LynxOS Networking Guide

LynxOS 4.0

DOC-0402-00

Contents

Preface

The *LynxOS Networking Guide* contains information about configuring LynxOS network components.

This manual assumes that users have a basic understanding of UNIX and is intended primarily for system administrators, network administrators, developers, and end-users of LynxOS. Many tasks in this manual include system administration and configuration tasks that require `root` privileges.

For More Information

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

- *Release Notes*

  This printed document contains details on the features and late-breaking information about the current release.

- *LynxOS User's Guide*

  This manual details system administration concepts, building custom LynxOS kernels, and additional features available in LynxOS.

- *LynxOS Networking Guide*

  This guide contains configuration and usage information on the networking capabilities in LynxOS. It provides information on supported protocols such as TCP/IP, NFS, DHCP, etc.

- *Writing Device Drivers*

  This guide contains details on writing device drivers for LynxOS.

- *GNU Zebra User's Guide*

  Contains information about configuring, and using GNU Zebra for LynxOS. This book is available online only. See the LynxOS Documentation CD-ROM.

- Online information

  The complete LynxOS documentation set is available on the Documentation CD-ROM. Books are provided in both HTML and PDF formats.

  Updates to these documents are available online at the LynuxWorks website: `http://www.lynuxworks.com`.

  Additional information about commands and utilities is provided online with the **`man`** command. For example, to find information about the GNU gcc compiler, use the following syntax:

  ```
  man gcc
  ```

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to filenames and commands are case sensitive and should be typed accurately.

Kind of Text	Examples
Body text; *italicized* for emphasis, new terms, and book titles	Refer to the *LynxOS User's Guide*.

Environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data
Commands that need to be highlighted within body text, or commands that must be typed as is by the user are **`bolded`**.
	`ls`
	`-l`
	`myprog.c`
	`/dev/null`
	`login:` **`myname`**
	`#` **`cd /usr/home`**

Text that represents a variable, such as a file name or a value that must be entered by the user	`cat` *`filename`*
	`mv` *`file1 file2`*

Kind of Text	Examples
Blocks of text that appear on the display screen after entering instructions or commands	`Loading file /tftpboot/shell.kdi into 0x4000 ..................... File loaded. Size is 1314816 Copyright 2000 LynuxWorks, Inc. All rights reserved. LynxOS (ppc) created Mon Jul 17 17:50:22 GMT 2000 user name:`
Keyboard options, button names, and menu sequences	**Enter**, **Ctrl-C**

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.

CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

LynuxWorks Technical Support is available Monday through Friday (holidays excluded) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The LynuxWorks World Wide Web home page provides additional information about our products, Frequently Asked Questions (FAQs), and LynuxWorks news groups.

LynuxWorks U.S. Headquarters

Internet: support@lnxw.com
Phone: (408) 979-3940
Fax: (408) 979-3945

LynuxWorks Europe

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

World Wide Web

http://www.lynuxworks.com

CHAPTER 1

TCP/IP

LynxOS supports TCP/IP (Transmission Control Protocol/Internet Protocol) networks. By default, TCP/IP is installed during the initial installation of LynxOS. TCP/IP can be configured, installed, or removed at any time.

LynxOS TCP/IP is an enhanced version of the FreeBSD 4.2 TCP/IP stack, and includes support for the BSD socket interface system and networking library functions needed to access the TCP/IP and UDP network protocols. The LynxOS TCP/IP stack is enhanced for real-time determinism and performance.

Installing/Removing TCP/IP Support

TCP/IP can be installed or removed after initial installation of LynxOS. Scripts are provided to install or remove TCP/IP support. The `Install.tcpip` and `Uninstall.tcpip` scripts are found in `/usr/bin`.

Installing TCP/IP

To install TCP/IP:

```
# Install.tcpip
```

To remove TCP/IP:

```
# Uninstall.tcpip
```

NOTE: These instructions are for IPv4 only. The IPv6 (and IPsec) protocols for LynxOS are not included with the standard LynxOS package. These components are available for purchase separately. For information on these products, please contact your LynuxWorks sales representative.

Configuring Ethernet Cards with ifconfig

`ifconfig` is used to assign an address to a network interface for each interface present on the system. (This is handled automatically if LynxOS is installed with network support.) `ifconfig` can be issued as a command to reconfigure the network interface or to obtain configuration information.

To use `ifconfig` to configure a network interface, the command should be placed in `/net/rc.network` (see `rc.network` below for more information on this file) in order for the network interface to be properly configured at boot time.

Issuing an `ifconfig` command without parameters displays the current configuration information of the network interface. Refer to the `ifconfig` man page for more information.

rc.network

The `rc.network` file contains command scripts that configure the network interface card and start other network related services. `rc.network` contains `ifconfig` entries for all network interface drivers supplied by LynxOS. To use one of the entries, enable it by removing the comment character and supply the appropriate host name and IP address.

To use a network interface driver not supplied with LynxOS, insert an `ifconfig` entry into `rc.config` specifying device driver file name, host name, and IP address.

Use `rc.network` also to enable or disable network services such as NFS and SNMP for example.

Common TCP/IP Utilities

The following table lists common LynxOS TCP/IP commands and files:

Table 1-1: LynxOS TCP/IP Components

Component	Definition
`ping`	Sends a packet of data to a system to determine if it is on the network. The `ping` command is used to test that TCP/IP is installed correctly and that a system is up on a network.
`hosts`	The hosts database. This can be the `/etc/hosts` file, the Network Information Service (NIS) hosts map, the Internet domain name server, or any combination of these.
`rlogin`	Allows the user to log onto a remote host on the network. This requires that the user name be the same on both the local and remote machine. If the `/etc/hosts.equiv` file is set up, a password is not needed when performing a remote log on. `rlogin` requires the host computer to have a UNIX-compatible operating system.
`/etc/hosts.equiv`	This is the file that contains the list of acceptable remote hosts.
`/etc/hosts`	The file that contains the list of hosts on the network.
`/etc/resolv.conf`	The `resolv.conf` file is used to determine a system's domain name, domain search paths, and IP addresses for name servers and routers.
`telnet`	A protocol that allows for remote access to a system. The system can run any operating system, UNIX-compatible or not.
`.rhosts`	A file that provides the remote authentication database for the `rlogin`, `rsh`, and `rcp` commands.
`rsh`	Allows users to connect and execute commands on a remote host. This command expects that the `/etc/hosts.equiv` and `.rhosts` files are configured properly. The `rsh` command only works when users are considered equivalent on the local and remote machine.

Table 1-1: LynxOS TCP/IP Components (Continued)

Component	Definition
`rcp`	A command that copies files and directories between different hosts. The remote copy command or `rcp` is a fast and efficient way to exchange data quickly between UNIX-compatible hosts. The `/etc/hosts.equiv` and `.rhosts` files must be correctly configured to use this command. Note that `rcp` does not copy symbolic links.
`ftp`	A file transfer program that uses the File Transfer Protocol. The `ftp` program transfers files to and from a remote network site. Passwords are required to access user directories.
`ifconfig`	Allows users to view and configure TCP/IP network interface parameters. The `ifconfig` command is not hardware dependent.
`netstat`	A command that lets users know the status of the network. It displays the contents of various network-related data structures.
`tcpdump`	Displays TCP/IP activity on a particular interface.

For more information on these components, see the appropriate `man` pages.

The next sections provide an overview of some of the most common TCP/IP utilities. These sections introduce basics of common TCP/IP utilities such as the `ping` command, remote computer access (`telnet`, `rlogin`) and file transfer between computers (`ftp`).

Testing TCP/IP (ping)

The simplest way to test the TCP/IP configuration of the system is to use the `ping` utility. Users can send a test message to any host on the network with the `ping` command. Users can ping either the IP address or host name of the machine. This test verifies the correct operation of hardware and TCP/IP software connecting the hosts. The `ping` command continues to send packets to the addressed host once every second until the command is terminated with a **Ctrl-C**.

The following figure shows the `ping` command testing TCP/IP configuration by sending data packets to the IP address of a system:

```
$ ping 192.168.1.102
PING 192.168.1.102: 56 data bytes
64 bytes from 192.168.1.102: icmp_seq=0.time=10. ms
64 bytes from 192.168.1.102: icmp_seq=1.time=0. ms
64 bytes from 192.168.1.102: icmp_seq=2.time=0. ms
64 bytes from 192.168.1.102: icmp_seq=3.time=0. ms
^C
---- 192.168.1.102 PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/2/10
$
```

Figure 1-1: Testing TCP/IP with ping

The following figure shows the `ping` command testing TCP/IP configuration by sending data packets to the hostname of a system.

```
$ ping shark
PING shark (192.168.1.101): 56 data bytes
64 bytes from 192.168.1.101: icmp_seq=0.time=10. ms
64 bytes from 192.168.1.101: icmp_seq=1.time=0. ms
64 bytes from 192.168.1.101: icmp_seq=2.time=0. ms
64 bytes from 192.168.1.101: 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 1-2: Using ping to test TCP/IP

Once the correct host setup is verified, every host on the network can use the `ping` command on other members of the network as shown in the following figure.

```
$ ping orca
PING orca (192.168.1.102): 56 data bytes
64 bytes from 192.168.1.102: icmp_seq=0.time=10. ms
64 bytes from 192.168.1.102: icmp_seq=1.time=0. ms
64 bytes from 192.168.1.102: icmp_seq=2.time=0. ms
64 bytes from 192.168.1.102: 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 1-3: Pinging Other Hosts on a Network

Troubleshooting ping

Problems related to testing TCP/IP configurations with **`ping`** sometimes occur due to host lookup failures, or problems in connectivity between the systems, not necessarily with the TCP/IP configuration on the local system.

The /etc/hosts File

Host lookup failures with `ping` can sometimes be attributed to incorrect `/etc/hosts`, or `/etc/resolv.conf` files. For example, if the host named `fish` in the `/etc/hosts` file is not defined, the `ping` command fails as shown in the following figure.

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

Figure 1-4: Host Not Defined in /etc/hosts

Because the hostname `fish` is not defined on the local system, `ping` returns a host name lookup failure. The `/etc/hosts` file provides a means of mapping IP addresses to hostnames. However, in larger networks, a Domain Name Service (DNS) server is typically used. The DNS server maintains a database of hostnames and IP addresses. If a user pings a system that is not defined in a local `/etc/hosts` file, the system then sends a request to a DNS server to translate the hostname to the IP address.

However, a preferred method to test TCP/IP functionality is to first ping the IP address of a system, then attempt to ping the hostname of a system. If the IP address of a host is found, but the host name lookup fails, TCP/IP is correctly configured, but the hostname resolution needs to be corrected. To resolve the hostname lookup failure, the correct IP address and hostname must be added to the `/etc/hosts` file.

The following table shows an example `/etc/hosts` configuration file:

```
#loopback address
127.0.0.1 localhost

#localhost address
192.168.1.103 stingray

#other host addresses
192.168.1.101 shark
192.168.1.102 orca
```

Figure 1-5: /etc/hosts example file

In this example, the localhost entry is called a “loopback” address, and is used to point to the local system. An entry also exists for the IP address and hostname of the local system. Any other IP address and hostname definitions point to other systems on the network.

The /etc/resolv.conf file

In addition to the `/etc/hosts` file, the `/etc/resolv.conf` file provides the domain name for the local system, domain search paths used when looking up hosts, and IP addresses for Domain Name Service (DNS) servers.

Users can **`ping`** a host without providing a domain name by entering the following command:

```
# ping fish
```

If `fish` is not defined in `/etc/hosts`, the local system uses the search paths and DNS servers provided in `/etc/resolv.conf` to determine a fully-qualified domain name and IP address. The system uses the `search` entries in the `resolv.conf` file to determine a fully qualified domain name, for example, `fish.domain1.com`. For the system to find the IP address of a host, it must have access to one or more DNS servers. These DNS servers contain indexes of fully qualified domain names and valid IP addresses.

The structure of the `/etc/resolv.conf` file is as follows:

```
domain domain1.com

search domain1.com
search domain2.com

nameserver 192.168.1.254
nameserver 192.168.1.253
```

Figure 1-6: /etc/resolv.conf example file:

In this example, the `domain` definition provides the domain of the local system. If the hostname is `stingray`, for example, the fully qualified hostname would be `stingray.domain1.com`.

The `search` definitions provide a means to resolve the fully-qualified domain name for hosts. For example, a system searching for the host `fish` first attempts to resolve the fully qualified domain name to the first search entry, or `fish.domain1.com` by sending a request to the local DNS server. If no entry exists in the DNS table for the system `fish.domain1.com`, the local system resolves the fully qualified domain name to the second search entry in `resolv.conf`, or `fish.domain2.com`. A second request is sent to the DNS server for the IP address of `fish.domain2.com`. This process continues until a valid host and domain is found or there are no more search paths. There is no limit to the number of search paths that can be used in `/etc/resolv.conf`.

`nameserver` definitions in `/etc/resolv.conf` are IP addresses pointing to local DNS servers. These DNS servers are used to translate the fully qualified domain name to a valid IP address.

ping Not Responding

If the **`ping`** command fails to respond with any output, terminate the program by pressing **Ctrl-C**. Common reasons for `ping` failure include:

- The machine `orca` is not connected to the network.
- The machine `orca` is down or powered off.
- The `/etc/hosts` file on `shark` has the wrong Internet address for `orca`.
- TCP/IP is not properly configured for `orca`.

Using traceroute

The `traceroute` utility is used to follow the route an IP packet takes to reach its destination. By sending simple UDP probe packets, `traceroute` displays the names and response times of the different gateways a packet traverses before reaching its destination. The traceroute syntax is as follows:

```
# traceroute <host>
```

Where `<host>` is the hostname or IP address of the destination system. Additional usage information is available in the `traceroute` man page.

Additionally, `traceroute6` can be used to follow the route of an IP packet through IPv6 networks.

Logging On to a Remote Computer (telnet, rlogin)

Users can log on to another host on the network using either one of the two utilities supplied with TCP/IP:

- `telnet`
- `rlogin`

telnet

The **`telnet`** utility allows users to log on 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 system from anywhere on the network.

`telnet` is invoked with the host name of a remote computer, as shown in the following figure.

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

LynxOS (shark)

user name: jones
password:

jones@shark$
```

Figure 1-7: Using telnet to Remotely Log in

To access the system, users must supply a user name and password.

Terminate a `telnet` session by logging out of the system with the `exit` command, as shown in the following figure.

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

jones@orca$
```

Figure 1-8: Terminating a telnet Session

rlogin

Users can use **`rlogin`** to remotely log on another computer similar to `telnet`. Unlike `telnet`, `rlogin` requires the host computer to have a UNIX-compatible operating system.

`rlogin` is invoked with the host name of the remote computer, as shown in the figure below:

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

jones@shark$
```

Figure 1-9: Using rlogin

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

If the user wants to log on to a system that does not have an identical user account, the login argument, `-l` followed by the desired user account must be added, as shown in the following figure.

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

doc@shark$
```

Figure 1-10: Remote Log in as Another User

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

Executing Commands Remotely (rsh)

Another utility included with TCP/IP is **`rsh`**, the remote shell command. This utility allows users to perform the following tasks:

- Access remote hosts and redirect output to the local machine.
- Execute a command on that host.

Accessing Remote Hosts and Redirecting Output to the Local Machine

The **`rsh`** command only works when the user is considered equivalent on the local and remote machine.

The syntax for the `rsh` command is as follows:

```
rsh host [-l <user_name>] <command>
```

An optional user name can be given to execute the command as a specific user. This is useful if the 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.

The `who` utility displays users who are currently logged into a remote host, as shown in the following figure.

```
davis@shark$ rsh orca who
rootatc0 Mon Dec 23 10:40:54
roottyp0:0.0Mon Dec 23 10:41:19
roottyp1:0.0Mon Dec 23 10:49:02
roottyp2:0.0Mon Dec 23 11:05:45
davis@shark$
```

Figure 1-11: Using who to Query Log ins on a Remote System

Executing a Command on a Remote Host

Also, commands can be invoked on a remote host as another user. In the following figure, user `jones` on host `orca` invokes the `whoami` command. This utility reports the current `login` account.

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

davis@shark$
```

Figure 1-12: Using rsh to Remotely Execute a Utility

Transferring Files Between Machines (ftp, tftp rcp)

There are several ways to copy files between hosts:

- `ftp` for hosts of differing operating systems.
- `rcp` for UNIX compatible operating systems.

File Transfer Protocol (ftp)

The File Transfer Protocol, or **`ftp`** allows users numerous configuration options. In addition to the options provided in this document, review the `ftp man` page for setting advanced options.

Starting ftp

In its simplest invocation, **`ftp`** is called with the host name of the remote machine. `ftp` prompts for a user login and password. A password must be provided to access user accounts.

In the following figure user `davis` on host `orca` connects to host `shark` and logs on 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 1-13: Connecting to a System with ftp

Retrieving Files from a Remote Host (get)

Once logged in, files can be retrieved from the remote host using the **`get`** command, as shown in the following figure.

```
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 1-14: Downloading Files with ftp

Sending Files to a Remote Host (put)

Alternatively, files can be sent to the remote host using the `put` command, as shown in the following figure.

```
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 1-15: Uploading Files with ftp

NOTE: By default, the **`ftp`** transfer program operates in an ASCII text mode. Set the transfer mode to binary by entering `binary` at the `ftp` prompt.

Transferring Binary Files

To transfer binary files, the transfer mode must be changed by entering the `binary` command at the `ftp` prompt. For example:

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

Transferring a binary file in ASCII mode results in a corrupt file. To preserve the integrity of the file, be sure to set FTP to binary mode before the transfer. Note that Binary file transfers are a little slower than ASCII file transfers.

Listing ftp Commands

`ftp` commands are displayed by entering a question mark (**?**) at the `ftp` prompt. The following list describes some common `ftp` commands

Table 1-2: Common ftp Commands

Command	Description
`ascii`	Sets file transfer type to ASCII.
`binary`	Sets file transfer type to binary.
`bye`	Terminates the `ftp` session and exits.
`cd`	Changes directory on the remote system.

Table 1-2: Common ftp Commands (Continued)

Command	Description
`delete`	Deletes file on the remote system.
`get`	Retrieves remote files to the local system
`help`	Displays a list of `ftp` commands. If a command is provided as an argument, displays specific command information.
`lcd`	Changes working directory on the local system.
`ls`	Lists contents of remote directory
`mdir`	Lists contents of multiple remote directories.
`mget`	Retrieves multiple files.
`mkdir`	Makes directory on the remote system.
`mput`	Sends multiple files.
`put`	Sends one file.
`pwd`	Prints remote system working directory.

Trivial File Transfer Protocol (tftp)

`tftp` (Trivial File Transfer Protocol) is a simple UDP transfer protocol. Typically, `tftp` is used for bootstrapping diskless clients and installing firmware into ROM.

`tftp` does not require client authentication, which may pose a security risk for some systems. LynxOS supports two `tftp` transfer methods: Simple and Secure.

- **Simple**: The client can access the entire file system. This is a simpler configuration, but it presents a larger security hole (anyone can access the password file). In Simple mode, only files open to the public can be read.

  Simple is the default tftp mode. No options are required:

  ```
  # tftpd
  ```

- **Secure**: Secure mode allows `tftp` access for a single directory. When invoked, the TFTP server uses a `chroot(2)` system call to change its root directory. All other directories are inaccessible to the client. Because of the `chroot(2)` system call, `tftpd` must be run as root. Additionally, secure mode allows for a particular user to be specified with the `-u` option.

Secure `tftp` is set with the `-s` option and a directory name. For example, to set up the directory `/tmp/project` for secure `tftp` for the user `fred`, use the following command:

```
# tftpd -s /tmp/project -u fred
```

For more information, see the `tftpd(8)` man page.

Remote Copy (rcp)

The remote copy command or **`rcp`** is an efficient way to exchange data between UNIX-compatible hosts. To access files using `rcp`, users must have already set up the `/etc/hosts.equiv` and `.rhosts` files correctly. Files can be copied 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` is copied from host `orca` to host `shark`:

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

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

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

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

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

Divert Sockets

`divert` provides a kernel packet diversion mechanism. Divert sockets are similar to raw IP sockets except that they can be bound to a specific divert port with the `bind` system call. A divert socket bound to a divert port receives all packets diverted to that port.

Divert sockets are normally used in conjunction with packet filtering. By reading from and writing to a divert port, matching packets can be passed through a filter as they travel through host machine. For more information refer to the `divert(4)` and `bind(2)` man pages.

NAT and IP Masquerading

IP masquerading is a form of NAT (Network Address Translation) that can be implemented in conjunction with the `natd` daemon.

This feature allows internally connected computers that do not have registered Internet IP addresses to communicate to the Internet by way of `natd` running on a LynxOS host gateway that has a registered Internet IP address.

Network Address Translation can be used to redirect an IP access to a particular client, or redirect specific internet services to a particular client and port.

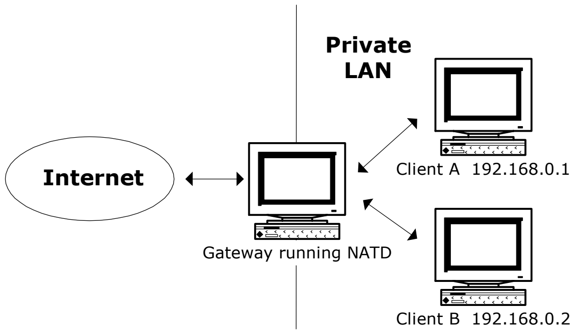

Figure 1-16: Network Address Translation

Enabling and Starting natd

NAT is enabled with a `sysctl`:

```
# sysctl -w net.inet.ip.forwarding=1
```

Once natd is enabled in the kernel, start the daemon with the proper configuration options. `natd` syntax is as follows:

```
# natd <redirection_options>
```

The following table describes the redirection options used with `natd`:

Table 1-3: natd Options and Descriptions

Option	Description
`-interface` *`<ifx>`*	Uses the IP address set to the interface *`<ifx>`* for aliasing.
`-redirect_address` *`<localIP> <publicIP>`*	Redirects access to a local IP address to public IP address. This option maps an external IP address to a local client IP. This is known as static NAT, and is useful if there are several external IP addresses available.
	`<localIP>` is the IP address of the local system
	`<publicIP>` is the external IP address
`-redirect_port` *`<protocol> <targetIP:targetPort> [<aliasIP>]:<aliasPort> [<remoteIP:remotePort>]`*	Redirects access to networking services.
	`<protocol>` -- the protocol used (`tcp` or `udp`).
	`<targetIP>``:`*`<targetPort>`* -- the IP address and port of the local client to direct services to. A range of ports can also be specified. For example, (`192.168.0.1:2000-3000`).
	`[`*`<aliasIP>`*`:]`*`<aliasPort>`* -- The services port being requested. A range of ports can also be specified. For example, (`2000-3000`).
	`[`*`<remoteIP>`*`:`*`<remotePort>`*`]` -- Used to specify the remote IP address and port of a requesting system.

For example, two clients, Client A (192.168.0.1) and Client B (192.168.0.2) are connected to a gateway. IRC services (port 6667) are required for ClientA, and HTTP services are required for ClientB (port 80). To configure `natd` to redirect incoming packets to these ports, enter the following commands:

```
# natd -interface pro0 -redirect_port tcp 192.168.0.1:6667 6667
# natd -interface pro0 -redirect_port tcp 192.168.0.2:80 80
```

Additional configurations and options are available in the `natd(8)` man page.

IPv6 Support

NOTE: The IPv6 and IPsec protocols for LynxOS are not included with the standard LynxOS package. These components are available for purchase separately. For information on these products, please contact your LynuxWorks sales representative.

The IPv6 protocol addresses technical limitations of IPv4. Most notably is the increase IP address space, which has changed from 32 to 128 bits per address.

IPv4 32-bit addresses are represented in dotted-decimal format divided along 8-bit boundaries. IPv6 IP addresses are 128-bit address divided along 16-bit boundaries, and each 16-bit block is converted to a 4-digit hexadecimal number and separated by colons. For example:

```
200A:00A3:2C5B:0000:02FF:FF00:FE38:934A
```

The IPv6 address representation can be further simplified with features such as leading zero suppression and zero compression. Detailed information on these features as well as other information regarding the IPv6 addressing architecture are described in RFC 2373.

Other notable improvements is the inclusion of the following services that were optional under IPv4:

- Autoconfiguration—IPv6 specifies a stateless host autoconfiguration mechanism which is an improvement on the optional DHCP mechanism used with IPv4.
- Security—IPv6 mandates support for IPsec. This guarantees that a secure IP connection can be established when communicating with IPv6 devices.
- Multicast—Muliticast is now mandatory under IPv6.

In addition, IPv6 has simpler packet header structures and also introduces a protocol header chain that allow for more flexible protocol extensions

Additional information on the specifications of IPv6 can be found at `www.ipv6.org`.

Setting an IPv6 Address Statically

Use the following steps to configure and test the system for assigning a static IP address. It is important to note that manually setting and maintaining IPv6 IP addresses can be complicated. Due to the complex nature of address notation, mistakes are more likely to occur with IPv6 than IPv4. Users are cautioned to double check addresses set manually.

1. Update `/etc/hosts` to include the IPv6 loopback address, the IPv6 address for the host, and an external IPv6 host address.

```
127.0.0.1              localhost       #IPv4 Loopback Address
::1                    localhost       #IPv6 Loopback Address

192.168.0.1            myhost          #IPv4 Hostname Address
3ffe:0501:1234:ffff:: myhost          #IPv6 Hostname Address

3ffe:0501:1111:ffff:: otherhost       #entry for external IPv6 host
```

Figure 1-17: Example IPv6 /etc/hosts file

2. Assign an interface an IPv6 address with `ifconfig`. Note that for IPv6 addresses, the switch `inet6` must be used:

```
# ifconfig <if> inet6 <address>
```

3. Use `ping6` to ping additional IPv6 networks

```
# ping6 otherhost
```

Setting up Hostname Resolution for IPv6 Addresses

Hostname resolution is configured by the `/etc/resolv.conf` file. IPv6 addresses can be used along with IPv4 address to perform hostname resolution. The following provides an example `resolv.conf` file:

```
nameserver     3ffe:0501:1111:ffff::
nameserver     192.168.99.1
```

Figure 1-18: Example IPv6 resolv.conf file

Setting up Routes with route6d

The `route6d` daemon is an extension of `routed` that includes support for RIP over IPv6. Refer to the `route6d(8)` man page for syntax and usage information.

Additionally, other routing demons that support IPv6 can be used; `zebra`, for example. For additional information, see the *GNU Zebra User's Guide*.

Using faithd to Connect IPv6 and IPv4 Networks

The `faithd` daemon is used to provide a IPv6 to IPv4 relay. `faithd` performs TCP relay similar to firewall gateways, but with the addition of address translation. `faithd` is used only to translate IPv6 addresses to IPv4.

The following provides an example of setting up and configuring `faithd` relay for `telnet`. On the translating router where faithd runs, perform the following:

1. Use `sysctl` to allow for route advertising, IP6 forwarding, and faith:

   ```
   # sysctl -w net.inet6.ip6.accept_rtadv=0
   # sysctl -w net.inet6.ip6.forwarding=1
   # sysctl -w net.inet6.ip6.keepfaith=1
   ```

2. Start the `faith0` interface, and create a route to `faith0`:

   ```
   # ifconfig faith0 up
   # route add -inet6 3ffe:501:1234:ffff:: \
   -prefixlen 96 ::1
   # route change -inet6 3ffe:501:1234:ffff:: \
   -prefixlen 96 -ifp faith0
   ```

3. Execute `faithd` for the telnet port as follows:

   ```
   # faithd telnet /net/telnetd
   ```

The first argument is a service name for TCP relay. The service can be specified either by the port number (`23`) or by service name (`telnet`). The second argument is a path name for the local IPv6 TCP server. If there is a connection to the router itself, this program is invoked.

Note that `faithd` must be invoked for each service required.

Hostname Resolution Between IPv6 and IPv4 Hosts

The simplest way to translate an IPv4 address to IPv6 address is to add an entry to the /etc/hosts file. On the IPv6 host, add a line that resolves the IPv6 and IPv4 addresses:

```
3ffe:0501:1234:ffff::192.168.0.1     hostname
```

Figure 1-19: /etc/hosts example

Driver Defaults

The following table shows the default values within the driver information files. These files are located in /sys/devices.

To change the defaults, edit the file, compile it, and install TCP/IP support again. For more information, see "Installing/Removing TCP/IP Support" on page 1.

Table 1-4: Default Values within Driver Information Files

Info File	Platform	Defaults	Notes

hbtcpip_info.c	All	Mbufs (multiple of 4): 4096
		clusters=(1/4 of mbufs)1024
		tcp_sendspace=16384
		tcp_receivespace=16384
		udp_sendspace=16384
		udp_receivespace=41984
		cluster_size=2048
		clshift_bits=11
		ipforwarding=1
		tcprexmtthresh=3
		tcp_mssdflt=512
		tcp_keepintvl=150
		tcp_do_rfc1323=0
		tcpip_max_prio=255
			This file contains info about memory usage and other control args for the TCP/IP stack. Please see the hbtcpip0(4) man page for detailed information on configuration options.

if_3c509info.c	x86	iobase=240
		intr=5
		slotno=0
		default BNC
			TCP/IP driver for the 3com 3c509 ISA and PCMCIA controllers. (Values are used only for PCMCIA controller, ISA card values are read from eeprom.

if_3c90xinfo.c	x86	None used	TCP/IP driver for 3com vortex/boomerang/cyclone PCI controllers.
if_amd970info.c	x86/ppc	None used	TCP/IP driver for and PCNET controllers (93970/93971/93972/ 93973)

Table 1-4: Default Values within Driver Information Files (Continued)

Info File	Platform	Defaults	Notes
`if_dec21040info.c`	x86/ppc	`probeint=10`	TCP/IP driver for any Digital Tulip compatible controller (2140/21041, 21140, 21142, 21143, 82168, 82169, 82115, 98100)
`if_epicinfo.c`	x86	None used	TCP/IP driver for SMC epic controller.
`if_fccinfo.c`	ADS 8260	None used	TCP/IP driver for 10/100 Base T Ethernet connected to FCC2.
`if_gxinfo.c`	x86	None used	TCP/IP driver for Intel 82235 Gigabit Ethernet chip.
`if_pro100binfo.c`	x86/ppc	None used	TCP/IP driver for Intel pro controller (82557/82558/82559)
`if_neinfo.c`	x86/pmc860	`iobase=300` `intr=5`	TCP/IP driver for NE2000 ISA/PCI controllers (values used only for ISA).

Additional Device Drivers may be included with the LynxOS ODE or BSP packages. Refer to the driver file or man page for additional tunable information.

NOTE: On the x86, if RAMBase is changed, 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: In the `hbtcpip_info.c` file, the checksum calculation for UDP packets can be enabled/disabled by setting the `udpcksum` filed to 1 (enable) or 0 (disable). Disabling the checksum increases the throughput for UDP packets.

For specific information on adding device drivers to a LynxOS system, please refer to the *LynxOS User’s Guide*.

Additional information on creating device drivers is available in the book *Writing Device Drivers for LynxOS*.

CHAPTER 2

Network Security

This chapter provides an overview of the network security components provided with LynxOS, including:

- Firewalls
- IPsec

Firewalls

LynxOS TCP/IP provides network protection from external intrusion with the incorporation of firewalls. Firewalls provide system administrators the ability to allow or deny the forwarding of packets through a network. Both inclusive and exclusive rule sets can be implemented, and protection from spoofing and bandwidth limitation can also be set. LynxOS supports the packet filter `ipfw` utility.

ipfw

IPFW is a kernel-based utility that is used to provide packet filtering and counting. Using a set of user-defined rules, IPFW controls whether or not network packets are forwarded or blocked. These user-defined rules, also called a firewall chain (or rule chain), can block or allow specific requests from specific hosts.

The user utility `ipfw(8)` is used as an interface to the IPFW kernel component.

`ipfw` can be set up to monitor both incoming and outgoing connections.

Enabling ipfw

IPFW is enabled with `sysctl`:

```
# sysctl -w net.inet.ip.fw.enable=1
```

NOTE: When enabled, IPFW leaves all ports open by default.

Changing ipfw Rules

The `ipfw` user command is used to set the packet filter rules. Adding a rule with the `ipfw` command updates the current firewall chain. The simplest usage syntax is as follows:

```
# ipfw <command> <action> <protocol> <address> <options>
```

The following table defines each of the arguments for `ipfw`.

Table 2-1: Changing ipfw Rules

Option	Command & Description
<command>	**add** -- Adds an entry to the firewall chain
	delete -- Deletes an entry to the firewall chain
<action>	**reject** -- Drop packet and send ICMP host unreachable to source
	allow -- Allow packet to pass
	deny -- Drop packet and do *not* notify source
	count -- Update packet counter
<protocol>	**all** -- Set rule for all protocols
	icmp -- Set rule for ICMP packets only
	tcp -- Set rule for TCP packets only
	udp -- Set rule for UDP packets only

<address>	Defines the IP address, mask, and (if required, port). Both From and To addresses can be specified, as well as the interface used (`eth0`, for example):
	`from <address/mask>[<port>] to <address/mask>[<port>] \`
	`[via <interface>]`
	
	Specific ports, or a range of ports can be set:
	`<port>, <port>, <port>`
	`<port>-<port>`

Table 2-1: Changing ipfw Rules (Continued)

Option	Command & Description
<options>	**via** *<if>* -- Packet must be going through interface *<if>*. **via** *<if*>* -- Packet must be going through interface <*ifX*>, where *X* is a unit number. **via** *<any>* -- Packet must be going through some interface. **via** *<ip>* -- Packet must be going through the interface with the IP address *<ip>*.
	frag -- Matches if the packet is a fragment.
	in -- Matches if packet is incoming.
	out -- Matches if packet is outgoing.
	ipoptions *<spec>* -- Matches if IP header contains options specified in <`spec`>.
	established -- Matches if the packet is part of an established TCP connection.
	setup -- Matches if the packet is attempting to setup a TCP connection (`syn` bit set, `ack` not set).
	tcpflags *<flags>* -- Matches if the TCP header contains the flags specified in <*flags*>. Supported flags include `fin`, `syn`, `rst`, `psh`, `ack`, `urg`.
	icmptypes <*types*> -- Matches if the ICMP type is specified in <*types*>.

For example, to set up a rule to deny all incoming packets from the address 192.168.1.1 to the address 192.168.2.2 over TCP/IP would be as follows:

```
# ipfw add deny tcp from 192.168.1.1 to 192.168.2.2 in
```

To deny packets from the Ethernet port 1 to the telnet port (port 23) of 192.168.1.1, the command is:

```
# ipfw add deny tcp from 192.168.1.1 to 192.168.2.2 23 \
via eth1
```

Listing ipfw Rules

The current set of `ipfw` rules is displayed with the `list` argument:

```
# ipfw list
```

Removing ipfw Rules

The flush argument to `ipfw` removes all rules currently set for packet filtering.

```
# ipfw flush
```

Clearing ipfw Counters

`ipfw` counters setup with the count option can be cleared with the `zero` argument to `ipfw`:

```
# ipfw zero [<index>]
```

`<index>` affects only the counter at the index number specified. If *`<index>`* is not used, all packet counters are cleared.

ip6fw

The `ip6fw` utility is an IPv6 implementation of IPFW. The syntax and usage is identical to IPFW, with the following exceptions to the *`<protocol>`* option:

Table 2-2: ip6fw Specific Options

Option	Command & Description
`<protocol>`	**`ipv6`** or **`all`** -- Set rule for all protocols
	`ipv6-icmp` -- Set rule for IPv6 ICMP packets only
	`tcp` -- Set rule for TCP packets only
	`udp` -- Set rule for UDP packets only
	`ipv6no/prefixlen` An IPv6 number with a prefix length of the form `fec0::1:2:3:4/112.`

IPsec

IPsec is a security protocol in the IP layer that provides a secure means of communications between two hosts. IPsec can create a tunnel between subnets (tunnel mode) or provide security between two hosts directly (transport mode.) Hosts resolve the encryption keys and certification allowing for encrypted packets to be exchanged. Security information can be exchanged manually, or automated with the `racoon` daemon. Both IPv4 and IPv6 are supported.

NOTE: The IPsec and IPv6 protocols for LynxOS are not included with the standard LynxOS package. These components are available for purchase separately. For information on these products, please contact your LynuxWorks sales representative.

IPsec policies are configured with the `setkey(8)` utility. See “Using setkey” on page 30 for additional information.

AH and ESP Security Protocols

IPsec uses two security protocols:

- AH (Authentication Header)

  AH contains hashes of data identification in the header, protecting the source and destination addresses of the packet. A shared secret between hosts ensure that packets are sent and received from the right system.

- ESP (Encapsulated Security Payload)

  The ESP header allows for encryption or decryption of the packet by a shared secret between hosts.

Tunnel Mode and Transport Mode

IPsec includes two modes of communicating packets:

- Tunneling -- A connection between two systems on the same subnet. Typically, IPsec is used in tunnel mode to establish a VPN. In tunnel mode, IPsec encrypts the payload, and encapsulates it in a packet before sending it to the host.
- Transport -- A connection between two systems, where the payload for IPsec packets are encrypted. In transport mode, IPsec appends outgoing IP packets with a security protocol header. The IPsec header is determined by the original packet, and security information is included by the packet header.

Setting the Security Policy Database (SPD)

A (SPD) Security Policy Database is kept in the kernel that determines what encryption algorithms should be used, the security protocol to use (AH or ESP), and what packets to encrypt. The utility `setkey(8)` is used to modify the policies kept in the SPD.

Setting the Security Association Database (SAD)

The encryption keys used in secure transactions are kept in the kernel table called SAD (Security Association Database). The SAD contains the list of keys that are required for secure communications. The SAD can be manually updated with the `setkey(8)` utility. For hosts supporting IKE (Internet Key Exchange), the SAD can be automatically updated with the `racoon` daemon.

Using setkey

The `setkey` utility, invoked with the `-c` option, reads commands from standard input. Invoked with the -f option, `setkey` reads commands from a filename. The syntax used with `setkey` is as follows:

```
# setkey -f <filename>
```

or

```
# setkey -c
... <rules>
<Ctrl-C>
```

The `ipfw` syntax is as follows:

```
<command> <src_addr> <dest_addr> <protocol> <spi> \
[<extension>] <algorithm> [-P <policy>]
```

NOTE: If using IPsec behind a firewall, be sure to open any required ports required by ESP. For UDP, for example:

```
ipfw add pass esp from any to any
```

The following table describes the available commands and arguments:

Table 2-3: setkey Command Options and Descriptions

Option	Commands and Descriptions
<command>	**add** -- Adds an SAD entry
	spdadd -- Adds an SPD entry
	get -- Show a SAD entry
	delete -- Removes an SAD entry
	flush *<protocol>* -- Clears all SAD entries matching *<protocol>*
	dump *<protocol>* -- Dumps all SAD entries matching *<protocol>*
<src_addr>	IP address of source system
<dest_addr>	IP address of destination system
<protocol>	**esp** -- ESP based on RFC2405
	esp-old -- ESP based on RFC1827
	ah -- AH based on RFC2402
	ah-old -- AH based on RFC1826
	ipcomp -- IP COMP
<spi>	Security Parameter Index (SPI) for the SAD and SPD. A decimal or hexadecimal number.
[*<extension>*]	**-m** *<mode>* -- Specify a security protocol mode for use. *<mode>* is one of following: `transport`, `tunnel` or `any`. The default value is `any`.
	-r *<size>* -- Specify size of bytes for replay prevention. *<size>* must be a 32-bit decimal number. If *<size>* is zero or not specified, replays are not checked.
	-u *<id>* -- Specify the identifier of the policy.
	-f *<pad_option>* -- Where *<pad_option>* is one of following: `zero-pad`, `random-pad` or `seq-pad.` **-f nocyclic-seq** -- Don't allow cyclic sequence numbers.
	-lh *<time>* -- Specify hard lifetime **-ls** *<time>* -- Specify soft lifetime.
<algorithm>	**-E** *<ealgo key>* -- Specify encryption algorithm.
	-A *<aalgo key>* -- Specify authentication algorithm. If `-A` is used for ESP, it will be treated as ESP payload authentication algorithm.
	-C *<calgo>* -- Specify compression algorithm.

Table 2-3: setkey Command Options and Descriptions (Continued)

Option	Commands and Descriptions
`[-P <policy>]`	**`-P`** *`<direction>`* **`discard`** -- Discards packet matching index. *`<direction>`* is **`in`** or **`out`**.
	`-P` *`<direction>`* **`none`** -- Specifies IPsec not to operate on packet. *`<direction>`* is **`in`** or **`out`**.
	`-P` *`<direction>`* **`ipsec`** *`<protocol>/<mode>/<src>-<dst>/<level>`* -- Specifies a Policy for IPsec to operate on a packet. *`<protocol>`* is `ah`, `esp`, or `ipcomp`. *`<mode>`* is `transport` or `tunnel` *`<src>-<dst>`* is the beginning and end point addresses used to specify the SAD *`<level>`* is either `default`, `use`, or `require`. `default` means the kernel consults to the system-wide default against the protocol specified when the kernel processes the packet. `use` means that the kernel uses an SA if it's available, otherwise the kernel operates normally. `require` means SA is required whenever the kernel deals with the packet. *`<direction>`* is `in` or `out`.

The following commands and files provide an example of using `setkey` for updating the SAD and SPD kernel tables. The following instructions must be completed for both systems using IPsec.

1. Flush the current SAD and SPD:

   ```
   # setkey -F
   ```

2. Create a file that contains the policies for SAD. In the example file below, two hosts (192.168.0.1 and 192.168.0.2) are enabled to use the 3des-cbc encryption algorithm using ESP. The first two lines enable the source and destination addresses, the last line updates the SPD table

   For Host A, use the following:

```
# IPv4 ESP
# local system: 192.168.0.1
# remote system: 192.168.0.2

add 192.168.0.1 192.168.0.2 esp 1234 -E 3des-cbc “123456781234567812345678”;
add 192.168.0.2 192.168.0.1 esp 9999 -E 3des-cbc “123456781234567812345678”;
spdadd 192.168.0.1 192.168.0.2 any -P out ipsec esp/transport//use;
```

Figure 2-1: Sample Configuration File (hostA.sample.policies)

For Host B, use the following (note the `spdadd` change):

```
# IPv4 ESP
# local system: 192.168.0.1
# remote system: 192.168.0.2

add 192.168.0.1 192.168.0.2 esp 1234 -E 3des-cbc “123456781234567812345678”;
add 192.168.0.2 192.168.0.1 esp 9999 -E 3des-cbc “123456781234567812345678”;
spdadd 192.168.0.2 192.168.0.1 any -P out ipsec esp/transport//use;
```

Figure 2-2: Sample Configuration File (hostB.sample.policies)

3. Point `setkey` to the SAD file for each host. For Host A:

   ```
   # setkey -f hostA.sample.policies
   ```

   For Host B:

   ```
   # setkey -f hostB.sample.policies
   ```

   `setkey` reads in the file and updates the SPD and SAD as necessary.

Additional configuration details are available in the `setkey(8)` man page. Also, refer to the `racoon` utility man page for information on automating the SAD update.

CHAPTER 3 *PPP*

PPP (Point-To-Point Protocol) is a protocol used to send and receive data packets reliably over a serial line. This chapter describes LynxOS PPP components and how to configure PPP servers and clients.

LynxOS PPP Components

PPP, unlike SLIP (also used to communicate over serial lines) allows the communicating side to negotiate options. PPP also provides for authentication, by way of exchanging secret tokens between the client and the server. The table below lists the LynxOS PPP components:

Table 3-1: LynxOS PPP Components

Component	Definition
`/net/pppd`	User level daemon that controls the PPP protocol.
`/net/chat`	Automated conversational script with a modem.
`/etc/ppp/options`	PPP controlling options.
`/etc/ppp/chap-secrets`	CHAP secrets file.
`/sys/drivers/if_ppp`	PPP kernel driver.

For more information on these components, see the `pppd` and `chat` man pages. These man pages cover the PPP controlling options, `chap-secrets` file, and the PPP kernel driver.

Installing/Removing PPP Support

PPP can be installed and/or removed after the initial installation of LynxOS. The `Install.ppp` and `Uninstall.ppp` scripts are located in `/usr/bin`.

Installing PPP

To install PPP, execute this script:

```
# Install.ppp
```

Removing PPP

To remove PPP:

```
# Uninstall.ppp
```

For more advanced information on these steps, see "Customizing LynxOS" in the *LynxOS User's Guide*.

Configuring the PPP Server

The PPP Server is configured in one of two ways:

- Running **pppd** on each serial line.
- Creating an account and login shell for a user `ppp`.

Running pppd on Each Serial Line

The advantages of using this method are as follows:

- Exclusive options can be provided to each of the PPP serial lines.
- The client can be assigned a unique IP address.

Use the following procedure to run **pppd** on each of the serial lines:

1. Log in as `root`.
2. Run the `pppd` daemon by entering the following command:

```
# pppd /dev/com2 9600 <server>:<client> \
auth silent persist &
```

where `<client>` is the client and `<server>` is the PPP servername. When the PPP runs over the com2 serial line, the client connects with a hostname (`client`) and the connection is authorized using `CHAP` secrets.

NOTE: This line can be added to system startup scripts to automate the PPP server at boot time.

This command also ensures that `pppd` waits for an incoming connection, and upon termination of connection, the daemon is ready for new connections.

The rest of the `pppd` options are taken from the `/etc/ppp/options` file. In this case, the login on the serial port com2 should be disabled by the `/etc/ttys` controls.

If no authorization is preferred, the option `noauth` should be used instead of `auth` when starting the `pppd` daemon. For more detailed information, see the `pppd` and `chat` man pages.

For a complete list of configuration options for the `ppp` daemon, please see the `pppd(8)` man page.

Creating a ppp Account and login Shell for pppd

A special account must be created called `ppp`. In addition, the `login` shell for the serial device must be configured as the `pppd` daemon. Use the following instructions to create a `ppp` account:

1. Use the `adduser` script to add an account for a user called `ppp`.
2. Add a password to the `ppp` account. A typical entry in the `/etc/passwd` file for a `ppp` account is as follows:

   ```
   ppp:KByn.VTqKXWsE:101:101:PPP\
   connections:/usr/ppp:/net/pppd
   ```

   In this example, the client is authenticated with the `ppp` password, in addition to possible authentication with `CHAP` secrets.
3. Enable the shell login on the serial port by entering the serial device in `/etc/ttys` file as follows:

   ```
   /dev/com2:1:ppp:vt100:/bin/login
   ```

4. Add the terminal configuration for this serial line to the `/etc/tconfig` file as `ppp`:

```
ppp:\ :si=9600:so=9600:\ :tc=default:
```

The PPP client is expected to login as the user `ppp` with the appropriate password to initiate the PPP protocol. The `ppp` sessions on the server are controlled with the `/etc/ppp/options` file. Therefore, when supporting PPP over multiple serial lines, it is assumed that the client comes with it's own IP address, asking for default route and proxy ARP supports. For a complete list of options for the `pppd` daemon, see the man page for `pppd`.

Configuring the PPP Client

It is recommended that the LynxOS PPP client kernel be built with PPP driver support as described in the previous section. To establish the PPP connection to the server, perform the following tasks:

- Dial the server and establish the link.
- Process the PPP protocol and authentication to effect the connection.

NOTE: `syslog` must be running to view any output from the ppp daemon. Start syslog with the command **`syslogd`**. For configuration information, see the `syslog(3)` man page.

Dialing the Server

The **`chat`** utility can be used to dial the server, and as an argument to `pppd` by using the connect option. The `chat` utility supplies a set of strings that reflect the input and response from the serial line.

Different arguments are required with the chat program, depending on the requirements of the server:

- Special login with a `ppp` user-ID in addition to a Dial-in.
- Dial-in facility alone.

Using chat with Special Log in ppp User-ID

The following figure shows a sample `chat` script that assumes the client login to be user-ID `ppp.`

```
chat -v ABORT BUSY ABORT “NO CARRIER” ABORT “NO DIAL TONE” \
““ATZ OK ATDT192 CONNECT ““ ““ ““ ““ ““ name: ““ ““ ““\ name: ““ ““ ““ name:
ppp ssword: allowme
```

Figure 3-1: chat Script for a Special Log in and ppp User-ID

This script shows how to respond to a given input string/condition. If a NULL command is given, `chat` sends an `ATZ` command to the modem to initialize it, and on receiving the OK, dials out the number 192. The script expects `CONNECT` to come from the remote side, and sends a set of null commands ultimately expecting the `name` string from the remote side. It responds with `ppp` and on receiving the string `ssword` sends the previously agreed password `allowme`. This `chat` session assumes that the PPP server is configured with a user ID `ppp` to initiate the PPP connection with the password `allowme`.

The following figure shows a typical `pppd` command for the `chat` script that uses the `ppp` user-ID:

```
pppd /dev/com2 9600 \
connect ‘chat -v ABORT BUSY ABORT “NO CARRIER” ABORT “NO DIAL TONE” \
ATZ OK ATDT192 CONNECT ““ ““ ““ ““ ““ name: ““ ““ ““ name: ““ ““ ““ \
name: ppp ssword: allowme’ \
xonxoff nocrtscts asyncmap a0000 noipdefault defaultroute bsdcomp 15 \
nodeflate remotename lynx_se
```

Figure 3-2: chat Script for PPP Server

In this example, the `pppd` command assumes that the PPP server is configured so that the `pppd` daemon is started by the `ppp` user ID login session.

Using chat as a Dial-in Facility

Alternatively, the PPP server may already be started on the serial line, waiting for incoming IPCP packets. The following figure shows this case, where the `chat` script dials out to initiate the modem connection.

```
chat -v ABORT BUSY ABORT “NO CARRIER” ABORT “NO DIAL TONE” \
ATZ OK ATDT192 CONNECT
```

Figure 3-3: chat Script for a Dial-in Facility

Typically, this command is used with `pppd` as shown in the following figure.

```
pppd /dev/com2 9600 \
connect ‘chat -v ABORT BUSY ABORT “NO CARRIER” ABORT “NO DIAL TONE” \
ATZ OK ATDT192 CONNECT ‘ \
xonxoff nocrtscts asyncmap a0000 noipdefault defaultroute bsdcomp 15\
nodeflate remotename lynx_se user lynx_cl
```

Figure 3-4: Sample pppd Command for a Dial-in Facility

For more detailed information on setting up the PPP client and server, see the `pppd` and `chat` man pages.

CHAPTER 4 DHCP

DHCP (Dynamic Host Configuration Protocol) enables systems to request network configuration information (IP address, local nameservers, and routers) from a DHCP server. This chapter details DHCP concepts, installation and configuration.

Introduction

Dynamic Host Configuration Protocol (DHCP) enables individual clients on an IP network to receive network configurations from a DHCP server. The DHCP server provides pre-defined network configuration information dynamically, including client IP addresses, nameservers, and routers. DHCP reduces the amount of overhead in administering a large IP network by freeing the system administrator from having to reconfigure each system individually.

DHCP also enables the storage and distribution of network parameters for clients and groups. Additionally, DHCP allows for recovery and reallocation of these network addresses through a leasing mechanism. Client network configurations expire after a designated period of time, after which the client system's network configuration is either renewed or reissued from the DHCP server. Another feature of this protocol is that it can be routed. DHCP support consists of a server daemon, client utility, a relay daemon and a diagnostic utility and a set of configuration database files. DHCP support requires TCP/IP.

LynxOS DHCP Components

LynxOS uses the ISC (Internet Software Consortium) Open Source DHCP version 2.0 distribution. DHCP consists of two software components:

- DHCP Server
- DHCP Client

The DHCP Client sends a request to a DHCP server, which responds with particular network parameters for the client to use. The following sections describe DHCP files and configurations.

LynxOS DHCP Files

This table details the various configuration files used to configure a LynxOS system for DHCP.

Table A-1: LynxOS DHCP Files

Component	Definition
`/bin/dhclient`	DHCP client configuration binary
`/etc/dhcpd.conf`	DHCP configuration file
`/etc/dhcpd.leases`	DHCP lease configuration file
`/etc/dhclient-script`	DHCP client configuration script
`/net/dhcpd`	DHCP daemon
`/net/dhcrelay`	DHCP relay agent

DHCP man pages

Additional DHCP configuration information is available in these man pages:

Table A-2: DHCP man pages

man page	Description
`dhcpd(1)`	The DHCP server
`dhcpd.conf(5)`	The DHCP server configuration file
`dhcp-options(1)`	DHCP option statements

Table A-2: DHCP man pages (Continued)

man page	Description
`dhclient(1)`	DHCP client for LynxOS
`dhclient.conf(5)`	DHCP client for LynxOS config file
`dhrelay(1)`	DHCP relay Agent
`dhclient.leases(5)`	DHCP lease file

Online Resources

Additional information on DHCP is available at the Internet Software Consortium web page:

```
http://www.isc.org
```

Installing DHCP

DHCP is included with the LynxOS base product, and is installed by default. Configuring DHCP is explained in the following sections.

The DHCP Server

The DHCP server responds to requests from DHCP clients with network configuration information. DHCP server components include: `dhcpd` and `dhcpd.conf`.

dhcpd

The DHCP server daemon (`dhcpd`) serves DHCP client requests based on the configuration file `dhcpd.conf`. The server also monitors the lease details of the client IP addresses, facilitating the recovery and reallocation of IP addresses through predefined lease agreements. Additionally, the server also services `BOOTP` clients. The DHCP server maintains all current client configurations in the `dhcpd.leases` file. For more information on `dhcpd.leases`, See “dhcpd.leases” on page 45.

By default, `dhcpd` uses UDP port 67 to receive and UDP port 68 to send DHCP information.

NOTE: It is recommended that the DCHP server be incorporated into system startup scripts to automate this functionality; for example adding a line executing `dhcpd` in `/etc/rc/network`.

---

dhcpd.conf

The DHCP configuration file, `dhcpd.conf`, must be created by hand. This configuration file sets specific network topologies and parameters the DHCP server uses to assign IP addresses, nameservers, and routers to clients. In addition to a range of available IP addresses, `dhcp.conf` must provide the IP address subnet mask, and at least one router and nameserver on the network.

Network topology declarations include the `shared-network` and `subnet` declarations. Dynamically assigned addresses must include the `range` declaration with a `subnet` declaration. DHCP options are declared per subnet. `lease` durations are expressed in seconds.

The following is an example of a `dhcpd.conf` file:

```
# shared-network
subnet 192.168.11.0 netmask 255.255.255.0 {
   range 192.168.11.1 192.168.11.251;
   option routers 192.168.11.254;
   option subnet-mask 255.255.255.0;
   option domain-name-servers 192.168.11.253;
   # default lease time=30 Days
   default-lease-time 2592000;
   # max lease time=45 Days
   max-lease-time 3888000;
   }
```

Figure 4-1: example dhcpd.conf file

Additional information on specific options in configuring `dhcpd.conf` can be found in the `dhcpd.conf(5)` man page.

After editing the `dhcpd.conf` file, restart the DHCP daemon:

1. Find the Process ID (PID) of the DHCP daemon (`dhcpd`) with the following command:

```
# ps -axon | grep dhcpd
```

2. Use the `kill` command to stop the process.

   ```
   # kill <PID>
   ```

3. Restart the DHCP daemon:

   ```
   # /net/dhcpd
   ```

NOTE: X86 systems that netboot LynxOS use RARP and TFTP. However, PXE compatible devices must use DHCP.

NOTE: PPP Server and Client IP address allocation are not integrated with DHCP and continue to work independently.

NOTE: Dynamic DNS update is not currently supported.

---

dhcpd.leases

This is a generated file that contains lease expirations for client network configurations. The DHCP server maintains up-to-date information on client leases in this file. In the event of a system reboot, the DHCP server can resume all current connections without having to reconfigure each client on the network. This file should not be edited.

Relay Agents

Relay Agents allow a DHCP server to configure systems on more than one network segment. Relay agents forward DHCP requests from clients to the DHCP server, allowing a single DHCP server to serve multiple clients in multiple network segments. Using relay agents provides flexibility in configuring a DHCP network.

The DHCP Client

dhclient

The DHCP client (`dhclient`) sends requests to a DHCP server for network configuration information. The primary information sought from the server is the client IP address. The IP address is issued for a specific “lease” period, varying between a few hours up to a year. Initially, `dhclient` broadcasts DHCP requests using the UDP protocol over a chosen network interface (passed as an argument) to the DHCP server.

Once it establishes contact with the server, the network configuration is applied to the client. The client network configuration is valid until the lease period expires, after which, the client attempts to renew the lease, or obtain a lease on new network configurations. After configuring the client, the dhclient script becomes a background process to continue to contact the DHCP server.

While configuring a `netbooted` client to use `bootp` and DHCP, choose the DHCP option in the scripts. If not, the traditional RARP is used to netboot the client.

In addition, online man pages contain detailed information.

Starting dhclient

Start the `dhclient` script by typing:

```
# dhclient
```

NOTE: It is recommended that `dhclient` be incorporated into system startup scripts to automate this functionality; for example adding a line executing `dhclient` in `/net/rc/network`.

CHAPTER 5 *NFS*

During initial installation of LynxOS, NFS (Network File System) support is installed with the `installit` utility. NFS can also be configured, installed, or removed at any time after initial installation. Note that NFS requires TCP/IP to function. This chapter describes NFS basics, as well as advanced NFS configuration options.

Overview

NFS is a suite of user programs and kernel functionality that allow access to a remote host's filesystems as if they were local. 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. 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
- 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 able to recover from NFS server/client crashes.

NFS software is divided into two parts:

- NFS server
- NFS client

When attempting 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 to the local disk.

Installing/Removing NFS Support

NFS can be installed and/or removed after initial installation of LynxOS. Note that NFS requires TCP/IP. The `Install.nfs` and `Uninstall.nfs` scripts are located in `/usr/bin`.

Installing NFS

To install NFS:

```
# Install.nfs
```

Removing NFS

To remove NFS:

```
# Uninstall.nfs
```

These installation and uninstallation scripts automatically install and configure `nfs` for LynxOS.

Tuning the NFS Server Kernel

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

To change any of these parameters, edit the file `/sys/devices/nfssvc_info.c`. After editing, the device library must be updated and the kernel rebuilt.

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

NFS Server Tunable Parameters

Tunable parameters include:

- 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.

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

  The default is 16.

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

  The default is 16. For security reasons, this parameter can be decreased.

- 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, this parameter can be decreased.

- 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, multiple daemons should be started. This is 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 read: `/net/nfsd 3`.

Tuning the NFS Client Kernel

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

- 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.

- The maximum number of NFS directories that can be mounted.

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

- 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, multiple client daemons should be started. This can be done by duplicating the `/net/unfsio` line in `/net/rc.network`.

- 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.

- 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, the device library must be updated, and the kernel rebuilt. The *LynxOS User's Guide* describes how to update and rebuild the kernel.

Tuning the NFS Client-Side Cache

The amount of memory allocated for NFS client-side caching can be tuned by increasing or decreasing 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 */
};
```

Figure 5-1: nfs_cache_info.c example file

The various fields in this structure are explained below:

Table 5-1: nfs_cache_info struct Parameters

Struct Parameter	Description
`num_attrcache`	This parameter controls the total number of simultaneous file/directory entries that are 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 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; used to store the file name and its NFS File Handle. The default value for this field is 512. Each file name entry associated with the directory takes 40+`FileNameLen` bytes.

Table 5-1: nfs_cache_info struct Parameters (Continued)

Struct Parameter	Description
`num_datacache`	This parameter indicates the number of data cache headers 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 are allocated for data caching. The default value for this field is 512 (memory usage 512KB.)
`hash_tblsize`	This parameter controls the size of the hash table associated with each entry of the file attribute cache. Thus, each file attribute entry has `hash_tblsize` number of pointers to the associated directory name lookup entries (in case of directory files) or data cache entries (for regular files.) For directory files, the hashing is based on the file names associated with that directory. For regular files, the hashing is 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 is overridden by the read/write block size given at NFS mount time, for that mount. The default value for this field is 8192.
`dis_cache`	This field, if non-zero, disables 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, disables 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 are held with the client after the last modification, before enqueueing for updating with the NFS server. This field applies for regular files. The default value for this field is 10.
`acregmax`	This field specifies the maximum time in seconds that the file attributes are expected to be the same as 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.

Table 5-1: nfs_cache_info struct Parameters (Continued)

Struct Parameter	Description
`acdirmin`	This field specifies the minimum amount of time in seconds, that the directory file attributes are considered to be valid. The attributes are ready for refresh beyond this duration. The default value for this field is 30 seconds.
`acdirmax`	This field specifies the maximum time in seconds that the directory file attributes are expected to the same. 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 seconds.

Once the desired values for the tuning parameters are updated in `/sys/devices/nfs_cache_info.c`, the device library must be updated and the kernel rebuilt. These steps are described in “Customizing LynxOS” in the *LynxOS User’s Guide*.

Tuning NFS File Locking

The lock device driver facilitates *advisory file locking* over NFS. Advisory file locking works with only 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 are monitored from both client and server side. The file `/sys/devices/lock_info.c` controls the number of NFS file locking client requests that are 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`, the device library must be updated and the kernel rebuilt. These steps are described in “Customizing LynxOS” in the *LynxOS User’s Guide*.

Configuring the NFS Server

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

```
<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, the directory `/mydata` is 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 fails, 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
```

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 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` have root access to the exported directory, and only users on `shark` and `orca` have read/write capabilities. Users on `fish` can access the files only in a read-only state.

CHAPTER 6 Samba

During the initial installation of LynxOS, Samba is installed with the `installit` utility. Samba can also be configured, installed, or removed at any time after initial installation.

This chapter describes Samba basics and how to configure and run the Samba utilities to connect LynxOS to network clients.

What is Samba?

Samba is server software for computers that run under UNIX or UNIX-like operating systems with standard TCP/IP. The Samba suite is a collection of programs that implements the following protocols for UNIX:

- Server Message Block (SMB)
- Common Internet Filesystem (CIFS)

The SMB protocol is sometimes referred to as the Lan Manager or Netbios protocol.

In addition, Samba includes a program, smbclient that implements a simple ftp-like interface that lets users on LynxOS clients access file space and printers on SMB/CIFS servers.

The Samba software is licensed under the GNU Public License. The latest Samba information can be found online at:

```
http://www.samba.org
```

Installing/Removing Samba Support

Samba can be installed or removed after initial installation of LynxOS. `Install.samba` and `Uninstall.samba` scripts are found in `/usr/bin`.

Installing Samba

To install Samba, execute the following script:

```
# Install.samba
```

The Samba binary distribution is located in `/usr/samba`. The `bin` sub-directory contains the Samba executables and script files. The `docs` sub-directory contains documents and the `examples` sub-directory contains some sample `smb.conf` files.

Read the `UNIX_INSTALL.txt` and `DIAGNOSIS.txt` files to confirm that the proper prerequisites for the Samba suite are met.

For more information on Samba components, see “LynxOS Samba Components,” on the following pages.

Removing Samba

To remove Samba:

```
# Uninstall.samba
```

LynxOS Samba Components

The Samba suite includes several components. The following table lists the LynxOS Samba components.

Table 6-1: LynxOS Samba Components

Component	Definition
`smbd`	A daemon that provides the file and print services to SMB clients. The configuration file for this daemon is `smb.conf`.
`nmbd`	A daemon that provides Netbios nameserving and browsing support. This daemon can be run interactively to query other name service daemons.
`smbclient`	A program that implements an `ftp`-like client.
`testparm`	A utility that tests the `smb.conf` configuration file.
`smbstatus`	A utility that displays the use of the `smbd` server.

Each Samba component has its own `man` page. It is recommended that users read the `man` pages for these components.

Configuring the Samba Server

Samba can be configured to allow connections between LynxOS and Windows clients. LynxOS can behave like a LAN Server, Windows NT Server, or Pathworks machine. Once Samba is installed and configured, directories and printers can be exported to heterogeneous network clients including:

- LanManager
- Windows for Workgroups
- Windows NT
- Linux
- OS/2
- AIX

To enable Samba service on the server, perform the following tasks:

- Create and test the Samba configuration file.

- Start the `smbd` and `nmbd` daemons.
- Listen to the shares on the server.

Creating and Testing the Samba Configuration File

Use the following procedure to set up and test the Samba server configuration file.

1. Change to the `/usr/samba/lib` directory.

   ```
   # cd /usr/samba/lib
   ```

   The Samba configuration file must be in this directory.

2. Create the SMB configuration file, `smb.conf`.

   ```
   # vi smb.conf
   ```

   Sample configuration files can be found in `/usr/samba/examples`. A simple configuration file is provided below:

```
[global]
   workgroup = lynuxworks
   wins server = server.lynuxworks.com
   wins support = no

[tmp]
   comment = Temporary file space
   path = /tmp
   read only = yes
   public = yes

[public]
   path = /home/share
   public = yes
   only guest = yes
   writable = yes
   printable = no
```

Figure 6-1: Sample smb.conf file

   This configuration provides global settings for the Windows workgroup, wins support and allows the client to access the `/tmp` directory in read-only mode. Additionally, the path `/home/share` is set up as a read/write directory that anyone can access. The configuration file describes the runtime configuration information for the `smbd` program. See the `smb.conf` man page for additional options.

3. Test the configuration file with the `testparm` utility:

   ```
   # /usr/samba/bin/testparm smb.conf
   ```

It is important that the validity of `smb.conf` is tested using the `testparm` program. If `testparm` runs without problems, it lists the loaded services. If errors are discovered, `testparm` returns an error message.

Make sure that the Samba configuration is correct before starting the `smbd` and `nmbd` daemons.

Starting the smbd and nmbd Daemons

Start the `smbd` and `nmbd` daemons in one of two ways:

- From the `inetd.conf` file (default),
- From the command line, or
- From a setup script.

In either case, the NETBIOS Name Service and NETBIOS session service ports are monitored for serving any requests. If the Samba daemons are started from the command line, lines must be added to the `/net/rc.network` file to enable Samba

Starting smbd and nmbd from inetd.conf

NOTE: `smbd` and `nmbd` daemons can be started from the `inetd.conf` file or the command line. However, using both means of starting the daemons does not work.

To start `smbd` and `nmbd` from `inetd`, execute the `Install.samba` script.

Starting smbd and nmbd from the Command Line

To start the smbd and nmbd daemons from the command line, type the following commands:

```
# /usr/samba/bin/smbd -D
# /usr/samba/bin/nmbd -D
```

The -D option specifies the command to run as a daemon. For more information on `smbd` and `nmbd`, see the respective man pages.

Starting smbd and nmbd from a Script

`smbd` and `nmbd` can be started from a script. The advantage is that `Samba` responds quicker to an initial connection request.

Use the following procedure to start `smbd` and `nmbd` from a script.

1. Create a script to start the `smbd` and `nmbd` daemons. A sample `startsmb` script can be created with these lines:

   ```
   #!/bin/sh
   /usr/samba/bin/smbd -D
   /usr/samba/bin/nmbd -D
   ```

2. Make the script executable.

   ```
   # chmod +x startsmb
   ```

   The script can be run manually, or it can be executed from `/net/rc.network`.

3. To terminate the `nmbd` and `smbd` daemons, find them with the `ps` command and then send a `kill` signal to the `nmbd` and `smbd` daemons.

   ```
   # ps -ax | grep nmbd smbd
   # kill PID
   ```

Listing Shares on the Server

Check if the shares (exported services) are available on the server. To do so, run the `smbclient` program.

The syntax for this command is as follows:

```
# smbclient -L <hostname>
```

A list of available shares is returned. If the error message “`Bad password`” returns, then either an incorrect `hosts allow`, `hosts deny`, or `valid users` line exists in the `smb.conf` file or the guest account is not valid. Check the guest account with the `testparm` program and temporarily remove any `hosts allow`, `hosts deny`, `valid users`, or `invalid users` lines.

If the error message `connection refused` is returned, then the `smbd` server might not be running. If the `smbd` server is initialized from `inetd.conf`, the line may be malformed. If initialized as a daemon, check that the daemon is running, and that the `netbios-ssn` port is in a `LISTEN` state with this command:

```
# netstat -a
```

If an error message `“session request failed”` is returned, then the server refused the connection. If an error returns `“your server software is being unfriendly”,` then it is probably due to an invalid command line parameters set to `smbd`. Check the configuration file (`smb.conf`) for syntax errors with `testparm` and for the existence of the various directories where Samba keeps the log and lock files.

Another common cause of these two errors is if another process is running on port 139, such as Samba. For example, if `smbd` is running from `inetd` already or something like Digital’s Pathworks is running, an error is returned. Check the `inetd.conf` file before trying to start `smbd` as a daemon.

CHAPTER 7

net-SNMP and OpenSSL

Introduction

net-SNMP (previously known as UCD-SNMP) is an implementation of the SNMP (Simple Network Management Protocol) protocol.

NOTE: Though the UCD-SNMP project has been renamed to net-SNMP, the current distribution of net-SNMP files is still called UCD-SNMP.

SNMP is used to deliver network management information between networked hosts. Administrators can manage certain aspects of networking using net-SNMP, including performance management and problem detection. net-SNMP is comprised of various tools related to SNMP management, including:

- An extensible Agent
- An SNMP library
- Tools to request or set information from SNMP Agents
- Tools to generate and handle SNMP traps
- A version of the UNIX `netstat` command, using SNMP
- A Tk/perl MIB browser

LynxOS includes net-SNMP version 4.1.1 which supports SNMPv2 and SNMPv3 (see following note), and MIBI and MIBII.

NOTE: The SHA authentication and DES encryption components of SNMPv3 require the OpenSSL package. This OpenSSL package is an unmodified version of the open-source distribution built on LynxOS and is provided for the SNMPv3 encryption functionality only.

Use of OpenSSL outside of SNMPv3 is unsupported. Refer to "OpenSSL Legal Issues" on page 71 for additional legal restrictions.

---

net-SNMP sends and receives information through UDP ports 161 (SNMP) and 162 (SNMP Traps).

Installing net-SNMP

net-SNMP components are installed during the initial LynxOS installation. To enable net-SNMP functionality, users must run the `Install.snmp` script to update the `/net/rc.network file`. net-SNMP can be disabled with the `Uninstall.snmp` script.

Run the Install script to enable snmp functionality:

```
# Install.snmp
```

The `Install.snmp` and `Uninstall.snmp` scripts are found in the `/usr/bin/` directory.

SNMP Overview

SNMP architecture is comprised of three elements: Managed Devices, Agents, and Network Management Stations (NMS).

Managed devices can be any device node on a network, including PCs, Hubs, Routers, and Printers. Agents are software modules that are responsible for maintaining information on a specific device node. Agents collect and store information about a particular device in a local management database, for use by network management stations. The Network Management Station (NMS) provides a user interface to applications and network information. Network management stations collect information from Agents for the Management Information Base

(MIB), and can set the types of data the Agents report. The MIB is a hierarchical database of all managed devices on a network managed by SNMP.

The following figure shows the communication between managed devices, Agents, and the network management station.

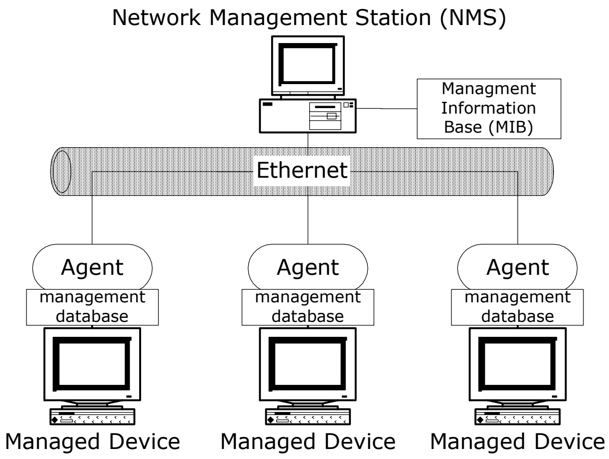

Figure 7-1: SNMP Basic Architecture

In this example, Agents act as an interface between the network management station and the managed devices on the network. Each Agent resides on the device as a software module and provides information to the NMS. The NMS maintains the Management Information Base (MIB), a hierarchical table of all entities on the network.

net-SNMP Documentation

Included with the net-SNMP distribution for LynxOS are several documents, including:

- `FAQ`
- `README`
- `PORTING`
- `EXAMPLE.conf`
- `AGENT.txt`
- man pages for individual tools, files, and the API

Additional resources and documentation are also available online at:

- `http://net-snmp.sourceforge.net`

net-SNMP Components

The following tables describe the net-SNMP daemons and applications included with the LynxOS distribution. Each of the following components is described in its respective man page.

Table 7-1: net-SNMP Daemon Components

Component	Description
`snmpd`	SNMP Agent daemon that responds to SNMP requests
`snmpd.conf`	SNMP Agent configuration file
`snmptrapd`	SNMP Trap daemon
`snmptrapd.conf`	SNMP Trap configuration file
`snmpcmd`	Common options used with SNMP commands.
`snmp.conf`	Configuration file for SNMP applications

Table 7-2: net-SNMP Application Components

Component	Description
`snmpget`	Queries information from managed devices
`snmpset`	Sets network information
`snmpwalk`	Queries for a tree of information from managed devices
`snmptrap`	Uses `TRAP` to send network information
`snmpbulkwalk`	Uses `BULK` requests to query for a tree of information from managed devices
`snmpdelta`	Monitors changes in SNMP variables
`snmpgetnext`	Uses `GET NEXT` to query for information on a managed device
`snmpnetstat`	Show network status through SNMP
`snmpstatus`	Retrieve status from a managed device
`snmptable`	Outputs an SNMP table
`snmptest`	Tests network connectivity with SNMP requests
`snmptranslate`	Translates SNMP values to other formats
`snmpusm`	Creates and Maintains SNMPv3 users on a remote managed device
`snmpbulkget`	Communicates with managed device with `BULK GET` requests

Configuring SNMPv3

Use the following instructions to set up users for SNMPv3.

Creating a User

To create an SNMPv3 user, follow these instructions:

1. Before creating the SNMPv3 user, update the `$ENV_PREFIX/usr/snmp/share/snmpconf/snmpd.conf` file to provide the users access to SNMPv3. For example:

   ```
   # cd $ENV_PREFIX/usr/snmp/share/snmp/snmpconf
   # vi snmpd.conf
   ```

   Add this line:

   ```
   rwuser <myuser>
   ```

   where `<myuser> is the name of the user account you want to create.`

2. To create the user, edit the file `$ENV_PREFIX/usr/snmp/share/snmp/snmpconf/snmpd.conf` and add the `createUser` command:

   ```
   # cd $ENV_PREFIX/usr/snmp/share/snmp/snmpconf
   # vi snmpd.conf
   ```

   Add this line:

   ```
   createUser <myuser> MD5 <my_password> DES
   ```

   where `<myuser>` and `<my_password>` are the username and password of the user account. Passwords must be at least 8 characters long.

3. Test the user account by starting SNMPv3 and running the `sysUpTime` command:

   ```
   # cd $ENV_PREFIX/usr/snmp/sbin/snmpd
   # snmpget -v 3 -u myuser -l authNoPriv \
   -a MD5 -A my_password localhost sysUpTime.0
   ```

By placing the `createUser` line in `snmpd.conf`, the password is automatically erased from the file the next time the agent shuts down. This way, only the derived secret key are remembered. Also, only the localized secret key is remembered and if the machine is broken into, it is impossible to use this localized key to get access to any of your other hosts.

Creating Additional Users

Once the first user is created, additional users can be created from the command line. The `snmpusm` command appends the new user information to `snmp.conf` so the new user can be accessed when SNMP starts.

Use the following instructions to add additional users.

1. Before starting the SNMP agent, edit the `/usr/snmp/share/snmp/snmpconf/snmpd.conf` file to add the name of the new user. For example,

   ```
   # cd $ENV_PREFIX/usr/snmp/share/snmp/snmpconf
   # vi snmpd.conf
   ```

   Add the line:

   ```
   rwuser <newuser>
   ```

   where *<newuser>* is the name of the user account you want to add.

2. Start the SNMP agent

   ```
   # /usr/snmp/sbin/snmpd
   ```

3. Use the `snmpusm` command to add a user:

   ```
   # snmpusm -v 3 -u myuser -l authNoPriv \
   -a MD5 -A my_password localhost create \ <newuser>
   myuser
   ```

   Where <*newuser*> is the name of the new user to create. A new user is created with the same password as the `myuser` account. To change the password, type the following command:

   ```
   # snmpusm -v 3 -u newuser -l authNoPriv \
   -a MD5 -A my_password localhost passwd \
   -CO my_password -CN <newpassword>
   ```

   Where <*newpassword*> is the new password for newuser.

4. Test the user by running the `sysUpTime` SNMP function.

   ```
   # snmpget -v 3 -u newuser -l authNoPriv \
   -a MD5 new_passphrase localhost sysUpTime.0
   ```

Extending the Agent with MIB modules

Custom modules can be added to extend the functionality of Agents. Refer to the documentation on AgentX, SMUX and proxied SNMP included with the net-SNMP distribution for more details. All three mechanisms use the same module API, which is described in the `AGENT.txt` file, included with the distribution.

There is also an HTML version accessible from the net-SNMP project web page (`http://net-snmp.sourceforge.net`).

The `mib2c` tool can be used to facilitate writing MIB modules. `mib2c` generates most of the necessary skeleton code from the description in the MIB file. Note that the net-SNMP suite does not currently include support for SMUX subagents.

License & Copyright

net-SNMP is free software distributed under the GNU General Public License (GPL). Other Documents and product updates related to net-SNMP are available from: `http://net-snmp.sourceforge.net`. Some of the documentation in this guide is taken from the net-SNMP FAQ, man pages, and Readme files. In some cases, content has changed for LynxOS specific environments. Unmodified versions of these documents can be found on the net-SNMP homepage.

OpenSSL

The SHA authentication and DES encryption components of SNMPv3 require the OpenSSL package, which is provided as a tarball on a separate CD-ROM in the LynxOS package. Please be aware that this version of OpenSSL is an unmodified version of the open-source distribution built on LynxOS and is provided for the SNMPv3 encryptionfunctionality only. Use of OpenSSL outside of SNMPv3 is unsupported.

Refer to the *LynxOS Installation Guide* for instructions on installing OpenSSL.

OpenSSL Legal Issues

The OpenSSL package (required for certain SNMPv3 components) uses strong cryptography, which may fall under certain import/export restrictions in certain countries. Use of strong cryptography, use of cryptography hooks, or communicating technical details about cryptography software is illegal in some countries. Please be aware of any import/export and/or use laws which apply.

CHAPTER 8 SCMP

SCMP (Snugly Coupled Multiprocessing) allows multiple CPU boards connected to the same backplane (VME) to communicate with each other using the socket interface and TCP/IP protocols.

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

Overview

SCMP splits complex applications across multiple processors resulting in faster overall computational throughput and real-time response. Some of the features of SCMP support include:

- Interprocess communication using socket interface and TCP/IP protocols.
- Availability of TCP/IP networking utilities like `rlogin`, `rcp`, and so on.
- 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 VME boards on the backplane.

SCMP 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, SCMP connections looks like a Ethernet or SLIP connections.

The following figure shows a generic network layer model with SCMP.

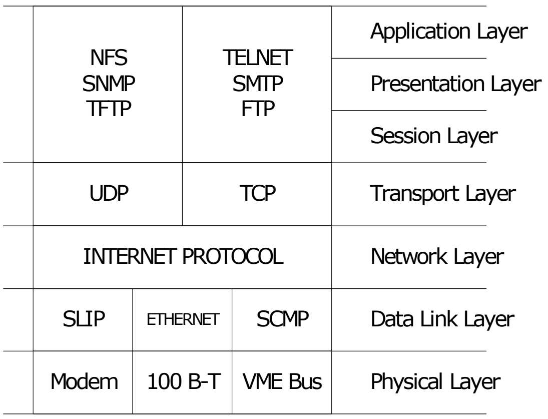

Figure 8-1: Generic Network Model based on OSI/ISO

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 bus has a local backplane disk with LynxOS installed. To configure diskless clients for SCMP, see "Starting SCMP" on page 85.

The *backplane network driver* is responsible for transferring packets over the 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 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 backplane using sockets.

The following figure shows a hypothetical configuration of VME processors that communicate via SCMP over a shared VME backplane.

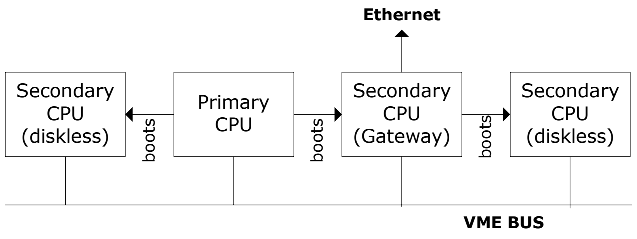

Figure 8-2: Hypothetical SCMP Configuration

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

Configuration

When configuring an SCMP environment, it 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).

SCMP - VME

Bus Example Used in This Chapter

For simplicity, this chapter uses an example SCMP setup (shown in the following example) to describe how to configure an SCMP environment.

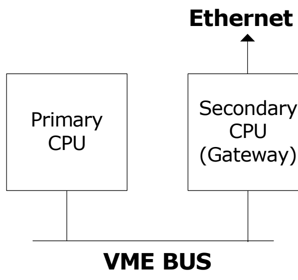

Figure 8-3: Example SCMP Configuration

Setting Up the Hardware for SCMP

LynxOS SCMP supports up to eight VME boards sharing a single VMEbus. Boards of one architecture type can be included (PowerPC-only). LynxOS currently supports the following CPU boards and VME bridge chips for SCMP:

Table 8-1: VMEbus Driver

Board	VME Bridge Chip	LynxOS VME Driver
MVME 5100	Universe	`Uvme`
MVME5101		

Documentation should be provided for the particular processor boards and the VME cage available during the initial setup of the hardware for SCMP. Before installing SCMP, review VMEbug commands and concepts.

NOTE: In this chapter, the term VMEbug is used to generically refer to the Bug Monitor of the PowerPC system.

Assembling the System

Select one processor board to be the *system controller* on the VMEbus. To do this, set the appropriate jumpers on the board (as described in the hardware manual) and place the board in slot 0 of the VME cage. Some VME chassis may require the boards to be in 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 processor boards communicate by accessing each other's DRAM in the VMEbus A32/D32 address space. They also communicate by using the Universe SW-INT interrupt. To configure an SCMP environment, assign a unique address to each board so 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. Using the MVME 5100 as an example, the following figure illustrates how shared memory is accessed through VME space. This example assumes there are two boards (board 1 and board 2) in the SCMP setup, both making the top 0x03FFF000 bytes of their DRAM available to the VME space.

Note that for each processor board, VME space address 0x0 is accessible at processor address 0xD0000000. Therefore in Figure 8-5 , where board 2's DRAM is mapped to VME address 0x04000000, it is available to board 1 at 0xD0000000+0x04000000. Similarly board 1's DRAM is mapped to VME address 0x0 and available to board 2 at address 0xD0000000+0x0.

The following figure illustrates how board 1's DRAM is mapped to VME space and where it is visible in the board 2's address space.

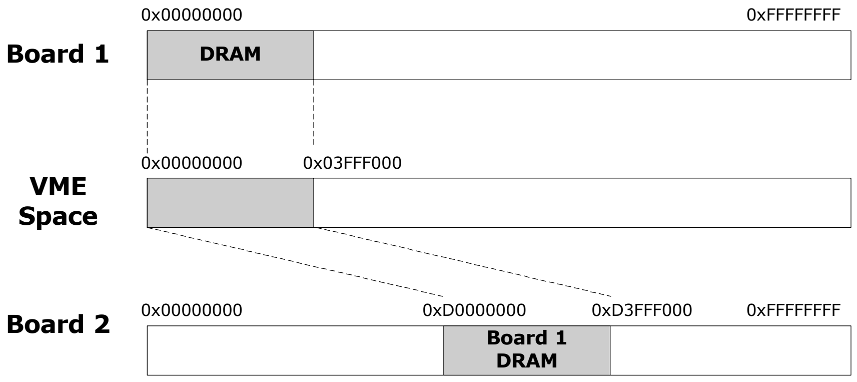

Figure 8-4: Board 1 to Board 2 Shared Memory Example

The following figure illustrates how board 2's DRAM is mapped to VME space and where it is visible in the board 1's address space.

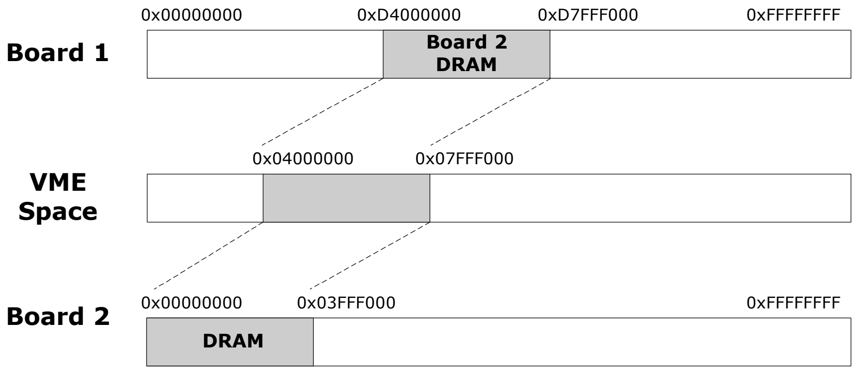

Figure 8-5: Board 2 to Board 1 Shared Memory Example

For each processor in the SCMP environment, select and program the following addresses into the VME device. In case of Universe chip, the VME device installed on MVME5100 and MVME 5101 boards, this requires programming the following:

- VME Slave Image Base Address
- VME Slave Image Bound Address
- VME Slave Image Translation Offset

Selecting the Slave Addresses

Each board also must have a unique slave address. The VME Slave Image Base Address mentioned above is the starting address in VME space where a processor board's DRAM will be mapped. The VME Slave Image Bound address determines the amount of local DRAM mapped. Once DRAM of a board is mapped to the VME space at any address, Y for example, it becomes visible to all other boards in the system at physical address 0xD0000000+Y. Note that this is valid only after the system boots with LynxOS.

Programming the VME Bridge Chip

Use the addresses in the following table to program the VME bridge chip by entering a firmware-specific command at the VMEbug monitor that modifies the environment variables. Make sure that there is no overlap in the address ranges for each board.

Table 8-2: Example DRAM Start and End Address

System	Amount of DRAM mapped	Starting DRAM (Slave) Address	Ending DRAM (Slave) Address
CPU1	64 MB	`0x00000000`	`0x03FFF000`
CPU2	64 MB	`0x04000000`	`0x07FFF000`

Programming the Universe Chip

The Universe chip must be programmed on each board with the DRAM slave addresses. For primary processor and secondary processors that are booted from the disk, firmware values for VME slave images will be overwritten by the

Universe driver. Refer to `/sys/dheaders/uvmeinfo.h` for default values. For secondary boards that are booted over the backplane using the slaveboot utility, these values need to be programmed with VMEBug firmware (PPCBUG in case of MVME5100/MVME5101 boards). Below is an example of how the universe registers can be setup in the secondary processor to enable it for a slaveboot.

Example Settings

Set the VMEbus Slave Image 0 Control value to 0xC0F20000 for all MVME5100 boards in the configuration.

It is important to note that the MVME5100 and MVME5101 boards start in CHRP address mapping mode and when LynxOS boots up, it transitions the board to PReP mode. Therefore when programming the board's VME slave images from PPCBUG users need to do it in accordance with CHRP.

In CHRP mode the VME Slave Image Translation Offset is calculated by subtracting the VME Slave Image Base Address from 0x00000000. For example:

```
0x00000000 – 0x04000000 = 0xFC000000
```

In PReP mode, the VME Slave Image Translation Offset is calculated by subtracting the base address from 0x80000000. For example:

```
0x80000000 - 0x04000000 = 0x7C000000
```

The following figure uses a Slave Image Base Address Register (DRAM address) of 0x04000000. This makes the DRAM of this board available at VME address 0x04000000

```
PPC1_Bug> env
[...]
[...]
VMEbus Slave Image 0 ControlC0F20000
VMEbus Slave Image 0 Base Address Register04000000
VMEbus Slave Image 0 Bound Address Register07FFF000
VMEbus Slave Image 0 Translation OffsetFC000000
[...]
[...]
Update Non-Volatile RAM (Y/N)? Y
```

Figure 8-6: Programming the Universe Chip on an MVME2600 Board

Verifying the Hardware Setup

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

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 the example setup in this chapter.

Troubleshooting a VME Board that Uses the Universe Chip

If SCMP configuration is not working, verify the hardware in the system is correct. Check the firmware in the system as described below. Assume that there are two boards in the SCMP setup.

1. On both boards in the SCMP configuration, program the following values for the PCI Slave Image #1 with the `env` command.

```
PCI Slave Image 1 Control Register            C0820000
PCI Slave Image 1 Base Address Register       80000000
PCI Slave Image 1 Bound Address Register      90000000
PCI Slave Image 1 Translation Offset Register  800000000
```

   The above translation indicates that PCI address range 0x80000000-0x90000000 is mapped to VME address 0x80000000 + 0x80000000 = 0x0. Since the board is in CHRP mode, PCI address 0x80000000 is also CPU/physical address 0x80000000. Therefore any access to Physical address range 0x80000000-0x90000000 will translate to VME address range 0x0-0x10000000.

2. Program the VME slave images on all boards such that DRAMs of all boards are available at unique VME addresses. Unique VME addresses assigned to top portion (64MB in this example) of DRAM on all boards must lie within the range 0x0-0x10000000. For example on board 2 program the following values into the VME slave image:

```
VMEbus Slave Image 0 Control                  E0F20000
VMEbus Slave Image 0 Base Address Register    04000000
VMEbus Slave Image 0 Bound Address Register   07FFF000
VMEbus Slave Image 0 Translation Offset       FC000000
```

   This maps the DRAM of this board to VME address range 0x04000000-0x07FFF000. The first board can now access this board’s DRAM at address 0x84000000. Similarly setup board 1's DRAM to be visible at VME address range 0x0000000-0x03FFF000.

3. Use the PPC1Bug `md` (memory dump) command to verify that the boards can access each other's memory across the VMEbus.

   Can the board 1 see board 2?

   ```
   [board1]PPC1-Bug> md 80000000 20
   ```

   ```
   [board2]PPC1-Bug> md 0 20
   ```

   See the PPC6-Bug firmware documentation for more information on the `md` command.

4. Now we need to make sure the boards can see each other's Universe chip registers. On board 2, setup the following values in the VMEbus Register Access registers with the PPCBUG env command:

```
VMEbus Register Access Image Control Register      80F20000
VMEbus Register Access Image Base Address Register  07FFF000
```

On board 1, set the base address to 0x03FFF000. Can the boards see each other's universe registers?

```
[board 1]     PPC6-Bug> md 87FFF000 20
```

This should show the vendor id / device id combination of the Universe chip on board 2. Note that 0x07FFF000 is a VME address and 0x87FFF000 is the CPU address at which this VME address is visible. Similarly, board 1's Universe registers should be visible to board 2 at address 0x83FFF000.

Setting Up LynxOS for SCMP

This is the software stage of the SCMP setup. At this point, these steps must be performed:

- Configure the LynxOS kernel with the VME backplane driver.

This section assumes that LynxOS is already installed on the local disk of each VME board in the SCMP environment. If LynxOS is not installed, 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 a diskless secondary processor and a disk-based primary processor are used, configure the primary processor as described in this chapter. Then refer to Chapter 9, "Network Booting Diskless Clients with LynxOS" on page 105.

Copying the Current LynxOS Kernel

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

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

If there are problems, this copy of the kernel can be booted instead. See the `preboot` man page for details on booting alternate kernels.

Choosing the Primary Processor

Select the board to be the primary processor in the SCMP environment. All processor boards contains a database of information about all processors in the SCMP system. This database is contained in the `/etc/bplane.conf` file. The first entry in this database belongs to the primary processor. The following information is required for each processor in the system:

- Hostname
- DRAM starting address
- VRAI Board Base Address (Universe chip)

Set the VRAI base address in the A32 address space.

The following table provides an example of the kind of information that is required to supply for each processor:

Table 8-3: Information Required for Each Processor

Hostname	DRAM Start Address	Combined IO Addresses
scmp-cpu1	`0x00000000`	`0xD3FFF000`
scmp-cpu2	`0x04000000`	`0xD7FFF000`

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. The following figure shows an example `/etc/bplane.conf` file.

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

   9    0        0x00000000    0xD3FFF000 192.1.1.1
   9    0        0x04000000    0xD7FFF000 192.1.1.2
```

Figure 8-7: Example bplane.conf File

The `/etc/bplane.conf` fields are as follows:

- **Board Type** -- `9` refers to the MVME5100 and MVME5101 boards.
- **Boot Flags** -- Unused field; must be set to 0.
- **DRAM Address** -- The desired starting address for the board as specified with the `env` command.
- **I/O Address** -- CPU address where VME bridge chip registers of this board are visible.
- **Hostname** -- Unique IP address or hostname for each board. When using the hostname, ensure that `/etc/hosts` contains the hostname with its appropriate IP address.

For more information on `bplane.conf` file, see its man page.

Modifying the Backplane Driver Information

1. Move to the `/sys/devices` directory:

   ```
   # cd /sys/devices
   ```

2. Edit the `if_bplaneinfo.c` file.

   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 Information

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 are modified, rebuild the kernel to incorporate the changes:

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

Choose the `SCMP` option and answer the questions presented by the script.

This step allows the user to enable the bplane and VME chip drivers.

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 processor boards on the backplane. These values should correspond to the values specified in `/etc/bplane.conf`.

Starting SCMP

After rebuilding the kernel, reboot the primary processor. This brings up the SCMP-configured kernel on the primary processor. The `-N` option remakes the `/dev` directory with the device `bplane` added.

```
# reboot -aN
```

NOTE: The primary processor must be running SCMP before the secondary processor(s) can run SCMP. To boot a client processor, please refer to Chapter 9, “Network Booting Diskless Clients with LynxOS” on page 105.

If the primary processor is disk-based and is exporting file system for the secondary processors, follow steps 1 through 3 below:

1. Enable TFTP Services:

   - Uncomment the line starting with `tftp` in the file `/etc/inetd.conf`.

   - Kill the process `/net/inetd`:

   the command `ps -ax | grep inetd` gives the process id of the `inetd` process. Use this process id to kill the process and restart it.

   ```
   # ps -ax | grep inetd
   # kill -1 <process_id>
   ```

2. Export root directory for the diskless client via NFS:

   For example, if `/clients/cpu2` is the directory containing, or linked to the root file system meant for the diskless client, this directory must be exported.

   A)Add the following line to the file `/etc/exports`:

   ```
   /clients/cpu2 root=<client_processor>
   ```

   where <*client_processor*> is the diskless client processor importing the root file system from the server.

   B)Export the directory to the client:

   ```
   # exportfs -av
   ```

   For more information, see “Ethernet Netboot” on page 91.

3. Build diskless client configuration files, the SCMP-bootable image and the necessary start-up files:

   The SCMP netbootable image is built in the directory `$ENV_PREFIX/sys/romkit/scmp-boot`.

NOTE: `ENV_PREFIX` is `/` for native development systems.

The template file `spec.tmpl` is used to create the `spec` file for `mkimage: ppc-x.netboot.spec`, where `x` is the `CPUID`, which is 1 by default. This spec file can be edited to add/delete any item(s) from the RAM disk.

C)Configure the diskless client:

```
# cd ENV_PREFIX/sys/romkit/scmp-boot
# make config
```

This asks a few questions in order to configure the diskless client's image. It creates the directory sys-1 where the kernel for the diskless client is copied.

D)Build the SCMP netbootable image for the diskless client:

```
# make all
```

E)Install the image in the download directory (`/tftboot` by default):

```
# make install
```

For configuring and building images for multiple clients, one needs to specify the variable `CPUID` with the make command:

```
# make CPUID=<x> config
# make CPUID=<x> all
# make CPUID=<x> install
```

where <*x*> is the client id. When CPUID is not specified, it defaults to one.

To manually boot the client processor, type the following at the firmware prompt of the client processor:

```
Motorola PowerPC client:
PPC1-Bugs go 4020
```

CHAPTER 9 *Network Booting Diskless Clients with LynxOS*

LynxOS supports network booting (netboot) of remote machines over a TCP/IP network. This chapter describes how to netboot diskless clients over an Ethernet or an SCMP connection.

Overview

Netboot allows a configured LynxOS kernel image to be downloaded onto a machine without a hard disk. The kernel image can be downloaded in one of these ways:

- Ethernet (see page 91)
- PXE Netboot (see page 103)
- Snugly Coupled Multi-Processing (SCMP) (see page page 105)

These diskless clients can also access remote file systems over the network. See the *LynxOS Hardware Support Guide* (available on the Documentation CD-ROM and the lynuxworks website) for the list of hardware configurations that support network booting.

Booting a diskless client involves these tasks:

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

These tasks apply to all netboot protocols.

Using the LynuxWorks Netboot Scripts

LynuxWorks provides ready-to-use scripts for both Ethernet and SCMP network booting. These scripts are in the following locations:

- `/sys/romkit/remote_boot` (Ethernet)
- `/sys/romkit/scmp_boot` (SCMP)

LynuxWorks created these scripts to produce a typical configuration for a diskless client. They can be used in their default configuration or as a template for other applications.

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

These netboot scripts should be copied before editing them. Use the following procedure to make a copy of the netboot scripts.

1. Move to the directory that contains the netboot scripts.

   ```
   # cd /sys/romkit
   ```

2. Make a copy of the netboot scripts.

   ```
   # cp -r remote_boot ethernet_client
   ```

   This produces 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 on either a LynxOS native system or a compatible cross-development system. For example, to create a netboot image for a diskless client, the scripts must be executed on a disk-based LynxOS native system or a supported LynxOS cross-development system.

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.

Before Beginning

The easiest way to netboot a diskless client with LynxOS is to use the scripts in the following location: `/sys/romkit/remote_boot`

These scripts assume that a single disk-based LynxOS system is being used for the diskless client via NFS to perform the following tasks:

- Build the netboot image.
- Boot the diskless client.
- Serve the root file system.

Before running the netboot scripts, make sure that the disk-based server is of the same architecture type (x86, PowerPC) as the intended diskless client. Or, on a cross-development host, make sure that the appropriate LynxOS tools are available for the target architecture.

Before proceeding with the scripts, information about the disk-based server and diskless client(s) is required. The following table provides a list of this information, with examples.

Table 9-1: 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-Z a-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, `192.1.1.2` is a valid IP address.

Table 9-1: Server and Client Information Needed to Configure Netboot (Denoted by X)

Parameter	Disk-Based Server	Diskless Client	Notes
Ethernet Interface	X	X	The LynxOS device name for the network interface card on the diskless client. The `intro` man page 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 (PowerPC clients) to find the Ethernet address. On x86 clients, look for a label on the Ethernet card. On Solaris clients, use the `ifconfig ie0` command.

Configuring the Disk-Based Server

To configure the disk-based server, perform the following tasks:

- Enable TFTP services.
- Export the root directory for diskless client via NFS.
- Build the 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 tasks involve. For additional information, See “Example—Netbooting a FORCE PowerCore 680 Board” on page 98.

NOTE: TFTP is a potential security risk. Use caution when configuring it.

For instructions on configuring PXE support, see “Configuring PXE Netboot Support” on page 103.

Enabling TFTP for LynxOS

The following instructions are for LynxOS native development systems. TFTP cross development system instructions vary. See "Configuring TFTP on Cross Development System" on page 93.

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

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

If this information is added, a `SIGHUP` signal must be sent to the `inetd` process. Find the process ID using the `ps` command. Then send the `SIGHUP` signal with the `kill` command.

1. Find the process id for `/etc/inetd.conf` with the `ps` command.

   ```
   # ps -ax | grep inetd
   ```

2. Send the `SIGHUP` signal to the `inetd` process with the `kill` command.

   ```
   # kill <process_id>
   ```

Configuring TFTP on Cross Development System

TFTP instructions differ on different cross development systems. Solaris TFTP configurations are similar to LynxOS. Refer to the Solaris `tftp` man page for instructions.

On Linux systems, users must edit the `tftp` file in `/etc/xinetd.d/` and restart internet services with `xinetd`. Refer to the `xinetd` and `tftp` man pages for instructions. Sample instructions for configuring TFTP on a Linux system is provided in "Example—Netbooting a FORCE PowerCore 680 Board" on page 98.

Windows does not include a TFTP server in its distribution. LynuxWorks includes a shareware TFTP server on the Cross Development Kit (CDK) distribution. Proceed as follows:

1. Install the CDK CD-ROM on the Windows cross development host. Refer to the section, "Windows Cross Development Installation" in the *LynxOS Installation Guide*.
2. Consulting Windows documentation, set up TCP/IP support on the host.
3. Locate the file `c:\Lynx\4p0p0\tftppro.zip`, the shareware TFTP server from WaluSoft in zipped format. Follow instructions provided with the WaluSoft TFTP distribution to set up TFTP service. Pay special

attention to the outgoing server path from the **Setup** menu; it should point to the host CD-ROM.

Exporting the Root File System Via NFS

The disk-based server must be set up so that it can provide a `root` file system to the diskless client. The server can then export this file system via NFS. When using the default settings for the netboot scripts, certain directories and files are required in the `root` file system for the diskless client.

Use the following procedure to export the `root` file system to the diskless client:

1. Copy the following directories from the LynxOS distribution to the new `root` file system directory.

```
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/
```

2. (Optional) In addition, the following utilities can be copied:

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

3. Export the `root` file system with the `exportfs` command.

```
# exportfs -av
```

For more information on the `exportfs` command, see `exportsfs` man page.

Building the Netboot Image

The `Makefile` in `/sys/romkit/remote_boot` builds a client-specific configuration file for each diskless client. LynxOS uses this configuration file to build the kernel image and supporting files that it downloads 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, perform these steps:

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

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

For instructions on configuring PXE support, see "Configuring PXE Netboot Support" on page 103.

Setting Up the PowerPC System

For the PowerPC system, the network booting code is available as a programmable option in the `env` settings of the VMEbug Monitor.

Use the following procedure to set this parameter.

1. Reboot/reset the CPU board.
2. Enter the Bug Monitor (this typically involves pressing the **Break** key during CPU reset).

Enabling NVRAM on the PowerPC

In newer PowerPC boards, the real-time clock is disabled. The real-time clock must be enabled before netbooting LynxOS. Use the `set` command at the VMEbug prompt (abbreviated `xxxx-Bug`) to set the time, thus enabling the clock:

```
# set MMDDYYhhmm
```

Disabling PReP-Boot on PowerPC

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

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

Setting Netboot Parameters

NOTE: The boot filename is normally just the filename. If the full path is given, it may fail to load.

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

Only a few parameters need to be changed. For more details about the commands, consult the Motorola VMEbug documentation.

```
xxxx-Bug> niot

Client IP Address = [192.1.1.2] ?
Server IP Address = [192.1.1.1] ?
...
(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 following table shows the boot file load and execution addresses, which are platform-dependent.

Table 9-2: Boot File Addresses for PowerPC Systems

Target	Boot File Load Address	Boot File Execution Address
PowerPC	00004000	00004020

Booting the Remote Kernel Image

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

```
xxxx-Bug> nbo
```

Setting Up PPC PowerCore Systems

Use the following procedure to set up PPC PowerCore systems:

1. At the `PowerBoot` prompt, enter the following command for an individual boot-image:

```
PowerBoot> netload <netboot_file> 400000 <target_IP> \
<host_IP>
```

where *<host_IP>* is the Host IP address and *<target_IP>* is the target system IP address.

NOTE: Depending on how the server is configured, the path may or may not require the full path to the netboot image (**/tftboot/***<netboot_file>*).

2. After the download finishes, enter the following command:

```
PowerBoot> go 400020
```

Setting Up Thales VMPC Systems

Thales VMPC boot firmware netboot a PreP image using the `bootp` protocol. This requires the server to run the `bootpd` daemon with the `/etc/bootptab` file populated with the appropriate information on the netbootable image. For more detailed information, see the man page on `bootp` on the server machine.

The netbootable image generated by `mkimage` would be run through the `mkbootprep` utility in order to create a prep-bootable image for Thales machines.

Setting Up Thales VMPC

Use the following procedure to set up Cetia VMPC:

1. Make sure that the `/etc/bootptab` file on the server has an entry for the required image, and that the `bootpd` daemon is running.
2. At the VMPC Bug prompt, enter the following command:

```
COMMAND> ebop
```

Example—Netbooting a FORCE PowerCore 680 Board

The following table shows an example ethernet netboot configuration that is used in the steps on configuring the disk-based server and diskless client for the PowerPC.

NOTE: Most of the steps to configure the disk-based server for the PowerPC and for Sun hosts (SunOS and Solaris) are the same. Any differences that do exist are explained below.

Table 9-3: Ethernet Netboot Configuration

Parameter	Disk-Based Server	Diskless Client
Platform	Linux Host LynxOS Cross Development Environment with `pc_680` BSP installed	Force PowerCore 680
Hostname	`linuxcdk`	`fpc1`
IP Address	`192.1.1.1`	`192.1.1.2`
Ethernet Address	Not needed.	`08:00:3E:23:8C:BD`

Preparing the Board

To prepare the target board for downloading, proceed as follows:

1. Obtain an Internet Protocol (IP) address and network host name for the target board. In the instructions below, `192.1.1.2` is used as an IP address and `fpc1` is used as the PowerCore network host name.

2. Obtain the Ethernet address of the PowerCore target board. This address is displayed during the power-up self test after a reboot or power cycle of the board. These instructions use `00:80:42:0E:0C:02`.
3. Follow the PowerCore manufacturer's installation instructions to power up the board, attach the serial terminal, and establish a method of inputting keystrokes to the PowerBoot firmware. Proper setup will result in the `PowerBoot>` prompt being displayed on the serial terminal.
4. Attach the PowerCore target board to the local network using the twisted pair Ethernet connector located on the front panel of the PowerCore target board.

   To set up a point-to-point network (for example, to attach directly to a laptop notebook), please remember to use an *uplink* cable. This is a Category 5 twisted pair cable that has been *crossed* for uplink. If connecting to a hub, a *regular* Category 5 cable suffices.

Configuring a Network Server

The firmware monitor on the PowerCore 680 board, PowerBoot, has built-in support for downloading bootable images over the Ethernet using TFTP (Trivial File Transfer Protocol).

Configuring TFTP on a Linux Cross Development Host

TFTP is required by the target to load the `developer.kdi`. Use the following instructions to enable TFTP on the host system:

1. Edit `/etc/ethers` to include the Ethernet address of the Target board:

```
# vi /etc/ethers
```

```
#Ethernet Address     Client Hostname
08:00:3E:23:8C:BD     fpc1
```

Figure 9-1: /etc/ethers file

2. Edit the `/etc/hosts` file to include the hostname and IP address of the Target board:

   ```
   # vi /etc/hosts
   ```

```
#IP Address      Client Hostname

192.168.1.2      fpc1
```

Figure 9-2: /etc/hosts file

3. Create the `tftpboot` directory:

   ```
   # mkdir /tftpboot
   ```

4. Enable TFTP by editing the `tftp` file

   ```
   # cd /etc/xinetd.d/
   # vi tftp
   ```

5. In the `disable` field, type "**no**" to enable TFTP.
6. In the `server_args` field type "**/tftpboot**". The following provides a sample `tftp` file.

```
# default: off
# description: The tftp server serves files using the trivial file transfer \
# protocol. The tftp protocol is often used to boot diskless \
# workstations, download configuration files to network-aware printers, \
# and to start the installation process for some operating systems.
service tftp
{
   socket_type = dgram
   protocol = udp
   wait = yes
   user = root
   server = /usr/sbin/in.tftpd
   server_args = /tftpboot
   disable = no
}
```

Figure 9-3: Sample tftp configuration file

7. Restart the `xinetd` services:

   ```
   # cd /etc/rc.d/init.d
   # ./xinetd restart
   ```

Network Booting the Target Board

Once the TFTP server is configured and operational on the host, demo Kernel Downloadable Images (KDIs) are downloaded to the PowerCore target board. These demos are located in the directory `\demo\demo.<bsp_name>` on the distribution CD-ROM.

1. Insert the LynxOS Open Development Environment (ODE) CD-ROM into the host system CD-ROM drive and mount it.
2. Copy the `hello.kdi` file from the `/demo/demo.pc_680` directory of the distribution CD-ROM into the `/tftpboot` directory on the TFTP server so that it can be downloaded from there onto the target board.
3. Download the demo KDI to the PowerCore target board. At the `PowerBoot>` prompt on the target, type:

   ```
   netload hello.kdi 400000 <target_IP> <host_IP>
   ```

   where

<target_IP>	IP address of the target
<host_IP>	IP address of the server

   For example:

   ```
   PowerBoot> netload hello.kdi 400000 192.168.1.1 \
   192.168.1.2
   ```

NOTE: If the security feature of TFTP is disabled or not supported on the server, the file name is prepended with the `/tftpboot` path component. Thus, in the example above, `/tftpboot/hello.kdi` is specified in this case.

4. Boot the demo KDI on the PowerCore target:

   ```
   PowerBoot> go 400020
   ```

   The following output is displayed on the console:

```
LynxOS POWERPC Version 4.0.0
Force PowerCore PC_DRM board support package
Copyright 1997-2002 LynuxWorks Inc.
All rights reserved.

LynxOS (ppc) created Tue Jan 02 19:49:10 PDT 2002

hello_world from C - reset machine or cycle the power to end this demo
hello_world from C++ - reset machine or cycle the power to end this
demo
```

Reset the power to end the demo.

The preceding process can be used to download a custom LynxOS KDI as well as other KDIs supplied with the distribution.

Since the image is in RAM, it is volatile and is erased as soon as the board is reset. To keep a static version of the image, it must be burnt into target board flash memory. The next section details this procedure.

Booting from Flash Memory

The previous procedure is useful for booting a target board quickly. Because the image is loaded into target RAM, it is volatile and is erased as soon as the board is reset.

To keep the image permanently on the target board, it is burned into target flash memory. The following outlines this procedure:

1. Erase flash bank #1, `user_flash1`, on the target board:

   ```
   PowerBoot> ferase user_flash1
   ```

2. Download the demo KDI to the PowerCore target board. At the `PowerBoot>` prompt on the target, type:

   ```
   netload hello.kdi 400000 <target_IP> <host_IP>
   ```

   where

<target_IP>	IP address of the target
<host_IP>	IP address of the server.

NOTE: If the security feature of TFTP is disabled or not supported on the server, the file name is prepended with the `/tftpboot` path component. Thus, in the example above, `/tftpboot/hello.kdi` is specified.

3. Program the image into flash bank #1:

   ```
   PowerBoot> fprog user_flash1 400000
   ```

4. Boot the demo KDI from the PowerCore target's flash memory

   ```
   PowerBoot> go ff000020
   ```

The following output is displayed to the console:

```
LynxOS POWERPC Version 4.0.0
Force PowerCore PC_DRM board support package
Copyright 1997-2002 LynuxWorks Inc.
All rights reserved.

LynxOS (ppc) created Tue Jan 02 19:49:10 PDT 2002

hello_world from C - reset machine or cycle the power to end this demo
hello_world from C++ - reset machine or cycle the power to end this
demo
```

Reset the power to end the demo.

Now, every time the board is reset, boot the image by entering the following command at the `PowerBoot>` prompt:

```
PowerBoot> go ff000020
```

The above process can be used to download a custom LynxOS KDI as well as other KDIs supplied with the distribution.

Configuring PXE Netboot Support

Network cards enabled with the Prebooting Execution Environment (PXE) specification provide x86 client systems the ability to boot software images without having to reconfigure hardware or burn EPROMs. PXE-enabled network cards receive network configuration from a LynxOS PXE server via DHCP. Network configurations include local IP address, DNS, and Router information. The client also receives a boot loader file. The boot loader file locates the system serving the kernel image, and downloads it to the client system using `tftp`.

The following sections describe the steps required to configure the PXE Server and Client systems.

NOTE: PXE Netboot support is available for x86 systems only.

Configuring the PXE Client

Configuration of the PXE client varies depending on the hardware used. Configure the PXE network card according to the specifications provided by the card's manufacturer.

Configuring the PXE Server

Configuring a LynxOS system as a PXE server involves configurations in several areas, enabling DHCP, and enabling TFTP. Use the following sections to configure the system to respond to PXE requests.

Configure DHCP

DHCP must be enabled and running to provide PXE support. Information on configuring DHCP is provided in Chapter 4, “DHCP” on page 41. To configure DHCP to accept requests from PXE clients, it is necessary to specify two options in `dhcpd.conf` that allow for PXE netbooting:

```
option dhcp-class-identifier
“PXEClient”;
allow booting;
allow bootp;
filename ”/tftpboot/<lynxOS.0>”;
```

Figure 9-4: dhcpd.conf addition for PXE devices

Where <*lynxOS.0*> is the filename of the bootloader.

The second option requires a hexadecimal string that must exist on the same line as the `option` statement with no spaces or line breaks:

```
option vendor-encapsulated-options
09:0f:80:00:0c:4e:65:74:77:6f:72:6b:20:62:6f:6f:74:0a:07:00
:50:72:6f:6d:70:74:06:01:02:08:03:80:00:00:47:04:80:00:00:0
0:ff;
```

Figure 9-5: dhcpd.conf hexadecimal string addition

The `dhcpd` daemon must be restarted after editing `dhcpd.conf`:

1. Find the Process ID (PID) of the DHCP daemon (`dhcpd`) with this command:

   ```
   # ps -axon | grep dhcpd
   ```

2. Use the `kill` command to stop the process.

   ```
   # kill PID
   ```

3. Restart the DHCP daemon:

   ```
   # /net/dhcpd
   ```

Configure tftp

`tftp` must be enabled and running for the PXE client to receive the kernel image. For information on configuring `tftp`, see “Enabling TFTP for LynxOS” on page 93.

Configure Bootloader, KDI, and preboot files

The bootloader file, along with the KDI (Kernel Downloadable Image) and modified preboot files must be accessible via `tftp`. These files should be placed in the `tftp` root directory (`/tftpboot`) as:

- *lynxOS*.0 (bootloader)
- *lynxOS*.1 (preboot)
- *lynxOS*.2 (KDI)

Where *lynxOS* is the basename of the loader and image files. By default, the basename is `lynxOS`, but the file can be renamed, if desired. However, the file extensions (.0, .1, .2) must be used consistently. If, for example, a user wanted to rename the file to the hostname of the system, it must be renamed `hostname.0`, `hostname.1, and hostname.2`.

NOTE: The Kernel Downloadable Image (KDI) file, `lynxOS.2`, must be created by the user. Refer to the *LynxOS User's Guide* for more information on creating a KDI.

SCMP Netboot

The Netboot facility for systems using SCMP is very similar to the Ethernet version. The main differences are as follows:

- SCMP runs only on the following platform:

  PowerPC --MVME5100
- Boot image sharing is not possible.

- A set of tools is necessary to operate SCMP (see `bpconfig`, `slaveboot`, `bplane.conf`).

NOTE: Before booting a diskless client over a SCMP network, configure the SCMP network as described in Chapter 8, “SCMP” on page 73.

How Does It Work?

As in Ethernet netboot, a diskless client is booted in two steps. The tools and protocols, however, differ slightly as follows:

- The server downloads the boot-image using the `slaveboot` command. This automatically starts the client.
- The boot procedure establishes the client as an SCMP network node, determines its `root` file system server, and then loads the LynxOS kernel.

Configuring the Disk-Based Server

As in Ethernet netboot, the following steps need to be performed 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 See “Ethernet Netboot” on page 91. 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/scmp_boot` builds a client-specific configuration file for each diskless client. This configuration file is used to build the kernel image and supporting files to be downloaded over the network to the diskless client.

```
# cd /sys/romkit/scmp_boot
# make config
```

The following figure shows an example SCMP netboot configuration.

```
Choose the bsp for which the scmp image is needed
   1: [ gen1 ]
   2: [ pp_drm ]
   3: [ cpci_drm ]
   4: [ pc_drm ]
   5: [ mvme5100 ]

   q/Q: Quit

enter selection: 2

SCMP Netboot Configuration
**************************
TFTP-Server Configuration
=========================
Please specify the location of the TFTP-directory
[/tftpboot]:<Return>
Enter the IP-Address of the TFTP-Server (aaa.bbb.ccc.ddd) [192.1.1.1]:<Enter
IP>

   Checking IP-Address 192.1.1.1...ok.

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]:
192.1.1.2
   Checking IP-Address 192.168.1.80...ok.
Hosts database to use on this client [/etc/hosts]:
   Writing file netboot.config-1...ok.
```

Figure 9-6: Example, SCMP Netboot Configuration

Multiple clients can be configured 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, the IP address in dot-notation can also be used).

If `slaveboot` is used to copy the image, start the client by typing at the prompt:

```
# Go <start_addr.nhex>
```

where `<start_addr.n_hex>` is 4020 for Motorola boards and 400020 for PowerCore.

For more information on the `slaveboot` command, see the `slaveboot` man page.

Troubleshooting

- When booting the client a message "`inetd: I/O error`" appears.

  The client's root (`/`) file system is not exported with `root` permissions. For example, if the client's name is `flipper`, the `/etc/exports` file should contain:

  ```
  /clients/server_ppc root=flipper
  ```

- The downloaded boot-image cannot be executed.

  This can happen if separate machines are 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 does not work.

  Make sure that `/etc/bplane.conf` file contains IP addresses only (not symbolic hostnames). The `/etc/hosts` hosts database is downloaded *after* the initial SCMP configuration If symbolic hostnames are required in `/etc/bplane.conf`, modify the SCMP kernel specification file to include a hostname database.

Advanced Issues for Ethernet Netboot

The LynxOs Ethernet Netboot build environment supports shared boot-images. This reduces the required disk space and makes kernel updates or changes easy, because they automatically apply to all clients sharing the same boot-image.

Make sure that all of the clients can actually share the same boot image. For example, a boot image cannot be shared between several x86-based systems because they may have different network controllers.

Sharing a Boot Image

Use the following procedure to create shared boot-images.

1. For all clients, select the shared-KDI option when running `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 `make all` and `make install`, `make kdi and make install_kdi` are added.

2. For *only one* client, enter the following command to build and install a sharable boot image:

   ```
   # make install_kdi
   ```

3. For *multiple clients,* enter the following command to build and install a sharable boot image:

   ```
   # make [CPUID=xxx] install
   ```

   For more information on *`CPUID`*, see See “Configuring a Second Client” on page 112..

Cleaning Up the Working Directory

During development (or after) the working directory can be cleaned up. The following `make` command deletes the kernel build subdirectory.

```
# make clean
```

Client Information Files

When running the `make config` script, it creates a dedicated subdirectory, `sys-`*CPUID* for each client. In this directory, `make config` creates the following files:

- Kernel
- Boot-image
- Kernel-specific files
- Client-specific files

The `make config` script keeps each client's configuration information in the `netboot.config-`*CPUID* file. 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

Depending on the architecture of the system, edit the `<`*`target`*`>.netboot.spec` file and enter files and directories as needed. See the `mkimage` and `mkimage.spec` man pages for the 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 `rc.sh` file is a self-extracting script that creates the final version of the file containing the network interface name. The `rc.network` file 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`
- `/etc/hosts`

The file system description database, `/etc/fstab`, is built from `fstab.sh`. This is a self-extracting script that creates the final `fstab` with the proper server and pathnames.

Adding Client-Specific Files

The following files must be edited to include another client-specific file:

```
Config-Netboot

Makefile

tftplist
```

Config-Netboot

Use the following procedure as an example to modify a hostname database.

1. Edit `Config-Netboot`.

   ```
   # vi Config-Netboot
   ```

2. Add a default value for the file in the header section and provide an empty string if there is no default.
3. In the `Diskless Client Configuration` section, add the following code:

   ```
   use_default client_hosts_file

   promptstr "user prompt" $<def_var_name> <var_name>

   echo ""
   ```

Makefile

The `TFTPFILES` variable contains the list of files needed to install as client-specific files. Add the file name here. If the file is specified it in `Config-Netboot`, should now be refer to using `$(var-name)`.

tftplist

Use the following procedure to modify the `tftplist` file.

1. Edit the `tftplist` file.

   ```
   # vi tftplist
   ```

2. Add another line to this file.

   Provide the file name, the TFTP-server (without the path and IP address), and the final destination in the RAM disk as included by the `ramdisk` driver.

3. (Optional) Add the access permissions for the file.

Other Options

Self-extracting scripts can be used to produce a file that uses one or more of the configuration values. See the `rc.sh` and `fstab.sh` files on how to do this. The `Makefile` contains generic rules how to run those scripts, so only one file needs to be added `$(SYSDIR)/myscript` to the `TFTPFILES` variable and a `myscript.sh` file in the build directory.

Configuring a Second Client

When running `make config`, the script saves the information that entered in the `netboot.config-`*`CPUID`* file. The *`CPUID`* file extension distinguishes multiple diskless clients and can be any unique string or number.

1. To configure a second client with a different *`CPUID`*, enter the following command:

   ```
   # make CPUID=2 config
   ```

2. A boot image for this and other additional clients must be specified on the command line.

   ```
   # make CPUID=2 all
   ```

CHAPTER 10 *Raw Ethernet Support*

Raw Ethernet Support is a facility available with LynxOS used 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 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 multicast 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 Support

AF_RAWETH Description

`AF_RAWETH` includes sockets of type `SOCK_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 fail with 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_SET_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 as follows:

- 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, TCP/IP packets can be sent to a raw Ethernet socket by setting other parameters such as the protocol range, filter pattern, or multicast 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 6 bytes long, and the bit-0 of first byte should be zero. This conforms to the Ethernet/IEEE 802.3 specification, which defines the distinction between physical and multicast 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 source 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 multicast addresses which the interface receives as the destination node. Thus, data packets can be sent to a set of nodes, where a certain multicast address is acceptable. This address is used as the destination station address in such packets. According to the IEEE802.3 specifications, a multicast 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 multicast address. The interface needs to be programmed to accept a specific multicast 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. 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. The example below tests for non-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 gets the station address 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 filtering for the incoming packets with respect to the Ethernet header (first 14 bytes.) For example, the received packets falling 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 socket receive buffer.

By default, the filter is deactivated (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);
};
```

APPENDIX A Supported Networking RFCs

The following table describes the supported RFCs for this release of LynxOS.

Table A-1: Supported RFCs

RFC/IEEE	Description
3083	Baseline Privacy Interface Management Information Base for DOCSIS Compliant Cable Modems and Cable Modem Termination Systems
3008	Domain Name System Security (DNSSEC)
3007	Secure Domain Name System (DNS) Update
2931	DNS Request and Transaction Signatures (SIG(0)s)
2930	Secret Key Establishment for DNS (TKEY RR)
2929	Domain Name System (DNS)
2915	The Naming Authority Pointer (NAPTR) DNS Resource Record
2894	Router Renumbering for IPv6
2874	DNS Extensions to Support IPv6 Address Aggregation and Renumbering
2863	The Interface Group MIB
2845	Secret Key Transaction Authentication for DNS (TSIG)
2819	Remote Network Monitoring Management Information Base
2796	BGP Route Reflection
2790	Host Resource MIB
2789	Mail Monitoring MIB
2788	Network Services Monitoring MIB
2742	Definitions of Managed Objects for Extensible SNMP Agents
2741	Agent Extensibility (AgentX) Protocol version 1

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
2740	OSPF for IPv6
2737	Entity MIB (version 2)
2711	IPv6 Router Alert Option
2710	Multicast Listener Discovery (MLD) for IPv6
2672	Non-Terminal DNS Name Redirection
2671	Extension Mechanisms for DNS (EDNSO)
2667	IP Tunnel MIB
2663	IP Network Address Translator (NAT)
2616	Hypertext Transfer Protocol (HTTP) 1.1
2594	Definitions of Managed Objects for www services
2593	Script MIB Extensibility Protocol Version 1.0
2592	Definitions of Managed Objects for the Delegation of Management Script
2591	Definitions of Managed Objects for Scheduling Management Operations
2588	IP Multicast and Firewalls
2580	Conformance Statement for SMIv2
2579	Textual Conventions for SMIv2
2578	Structure of Management Information Version 2 (SMIv2)
2576	Coexistence between version 1, version 2, version 3 of the Internet-standard Network Management Framework
2575	View-based Access Control Model (VACM) for the Simple Management Protocol (SNMP)
2574	User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)
2573	SNMP Applications
2572	Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)
2571	An Architecture for Describing SNMP Management Frameworks
2564	Application Management MIB

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
2560	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP
2559	Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2
2553	Basic Socket Interface Extensions for IPv6
2545	Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing
2539	Storage of Diffie-Hellman Keys in the Domain Name System (DNS)
2538	Storing Certificates in the Domain Name System (DNS)
2535	Domain Name System Security Extensions
2529	Transmission of IPv6 over IPv4 Domains without Explicit Tunnels
2526	Reserved IPv6 Subnet Anycast Addresses
2473	Generic Packet Tunneling in IPv6 Specification
2472	IP Version 6 over PPP
2464	Transmission of IPv6 Packets over Ethernet Networks
2463	Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification
2462	IPv6 Stateless Address Autoconfiguration
2461	Neighbor Discovery for IP Version 6 (IPv6)
2460	Internet Protocol, Version 6 (IPv6) Specification
2459	Internet X.509 Public Key Infrastructure Certificate and CRL Profile
2453	RIP Version 2
2437	PKCS #1: RSA Cryptography Specifications Version 2.0
2411	IP Security Document Roadmap
2410	The NULL Encryption Algorithm and Its Use With IPsec
2409	The Internet Key Exchange (IKE)
2406	IP Encapsulating Security Payload (ESP)
2405	The ESP DES-CBC Cipher Algorithm With Explicit IV
2404	The Use of HMAC-SHA-1-96 within ESP and AH
2403	The Use of HMAC-MD5-96 within ESP and AH

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
2402	IP Authentication Header
2401	Security Architecture for the Internet Protocol
2396	Uniform Resource Identifiers (URI): Generic Syntax
2395	IP Payload Compression Using LZS
2394	IP Payload Compression Using DEFLATE
2393	IP Payload Compression Protocol (IPComp)
2391	Load Sharing using IP Network Address Translation (LSNAT)
2375	IPv6 Multicast Address Assignments
2374	An IPv6 Aggregatable Global Unicast Address Format
2373	IP Version 6 Addressing Architecture
2367	PF_KEY Key Management API, Version 2
2362	Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification
2358	Definitions of Managed Objects for the Ethernet-like Interface Types
2344	Reverse Tunneling for Mobile IP
2328	OSPF Version 2
2308	Negative Caching of DNS Queries (DNS NCACHE)
2292	Advanced Sockets API for IPv6
2283	Multiprotocol Extensions for BGP-4
2275	View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)
2257	Agent Extensibility (AgentX) Protocol Version 1
2253	Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names
2249	Mail Monitoring MIB
2248	Network Services Monitoring MIB
2247	Using Domains in LDAP/X.500 Distinguished Names
2236	Internet Group Management Protocol, Version 2
2233	The Interfaces Group MIB using SMIv2

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
2230	Key Exchange Delegation Record for the DNS
2181	Clarifications to the DNS Specification
2178	OSPF Version 2
2163	Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)
2156	MIXER (Mime Internet X.400 Enhanced Relay): Mapping between X.400 and RFC 822/MIME
2144	The CAST-128 Encryption Algorithm
2137	Secure Domain Name System Dynamic Update
2136	Dynamic Updates in the Domain Name System (DNS UPDATE)
2133	Basic Socket Interface Extensions for IPv6
2132	DHCP Options and BOOTP Vendor Extensions
2131	Dynamic Host Configuration Protocol
2119	Key words for use in RFCs to Indicate Requirement Levels
2117	Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification
2104	HMAC: Keyed-Hashing for Message Authentication
2096	IP Forwarding Table MIB
2080	RIPng for IPv6
2074	Remote Network Monitoring MIB Protocol Identifiers
2073	An IPv6 Provider-Based Unicast Address Format
2072	Router Renumbering Guide
2071	Network Renumbering Overview: Why would I want it and what is it anyway?
2065	Domain Name System Security Extensions
2040	The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms
2023	IP Version 6 over PPP
2013	SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
2012	SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2
2011	SNMPv2 Management Information Base for the Internet Protocol using SMIv2
2003	IP Encapsulation within IP
2001	TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms
1997	BGP Communities Attribute
1996	A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)
1995	Incremental Zone Transfer in DNS
1994	PPP Challenge Handshake Authentication Protocol (CHAP)
1982	Serial Number Arithmetic
1972	A Method for the Transmission of IPv6 Packets over Ethernet Networks
1971	IPv6 Stateless Address Autoconfiguration
1970	Neighbor Discovery for IP Version 6 (IPv6)
1966	BGP Route Reflection An alternative to full mesh IBGP
1950	ZLIB Compressed Data Format Specification version 3.3
1933	Transition Mechanisms for IPv6 Hosts and Routers
1918	Address Allocation for Private Internets
1908	Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework
1907	Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)
1906	Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)
1905	Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)
1904	Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)
1903	Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
1902	Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)
1901	Introduction to Community-based SNMPv2
1893	Enhanced Mail System Status Codes
1887	An Architecture for IPv6 Unicast Address Allocation
1886	DNS Extensions to support IP version 6
1885	Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)
1884	IP Version 6 Addressing Architecture
1883	Internet Protocol, Version 6 (IPv6) Specification
18876	A Means for Expressing Location Information in the Domain Name System
1858	Security Considerations for IP Fragment Filtering
1853	IP in IP Tunneling
1850	OSPF Version 2 Management Information Base
1829	The ESP DES-CBC Transform
1828	IP Authentication using Keyed MD5
1827	IP Encapsulating Security Payload (ESP)
1826	IP Authentication Header
1825	Security Architecture for the Internet Protocol
1812	Requirements for IP Version 4 Routers
1795	Data Link Switching: Switch-to-Switch Protocol AIW DLSw RIG: DLSw Closed Pages, DLSw Standard Version 1
1779	A String Representation of Distinguished Names
1771	Lightweight Directory Access Protocol
1770	IPv4 Option for Sender Directed Multi-Destination Delivery
1757	Remote Network Monitoring Management Information Base
1750	Randomness Recommendations for Security
1724	RIP Version 2 MIB Extension

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
1723	RIP Version 2 - Carrying Additional Information
1722	RIP Version 2 Protocol Applicability Statement
1721	RIP Version 2 Protocol Analysis
1716	Towards Requirements for IP Routers
1712	DNS Encoding of Geographical Location
1706	DNS NSAP Resource Records
1662	PPP in HDLC-like Framing
1661	The Point-to-Point Protocol (PPP)
1660	Definitions of Managed Objects for Parallel-printer-like Hardware Devices using SMIv2
1659	Definitions of Managed Objects for RS-232-like Hardware Devices using SMIv2
1658	Definitions of Managed Objects for Character Stream Devices using SMIv2
1657	Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2
1655	Application of the Border Gateway Protocol in the Internet
1654	A Border Gateway Protocol 4 (BGP-4)
1644	T/TCP -- TCP Extensions for Transactions Functional Specification
1631	The IP Network Address Translator (NAT)
1623	Definitions of Managed Objects for the Ethernet-like Interface Types
1612	DNS Resolver MIB Extensions
1611	DNS Server MIB Extensions
1592	Simple Network Management Protocol Distributed Protocol Interface Version 2.0
1591	Domain Name System Structure and Delegation
1583	OSPF Version 2
1573	Evolution of the Interfaces Group of MIB-II
1572	Telnet Environment Option
1571	Telnet Environment Option Interoperability Issues
1566	Mail Monitoring MIB

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
1565	Network Services Monitoring MIB
1548	The Point-to-Point Protocol (PPP)
1542	Clarifications and Extensions for the Bootstrap Protocol
1541	Dynamic Host Configuration Protocol
1534	Interoperation Between DHCP and BOOTP
1533	DHCP Options and BOOTP Vendor Extensions
1532	Clarifications and Extensions for the Bootstrap Protocol
1531	Dynamic Host Configuration Protocol
1525	Definitions of Managed Objects for Source Routing Bridges
1514	Host Resources MIB
1497	BOOTP Vendor Information Extensions
1493	Definitions of Managed Objects for Bridges
1452	Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework
1451	Manager-to-Manager Management Information Base
1450	Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2
1449	Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2)
1448	Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)
1447	Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2)
1446	Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2)
1445	Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)
1444	Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2)
1443	Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2)

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
1442	Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)
1441	Introduction to version 2 of the Internet-standard Network Management Framework
1424	Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services
1423	Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers
1422	Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management
1421	Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures
1389	RIP Version 2 MIB Extensions
1388	RIP Version 2 Carrying Additional Information
1387	RIP Version 2 Protocol Analysis
1382	SNMP MIB Extension for the X.25 Packet Layer
1381	**SNMP MIB Extension for X.25 LAPB**
1361	Simple Network Time Protocol (SNTP)
1354	IP Forwarding Table MIB
1350	The TFTP Protocol (Revision 2)
1348	DNS NSAP RRs
1340	Assigned Numbers
1337	TIME-WAIT Assassination Hazards in TCP
1334	PPP Authentication Protocols
1333	PPP Link Quality Monitoring
1332	The PPP Internet Protocol Control Protocol (IPCP)
1327	Mapping between X.400(1988) / ISO 10021 and RFC 822
1323	TCP Extensions for High Performance
1321	The MD5 Message-Digest Algorithm
1320	The MD4 Message-Digest Algorithm

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
1319	The MD2 Message-Digest Algorithm
1315	Management Information Base for Frame Relay DTEs
1305	Network Time Protocol (Version 3) Specification, Implementation
1288	The Finger User Information Protocol
1286	Definitions of Managed Objects for Bridges
1282	BSD Rlogin
1271	Remote Network Monitoring Management Information Base
1269	Definitions of Managed Objects for the Border Gateway Protocol: Version 3
1258	BSD Rlogin
1256	ICMP Router Discovery Messages
1229	Extensions to the generic-interface MIB
1228	SNMP-DPI: Simple Network Management Protocol Distributed Program Interface
1227	SNMP MUX protocol and MIB
1215	Convention for defining traps for use with the SNMP
1213	Management Information Base for Network Management of TCP/IP-based internets:MIB-II
1212	Concise MIB definitions
1191	Path MTU discovery
1186	MD4 Message Digest Algorithm
1184	Telnet Linemode Option
1183	New DNS RR Definitions
1179	Line printer daemon protocol
1157	Simple Network Management Protocol (SNMP)
1156	Management Information Base for network management of TCP/IP-based internets
1155	Structure and identification of management information for TCP/IP-based internets
1122	Requirements for Internet Hosts - Communication Layers
1119	Network Time Protocol (version 2) specification and implementation

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
1116	Telnet Linemode option
1098	Simple Network Management Protocol (SNMP)
1094	NFS: Network File System Protocol specification
1091	Telnet terminal-type option
1084	BOOTP vendor information extensions
1075	Distance Vector Multicast Routing Protocol
1067	Simple Network Management Protocol
1066	Management Information Base for network management of TCP/IP-based internets
1065	Structure and identification of management information for TCP/IP-based internets
1063	IP MTU discovery options
1058	Routing Information Protocol
1057	RPC: Remote Procedure Call Protocol specification: Version 2
1050	RPC: Remote Procedure Call Protocol specification
1048	BOOTP vendor information extensions
1035	Domain names - implementation and specification
1034	Domain names - concepts and facilities
1010	Assigned numbers
1002	Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications
1001	Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods
0988	Host extensions for IP multicasting
0961	Official ARPA-Internet protocols
0959	File Transfer Protocol
0958	Network Time Protocol (NTP)
0951	Bootstrap Protocol
0950	Internet Standard Subnetting Procedure
0922	Broadcasting Internet datagrams in the presence of subnets

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
0919	Broadcasting Internet Datagrams
0904	Exterior Gateway Protocol formal specification
0903	Reverse Address Resolution Protocol
0894	Standard for the transmission of IP datagrams over Ethernet networks
0893	Trailer encapsulations
0887	Resource Location Protocol
0877	Standard for the transmission of IP datagrams over public data networks
0861	Telnet Extended Options: List Option
0860	Telnet Timing Mark Option
0859	Telnet Status Option
0858	Telnet Suppress Go Ahead Option
0857	Telnet Echo Option
0856	Telnet Binary Transmission
0855	Telnet Option Specifications
0854	Telnet Protocol Specification
0826	Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware
0822	Standard for the format of ARPA Internet text messages
0815	IP datagram reassembly algorithms
0793	Transmission Control Protocol
0792	Internet Control Message Protocol
0791	Internet Protocol
0790	Assigned numbers
0777	Internet Control Message Protocol
0768	User Datagram Protocol
0765	File Transfer Protocol specification

Table A-1: Supported RFCs (Continued)

RFC/IEEE	Description
0764	Telnet Protocol specification
0742	NAME/FINGER Protocol

Index