chapter describes some basics about NFS, and some advanced NFS configuration options.

Overview

NFS is a suite of user programs and kernel functionality that allows the user access to a remote host's hard drives as if they were local hard drives. All or part of a remote host's file system is mounted into the local host's file system, allowing transparent access to remote files by local users. Once mounted, any file on the remote file system is accessible with a pathname. Such files can be operated on by most utilities, functioning no differently than a file located on the local disk.

In addition to basic file system accesses over NFS, LynxOS also supports client-side caching and file locking over NFS. NFS client-side caching supports the caching of directory entries, file attributes and file data information depending on the type of access. File locking support facilitates advisory record locking between cooperating processes over NFS. This facility also supports monitored class locking, and is thus able to recover from NFS server/client crashes.

NFS software is divided into two parts, the NFS server and the NFS client. When a user attempts to access a file in an NFS-mounted directory, the NFS client sends a request to the NFS server on the remote system. The NFS server accepts and manages these requests from the remote NFS client for access to the local disk. The server enforces permissions and performs the actual manipulations of the local disk.

Installing/Removing NFS Support

You can install and/or remove NFS support after initial installation of LynxOS.

To install:

```
cd /sys/lynx.os
make install.nfs
```

To remove:

```
cd /sys/lynx.os
make uninstall.nfs
```

For more advanced information on these steps, see Chapter 3, “Customizing LynxOS”.

Tuning the NFS Server Kernel

The NFS server can be tuned by decreasing or increasing the values of five parameters. These parameters are within the structure `nfssvc_info` that is located in `/sys/devices/nfssvc_info.c`.

To change any of these parameters, edit the file `/sys/devices/nfssvc_info.c`. You must then update the device library and rebuild the LynxOS kernel.

NOTE: *The* `/etc/exports` *file specifies the directories to be exported and the corresponding access list.*

NFS Server Tunable Parameters

1. *The maximum number of directories that can be exported.*

   The default is 16. If the system is used as a file server, this parameter may need to be increased.

2. *The maximum number of hosts that can be specified in the access list of an exported directory.*

   The default is 16.

3. *The maximum number of hosts that can be specified with root access for an exported directory.*

   The default is 16. For security reasons, you may wish to decrease this parameter.

4. *The maximum number of hosts that can be specified with read and write access for an exported directory.*

   The default is 16. For security reasons, you may wish to decrease this parameter.

5. *The maximum number of NFS server daemons that can be started at any time.*

   The default is 8. If the system is used as a file server, you may want to start multiple daemons. This can be done by adding a count parameter to the line `/net/nfsd` in `/net/rc.network`. For example, to start three NFS server daemons, modify the line to be `/net/nfsd 3`.

Tuning the NFS Client Kernel

The NFS client can be tuned by changing the values of six kernel parameters. The structure `unfs_info`, found in `/sys/devices/nfsinfo.c`, contains six tunable parameters:

1. *The maximum number of NFS file nodes that can be open at any time.*

   The default is 64. The value of this parameter should be increased for heavy NFS traffic.

2. *The maximum number of NFS directories that can be mounted.*

   The default is 8. If this value is increased, make sure that `NMOUNT` in `/sys.os/uparam.h` is also increased to an equal or greater value.

3. *The maximum number of NFS client daemons that can be started at any time.*

   The default is 32. In case of heavy NFS client traffic, you may want to start multiple client daemons. This can be done by duplicating the `/net/unfsio` line in `/net/rc.network`.

4. *The maximum number of NFS client requests that can be in the queue at any time.*

   The default is 32. The value of this parameter should be increased for heavy NFS traffic.

5. *The maximum number of bytes in an NFS read/write request.*

   The default is 8192. This value should be reduced to 4096 or less to interface with systems that have slower (i.e. 8-bit) Ethernet boards.

Edit the `/sys/devices/nfsinfo.c` file to change any of these parameters. Be sure to change only the values. After making the desired changes, you must then update the device library and rebuild the LynxOS kernel. These steps are described in Chapter 3, “Customizing LynxOS”.

Tuning the NFS Client-side Cache

The amount of memory allocated for NFS client side caching can be tuned by decreasing or increasing the values of the various fields in the `nfs_cache_info` structure in `/sys/devices/nfs_cache_info.c` file.

```
struct nfs_cache_info {
  int num_attrcache;      /* num attribute cache blocks */
  int num_dnlchdrs;       /* num of dir name lookup
                                       cache headers */
  int size_dnlcblk;   /* size of each dnlc cache block */
  int num_datacache;          /* num data cache headers */
  int size_datacache;  /* amount memory for data cache */
  int hash_tblsize;    /* hash table size for data/dnlc
                          blocks                        */

/* default mount-time options */
    int blksize;                /* caching block size */
    int dis_cache;             /* disable all caching */
    int dis_datacache;    /* disable file data caching */
    int acregmin;    /* min time in secs for reg files */
    int acregmax;    /* max time in secs for reg files */
    int acdirmin;    /* min time in secs for dir files */
    int acdirmax;    /* max time in secs for dir files */
};
```

The various fields in this structure are explained below:

`num_attrcache` — This parameter controls the total number of simultaneous file/directory entries that will be allowed to be cached in this client. Each entry for this parameter consumes 180 bytes. Default value for this field is 512 (total memory consumption 512*180 bytes).

`num_dnlchdrs` — This parameter controls the maximum number of directory name lookup buffers that are cached in the client. Each entry corresponding to this field consumes 32 bytes. The default value for this field is 128 (total memory consumption 32*128 bytes). Depending up on the size of

	the consumes 32 bytes. The default value for this field is 128 (total memory consumption 32*128 bytes). Depending up on the size of the directory files, more than one entry may be used for each directory.
`size_dnlcblk`	This parameter controls the amount of buffer allocated for each directory name lookup cache header, to store the file name and it's NFS File Handle under that directory. The default value for this field is 512. Each file name entry associated with the directory takes 40+<FileNameLen> bytes.
`num_datacache`	This parameter indicates the number of data cache headers that will be allocated for the NFS cache. Effectively, this parameter controls the total number of Data blocks (of variable size) that can be cached by the driver. Each entry corresponding to this field consumes a total of 96 bytes. The default value for this field is 128 (total memory consumption 128*96 bytes).
`size_datacache`	This parameter controls the total amount of memory in Kilobytes that will be allocated for Data caching. The default value for this field is 512 (memory usage 512KB).
`hash_tblsize`	This parameter controls the amount of hash table associated with each entry of the file attribute cache. Thus each file attribute entry will be getting `hash_tblsize` number of pointer's to the associated directory name lookup entries (in case of directory files) or data cache entries (for regular files). In case of directory

	files, the hashing will be done based on the file names associated with that directory and in case of regular files, the hashing is done based on the block number of the data block. The default value of this field is 32 (32*4 bytes).
`blksize`	This parameter determines the default logical blocksize for data caching. This field will be overridden if read/write block size given during at NFS mount time, for that mount. The default value for this field is 8192.
`dis_cache`	This field, if non zero, will disable system-wide NFS caching. The default value for this field is zero and should remain zero.
`dis_datacache`	This field, if set to non zero, will disable NFS data caching in the system. This field should remain zero.
`acregmin`	This field specifies the minimum time in seconds that the attribute cache entries will be held with the client after the last modification, before enqueueing for updating with the NFS server. This field applies for the regular files. The default value for this field is 10.
`acregmax`	This field specifies the maximum time in seconds that the file attributes is expected to be sane with the NFS client. Beyond this period, the attribute cache is discarded and is used for refreshing. The default value for this field is 60 seconds.
`acdirmin`	This field specifies the minimum amount of time in seconds, that the

	directory file attributes are considered to be valid. The attributes will be ready for refresh beyond this duration. The default value for this field is 30.
`acdirmax`	This field specifies the maximum time in seconds that the directory file attributes is expected to be sane. After this time period, the attribute cache entry for the files in this directory, if needed, are refreshed from the NFS server. The default value for this field is 60.

Once the desired values for the tuning parameters are updated in `/sys/devices/nfs_cache_info.c`, you must then update the device library and rebuild the LynxOS kernel. These steps are described in Chapter 3, “Customizing LynxOS”.

Tuning NFS File Locking

The lock device driver facilitates *advisory file locking* over NFS. Advisory file locking will work fine only with co-operating processes. File locking is supported using `fcntl()` and `lockf()` system calls. This device driver also supports monitored class locking in which all the locks will be monitored from both client and server side. The file `/sys/devices/lock_info.c` controls the number of NFS file locking client requests that will be enqueued in the driver.

```
struct lock_info {
    int nports;/* max requests that can be enqueued */
};
```

Each entry that corresponds to the request queue takes 3068 bytes. The default value for this field is set as 10 (total memory allocated 10*3068 bytes).

Once the desired parameter value is updated in the file `/sys/devices/lockinfo.c`, you must then update the device library and rebuild the LynxOS kernel. These steps are described in Chapter 3, “Customizing LynxOS”.

Configuring the NFS Server

The only file that must be modified to allow other systems to access data on your hard disk is `/etc/exports`. This file is a database used by the NFS server to determine if the requesting host is authorized to share your data. The syntax for each entry is:

directory [**option**] [**,option**]...

The first field is the directory that is to be exported. If no options are given, any host may mount this directory and access the files for reading and writing.

Access to the directory may be given only to specific hosts. In the following example, directory `/mydata` will be exported for access by hosts `shark` and `orca`.

```
/mydata access=shark:orca
```

Permission to access an NFS-mounted directory as root must be explicitly declared. An attempt by a remote system to write to an NFS-mounted directory as root will fail, even if the directory is mounted read/write. To allow the remote system `orca` to have root access to the directory `/mydata`, the following line should be added to `/etc/exports`:

```
/mydata root=orca
```

As previously mentioned, all directories are exported as read/write unless otherwise specified. The `ro` flag is used to export a directory as read-only to everyone. The `rw` flag is used to export a directory as read/write to specific users; all other users will have read-only access. In the following example `/mydata` is exported read-only.

```
/mydata ro
```

To restrict the read/write access of /mydata to only the hosts shark and orca, but allow read-only access to everyone else, the following entry would be added to /etc/exports:

```
/mydata rw=shark:orca
```

Any of the previous examples may be grouped, giving multiple accesses:

```
/mydata access=shark:orca:fish root=shark,rw=shark:orca
```

In the above example, access of the exported directory is limited to hosts shark, orca, and fish. Only users with root access on shark will have root access to the exported directory, and only users on shark and orca will have read/write capabilities. Users on fish can access the files only in a read-only state.

CHAPTER 6

STREAMS

During your initial installation of LynxOS, you may have chosen to install STREAMS support from the menus within the `installit` utility. You can also configure, install, or remove STREAMS support at any time. This chapter describes some basics about STREAMS, and some advanced STREAMS configuration options.

Overview

LynxOS STREAMS, compatible with the STREAMS support on UNIX System V.3, provides the kernel functions, system calls, and library functions that are required to develop STREAMS drivers, modules, and applications. It also provides a STREAMS mapping driver to execute System V binaries which implement an interface between the “socket” library functions and the BSD TCP/IP protocol modules in the kernel.

LynxOS STREAMS support consists of:

- Basic STREAMS Support and TLI Library
- System V Binary Compatibility Support

The kernel software for the above components can be configured as special character device drivers.

Installing/Removing STREAMS Support

You can install and/or remove STREAMS support after initial installation of LynxOS.

To install:

```
cd /sys/lynx.os
make install.streams
```

To remove:

```
cd /sys/lynx.os
make uninstall.streams
```

For more advanced information on these steps, see Chapter 3, “Customizing LynxOS”.

STREAMS Support and TLI Library

Basic STREAMS support provides the following facilities in the kernel.

1. STREAMS system calls:

   `getmsg    putmsg    poll`

2. STREAMS `ioctl` commands.

```
I_PUSH,     I_POP,      I_LOOK,
I_FLUSH,    I_SETSIG,   I_GETSIG,
I_FIND,     I_PEEK,     I_SRDOPT,
I_GRDOPT,   I_NREAD,    I_FDINSERT,
I_STR,      I_SENDFD,   I_RECVFD,
I_LINK,      I_UNLINK
```

3. Kernel utility functions for driver and module development.

```
adjmsg,     allocb,     backq,      bufcall,
canput,     copyb,      copymsg,    datamsg,
dupb,       dupmsg,     enableok,   flushq,
freeb,      freemsg,    getq,       insq,
linkb,      msgdsize,   noenable,   OTHERQ,
pullupmsg,  putbq,      putctl,     putctl1,
putnext,    putq,       qenable,    qreply,
qsize,      RD,         rmvb,       rmvq,
strlog,     testb,      testb,      unlinkb,
WR
```

The above functions are configured in the kernel when the STREAMS character device driver is configured into the kernel.

4. Transport Layer Interface (TLI) library functions.

The `libnsl.a` library provides the following functions:

```
t_accept,     t_alloc,     t_bind,
t_close,      t_connect,   t_error,
t_free,       t_getinfo,   t_getstate,
t_listen,     t_look,      t_open,
t_optmgmt,    t_rcv,       t_rcvdis,
t_rcvconnect, t_rcvrel,    t_rcvudata,
t_rcvuderr,   t_snd,       t_snddis,
t_sndrel,     t_sndudata,  t_sync,
t_unbind
```

5. Library functions that are not currently implemented:

```
nlsgetcall    nlsprovider    nlsrequest
```

System V Binary Compatibility

In LynxOS a special character device driver (`hbtcpip`) is used to provide TCP/IP protocol-based socket-interface system calls. This software is used to provide TCP/IP protocol support for

System V binaries. A STREAMS driver called “STREAMS Mapping Driver” provides the function of interfacing the TLI commands with the BSD TCP/IP software in the kernel. This module also handles the System V socket library functions which issue corresponding TLI functions to the kernel.

The “STREAMS Mapping Driver” allows the use of TCP, UDP, and RAW IP protocols. The System V binaries using either the TLI library functions or the socket library functions can be executed with this support.

The above functions are configured into the kernel when the `stsocket` device driver is added to the kernel. This driver can be added to the kernel only if the device drivers `streams` and `hbtcpip` (which provides BSD TCP/IP support) are configured in the kernel.

STREAMS Tunable Parameters

Basic STREAMS Support

The `streams_info.h` file in the `/sys/dheaders` directory defines a structure `streams_info` to specify the tunable parameters for the Basic Streams Support. These parameters are used by the `streams.o` module in the `/sys/drivlib.a` driver library.

The actual values of these parameters are defined in the `streams_info.c` file in the `/sys/devices` directory.

These parameters are:

`NSTREAM`	Total number of stream heads in the system. When a STREAMS device is opened, one stream head is allocated for it.
`NQUEUE`	Total number of queue structures in the system. This should be a multiple

	of two. When a STREAM device is opened, a total of four queues are allocated — two for the STREAM head and two for the driver. Also, two queues are allocated whenever a module is pushed.
`NSTREVENT`	Total number of STREAM event structures in the system. These structures are used to support the `bufcall`, `poll`, and `I_SETSIG` `ioctl` calls.
`NSTRPUSH`	Maximum number of modules that may be pushed on a single stream.
`NBLK4096`	Total of 4096 byte data blocks allocated for STREAM operations.
`NBLK2048`	Total of 2048 byte data blocks allocated for STREAM operations.
`NBLK1024`	Total of 1024 byte data blocks allocated for STREAM operations.
`NBLK512`	Total of 512 byte data blocks allocated for STREAM operations.
`NBLK256`	Total of 256 byte data blocks allocated for STREAM operations.
`NBLK128`	Total of 128 byte data blocks allocated for STREAM operations.
`NBLK64`	Total of 64 byte data blocks allocated for STREAM operations.
`NBLK32`	Total of 32 byte data blocks allocated for STREAM operations.
`NBLK16`	Total of 16 byte data blocks allocated for STREAM operations.
`NBLK8`	Total of 8 byte data blocks allocated for STREAM operations.
`NBLK4`	Total of 4 byte data blocks allocated for STREAM operations.

NMUXLINK	Maximum number of STREAMS in a system that can be linked as lower STREAMS to multiplexor drivers.
NSEPGCNT	Not used.
STRMSGSZ	Maximum number of bytes of information that can be passed in a single system call to a STREAM head to be placed in the data part of a message (in M_DATA blocks). If a larger size is specified in a write system call, it will be sent as multiple messages. If larger size is specified in a putmsg system call, it will fail with ERANGE error.
STRCTLSZ	Maximum number of bytes of information that can be passed in a single system call to a STREAM head to be placed in the control part of a message (in an M_PROTO or M_PCPROTO block). If a larger size is specified in a putmsg system call, it will fail with ERANGE error.
STRLOFRAC	The percentage of data blocks of a given class at which low priority block allocation requests are automatically failed. This value must be an integer in the range 0 < STRLOFRAC <= STRMEDFRAC < 100.
STRMEDFRAC	The percentage of data blocks of a given class at which medium priority block allocation requests are automatically failed.
NSTRSEM	Not used.
MAX_SPIPE	Total number of times the STREAM pipe device /dev/spx may be open at the same time in the system. To create a single STREAM pipe, the

/dev/spx device should be opened twice.

STREAMS Mapping Driver

The stsocket_info.h file in the /sys/dheaders directory defines a structure (stsocket_info) to specify the tunable parameters for the STREAMS Mapping Driver. These parameters are used by the modules in the STREAMS library, /sys/lib/streams.a.

The actual values of these parameters are defined in the stsocket_info.c file in the /sys/devices directory. These parameters are:

SMD_NTCP	Total number of times the /dev/tcp device may be open at the same time.
SMD_NUDP	Total number of times the /dev/udp device may be open at the same time.
SMD_NIP	Total number of times the /dev/ip device may be open at the same time.

Kernel Configuration

Edit the basic STREAMS support parameters in streams_info.c and the mapping driver parameter values in stsocket_info.c. You must then update the device library and rebuild the LynxOS kernel with the new values. These steps are described in the section "Rebuilding the Kernel with a Modified Device Info File" of the *Cross-Development Guide*. Substitute all.streams for *make_target* and streams_info.o stsocket_info.o for *dev_info_object*.

CHAPTER 7

SKDB–Simple Kernel Debugger

The Simple Kernel Debugger (SKDB) is a machine-level, symbolic debugger. In order to use the debugger, you must install the SKDB module into the kernel.

Overview

Installing/Removing SKDB Support

You can install and/or remove SKDB support after initial installation of LynxOS.

To install:

```
cd /sys/lynx.os
make install.streams
```

To remove:

```
cd /sys/lynx.os
make uninstall.streams
```

For more advanced information on these steps, see Chapter 3, “Customizing LynxOS”.

Automatically Starting After a Kernel Crash

In order to have SKDB set up to automatically begin after a kernel crash, you must at least once manually invoke SKDB and escape out. In addition, if the LynxOS kernel hits a breakpoint set by SKDB, the debugger will be entered automatically. SKDB will respond with an `*` (asterisk) prompt character on the console. The normal LynxOS operating environment will be frozen while in this mode. To return to normal operation, press the **Esc** key.

Design Notes

The Lynx SKDB has the following characteristics by design:

- The *default* SKDB port specifies the port to which SKDB reports a crash by default. Other SKDB-ready ports still accept their respective hot keys to invoke unless the entire SKDB is disabled (uninstalled).
- To break into SKDB with a hot key from a port, the port must be explicitly opened by a process.

Changes to SKDB

The SKDB now supports kernel mode in the TotalView Debugger, involving some changes to the user interface of SKDB. The items below are available to all platforms.

Using Hot Keys

Once installed, you can invoke the debugger by pressing **Shift-Ctrl-Minus** on the keyboard. Some keyboards also may support using **Ctrl-Minus**.

“Break” Key

When using the the **z** commands within SKDB, it will prompt for a new key combination. Simply hit the desired key combination. SKDB will prompt for the same key combination for confirmation. Hit the same key combination, or something else to cancel the change.

Kernel Status Display

At each invokation, SKBD prints a line like the following:

```
OK:PID.TID@trapcode,slevel,econtext,PSW,PC,checksum
```

`PID`	Current process ID
`TID`	Current thread ID
`trapcode`	Trap code; trap code is the same as the architecture’s exception code with the addition of -1 for invokation from keyboard and -2 for a panic situation.
`slevel`	Last slevel; slevel is the kernel preemption level (0: user, 1: kerenl, 2: no context switching, 3: no interrupts).
`econtext`	Econtext address; econtext is the per-thread register stack into which the kernel saves registers.
`PSW`	Processor status word (PSW); called flags or status register in some architectures.
`PC`	Last program counter
`checksum`	String checksum

For example:

```
OK:0.0@-1,1,DB0AB19C,000199A0,00000207 C961
```

Kernel Status Re-display

Hitting **Ctrl-B** then **?** and **Return** re-displays the above information. This option is currently available on serial terminals only, not on video consoles.

Singlestepping

Hitting **Ctrl-B** then **s** and **Return** singlesteps the current thread. This option is currently available on serial terminals only, not on video consoles.

Setting Breakpoints

Up to 10 breakpoints can be set with SKDB. When the CPU tries to execute the instruction at the breakpoint, it traps to the debugger. From the debugger, the registers or memory may be examined. The breakpoints remain set until explicitly unset. While at a breakpoint, pressing the **Esc** key continues from that breakpoint.

SKDB Commands

SKDB commands for each of the platforms are described on the following pages in table format.

Table 7-1: SKDB Commands on the x86 and PowerPC

Command Format	Example	Description
Examine Memory		
addr	`* 0xdb100000`	Examines (displays) the 4 bytes at the location of *addr*
$*sym*	`* $currpid`	Examines the 4 bytes at the location of *sym*
addr size	`* 0xdb100000 40`	Examines *size* bytes of memory starting at *addr*
$*sym size*	`* $main 40`	Examines *size* bytes of memory starting at the address of *sym*
T *pid addr*	`* T 0xdb100000 9`	Translate virtual address *addr* to physical using *pid* mapping
+	`* +`	Examines the next 32 bytes of memory
–	`* –`	Examines the last 32 bytes of memory
+ *size*	`* + 64`	Examines the next *size* bytes of memory
– *size*	`* – 64`	Examines the last *size* bytes of memory
Change Memory		
c *addr data*	`* c 0xdb100000 0x200`	Stores *data* as a long word (4 bytes) at *addr*
c $*sym data*	`* c $job 3`	Stores *data* into the variable *sym* (assumes 4 bytes)
Find Symbol		
&*sym*	`* &proctab`	Display address of *sym*

Table 7-1: SKDB Commands on the x86 and PowerPC (Continued)

Command Format	Example	Description
`f` *addr*	`* f 0xdb100000`	Display closest symbol with offset to the *addr*
Display Data Structure		
`s st`	`* s st`	Display contents of st_entry structure for the current thread
`s st` *tid*	`* s st 5`	Display contents of st_entry structure for thread ID *tid*
`s st` *addr*[1]	`* s st 0xdb100000`	Display contents of st_entry structure at address *addr*
`s proc`	`* s proc`	Display contents of pentry structure for the current process
`s proc` *pid*	`* s proc 25`	Display contents of pentry structure for process ID *pid*
`s proc` *addr*	`* s proc 0xdb100000`	Display contents of pentry structure at address *addr*
`s pss`	`* s pss`	Display contents of pssentry structure for the current process
`s pss` *pid*	`* s pss 40`	Display contents of pssentry structure for process ID *pid*
`s pss` *addr*	`* s pss 0xdb100000`	Display contents of pssentry structure at address *addr*
`s inode` *num*	`* s inode 45`	Display contents of inode_entry structure at index *num*
`s inode` *addr*	`* s inode 0xdb100000`	Display contents of inode_entry structure at address *addr*
`s block` *num*	`* s block 22`	Display contents of buf_entry structure at index *num*
`s block` *addr*	`* s block 0xdb100000`	Display contents of buf_entry structure at address *addr*
`s ihead` *num*	`* s ihead 30`	Display contents of ihead_entry structure at index *num*

Table 7-1: SKDB Commands on the x86 and PowerPC (Continued)

Command Format	Example	Description
`s ihead` *addr*	`* `**`s ihead 0xdb100000`**	Display contents of ihead_entry structure at address *addr*
`s file` *num*	`* `**`s file 12`**	Display contents of file structure at index *num*
`s file` *addr*	`* `**`s file 0xdb100000`**	Display contents of file structure at address *addr*
`s fifo` *num*	`* `**`s fifo 49`**	Display contents of fifo structure at index *num*
`s fifo` *addr*	`* `**`s fifo 0xdb100000`**	Display contents of fifo structure at address *addr*
`s cdev` *num*	`* `**`s cdev 3`**	Display contents of cdevsw_entry structure at index *num*
`s cdev` *addr*	`* `**`s cdev 0xdb100000`**	Display contents of cdevsw_entry structure at address *addr*
`s bdev` *num*	`* `**`s bdev 0`**	Display contents of bdevsw_entry structure at index *num*
`s bdev` *addr*	`* `**`s bdev 0xdb100000`**	Display contents of bdevsw_entry structure at address *addr*
`s fdentry` *addr*	`* `**`s fdentry 0xdb100000`**	Display contents of fdentry structure at address *addr*
`s +`	`* `**`s +`**	Display contents of the next (in memory) data structure (of the type being displayed)
`s -`	`* `**`s -`**	Display contents of the last (in memory) data structure (of the type being displayed)
`s next`	`* `**`s next`**	Display contents of the data structure pointed to by the `next` (or equivalent) field of the currently-displayed data structure
`s prev`	`* `**`s prev`**	Display contents of the data structure pointed to by the `prev` (or equivalent) field of the currently-displayed data structure

Table 7-1: SKDB Commands on the x86 and PowerPC (Continued)

Command Format	Example	Description
	Help	
h	* **h**	Display a description of all available commands
?	* **?**	Same as **h**
	Stack Trace	
t	* **t**	Symbolic stack trace of current process or thread
t *pid*	* **t 1**	Symbolic stack trace of process *pid*
t −*tid*	* **t −5**	Symbolic stack trace of thread *tid*
v	* **v**	Toggle verbose mode for trace
	Display Registers, Processes & Set Priority	
r	* **r**	Display processor registers
p *count*	* **p 20**	Display process table (all or *count* lines worth)
P *prio tid*	* **P 15 8**	Change priority of thread *tid* to *prio* divided by 2
	Breakpoints	
b	* **b**	Show all breakpoints set
b *num addr*	* **b 1 0xdb100000**	Put breakpoint *num* at the memory location *addr*
b *num* $*sym* [*off*]	* **b 2 $getpid**	Put breakpoint *num* at the address of the routine *sym* or the address of *sym* + *off*
u *num*	* **u 5**	Unset breakpoint *num*
	PowerPC Specific	
S	* **S**	See segment register.
d *addr* d $*sym*	**d $.resched**	Disassemble at *addr*.

1. The specified address (*addr*) must be inside the table of items. For example, if the address 0xdb100000 is not inside the `PROC` array, the command “s proc 0xdb100000” will not display the requested information.

Table 7-2: SKDB Commands on the 68k

Command Format	Example	Description
	Examine Memory	
addr	`* 0xea040000`	Examines (displays) the 4 bytes at the location of *addr*
`$`*sym*	`* $_currpid`	Examines the 4 bytes at the location of *sym*
addr size	`* 0xea040000 16`	Examines *size* bytes of memory starting at *addr*
`$`*sym size*	`* $_main 40`	Examines *size* bytes of memory starting at the address of *sym*
`T` *pid addr*	`* T 1 0xea040000`	Translate virtual address *addr* to physical using *pid* mapping
+	`* +`	Examines the next 32 bytes of memory
–	`* –`	Examines the last 32 bytes of memory
+ *size*	`* + 64`	Examines the next *size* bytes of memory
– *size*	`* – 64`	Examines the last *size* bytes of memory
	Change Memory	
`c` *addr data*	`* c 0xea040000 0x20`	Stores *data* as a long word (4 bytes) at *addr*
`c $`*sym data*	`* c $_main 3`	Stores *data* into the variable *sym* (assumes 4 bytes)
	Find Symbol	
`&`*sym*	`* &_proctab`	Display address of *sym*

Table 7-2: SKDB Commands on the 68k (Continued)

Command Format	Example	Description
`f` *addr*	`*` **`f 0xea040000`**	Display closest symbol with offset to the *addr*
Display Data Structure		
`s st`	`*` **`s st`**	Display contents of st_entry structure for the current thread
`s st` *tid*	`*` **`s st 5`**	Display contents of st_entry structure for thread ID *tid*
`s st` *addr*	`*` **`s st 0xea040000`**	Display contents of st_entry structure at address *addr*
`s proc`	`*` **`s proc`**	Display contents of pentry structure for the current process
`s proc` *pid*	`*` **`s proc 25`**	Display contents of pentry structure for process ID *pid*
`s proc` *addr*	`*` **`s proc 0xea040000`**	Display contents of pentry structure at address *addr*
`s pss`	`*` **`s pss`**	Display contents of pssentry structure for the current process
`s pss` *pid*	`*` **`s pss 40`**	Display contents of pssentry structure for process ID *pid*
`s pss` *addr*	`*` **`s pss 0xea040000`**	Display contents of pssentry structure at address *addr*
`s inode` *num*	`*` **`s inode 45`**	Display contents of inode_entry structure at index *num*
`s inode` *addr*	`*` **`s inode 0xea040000`**	Display contents of inode_entry structure at address *addr*
`s block` *num*	`*` **`s block 22`**	Display contents of buf_entry structure at index *num*
`s block` *addr*	`*` **`s block 0xea040000`**	Display contents of buf_entry structure at address *addr*
`s ihead` *num*	`*` **`s ihead 30`**	Display contents of ihead_entry structure at index *num*

Table 7-2: SKDB Commands on the 68k (Continued)

Command Format	Example	Description
`s ihead` *addr*	`* s ihead 0xea040000`	Display contents of ihead_entry structure at address *addr*
`s file` *num*	`* s file 12`	Display contents of file structure at index *num*
`s file` *addr*	`* s file 0xea040000`	Display contents of file structure at address *addr*
`s fifo` *num*	`* s fifo 49`	Display contents of fifo structure at index *num*
`s fifo` *addr*	`* s fifo 0xea040000`	Display contents of fifo structure at address *addr*
`s cdev` *num*	`* s cdev 3`	Display contents of cdevsw_entry structure at index *num*
`s cdev` *addr*	`* s cdev 0xea040000`	Display contents of cdevsw_entry structure at address *addr*
`s bdev` *num*	`* s bdev 0`	Display contents of bdevsw_entry structure at index *num*
`s bdev` *addr*	`* s bdev 0xea040000`	Display contents of bdevsw_entry structure at address *addr*
`s fdentry` *addr*	`* s fdentry 0xea040000`	Display contents of fdentry structure at address *addr*
`s +`	`* s +`	Display contents of the next (in memory) data structure (of the type being displayed)
`s -`	`* s -`	Display contents of the last (in memory) data structure (of the type being displayed)
`s next`	`* s next`	Display contents of the data structure pointed to by the “next” (or equivalent) field of the currently-displayed data structure
`s prev`	`* s prev`	Display contents of the data structure pointed to by the “prev” (or equivalent) field of the currently-displayed data structure

Table 7-2: SKDB Commands on the 68k (Continued)

Command Format	Example	Description
Help		
h	* **h**	Display a description of all available commands
?	* **?**	Same as "h"
Stack Trace		
t	* **t**	Symbolic stack trace of current process or thread
t *pid*	* **t 1**	Symbolic stack trace of process *pid*
t –*tid*	* **t –9**	Symbolic stack trace of thread *tid*
v	* **v**	Toggle verbose mode for trace
Display Registers, Processes & Set Priority		
r	* **r**	Display processor registers
p [count]	* **p 20**	Display process table (all or *count* lines worth)
P *prio tid*	* **P 19 9**	Sets priority of thread *tid* to prio divided by 2
Breakpoints		
b	* **b**	Show all breakpoints set
b *num addr*	* **b 1 0xea040000**	Put breakpoint *num* at the memory location *addr*
b *num* $*sym* [off]	* **b 2 $_getpid**	Put breakpoint *num* at the address of the routine *sym* or the address of *sym* **+** *off*
u *num*	* **u 1**	Unset breakpoint *num*

Table 7-3: SKDB Commands on the SPARC

Command Format	Example	Description
	Examine Memory	
addr	* **0xe4004000**	Examines (displays) the 4 bytes at the location of *addr*
$*sym*	* **$_currpid**	Examines the 4 bytes at the location of *sym*
a *asi addr*	* **a 0x20 0x0**	Examines the 4 bytes at *addr* in the address space *asi*
ah *asi addr*	* **ah 0x20 0x0**	Examines the 2 bytes at *addr* in the address space *asi*
ab *asi addr*	* **ab 0x20 0x0**	Examines the 1 byte at *addr* in the address space *asi*
addr size	* **0xe4004000 40**	Examines *size* bytes of memory starting at *addr*
$*sym size*	* **$_main 40**	Examines *size* bytes of memory starting at the address of *sym*
T *addr* [*context*]	* **T 0xe4004000 1**	Examines the MMU mapping tree for virtual address *addr* in *context* (default=0)
+	* **+**	Examines the next 32 bytes of memory
–	* **–**	Examines the last 32 bytes of memory

Table 7-3: SKDB Commands on the SPARC (Continued)

Command Format	Example	Description
	Examine Memory	
c *addr data*	* **c 0x0000 0x200**	Stores *data* as a long word (4 bytes) at *addr*
c $*sym data*	* **c twake_max 1**	Stores *data* into the variable *sym* (assumes 4 bytes)
a *asi addr data*	* **a 0x20 0x0 0x12345678**	Stores *data* (4 bytes) at *addr* in the address space *asi*
ah *asi addr data*	* **ah 0x20 0x0 0x1234**	Stores *data* (2 bytes) at *addr* in the address space *asi*
ab *asi addr data*	* **ab 0x20 0x0 0x12**	Stores *data* (1 byte) at *addr* in the address space *asi*
	Find Symbol	
& *sym*	* **&_proctab**	Display address of *sym*
f *addr*	* **f 0xe4004000**	Display closest symbol with offset to the *addr*
	Stack Trace	
t	* **t**	Symbolic stack trace of current process to thread
t *pid* [*count*]	* **t 1**	Symbolic stack trace of process *pid*
t –*tid* [*count*]	* ***t –tid***	Symbolic stack trace of thread *tid*
v	* **v**	Toggle verbose mode for trace

Table 7-3: SKDB Commands on the SPARC (Continued)

Command Format	Example	Description
Display Registers, Processes, & Reset CPU		
r [*level*]	* **r 1**	Display processor registers *level* windows back (default is current window)
P	* **P**	Display processor registers at user trap
k *context_num*	* **k 1**	Change context
K	* **K**	Display context
p [*count*]	* **p 20**	Display process table (all or count lines worth)
R	* **R**	Reset CPU. Resets the CPU by jumping into the SPARC PROM.
Breakpoints		
b	* **b**	Show all breakpoints set
b # *addr*	* **b 1 0xe4004000**	Put breakpoint number # at the memory location *addr*
b # $*sym* [*offset*]	* **b 2 $_getpid**	Put breakpoint # at the address of the routine *sym* or the address of *sym* + an *offset*
u #	* **u 5**	Unset breakpoint #
Argument Stack		
V *command*	* **V $_currpid**	Push command result on argument stack
^	* **^**	Pop argument stack
.	* **.**	Show top of argument stack
s [*count*]	* **s 5**	Show *count* lines of argument stack

Platform-Specific Notes

x86 Notes

1. The debugger uses the symbol table loaded as part of LynxOS.

   If you strip the kernel, the symbolic lookup will not work.

2. When the verbose mode is toggled, the stack trace prints the contents of the stack between subsequent values of the frame pointer, EBP.

   Using the verbose mode, you can decipher the contents of the stack, including local variables and parameters.

   As an example, consider the following. In SKDB, one traces a task using the **t** command:

   ```
   * t -16
   pc = swait+0x1b0
   pc = wd3e_stask+0x2e
   pc = stret+0x0
   ```

   When verbose mode is toggled, the stack trace appears as:

   ```
   * t -16

   pc = swait+0x1b0

       0xdb18c930 0xdb02fc28 0x00000240
       0xdb0da170 0x00000001

   pc = wd3e_stask+0x2e

       0xdb18c954 0xdb0701be 0xdb0da1e0
       0x00000000 0x00000000 0xdb0b9878
       0x00000000 0xdb0da170 0xdb0da230

   pc = stret+0x0
   ```

   The stack contents below **`pc = wd3e_stask+0x2e`** pertain to the call to `swait`. The first value, `0xdb18c954`, is the previous frame pointer for the frame being used by `swait`. The second value is the return PC for the `swait` frame (`0xdb0701be` is `wd3e_stask+0x2e`). After those values, the arguments to `swait` are displayed: `0xdb0da1e0`, `0x00000000`.

3. If SKDB is installed on both the console and serial port and the LynxOS kernel crashes or hits a breakpoint set by SKDB, the debugger will be entered at whichever interface was used last to enter SKDB (the console or the serial port).

Consider the following sequence of events:

- SKDB is entered at the console via **F8**
- a breakpoint is set
- SKDB is exited via **Esc**
- SKDB is entered at the serial port via **Ctrl–T**
- SKDB is exited

At this point, if the kernel hits the breakpoint, the debugger will be entered at the serial port, even though the breakpoint was set up using the console, because the serial port is where SKDB was entered last.

68k Notes

1. SKDB does not validate the arguments to the commands.

Be careful to provide valid arguments. Invalid arguments may cause the system to crash!

2. Setting breakpoints at branch instructions or in their delay slots is not supported and really nasty consequences will result.

3. Clearing the current breakpoint causes the kernel to hang. This problem manifests itself when you try to clear the breakpoint just taken.

Suggested work around: set another breakpoint at some other address. Then when that new breakpoint is taken, clear the original.

4. Note that the debugger uses the symbol table loaded as part of LynxOS.

   If you strip the kernel, the symbolic lookup will not work.

SPARC Notes

1. Setting breakpoints at branch instructions or in their delay slots is not supported.
2. Note that the debugger uses the symbol table loaded as part of LynxOS. If you strip the kernel, the symbolic lookup will not work.

CHAPTER 8

Creating Kernel Images for Embedded Applications

LynxOS supports embedded development by providing utilities that combine the LynxOS kernel and applications into bootable images. These bootable images can be burned into a PROM, put on media (as is done with the LynxOS boot media), booted over the network, or maintained on disk. This capability allows embedded developers to create "application ready" embedded systems that include the LynxOS kernel and their own custom applications.

The LynxOS utility that provides this capability is called `mkimage`. This chapter describes the mkimage utility and provides examples of its use.

Overview

A LynxOS kernel *image* is a special file created for booting that contains LynxOS and a RAM-based file system (called a RAM disk) with minimal files for operating system functions and applications. When a machine is booted with the image, the LynxOS kernel recognizes and configures the RAM file system through the `ramdisk` driver. Depending on the set of utilities that are put into the image, the machine can function as a minimal application target or as a complete LynxOS development system.

LynxOS Image Utility—mkimage

`mkimage` is a new utility that replaces the `mkromk` utility from previous LynxOS releases.

`mkimage` creates a single file, the bootable image, with the following components:

- the LynxOS kernel
- the memory image of a RAM disk with a specified initial set of files
- (optionally) the `TEXT` segments of applications loaded at initialization time[†]

`mkimage` creates the image with attributes that are set in a *specification file*. By defining the fields in the specification file, you can create customized images for your applications.

The syntax for `mkimage` is:

`mkimage [ -v? ]` *specfile outfile*

The implementation of `mkimage` allows you to make kernel images in any directory with any number of specified files. You do not need root privileges to use `mkimage`. In addition, `mkimage` can be executed natively on a LynxOS system or from a LynxOS cross development host system, such as SunOS.

See the manual pages `mkimage(1)` and `mkimage.spec(5)` for more information about command syntax.

†. Resident text is not supported on SPARC systems in this release.

LynxOS Kernel

You can specify that mkimage use the current LynxOS kernel in /lynx.os when building the image. However, you will most likely create a custom kernel for use in your image. For example, you may choose to remove any unused device drivers from the kernel in order to reduce the size of kernel, and therefore, the final image. See Chapter 3, “Customizing LynxOS” for more information on creating custom kernels.

To save space, mkimage does not expand the BSS section in the image. The BSS section is expanded and initialized to zeros by kernel code.

RAM-based File System

mkimage creates a RAM-based file system containing a basic set of initial files. LynxOS accesses this file system via the ramdisk driver.

Resident TEXT Segments

Optionally, each executable can start its TEXT segment on a page boundary. In this case, the RAM disk file may contain only the DATA segments and is created with a special magic number of 0410 (IMAGIC) to identify it. This saves RAM space in the executing system.

When deciding whether or not to make resident TEXT segments in the image, consider the speed of the hardware where the image will reside relative to the speed of RAM. If the target hardware will be a PROM (generally slower than RAM), you should not make resident TEXT segments in your image. In this case, the file will be loaded from the ROM disk into RAM for faster execution.

If the target hardware is flash RAM that is roughly the same speed as RAM, you should consider making an image with resident TEXT segments in order to conserve RAM.

How Does it Work?

The following sections provide more details on how the created image works in two common applications—remote booting and booting from ROM.

Images Booted Over a Network

To network boot LynxOS (also called *netboot*), the image file is stored on a remote server and copied over the network into the RAM of a remote node by the firmware-resident `tftp` boot code. Generally, the node is diskless.

Since everything is in RAM, the boot code gives control to the loaded OS, which in turn configures the attached RAM disk as its root file system. In this case, since all the executables are in RAM, it is useful if they are executed directly without copying them again into RAM. Thus, the `TEXT` segments of the executables are loaded as part of system initialization at boot time, rather than on demand from the root file system.

In the remote booting configuration, the RAM memory map is sequential as shown below:

Table 8-1: Example RAM Memory Map

INT Vectors
OS TEXT Segment
OS DATA Segment
OS BSS Segment
OS Symbols
RAM Disk File System
Application TEXT Segments

Images Booted from ROM

The firmware-resident ROM boot code always passes control to the ROM-based kernels. At this point, depending on the options defined in the specification file, all or portions of the image will be moved to RAM. LynxOS always moves the `DATA` and `BSS` sections to RAM.

LynxOS sets up the memory map such that some portions are mapped into ROM and other portions are mapped into RAM. This is required in order to fully utilize the ROM contents, without copying them into RAM. Figure 8-1 shows an example mapping scheme in which the LynxOS kernel text, root file system, and resident `TEXT` segments are all in ROM.

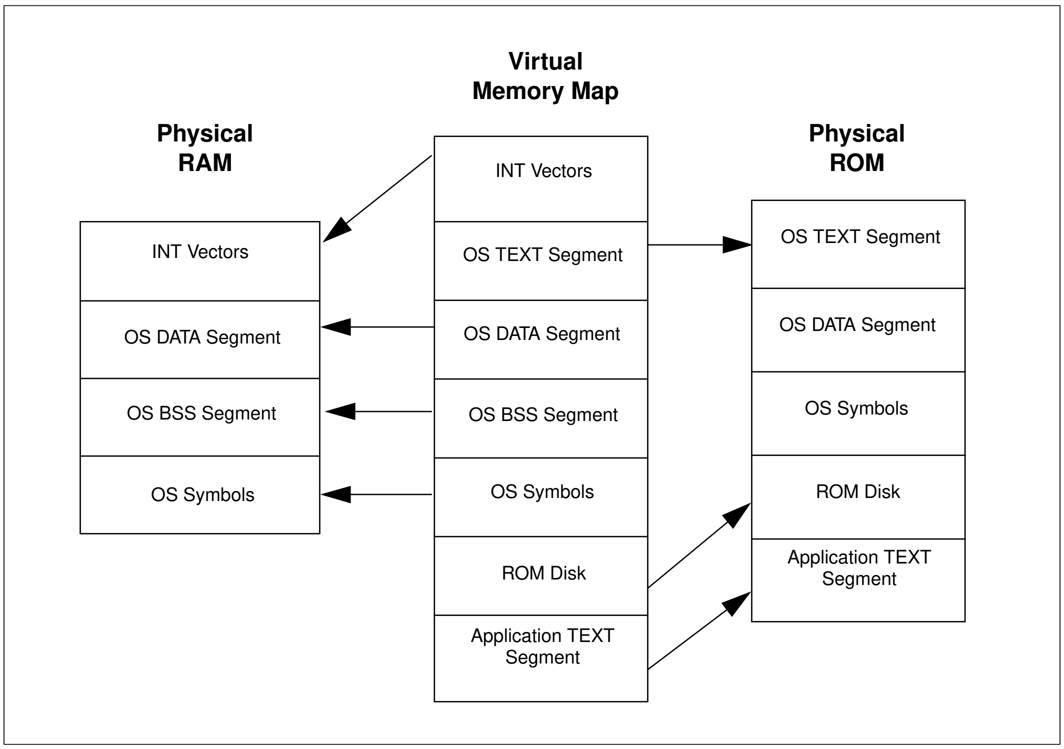

Figure 8-1: Example Memory Map Configuration

As shown in Figure 8-1, the first pages, containing the interrupt and exception vectors, are mapped into RAM. The next few pages are mapped into ROM for the OS `TEXT` section. The following few pages are mapped into RAM for the OS `DATA` section (copied from ROM), for the OS `BSS` section (initialized in RAM) and for OS symbols (copied from ROM). The root file system and resident `TEXT` segments are mapped to ROM.

If the root file system is read-only and the application needs a writable file system, the image must contain a script to create a writable RAM disk. The script should invoke the `mkramdisk` utility to create the writable RAM disk after the system is up and running.

Basic Procedure

The same basic steps are used to create images for the range of applications, from network booting to ROM booting.

- Configuring a LynxOS kernel with desired functionality and the `ramdisk` driver;
- Creating a specification file for `mkimage` that defines the LynxOS kernel to use and the applications to include;
- Running `mkimage`;
- Testing the image;
- Putting the image into the target environment.

These steps are shown in the example “Making Installation Boot Media” on page 125. However, if you plan to network boot the image, please read Chapter 10, “Network Booting Diskless Clients with LynxOS”, for more information about this specific application.

Building an Image

The basic steps for making a bootable image are:

1. Create your own copy of `/sys/lynx.os` and make file modifications and the image in this directory.

   ```
   cp -r /sys/lynx.os /tmp/test.os
   cd /tmp/test.os
   ```

2. Enable the `ramdisk` driver in `CONFIG.TBL`.
3. Make any other modifications to `uparam.h`, Makefile, etc.
4. Create a new kernel with `make a.out`.
5. Create a specification file with desired attributes.
6. Run `mkimage`.

If you plan to make a remotely bootable kernel, see Chapter 10, "Network Booting Diskless Clients with LynxOS" for a description of additional sample scripts you can use.

NOTE: *Remember that when developing from a LynxOS cross development host, you need to use* `$ENV_PREFIX/`*pathname* *where absolute paths are shown in this chapter.*

Enabling the RAM Disk Device Driver

To enable the `ramdisk` driver, you must uncomment (by removing the `#` sign) the line `I:ramdisk.cfg` in the file `CONFIG.TBL`.

Modifying Kernel Parameters

To reduce the size of the kernel, all drivers not required should be removed. See "Example—Removing Unused Device Drivers" on page 54 for an example of this procedure.

Creating a Specification File

The easiest way to create a specification file is by copying one of the example files that are in `/sys/romkit/installation` or by using the example in `mkimage.spec(5)` manual page as a template.

CAUTION! *Because of a limitation in the SPARC processor, text cannot be executed directly from ROM if any DMA activity can occur. There is a hardware race condition that will cause the CPU to fetch an instruction from an invalid cache location. For this reason,* `mkimage` *does not allow any portion of the SPARC running system to reside in ROM.*

The `mkimage` specification file is described in detail in the manual page `mkimage.spec(5)`. Some important attributes are:

`ostext=[ram|rom]`	The execution environment of the kernel.
`rom`	The base address of the ROM space where the image will be located. It is used to generate a static and runtime load map for the image.
`kernel=`*path*	The path of the LynxOS kernel to be used in the image. This could be `/lynx.os` but most likely will be a customized kernel that you have built elsewhere on your system.
`nodetab=`*path*	The device node table corresponding to the kernel.
`directory`	A directory on the target file system.
`file`	A file on the target file system.

Testing Kernel Images

You can use the LynxOS utility `preboot` to directly test the kernel images that you make with `mkimage`. (See the `preboot(1)` manual page for the specific commands used on your platform.) For example, to test the kernel image `/tmp/test/images/image1` created on a LynxOS x86 machine, do the following:

1. Reboot the machine.

   `reboot -a`

2. At the `preboot` prompt, enter

   `Command?` **`b /tmp/test/images/image1`**

Making Installation Boot Media

This example illustrates the basic steps you need to create an embedded application with LynxOS. The “embedded” targets in this example are bootable floppies and tapes for each LynxOS architecture type. The example “application” is the LynxOS installation utility, `installit`.

The scripts described in this example are used to make the boot media for the LynxOS distribution. Use this example and the concepts described within as a template for your application.

Overview of the Build Environment

For this example, we’ve already created a build environment, `/sys/romkit/installation`, that contains our application, image specification files for each platform, and scripts that capture the steps needed to build the bootable media.

The files in `/sys/romkit/installation` are:

`Build`	Script that creates the installation media
`Makefile`	File that describes the build rules for each target
`install-secondary`	Application file
`installit`	Application file
`installit.help`	Application file
`profile.install`	Application file
`profile.secondary`	Application file
`rc.install`	Start-up file
`030_M147.spec`	`mkimage` specification file for MVME147.
`040_MVME.spec`	`mkimage` specification file for MVME162/167/177.
`ppc.spec`	`mkimage` specification file for PowerPC systems
`usparc.spec`	`mkimage` specification file for SPARC.
`x86.spec`	`mkimage` specification file for x86.

The easiest way to build bootable media is to use the script `/sys/romkit/installation/Build`. This script has been pre-configured to create a small LynxOS kernel (removing TCP/IP, NFS, and STREAMS), make a bootable image (using `mkimage`), and copy the image to the boot media (platform-dependent).

Before You Begin

You must be on a native LynxOS machine and be superuser (`root`) to run the scripts.

```
user name: root
Password:
```

x86 Specifics

If you are on a LynxOS x86 system, you need two 1.44 MB formatted floppy disks. (See the `fmtflop(1)` manual page for more information about formatting floppies.)

Also, the script assumes that there are no file systems mounted on the directory `/mnt`. Be sure that `/mnt` is available for use before using the script. (See the `mount(1)` manual page for more information about mounted file systems.)

```
umount /mnt
```

PowerPC, 68k, and SPARC

If you are on a PowerPC, 68k, or SPARC system, you need one 15 MB quarter inch tape.

Running the Scripts

For this example, we recommend that you duplicate `/sys/romkit/installation` into a temporary location so that you can freely modify files without losing the defaults.

```
cp -r /sys/romkit/installation /tmp/myinstall
cd /tmp/myinstall
./Build
```

follow the instructions printed on the screen

ROMing Issues

The tools and steps needed to put an image created by `mkimage` into a PROM are specific to the target hardware. Depending on the PROM burner and hardware target that you use, you may be able to put the image file as is (with no conversion to an intermediary file format) into a PROM. Refer to your hardware documentation for more information.

On x86 and 68k systems, LynxOS provides a utility called `torom`. This utility converts an image into Intel hexadecimal format files for burning into a PROM with a PROM burner that accepts this format. This section describes how to use the `torom` utility on x86 and 68k systems.

Overview of torom

The `torom` utility converts the given image into multiple Intel hexadecimal format files. This format is required by common PROM burning hardware. It only can generate image files for optional slots on the MVME 68k boards. The syntax of `torom` is platform-dependent. On x86 systems, it is:

`torom` *image_file eprom_size*`[k] [main/opt]`

On 68k systems, it is:

`torom` *image_file eprom_size*`[k] [0|8|16]`
`[main|debug|opt] [flat] [raw|bin|intel] [swap]`

The formatted files for the 68k and the x86 can be burnt into ROMs. We provide the sources for the `torom` utility in `$ENV_PREFIX/src/bin`. You can modify this file to suit your ROM environment.

Options to torom

For 68k systems, *image_file* is split into low/high byte pair files of size *eprom_size* and converted into Intel hexadecimal format files. Also on 68k platforms, the Motorola VMEbug specified signature will be added to the beginning of the first pair of image files.

For x86 systems, the option **`main`** causes `torom` to strip the `struct exec` header of an `a.out` format kernel image file (created by `mkimage`). This option is used on the "jump code" *image_file* (`rkjump`) to complete the necessary BIOS signature. (See "Building the Jump Code" on page 129 for more information.) The **`opt`** option results in regular conversion of *image_file* to PROM-ready format.

Generating the PROM Images on x86 Systems

Building the Jump Code

BIOS limitations on the x86 architecture require that a small object module, `rkjump`, be used to bootstrap the loading of a LynxOS kernel from ROM. It must be available within the address range 640 KB to 1 MB at system power up. On most BIOS types, `rkjump` must reside in the address range `0xC0000` to `0xDFD00` on a `0x200` byte boundary.

After taking control from the system BIOS, the `rkjump` module initializes the Segment Descriptor Registers, gets the CPU to protected mode, and transfers control to the ROM segment containing the LynxOS image.

`rkjump` is very specific to the memory architecture and must be tuned to the specific target needs. The sources and build environment for rkjump are in `/src/bin/rkjump386`. Before building the `rkjump` module, the kernel image ROM address needs to be defined in the `rkconfig.h` file in this directory.

For example, if you have created a kernel image with a load address of `0x00100000`, you must specify this address in `/src/bin/rkjump/rkconfig.h`. You then need to rebuild `rkjump`.

```
cd /src/bin/rkjump386
vi rkconfig.h
```

Change the value of RKJUMP_ADDRESS

```
#define RKJUMP_ADDRESS  0x00100000
```

```
touch rkjump.c
make all
```

Converting the Jump Code

After building the specific `rkjump` for your image, use `torom` to convert it into hex format for burning into a PROM.

```
torom rkjump 32k main
```

This will generate file `rkjump00` which should be transferred to a PROM.

Converting the Kernel Image

Now convert the kernel image that you created with `mkimage` into hex format using the `torom` utility.

`torom romable.os` *size opt*

Where *size* is the appropriate size of each PROM (32k, 64k, 128k, etc.). Please refer to `torom(1)` for more information.

Generating the PROM Images on 68k Systems

Run ENV from the VMEbug prompt and specify the following settings:

```
xxxx-Bug> env
Local SCSI Bus Reset on Debugger Startup [Y/N] = Y
ROM Boot Enable [Y/N] = Y
ROM Boot at Power-up only [Y/N] = N
ROM Boot Enable search of VME Bus [Y/N] = N
ROM Boot Abort Delay = 3
ROM Boot Direct starting Address = FFA00000
ROM Boot Direct Ending Address = FFBFFFFC
```

All of these options work in the MVME162/167 Debugger/Diagnostics Release Version 1.4 and 1.5. Make note of the version from the start-up message on reset/power-on.

Finally, generate the PROM images for burning in using the `torom` utility. For example,

`torom romable.os` *size opt*

Where *size* is the appropriate size of each PROM (32k, 64k, 128k, etc.). Please refer to `torom(1)` for more information.

CHAPTER 9

Snugly Coupled Multi-Processing

Snugly Coupled Multi-Processing (SCMP) allows multiple MVME CPU boards (MVME162/167/177 and MVME1600) connected to the same VME backplane to communicate with each other using the socket interface and TCP/IP protocols. By using SCMP, a complex application can be split across the multiple processors, resulting in faster overall computational throughput and real-time response. Some features of SCMP support are:

- Interprocess communication using socket interface and TCP/IP protocols.
- Availability of TCP/IP networking utilities like `rlogin`, `rcp`, etc.
- Diskless operation with booting over the backplane and NFS for remote file access.
- Backplane NFS transport for efficient access to remote files.
- Support for up to eight MVME boards on the backplane.

This chapter describes how to configure an SCMP environment for use with LynxOS.

Overview

Concepts

SCMP functions at the “data link” layer in a generic networking model. Communication above the SCMP layer relies on TCP/IP protocols. To an application, an SCMP connection looks like an Ethernet or SLIP connection. Figure 9-1 shows a generic network layer model with SCMP.

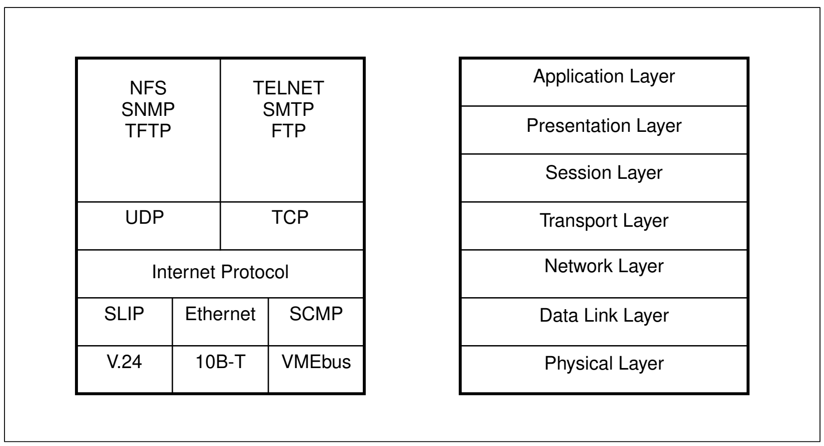

Figure 9-1: Generic Network Model Based on OSI/ISO.

The SCMP configuration consists of a primary processor and one or more secondary processor(s). The processor boards in the SCMP configuration can have local disks or be diskless clients.

NOTE: *The description of SCMP configuration in this chapter assumes that each processor on the shared VMEbus has a local disk with LynxOS installed. To configure diskless clients for SCMP, see “SCMP Netboot” on page 164.*

The *backplane network driver* is responsible for transferring packets over the VME backplane. This driver provides the same interface as an Ethernet driver to the TCP/IP protocol code and is implemented as a kernel thread. In this way, a cluster of processors running LynxOS can work together while sharing the common VME backplane. They can be configured to work on different applications or a complex application that can be split across processors. Such applications can communicate over the VME backplane using sockets.

Figure 9-2 shows a hypothetical configuration of VME processors that communicate, via SCMP, over a shared VME backplane.

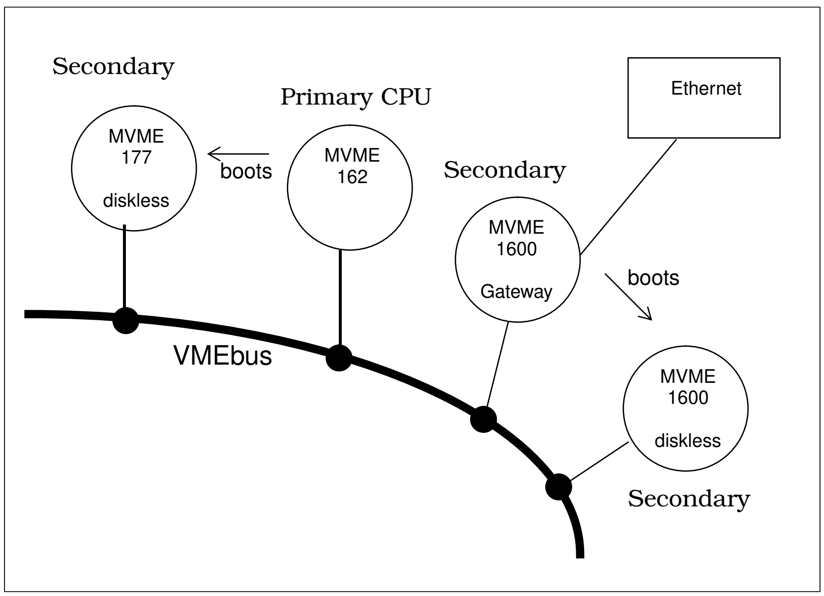

Figure 9-2: Hypothetical SCMP Configuration with Disk-based and Diskless Clients

The VMEbus backplane is considered to be an Internet network with its own network ID and each of the MVME board on the backplane is assigned a unique IP address within that network.

Configuration

Configuring an SCMP environment involves installing the hardware properly in the VME cage, configuring LynxOS with the appropriate information about the hardware, and initializing SCMP on the LynxOS machine.

The following sections describe these steps:

- Setting up the hardware for SCMP;
- Configuring the LynxOS backplane driver for each processor;
- Building a new LynxOS kernel with the backplane driver for each processor;
- Modifying start-up files on the primary processor;
- Starting the primary processor;
- Starting the secondary processor(s).

Example Used in Chapter

For simplicity, this chapter uses an example SCMP setup (shown in Figure 9-3) to describe how to configure an SCMP environment.

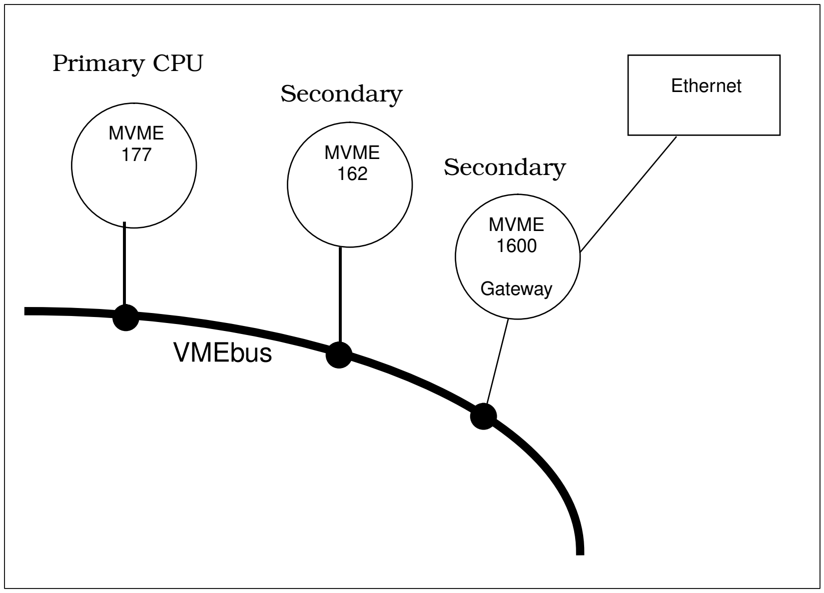

Figure 9-3: Example SCMP Configuration Used in Chapter

The processors in the example configuration have the following characteristics:

Table 9-1: Characteristics of Example SCMP Configuration

CPU Type	DRAM	Notes
MVME177 (68k)	8	
MVME162 (68k)	12	
MVME1600 (ppc)	8	Gateway to Ethernet

Setting Up the Hardware for SCMP

Selecting the Processor Boards

LynxOS SCMP supports up to eight MVME boards sharing a single VMEbus. The boards may be of one architecture type (68k-only, PowerPC-only) or be a mixture of architecture types (68k and PowerPC). LynxOS currently supports MVME162/167/177 (68k-based), and MVME1600 (PowerPC-based) processor boards.

You should have the documentation for your particular processor boards and the VME cage available during the initial setup of the hardware for SCMP. You also should be familiar with VMEbug commands and concepts before installing SCMP.

NOTE: *In this chapter, the term* VME*bug is used to generically refer to the Bug Monitor of* MVME *68k and PowerPC systems.)*

Assembling the System

Select one MVME board to be the *system controller* on the VMEbus. To do this, you must set the appropriate jumpers on the board (as described in your hardware manual) and place the board in slot 0 of the VME cage. Put the other board(s) in the adjacent slots.

From the perspective of LynxOS, there is one *primary processor* and one or more *secondary processor*(s). The primary processor may or may not be the same board as the system controller.

Assigning the Board Addresses

The MVME boards communicate by accessing each other's DRAM in the VMEbus A32/D32 address space. They also communicate by using the VME2chip's `SIG0` interrupt. To configure an SCMP environment, you need to assign unique addresses to each board so that this communication can occur.

VME Shared Memory Basics

Each board connected to the VMEbus is assigned a unique 32-bit address range for its DRAM. Figure 9-4 illustrates how shared memory is accessed through extended VME.

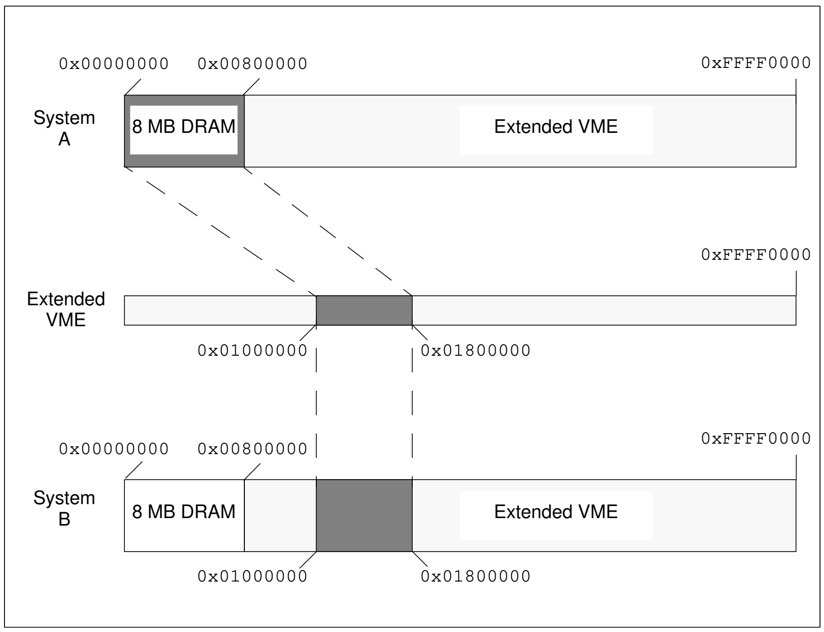

Figure 9-4: Example of Shared Memory on the VMEbus

For each processor in the SCMP environment, the following addresses must be selected and programmed into the VMEchip2:

- GCSR Group Base Address
- GCSR Board Base Address
- Slave Address

Selecting the Global Register Addresses

The Global Register (GCSR) addresses, group and board, are used together to define the address range accessible by VME short I/O space. The group address default is `CC`; the board address can have any value between `0` and `F`. They are put together as *xxy*0, where *xx* is the group address and *y* is the board address. For example:

Table 9-2: Assigning the GCSR Addresses

Processor	GCSR Group Address	GCSR Board Address	Combined GCSR Addresses
MVME177	CC	1	CC10
MVME162	CC	2	CC20
MVME1600	CC	3	CC30

Selecting the Slave Addresses

Each board also must have a unique slave address. The first usable slave address of the system is a function of the largest amount of DRAM on any one board, rounded up to a 16 MB boundary. The next available board address depends on the amount of DRAM of the individual board, again rounded up to the 16 MB boundary. Figure 9-3 shows an example.

Table 9-3: Example DRAM Start and End Address

Processor	DRAM	Starting DRAM (Slave) Address	Ending DRAM (Slave) Address
MVME177	8 MB	0x01000000	0x01FFFFFF
MVME162	12 MB	0x02000000	0x02FFFFFF
MVME1600	8 MB	0x03000000	0x03FFFFFF

These addresses will be used to program the VMEchip2 using the `env` command. There should be no overlap in the address ranges for each board.

Programming the VMEchip2

Now, the VMEchip2 on each board must be programmed with the GCSR and DRAM slave addresses. To program the VMEbug monitor, power up the system and execute the `env` command. Specify the configuration information as shown in Figure 9-5. End the session with a period (`.`) for the next query, and update the new values in non-volatile RAM. Remember to do this step for each processor board.

```
   177-Bug> env
Slave Enable #1 [Y/N] = Y? Y
Slave Starting Address #1 = 80000000? 01000000
Slave Ending Address #1   = 80FFFFFF? 01FFFFFF
Slave Address Translation Address #1 = 00000000? 0
Slave Address Translation Select #1  = FE000000?
FF000000
Slave Control #1 = 03FF? <Enter>
[...]
Master Enable #1 [Y/N] = N? Y
Master Starting Address #1 = 01000000? 01000000
Master Ending Address #1   = EFFFFFFF? EFFFFFFF
Master Control #1 = 0D? 0D
[...]
Short I/O (VMEbus A16) Enable [Y/N] = Y? Y
Short I/O (VMEbus A16) Control      = 01? 01
[...]
VMEC2 GCSR Group Base Address = CC? CC
VMEC2 GCSR Board Base Address = 01? 01
[...]
Update Non-Volatile RAM (Y/N)? Y
```

Figure 9-5: Programming the VMEchip2 on a MVME177 Board

Verifying the Hardware Setup

Before beginning the software (LynxOS) phase of setting up SCMP, you should verify that you have properly installed and configured the hardware.

A quick way to check the hardware setup is by using the `md` (memory dump) command at the VMEbug prompt. The following steps show how to test example setup in this chapter.

Can the MVME167 see the MVME177?

```
177-Bug> md 0 20
00000000  0000 0AD8 0000 0AD8 0000 0AD8 0000 0AD8
................
00000010  0000 0AD8 0000 0AD8 0000 0AD8 0000 0AD8
................

167-Bug> md 01000000 01000020
01000000  0000 0AD8 0000 0AD8 0000 0AD8 0000 0AD8
................
01000010  0000 0AD8 0000 0AD8 0000 0AD8 0000 0AD8
................
```

Can the MVME1600 see the MVME167?

```
167-Bug> md 0 20
00000000  0000 0AD0 0000 0AD0 0000 0AD0 0000 0AD0
................
00000010  0000 0AD0 0000 0AD0 0000 0AD0 0000 0AD0
................

PPC1-Bug> md C1000000 C1000020
C1000000  00000AD0 00000AD0 00000AD0 00000AD0
................
C1000010  00000AD0 00000AD0 00000AD0 00000AD0
................
```

Can the MVME167 see the MVME1600?

```
PPC1-Bug> md 0 20
00000000  7C3043A6 7C2802A6 7C3143A6 4BF04111
................
00000010  00000000 00000000 00000000 00000000
................
```

```
167-Bug> md 03000000 03000020
03000000  7C30 43A6 7C28 02A6 7C31 43A6 4BF0 4111
................
03000010  0000 0000 0000 0000 0000 0000 0000 0000
................
```

Error Messages

If you try to access a non-existent or incorrectly configured board, you will see the following type of message on the screen:

```
177-Bug> md 09000000
Exception: Access Error (Local Off Board)
PC =FF859A74 SR =2704
Format/Vector =4008
Effective/Instruction Address =09000000
Fault Status Longword         =01450028
```

Note that numbers and memory contents depend on the Bug version running or a previously executed application. The important thing is that the memory content is the same, and that no bus exception occurs.

For More Information

Please refer your hardware documentation for more details on the VMEbug monitor.

- MVME162BUG™ Debugging Package User's Manual
- MVME167BUG™ Debugging Package User's Manual
- PowerPCBUG™ Debugging Package User's Manual
- MVME162/MVME177 Single Board Computers Programmer's Reference Guide for more details.
- MVME167/MVME177 Single Board Computers Programmer's Reference Guide for more details.

Setting Up LynxOS for SCMP

This is the software stage of the SCMP setup. At this point, you'll be configuring the LynxOS kernel with a VME backplane driver and running the provided scripts that modify startup files to initialize SCMP after startup. This section assumes that you have already installed LynxOS on the local disk of each VME board in the SCMP environment. If you have not installed LynxOS, you should do so now.

NOTE: *Any number of nodes in an SCMP environment can be diskless. However, it is easier to configure the system if at least the primary processor has a local disk. If you have diskless secondary processor and a disk-based primary processor, configure the primary processor as described in this chapter. Then refer to "SCMP Netboot" on page 164.*

Copying the Current LynxOS Kernel

Since the SCMP setup scripts change the LynxOS kernel, it's a good idea to make a copy of the current running kernel before proceeding with scripts.

```
cp /lynx.os /lynx.os.pre-scmp
```

If you have difficulties booting LynxOS after configuring SCMP, you can boot this kernel copy instead. Refer to the `preboot` manual page for details on booting alternate kernels.

Choosing the Primary Processor

Select the board that will be the primary processor in your SCMP environment. The primary processor will contain a database of information about all the other processors in the SCMP system. You also need the following information for each processor in the system:

- Hostname
- DRAM starting address
- GCSR Board Base Address

For example:

Table 9-4: Information Needed for Each Processor

CPU Type	Hostname	DRAM Start Address	Combined GCSR Addresses
MVME177	scmp-cpu1	0x01000000	CC10
MVME162	scmp-cpu2	0x02000000	CC20
MVME1600	scmp-cpu3	0x03000000	CC30

Configuring the LynxOS Backplane Driver

The LynxOS backplane device driver must be configured for each of the processors in the SCMP environment. Complete the following steps for each processor in the system, starting with the primary processor.

Creating the Backplane Configuration File

First, create the file `/etc/bplane.conf`. This configuration file specifies the board ID, DRAM address, I/O address, and hostname for each board in the SCMP system. Figure 9-6 shows an example `/etc/bplane.conf` file.

```
# Type Boot-Flag DRAM-Address IO-Address Hostname

   1    0         0x0100000    0xFFFFCC10 scmp-cpu1
   1    0         0x0200000    0xFFFFCC20 192.1.1.2
   2    0         0x0300000    0xCFFFCC30 192.1.1.3
```

Figure 9-6: Example bplane.conf File

The fields specified in `/etc/bplane.conf` are:

Board Type	1 for 68k; 2 for PowerPC.
Boot Flags	Unused field; must be set to 0.
DRAM Address	Starting address for the board as specified with the `env` command. See “Programming the VMEchip2” on page 141.
I/O Address	Based on the combined GCSR group and board base address programmed into the VMEbug. See Table 9-2 and Table 9-4 for examples.
Hostname	Unique IP address or hostname for each board. When using the hostname, you must ensure that `/etc/hosts` contains the hostname with its appropriate IP address.

Refer to the manual page `bplane.conf(5)` for more information.

Modifying the Backplane Driver

Change to the `/sys/devices` directory:

`cd /sys/devices`

Edit the file if_bplaneinfo.c. The parameter MAX_NO_PROCESSORS must be equal to or greater than the total number of boards on the system (eight maximum). For example:

```
MAX_NO_PROCESSORS = 3
```

Rebuilding the Backplane Driver

Rebuild the backplane driver information file by typing “make” in the /sys/devices directory:

```
make
```

Rebuilding the LynxOS Kernel

Now that the configuration files and backplane driver files have been modified, rebuild the LynxOS kernel to incorporate the changes:

```
cd /sys/lynx.os
make install.tcpip
```

Choose the option “SCMP” and answer the questions presented by the script.

The VME backplane is considered to be an Internet network with its own Internet network ID. Choose a unique network address for the backplane network. Similarly, enter the corresponding Internet host IDs and hostnames for the processors boards on the backplane. These values should correspond to what was specified in /etc/bplane.conf.

Starting SCMP

After you have rebuilt the kernel, reboot the board to start SCMP.

```
reboot -a
```

NOTE: *The primary processor must be running SCMP before the secondary processor(s) can run SCMP.*

CHAPTER 10

Network Booting Diskless Clients with LynxOS

LynxOS supports network booting (*netboot*) of remote machines over a TCP/IP network. Netboot provides a way download, via an Ethernet or a Snugly Coupled Multi-Processing (SCMP) connection, a configured LynxOS kernel image on machines that lack a local hard disk. These diskless clients can also access remote file systems over the network. See the *LynxOS Hardware Support* document for the list of hardware configurations that support network booting.

Overview

Booting a diskless client involves:

- Creating a LynxOS kernel image to netboot.
- Configuring the diskless client.
- Downloading the LynxOS kernel.

These steps are generic to Ethernet and SCMP netboot.

Netboot Scripts

LynxOS provides ready-to-use scripts for both Ethernet and SCMP network booting. The Ethernet netboot scripts are in `/sys/romkit/remote_boot` and the SCMP netboot scripts are in `/sys/romkit/scmp_boot` for SCMP. These scripts have been created to produce a typical configuration for a diskless client. You can use them in their default configuration or as a template for your own application.

NOTE: *On LynxOS cross development systems, these scripts are located in* `$ENV_PREFIX/sys/romkit/remote_boot` *and* `$ENV_PREFIX/sys/romkit/scmp_boot`.

Copying Scripts Before Customizing

Since you may want to modify the default netboot configuration for your own application, we recommend that you make a working copy of the scripts. For example:

```
cd /sys/romkit
cp -r remote_boot ethernet_client
```

This will produce a working copy named `ethernet_client`.

NOTE: *This chapter uses the original name of the directories in the examples.*

Restrictions

The netboot scripts must be executed either on a LynxOS native or cross development system with the same type of architecture as the diskless client. For example, to create a netboot image for a MVME177 diskless client, you must execute the scripts on a disk-based LynxOS 68k machine or on a Sun Solaris machine running LynxOS cross development for 68k.

Ethernet Netboot

In Ethernet netboot, the client-specific or shared LynxOS kernel image is downloaded from a server using the Trivial File Transfer Protocol (TFTP), an architecture-dependent feature. For example, on LynxOS MVME (PowerPC, 68k) and SPARC systems, the firmware can be enabled directly to support TFTP.

Before You Begin

The easiest way to netboot a diskless client with LynxOS is to use the scripts in `/sys/romkit/remote_boot`. These scripts assume that a single disk-based LynxOS system will be used to build the netboot image, boot the diskless client, and serve the root file system for the diskless client via NFS.

To use the netboot scripts, the disk-based server must be the same architecture type (x86, PowerPC, 68k, SPARC) as the intended diskless client. Or, it can be a cross development host with the appropriate LynxOS tools for the target architecture.

Before proceeding with the scripts, you need some information about the disk-based server and diskless client(s). Table 10-1 shows a list of this information, with examples, to help you get started.

Table 10-1: Server and Client Information Needed to Configure Netboot (Denoted by X)

Parameter	Disk-Based Server	Diskless Client	Notes
Hostname	X	X	A text string created of letters (A-Za-z), digits (0-9), hyphen (-), and or slash (/). For example, client-ppc2 is a valid hostname.
IP Address	X	X	A valid internet protocol address, usually of the form A.B.C.D. For example, 198.1.1.2 is a valid IP address.
Ethernet Interface	X	X	The LynxOS device name for the network interface card on the diskless client. The manual page `intro(4)` contains a list of valid device names by hardware type.
Ethernet Address		X	A 6 byte hexadecimal number. For example, 08:00:3E:23:8C:BD is a valid Ethernet address. Use the `cnfg` command from the VMEbug monitor (68k and PowerPC clients) to find the Ethernet address. On x86 clients, look for a label on the Ethernet card.

Configuring the Disk-Based Server

To configure the disk-based server, you need to:

- Enable RARP.
- Enable TFTP services.
- Export root directory for diskless client via NFS.
- Build diskless client configuration files, the netbootable image, and necessary start-up files.
- Put the image and start-up files in a download directory.

The following sections give a brief description of what these steps involve. You can also refer to “Example—Netbooting a MVME 1600 (PowerPC)” on page 159 for step-by-step instructions.

Configuring RARP

Reverse Address Resolution Protocol (RARP) is a protocol to determine an associated Internet host address for a given Hardware-Address. On Ethernet networks, this is the Ethernet address. In this release of LynxOS, the RARP service facility has been moved to a separate utility called `rarpd`. This facility:

1. Reads the `/etc/ethers` database.
2. Connects to a network interface and listen to incoming RARP requests.
3. Answers any incoming requests for an Ethernet address listed in `/etc/ethers`.

To configure `rarpd`, just enter the Ethernet address and hostname of your diskless client in the file `/etc/ethers` on the LynxOS server. (See the `ethers(5)` manual page for more information.) You also need to add a corresponding entry with the IP address and hostname of the client into the `/etc/hosts` file.

Use the `rarpd` command to start the RARP daemon. The syntax of `rarpd` is shown below. (See the `rarpd(5)` manual page for more information.)

`rarpd` *ethernet_interface_name*

Enabling TFTP

The file `/etc/inetd.conf` contains the TFTP service information. This file must contain the following line:

```
tftp  dgram   udp  wait  root  /net/tftpd  tftpd
```

If you had to add this information, you will need to send a `SIGHUP` signal to the `inetd` process. Find the process ID using the `ps` command. Then send the `SIGHUP` signal with the `kill` command.

Exporting the Root File System Via NFS

The disk-based server needs to provide a root file system for the diskless client. The server exports this file system via NFS. When using the default settings for the netboot scripts, you must have the following directories and files in the root file system for the diskless client. Copy them from the LynxOS distribution to the directory that will be the root file system.

```
ENVIRONMENT
bin/
     bash
     exportfs
     ifconfig
     login
     syncer
     reboot
     touch
lib/
net/
     inetd
     portmap
     mountd
     nfsd
     rpc.lockd.clnt
     rpc.lockd.svc
     rpc.statd
sys/
tmp/
```

In addition, the following utilities are useful:

```
bin/
     cat
     df
     ftp
     less
     ls
     ping
     ps
     rlogin
```

Building the Netboot Image

The `Makefile` in `/sys/romkit/remote_boot` builds a client-specific configuration file for each diskless client that you want to boot. This configuration file is used to build the kernel image and supporting files that will be downloaded over the network to the diskless client.

Putting the Netboot Files in the Download Directory

This final step creates the directory `/tftpboot` and copies the netboot image and start-up files to this directory. The firmware of the diskless client must be configured with the full path name of this directory and the netbootable image.

Configuring the Diskless Client

To remotely boot the diskless client, you need to:

- Configure the firmware with information about the server, the download files, and the load address.
- Start netboot.

Since these commands are platform-dependent, the following sections briefly describe the commands by platform type.

Setting Up MVME (68k and PowerPC) Systems

For MVME 68k and PowerPC systems, the network booting code is available as a programmable option in the `env` settings of the VMEbug Monitor. To set this parameter, reboot/reset the CPU board and enter the VMEbug Monitor (this typically involves pressing the **Break** key during CPU reset).

Enabling NVRAM on 68k and PowerPC

We have observed that the real-time clock is disabled on new MVME (68k and PowerPC) boards. If you have a new board, you must first enable the real-time clock before network booting LynxOS. You only need to enable the real-time clock once on any given MVME board.

Use the `set` command at the VMEbug prompt (abbreviated `xxxx-Bug`) to set the time, thus enabling the clock:

```
xxxx-Bug> set <MMDDYYhhmm>
```

Disabling PReP-Boot on PowerPC

The PReP-Boot setting defaults to "yes" on PowerPC boards. Disable this ("no") with the `env` command.

```
PPC1-Bug> env
Network PReP-Boot Mode Enable [Y/N] = Y? N
...
Update Non-Volatile RAM (Y/N)? Y
```

Setting Netboot Parameters

Use the `niot` command at the VMEbug prompt to set the network booting parameters specific to your hardware configuration.

Only a few parameters need to be changed (as shown below). Consult the Motorola VMEbug documentation for more details about the commands.

```
xxxx-Bug> niot

Client IP Address = [198.1.1.2] ?
Server IP Address = [198.1.1.1] ?
...
(Choose the Subnet IP Address Mask appropriately)
...
Boot File Name = [/file] ?
Boot File Load Address = [00000000] ?
Boot File Execution Address = [00000000] ?
...
Update Non-Volatile RAM (Y/N)? Y
```

The boot file load and execution addresses are platform dependent as shown in Table 10-2.

Table 10-2: Boot File Addresses for 68k and PowerPC Systems

Target	Boot File Load Address	Boot File Execution Address
68k	00010000	00010020
PowerPC	00004000	00004020

Booting the Remote Kernel Image

Use the `nbo` command to network boot the image.

```
xxxx-Bug> nbo
```

Setting Up SPARC Systems

At the OpenBoot prompt, enter the following command for an individual boot-image (remember that *hex_IP_addr* is the hexadecimal representation of the diskless clients IP address).

```
ok 4000 dload /tftpboot/hex_IP_addr.lynxos.netboot
```

After the download has finished, enter:

```
ok init-program go
```

Setting Up x86 Systems

For LynxOS x86 machines, the `tftpboot` boot code is distributed on a 64K PROM†. Place this PROM in the `BOOTPROM` slot of the SMC Ethernet card. The `BOOTPROM` is different for the SMC Elite and Elite Ultra/EtherEZ Ethernet cards.

†. The netboot PROM for x86 systems must be purchase separately.

The `BOOTPROM` is enabled using the `ezsetup` command. This command can be given either after booting LynxOS from the local disk or after booting the system using DOS floppy disk and enabling `BOOTPROM` slot using the utility disk that comes with the Ethernet card.

The options for enabling network booting for SMC Elite Ethernet cards are:

```
IO Base Address :  0x240
RAM Base Address:  0xCC000
IRQ level       :  5
ROMBOOT Enabled :  Y
ROM Size        :  64K
ROM Base Address:  0xD0000
```

The options for enabling network booting for SMC Ultra Elite and SMC EtherEZ Ethernet cards are:

```
IO Base Address :  0x240
RAM Base Address:  0xCC000
IRQ level       :  5
ROMBOOT Enabled :  Y
ROM Size        :  32K
ROM Base Address:  0xD0000
```

The PROM occupies 64 KB of address space from `0xD0000`. Hence, care should be taken to avoid overlapping address with other cards in this address space.

Note that when the system is rebooted, the `tftpboot` code gets loaded by default. The `tftpboot` code sends RARP request message with its own Ethernet address to get its Internet address. When it receives the RARP response message from a server machine, it forms the kernel file name as:

`/tftpboot/`*hex_IP_addr*`.lynxos.netboot`

where *hex_IP_addr* is the uppercase hexadecimal representation of its Internet address. Then it uses TFTP on the server to get the kernel file.

Example—Netbooting a MVME 1600 (PowerPC)

The steps shown in this example assume the simple configuration shown in Table 10-3.

Table 10-3: Example Ethernet Netboot Configuration

Parameter	Disk-Based Server	Diskless Client
Platform	MVME 1600 (PowerPC)	MVME 1600 (PowerPC)
Hostname	server-ppc1	client-ppc2
IP Address	198.1.1.1	198.1.1.2
Ethernet Interface	dec0	dec0
Ethernet Address	Not needed.	08:00:3E:23:8C:BD
TFTP Directory	/tftpboot	Not applicable.
Root File System Directory on Server	/clients/client-ppc2	Not applicable.
Netboot Image on Server (after running netboot scripts).	/tftpboot/C6010102.lynxos.netboot	Not applicable.

Configuring the Disk-Based Server

1. Login to the disk-based server (`server-ppc1`).
2. Add client information to `/etc/ethers`.

`vi /etc/ethers`

```
# Each client must have a line with the
# Ethernet address and hostname in the
# following form:
# xx:xx:xx:xx:xx:xx   hostname
08:00:3E:23:8C:BD  client-ppc2
```

3. Add the IP address and hostname of the diskless client to the `/etc/hosts` file.

   `vi /etc/hosts`

   ```
   # IPaddress    hostname
   # 198.1.1.2     client-ppc2
   ```

4. Start the RARP daemon on the server. Remember that the device name specified to `rarpd` is the Ethernet interface on the server.

   `rarpd dec0 &`

5. Edit `/etc/inetd.conf`.

   `vi /etc/inetd.conf`

   ```
   # Add the following line
   tftp dgram udp wait root /net/tftpd  tftpd
   ```

6. Restart the `inetd` daemon. (See `ps(1)`.)

   `ps -ax | grep inetd`

   ```
   21  1 21 17  52 8 36 0.17 root W /net/inetd
   ```

   `kill -1 21`

7. Create the directory that will be exported to the diskless client. This example uses a symbolic link to the server's root directory as the root file system. (See "Exporting the Root File System Via NFS" on page 154 for the minimum list of files needed.)

   `mkdir /clients`
   `ln -s / /clients/client-ppc2`

8. Configure the exports file with the name and access permissions of the directory containing the root file system for the client. (See `exports(5)`.)

   `vi /etc/exports`

   ```
   # Export-directory     access=hostname
   /clients/client-ppc2   root=client-ppc2
   ```

9. Export the root file system with the `exportfs` command. (See `exportfs(1)`.)

   `exportfs -av`

10. Verify that the `portmap`, `mountd`, and `nfsd` demons are running using the command `ps -ax`. These daemons can also be started at the shell prompt.

```
pid ppid pgrp pri text stk data time dev user S name
21   1  21  17   52   8   36 0.17 root  W /net/inetd
26   1  26  17   40   12  32 0.04 root  W /net/portmap
28   1  28  17   52   8   52 0.03 root  W /net/mountd
30   1  30  17   40   8   40 1.58 root  W /net/nfsd
...
```

11. Begin the netboot configuration scripts. See Figure 10-1 for the example output.

    cd /sys/romkit/remote_boot
    make config

```
Ethernet Netboot Configuration
******************************
TFTP-Server Configuration
=========================
Please specify the location of the TFTP-directory[/tftpboot]:<Return>

NFS Server Configuration
========================
Specify the name of the NFS-Server for the root filesystem
[lynxdemo]:server-ppc1
Specify the directory containing the root filesystem on the NFS-Server
[/clients/cpu2]:/clients/client-ppc2

Diskless Client Configuration
=============================
Enter the IP-Address of the diskless client (aaa.bbb.ccc.ddd)
[192.1.1.2]: 198.1.1.2
   Checking IP-Address 192.1.1.2...ok.

Enter the name of the Ethernet interface [oblan0]: dec0
Do you want to use a shared KDI for this client? [y,n] (n): n
Hosts database to use on this client [/etc/hosts]: <Return>

   Writing file netboot.config-1...ok.
```

Figure 10-1: Example Ethernet Netboot Configuration

12. Create the image and client files (still in `/sys/romkit/remote_boot`).

    `make all`

    Select the network type (Ethernet) and interface name for the client. Select the defaults for NFS support.

13. Put the new kernel image and supporting files into the download directory, `/tftpboot`.

    `make install`

At the end of this procedure, the `/tftpboot` directory on the disk-based server will have the following files:

`C6010102.lynxos.netboot`	LynxOS netbootable kernel image.
`C6010102.hosts`	Hostname database table
`C6010102.fstab`	File system table with mounted directory information.

Note that `C6010102` is the hexadecimal representation of the client's IP address (198.1.1.2). Write down the name of the netbootable kernel image so that you can configure the firmware of the diskless client.

Starting the Diskless Client

1. Reset the diskless client (client-ppc2) and enter the VMEbug monitor by pressing the **Break** key at system start-up.

2. Disable PReP-Boot with `env` command.

```
xxxx-Bug> env
Network PReP-Boot Mode Enable [Y/N] = Y? N
...
Update Non-Volatile RAM (Y/N)? Y
```

3. Use `niot` to specify the client and server information. Also, set TFTP/ARP request retry to a non-zero number to ensure that the netboot will retry on a busy network.

```
xxxx-Bug> niot
Client IP Address = 198.1.1.2
Server IP Address = 198.1.1.1
...
Boot File Name =
/tftpboot/C6010102.lynxos.netboot
Boot File Load Address = 4000
Boot File Execution Address = 4020
...
TFTP/ARP Request Retry =00? 20
```

4. Use the `nbo` command to boot the remote kernel.

```
xxxx-Bug> nbo
```

Now, you are up and running LynxOS!

SCMP Netboot

The Netboot facility for systems using SCMP is very similar to the Ethernet version. The main differences are:

1. SCMP only runs on 68k and PowerPC platforms that employ the VMEchip2.
2. Boot image sharing is not possible.
3. There is no RARP server needed.
4. A set of tools is necessary to operate SCMP (see `bpconfig(1)`, `slaveboot(1)`, `bplane.conf(5)`).

NOTE: Before booting a diskless client over a SCMP network, configure the SCMP network as described in Chapter 9, “Snugly Coupled Multi-Processing”.

How Does It Work?

Like in Ethernet netboot, a diskless client is booted in two steps. The tools and protocols are, however, slightly different.

1. The server downloads the boot-image using the `slaveboot` command. This will automatically start the client.†
2. The boot procedure establishes the client as an SCMP network node, determines its root file system server, and then loads the LynxOS kernel.

†. On some MVME1600 PowerPC systems you will have to start the client manually by typing “GO 4020”.

Configuring the Disk-Based Server

As in Ethernet netboot, the following steps are need to configure the disk-based server:

- Enable TFTP services.
- Export root directory for diskless client via NFS.
- Build diskless client configuration files, the netbootable image, and necessary start-up files.
- Put image and start-up files in a download directory.

See "Ethernet Netboot" on page 151 for an overview of these steps. Then read the following sections for SCMP-specific information.

Building the SCMP Netboot Image

The `Makefile` in `/sys/romkit/remote_boot` builds a client-specific configuration file for each diskless client that you want to boot. This configuration file is used to build the kernel image and supporting files that will be downloaded over the network to the diskless client.

```
cd /sys/romkit/scmp_boot
make config
```

Figure 10-2 shows an example.

```
SCMP Netboot Configuration
**************************

TFTP-Server Configuration
=========================
Please specify the location of the TFTP-directory[/tftpboot]:<Return>

Enter the IP-Address the TFTP-Server (aaa.bbb.ccc.ddd) [192.1.1.1]:
198.1.1.1
   Checking IP-Address 192.1.1.1...ok.

NFS Server Configuration
========================
Specify the name of the NFS-Server for the root filesystem
[lynxdemo]:server-ppc1

Specify the directory containing the root filesystem on the NFS-Server
[/clients/cpu2]:/clients/client-ppc2

Diskless Client Configuration
=============================
Enter the IP-Address of the diskless client (aaa.bbb.ccc.ddd)
[192.1.1.2]: 198.1.1.2
   Checking IP-Address 192.1.1.2...ok.

Hosts database to use on this client [/etc/hosts]: <Return>

SCMP configuration database to use on this client [/etc/bplane.conf]:
<Return>

   Writing file netboot.config-1...ok.
```

Figure 10-2: Example SCMP Netboot Configuration

You can also configure multiple clients within a single build environment by giving them different CPU IDs:

```
make CPUID=2 config
```

The remainder of the build procedure is identical to the Ethernet version. To create the netboot image (still in `/sys/romkit/scmp_boot`):

`make all`

To install the image and client files in the download directory: (`/tftpboot` by default):

`make install`

Starting a Diskless Client

On the server, which also must run SCMP, type:

`slaveboot /tftpboot/`*hex_IP_addr***`.lynxos.netboot`**
client-hostname

where *hex_IP_addr* is the hexadecimal notation of the client's IP address (e.g. `C0010203` for `192.1.2.3`) and *client-hostname* is the hostname of the client within the SCMP network (instead of hostname, you may also use the IP address in dot-notation). This will automatically start the client.† See also the `slaveboot(1)` manual page.

†. On some MVME1600 PowerPC systems you will have to start the client manually by typing ``GO 4020".

Troubleshooting

When booting the client a message "inetd: I/O error" appears.

The client's root file system was not exported with root permissions. For example, if the client's name is "flipper," the `/etc/exports` file should contain:

```
/clients/server_68k root=flipper
```

The downloaded boot-image cannot be executed.

This may happen if separate machines are being used as servers–one for TFTP and one for RARP. This error message indicates that the TFTP-server answered the RARP request of the diskless client. Make sure that only the real server machine is able to reply to a client's RARP request.

If this is not feasible, change the boot script `rc` that is embedded in `rc.sh`. Search for the `netboot` utility call and add the argument `-s` *tftpserver*. This argument forces the use of a specific server.

The SCMP network doesn't seem to work

Make sure that `/etc/bplane.conf` contains IP addresses only (not symbolic hostnames). The hosts database `/etc/hosts` is downloaded *after* the initial SCMP configuration If you need to use symbolic hostnames in `/etc/bplane.conf`, modify the SCMP kernel specification file to include a hostname database.

Advanced Issues for Ethernet Netboot

Sharing a Boot Image

The build environment for Ethernet Netboot is prepared to support shared boot-images. This reduces the needed disk space and makes kernel updates or changes easy, since they will automatically apply to all clients sharing the same boot-image.

For all clients, select the shared-KDI option during `make config`:

```
Do you want to use a shared KDI for this client? [y,n]
(n): y
```

In this mode of configuration, the `Makefile` offers slightly different options. Instead of only `make all`, and `make install`, you have now `make kdi` and `make install_kdi`.

For *only one* client, enter the following commands to build and install a sharable boot image:

`make install_kdi`

For more than one client, enter the following commands to build and install a sharable boot image. See "Configuring a Second Client" on page 172 for more information about *CPUID*.

`make [CPUID=`*`xxx`*`] install`

Make sure that all of your clients can actually share the same boot image. For example, you may not be able to share a boot image between several x86-based systems because they may have different network controllers.

Cleaning Up the Working Directory

During development (or after) you may decide to clean up the working directory. The following make command deletes the kernel build subdirectory.

```
make clean
```

Client Information Files

For each client, a dedicated subdirectory `sys-`*CPUID* is created, where the kernel, the boot-image and the additional kernel and client-specific files will be created (see below). Each client's configuration information is kept in a file called `netboot.config-`*CPUID*. This file can be included in shell scripts and in Makefiles. In a shell script, refer to a variable using `$varname`. In a Makefile, use `$(varname)`.

Adding Files to the RAM Disk

For the architecture you are working with, edit the file *target*`.netboot.spec` and enter files and directories as needed. See the `mkimage(1)` and `mkimage.spec(5)` for command syntax.

More About Kernel-Specific Files

Kernel-specific files contain information or parameters that are special for the kernel being run by the diskless client.

The file `rc.sh` is a self-extracting script that creates the final version of the file containing the network interface name. `rc.network` is actually copied from `rc.network.tmpl` and is manipulated by the kernel configuration scripts to enable or disable certain network daemons depending on what modules are configured into the kernel.

More About Client-Specific Files

Client-specific files contain information or parameters that are special for the diskless client. In the standard LynxOS distribution, there are two client-specific files: `/etc/fstab` and `/etc/hosts`. The file system description database, `/etc/fstab`, is built from `fstab.sh`. This is an self-extracting script that creates the final `fstab` with the proper server and pathnames.

Adding Client-Specific Files

The following files must be changed to include another client-specific file:

```
Config-Netboot
Makefile
tftplist
```

Config-Netboot

To add additional files, modify `Config-Netboot`. For example, to add a hostname database:

- Add a default value for your file in the header section, provide an empty string if there is no default.
- In section “Diskless Client Configuration”, add the following code:

```
use_default client_hosts_file
promptstr "user prompt" $def_var_name
var_name
echo ""
```

Makefile

The variable `TFTPFILES` contains the list of files that are to be installed as client-specific files. Add your file name here. If you specified it in `Config-Netboot`, you should now refer to it using `$(var-name)`.

tftplist

Add another line to this file. Provide the file name, the TFTP-server (without the path and IP address), the final destination in the RAM disk as included by the `ramdisk` driver, and (optionally) the access permissions for the file.

Other Options

You can also use self-extracting scripts to produce a file that uses one or more of the configuration values. See the files `rc.sh` and `fstab.sh` on how to do this. The `Makefile` contains generic rules how to run those scripts, so you only have to add a file `$(SYSDIR)/myscript` to the `TFTPFILES` variable and a file `myscript.sh` in the build directory.

Configuring a Second Client

When you invoke `make config`, the information that you enter is saved in a file called `netboot.config-`*CPUID*. The *CPUID* file extension distinguishes multiple diskless clients and can be any unique string or number.

To configure a second client with a different *CPUID*, type:

```
make CPUID=2 config
```

To create the boot image for this and other additional clients, you must specify it on the command line.

```
make CPUID=2 all
```

CHAPTER 11

Shared Libraries

A shared library is a collection of routines organized in such a way as to allow the code to be shared among processes, with the primary goal of reducing memory usage both on the system and on disk. Both the usage and the building of shared libraries are explained in this chapter.

NOTE: *This release of LynxOS supports shared libraries only on x86, 68k, and SPARC systems.*

Components of a Shared Library

A shared library has two components—the host shared library and the target shared library. The host shared library is an ordinary archive file that is used to link-edit programs with the shared library. It is a true archive file, which means that it can be used like any other non-shared archive library.

The target shared library has the form of an executable COFF object file. Whenever a program that uses the shared library is executed, the operating system maps the target shared library's text into the address space of that process and creates a separate copy of the target shared library's data for that process. The text is shared, but the data is not. The code for a target shared library is not kept in core when no process is using it.

In order for the text of the target shared library to be shared among multiple processes, it must reside at a pre-determined virtual address. Moreover, since this text must be identical for all processes, any data it accesses must also reside at a pre-determined virtual address. This is true even though the data itself is not shared.

In order for a process to use the code in the target shared library, it must know the virtual address of the routines in the library. This is where the host shared library comes in. The host shared library contains object files with no loadable text or data sections, whose only useful information are their symbol tables. (The host shared library may also contain object files that do contain loadable sections. This will be explained later.) The symbol table of such an object file will contain the absolute virtual addresses of symbols in the target shared library.

Effects of a Shared Library

The effects of using a shared library instead of an ordinary non-shared archive library are as follows:

System Memory

System memory usage will be different because multiple processes can share the library code. The amount of system memory that is saved (or lost) by using a shared library depends on three factors—the contents of the shared library, the set of programs that are typically run on the system, and the load on the system. The following relationships are true in general:

- A shared library consisting of commonly-used routines saves more system memory than one that has lots of rarely-used routines.
- A shared library that is used by a varied and numerous assortment of programs will save more system memory than a shared library that is used by only one or two programs.

A shared library will save more system memory on a heavily-loaded system than on a lightly-loaded one.

Two rules to keep in mind when using shared libraries are:

1. Multiple copies of the same program will never save system memory among themselves by using a shared library because the operating system ensures that their text sections are shared anyway. In fact, among themselves they are almost sure to waste some system memory.
2. Routines and data items in the shared library that are rarely used will waste system memory simply by their lack of use: A rarely used routine wastes memory because the entire shared library resides in memory whenever it is used, regardless of which subset of routines from the library are actually linked. A rarely used data item is even worse because it wastes system memory equal to its size multiplied by how many processes are currently using the shared library. This is because each process has its own copy of the data.

Disk Memory

Disk memory usage will typically be lower because a program linked with a shared library is almost always smaller than the same program linked with an equivalent archive library. The shared library itself is not a factor because it is comparable in size to an equivalent archive library.

Program Maintenance

Maintaining programs that use a shared library is somewhat easier than maintaining programs that use an archive library because if an error in the library is fixed, only the shared library needs to be remade. The programs do not need to be relinked with the library because the operating system maps the shared library code into their address spaces at run time. In effect, modifying the contents of a shared library modifies all of the programs that use it.

Note that the above statements hold true only when the change to the shared library is backward compatible. Backward compatibility should never be a problem, though, as long as the shared library is built using the guidelines described later in this chapter.

Using a Shared Library

Using a shared library is almost as easy as using an ordinary non-shared archive library. The method is simply to link with the host shared library. For example, the shared version of the standard C library resides in the host shared library `/lib/libc_s.a` and its target shared library `/shlib/libc_s`. To link with this library, the following line would be used:

ld <*object files*> /lib/libc_s.a

When compiling and linking, the shorthand notation works also:

```
gcc -o <program> <source files> -lc_s
```

It is possible to have a shared library that depends on another shared library (i.e., a shared library that references symbols from another). In this case, both shared libraries must be specified on the `gcc` or `ld` line in the order of their dependency.

There is nothing special about linking with a host shared library, so any source code that can be linked with a given archive library will also link with an equivalent shared library. According to the standard adopted by System V UNIX, a shared library will be identified by a `_s` suffix. Shared libraries under LynxOS will follow the same standard.

More on Using Shared Libraries

From a user's point of view, linking with a host shared library works the same as linking with an archive library. Internally, however, there are several points where the two differ. The following paragraphs explain these differences.

Whenever the linker resolves an undefined symbol using an object module from a library (any library), it copies the regular sections of that object module (i.e..`text`, `.data`, `.bss`) to the program being created, and it adds any new symbols used in that object module, but not yet defined, to the current list of undefined symbols. In an archive library, these regular sections are always present. In a shared library, however, if the object module is shared (it is possible to have non-shared object modules in a shared library - more on this later), then it will not have any `.text`, `.data`, or `.bss` sections. It may have an `.init` section, which the linker will copy into the program. The purpose of the `.init` section will be explained in the next paragraph. The only other thing for the linker to do with such a module is add its undefined symbols to the list. The symbol table of each shared object module in a given shared library contains an undefined version of a particular symbol that is unique to the shared library. This symbol is defined in a special object module in the shared library, so if the program being created needs any of the shared object modules from the library, then it is guaranteed to also need this special object module, in order to resolve the unique symbol. This special object module contains no `.text`, `.data`, or `.bss` sections (just like any other shared object module), but it does contain a `.lib` section, which will get copied into the program by the linker. The `.lib` section contains the pathname of the target shared library. When the operating system executes the resulting program, it will use the pathname in the `.lib` section to locate the target shared library file. This fact can be used to determine whether a given COFF program uses shared libraries: If it does, then the `size` utility with the `-n` option will reveal the existence (and size) of any shared libraries it uses. To determine the names of the shared libraries used by a given COFF program, use the `dump` utility with the `-L` option.

As mentioned in the preceding paragraph, some of the shared object modules in a shared library contain an `.init` section. This section contains initialization code that must be executed before any of the code in the object module's `.text` section is executed. To guarantee this, the first object module on the linker command line must be `/lib/init1.o` and the last object module (after all of the libraries) must be `/lib/initn.o`. The object file `/lib/init1.o` contains an `.init` section in addition to its other regular sections. This `.init` section contains no code - only the label `_imain`. The `.text` section of `/lib/init1.o` calls `_imain` before calling `main`. The object file `/lib/initn.o` also contains an `.init` section, and this sections consists of a single assembly `RET` (return from procedure) instruction. Since `/lib/init1.o` comes first on the linker line and `/lib/initn.o` comes last, then the `.init` sections from all of the shared object modules will be put between the `.init` sections of `/lib/init1.o` and `/lib/initn.o`. The result is that the `_imain` routine contains the initialization code for all of the shared object modules. The `-k` (COFF output) option of `cc` calls the linker with `/lib/init1.o` first and `/lib/initn.o` last. This is sufficient because currently a program must be COFF in order to use shared libraries.

The initialization code itself consists entirely of statements of the following form:

`ptr = &import;`

The variable *import* is known as an imported symbol. It is defined somewhere outside the shared library, either in the user program or in some non-shared object module in the shared library. The variable *ptr* is defined in the shared library and is used by the shared library code to indirectly reference the variable, *import*. This is why the initialization code must be executed first. If it were not, then `ptr` would not be pointing to *import* when it is used by the shared library code.

Recall that a shared object module from a shared library contains no `.text`, `.data`, or `.bss` sections. Its symbol table, however, still contains all of the defined symbols that its non-shared counterpart would have. The difference is that the symbols defined in the shared object module are absolute symbols whose values are the virtual addresses of those symbols in the target shared library. When such a symbol is used to resolve a reference from the user program, this virtual address is written into the user program so that it will directly access the symbol from the target shared library. Remember that at run time, the operating system maps the target shared library into the virtual address space of the process.

It was mentioned previously that the host shared library may contain non-shared object modules in addition to its shared object modules. This is accomplished in two steps: First a "pure" host shared library is built using the `mkshlib` utility. Second, additional non-shared object modules are added to the host shared library using the `libr` utility in the normal fashion. This is possible because the host shared library is a true archive library.

The Branch Table

The routines in a shared library are organized using a branch table. The branch table is a table of `JUMP` instructions at the beginning of the target shared library, where each `JUMP` instruction jumps to some routine in the library. When code in the user program calls a shared library routine, it calls the address of the `JUMP` instruction, not the routine itself. Unlike the other features of a shared library, the branch table is really not necessary at all, but it does make maintenance easier.

For example, assume that a program has been linked with a shared library and that the shared library is subsequently updated. If the routines were not organized using a branch table, then the virtual addresses of the routines might be different after the update. If the old program is then executed with the new shared library in place, then the user code in the program would call routines at addresses that are no longer valid.

The program would have to be re-compiled in order to work with the new library. The branch table eliminates this problem by fixing the call address of each routine in the library as some slot in the branch table. The address of this slot remains unchanged across library updates.

Building a Shared Library

This section describes the steps to follow in order to build a shared library. It is written assuming the reader is familiar with the construction of ordinary non-shared archive libraries.

Naming a Shared Library

Naming a shared library involves the following:

1. Choosing the root name

   As an example, the shared version of the standard C library has the root name `libc_s`. As was mentioned previously, it is customary to use the `_s` suffix when naming a shared library. The root name is also the name of the target shared library. The name of the host shared library is obtained by adding `.a` to the end of the root name.

2. Determining the directory for host shared library

   Host shared libraries usually reside in `/lib` or `/usr/lib`.

3. Determining the directory for target shared library

   Target shared libraries usually reside in `/shlib`. Keep in mind that the target shared library's location is somewhat more important than the host shared library's location because the full pathname of the target shared library is recorded into programs that use it.

Choosing Virtual Addresses

The sections of a target shared library reside in memory at a pre-determined virtual address. These addresses must be decided upon before the shared library is built. Table 11-1 shows the valid range for shared library virtual address assignments for each LynxOS platform.

Table 11-1: Virtual Address Range for Shared Libraries, By Platform

Platform	Start Address	End Address
x86	`0x80000000`	`0xDA800000`
68k	`0x80000000`	`0xDA800000`
SPARC	`0x1E0A6000`	`0x1F8A6000`

It is possible for two or more shared libraries to have the same virtual address assignments. Even if these libraries are used concurrently, there would only be a problem if they were being used by the same program. This is because each process has its own virtual address space.

Nevertheless, in order to avoid this problem, it is generally a good idea to choose virtual address assignments that do not conflict with existing shared libraries. You should also make sure that the sections within a single shared library do not overlap, as this will obviously cause problems.

Finally, the `.bss` section of a target shared library need not be given a virtual address assignment. The default is to place it immediately following the end of the `.data` section.

Shared Library Contents

Choosing the contents of a shared library involves more considerations than choosing the contents of an ordinary non-shared archive library. It is perhaps the most important factor determining whether or not the shared library will save system memory usage. When choosing routines for the shared library, use the following guidelines:

1. It is generally a good idea to include routines such as `printf` that are used often by many different programs.
2. It is generally a bad idea to include routines that are rarely used. Recall that when a shared library is used, the entire target shared library is loaded into memory, regardless of what routines were linked.
3. It is generally a bad idea to include routines that have large amounts of data. Recall that the data of a shared library is not shared, so each process that uses the library gets its own private copy. Even if the routine is commonly used, it can still waste space because any program that does not happen to use it still gets a copy of its data.

4. It is generally a good idea to make the shared library completely resolved. Although it is possible to have one shared library depend on another shared library (see the `#objects` noload directive), it is often worth the effort to include the needed routines in the first shared library, because the resulting target shared library will almost certainly be smaller than the sum of the two target shared libraries. (Recall that if one shared library depends on another, then both target shared libraries must be loaded into memory whenever a program uses it, because the program will explicitly need the other library as well.)

   It should be noted here that it is impossible for a shared library routine to directly call a routine defined in the user program. The only way for a shared library routine to call a user-defined routine is to make the user-defined routine an imported symbol to the library and to modify the shared library code to call the imported routine indirectly. See the next subsection for a discussion of imported symbols.

Modifying Source Code

The C code from an existing archive library will not necessarily work as part of a shared library because it may reference variables outside the library. The existing C code may also waste too much space, or it may be written in such a way that maintaining backward compatibility in the future would be difficult. All of these issues—imported symbols, saving space, and backward compatibility—are discussed in the following subsections, with instructions on how to modify existing C code to meet them.

Imported Symbols

As mentioned above, C code from an existing archive library that references variables outside the library will not work as part of a shared library. In order to work in the shared library, the existing C code must be modified to use pointers to imported symbols in place of any referenced symbols that will not be in the shared library. For example, assume the source file `file.c` contains the following line:

```
abc = xyz;
```

and that `xyz` is an `external` symbol that will not be in the shared library. In this case, `xyz` must be turned into an imported symbol by defining a new symbol, call it `xyz_ptr`, in `file.c`, and modifying the above C line to read:

```
abc = *xyz_ptr;
```

The new symbol `xyz_ptr` points to the symbol `xyz`. The next step is to ensure that `xyz_ptr` is assigned the address of `xyz` before any reference to `*xyz_ptr`. This is done by adding to the shared library specification file an initialization specification for the object file `file.o` that contains the following line:

```
xyz_ptr xyz
```

In effect, this line adds assembly code equivalent to the following C code

```
xyz_ptr = &xyz;
```

to the `.init` section of `file.o` (see the next section). As was mentioned previously, the linker ensures that all of the code collected from the `.init` sections of shared object modules will be executed prior to calling `main`, so all imported symbol pointers will be assigned before they are referenced.

Imported symbols do not have to be data symbols. Function names may also be imported. The modifications to the existing code are analogous to the above example.

Saving Space

In order to make the shared library save space, global data should be minimized. One technique for doing this is to use local (stack) variables instead of global variables wherever possible. Another technique is to allocate buffers dynamically (e.g., with `malloc`) instead of statically. These techniques are effectively identical methods of reducing the `.data` section of the shared library. This is important because each process that uses the shared library gets its own separate copy of the `.data` section.

The above space-saving techniques have the added benefit of making maintenance of the shared library easier by reducing the number of external symbols in the library. The number of external symbols can be further reduced by explicitly using static variables wherever possible.

Backward Compatibility

Whenever a shared library is updated, all programs that use it are effectively updated as well. Care must be taken to ensure that the addresses of all `external` library symbols remain unchanged across the update. Otherwise some programs that use the library will no longer work (and therefore would need to be relinked). It therefore pays to organize the code in the shared library in such a way that backward compatibility is easy to maintain.

There are two types of `external` symbols in a shared library—function symbols and data symbols. The branch table ensures that the addresses of function symbols will not change across updates, but it does nothing for data symbols. In order to ensure that the addresses of `external` data symbols do not change across updates, the following two guidelines should be followed when the shared library is created:

1. Initialize all `external` data symbols
2. Define text and `external` data in separate source files, and list all data modules ahead of the text modules in the specification file.

Initializing all `external` data symbols means that there will be no `.bss` (uninitialized data) section in the shared library. Under System V, this is a strict requirement, because the System V `mkshlib` utility will give a fatal error if it finds an uninitialized `external` symbol. Under LynxOS, this is not a strict requirement, but it is recommended because there is no way of ensuring that the addresses of uninitialized data symbols will not change across updates. Initializing all `external` data symbols has the side affect of increasing the size of the `.data` section, causing the target shared library to take up more disk space, but the maintenance benefits far outweigh this cost.

The purpose of separating the `external` data from the text is actually to separate the `external` data from any `static` data that may be embedded within the text (such as `static` strings). In addition to this, the list of object modules given in the specification file (see the next section) should be ordered so that all of the data modules are listed before any of the text modules. Doing this ensures that every external data symbol will have a lower virtual address than every static data symbol. With this arrangement, the addresses of the external data symbols are immune to changes in the text modules, because they will always be lower.

If any new external data symbols are added during an update, they should of course be added to the end of the group of existing external data symbols. This way, the address assignments of the existing external data symbols remains unchanged.

The Specification File

The shared library specification file outlines the construction of both the host and target shared libraries. It is organized using a simple grammar that uses directives. Some directives are one-line directives and some are multi-line. Multi-line directives are followed by one or more specification lines and remain "in effect" until the next directive or until the end of the file.

`#address` *sectname address*

Specifies that section, *sectname*, in the target shared library will start at address, *address*. This directive must always be specified for the `.text` section, and it must be specified for the `.data` section if the `.data` section is not empty, or if the `.bss` section is not empty and the `.bss` section is not given its own starting address directive. If the `.bss` section is not given its own starting address directive, then the `.bss` section will start at the end of the `.data` section. Recall, however, that backward compatibility is easier to maintain if there is no `.bss` section at all (see the previous section).

`#target` *pathname*

Specifies that the target shared library is expected to reside in file *pathname* on the target machine. This is the pathname that the operating system uses to locate the target shared library file when executing programs that use this shared library. This pathname will usually be equal to the target given with the `-t` command line option to `mkshlib`, but it doesn't have to. The difference is that the `#target` directive specifies the pathname that actually gets recorded into the host shared library (and consequently into any program that links with it), while the `-t` option specifies the name of the target shared library file created by `mkshlib`. This allows the target shared library file created by `mkshlib` to be moved to a different pathname, perhaps on a different system. In order to be used, of course, the target shared library file must end up with a pathname that exactly matches the `#target` directive's *pathname*.

This directive must be specified exactly once.

`#branch`

Specifies the beginning of a multi-line branch table specification. Each line of the branch table specification has the following format:

`funcname` *whitespace* `position`

where `funcname` is the name of an external function symbol and `position` is a positive integer or range of positive integers specifying a slot or range of slots in the branch table. A range has the form `position1-position2`. The only restriction on the branch table is that it must have exactly one symbol associated with each position between one and the highest specified position, inclusive.

In addition to using a range of positions, a symbol may be associated with more than one position by specifying it on more than one specification line. Whatever the case, if a symbol is associated with more than one position, then it is defined to have the address of the slot in the branch table corresponding to the highest such position. All other slots associated with that symbol are "dummy" slots that are not used. In future versions of the shared library, these "dummy" slots can be used for new function symbols.

This directive must be specified exactly once.

`#objects` Specifies the beginning of a list of object files used to create the host and target shared libraries. The object files in this list are separated by white space, so that several may be specified per line. The ordering of this list is significant, because it is used in both linking the target shared library and creating the host shared library.

This directive must be specified exactly once.

`#objects` *noload*

Specifies the beginning of a list of host shared libraries to be searched when resolving symbols not defined in this target shared library. These libraries are searched in the order given. The host shared libraries in this list are separated by white space, so that several may be specified per line. Each host shared library can be specified either by its full pathname or in the form `-lX`, where `/lib/libX.a` is the name of the host shared library. This shorthand notation works the same way as for `cc`.

This directive may be specified any number of times.

`#hide linker` [*]

Specifies that certain symbols in the host shared library's object files that would normally be external shall be made static instead. If the optional * is specified, then the directive is a one-line directive, and it causes all external symbols to be made into static symbols. Otherwise, the directive is followed by one or more lines, each specifying one or more patterns separated by white space. Each external symbol that matches any of the patterns is made static.

The `#hide` directive has no effect on symbols specified in the `#init` or `#branch` directives. Such symbols are always external.

This directive may be specified any number of times.

`#export linker` [*]

Specifies that certain external symbols in the host shared library's object files that would otherwise have been switched to static by a `#hide` directive, should nonetheless remain external. This directive is useful when it is desirable to "hide" all but a small list of symbols. For example,

```
#hide linker *
#export linker
symbol1
symbol2
```

causes all `external` symbols to be made `static` except `symbol1`, `symbol2`, and of course those specified in the `#init` or `#branch` directives. This directive may be specified any number of times.

`#init` *object*

Specifies the beginning of a multi-line initialization specification for the object file, *object*. Each line of an initialization specification has the following format:

`ptr` *whitespace* `import`

where `ptr` is a pointer to the imported symbol, `import`. The symbol `ptr` must be defined in the object file, *object*. `mkshlib` generates assembly code for each such line that is equivalent to the following C code:

`ptr = &`*import*`;`

This directive may be specified at most once per object file.

`#ident` *string* — Specifies that a `.comment` section, containing the string *string* is to be added to the target shared library and to each object file in the host shared library.

`##` — Specifies a comment that extends to the end of the line. `mkshlib` ignores all comment lines.

The mkshlib Utility

The `mkshlib` utility is used to create a shared library. It builds both the host and target shared libraries. The command line syntax is as follows:

`mkshlib -s` *specfil* `-t` *target* `[-h` *host*`] [-n] [-L` *dir* ...`] [-q]`

The contents of the specification file were described in the previous section. The command line options are described below.

`-s` *specfil* — Specifies the shared library specification file, *specfil*.

`-t` *target* — Specifies that *target* is the name of the target shared library file to be created by `mkshlib`, or, if the `-n` option is

	given, that *target* is the name of the target shared library file from which to build the host shared library.
`-h` *host*	Specifies that *host* is the name of the host shared library file to be created by `mkshlib`. The host shared library will not be created if this option is not given.
`-n`	Specifies that the target shared library shall not be created. In this case, the `-t` option is used to specify the pre-existing target shared library file from which to build the host shared library.
`-L` *dir* ...	Specifies the directories to be searched ahead of the default directories when searching for other host shared libraries given in an `#objects noload` directive. If this option is specified more than once on the command line, then the resulting list of directories will be searched in the order given before searching any of the default directories.
`-q`	Suppress (quiet) any warning messages.

CHAPTER 12 *Raw Ethernet Support*

Raw Ethernet Support is a facility available with LynxOS to communicate among nodes over an Ethernet/IEEE 802.3 data link from an application layer. This facility provides for a direct link between application and link layers and it transfers the burden of protocol-related processing to the application level. This implementation facilitates the following interface-specific functions:

- setting up the interface to allow raw ethernet reception
- setting up the station address
- setting up the protocol list
- setting up the multi-cast address list
- setting/resetting the promiscuous reception
- setting/resetting the interface in blocking mode
- enabling/disabling interface level debugging
- getting the interface statistics, current station address and factory address
- resetting the interface to stop raw ethernet reception
- filtering the incoming packets with respect to an ethernet header

Raw Ethernet Description

Raw Ethernet is supported using sockets of type SOCK_RAWETH in an exclusive domain, called AF_RAWETH. The raw Ethernet functions are implemented with a set of ioctl calls on a raw Ethernet socket, providing the interface between an application and the Ethernet interface driver.

Multiple sockets can be operating simultaneously in raw Ethernet mode. Many of the operations are restricted to use by a super-user and shall fail and return -1, if used otherwise. The write and read system calls may be used to send and receive Ethernet packets containing the Ethernet header and data fields. The following is the list of ioctl functions available to the user with which to set up the interface in raw Ethernet mode and communicate among nodes in Ethernet/IEEE 802.3 datalink.

- GE_SET_DEV
- GE_SET_SADDR
- GE_SET_PROTO
- GE_SET_MCAST
- GE_PROM
- GE_BLOCKING
- GE_DEBUG_MODE
- GE_GET_CTRS
- GE_GET_SADDR
- GE_GET_FADDR
- GE_CLRIFNAME
- GE_SET_FILTER

The Ethernet driver facilitates simultaneous reception of regular TCP/IP packets (ARP, RARP, and IP type) and processing, in case the Internet support exists. Facilities exist for exclusive reception of all packets through raw Ethernet sockets. The following sections explain each of the ioctl functions.

GE_SET_DEV

To use the interface in raw Ethernet mode, a socket needs to be created in the AF_RAWETH domain of the SOCK_RAWETH type. The returned socket descriptor can be used to set the device in raw Ethernet mode.

This ioctl function sets up the interface to accept raw Ethernet packets. Optionally, this socket can be made to receive TCP/IP packets, where regular TCP/IP processing is suspended until this socket gets closed for processing.

The arguments to this ioctl call, are

- interface name
- maximum queue length for the raw Ethernet interface
- flag indicating the reception of all packets to raw Ethernet sockets. (flag - RECV_ALL). If this flag is passed, TCP/IP packets are sent to the AF_RAWETH domain to see if a socket is waiting.

Example

```
typedef struct {
    int qlen;      /* max. Qsize for this interface */
    char ename[13];               /* interface name */
    int recv_all; ,/* flag indicating the receiving
                 of all packets to AF_RAWETH domain */
} T_GE_DEV;


int s, ret_val;
T_GE_DEV re_dev;
s = socket(AF_RAWETH, SOCK_RAWETH, 0);
re_dev.qlen = IFQ_MAXLEN; /* num packets to be cached
                                    IFQ_MAXLEN = 50 */
bcopy("wd0", re_dev.ename, sizeof("wd0"));
re_dev.recv_all = 0;         /* TCP/IP packets not be
                            received to this socket */
if ((ret_val = ioctl (s, GE_SET_DEV, &re_dev)) != 0) {
```

```
        perror("GE_SET_DEV:");
        exit(1);
    };
```

This socket descriptor is used for all the raw Ethernet `ioctl` functions.

NOTE: *This implementation does not support simultaneous processing of TCP/IP packets by TCP/IP code and by raw Ethernet sockets. As explained above, you can also receive TCP/IP packets to a raw Ethernet socket by setting other parameters such as the protocol range, filter pattern, or multi-cast address list. Normal TCP/IP processing is restored, when such a socket is properly closed (refer to the* `GE_CLRIFNAME` *ioctl call).*

Facilities exist for configuring the Ethernet driver and protocol processing code to get rid of TCP/IP processing entirely, such that any packet for this interface is received by a properly-matching raw Ethernet socket.

GE_SET_SADDR

This `ioctl` call is used to change the station address (Ethernet address) of this interface. The address must be of length 6 bytes and the bit-0 of first byte should be zero (as per the Ethernet/IEEE 802.3 specification, which defines the distinction between physical and multi-cast addresses depending on the bit-0 of the first byte of the node address). The newly-set address identifies this node and all packets with this address as the destination Ethernet address.

```
char saddr[6];
saddr[0] = 0xAA; saddr[1] = 00; saddr[2] = 04;
     saddr[3] = 00; saddr[4] = 04; saddr[5] = 00;

if ((ret_val = ioctl (s, GE_SET_SADDR, saddr)) != 0) {
    perror ("GE_SET_SADDR:");
    exit (1);
};
```

GE_SET_PROTO

This `ioctl` facilitates the setting up of the protocol range which the raw Ethernet socket needs to receive. As many as four protocol ranges can be received by each raw Ethernet socket.

```
typedef struct {
    int lowp;          /* lower bound protocol type */
    int highp;         /* upper bound protocol type */
} T_GE_LIST;

typedef struct {
    int num_items;           /* num entries in list */
    T_GE_LIST plist[GE_MAX_PROTO];/* GE_MAX_PROTO=4 */
} T_GE_PROTO;

T_GE_PROTO pr_list;
pr_list.num_items = 2;
pr_list.plist[0].lowp = 0x100;
pr_list.plist[0].highp = 0x125;
pr_list.plist[1].lowp = 0x150;
pr_list.plist[1].highp = 0x175;

if ((ret_val = ioctl (s, GE_SET_PROTO, &pr_list)) !=
0) {
    perror ("GE_SET_PROTO:");
    exit (1);
};
```

GE_SET_MCAST

It is possible to set up a list of multi-cast addresses which the interface receives as the destination node. Thus, data packets can be sent to a set of nodes, where a certain multi-cast address is acceptable. This address is used as the destination station address in such packets. As per the IEEE802.3 specifications, a multi-cast address is identified from the physical address by the `D0` bit of the first byte in the 6-byte station address. If this bit is set, it is considered as a multi-cast address. The interface needs to be programmed to accept a specific multi-cast address.

```
typedef struct {
    int num_items;       /* num entries in the list */
    char mlist[GE_MAX_MCAST][6];  /* GE_MAX_MCAST=3 */
} T_GE_MCAST;

T_GE_MCAST m_list;

m_list.num_items = 2;
m_list.mlist[0][0] = 0xCF;
m_list.mlist[0][1] = 0;
m_list.mlist[0][2] = 0;
m_list.mlist[0][3] = 0;
m_list.mlist[0][4] = 0;
m_list.mlist[0][5] = 0;
m_list.mlist[1][0] = 0xAB;
m_list.mlist[1][1] = 0;
m_list.mlist[1][2] = 0;
m_list.mlist[1][3] = 3;
m_list.mlist[1][4] = 0;
m_list.mlist[1][5] = 0;

if ((ret_val = ioctl (s, GE_SET_MCAST, &m_list)) != 0)
{
    perror ("GE_SET_MCAST:");
    exit (1);
};
```

GE_PROM

This `ioctl` allows the interface to receive all packets, whether or not all packets are destined for that node. Normally, this mode is not set, as it heavily loads the system.

```
int prom;
prom = 1;
if ((ret_val = ioctl (s, GE_PROM, &prom)) != 0) {
    perror ("GE_PROM:");
    exit (1);
};
```

GE_BLOCKING

This call puts the raw Ethernet socket in blocking mode (i.e. this call can be made to be blocked or return with error for lack of input in case of `read`, or for lack of output space in case of `write`). Note that the flag is set to `0` to put the socket in blocking mode.

```
int donot_block;
donot_block = 1;/* set interface in non-blocked mode */
if ((ret_val = ioctl (s,GE_BLOCKING,&donot_block)) != 0)
{
    perror ("GE_BLOCKING");
    exit (1);
};
```

GE_DEBUG_MODE

This `ioctl` call puts the interface in debug mode. The data which transfers through this interface is displayed in the debug terminal.

```
int debug;
debug = 1;
if ((ret_val = ioctl (s, GE_DEBUG_MODE, &debug)) != 0)
{
    perror ("GE_DEBUG_MODE:");
    exit (1);
};
```

GE_GET_CTRS

This `ioctl` call gets the raw Ethernet interface statistics.

```
typedef struct {
    long
        last_zeroed,  /* unused now */
        msg_rcv,      /* messages received */
        msg_xmt,      /* messages transmitted */
        byt_rcv,      /* bytes received */
        byt_xmt,      /* bytes transmitted */
        xmt_err,      /* transmit errors */
        msg_no_buff,  /* no buffer occurrences */
```

```
        msg_qfull;      /* no of qfull occurrences */
    int zero;           /* unused now */
} T_GE_CTRS;

T_GE_CTRS ctrs;
if ((ret_val = ioctl (s, GE_GET_CTRS, &ctrs)) != 0) {
    perror ("GE_GET_CTRS:");
    exit (1);
};
```

GE_GET_SADDR

This call gets the current station address of the raw Ethernet interface.

```
char saddr[6];
if ((ret_val = ioctl (s, GE_GET_SADDR, saddr)) != 0) {
    perror ("GE_GET_SADDR:");
    exit (1);
};
```

GE_GET_FADDR

This function would get the station address as assigned by the manufacturer for the Ethernet interface hardware.

```
char faddr[6];
if ((ret_val = ioctl (s, GE_GET_FADDR, faddr)) != 0) {
    perror ("GE_GET_FADDR:");
    exit (1);
};
```

GE_CLRIFNAME

This `ioctl` call closes the interface for raw Ethernet reception. It needs to be followed by the closing of the raw Ethernet socket. Note that the flag `RECV_ALL` needs to be set in the `recv_all` field, in case that option was chosen while opening this socket.

```
typedef struct {
    int qlen;      /* max. Qsize for this interface */
    char ename[13];                /* interface name */
    int recv_all;   /* flag indicating the receiving
                    all packets to AF_RAWETH domain */
} T_GE_DEV;

int s, ret_val;
T_GE_DEV re_dev;
re_dev.recv_all = 0; /* set to RECV_ALL if set while
                                    opening the i/f */

if ((ret_val = ioctl (s, GE_CLRIFNAME, &re_dev)) != 0)
{
    perror ("GE_CLRIFNAME");
    exit (1);
};
if (close (s) < 0) {
    perror ("close:");
    exit (1);
};
```

GE_SET_FILTER

This `ioctl` provides the filtering of the incoming packets with respect to the Ethernet header (first 14 bytes). For example, the received packets which fall within the protocol range set for this socket are passed through the filter and compared against a given mask. In case of a match, the packet is sent to the socket receive buffer. The argument for this `ioctl` contains two fields— `filter` and `mask`. The bits in the header that are of interest for this socket are set to `1` in the `filter` field. The bits that are expected to be set (logic `1`) in the incoming pattern are reset (set to `0`) in the `mask` field.

The filtering is released by logically `EX-NOR`ing the incoming packet header with the `filter` pattern. The resultant string is logically `OR`ed with the `mask` pattern. The output pattern is checked to see is all bits are set to `1`. If yes, that packet is received in the carbon-dating socket receive buffer.

By default, the filter is de-activated (i.e, there is no filtering on a received packet).

```
typedef struct {
    u_char filter[14];
    u_char mask[14];
} T_GE_FILTER;

T_GE_FILTER filter;
filter.filter[0] = 0xff;          /* Pass the whole */
filter.filter[1] = 0xff;       /* header (14 bytes) */
filter.filter[2] = 0xff;      /* through the filter. */
filter.filter[3] = 0xff;
filter.filter[4] = 0xff;
filter.filter[5] = 0xff;
filter.filter[6] = 0xff;
filter.filter[7] = 0xff;
filter.filter[8] = 0xff;
filter.filter[9] = 0xff;
filter.filter[10] = 0xff;
filter.filter[11] = 0xff;
filter.filter[12] = 0xff;
filter.filter[13] = 0xff;
filter.mask[0] = ~0xab;           /* Mask set for a */
filter.mask[1] = ~0;         /* certain pattern of */
filter.mask[2] = ~0;                /* destination */
filter.mask[3] = ~3;         /* address and source */
filter.mask[4] = ~0;                /* address but */
filter.mask[5] = ~0;           /* doesn't care for */
filter.mask[6] = ~0;            /* protocol field. */
filter.mask[7] = ~0x10;      /* Note that the bits */
filter.mask[8] = ~0xc0;       /* that are expected */
filter.mask[9] = ~0x40;             /* to be 1 are */
filter.mask[10] = ~0xab;      /* set to '0' in the */
filter.mask[11] = ~0x66;     /* mask and vise-versa.*/
filter.mask[12] = 0xff;         /* Do not care for */
filter.mask[13] = 0xff;         /* protocol field. */

if ((ret_val = ioctl (s,GE_SET_FILTER,&filter)) != 0) {
```

```
    perror ("GE_SET_FILTER");
    exit (1);
};
```

CHAPTER 13

Setting up a Printer

The LynxOS print spooling system is based on the BSD 4.3 print spooling system and installing a printer onto a LynxOS machine is similar to installing printers onto BSD-based machines. LynxOS supports local and remote printers, printers attached to serial or Centronics lines, and laser or impact printers.

Users that are familiar with the LynxOS print spooling system can skip to “Printer Quick Install” on page 218.

Before trying to install the printer, review the following pre-installation checklist and make sure that everything listed is available.

- The printer.
- The printer owners/operators manual.
- The appropriate cable (parallel, serial).
- Superuser privileges on the LynxOS machine.

The installation procedure for an ASCII printer is split into two basic steps.

1. *Physical installation of the printer.*
   The details of setting up a printer, either parallel or serial, are found in the documentation from the printer manufacturer. The manufacturer’s

documentation should be read completely and understood before the installation begins.

2. *Installation of the printer into the LynxOS print spooling system.*

   In addition to parallel and serial printer installation instructions, special installation instructions for a remote printer are also included in this chapter.

Whenever possible, examples will be used to illustrate a particular step.

Serial Printer Installation

The serial printer set up can be divided into communications set up and printer set up.

Some of the following communications set-up options are not configurable by all serial printers. An option that is not configurable must be hard wired to the correct setting. Printers that cannot be correctly set up cannot be used. Printer set-up options control things like automatic linefeed, character set, font, pitch, and form length. Refer to the manufacturer's documentation to find out what options are available and how to set them.

1. Enable DTE (Data Terminal Equipment) and disable DCE (Data Communications Equipment).
2. Disable RTS/CTS (Ready-To-Send/Clear-To-Send) and DSR (Data-Set-Ready).

   If RTS/CTS and DSR are required, then a special cable that jumps pins 4 and 5 together and 6, 8, and 20 together must be used.
3. Enable XON/XOFF Flow Control.
4. Set the baud rate to 9600.
5. Set the data word to 8-bit no parity.

Printer Cable

A null modem cable up to 50 feet in length can be used with a serial printer. A null modem cable is wired with lines 2 and 3 crossed and line 7 straight through. If the special cable is required to handle RTS/CTS and DSR signals, then on the printer end of the null modem cable jump pins 4 and 5 together and 6, 8, and 20 together.

Testing the Connection

After setting up and connecting the printer to the LynxOS machine, and before proceeding with the installation of the printer into the print spooling system, the printer should be tested.

The following procedure assumes that the printer is attached to the port `/dev/com2`. It will test and diagnose the printer connection.

1. Login to the LynxOS Machine as “root”.
2. Verify that the `/bin/login` program is not running on the `tty` port to which the printer is attached.

Enter the command:

```
ps -axon | grep com2
```

Login is running if this command returns a line that ends with `/bin/login`. To eliminate login first edit the `/etc/ttys` file and change the second field of the line that starts `/dev/com2` from a 1 to a 0. For example:

```
/dev/com2:1:default:wyse50:/bin/login
```

changes to

```
/dev/com2:0:default:wyse50:/bin/login
```

Then enter the command `kill -1 1`. This forces the `init` program to reread `/etc/ttys` and kill the login process running on COM2.

Here is a simple test. Enter the command:

```
(stty 9600; cat /etc/passwd) > /dev/com2
```

If the `/etc/passwd` file prints correctly, the printer setup and connection are fine. However, if the `/etc/passwd` file prints, but text is garbled, the `stty` settings may not be correct on the LynxOS machine or the printer. First double check the printer options.

Next, use the following parameters to `stty` to manually set up the LynxOS machine.

`parity`	To turn off parity
`tabs`	To turn on tab expansion
`onlcr`	To enable automatic carriage return/linefeed

For example, the command

```
(stty 9600 -tabs; cat /etc/passwd) > /dev/com2
```

will expand tabs to eight spaces.

If `/etc/passwd` fails to print and no message appears on the terminal then make sure that the cable is connected. The cable may be damaged or incorrectly wired.

Parallel Printer Installation

In general, the factory defaults are acceptable. MVME162 hardware does not provide parallel printer port support.

For the Panasonic KX-P1180 the basic configuration is controlled by DIP switches.

Table 13-1: Recommended Printer Settings for LynxOS

Switch #	Function	Recommended Setting	Switch Position
1	Printer Mode	Standard Mode	Off
2	Skip Perforation	No Skip	Off
3	Automatic LF	CR	Off
4	Cut Sheet Feeder	Not Installed	Off
5	7/8 Bit	8-Bit	Off
6	Character Set	USA	Off
7			Off
8			Off

Many parallel printers allow other characteristics to be configured. The Panasonic KX-P1180 allows the default font, pitch, and formlength to be configured through the front panel.

The default settings of programmable font, programmable pitch, and 11" formlength should be fine for LynxOS.

Printer Cable

If the printer did not come with a parallel printer cable, one can be purchased at a local electronics store. Check the manufacturer's documentation for the cable specifications. The cable specifications for the Panasonic KX-P1180 are: Shielded Centronics interface cable 2 meters or less in length. 57-30360 (AMPHENOL) or equivalent connector on the printer end.

Testing the Connection

After setting up and connecting the printer to the LynxOS machine, and before proceeding with the installation of the printer into the print spooling system, the printer should be tested. The following procedure will test and diagnose the printer connection.

1. Login to the LynxOS machine as `root`.
2. Enter the command `cat /etc/passwd > /dev/parallel`
3. If the `/etc/passwd` file prints correctly then the printer setup and connection are fine.
4. If the `/etc/passwd` file prints, but the text is garbled then the `stty` settings may not be correct on the LynxOS machine or the printer.
5. If the `/etc/passwd` file fails to print...
6. If the Message `/dev/parallel: protection violation` appears on the terminal, then write permission on `/dev/parallel` has probably been disabled. The command `chmod +w /dev/parallel` will enable write permission.
7. If the file still fails to print then the driver may not be installed. To check this, examine the file `/sys/lynx.os/CONFIG.TBL` and look for the string `I:parallel.cfg`. If it is preceded by a `#`, remove the # sign.

   Now follow the instructions in Chapter 13 to reconfigure the kernel. After the new kernel is built, enter the command `reboot -aN` to reboot LynxOS and rebuild the device nodes.
8. `/etc/passwd` fails to print

   If no message appears on the terminal, then check to make sure that the cable is connected securely and the printer is turned on.
9. Printer still does not print

   Contact Lynx Technical Support.

Print Spooler System Overview

The UNIX print spooling system allows multiple users from multiple machines to access a printer. Both local and remote requests are queued up on the disk to be printed in the order of their arrival. Once a print job has been queued for printing the user can continue on with other work.

Utilities are available to the user to add jobs to the queue, remove jobs from the queue and examine the queue. The actual printing is done by a daemon process that controls all of the printers and their queues.

Component Files

1. `/etc/printcap`

   The configuration information for a printer and its corresponding queue is found in the file `/etc/printcap`. For every local printer named in the file, the device that the printer is attached to and the spool directory where the print job is to be queued, are listed.

   Other information about a local printer that can be specified in the printcap file includes the baud rate and other `stty` information, a filter program to be invoked, the page width and page height, and the maximum file size.

   For a complete list, refer to the `printcap(5)` manpage in the *LynxOS Reference Manuals*. For a remote printer there is no device listed but a remote machine and remote printer name are listed as well as a local spool directory.

2. Spool Directory

   The spool directory specified in the printcap file is usually a sub-directory of the `/usr/spool` directory. The spool directory is where the print jobs are stored when they are queued to print. Information on the printer's status is stored here in the file status as well as the next sequence number to be used when a new print job is queued.

Component Programs

The component programs are all located in the /bin directory and are all owned by root. All perform a different function and all must be present to effectively print from a LynxOS system.

1. `lpd`

   The `lpd` program is the daemon process that manages all of the printers and their queues. Whenever a file is queued up for a printer `lpd` forks a child of itself to do the actual work of printing the file.

   The child process locks the printer's device, invokes any filter programs necessary, and sends the file to the printer. When the file is printed, it checks to see if there is another file to print. If there is nothing more to do, it terminates.

   If there are multiple printers `lpd` will fork multiple child processes allowing it to drive multiple printers at the same time.

2. `lpr`

   The `lpr` program starts the printing process by copying the file into the spool directory. It also passes any parameters along to `lpd` via a control file that is associated with every print job. If the printer requested does not exist in /etc/printcap, `lpr` will return an unknown printer message.

3. `lpq`

   The `lpq` program examines the queue of a specific printer reporting back the files in the queue, their sequence numbers and the status of the printer.

4. `lprm`

   The `lprm` program takes as parameters a printer name and a sequence number. It then checks the spool directory for the existence of a print job with the specified sequence number and removes **it.**

5. `lpc`

   The `lpc` program is an administrative utility that allows the system administrator to examine the print queues of any printer, enable or disable queueing for any printer, and stop or start printing for any printer.

Adding a Printer to a Spooling System

There are four simple steps to adding a printer to the print spooling system. They are:

1. Login to the LynxOS machine as `root`.
2. Update the `/etc/printcap` file.
3. Create the spool directory.
4. Kill and restart the line printer daemon `lpd`

Adding an Entry to /etc/printcap

The first step to adding any printer to the print spooling system is to add an entry for the printer to the `/etc/printcap` file. For a complete description of the syntax for this file, see the `printcap(5)` manual page.

A `printcap` entry starts with the printer's name in the first column followed by any aliases all separated by the | character. This first line is terminated with a `:\` with no white space before or after the `:\`.

The subsequent lines contain the printer configuration information. These lines must start with white space, normally a tab, followed by a colon. They are terminated like the first line. The last line of printer information does not terminate with a `\`.

For example:

```
# printcap entry for a local serial ASCII printer

lynxsptr|lp|Serial_Printer:\
        :lp=/dev/com2:\
        :sd=/usr/spool/lynxsptr:\
        :br#9600:fc#0177777:fs#06021:xc#0177777:\
        :sh:\
        :pw#80:pl#66:
```

Figure 13-1: Example Serial Printer Entry in /etc/printcap

The printer defined above is attached to `/dev/com2`, its spool directory is `/usr/spool/lynxsptr` running at 9600 baud, parity is turned off; automatic CRLF is turned on; tab expansion is enabled; XON/XOFF is enabled; the page width is 80 characters; and the page length is 66 lines.

Two-letter codes name the fields. Codes followed by = take a string as an argument, codes followed by # take a number as an argument. For a complete list of the codes, their meanings, default values, and acceptable argument values see `printcap` man page in the *LynxOS Reference Manuals*.

NOTE: *Two things to note from the above example. First, lines that begin with a # are comments. Second, more than one code and its argument may be on the same line as long as they are separated by a : (colon).*

Examples of serial, parallel, and remote printcap entries are found in the following paragraphs.

1. *Serial Printer* `/etc/printcap` *entry*

   The `/etc/printcap` entry for a serial printer must specify the baud rate of the printer and the stty settings in addition to the physical device and the spool directory of the printer.

   In the following example the baud rate is 9600 and the stty settings are parity disabled, automatic

`CRLF` enabled, tab expansion enabled and `XON`/`XOFF` flow control enabled.

The `lp` field specifies that the printer is attached to the serial port `/dev/com2`. The `sd` field specifies that the spool directory is `/usr/spool/lynxsptr`

2. *Parallel Printer* `/etc/printcap` *entry*

The `/etc/printcap` entry for a parallel printer is the most basic, only the physical device and the spool directory of the printer must be specified.

The `lp` field specifies that the printer is attached to the parallel port `/dev/parallel`. The `sd` field specifies the spool directory is `/usr/spool/lynxpptr`.

```
# printcap entry for a local parallel ASCII printer

lynxpptr|lpp|Parallel_Printer:\
       :lp=/dev/parallel:\
       :sd=/usr/spool/lynxpptr:\
       :pw#80:pl#66:
```

Figure 13-2: Example Parallel Printer Entry in /etc/printcap

3. Remote Printer **`/etc/printcap`** entry

The `/etc/printcap` entry for a remote printer does not have to specify the physical device. It must specify the local spool directory, the remote machine where the printer is attached and the name of the printer on the remote machine.

The `lp` field is left empty. The `sd` field specifies the local spool directory as `/usr/spool/lynxrptr`. The `rm` field is for the remote machine name and the `rp` field is for the remote printer name.

```
# printcap printcap entry for a remote printer
lynxrptr|Remote_Printer:\
       :lp=\
       :sd=/usr/spool/lynxrptr:\
       :rm=rem_mach:rp=rem_ptr:
```

Figure 13-3: Example Remote Printer Entry in /etc/printcap

Making the Spool Directory

The next step in installing the printer into the spooling system is to make the spool directory. It should be owned by root and have the permissions `drwxr-xr-x`. For the serial example the directory `/usr/spool/lynxsptr` must be created. This can be done with the following commands:

```
mkdir /usr/spool/lynxsptr

chown root /usr/spool/lynxsptr

chmod 755 /usr/spool/lynxsptr
```

Restarting lpd

The last step to installing the printer is to stop and restart `lpd`. Use the `ps -axon` command to find the PID (Process ID) for `lpd` and then use `kill` to stop it. To restart `lpd` just enter `lpd`.

Debugging the Installation

The following steps will help diagnose any problems with the installation of the printer into the print spooling system. This procedure assumes that the printer name is `lynxsptr`.

1. *Refer to sections and to check the connection.*
2. *Check file ownership and permissions.*

   Verify file ownership and permissions, changing where necessary, to match those below:

Table 13-2: File Permissions for Printer Setup

Permissions	Owner	File
`-rwxr-sr-x`	`root`	`/bin/lpc`
`-rwxr-xr-x`	`root`	`/bin/lpd`
`-rwsr-sr-x`	`root`	`/bin/lpq`
`-rwsr-sr-x`	`root`	`/bin/lpr`
`-rwsr-sr-x`	`root`	`/bin/lprm`
`drwxr-xr-x`	`root`	`/usr/spool/lynxsptr`

3. *Enter* `lpc stop lynxsptr` *to stop printing.*

   If `lpc` returns with the message `cannot open printer description file`, then the `/etc/printcap` file does not exist or cannot be opened.

   If `lpc` returns with the message `unknown printer lynxsptr`, then it cannot find the printer in the `/etc/printcap` file. Check `/etc/printcap` for typographical errors in the printer's name.
4. *Queue a file for printing.*

   Enter `lpr -Plynxsptr /etc/passwd` to queue a file for printing.
5. *Check the printer's queue.*

   Enter `lpq -Plynxsptr` to check the printer's queue. An entry for the file just queued should be displayed. If the queue is empty, recheck steps 1 through 6 and then call Lynx Real-Time Systems, Inc. technical support.
6. *Restart printing.*

   Enter `lpc start lynxsptr` to restart printing. If the file fails to print, check the `printcap` entry and make sure that the device name, baud rate, and `stty` setting are correct.

7. *Printing still fails.*

   If everything appears correct and printing still fails, call Lynx Real-Time Systems, Inc. technical support.

Printer Quick Install

1. *Set up and attach printer to a LynxOS machine.*

   Refer to printer manufacturer's documentation on how to perform printer set-up.

2. *Add* `printcap` *entry for the printer.*
3. *Make spool directory for the printer.*
4. *Kill and restart* `lpd`*.*

CHAPTER 14

X Server Advanced Topics (x86 and PowerPC)

The X server is the hardware and software interface to the Graphical User Interface (GUI). LynxOS supports a X server on x86 and PowerPC systems only. LynxOS for SPARC and LynxOS for 68k only support X clients.

This chapter describes some special features of the X server and provides more advanced configuration information. It assumes that you have already installed and initially configured the X and Motif Development Package on your system.

Special X Server Features

X Server Hot Key Exit

The X server supports a Hot Key Exit. This is useful for terminating the X server when the last client in the user's `.xinitrc` or `.xsession` does not provide a method of exiting the session. The Hot Key Exit key combination is Ctrl-Alt-Backspace.

Hot Key Resolution Switching

When configuring the X and Motif Development Package, you can select several screen resolutions. You can then change resolutions on the fly while running your X server by pressing the appropriate key combination. To switch to a higher resolution, press Ctrl-Alt-+. To switch to a lower resolution, press Ctrl-Alt-–.

Hardware Panning

The X and Motif Development Package supports hardware panning. This lets you set the virtual resolution higher than the terminal's physical resolution. Windows can then extend beyond the screen's physical boundaries and can be viewed by scrolling into the extended area.

Dual Servers

Several server options implement a twin-headed X server with which a single keyboard and mouse can be used to control up to four screens on separate monitors. This allows you to run more applications without overlapping windows, and is particularly useful for image-processing work.

In a multiple screen configuration there is one X process, one mouse, and one keyboard— but several screens. The mouse/cursor can be moved between the different screens. Clients connect to the different screens with the syntax

```
-display :0.X
```

where *X* is the screen number.

To get the dual server functionality, use one or more (up to four) Matrox MGA Millennium graphics adapters.

Touch-Screen Support

The X and Motif Development Package includes touch-screen support for some Elographics serial touch-screen controllers. The touch-screen controllers are simple devices that transmit absolute x and y coordinates to the serial driver when the screen is pressed. The X11 server reads these coordinates and converts them into X11 events, and treats the touch screen just like a one button mouse.

Touch Screen Calibration

All touch-screens need to be calibrated to match the display. Some touch-screens are pressure sensitive, so the calibration process uses variations in pressure to distinguish between a button click or a simple move. Since touch-screens vary, the calibration process must be set individually for each.

1. In `configX`, make sure the mouse entry displays the touch-screen controller you are using.
2. Save and exit `configX`.
3. Make sure X is *not* running.
4. Login as `root` and run the program:
   `/usr/bin/X11/touchcalibrate`
5. Follow the instructions printed to the screen to set the calibration. For example:

```
Configuring ELO Graphics 2300 Serial Touch Screen.

Touch the top left corner of the screen now.
Press Enter to continue.

Touch the bottom right corner of the screen now.
Press Enter to continue.

This touch screen is pressure sensitive.
This allows the X server to use the touch pressure to
distinguish between a button click and a simple move.

Touch the screen as if you were pressing a button.
Press Enter to continue.

Drag your finger lightly across the screen as you
would to move the cursor.

Calibration complete.
/usr/lib/X11/Metro/Xconfig has been updated.
```

Figure 14-1: Example Touch Screen Calibration

Dual Touch-Screens

Two touch-screens can be used simultaneously after executing the following additional steps shown below.

1. While in `configX`, set the mouse entry to display the touch-screen controller you are using as screen 0.
2. Save and exit `configX`.
3. Make sure X is *not* running.
4. Login as `root` and run the program:
   `/usr/bin/X11/touchcalibrate`
5. Follow the instructions printed to the screen to calibrate touch-screen 0.
6. Start X.

Networking and the X Server

The X11 graphics system uses a client/server connection model. The X server program controls the display of the graphics on the system's graphical display device. This program is configured for the appropriate graphics adapter and normally resides in the file `/usr/bin/X11/X`.The X server accepts connections and requests from other programs which are X clients. These X clients instruct the X server what to paint on the graphics display. The program `xterm` (the terminal emulator) is a sample X client. These X clients may connect to the server using a networking protocol such as TCP/IP, or with a faster local mechanism based upon UNIX domain sockets.

TCP/IP connections are used for connections between the client and server from different machines or systems. The UNIX Domain connections, on the other hand, are used for connections between the client and server from the same machine or system. The UNIX domain connection as a local connection is generally faster than a TCP/IP connection.

When the X server starts, it attempts to create a TCP/IP listening socket to accept connections from X clients. Then it tries to make a UNIX domain listening endpoint for local connections. If both connection attempts fail, the server aborts. Both listening end points may be active at the same time.

The X clients try to connect to the X server via the local UNIX Domain connection, as this is presumably faster. If the UNIX domain connection fails, the X clients then attempt to make a TCP/IP connection by calling the X server TCP/IP listening socket 6000.

Configuration Files

This section provides an overview of the configuration files used to configure the X server. This information is presented as a supplement to help you develop special configurations.

Xconfig

`Xconfig` is the primary configuration file for the X server. The server looks for this file in the following locations and in the following order:

```
$HOME/Xconfig
$HOME/.Xconfig
/usr/lib/X11/Metro/Xconfig
```

Most options are set with the `configX` program as described in the *LynxOS Installation Guide*. Some additional options can be added with an editor and `configX` will preserve these.

CAUTION! *If inappropriate changes are made to this file, X may not function properly!.*

Keywords

Descriptions of the `Xconfig` keywords are listed below. They are alphabetized here for your convenience and any order dependencies are noted within the individual entries.

Options Set In Configuration Utility

The following list of `Xconfig` keywords are options controlled via the `configX` configuration utility. Any changes to these options should be made through the utility to avoid accidental damage to the function of X.

BitsPerPixel <*number*>

Some X drivers support packed 24 bits per pixel memory organization. This means that each pixel takes up only three bytes of video memory. In this case, `configX` inserts the line:

```
BitsPerPixel 24
```

Buttons <*number*>

This specifies the number of buttons on the mouse.

This entry must immediately follow the `Mouse` *entry in the* `Xconfig` *file.*

Card <*string*>

This designates the model of the graphics card in use. This is only used by `configX`.

ChipSet <*string*>

This displays the chipset used by the server. Normally `configX` inserts the correct chipset into the `Xconfig` file.

Clocks <*number*> [<*number*>]

This contains a list of numbers specifying the frequencies associated with the frequency/pixel clock generator on the card. The numbers are specified in Mhz.

Depth <*number*>

This references the color depth of the screen, which controls how many colors can be displayed. Different cards and drivers support different depth values: 4, 8, 16, or 32. Depth 4 supports 16 colors and depth 32 supports 16-million colors.

Dimensions <*width*> <*height*>

This keyword represents the dimensions of the screen in millimeters, which is passed to X clients upon request.

DPMSoffDelay <*number*>

This controls the number of seconds after the last user input before the screen enters DPMS off mode. Off mode provides maximum power reduction. To activate this mode, the X server disables both the horizontal and vertical sync signals to the monitor.

DPMSstandbyDelay <*number*>

This controls the number of seconds after the last user input before the screen enters DPMS standby mode. Standby mode provides minimal power reduction. To activate this mode, the X server disables the horizontal sync signal to the monitor.

DPMSsuspendDelay <*number*>

This controls the number of seconds after the last user input before the screen enters DPMS suspend mode. Suspend mode provides significant power reduction. To activate this mode, the X server disables the vertical sync signal to the monitor.

Driver <*string*>

This references the name of the driver, and usually communicates to the server as to which modules need to be loaded. For example,

```
Driver 8514
```

initiates the loading of all modules relating to definition of the 8514 driver.

Emulate3Buttons

This entry is used when a two-button mouse is emulating a three-button mouse.

This entry must immediately follow the `Mouse` *entry in the* `Xconfig` *file.*

FontPath <*string*>

`FontPath` is followed by a comma-delimited string of font directories. A font directory is any directory which contains `fonts` and a `fonts.dir` file. The first entry in the `FontPath` is mandatory and must exist or the server will fail. If other entries in the list do not exist, they will be silently ignored.

HotKeyExit

This allows you to shut down the X server with the key sequence Ctrl-Alt-Backspace. If this keyword is not present, Ctrl-Alt-Backspace does nothing special.

KbdDesc <*string*>

The quoted string must match one of the entries in the `KeyboardInfo` file.

KbdDescFile <*string*>

The string describes the keyboard.

LogFile

Errors from the X server are logged here.

Modes <*string*> [<*string*>]

The entries following the `Modes` line match entries in the `Modes` section. If more than one mode is specified, the server can switch between modes. See “Modes” on page 233 for more details.

Monitor <*string*>

The quoted string must match one of the entries in the `MonitorInfo` file.

Mouse <*type*> [*device*]

This describes the type of mouse in use. `Type` is a keyword that is OS dependent. `Device` is usually required and is a file in the `/dev` directory.

MouseDesc *<string>*

The quoted string must match one of the entries in the `MouseInfo` file.

ScreenLayout *<screen> <left_neighbor> <right_neighbor> <top_neighbor> <bottom_neighbor> <left_edge> <right_edge> <top_edge> <bottom_edge>*

This defines how the mouse cursor behaves when it gets to the edge of the screen. There is one `ScreenLayout` entry for each configured screen.

CAUTION! *Any changes made to this entry will be lost if you run the* `configX` *utility at a later time.*

The `*_neighbor` fields tell which screen to move the mouse to. A value of `-1` indicates that the mouse should stop at that edge. The `*_edge` fields tell at which edge of the corresponding `*_neighbor` screen the mouse will appear. Screens are numbered from 0 to 4. Edges are numbered as follows:

Table 14-1: ScreenLayout Edge Numbering

Edges	Number
ENTER_LEFT	1
ENTER_RIGHT	2
ENTER_UP	3
ENTER_DOWN	4

ScreenLayoutIndex *<number>*

This tells which of the predefined screen layouts was chosen in `configX`.

Virtual <*width*> <*height*>

This designates the virtual width and height of the screen. The virtual width and height may be larger than the physical width and height if hardware panning is used.

Weight <*string*>

This designates the RGB color specification, which is dependent on the depth value. When the depth value is 16, the options are:

`R5:G5:B5 (default)` *or*
`R5:G6:B5`

When the depth value is 32, the options are:

`R8:G8:B8 (default)` *or*
`B8:G8:R8`

When the depth value is 4 or 8, the weight value does not apply.

XscreenSaverDelay

This controls the number of seconds (with no input) before the screen is blanked.

Options Not Set By configx

The following keywords control options that are not set by `configX` and require manual editing of the `Xconfig` file.

BackBuffer

This entry tells the X server to allocate off-screen memory for a BackBuffer.

CursorAdjustX
CursorAdjustY

If the hardware cursor does not position correctly over the X Window screen, `CursorAdjustX` and `CursorAdjustY` allow you to specify a delta (number of pixels) to adjust the cursor left or right.

CursorScreenOne

In a multiple screen configuration, this setting constrains the cursor to screen 0.

NoBackingStore

This turns off backing store. Backing store is a server feature that allows programs to create complex drawings and have the server automatically handle exposure requests, thus preventing lengthy re-draw cycles. Backing store uses system memory to hold the window contents and can actually slow down the system if there is limited memory.

NoCursorWrap

In a multiple screen configuration, this setting prevents the cursor from wrapping from screen 1 to screen 0.

NoFontCache

This turns off font caching.

NoKeyboard

This setting disallows keyboard input.

NoLEDs

No LED processing is done by the server.

NoPixmapCache

This turns off pixmap and stipple caching.

NotPrimaryMonitor

No console processing is done for Virtual Terminal Support. This is normally used when X is run on a graphics card that is not driving the console.

Option <*string*>

This specifies a special option passed to the server. It might specify the card model or a specific feature not typically used. Normally `configX` inserts the correct options in the `Xconfig` file.

`Option HardwarePex`

Add this line to the `Xconfig` file to use the Hardware Z Buffer on the Matrox MGA Impression Plus graphics card.

NOTE: *This file entry must immediately follow the* `Driver MGA` *entry*

`Option NoHardwareCursor`

If the cursor will not align properly with the X Window System, your X driver probably has a hardware cursor. If so, add this line to disable hardware cursor support..

NOTE: *This file entry must immediately follow the* `Driver` *entry*

`RemapCursorPad`

This configuration causes the X server to send back different keycodes for the keypad arrow keys and cursorpad arrow keys. This is a minor violation of the X protocol, but it allows clients to differentiate between these keys.

`SnfFormat` ***<number> <number> <number> <number>***

X X11.5 can read X11.4 SNF (Server Natural Format) fonts. Since there is no standard for SNF fonts, the following four numbers can be adjusted to read most SNF fonts. The four `number` parameters represent the following values, normally built into the server at compile time:

```
BITMAP_BIT_ORDER
IMAGE_BYTE_ORDER
GLYPHPADBYTES
BITMAP_SCANLINE_UNIT
```

For example, to read SCO SNF fonts, add this line to the `Xconfig` file:

```
SnfFormat 1 0 1 1
```

Or, to read SNF fonts compatible with X11 R4 from Interactive Systems Corporation, add this line to the `Xconfig` file:

```
SnfFormat 1 1 4 4
```

Terminate

`Terminate` causes the server to exit when the last client disconnects. Normally `xinit` kills the X server when the last client disconnects, before the server can reset.

TouchButton4

This entry specifies that a touch-screen is in use, rather than a mouse.

NOTE: *All touch-screen related entries in the* `Xconfig` *file must immediately follow the* `Mouse` *entry.*

TouchCoordinates *top left bottom right*

When touch-screens are used, the four numbers with this keyword are the top, right, bottom, and left coordinates returned by the controller. See “Touch-Screen Support” on page 221 for details.

TouchScreenNo *<number>*

This shows the screen number associated with the touch-screen.

TouchZThreshold *<number>*

This controls how hard the touch-screen must be pressed to generate a button event on touch-screens that report pressure. The number should be `-1` if the ELO 400x controller is jumpered for Moving Mode.

Trace *<number>*

This prints additional debugging information.

`VideoRam` ***<number>***

This displays the amount of video memory on the graphics card, specified in terms of 1024-byte units (with one megabyte equal to a value of 1024). The X server usually detects how much memory is on the card.

`VisualClass` ***<number>***

This uses the default `VisualClass` from `/usr/include/X11/X.h`, and is based on the following table:

Table 14-2: Visual Classes

Visual Class	Number
StaticGray	0
GrayScale	1
StaticColor	2
PseudoColor	3
TrueColor	4
DirectColor	5

On monochrome monitors the line

```
VisualClass 1
```

should be added to the `Xconfig` file.

Modes

The last section of the `Xconfig` file is the Mode database (`Modes`). It contains the VESA timing specifications for various resolutions. The default values work for most graphics cards, but they can be tweaked to center the image and adjust its size on the monitor.

The X server uses this database to configure most of the card for graphics mode. The numbers indicate which clock to use from the `DisplayConfig` file's `CLOCKS` line, and also show the horizontal and vertical timing values. More specifically, an example of the numbers are:

```
"1024x768"  65  1024  1092 1220 1344 768  786  791  810
```

The following table shows the definition of each `Modes` number from the prior example.

Table 14-3: Modes Definitions

		Horizontal				**Vertical**			
"1024X768"	**65**	**1024**	**1092**	**1220**	**1344**	**768**	**786**	**791**	**810**
Screen Resolution in Pixels	Pixel Clock Mhz	Pixels Displayed	Sync Start	Sync End	Total	Pixels Displayed	Sync Start	Sync End	Total

These numbers represent the ranges the electron beam traverses per one vertical refresh of the screen. They determine where the image is positioned on the screen, and what the refresh rate is.

Adjusting the Image Position

You can edit the `Modes` numbers if the screen image does not properly center or fill the monitor screen. Be sure to center the image before stretching or shrinking it.

Any adjustments you make will go into effect after restarting the X server.

Centering the Image

The `Pixels Displayed` numbers must match the desired physical screen resolution. The `Sync Start` and `Sync End` numbers must be increased or decreased (to move left or right; up or down) at the same time and by the same amount, and stay within the range of the `Displayed` and `Total` values.

- Image too far right

  To adjust an image that is too far to the right, increase the `Horizontal Sync Start` and `Horizontal Sync End` values simultaneously.
- Image too far left

  To adjust an image that is too far to the left, decrease the `Horizontal Sync Start` and `Horizontal Sync End` values simultaneously.

For example, to adjust an image that is too far to the left, you could begin the process by adding 8 to the two horizontal timing numbers (`Sync Start` and `Sync End`). Make sure that the `Sync End` number does not exceed the `Horizontal Total` number.

The same technique applies for vertical positioning:

- Image too low

  To adjust an image that is too far to the bottom, decrease the `Vertical Sync Start` and `Vertical Sync End` values simultaneously.
- Image too high

  To adjust an image that is too far to the top, increase the `Vertical Sync Start` and `Vertical Sync End` values simultaneously.

For example, to adjust an image that is too high, you could begin the process by adding 8 to the two vertical timing numbers (`Sync Start` and `Sync End`). Make sure that the `Sync End` number does not exceed the `Vertical Total` number.

- Stretching or Shrinking the Image

  The `Pixels Displayed` numbers must match the desired physical screen resolution.The `Total` value is increased or decreased to stretch or shrink the image. In addition, to hold the center position, the `Sync Start` and `Sync End` numbers must be increased or decreased equally, based on one-half of the difference in the changed `Total` value. For instance, if you added 10 to the `Horizontal Total` value, you would also add 5 to the `Horizontal Sync Start` value and 5 to the `Horizontal Sync End` value.

- Image too narrow horizontally

  To stretch an image that is too narrow horizontally (extra space appears on the left and right), increase the `Horizontal Total` by amount *x*, then add one-half of *x* to both the `Horizontal Sync Start` and `Horizontal Sync End` values.

- Image too narrow vertically

  To stretch an image that is too narrow vertically (extra space appears on the top and bottom), increase the `Vertical Total` by amount *x*, then add one-half of *x* to both the `Vertical Sync Start` and `Vertical Sync End` values.

- Image too wide horizontally

  To shrink an image that is too wide horizontally (the image extends beyond the left and right edges), decrease the `Horizontal Total` by amount *x*, then subtract one-half of *x* from both the `Horizontal Sync Start` and `Horizontal Sync End` values.

- Image too tall vertically

  To shrink an image that is too tall vertically (the image extends beyond the top and bottom edges), decrease the `Vertical Total` by amount *x*, then subtract one-half of *x* from both the `Vertical Sync Start` and `Vertical Sync End` values.

Extensions

Extensions are configured in the `/usr/lib/X11/Metro/Extensions` file. Most extensions are dynamically loaded; that is, if they are needed they are loaded automatically without any additional manual configuration.

An extension that is “ON” is loaded at server start-up and never unloaded. An extension that is “DYN” is dynamically loaded on request by an X client `XQueryExtension` call. These extensions display with `XListExtensions` or `xdpyinfo`. An extension that is “OFF” cannot be used, nor is it displayed with `XListExtensions` or `xdpyinfo`.

Special Notes

`X3D-PEX` and `DOUBLE-BUFFER` allocate a window private, which means they can’t be dynamically loaded. In addition, if they are turned “ON”, they will use extra memory (four bytes) per every X window regardless of whether or not that extension is being used. A later release of PEX will not have this restriction.

APPENDIX A

x86 Device Defaults

The following is a list of devices with their interrupt (also called IRQ) levels, I/O addresses and additional default information. No two devices with the same IRQ can be configured into the LynxOS kernel at the same time. Dashes (-) in the table mean "not applicable."

Table A-1: Default Device Configuration on the x86

IRQ[1]	IOBASE[2]	DMA Channel[3]	Device	Comments
0	-	-	`timer`	

1	`0x3B4`
	`0x3D4`
	`0x3C4`
	`0x3C5`
	`0x3CE`
	`0x3CF`
			`atc`	Keyboard

2	-	-		Cascade
3	`0x2F8`	-	`com2`	

Table A-1: Default Device Configuration on the x86 (Continued)

IRQ[1]	IOBASE[2]	DMA Channel[3]	Device	Comments
3	–	–	atcinfo	
4	0x3F8	–	com1	
5	0x3E8	–	com3	
5	0x240	–	wd3e	
5	0x3F0	–	if_3c579	Slot number 7 TP = 0, AUI = 1, BNC = 3 (default 3)
5	0x240	–	if_3c509	Slot number 0 TP = 0, AUI = 1, BNC = 3 (default 3)
5	0x240	–	if_wd3e	On board RAM address[4]: 0xcc000 16 bit access 1
6	0x3F2– 0x3F7	–	flopinfo	8237 DMA controller
7	0x378	–	ptrinfo	
7	0x200	–	pcxe	On board RAMbase: 0xd0000
8		–	rtclock	Real-time clock
9		–	atc	Vertical sync. interrupt
9	0x3E0	–	com4	

Table A-1: Default Device Configuration on the x86 (Continued)

IRQ[1]	IOBASE[2]	DMA Channel[3]	Device	Comments
10				Unused by default.
11	0x330- 0x332	5	a_scsiinfo.c sim1542_info	bus_on[5] : 8 bus_off[6] : 40
11	0x330[7]	5	sim_1742_info	Edge/Level[8]: edge(1)
11	-	5	sim2742_info	EISA slot[9]: 2
12				Unused by default.
13				Unused by default.
14	0x1F1- 0x1F7, 0x3F6- 0x3F7		hdinfo	Defaults are hard coded in driver; primary controller.
15				Unused by default.
-	0x340	-	sim1522_info	PIO mode only
10	-	-	sim2940_info	No user configurable options.

1. IRQ: interrupt used by the device.
2. IOBASE: 1 or memory locations that are used to communicate with the device.
3. DMA channel: DMA channel used by the device
4. On board RAM address: Some boards have memory mapped to a particular range in the I/O space. This number is the start of such memory. The size of memory varies from card to card.
5. bus_on- micro-seconds a device is allowed to stay on the bus.
6. bus_off- micro-seconds a device stay off after being on the bus.
7. Only applies in 1542-mode.

8. Edge/Level- A type of interrupt; Lynx does not support level interrupts.
9. EISA slot- A specific slot that a card must be in for the device to work.
10. PCI card; the interrupt is assigned by the system BIOS.

APPENDIX B

PowerPC Device Defaults

The following is a list of devices with their interrupt (also called IRQ) levels, I/O addresses and additional default information. No two devices with the same IRQ can be configured into the LynxOS kernel at the same time.

Table B-1: Default Device Configuration on PowerPC

IRQ	MVME 1600[1]	Motorola Ultra[2]	Motorola Series E[3]
0	Timer 1/ Counter 0	Timer 1/ Counter 0	Timer 1/ Counter 0
1	Keyboard	Keyboard	Keyboard
2	Cascade	Cascade	Cascade
3	COM2	COM2	COM2
4	COM1	COM1	COM1
5	(unused)	ISA Slots	(unused)
6	Floppy DMA channel 2	Floppy DMA channel 2	Floppy DMA channel 2
7	Parallel Port	Parallel Port	Parallel Port
8	(unused)	(unused)	(unused)
9	Z85230 ESCC	Ethernet, PCI Slot	Z85230 ESCC

Table B-1: Default Device Configuration on PowerPC (Continued)

IRQ	MVME 1600[1]	Motorola Ultra[2]	Motorola Series E[3]
10	Ethernet DEC 21040	(unused)	Ethernet DEC 21040
11	VME2PCI	PCI Slot	(unused)
12	(unused)	(unused)	(unused)
13	(unused)	(unused)	(unused)
14	SCSI	SCSI	SCSI
15	(unused)	(unused)	(unused)

1. Motorola MVME1600 VME Boards
2. Motorola Ultra Boards and Ultra-based PowerStack RISC PC Systems
3. Motorola PowerStack Series E Systems

APPENDIX C

68k Device Defaults

The following table lists the default interrupt vectors and associated IRQ on the 68k platform.

Table C-1: Default Device Configuration on 68k

IRQ	Interrupt Exception Source	Vector #	Comments
3	TMR2_IRQ	72	Timer
3	TMR1-IRQ	73	Timer
4	HWK_VECTOR	112	Hawk Ethernet
5	SCC_VCC	80	Serial I/O
5	SCSI_PRT_IV	69	SCSI
6	LANC_ERR	70	Lance error
6	LANC_IRQ	71	On-board lance
7	PARITY_ERR	75	DRAM parity error
x	ABORT_IRQ	78	Abort switch

APPENDIX D

SPARC Device Defaults

The following table lists the default interrupt vectors and associated IRQ levels on the SPARC platform.

Table D-1: Default Device Configuration on SPARC

Level	Trap Number	Physical Address	SBus IRQ	Comment
1	17			
2	18		1	
3	19	7C800000	2	Parallel Port (MACIO)
4	20	78800000		SCSI (MACIO)
5	21		3	
6	22	78C00000		Ethernet (MACIO)
7	23		4	
8	24			
9	25		5	
10	26	71D00000		System Timer (SLAVIO)
11	27	71400000	6	Floppy (SLAVIO)

Table D-1: Default Device Configuration on SPARC(Continued)

Level	Trap Number	Physical Address	SBus IRQ	Comment
12	28	71000000		Serial, Keyboard, Mouse (SLAVIO)
13	29		7	
14	30	71D00000		Auxiliary Timer (SLAVIO)
15	31			Async. Hardware errors

Index

O

P

R

S

T

U