Power Hawk^{TM} Series 900
Diskless Systems Administrator’s Guide

concurrent

0891090-030
January 2006

Revision History:		Level:	Effective With:
Original issue	January 2006	000	PowerMAX OS Release 6.0 beta
Pevious Release	January 2004	010	PowerMAX OS Release 6.1
Previous Release	November 2004	020	PowerMAX OS Release 6.2
Current Release	January 2006	030	PowerMAX OS Release 6.3

Preface

Scope of Manual

Intended for system administrators responsible for configuring and administering diskless system configurations.

Structure of Manual

This manual consists of a title page, this preface, a master table of contents, four chapters, local tables of contents for the chapters, two appendices, glossary of terms, and an index.

- Chapter 1, *Introduction*, contains an overview of Diskless Topography, Diskless boot basics, configuration toolsets, definition of terms, hardware overview, diskless implementation, configuring diskless systems and licensing details.
- Chapter 2, *Netboot System Administration*, provides an overview of the steps that must be followed in configuring a loosely-coupled system (LCS) configuration.
- Chapter 3, *Flash Boot System Administration*, is a guide to configuring a diskless single board computer (SBC) to boot PowerMAX OS from flash memory.
- Chapter 4, *Debugging Tools*, covers the tools available for system debugging on a diskless client. The tools that are available to debug a diskless client depend on the diskless system architecture.
- Appendix A provides instructions on how to add a local disk to a client.
- Appendix B provides instructions on how to make a client system run in NFS File Server mode.
- The Glossary explains the abbreviations, acronyms, and terms used throughout the manual.
- The Index contains an alphabetical list of all paragraph formats, character formats, cross reference formats, table formats, and variables.

Syntax Notation

The following notation is used throughout this guide:

italic	Books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italic*.
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and man page references also appear in **`list bold`** type.

`list`	Operating system and program output such as prompts and messages and listings of files and programs appears in `list` type.
`[]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify such option or arguments

Referenced Publications

Concurrent Computer Corporation Manuals

Title	Pubs No.
System Administration Manual (Volume 1)	0890429
System Administration Manual (Volume 2)	0890430
Power Hawk Series 900 PowerMAX OS Version 6.3 Release Notes	0891089-6.3

Vendor Manuals

Manual Name	Synergy Document Number
Raptor DX VMEbus Dual G4, Dual PMC & StarFabric PowerPC Single Board Computer User Guide	02-0426/UG-VYFD-<REV>
MantaQX3/VAFQ VMEbus Quad G4, Single PMC & StarFabric PowerPC Single Board Computer User Manual.	815138 - Version <VER>
Synergy Microsystem STAR User Guide	03-0072/UG-STAR-<REV>
Synergy Microsystem ASTRix User's Guide	03-0076/UG-ASTX-<REV>

Related Specifications

Title	Pubs No.
IEEE - Common Mezzanine Card Specification (CMC)	P1386 Draft 2.0
IEEE - PCI Mezzanine Card Specification (CMC)	P1386.1 Draft 2.0
Compact PCI Specification	CPCI Rev 2.1 Dated 9/2/97

Contents

1
Introduction

1
Introduction

1.1. Overview

This manual is a guide to diskless operation of PowerMAX OS. Diskless operation encompasses the ability to configure, boot, administer and debug systems that do not have attached system disks. It should be noted that such a system might have attached non-system disks. Each diskless system runs its own copy of the PowerMAX operating system.

This manual specifically discusses Power Hawk Series 900 diskless configuration. Power Hawk Series 900 systems are currently supported in Loosely Coupled System (LCS) configurations, where a mix of Power Hawk Series 700 and Series 900 systems may be configured within the same LCS configuration.

Only the configuration and use of Power Hawk Series 900 systems are discussed in this manual. The user should refer to the *Power Hawk Series 700 Diskless System Administrator's Guide (0891089)* for details that are specific to setting up Power Hawk Series 700 systems in LCS configurations that may or may not include Series 900 systems.

1.1.1. Loosely-Coupled Systems (LCS)

The LCS configuration (see Figure 1-1) is supported when the only attachment between the fileserver and the diskless system is from an ethernet network. Inter-process communication between processes running on separate single board computers is limited to standard networking protocols across ethernet.

There are two possible ways of configuring a diskless client system. The difference between these client configurations is whether the client system maintains an NFS connection to the fileserver after boot such that file system space is available for the client system on the File Server. It is important to note that the type of client system configuration selected will impact the resource requirements of the File Server as will be explained in more detail later.

The two client configurations are:

Embedded client - Embedded clients are either stand-alone systems which have no attachments to other SBCs or they are not configured with networking and therefore do not use existing network attachments once the system is up and running. The embedded applications must be a part of the original boot image which is downloaded onto the client system and those applications begin execution at the end of the boot sequence.

NFS client - In an NFS client configuration, the File Server provides UNIX file systems for the client system. A client system operates as an NFS client of the File Server. This configuration allows substantially more file system space to be available to the client system for storing an application and application data than an embedded configuration.

The LCS configuration described may contain a mix of Power Hawk Series 700 and Power Hawk Series 900 systems. The file server in this LCS configuration may be either a Series 700 or Series 900 system.

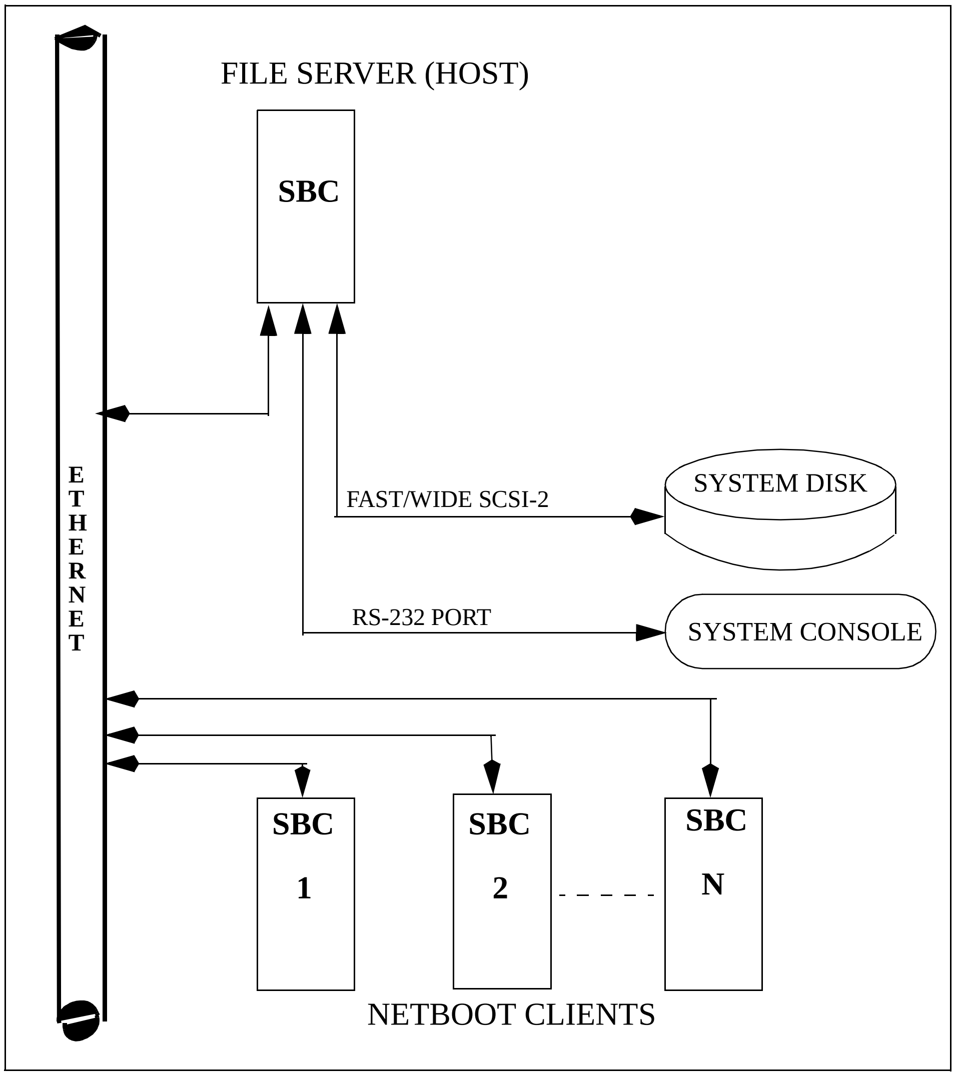

Figure 1-1. Loosely-Coupled System Configuration

1.1.2. Diskless Boot Basics

The first step in creating a diskless system is to create a boot image which contains both the operating system and a file system that contains at a minimum the executable needed to boot the PowerMAX OS. This file system, which is bundled into the boot image, can also be used to store application programs and data, UNIX commands and libraries or any other file that might live in a disk-based partition. The size of this file system is limited, since it must either be copied into memory or must reside in flash ROM.

The File Server is an SBC with attached disks where the boot image and a virtual root partition for each configured diskless system is created. The virtual root is both the environment used to build the boot image and it is also mounted by diskless systems that maintain an NFS connection to the File Server. Embedded diskless configurations do not maintain such an NFS connection. When the virtual root is mounted by the diskless system, it is used to hold system commands and utilities as well as user-defined files and application programs. The virtual root can be viewed as a resource for additional disk space for a diskless system.

Once a boot image is created, it must be copied from the File Server to the diskless system. There are two supported mechanisms for transferring a boot image to a diskless system:

1. A diskless system that is configured to boot from the network will read the boot image via an ethernet network connection to the File Server. The firmware uses the Trivial File Transfer Protocol (TFTP) over an ethernet connection to download the boot image.
2. The boot image may have already been burned into flash ROM. In this case, the board’s firmware (**`ASTRix`**) is configured to execute the boot image from flash ROM.

Closely related to the technique for copying a boot image to a diskless SBC, is the technique for initiating the boot sequence on the diskless SBC. There are two techniques for initiating the boot sequence on a diskless system. In some cases, the loading of the boot image cannot be separated from the initiation of execution within that image.

1. To boot from the ethernet network, the board's firmware (**`ASTRix`**) must be configured to boot from the network. The boot sequence is initiated by resetting the board, by cycling the power on the board, or by manually issuing the **`ASTRix`** command to execute a TFTP boot. The manual **`ASTRix`** method is only available when a console terminal is connected to the diskless system.
2. To boot from flash ROM, the board's firmware (**`ASTRix`**) must be configured to boot from flash, through the use of a **`ASTRix`** startup script. The boot sequence will be initiated whenever the board is reset, power is cycled on the board or by manually executing the **`ASTRix`** startup script or flash boot commands. The manual **`ASTRix`** method is only available when a console terminal is connected to the diskless system.

1.1.3. Net Boot Toolset

The Net Boot toolset is provided for creating the diskless configuration environment on the File Server and for creating boot images. The diskless configuration environment includes the generation of the virtual root as well as the creation and modification of relevant system configuration files. The virtual root serves as the environment for configuring a client’s kernel, building the boot image and as one of the partitions which is NFS mounted by an NFS client. The Net Boot tool is executed on the File Server.

The Net Boot toolset consists of the tools **netbootconfig** and **mknetbstrap**. These tools handle loosely-coupled clients that boot via an ethernet network or from flash ROM, where no common I/O bus based communication will be utilized on the client system.

The **netbootconfig** tool is used to create the diskless configuration environment for a diskless client. The **mknetbstrap** tool is used for creating a diskless client's boot image. More information is provided on the Net Boot tool in Chapter 2, "Netboot System Administration".

1.2. Definitions

Loosely-Coupled System (LCS)	A Loosely-Coupled System (LCS) is a network of Single-Board Computers (SBCs). One of the SBCs must have a system disk and is referred to as the File Server and all other SBCs are generally referred to as clients. An ethernet connection between the File Server and the client systems provides the means for inter-board communication.
File Server	The File Server has special significance in a Loosely-Coupled system as it is the only system with physically attached disk(s) that contain file systems and directories essential to running the PowerMAX OS™ (**/etc**, **/sbin**, **/usr**, **/var**, **/tmp**, and **/dev**). The File Server boots from a locally attached SCSI disk and provides disk storage space for configuration and system files for all clients. There is only one File Server in a Loosely-coupled system. All clients depend on the File Server for booting since all the boot images are stored on the File Server's disk.
Client	All SBCs, except for the File Server are considered clients. Clients do not have their own "system" disk. Clients must rely on the File Server for such support. However, clients may have local, non-system disk drives configured. The two client configurations, embedded and NFS, are described below:
1) Embedded Client	An embedded client runs self-contained from an internal memory-based file system; they do not offer console or network services. There is no swap space, because there is no media that can be used for swapping pages out of main memory. Applications run in single user mode (**init state 1**).
2) NFS Client	NFS clients are diskless SBCs that are configured with networking and NFS. Most directories are NFS mounted from the File Server. In addition to NFS, all standard PowerMAX OS™ network protocols are available. Swap space is configured to be remote and is accessed over NFS. Applications run in multi-user mode (**init state 3**).

System Disk	The PowerMAX OS™ requires a number of "system" directories to be available in order for the operating system to function properly. These directories include: **`/etc`**, **`/sbin`**, **`/dev`**, **`/usr`**, **`/var`** and **`/opt`**. The File Server is configured so that these directories are available on one, or more, locally attached SCSI disk drives. Since clients do not have locally attached system disk(s), they will NFS mount these directories from the File Server (an "NFS Client"), or create them in a memory file system which is loaded with the kernel (an "Embedded Client").
Net Boot (or Network Boot)	A client/server kernel boot method that uses standard TFTP protocols for kernel loading from the File Server. Any client can be configured to initiate a net boot operation from the File Server.
Flash Boot	A client boot method where the boot image executed comes from the client's own Flash memory.
Boot Image	This is the object that is downloaded into the memory of a diskless client. It contains a UNIX kernel image and a memory-based root file system. The memory-based file system must contain the utilities and files needed to boot the kernel. In the case of an NFS client, booting must proceed to the point that remote file systems can be mounted. For an embedded kernel, the memory-based file system is the only file system space that is available on the diskless system. Users may add their own files to the memory-based file system.
Synergy Monitor (STAR/`ASTRix`)	A board-resident ROM monitor utility that provides a basic I/O system (BIOS), a boot ROM, and system diagnostics for Power Hawk Series 900 single board computers (SBCs).
Trivial File Transfer Protocol (TFTP)	Internet standard protocol for file transfer with minimal capability and minimal overhead. TFTP depends on the "connectionless" datagram delivery service (UDP).
System Run Level Init Level	A term used in UNIX-derived systems indicating the level of services available in the system. Those at "**init level 1**" are single user systems which in turn is typical of embedded systems running on client SBCs. Those at "**init level 3**" have full multi-user, networking, and NFS features enabled, and is typical of client SBCs that run as netboot clients. See **`init(1M)`** for complete details.
swap space	Swap reservation space, referred to as 'virtual swap' space, is made up of the number of real memory pages that may be used for user space translations, plus the amount of secondary storage (disk) swap space available. Clients in the NFS configuration utilize a file accessed over NFS as their secondary swap space. Embedded clients, which are usually also Flashboot clients, generally do not utilize a swap device, but if a local disk is available then they too may be configured with a swap device.

1.3. Hardware Overview

The Power Hawk Model 920 and 940 platforms are supported only as a loosely-coupled client.

Motherboard Designation	System Platform	Number of CPUs	Form Factor	Netboot	Flashboot
VYFD	Power Hawk Model 920	2	VME 6U	yes	yes
VAFQ	Power Hawk Model 940	4	VME 6U	yes	yes

1.3.1. Model 920 Hardware Feature Summary

* SMP compliant
* PCI-to-VME64 bridge rated at 50 MB/sec.
* Dual G4 PowerPC 7455/7457
* 2 MB backside L3 cache per CPU
* 256 MB - 1 GB onboard memory
* 64 MB Boot/User Flash
* 128 MB of NVRAM, battery backed
* Two mailboxes (Hi & Lo) per CPU
* Real-time clock/calendar (4-digit year)
* Two asynchronous RS-232/423 serial ports up to 115.Kbps
* Fast Ethernet 10Base-T/100Base-TX (3 ports)
* Programmable 4-bit TTL-compatible general purpose I/O (GPIO)
* Dual 64-bit PMC slot

1.3.2. Model 940 Hardware Feature Summary

* SMP compliant
* PCI-to-VME64 bridge rated at 50 MB/sec.
* Dual G4 PowerPC 7457
* 2 MB backside L3 cache per CPU
* 256 MB - 1 GB onboard memory
* 64 MB Boot/User Flash
* 128 MB of NVRAM, battery backed
* Two mailboxes (Hi & Lo) per CPU
* Real-time clock/calendar (4-digit year)
* Four asynchronous RS-232/423 serial ports up to 115.Kbps
* Dual Gigabit Ethernet Ports
* Single 64-bit PMC slot

1.4. Diskless Implementation

1.4.1. Virtual Root

The virtual root directory is created on the File Server for each client when the client is configured. The virtual root directory is used to store the kernel build environment, cluster configuration and device files. In addition, for clients configured with NFS, the client's **/etc**, **/var**, **/tmp** and **/dev** directories are created here and NFS mounted on the client during system initialization. Each configured client has its own, unique virtual root on the File Server which is used as the configuration environment for that client.

A client's virtual root directory may be generated in any file system partition on the file server except for those used for the **/** (**root**) and **/var** file systems.

Virtual roots are created on the host for all clients. Clients running embedded systems will utilize their virtual root for configuring the clients's kernel and building the boot image.

1.4.2. Boot Image Creation and Characteristics

One of the primary functions of the virtual root is as the development environment for building the boot image that will be downloaded to diskless client systems. After a client's virtual root development environment has been created, users have the opportunity to tune the development environment in various ways, including that of adding in their own applications and data.

The boot image file, known as **unix.bstrap**, is composed primarily of two intermediate files: **unix**, and **memfs.cpio**. These are located in the same directory as **unix.bstrap**. **unix** is the client's kernel as built by **idbuild(1M)**. **memfs.cpio** is a compressed **cpio** archive of all the files which are to be the contents of that client's **memfs** root filesystem. This archive was compressed using the tool **rac(1)**. Conversely, if the user wants to examine the contents, **rac(1)** must be used to decompress it.

The final boot image, **unix.bstrap**, will contain a compressed version of the text and data regions of the unix kernel. These were extracted from the **unix** file. It will also contain bootstrap code, which decompresses the kernel and sets up its execution environment when the boot image is executed on the client, a copy of the compressed **cpio** image from **memfs.cpio**, and a bootstrap record used to communicate information about the client to the kernel and its bootstrap.

At the time of booting, boot files are created as needed based on dependencies established by the makefile "**bstrap.makefile**" under the **/usr/etc/diskless.d/sys.conf/bin.d** directory (see table below).

Boot File	Description	Dependencies
`unix`	unix kernel	**`kernel.modlist.add`**
`memfs.cpio`	cpio image of all files to be loaded in the client's memory-based root file system	**`unix`**, **`memfs.files.add`**, system configuration files
`unix.bstrap`	bootstrap image	**`unix`**, **`memfs.cpio`**

1.4.3. MEMFS Root Filesystem

A memory-based filesystem, called the memfs filesystem, becomes the root filesystem of a client as part of its booting process. As the client completes its boot, it may mount other file systems that are available to it, perhaps those on local disks or from across the network.

These other file systems do not replace the original memfs root filesystem but instead augment it with their extra directories and files. Files needed by diskless applications can be located either in the memfs root filesystem of a client or on the File Server in the client's virtual root directory.

For embedded systems, all user applications and data must be placed into the memfs root filesystem, since by definition no other file systems are available to such clients.

The tools used to build boot images provide a mechanism for adding user-defined files to the memfs filesystem contents and wraps those contents into the boot image that will be later downloaded into the client. When the boot image is downloaded into a diskless client system, it is resident in memory whether or not the files are being used. This means that the number and size of files that can be placed into the memfs file system is thus limited. This effect is minimized if the boot image is burned into flash using the *raw write* method (via the ASTRix *fw* flash write command). When the boot image resides in Flash, using the *raw write* method, then only those files actually in use will reside in (i.e. be copied into) physical memory at any time. In this mode of operation, the root filesystem behaves more like a normal filesystem: pages are automatically fetched from the Flash as needed, and, if not modified by applications, are automatically released when other needs for the space become more urgent.

It is possible for applications running on the client to write to memfs files; however, there not being a disk associated with these files, the changes will be lost on the next reboot of that client. Moreover, such pages remain permanently in memory until the files containing them are deleted or truncated. This ties up precious physical memory. This can be alleviated only by the addition of a swap device to the system, to which the system can write these dirty pages to as necessary, or by careful consideration and minimization of how much file writing is done by embedded applications into the memfs root filesystem.

Memfs file systems stored in Flash will be in a compressed format, in order to make maximum use of this relatively tiny device. Please refer to the *Flash Boot System Administration* chapter for more details on Flash boot configuration.

1.4.4. Booting

Once a bootstrap image is generated, it must be loaded and started on the client for which it was built. Two methods of booting are provided: Net Boot and Flash Boot. These methods are explained in more detail in the following sub-sections.

Booting a diskless client consists of two (Net boot) or three (Flash boot) distinct operations.

Step 1: Load the boot image into the client's SDRAM.
Step 2: Burn the boot image into flash (Flash boot method only).
Step 3: Initiate execution of the boot image.

Step 1

Regardless of whether the user is using Net boot or Flash boot method, the boot image must be downloaded to the diskless client. Downloading is performed across an ethernet network with the ASTRix ***tftp*** command. The tftp download is initiated automatically with an ASTRix startup script, or manually at the client's attached terminal console terminal. The ASTRix ***tftp*** command will read the boot image from the file server and download it into the client's SDRAM.

Step 2

This step applies only to the Flash boot method. After downloading the boot image to SDRAM, the image may be burned into flash. This is accomplished manually at the client's attached console terminal using the ASTRix ***ffsw*** (Flash File System Write) command, or the ASTRix ***fw*** (flash write) command. Once burned into flash, the boot image no longer needs to be downloaded across the network int SDRAM; step 1 is no longer required.

Step 3

The final step of the boot operation is to execute the downloaded boot image. This step is accomplished with the ASTRix ***boot*** command. The ASTRix ***boot*** may be initiated automatically within an ASTRix startup script, or manually at the client's attached console terminal. The ASTRix ***boot*** command allows for booting images from various devices, including booting images directly from memory (for the Net boot method), and booting images from flash (the Flash boot method).

The entire Net boot method may be set up so that the sequence is automatically initiated after any reset or power up cycle, by using the ASTRix startup script. In addition, once step 2 (Flash burn) has been accomplished for the Flash boot method, an ASTRix startup script may be used to automatically load and boot the flash boot image after any reset or power up cycle.

Booting from flash is significantly faster and more reliable than the Net boot method, and is recommended for final deployed environments. While actively testing and modifying the application and boot image the Net boot method may be used via an ethernet connection. The Flash boot method can then be used to burn the final version of the boot image into flash when the application is deployed.

1.4.4.1. Net Boot

Net Boot is a method of loading a kernel image into a client's SDRAM over an Ethernet connection, and then initiating execution of that image.

Since the file server and the client are only connected via Ethernet, the file server cannot actively force a client to accept and boot an image, rather, the client must initiate the transfer from the fileserver, and cooperate with the file server to complete the transfer. This client initiation takes one of two forms:

- manually entering and executing ASTRix commands at the client's attached console terminal,

  or

- automatic execution of the ASTRix commands by the client SBC whenever it is powered up or reset, through the use of an ASTRix startup script.

Net booting is performed by ASTRix using the TFTP (Trivial File Transfer Protocol, RFC783). This is a standard protocol that is supported by PowerMAX OS. For additional information pertaining to TFTP, refer to the Network Administration manual (0890432).

Any client SBC can be Net Booted as long as the client SBC has access to the file server through the client's ethernet connection. When the client is not on the same local Ethernet network, the system administrator may setup a routing table entry that will provide access to a remote file server through a local gateway.

Whether manually booting or autobooting a client, the client SBC must first be set up with the information it needs to do Net Booting. This is accomplished with the STAR 'config' command, and possibly with an additional ASTRix ***route*** command. or the appropriate commands in an ASTRix startup script. Once this setup is done, the client should not need to be reconfigured unless one or more of these parameters change, since the STAR ***config*** command information and ASTRix startup script is saved in NVRAM and is preserved across reboots and power cycles.

The following information is required by the STAR ***config*** command in order to configure a client that will be performing a net boot:

Target IP address:

> The local client SBC's IP address which ASTRix will use as the return address for TFTP data transfer. For NFS clients, this is the Ethernet IP Address of this client SBC.

Host IP address:

> The IP address for the File Server which should already be defined in the /etc/hosts file.

You may also Net Boot an Embedded Client. Since this client kernel does not support networking, no IP address has yet been defined for it. In this situation, select a unique IP address to use. An address should be selected that decodes to the same local subnet that the client is connected to, and that does not conflict with any other IP addresses used in the network.

If the client SBC's access to the File Server is through a gateway, then a routing table entry must be setup with the ASTRix ***route*** command. Usually, this would be done in the ASTRix startup script, so that this setup would be automatically executed every time ASTRix was initiated following a reset or power cycle.

After STAR has been initialized with the proper network parameters and any required routing table entry has been added with the ASTRix ***route*** command, a transfer of the boot image across the Ethernet connection may be initiated via the ASTRix ***tftp*** command. Refer to the chapter on Netboot System Administration for more information on Net Booting.

1.4.4.2. Flash Boot

Flash Boot is a method of loading and executing a kernel image from flash. Flash Boot is the preferred method of booting a diskless client in the production or deployed phase of an application. There are two advantages to booting from flash:

- Flash Boot provides very fast boot times because there are no rotational delays that would normally be associated with reading from a local system disk, and no networking delays that would normally be associated with downloading a bootable image from a file server, when using the Net Boot method.
- The root file system resides as a read-only image in flash, thus providing greater system stability because the root file system cannot be corrupted by unexpected system crashes which might leave a writable file system in an inconsistent state.

The boot image that is burned into flash is the same boot image that is downloaded via an Ethernet network connection when a developer is using the Net Boot method. As such, a developer may use the Net boot method when they are actively modifying the boot image during the development phase, and then they may burn the final boot image into flash when they enter into the deployed phase of the application.

There are no tools specifically targeted towards creating boot images for Flash Booting. Instead, the standard loosely-coupled configuration tools are used for building the image.

Since the File Server is only connected to the client system via an ethernet connection, the flash must be burned via a process known as network loading. Preparation for loading the boot image into flash is the same method used in the Net Boot method. The ASTRix ***tftp*** command is used to load the boot image into memory without executing that image. The boot image is then burned into flash from memory by using either the ASTRix ***ffsw*** (flash filesystem write) command, or the ASTRix ***fw*** (flash write) command.

Once the image is burned into flash, then all subsequent booting can be automatically initiated at reset or power cycle time by setting up an ASTRix startup script that will execute the appropriate “boot” command that will boot the bootable image that is stored in flash. Refer to Flash Boot System Administration chapter for more details.

1.4.5. Remote File Sharing

Clients configured as "Embedded Clients" must have in their memory-based root file system all the files needed for booting and all the files needed to run applications. This is because Embedded Clients do not have networking by definition and therefore will not have access to remote files on the File Server.

The **`memfs`** root file system of a client configured with NFS need only contain the files required for booting. When the client system reaches **init state 3** it is able to NFS mount and access the File Server's directories. The NFS mounts are executed from a start-up script in **`/etc/rc3.d`**.

Two different **`inittab`** files are used in booting an NFS configuration. When the **`etc`** directory in a client's virtual root is NFS mounted, the original **`inittab`** file is overlaid with the one in the virtual root. The directory **`etc/rc3.d`** is then re-scanned to execute start-up scripts in the virtual root.

The directories **`/usr`**, **`/sbin`** and **`/opt`** are completely shared with the server, while **`/etc`** and **`/var`** are shared on a file-by-file basis.

Listed below is the NFS mount scheme in use:

Path on File Server	Mount Point on Client
`/usr`	**`/usr`**
`/sbin`	**`/sbin`**
`/opt`	**`/opt`**
`<virtual_rootpath>/etc`	**`/etc`**
`<virtual_rootpath>/var`	**`/var`**
`<virtual_rootpath>/dev`	**`/dev`**
`<virtual_rootpath>/tmp`	**`/tmp`**
`virtual_rootpath>/users`	**`/users`**
`/etc`	**`/shared/etc`**
`/var`	**`/shared/var`**
`/dev`	**`/shared/dev`**

The /etc and /var directories under the client's virtual root contain some files that are client specific, therefore these files are not shared. Each client has its own unique version of these files. These directories also contain some files which have the same content for the File Server and all client virtual roots on the File Server. These files are shared between the File Server and all of the File Server's clients. Because the /etc and /var directories contain a mix of both shared and non-shared files, these directories require special handling within the client's virtual root.

The files **`/etc/nodename`** (not shared) and **`/etc/chroot`** (shared) will be used to illustrate how shared and non-shared files are handled in **`/etc`** and **`/var`**.

The **/etc/nodename** file is simply created as a real file in the client's virtual root under the **<virtual_rootpath>/etc** directory. The **<virtual_rootpath>/etc** directory is mounted on the diskless client under the **/etc** directory.

The **/etc/chroot** file is created in the client's virtual root not as a real file, but as a symbolic link to the file name **/shared/etc/chroot**. On the File Server there is no such directory as **/shared**. On the client system, **/shared** is used as the mount point for mounting the File Server's actual **/etc/** directory. Thus any reference on the diskless client to **/etc/chroot** will actually be referencing the **/etc/chroot** file that exists in the File Server's **/etc** directory.

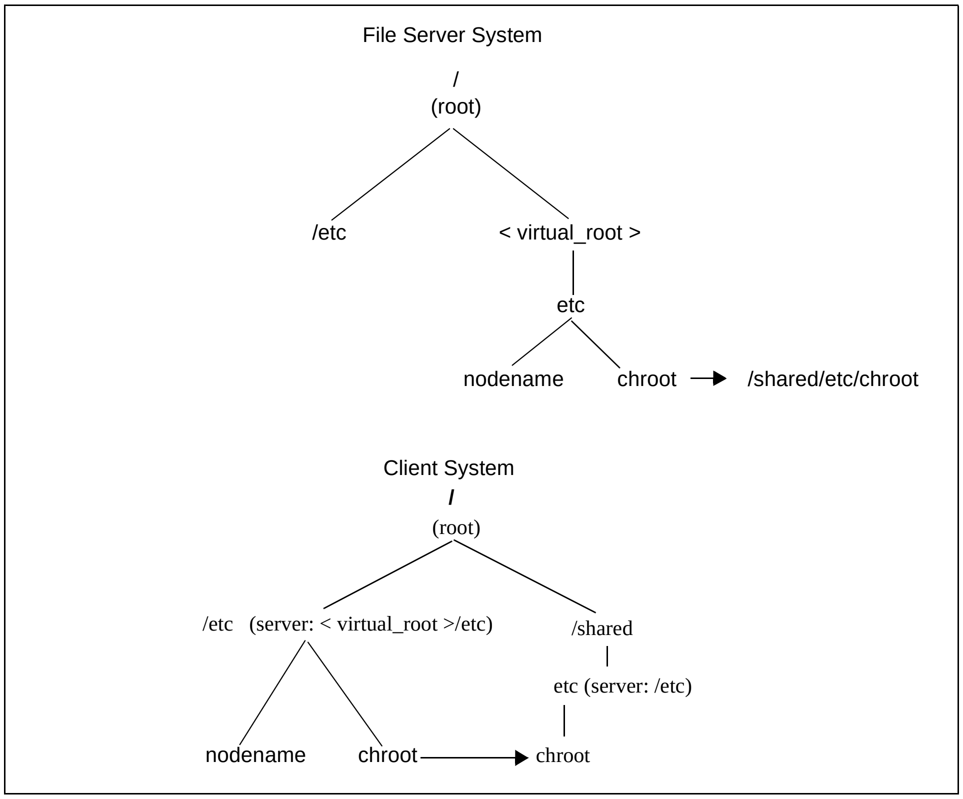

Figure 1-2. Power Hawk Networking Structure

As new files are added and removed from the File Server's **/etc** and **/var** directories, the symbolic links under the client's virtual root may become stale. The configuration utility **mknetbstrap** can be used to update the links in these directories to match the current state of the File Server.

The directories **/dev** and **/tmp** are also created under the client's virtual root but do not share any files with the File Server. Device files may have kernel dependencies and so these files are not shared. The directory **/tmp** is created as an empty directory. The directory **/users** is also empty and may be used to access user files across NFS.

Once the client system is up and running, the files in the memory-based root file system required for booting are no longer needed and are removed to free up memory.

Permission to access remote files on the File Server is automatically granted. During client configuration, the **/etc/dfs/dfstab** (see **dfstab(4)**) and **/usr/etc/diskless.d/cluster.conf/dfstab.diskless** tables are modified to allow a client either read or read/write access to files which reside on the File Server.

The **dfstab.diskless** file is generated when the first client is configured. Sample entries from this file are listed below. These entries should not be modified.

```
share  -F  nfs  -o  ro,root=$CLIENTS           -d  "/etc/"   /etc   /shared/etc
share  -F  nfs  -o  ro,root=$CLIENTS           -d  "/dev/"   /dev   /shared/dev
share  -F  nfs  -o  rw=$CLIENTS,root=$CLIENTS  -d  "/var/"   /var   /shared/var
share  -F  nfs  -o  ro,root=$CLIENTS           -d  "/sbin/"  /sbin  /sbin
share  -F  nfs  -o  rw=$CLIENTS,root=$CLIENTS  -d  "/usr/"   /usr   /usr
share  -F  nfs  -o  rw=$CLIENTS,root=$CLIENTS  -d  "/opt/"   /opt   /opt
```

The dfstab.diskless file is referenced from a command line entry in dfstab, generated when the first client is configured. For every client configured thereafter, the client's name is added to the CLIENTS variable. For example, after configuring two clients, named client1 and client2 the following line appears in the dfstab table:

```
 CLIENTS=client1:client2 /usr/sbin/shareall \
-F nfs /usr/etc/diskless.d/cluster.conf/dfstab.diskless
```

In addition, an entry to make each client's virtual root directory accessible is generated at configuration time. If a parent directory of the client's virtual root directory is already currently shared (has an entry in sharetab(4)), then the matching entry in dfstab(4), if found, is modified to include the client in its rw= and root= attribute specifications. For example, if the virtual root directory for the client named client1 is in /home/vroots/client1 and the /home/vroots directory is currently shared; then the sample entry below would be changed as shown `below.`

from:

```
/usr/sbin/share -F nfs -d "/home/vroots" /home/vroots vroots
```

to:

```
/usr/sbin/share -F nfs -o root=client1 -d \
"/home/vroots" /home/vroots  vroots
```

There is no need to add an `rw=` attribute since, when not specified, it defaults to read/write access permissions to all.

If no entry is currently shared that covers the client's virtual root directory, then a specific entry for each client is appended to the **dfstab.diskless** file. For example, for the client named **client1**, whose virtual root directory is **/home/vroots/client1**, the following entry is generated:

```
/usr/sbin/share -F nfs -o rw=client1,root=client1\
-d /home/vroots/client1 /home/vroots/client1 vroot
```

After the files are updated, the **"shareall -F nfs"** command is executed to update the File Server's shared file system table.

When a client's configuration is removed, all references to the client and it's virtual root directory are removed from both the **`dfstab`** and **`dfstab.diskless`** files and the **`unshare(1M)`** command is executed to update the shared file system table.

1.4.6. Swap Space

Embedded systems generally do not have swap space. Nevertheless, some aspects of swap space configuration do affect even embedded systems, so this section should be read even for these users.

Normal systems have a disk partition reserved for swap space. For diskless nfs clients, swap space is implemented using a regular file created in the client's virtual root directory and accessed over NFS. The size of the swap file is user-configurable.

Swap reservation space, referred to as 'virtual swap' space, is made up of the number of real memory pages that may be used for user space translations, plus the amount of secondary storage (disk) swap space available. If no secondary storage swap space is available, then the amount of virtual swap space degenerates to the number of real memory pages available for user address space.

A virtual swap space reservation is made by decrementing the amount of available virtual swap space. If no virtual swap space is available, then the reservation will fail, and subsequently, either the page fault or segment create operation will not succeed. Virtual swap reservations are made so that as real memory becomes low, the pageout and process swap daemons can guarantee that there will be an appropriate number of user pages that can be swapped out to secondary storage and subsequently freed, in order to maintain an adequate level of free memory pages for new page allocations.

Even when there is no swap space configured into the kernel, the virtual swap reservations will prevent the kernel from over committing real memory to user pages that cannot be swapped and freed if and when the number of free real memory pages becomes low. If these daemons did not maintain an adequate number of free memory pages for page allocations, then applications might become blocked forever in the kernel, waiting for their page allocation requests (or internal kernel memory page allocation requests) to complete.

There are a number of possible swap space configurations on client systems:

1. no swap space	typically, this would be a client configured in Embedded mode
2. remote swap space	client would be configured as a NFS diskless system with the swap space accessed through the NFS subsystem
3. local disk swap space	client configured in either NFS or Embedded mode, client configured to use a local disk for swap space

When there is no swap space, or a small amount of swap space, it may be necessary to modify the default values of certain system tunables in order to maximize system performance and user virtual space capacities.

The following are some of the system tunables that are relevant to system swap space management in a system with little or no secondary storage swap space.

1. Systems with no swap space should be tuned such that process swapping does not become aggressively active before process growth is limited by virtual swap reservations, as this will impact system performance without providing significant amounts of additional free memory. The address space aging interval should also be increased.

   System tunables that govern the address space aging interval are:

   ```
   INIT_AGEQUANTUM
   MIN_AGEQUANTUM
   MAX_AGEQUANTUM
   LO_GROW_RATE
   HI_GROW_RATE
   ```

   In order to ensure longer address space aging intervals, all of these tunables may be set to a higher than default value.

2. The **GPGSLO** tunable value can be decreased in order to lower the free memory level at which process swapping will become aggressively active.

3. The **DISSWAPRES** tunable disables virtual swap reservations by setting the amount of available virtual swap space to an artificially large value.

   The **DISSWAPRES** tunable allows more user page identities/objects to be created than what can be accommodated for in virtual swap space. Since typically, applications do not tend to access all the pages that may potentially be considered writable (and therefore require a virtual swap reservation), this tunable may allow for a larger number of applications to run simultaneously on a system by not requiring virtual swap space for every potentially writable user page.

   However, when the **DISSWAPRES** tunable is enabled, it becomes possible for page allocations to block forever, since the pageout and process swap daemons may not be able to swap out an adequate number of user pages in order to free up pages for additional allocations. At this point, the system will enter a state where little or no useful work is being accomplished. Therefore, caution is advised when using the DISSWAPRES tunable.

   The DISSWAPRES tunable may be useful when a fixed set of applications and their corresponding virtual address space working sets are known to fit into the amount of available real memory (and secondary storage swap space, if any), even though their total virtual swap space requirements exceed the system's virtual swap space capacity.

1.5. Configuring Diskless Systems

1.5.1. Loosely-Coupled System Hardware Prerequisites

A loosely-coupled configuration requires the following hardware:

- At least one VME card chassis.
- One Power Hawk Series 900 or Series 700 single board computer, with a minimum of 128 MB of DRAM, for use as the server SBC.
- One Power Hawk Series 900 or Series 700 single board computer, with a minimum of 128 MB of DRAM, for each client SBC.
- One SCSI 2 GB (4 GB or higher is preferred) disk drive for PowerMAX OS^{TM} software installation, connected to the Synergy PSCx SCSI/PMC interface card on the server SBC.
- One supported SCSI CD-ROM device, connected to the PSCx SCSI/PMC interface card for installation of system software on the server SBC.
- At least one system console terminal, which may be a video display terminal such as a Wyse 150, vt100, or comparable device connected to Serial Port A on the server SBC. Additional system console terminals may be attached to any client SBC's Serial Port A, for debug purposes.
- The server SBC must be accessible from all client SBCs via an Ethernet LAN. For Power Hawk Model 920 systems, the on-board Galileo GT64260 Ethernet controller should be used for this connection, for Power Hawk Model 940 systems, the on-board Marvell MV64460 Ethernet controller should be used. For Power Hawk Series 700 systems, the on-board Symbios Ethernet controller (Symbios SYM53C885) should be used for this connection.

1.5.2. Disk Space Requirements

The table below details the amount of available disk space required per client single board computer for the virtual root partition. These values are for the default shipped configuration. Added applications may increase disk space requirements. Values in this table do not include swap space for the diskless system. The amount of swap space is configurable, but should be at least one and one-half times the size of physical memory on the single board computer.

A client's virtual root directory can be generated in any disk partition on the File Server. The **`/(root)`** and **`/var`** file systems are not recommended for use as client virtual partitions.

Client Configuration	**Disk Space**
NFS	25 Megabytes
Embedded	15 Megabytes

1.5.3. Software Prerequisites

The following software packages must be installed on the host system prior to installing the diskless package (prerequisite packages listed alphabetically by package name):

Package Name	**Package Description**	**Package Dependencies (See Note)**
base	Base System (Release 6.0 or later)	
cmds	Advanced Commands	lp, nsu
gte	GT64260 Ethernet Driver (910/920)	nsu
mve	MV64460 Ethernet Driver (940)	nsu
sym	Symbios 53C885 Fast Ethernet Driver	nsu
dfs	Distributed File System Utilities	inet
inet	Internet Utilities	nsu
lp	Printer Support	
ncr	Internal NCR SCSI Driver	
netcmds	Commands Networking Extension	lp, inet
nfs	Network File System Utilities	nsu, inet, rpc, dfs
nsu	Network Support Utilities	
rpc	Remote Procedure Call Utilities	inet
Note: All packages are dependent on base package		

1.6. Licensing Information

The system installed on the File Server carries a license for the number of processors allowed to be booted. The license also carries a limit for the number of users allowed to log on to the File Server. All diskless client SBCs are limited to a maximum of 2 users each.

To print the processor and user limits set for your machine, use the **`-g`** option of the **`keyadm(1M)`** command.

2
Netboot System Administration

2
Netboot System Administration

2.1. Configuration Overview

This is a overview of the steps that must be followed in configuring a loosely-coupled configuration. Some of these steps are described in more detail in the sections that follow.

A loosely-coupled system consists of a File Server and one or more diskless clients which download their private boot image, which resides on the File Server. A loosely-coupled system uses an ethernet network connection between each diskless client and the File Server for communication. There is no sharing of a VME bus in this configuration.

The following instructions assume that all the prerequisite hardware has been installed and each netboot client's on-board Ethernet controller is attached to a subnet on which the File Server system may be accessed, either directly on the same subnet, or through a gateway. For details, see “Loosely-Coupled System Hardware Prerequisites” on page 1-17

2.1.1. Installing a Loosely-Coupled System

Follow these steps to configure a loosely-coupled system.

1. Install the File Server with the prerequisite software packages, the diskless package and all patches. Refer to the “Software Prerequisites” on page 1-18 and the applicable system release notes for more information.

2. On the File Server system, configure and mount file system(s) (other than **`/`** (root) or **`/var`**) that can be used to store the virtual root directories for each client. If not already present, an entry for this file system must be added to **`/etc/vfstab(4)`**. An existing file system can be used, but there must be sufficient file space to hold the client virtual root files.

   The boot images for each netboot client are placed into the **`/tftpboot`** directory by **`mknetbstrap(1M)`**. Since these images tend to be large in size, the system administrator may want to mount a sufficiently large file-system at the **`/tftpboot`** mount point, if the File Server's **`/`** root partition is not able to accommodate the projected client boot image disk space requirements.

   See the “Disk Space Requirements” section on page 1-18 for details of the amount of file space required for the client virtual root directories and **`/tftpboot`** boot images.

3. On the File Server system in the **`/etc/profiles`** directory, create a client profile file for each netboot client.

You can use **netbootconfig(1M)** to print out a template of a netboot client profile. The client profile file name must be equivalent to the client's Ethernet hostname. For example, to create a netboot client profile for a client with a hostname of '*wilma*', do the following:

```
netbootconfig -P > /etc/profiles/wilma
vi /etc/profiles/wilma
```

Edit the resulting client profile file to be relevant to the specific characteristics of the client SBC. The parameters listed in the client profile are described in "The Client Profile File" on page 2-13. Additional description of these parameters in the file **/usr/etc/diskless.d/profiles.conf/netboot.client.profile.README**.

4. Update the **/etc/hosts** file on the File Server SBC with the hostnames of all the netboot clients. The hostname(s) added to the hosts file should match the filename(s) of the client profile file(s). The IP addresses for each client should correspond to the IP address of the first onboard Ethernet controller (/dev/gte0 for Series 910/920, /dev/mve0 for Series 940).

5. On the File Server system, execute **netbootconfig(1M)** to configure the build environment of each diskless client to be configured. See the "Configuring Clients Using netbootconfig" on page 2-16 for more information.

6. On the File Server system, execute **mknetbstrap(1M)** to create the boot images of each diskless client. See the "Booting and Shutdown" section on page 2-31 section for more information.

7. On each client, connect a console terminal and power up the netboot client. Use STAR to set the hardware clock and networking configuration, and optionally use ASTRix to configure an ASTRix netboot startup script. Use `ASTRix` to set the hardware clock, to setup the `ASTRix` networking configuration, and to optionally configure a `ASTRix` netboot startup script. See "SBC Client Board Configuration" on page 2-3 for more details.

8. On each client, either:
 - manually boot into ASTRix from STAR (if STAR is not configured to automatically boot ASTRix), and then manually execute ASTRix *tftp* and boot commands to boot the client, or

   - reset or power-cycle the client board, if the client is configured to automatically start up ASTRix from STAR, and to execute an ASTRix tftp boot startup script.

   See "Net Booting" for more details on booting up the client.

   On the File Server system, customize the client's virtual root configuration as needed and then run **mknetbstrap(1M)** to process the changes. If a new boot image is created as a result of the changes, shutdown the client and then reboot it. See "Booting and Shutdown" on page 2-31 for more information.

2.1.2. Installing Additional Boards

To add additional boards after the initial configuration, follow steps 2 through 8 described above.

2.2. SBC Client Board Configuration

This section describes the procedure for configuring a SBC board as a client SBC in a loosely-coupled system.

The user should also refer to the *Synergy Microsystems ASTRix User Guide* and the *Synergy Microsystems S.T.A.R. User Guide* for additional information regarding STAR and ASTRix commands and features.

The following steps should be followed in order to set up a board as a client SBC:

1. Connect a terminal to Serial UART Port A/Console if one is not already connected.
2. The next step is to update some of the STAR configuration parameters. Power-on or reset the board and watch for the resulting input prompt.
   If the:

   STAR0>

   prompt appears, then skip the rest of this step and go to step #3. If STAR is automatically setup to boot ASTRix, then following prompt will eventually appear on your screen (without any user intervention):

   *****

   To return back to STAR from ASTRix, enter the following command at the ASTRix prompt:

   ***star**

   and then enter <cr>. The system will reboot itself into STAR without starting up **ASTRix**.

   At this point you should be running STAR, and sitting at the:

   STAR0>

   prompt.

   This step describes how to enable the automatic startup of ASTRix by STAR. To enable the automatic startup of ASTRix, use the STAR config command.

   An example of the entire output of the config command is shown below:

```
STAR0> config
Enter a new value or a return to skip, '-' to back up, or '.' to exit:
**** BOOT Configs (reset when defaults set) ****
Port A baudrate:                     9600 =
Port B baudrate:                     9600 =
Share console port:                  Y =
PCI Enum powerup delay (milliseconds):  1007 =
Enable quiet boot mode:                 N =
Enable ASTRix autoboot:                 N =
Enable "post" script:                   Y =
Skip PCI Enumeration:                   N =
Skip PMC Configuration:                 N =
L1 Miss Queue Depth (1,3,6):            3 =
L3 Size L3 (1, 2 MB):                   2 =
Enable Built-in self tests:
    1= at powerup only
    2= at all hard resets
    3= at all resets                     0 =

**** USER Configs (unchanged when defaults are set) ****
Board serial number:                    0x1301892 =
Target IP address:                      129.134.32.80 =
Host IP address:                        129.134.32.81 =
Print Boot Banner:                      Y =
Init mem EDC:                           N =
Legacy Mode Input:                      N =
Enable ChangeLog Message:               Y =
PCI 0 Start Addr (ex. 0x80000000):      0x80000000 =
PCI 0 Size (ex. 0x40000000 (1GB)):      0x30000000 =
PCI 1 Start Addr (ex. 0xB0000000):      0xB0000000 =
PCI 1 Size (ex. 0x10000000 (256MB)):    0x40000000 =
Hit Enter to finish:                      =
STAR0>
```

3. Hit the <cr> key until the “Enable **ASTRix** autoboot:” prompt appears, and then enter “Y <cr>”. Then continue entering <cr> until the
   STAR0>
   prompt reappears.

NOTE

Once the autoboot ASTRix parameter is enabled, it is still possible to get back into STAR without immediately going into ASTRix. To get back into STAR from ASTRix, simply enter the following command while running ASTRix:

star <cr>

This command will reboot the board and return to STAR without autobooting ASTRix, regardless of the current setting of the “Enable ASTRix autoboot:” parameter.

Additional parameters in the STAR 'config' command are listed below:

- The “Port A baudrate” parameter should be set to “9600”, and it is recommended that you also set the “Port B baudrate” to “9600”, especially if you plan to hook up a second terminal to Port B.
- The “Share console port” parameter MUST be set to “Y”.
- It is highly recommended that you do NOT modify the “Board serial number” parameter. This parameter is used to create unique Ethernet hardware addresses for all of the on-board Ethernet controllers.
- The PCI 0 and 1 Memory Space config command parameters should beset to the values shown above. If they are not setup with these values, then use the STAR config command to set them to the values shown above.

  After you have successfully netbooted the client, you may want to change the PCI Memory Space layout from the above default values.

  Please consult the Power Hawk Series 900 Console Reference Manual (Pubs No. 0830060) section "PCI Memory Space Configuration", for more information on this subject.
- It is highly recommended that you do NOT set the “Enable quiet boot mode:” to “Y”. This parameter disables all STAR output to the console terminal. However, if you do happen to enable quiet mode by accident, then enter the following command to disable quiet mode. The output of this command will NOT be echoed to the terminal:

  serialok=1 <cr>

  After this command has been entered, then execute the config command again and disable the “Enable quiet boot mode:” parameter.
- Even if you wish to enable ECC memory error correction and detection, leave the “Init mem EDC:” parameter set to “N”. The enabling of ECC will be accomplished through the use of a post script, and is described in a following step in this section.

4. The netboot client's hardware clock must be updated to match the date on the system designated as the File Server. Use the STAR 'date' command to display and/or set the current date and time. To display the date/time values, enter the date command with no arguments:

```
STAR0> date
 X: 2/10/2006 TUE 15:44:38 GMT-0
```

   To set the date and time to a value that matches the File Server's date and time (displayed via a date(1) command when executed on the File Server), use the STAR date command, where the new date and time values are specified in the following format:

```
date YYMMDD HHMMSS tz
```

   (The “help date” STAR command will output more information about the date command.)

It is best to use a 'tz' (timezone) value of 0 (GMT timezone) and adjust the hour field manually to adjust the hour to fit your own local timezone.

For example, to set the date to March 14, 2006 and the time to 1:15:00 P.M. in the EDT timezone, enter the following:

```
STAR0> date 060314 181500
X: 3/14/2006 TUE 18:15:00 GMT-0
```

Note

You must add 4 hours to the above HH field in order to enter the correct GMT time that corresponds to the current EST timezone with daylight savings time in affect (add 5 hours if daylight savings time is not in currently active).

5. This step describes how to configure the IP address networking parameters that will be used by the onboard Ethernet controller when executing under STAR and ASTRix. To setup the networking parameters, use the STAR config command again:

   STAR0> config

   Enter <cr> until the "Target IP address" prompt appears.

   This parameter should be set to the IP address of this client SBC. This IP address should be the onboard Ethernet address of this client SBC that is already in the File Server SBC's /etc/hosts file. Enter this IP address, followed by a <cr>.

   The next config parameter, "Host IP address" should be set to the IP address of the File Server for this client SBC. This IP address should be the same IP address of the File Server that is located in the File Server's /etc/hosts file. Enter the appropriate IP address followed by a <cr>. The address contained in the "Host IP address" parameter will be the $HOST environment variable that is used later on when running under ASTRix.

NOTE

If the client and server SBC are not connected to the same network and the client accesses the server SBC through a gateway, then this routing issue is handled under ASTRix, which is discussed later on in step 10 below.

6. You may optionally setup the client SBC so that ECC error correction and detection is enabled in the SDRAM memory. If you do NOT wish to enable ECC for the SDRAM memory, then set the

   Enable "post" script:

   parameter of the STAR config command to "N", and proceed to step 8.

   The enabling of ECC checking in SDRAM is accomplished through the use of post script commands. The execution of post script commands may be enabled when running STAR, but the creation and setup of these post scripts is accomplished while running under ASTRix.

So at this point, simply enable post script execution using the STAR config command. Enter <cr> until the following parameter appears:

```
        Enable “post” script:
```

and then set this parameter to “Y”. The creation of the proper post scripts will be discussed in step 8.

7. The remaining setup of the client is accomplished while running ASTRix. Since STAR configuration parameters have changed, reset the board so that the new STAR parameters will now take affect.
   Use the STAR 'reboot' command to reset the board:

   ```
           STAR0> reboot <cr>
   ```

   The board should reset itself, and STAR should automatically boot into ASTRix:

   ```
           Y: CPU 1 standing by.
   This is the example NVRAM startup script

   Synergy Microsystems ASTRix Rev: 1.01.06 Jan 29 2003 19:42:00

   *
   ```

   The “*” above is the default ASTRix command prompt.

NOTE

Some of the following steps discuss creating or modifying files that are located in the /nvram directory when running under ASTRix. The /nvram directory is created by ASTRix when ever it is started up, and the contents of the NVRAM filesystem are copied into this directory. Users can save up to 24KB of files in the NVRAM filesystem. Files placed in the /nvram directory are automatically copied to the NVRAM filesystem whenever the “boot”, “reboot”, “star” or “halt” commands are issued. Thus, the files in /nvram that are saved in the NVRAM filesystem are preserved across resets and power shut down, allowing users to retain files from one boot to the next.

8. If you do NOT wish to enable ECC checking in SDRAM memory, and you have already disabled post script execution in step 6 above, then skip ahead to step 9.

   The rest of the ECC configuration involves setting up two post scripts. The /nvram/post script is executed by STAR after reset or power-on, and the /nvram/post05 script is executed at the end of the ASTRix 'boot 5' command sequence just before the client netboot image execution begins.

   To create the /nvram/post script, issue the following ASTRix commands:

   ```
           * cd /nvram <cr>
           * rm post <cr>     (ignore any error messages if this file doesn't
   ```

```
                    exist)
        * vi post <cr>
```

Add the following lines to the post script file, using the vi editor:

```
 if (cpuid == 0) {
 imedc 0 0 1;
}
```

Use the ":wq" vi command to write the file and exit vi. You may then:

* **cat /nvram/post** <cr>

to check that the file contents are correct. If the contents are not correct, then use "vi post" to correct the contents.

The above post script will cause CPU 0 to initialize memory and enable ECC checking in SDRAM when ever STAR begins executing after a reset or power-on.

To create the post script that will be executed before the client netboot image is executed, issue the following commands:

```
        * cd /nvram <cr>
        * rm post05 <cr> (ignore any error messages if this file doesn't
                    exist)
        * vi post05 <cr>
```

Then add the following lines to the post05 script file, using the vi editor:

```
if (cpuid == 0) {
  serialok=1;
  printf("\nEnabling ECC\n");
  serialok=0;
  eecc;
}
```

Use the ":wq" vi command to write the file and exit vi. You may then:

* cat /nvram/post05 <cr>

to check that the file contents are correct. If the contents are not correct, then use "vi post05" to correct the contents of the post file.

When the client boot image is booted, the post05 script above will output the "Enabling ECC" message to the console terminal, and will then re-enable ECC checking in SDRAM.

9. This step will setup an ASTRix boot command configuration slot for booting the client netboot image. This step is required for both manual and automatic booting.

   To setup an ASTRix boot command "configuration" slot number that may always be used for booting the client netboot image:

   * **boot 5 -n /<client>.bstrap -s -c**

   The command above will setup the booting of the client's netboot image, where <client> should be replaced by the actual hostname of the client. For example, if this client's hostname is wilma, then the following boot command line to setup configuration slot number 5 would be:

```
* boot 5 -n /wilma.bstrap -s -c
```

The -s option sets the configuration without actually executing the boot operation, and the -c option clears the command line, which not used by PowerMAX OS client netboot images.

NOTE

The hostname of this loosely-coupled client MUST be equal to the name of the client profile file that is located on the File Server SBC in the /etc/profiles directory.

10. This step will configure the board for either manual or automatic downloading and booting of the client.

    If you wish to always manually boot the client, then go to 'b)' below.

    a) To have ASTRix automatically attempt to boot the client, the /nvram/startup script must be installed. To create a new netboot startup script, you may either manually type in the entire startup script, or you may start with a startup script file template so that you only need to modify just a few lines of the already existing template.

    If you wish to use the startup script template file, then you need to first download the template version of the startup script from the file server SBC:

    On the file server SBC, issue the following command as root:

    ```
    netbootconfig -P astrix > /tftpboot/astrix
    ```

    Then on the client SBC console terminal, issue the following ASTRix commands:

    ```
    * cd / <cr>
    * tftp -g -r astrix $HOST <cr>
    * cp astrix /nvram/startup <cr>
    * cd /nvram <cr>
    * vi startup <cr>
    ```

    At this point, simply modify the “CLIENT=” line using vi, so that CLIENT is set to the hostname of this client. For example, if wilma is the hostname of this client, then change this line to:

    ```
    CLIENT=wilma
    ```

    The only other 2 lines that MAY need to be uncommented and modified are the routing entry lines. Only uncomment and setup these 2 lines if the client accesses the file server SBC through a gateway.

    For example, if the file server's network is 129.148.42.0 and it is accessed locally by the client through the gateway 129.148.43.196, with the client residing on the 129.148.43.0 network, then the routing lines in the startup script should be uncommented and modified to be:

```
route add -net 129.148.42.0 gw 129.148.43.196 \
netmask 255.255.255.0 dev eth0
```

To instead create the startup file from scratch instead of using the template file above, issue the following ASTRix commands on the client:

```
* cd /nvram <cr>
* rm startup <cr> (ignore any errors if this file didn't exist)
* vi startup <cr>
```

Then add the following lines to the startup file using the vi editor (the commented out route lines are not always necessary - see the comments below):

```
CLIENT=
# route add -net xxx.xxx.xxx.xxx gw xxx.xxx.xxx.xxx \
#     netmask 255.255.xxx.xxx dev eth0
while [ 1]
do
        echo “tftp -g -r /$CLIENT.bstrap $HOST”
        tftp -g -r /$CLIENT.bstrap $HOST
        case $? in
                0) break;
                *) sleep 5
                echo “retrying...”;;
        esac
done
echo “Booting client $CLIENT netboot image...”
sync
boot 5
```

Modify the “CLIENT=” line above using vi, so that CLIENT is set to the hostname of this client. For example, if wilma is the hostname of this client, then change this line to:

```
CLIENT=wilma
```

Enter “:wq” in vi to write the /nvram/startup file and exit the vi editor.

The two “route” lines in the above startup script above should only be used when the client accesses the file server through a gateway.

For example, if the file server's network is 129.148.42.0, and it is accessed locally through the gateway 129.148.43.196, with the client residing on the 129.148.43.0 network, then the above routing lines in the startup script should be uncommented and modified to be:

```
route add -net 129.148.42.0 gw 129.148.43.196 \
netmask 255.255.255.0 dev eth0
```

If the client and file server reside on the same network, then comment-out or leave these lines out of the startup file.

The tftp “while” loop that downloads the client's netboot image serves two purposes. The first purpose of the loop is to continually retry the tftp command should any errors occur during the transfer while the file server SBC is up and running. The second purpose is to accommodate situations when the client and file server SBCs are in the same rack and it is desirable to have the netboot clients boot themselves up after a power cycle sequence without the need for any additional manual resets. In order to ensure that a client can successfully download its boot image via tftp, the file server system MUST be completely booted into run level 3 before attempting to netboot any clients. Thus, the tftp retry loop gives the client the ability to actively wait for the file server to boot up and enter run level 3.

The execution of the startup script may be manually aborted by entering “<Ctrl>c” at the console terminal, should there be a need to do so. Once aborted, the startup script may be manually started by entering:

```
* /nvram/startup <cr>
```

The startup script file uses boot configuration number 5 to reset the board and begin execution of the client's netboot image; therefore, boot configuration number 5 MUST have been previously setup in step #9 above.

At this point, the ASTRix netboot client configuration setup has been completed. Once the file server has created this client's netboot image, entry the following command to reboot the system and test the new client's netboot setup:

```
* reboot
```

b) If you wish to always manually boot the client from ASTRix, then take the following steps:

If this board was previously setup with a /nvram/startup script for automatic netbooting, then remove this /nvram/startup file so that automatic netbooting will no longer occur:

```
* rm /nvram/startup <cr>
```

The above command only needs to be issued once, and only if the board was previously setup for automatic netbooting of the client.

If the client accesses the file server through a gateway, then enter the following ASTRix command. Otherwise, skip this command:

```
* route add -net xxx.xxx.xxx.xxx gw xxx.xxx.xxx.xxx \
      netmask xxx.xxx.xxx.xxx dev eth0
```

Replace the above 'net' and 'gw' and 'netmask' IP addresses with the appropriate values. The 'net' IP address should be the remote network that the file server resides on, and the 'gw' IP address should be the local gateway system that provides the client with access to the remote network. 'netmask' is typically set to a value of 255.255.255.0 for class 3 IP addressing.

To download the client netboot image from the file server, enter the following command:

```
* tftp -g -r /<client>.bstrap <cr>
```

where <client> should be replaced by the actual hostname of the client. If the tftp command fails, then you may re-enter the tftp command to try the download again.

Lastly, to boot the client, enter the following ASTRix commands:

```
* sync <cr>
* boot 5 <cr>
```

Note

Boot configuration slot number 5 MUST have been previously setup as specified in step #9. The file server SBC should be at run level 3 before you attempt to boot the netboot client image.

2.3. Modifying an Existing Automatic Netboot Configuration

If you have setup STAR/ASTRix to automatically boot the client and at some point you wish to modify this configuration, then shutdown the client if it is currently running PowerMAX OS, or otherwise reset or power-cycle the client board, and then:

issue a <Ctrl>c from the console keyboard when the:

tftp -g -r /<client>.bstrap $HOST

message appears under ASTRix. This will abort the file transfer from the file server. At this point, you may make modifications to the ASTRix post scripts and/or the ASTRix startup script and then either issue:

reboot

if you wish to reboot the client without any STAR configuration changes, or issue:

star

if you wish to enter STAR and modify one or more of the STAR '*config*' parameters. Once the STAR '*config*' parameters have been modified, you may enter:

STAR0> **reboot** <cr>

to reboot the board and to have the STAR '*config*' parameters take affect and have STAR automatically boot ASTRix.

2.4. Client Configuration

This section describes the steps using **`netbootconfig(1M)`**, for creating the environment on the File Server that are necessary for supporting loosely-coupled netboot diskless clients. Major topics described are:

- client profile files
- Client Configuration Using **`netbootconfig`** (page 2-16)

Information about each client is specified in a client profile file. The system administrator creates and updates these profile files for each netboot client, and then invokes **`netbootconfig(1M)`** to create, on the File Server system, the file environment necessary to support a private virtual root directory and a private boot image for each client.

2.4.1. The Client Profile File

For each client SBC installed in the cluster, a client profile file must be created in the **`/etc/profiles`** directory. This section explains the various parameters that are contained in a client profile file. All of the parameters located in a client profile file are specific to that one client SBC.

You should use **`netbootconfig(1M)`** to print out a starting template of a client profile with the "**`-P'`** option (which defaults to -P client.

So for example, to create a client profile for a client with a hostname of '*wilma*', do the following:

```
netbootconfig -P > /etc/profiles/wilma
vi /etc/profiles/wilma
```

You must then update the resulting client profile file, modifying the parameter values in the file to fit the specific characteristics of that loosely-coupled netboot client SBC.

The client profile file is loaded by the loosely-coupled tools and various start-up scripts. After the initial client is configured with **`netbootconfig(1M)`**, this file must not be modified.

Each required parameter must be assigned a value in the ksh-loadable format **`<parameter=value>`**. No spaces are allowed on either side of the equal sign. Parameters specified as optional may be left blank **`<parameter=>`**.

An explanation of each of the parameters follows.

2.4.1.1. Required Parameters

The following client profile parameters are required for all loosely-coupled netboot clients.

VROOT=

This parameter is the directory path name under which the client's virtual root directory will be created. The directory will be created if it doesn't already exist.

Example:

```
VROOT=/home/vroots/wilma
```

SYS_CONFIG=

This parameter specifies the client configuration to be either NFS or embedded. An NFS client is configured with networking, executes in multi-user mode and has the ability to swap memory pages to a remote swap area on the File Server. An embedded client does not have networking support, cannot swap out memory pages and runs in single user mode. This parameter should be set to either '**`nfs`**' or '**`emb`**'.

Example:

```
SYS_CONFIG=nfs
```

BOOT_IFACE=net

Specifies the networking interface used for loading a diskless client's boot image. This parameter is set to '**`net`**' by default, and the user should not modify this setting.

2.4.1.2. Required NFS-Related Parameters

The following netboot client profile parameters are required only if the client is a NFS client (SYS_CONFIG=nfs). The values in these parameters are ignored if the client is an embedded client (SYS_CONFIG=emb).

AUTOBOOT=

This parameter specifies whether this client should be shutdown whenever the File Server is shutdown. The value for this parameter should be either '**`y`**' or '**`n`**'. If set to '**`y`**', then the client will be shutdown by the File Server when ever the File Server is shutting itself down.

Example:

AUTOBOOT=y

NOTE When this parameter is set to '**`y`**', then a hidden file named **`.autoboot`** will be created by **`netbootconfig(1M)`** under this client's virtual root directory (the **`VROOT`** parameter path). This file will serve to indicate that the client SBC should be automatically shutdown by the File Server SBC whenever the File Server is shutdown. This **`.autoboot`** file may be manually removed or created in the client's virtual root directory, as needed.

SWAP_SIZE=

This parameter is the size, in megabytes, of remote swap space. Swap space is implemented as a file named **`<virtual_root>/dev/swap_file`**, which resides on the File Server in the client's virtual root directory, and which is accessed over NFS. The recommended value for this parameter is 1.5 times the size of the amount of physical memory located on the client's SBC.

Example:

SWAP_SIZE=192

ETHER_SUBNETMASK=

This parameter specifies the ethernet interface subnetmask in decimal dot notation (xxx.xxx.xxx.xxx).

Example:

ETHERNET_SUBNETMASK=255.255.255.0

GATEWAY_IPADDR=

The IP address, in decimal dot notation (xxx.xxx.xxx.xxx), of a gateway system that provides this client with access to the File Server. This parameter is optional, and it is only necessary to setup this parameter when the File Server and this netboot client do not reside on the same physical subnetwork.

Example:

GATEWAY_IPADDR=129.158.64.40

2.4.1.3. Hosts Tables

For each loosely-coupled client profile file created in the **`/etc/profiles`** directory, an entry for that client's onboard ethernet networking interface must be added to the systems **`hosts(4)`** file, **`/etc/hosts`**.

The client hostname added to the **`/etc/hosts`** file must match the client profile filename of the client. For example, if a new client profile file named **`/etc/profiles/fred`** has just been created, then an entry with a hostname of '**`fred`**' must be added to the **`/etc/hosts`** file.

The corresponding IP address for each new **`/etc/hosts`** entry should be chosen based on local rules for the ethernet subnet. This IP address should match the value entered for this client SBC under the STAR config command's "Target IP address" parameter, as described under step #5 in the "SBC Client Board Configuration" section.

2.4.2. Configuring Clients Using netbootconfig

The **netbootconfig(1M)** tool is used to create, remove or update one or more diskless client configurations. For more details on running this tool, see the manual page available online.

Netbootconfig(1M) gathers information from the client profile files and stores this information in a ksh-loadable file, named**.client_profile**, under the client's virtual root directory. The**.client_profile** is used by **netbootconfig(1M)**, by other configuration tools and by the client initialization process during system startup. It is accessible on the client from the path **/.client_profile**.

Netbootconfig(1M) appends a process progress report and run-time errors to the client-private log file, **/etc/profiles/<client_hostname>.log**, on the File Server, or if invoked with the **-t** option, to stdout.

With each invocation of the tool, an option stating the mode of execution must be specified. The modes are create client (**-C**), remove client (**-R**) and update client (**-U**).

2.4.2.1. Creating and Removing a Client Configuration

When creating new client configurations, the client profile parameters must already be set up in the **/etc/profiles** client profile files (see "The Client Profile File" on page 2-13) before using **netbootconfig(1M)** to create the new client configurations. The **/etc/hosts** file should also already contain the appropriate entries for the new netboot clients (see "Hosts Tables" on page 2-15 for more details).

By default, when run in create mode (**-C** option), **netbootconfig(1M)** performs the following tasks:

- Populates a client-private virtual root directory.
- Modifies client-private configuration files in the virtual root.
- Creates the **<virtual_rootpath>/.client_profile**
- Modifies the **dfstab(4C)** table and executes the **shareall(1M)** command to give the client permission to access, via NFS, its virtual root directory and system files that reside on the File Server.
- Creates the client-private custom directory - **/etc/clients/<client_hostname>.net/custom.conf**, where the **<client_hostname>** is equal to the name of the client profile file.

  For example, if a client's client profile filename is '**fred**', then the client's custom directory would be:

  /etc/clients/fred.net/custom.conf

By default, when run in remove mode (**-R** option), **netbootconfig(1M)** performs the following tasks:

- Removes the virtual root directory.
- Removes client's name from the **`dfstab(4C)`** tables and executes an **`unshare(1M)`** of the virtual root directory.
- Removes the client-private log file - **`/etc/profiles/<client_hostname>.log`**.
- Removes the client-private custom directory - **`/etc/clients/<client_hostname>.net/custom.conf`**.

The update option (**`-U`**) indicates that the client's environment already exists and, by default, nothing is done. The task to be performed must be indicated by specifying additional options. For example, one might update the files under the virtual root directory. Some examples are shown below.

Example 1.

Create the diskless client configurations for all clients that have netboot client profile files in the **`/etc/profiles`** directory. Process at most three clients at the same time.

```
netbootconfig -C -p3 all
```

Example 2.

Remove the client virtual root configuration of netboot client '*rosie*'.
Send the output to stdout instead of to the client's log file.

```
netbootconfig -R -t rosie
```

Example 3.

Update the virtual root directories of netboot clients '*fred*' and '*barney*'. Process one client at a time.

```
netbootconfig -U -v -p 1 fred barney
```

2.4.2.2. Subsystem Support

A subsystem is a set of software functionality (package) that is optionally installed on the File Server during system installation or via the **`pkgadd(1M)`** utility. Additional installation steps are sometimes required to make the functionality of a package usable on a diskless client.

Subsystem support is added to a diskless client configuration via **`netbootconfig(1M)`** options, when invoked in either create or update mode. Subsystem support is added to a client configuration via the **`-a`** option and removed via the **`-r`** option. For a list of the current subsystems supported see the **`netbootconfig(1M)`** manual page or invoke **`netbootconfig(1M)`** with the help option (**`-h`**).

If the corresponding package software was added on the File Server after the client's virtual root was created, you must first bring the client's virtual root directory up to date by using the **-v** option of **netbootconfig(1M)** before adding subsystem support.

Example 1:

Create netboot client '*wilma's*' configuration and also add support for the RCFBS subsystem:

> **netbootconfig -C -a RCFBS** *wilma*

Example 2:

Remove support for the RCFBS subsystem from netboot clients '*wilma*' and '*fred*':

> **netbootconfig -U -r RCFBS** *wilma fred*

2.5. Customizing the Basic Client Configuration

This section contains information on the following major topics:

- Modifying the Kernel Configuration (page 2-18)
- Custom Configuration Files (page 2-20)
- Modifying the Client Profile Parameters (page 2-28)
- Launching Applications (page 2-29)
  - Launching an Application (Embedded Clients) (page 2-29)
  - Launching an Application (NFS clients) (page 2-30)

2.5.1. Modifying the Kernel Configuration

A diskless client's kernel configuration directory is resident on the File Server and is a part of the client's virtual root partition. Initially, it is a copy of the File Server's **/etc/conf** directory. The kernel object modules are symbolically linked to the File Server's kernel object modules to conserve disk space.

By default, a client's kernel is configured with a minimum set of drivers to support the chosen client configuration. The set of drivers configured by default for an NFS client and for an embedded configuration are listed in **modlist.nfs.netboot** and **modlist.emb.netboot** respectively, under the directory path **/usr/etc/diskless.d/sys.conf/kernel.d**. These template files should not be modified.

For diskless clients, only one copy of the unix file (the kernel object file) is kept under the virtual root. When a new kernel is built, the current unix file is over-written. System diagnostic and debugging tools, such as **crash(1M)** and **hwstat(1M)**, require access to the unix file that matches the currently running system. Therefore, if the kernel is being modified while the client system is running and the client is not going to be immediately rebooted with the new kernel, it is recommended that the current unix file be saved.

Modifications to a client's kernel configuration can be accomplished in various ways. All the commands referenced below should be executed on the file server system.

1. Additional kernel object modules can be automatically configured and a new kernel built by specifying the modules in the **`kernel.modlist.add`** custom file and then invoking **`mknetbstrap(1M)`**. The advantage of this method is that the client's kernel configuration is recorded in a file that is utilized by **`mknetbstrap(1M)`**. This allows the kernel to be easily re-created if there is a need to remove and recreate the client configuration.
2. Kernel modules may be manually configured or deconfigured using options to **`mknetbstrap(1M)`**.
3. All kernel configuration can be done using the **`config(1M)`** utility and then rebuilding the unix kernel.
4. The **`idtuneobj(1M)`** utility may be used to directly modify certain kernel tunables in the specified unix kernel without having to rebuild the unix kernel.

2.5.1.1. kernel.modlist.add

The **`kernel.modlist.add`** custom table is used by the boot image creating tool, **`mknetbstrap(1M)`** for adding user-defined extensions to the standard kernel configuration of a client system. When **`mknetbstrap(1M)`** is run, it compares the modification date of this file with that of the unix kernel. If **`mknetbstrap(1M)`** finds the file to be newer than the unix kernel, it will automatically configure the modules listed in the file and rebuild a new kernel and boot image. This file may be used to change the kernel configuration of one client or all the clients. For more information about this table, see "Custom Configuration Files" on page 2-20.

2.5.1.2. mknetbstrap

Kernel modules may be configured or deconfigured via the **`-k`** option of **`mknetbstrap(1M)`**. A new kernel and boot image is then automatically created. For more information about **`mknetbstrap(1M)`**, see the online manual page.

2.5.1.3. config utility

The **`config(1M)`** tool, may be used to modify a client's kernel environment. It can be used to enable additional kernel modules, configure adapter modules, modify kernel tunables, or build a kernel. You must use the **`-r`** option to specify the root of the client's kernel configuration directory.

If you do not specify the **`-r`** option, you will modify the File Server's kernel configuration instead of the client's. For example, if the virtual root directory for client *rosie* was created under **`/vroots/rosie`**, then invoke **`config(1M)`** as follows:

```
config -r /vroots/rosie
```

After making changes using **`config(1M)`**, a new kernel and boot image must be built. There are two ways to build a new boot image:

1. Use the Rebuild/Static menu from within **`config(1M)`** to build a new unix kernel and then invoke **`mknetbstrap(1M)`**. **`mknetbstrap(1M)`** will find the boot image out-of-date compared to the newly built unix file and will automatically build a new boot image.
2. Use **`mknetbstrap(1M)`** and specify "unix" on the rebuild option (**`-r`**).

2.5.1.4. idtuneobj

In situations where only kernel tunables need to be modified for an already built host and/or client kernel(s), it is possible to directly modify certain kernel tunable values in a client and/or host unix object files without the need for rebuilding the kernel.

The **`idtuneobj(1M)`** utility may be used to directly modify certain kernel tunables in the specified unix or Dynamically Linked Module (DLM) object files.

The tunables that **`idtuneobj(1M)`** supports are contained in the **`/usr/lib/idtuneobj/tune_database`** file and can be listed using the **`-l`** option of **`idtuneobj(1M)`**.

The **`idtuneobj(1M)`** utility can be used interactively, or it can process an ASCII command file that the user may create and specify.

Although the unix kernel need not be rebuilt, the tunable should be modified in the client's kernel configuration (see the "config utility" section above) to avoid losing the update the next time a unix kernel is rebuilt.

Refer to the **`idtuneobj(1M)`** online man page for additional information.

2.5.2. Custom Configuration Files

The files installed under the **`/usr/etc/diskless.d/cluster.conf/custom.conf`** directory may be used to customize a diskless client system configuration.

In some cases a client's configuration on the File Server may need to be removed and re-created. This may be due to file corruption in the client's virtual root directory or because of changes needed to a client's configuration. In such cases, the client configuration described by these files may be saved and used again when the client configuration is re-created. The **`-s`** option of **`netbootconfig(1M)`** must be specified when the client configuration is being removed to prevent these files from being deleted.

The custom files listed below and described in-depth later in this section, are initially installed under the '**`nfs`**' and '**`emb`**' directories under the **`/usr/etc/diskless.d/cluster.conf/custom.conf`** directory. Some of these files are installed as empty templates, while others contain the entries needed to generate the basic diskless system configuration. The files used for client customizing include:

`K00client`	to execute commands during system start-up
`S25client`	to execute commands during system shutdown
`memfs.inittab`	to modify system initialization and shutdown
`inittab`	to modify system initialization and shutdown (nfs clients only)
`vfstab`	to automatically mount file systems (nfs clients only)
`kernel.modlist.add`	to configure additional modules into the unix kernel
`memfs.files.add`	to add files to the memfs **`/`** (root) file system
`vroot.files.add`	to make a copy of specific non-system files in the client's virtual root directory (nfs clients only)

When a client is configured using **`netbootconfig(1M)`**, a directory is created specifically for that client under the **`/etc/clients`** directory. The client's custom configuration files are installed under this client's **`custom.conf`** directory, **`/etc/clients/<client_dir>/custom.conf`**, and are initially linked to the files in the cluster's custom.conf directory - **`/usr/etc/diskless.d/cluster.conf/custom.conf/nfs|emb`**.

When the client is a netboot client, then the name of the **`<client_dir>`** will be of the format:

```
<client_profile_filename>.net
```

So for example, if the client profile file named '**`fred`**' is for a netbooted client, then the corresponding private client directory name will be:

```
/etc/clients/fred.net/custom.conf
```

The files in these client-private directories are initially shared with the other embedded or nfs clients; therefore a change to one of these files will affect all the clients in the loosely-coupled system.

NOTE

Please note that if the File Server is a Power Hawk Series 700 system and it is also supporting Series 700 closely-coupled clients, then changes to the shared custom.conf files also affect the File Server's closely-coupled clients as well as the File Server's loosely-coupled clients. See the *Power Hawk Series 700 Diskless Systems Administrator's Guide* for more details on closely-coupled clients.

Under each client's private **`custom.conf`** directory two commands, **`mkprivate`** and **`mkshared`**, are available to change the state of a custom file from being shared to being private. Before creating a new version, **`mkshared`** will save the current version to a file named **`<customfile>.old`**, **`mkprivate`** will move the current version to a file named **`<customfile>.linked`**.

To make a change that is private to a client:

1. verify that the custom file is NOT symbolically linked

```
# cd /etc/clients/<client>.net/custom.conf
# ls -l <customfile>
```

2. if the file is currently symbolically linked, first break the link

```
#./mkprivate <customfile>
```

3. verify that the file is a regular file and edit the file

```
# ls -l <customfile>
# vi <customfile>
```

To make a change that will affect all the diskless clients configured to share this custom file:

1. Make the changes to the shared file (type is either nfs or emb):

```
# vi /etc/clients/cluster.conf/custom.conf/<type> \
/<customfile>
```

2. For each client to share these changes:

   a. Verify that the custom file is symbolically linked to the file edited above:

   ```
   # cd /etc/clients/<client>.net/custom.conf
   # ls -l <customfile>
   ```

   b. If the file is not currently symbolically linked, then re-link it:

   ```
   #./mkshared <customfile>
   ```

   c. Verify that the file is now symbolically linked:

   ```
   # ls -l <customfile>
   ```

For example, to make private changes to the **`K00client`** script for a netboot client named '*wilma*':

```
cd /etc/clients/wilma.net/custom.conf
./mkprivate K00client
vi K00client
```

Changes to the customization files are processed the next time the boot image generating utility, **`mknetbstrap(1M)`**, is invoked. If **`mknetbstrap(1M)`** finds that a customization file is out-of-date compared to a file or boot image component, it will implement the changes indicated. If applicable (some changes do not affect the boot image), the boot image component will be rebuilt and a new boot image will be generated.

The customization files are described below in terms of their functionality.

2.5.2.1. S25client and K00client rc Scripts

Commands added to these **`rc`** scripts will be executed during system initialization and shutdown. The scripts must be written in the Bourne Shell (**`sh(1)`**) command language.

These scripts are available to both NFS and embedded type client configurations. Since embedded configurations run in **`init level 1`** and NFS configurations run in **`init level 3`**, the start-up script is executed from a different **`rc`** level directory path depending on the client configuration.

Any changes to these scripts are processed the next time the **`mknetbstrap(1M)`** utility is invoked on the File Server. For embedded clients, a new **`memfs.cpio`** image and a new boot image is generated. An embedded client must be rebooted using the new boot image in order for these changes to take effect.

For NFS clients, the modified scripts will be copied into the client's virtual root and are accessed by the client during the boot process via NFS. Therefore, the boot image does not need to be rebuilt for an NFS client and the changes will take effect the next time the system is booted or shutdown.

These scripts may be updated in one of the two subdirectories (**`nfs`** or **`emb`**) under the **`/usr/etc/diskless.d/cluster.conf/custom.conf`** directory so that the changes apply globally to all clients. If the customizing is to be applied to a specific client, the customized **`rc`** file should be created in the **`/etc/clients/<client_profile_filename>.net/custom.conf`** directory. If there is already an existing shared customization file, and those customizations should also be applied to this client, then a private copy of the shared **`rc`** file should be created with the **`mkprivate`** tool script in the clients's **`custom.conf`** directory and edited there.

K00client	Script is executed during system shutdown. It is executed on the client from the path **`/etc/rc0.d/K00client`**. By default this file is empty.
S25client	Script is executed during system start-up. It is executed on a client configured with NFS support from the path **`/etc/rc3.d/S25client`**. For embedded configurations, it is executed from **`/etc/rc1.d/S25client`**. By default this file is empty.

2.5.2.2. memfs.inittab and inittab Tables

These tables are used to initiate execution of programs on the client system. Programs listed in these files are dispatched by the init process according to the **`init level`** specified in the table entry. When the system initialization process progresses to a particular init level the programs specified to run at that level are initiated. It should be noted that embedded clients can only execute at **`init level 1`**, since an embedded client never proceeds beyond **`init level 1`**. NFS clients can execute at **`init levels 1`**, **`2`** or **`3`**. **`Init level 0`** is used for shutting down the system. See the online man page for **`inittab(4)`** for more information on init levels and for information on modifying this table.

The **`memfs.inittab`** table is a part of the memory-based file system, which is a component of the boot image. Inside the boot image, the files to be installed in the memory-based file system are stored as a compressed cpio file. When the

memfs.inittab file is modified a new **memfs.cpio** image and a new boot image will be created the next time **mknetbstrap(1M)** is invoked. A client must be rebooted using the new boot image in order for any changes to take effect.

Any programs to be initiated on an embedded client must be specified to run at **init level 1**. NFS clients may use the **memfs.inittab** table for starting programs at **init levels 1-3**. However, part of the standard commands executed at **init level 3** on an NFS client is the mounting of NFS remote disk partitions. At this time, an NFS client will mount its virtual root. The memfs-based **/etc** directory is used as the mount point for the **<virtual_root>/etc** directory that resides on the File Server. This causes the **memfs.inittab** table to be replaced by the **inittab** file. This means that any commands to be executed in **init state 0** (system shutdown) or commands which are to be respawned in **init state 3**, should be added to both the **memfs.inittab** and the **inittab** file if they are to be effective.

After configuring an NFS client system, the **inittab** table contains entries that are needed for the basic operation of a diskless system configuration. The default entries created by the configuration utilities in the **inittab** file should not be removed or modified.

Changes to **inittab** are processed the next time **mknetbstrap(1M)** is invoked. The **inittab** table is copied into the client's virtual root and is accessed via NFS from the client system. Therefore, the boot image does not need to be rebuilt after modifying the **initab** table and the changes to this table will take effect the next time the system is booted or shutdown.

Like the other customization files, these tables may be updated in one of the two subdirectories (**nfs** or **emb**). Changes made under the **/usr/etc/diskless.d/cluster.conf/custom.conf** directory apply globally to all nfs or embedded clients that share this File Server. If the changes are specific to a particular client, then a private copy of the shared file should first be created in that client's private customization directory by using the **mkprivate** tool, and then edited in that client's **custom.conf** directory.

2.5.2.3. vfstab Table

The **vfstab** table defines attributes for each mounted file system. The **vfstab** table applies only to NFS client configurations. The **vfstab(4)** file is processed when the **mountall(1M)** command is executed during system initialization to **run level 3**. See the **vfstab(4)** online manual page for rules on modifying this table.

Configuring an NFS client configuration causes this table to be installed with entries needed for basic diskless system operation and these entries should not be removed or modified.

The **vfstab** table is part of the client's virtual root and is accessed via NFS. The boot image does not need to be rebuilt after modifying the **vfstab** table, the changes will take effect the next time the system is booted or shutdown.

Like other NSF-only customization files, these tables may be updated in the **client-shared nfs** subdirectory. Changes made under the **/usr/etc/diskless.d/cluster.conf/custom.conf/nfs** directory apply globally to all NFS clients that share this File Server. If the changes are specific to a particular client, then a private copy of the shared file should first be created in that client's private customization directory by using the **`mkprivate`** tool, and then edited in that client’s **`custom.conf`** directory.

2.5.2.4. kernel.modlist.add Table

New kernel object modules may be added to the basic kernel configuration using the **`kernel.modlist.add`** file. One module per line should be specified in this file. The specified module name must have a corresponding system file installed under the **`<virtual_rootpath>/etc/conf/sdevice.d`** directory. For more information about changing the basic kernel Configuration, see “Modifying the Kernel Configuration” on page 2-18.

Changes to this file are processed the next time **`mknetbstrap(1M)`** is invoked, causing the kernel and the boot image to be rebuilt. When modules are specified that are currently not configured into the kernel (per the module’s **`System(4)`** file), those modules will be enabled and a new unix and boot image will be created. If **`mknetbstrap(1M)`** finds that the modules are already configured, the request will be ignored. A client must be rebooted using the new boot image in order for these changes to take effect.

Like the other customization files, these tables may be updated in one of the two subdirectories (**`nfs`** or **`emb`**). Changes made under the **`/usr/etc /diskless.d/custom.conf/client.shared/`** directory apply globally to all NFS or embedded clients that share this File Server. If the changes are specific to a particular client, then a private copy of the shared file should first be created in the client’s private customization directory, by using the **`mkprivate`** tool, and then edited in that client’s **`custom.conf`** directory.

2.5.2.5. memfs.files.add Table

When the **`mknetbstrap(1M)`** utility builds a boot image, it utilizes several files for building the compressed cpio file system. The set of files included in the basic diskless memory-based file system are listed in the files **`devlist.nfs.netboot`** and **`filelist.nfs.netboot`** for NFS clients and **`devlist.emb.netboot`** and **`filelist.emb.netboot`** for embedded clients under the **`/usr/etc /diskless.d/sys.conf/memfs.d`** directory.

Additional files may be added to the memory-based file system via the **`memfs.files.add`** table located under the **`/usr/etc/diskless.d/ cluster.conf/custom.conf`** directory. Like the other customization files, this tables may be updated in one of the two subdirectories (**`nfs`** or **`emb`**). Changes made under the **`/usr/etc/diskless.d/cluster.conf/ custom.con`**f directory apply globally to all nfs or embedded clients that share this File Server. If the changes are specific to a particular netboot client, a private copy of the shared **`memfs.files.add`** file should first be created in that client's private customization directory, **`/etc/profiles/<client_profile_filename>.net/custom.conf`**, by using the **`mkprivate`** tool, and then editing it in that **`custom.conf`** directory.

Guidelines for adding entries to this table are included as comments at the beginning of the table.

A file may need to be added to the **`memfs.files.add`** table if:

1. The client is configured as embedded. Since an embedded client does not have access to any other file systems, then all user files must be added via this table.
2. The client is configured with NFS support and:
   a. the file needs to be accessed early during a diskless client's boot, before **`run level 3`** when the client is able to access the file on the File Server system via NFS
   b. it is desired that the file is accessed locally rather than across NFS.

For NFS clients the system directories **`/etc`**, **`/usr,`** **`/sbin`**, **`/dev`**, **`/var`**, **`/opt`** and **`/tmp`** all serve as mount points under which remote file systems are mounted when the diskless client reaches **`run level 3`**. Files added via the **`memfs.files.add`** table should not be installed under any of these system directories if they need to be accessed in **`run level 3`** as the NFS mounts will overlay the file and render it inaccessible.

Files added via the **`memfs.files.add`** table are memory-resident and diminish the client's available free memory. This is not the case for a system where the boot image is stored in flash using the "raw write" method with the ASTRix "fw" flash write command, since pages are brought into DRAM memory from flash only when referenced. Please refer to the "Flash Boot System Administration" chapter for more details on flash booting.

Changes to the **`memfs.files.add`** file are processed the next time **`mknetbstrap(1M)`** is invoked. A new memfs image and boot image is then created. A client must be rebooted using the new boot image in order for these changes to take effect.

You can verify that the file has been added to the **`memfs.cpio`** image using the following command on the File Server:

```
rac -d < <virtual_rootpath>/etc/conf/cf.d/memfs.cpio \
| cpio -itcv|grep <file>
```

2.5.2.6. vroot.files.add Table

This custom client configuration table may be used to optionally specify a set of non-system files that are located on the File Server to be automatically copied by **`mknetbstrap(1M)`** into a a client's virtual root directory so that they can be subsequently accessed from the client system.

This custom client configuration file may only be used by NFS netboot clients (the embedded netboot clients are unable to access their virtual root on the File Server system).

This table is processed by **`mknetbstrap(1M)`** whenever this table has been modified since the last invocation of **`mknetbstrap(1M)`**.

Although non-system files can be copied manually into a client's virtual root directories, the use of this table provides an automated method that provides the following advantages:

- This file table makes it easier to recreate a client's virtual root environment when a client is removed (**`-R`** and **`-s`** options) and then recreated (**`-C`** option) with **`netbootconfig(1M)`**.

- Entries in this file table may be setup to have **mknetbstrap(1M)** automatically re-copy the specified File Server source files into the client target virtual root directories every time this table is processed, with the '**a**' option (see below).

The format for each entry in this file is:

```
Path_on_server   Path_on_client   Options
```

Lines beginning with the pound sign '#' will be ignored. The fields in this table are described below:

Path_on_server:

This is the pathname of a file or directory located on the File Server system that is to be copied into the client's virtual root. When the pathname is a directory, then the contents of this directory will be recursively copied into the client's vroot directory.

Path_on_client:

This is the pathname of a file or a directory as it will be accessed from the client system. If a directory in this path does not currently exist in the client's virtual root directory, then it is created. This path must begin with one of the system directories already under the client's vroot: **/users**, **/dev**, **/etc**, **/tmp**, or **/var**. Any files in **/tmp** and **/var/tmp** are destroyed when the client system is rebooted. A dash "**-**" in this field may be used to indicate that the path name is the same as that specified for the "**Path on server**" field.

Options:

a	The always option. Update the file or directory each time this table is processed.
o	The once option. Install the file or directory only if it doesn't already exist.

Some example **vroot.files.add** entries are shown below.

Example 1.

This example specifies that the files contained in the directory **/home/me/test.dir** on the File Server system should be copied into the client's virtual root directory: **<client_virtual_root>/users/me/test.dir** whenever **mknetbstrap(1M)** processes this file (the '**a**' option):

```
/home/me/test.dir /users/me/test.dir a
```

Example 2.

This example specifies that the single file **/home/me/timer.c**, located on the File Server system, should be copied into **<client_virtual_root>/users/me/timer.c** whenever **mknetbstrap(1M)** processes the **vroot.files.add** file (the '**a**' option):

```
/home/me/timer.c  /users/me/timer.c  a
```

Example 3.

This example specifies the single file **/etc/appl1** on the File Server system should be copied to the **<client_virtual_root>/etc/appl1** if the target file does not already exist in the client's virtual root directory (‘**o**’ option):

```
/etc/appl1  -   o
```

The following are some additional considerations for adding entries to the **vroot.files.add** table:

The client's **/usr** and **/sbin** system directories are shared completely with the File Server; hence, these directories do not appear under a client's virtual root and may not be used in the **vroot.files.add** table.

The files in the **vroot.files.add** table are copied into a client's virtual root partition and therefore require disk space on the File Server system. In some cases it may be more efficient to NFS mount a user's working directory on the client system instead of duplicating the files in the client's virtual root directory.

Because of kernel dependencies, device files should be created locally in the client's virtual root directory; this **vroot.files.add** file table should NOT be used for this purpose.

To add a device file to a client's vroot, the corresponding kernel module must be enabled (**config -r <vroot_path>**), the corresponding **Node(4)** file under the client's vroot may need to be modified, and the client's kernel must be rebuilt and rebooted (**mknetbstrap -B -r unix <client_profile_filename>**).

2.5.3. Modifying the Client Profile Parameters

To modify the parameter values in a netboot client profile file, the client configuration must be removed and reconfigured. The only exceptions to this rule are the AUTOBOOT and SWAP_SIZE parameters (discussed separately below).

It is best that the netboot NFS client be shutdown before modifying any of its client profile parameters.

As an example, to modify most parameters in netboot client *wilma's* client profile file, take the following steps to remove, modify and re-create and client's configuration:

1. Remove the current client configuration for *wilma*, but preserve any client customization files for client *wilma*:

   netbootconfig -R -s *wilma*

2. Edit *wilma's* client profile file as needed:

   vi /etc/profiles/wilma

3. Re-create the configuration for client *wilma*:

   `netbootconfig -C` *wilma*

The client profile file parameters that MAY be modified without the need to remove and reconfigure the client are described below. These parameters only apply to NFS netboot clients. For these parameters, the actual client profile parameter value within the client's profile file are NOT modified; only the actual objects that these parameters act upon are modified.

AUTOBOOT

This parameter is implemented as a hidden file named **`.autoboot`** directly under the client's virtual root directory. This hidden file may be created and removed to enable or disable, respectively, the automatic shutdown of the client when the File Server shuts down. In this case, it is not necessary to modify the actual client profile file in order to modify this setting. See the section “The Client Profile File” on page 2-13 for more details on the AUTOBOOT parameter.

SWAP_SIZE

For NFS clients, a different sized **`dev/swap_file`** file from the one specified in the client's profile file may be created by invoking the **`mkswap`** command:

```
/usr/etc/diskless.d/sys.conf/bin.d/mkswap <vrootpath> <megabytes>
```

NOTE

As previously mentioned, the client should be first shutdown before issuing the **`mkswap`** command, if the NFS client is currently up and running.

2.5.4. Launching Applications

Following are descriptions on how to launch applications for:

- Embedded Clients
- NFS Clients

2.5.4.1. Launching an Application for Embedded Clients

For diskless embedded clients, all the application programs and files referenced must be added to the memfs root file system via the **`memfs.files.add`** file. See section “memfs.files.add Table” on page 2-25, for more information on adding files via the **`memfs.files.add`** file.

As an example, the command name **`myprog`** resides on the File Server under the path **`/home/myname/bin/myprog`**. We wish to automatically have this command executed from the path **`/sbin/myprog`** when the client boots. This commands reads from a data file expected to be under **/myprog.data**. This data file is stored on the File Server under **/home/myname/data/myprog.data**.

The following entries are added to the **memfs.files.add** table:

```
f   /sbin/myprog 0755   /home/myname/bin/myprog
f   /myprog.data 0444   /home/myname/data/myprog.data
```

The following entry is added to the client's start-up script:

```
#
# Client's start-up script
#
/sbin/myprog
```

See "Custom Configuration Files" on page 2-20 for more information about the **memfs.files.add** table and the **S25client rc** script.

2.5.4.2. Launching an Application for NFS Clients

Clients configured with NFS support may either add application programs to the memfs root file system or they may access applications that reside on the File Server across NFS. The advantage to using the memfs root file system is that the file can be accessed locally on the client system rather than across the network. The disadvantage is that there is only limited space in the memfs file system. Furthermore, this file system generally uses up physical memory on the client system. When the client system is booted from an image stored in flash using the "raw write" method with the ASTRix "fw" flash write command, this is not the case, since the memfs file system remains in flash until the pages are accessed and brought into memory.

To add files to the memfs root file system follow the procedures for an embedded client above.

When adding files to the client's virtual root so that they can be accessed on the client via NFS, the easiest method is to place the file(s) in one of the directories listed below. This is because the client already has permission to access these directories and these directories are automatically NFS mounted during the client's system initialization.

Storage Path on File Server	Access Path on the Client
/usr	**/usr**
/sbin	**/sbin**
/opt	**/opt**
<virtual_root>/etc	**/etc**
<virtual_root>/var	**/var**
<virtual_root>/users	**/users**

As an example, the command name **myprog** was created under the path **/home/myname/bin/myprog**. To have this command be accessible to all the diskless clients on the File Server we could **mv(1)** or **cp(1)** the command to the **/sbin** directory.

```
mv /home/myname/bin/myprog /sbin/myprog
```

If only one client needs access to the command, it could be moved or copied to the **/etc** directory in that client's virtual root directory.

```
mv /home/myname/bin/myprog <virtual_root>/etc/myprog
```

To access an application that resides in directories other than those mentioned above, the File Server's directory must be made accessible to the client by adding it to the **dfstab(4)** table and then executing the **share(1M)** or **shareall(1M)** command on the File Server. To automatically have the directories mounted during the client's system start-up, an entry must be added to the client's **vfstab** file. See "Custom Configuration Files" on page 2-20 for more information about editing the **vfstab** file.

2.6. Booting and Shutdown

This section describes the following major topics:

- The Boot Image (page 2-31)
- Creating the Boot Image (page 2-32)
- Net Booting (page 2-33)
- Verifying Boot Status (page 2-34)
- Shutting Down the Client (page 2-35)

2.6.1. The Boot Image

The boot image is the file that is loaded from the File Server to a diskless client. The boot image contains everything needed to boot a diskless client. The components of the boot image are:

- unix kernel binary
- compressed cpio archive of a memory-based file system
- a bootstrap loader that uncompresses and loads the unix kernel

Each diskless client has a unique virtual root directory. Part of that virtual root is a unique kernel configuration directory (**etc/conf**) for each client. The boot image file (**unix.bstrap**), in particular two of its components: the kernel image (**unix**) and a memory-based file system (**memfs.cpio**), are created based on configuration files that are part of the client's virtual root.

The makefile, **/etc/diskless.d/sys.conf/bin.d/bstrap.makefile**, is used by **mknetbstrap(1M)** to create the boot image. Based on the dependencies listed in that makefile, one or more of the following steps may be taken by **mknetbstrap(1M)** in order to bring the boot image up-to-date.

1. Build the unix kernel image and create new device nodes.

2. Create and compress a cpio image of the files to be copied to the memfs root file system.
3. Insert the loadable portions of the unix kernel, the bootstrap loader, the compressed cpio image and certain bootflags into the **`unix.bstrap`** file. The unix kernel portion in **`unix.bstrap`** is then compressed.

When **`mknetbstrap`** is invoked, updates to key system files on the File Server (i.e. **`/etc/inet/hosts`**) will cause the automatic rebuild of one or more of the boot image components. In addition, updates to user-configurable files also affect the build of the boot image. A list of the user-configurable files and the boot image component that is affected when that file is modified are shown in Table 2-1. These files are explained in detail under section “Customizing the Basic Client Configuration” on page 2-18.

Table 2-1. Boot Image Dependencies

Boot Image Component	**User-Configurable File**
`unix kernel`	**`kernel.modlist.add`**
`memfs cpio`	**`memfs.files.add`**
	`memfs.inittab`
	`K00client` **`(embedded client configurations only)`**
	`KS25client` **`(embedded client configurations only)`**

The boot image components are created under **`etc/conf/cf.d`** in the client’s virtual root directory. The boot image itself is installed into the **`/tftpboot`** directory, under the name of **`<client_profile_filename>.bstrap`**. For example, a netboot client with a client profile filename of **`target1`** would have a boot image installed with a pathname of:

```
/tftpboot/target1.bstrap
```

2.6.2. Creating the Boot Image

The **`mknetbstrap(1M)`** tool is used to build the boot image. This tool gathers information about the client(s) from each client’s client profile file, located in the **`/etc/profiles`** directory. Some example uses follow. Building a boot image is resource-intensive. When creating the boot image of multiple clients in the same call, use the **`-p`** option of **`mknetbstrap(1M)`** to limit the number of client boot images which are simultaneously processed.

2.6.2.1. Examples on Creating the Boot Image

Example 1.

Update the boot image of all the clients configured with netboot client profile files in the **`/etc/profiles`** directory. Limit the number of clients processed in parallel to 2.

```
mknetbstrap -p2 all
```

Example 2.

Update the boot image of clients *wilma* and *fred*. Force the rebuild of the unix kernels and configure the boot images to stop in **`kdb`** early during system initialization.

```
mknetbstrap -r unix -b kdb wilma fred
```

Example 3.

Update the boot image of all the clients configured with netboot client profile files in the **`/etc/profiles`** directory. Rebuild their **`unix`** kernel with the **`kdb`** module configured and the **`rtc`** kernel module deconfigured. Limit the number of clients processed in parallel to 3.

```
mknetbstrap -p 3 -k kdb  -k "-rtc" all
```

2.6.3. Net Booting

Netboot diskless clients boot from an image downloaded via an ethernet network connection. Net booting (also referred to as Network booting) is performed by the **`ASTRix`** ROM based firmware using the TFTP (Trivial File Transfer Protocol, RFC783) network protocol.

All netboot diskless clients depend on the File Server for the creation and storage of the boot image. Once booted, netboot clients configured with NFS support continue to rely on the File Server for accessing their system files via NFS. Clients configured as embedded do not depend upon the File Server system once they are up and running.

NOTE

A netboot client may download the boot image and, instead of booting from it, may burn the boot image into its User Flash Memory for later booting. This is called Flash booting and it is described in a separate chapter. Refer to Chapter 3, "Flash Boot System Administration" for more information on Flash Booting.

Prior to net booting, verify that the following steps have been completed:

1. Verify that the networking parameters have been setup on the client board with the STAR 'config' command.

   If the client is to automatically boot up after reset or power-cycle, then verify that STAR is configured to automatically boot ASTRix, and that an ASTRix startup script has been setup to download and boot the client image from the File Server. See the "SBC Client Board Configuration" section for details.

2. Verify that the boot image has been created. (See "Creating the Boot Image" on page 2-32.)

3. Verify that the File Server is up and running in **`run level 3`**.

2.6.3.1. Netboot Using ASTRix

Once the boot image is generated and the File Server is accepting network requests (is up and is at **init state 3**), you can test the network booting of a client by one of the following methods:

- If the client is configured to autoboot ASTRix, and an ASTRix startup script is setup to tftp download and boot the client's netboot image, then test the booting of the client by either resetting or power-cycling the board.

- If the client is not configured to autoboot, then type the following commands at the client's console terminal:

STAR0> astrix <cr>
* tftp -g -r /<client_profile_filename>.bstrap $HOST <cr>
* boot 5 <cr>

where <client_profile_filename> is equal to the client profile filename of the client on the File Server system. Boot configuration slot number 5 should have already been setup as specified in the section "SBC Client Board Configuration".

Verifying Boot Status

If the client is configured with NFS support, you can verify that the client was successfully booted using any one of the following methods:

- **`rlogin(1)`** or **`telnet(1)`** from the File Server system, or
- attach a terminal to the console serial port and login.

You can also use the **`ping(1M)`** command to verify that the network interface is running. However this does not necessarily mean that the system successfully booted.

If the client does not boot, verify that the NFS daemons are running by executing the **`nfsping(1M)`** command on the File Server. An example run of this command is shown below.

```
# nfsping -a
nfsping: rpcbind is running
nfsping: nfsd is running
nfsping: biod is running
```

```
nfsping: mountd is running
nfsping: lockd is running
nfsping: statd is running
nfsping: bootparamd is running
nfsping: pcnfsd is running
nfsping: The system is running in client, server, bootserver,
         and pc server modes
```

If there is a console attached to the client and the client appears to boot successfully but cannot be accessed from any other system, verify that the **`inetd(1M)`** daemon is running on the client.

2.6.4. Shutting Down the Client

From the client's console, the client may be shutdown using any of the system shutdown commands, e.g. **`shutdown(1M)`** or **`init(1M)`**.

A client configured with NFS can be shutdown from the File Server using the **`rsh(1)`** command. For example, the following **`shutdown(1M)`** command would bring the system configured with the ethernet hostname '*fred*' to **`init state 0`** immediately.

`rsh` *fred* **`/sbin/shutdown -g0 -y -i0`**

By default, clients configured in Embedded mode do not require an orderly shutdown but an application may initiate it.

3
Flash Boot System Administration

3
Flash Boot System Administration

3.1. Introduction

Flash is a mostly-read, memory-mapped device that behaves like normal memory (at least for reads), and whose contents are not lost across system reboots or power cycles. Power Hawk Series 900 systems come with at least 32 MB of Boot Flash.The actual STAR/ ASTRix boot code uses a small portion of this (8 MB). The rest of the Boot Flash space is free for users to store their own data.

Additional User Flash can be optionally added to the board by the factory in 32 MB increments, up to a total of 96 MB.

NOTE

For Power Hawk Series 920 systems, please refer to the “Boot and Flash Memory” section of the *Raptor DX VMEbus Dual G4, Dual PMC & StarFabric User Guide* for more information about User Flash modules and their use.

For Power Hawk Series 940 systems, please refer to the *Manta QX VMEbus Quad, Single PMC & StarFabric User Manual*.

While Flash reads behave much like normal memory reads, Flash writes are accomplished through use of a set ASTRix flash write commands that support bulk erasure and reprogramming of Flash with any desired set of user data that will fit onto the device.

The rest of this chapter discusses using some portion of Boot or User Flash to store a client's netboot image so that the client may be thereafter booted directly from Flash, without the need to re-download the client's boot image from the file server with tftp.

Flash booting would typically be done as part of the transition from the software development and testing phase of a project to the production or deployed phase:

- While in the development phase, each client would typically be booted using the netboot tftp method as previously described in the “Netboot System Administration” chapter.
- When entering the deployed phase, the client could be converted from network tftp booting to Flash booting, where the already existing client boot sequence would need to be slightly modified from copying a netboot image from the file server, to instead loading and booting the client boot

image directly from Flash. All the other steps of the client's boot sequence are performed as before.

Flash booting should not be thought of as a totally separate function from booting a client, but as a minor, and significant adjustment to the standard client boot sequence that must always occur.

Any client boot image which is netbootable can be stored into Flash and booted directly from Flash. No special preparation or treatment of the client boot image itself is needed.

There are several advantages to storing the client boot image in Flash:

- It gives a client some independence from a File Server SBC. For embedded clients especially, it is possible to develop a boot image that makes no reference to a File Server at all, thus resulting in a client that can be placed into a standalone configuration during the deployed phase.
- Bootup times are faster when booting from Flash as no boot image download occurs. This can be especially important during system startup to ward off the network congestion that occurs when many clients simultaneously download their boot images from a common File Server.

3.2. Flash Characteristics

The Flash on Power Hawk Series 900 boards is what is called a paged flash or a banked flash. Flash is accessed through two 32 MB windows, one for User Flash and one for Boot Flash, where the Boot Flash is always accessible. A particular single bank of 32 MB User Flash may be selected for access through the 32 MB User Flash window, where this selection is accomplished through the ASTRix 'fbs' (User Flash bank select) command.

PowerMAX OS supports two methods for storing and booting a client boot image in Flash; the Raw Write method, and the Flash file system method

The Raw Write method uses the ASTRix 'fw' (Flash write) command to write the client boot image to Flash in a direct raw write manner.

The Flash filesystem method uses the ASTRix 'ffsw' (Flash filesystem write) command to write the client boot image into a Flash file system area. The format of this area is an ASTRix-supported Flash filesystem that may contain multiple files.

The advantages of using the Flash filesystem method are:

- The ASTRix Flash filesystem support is used for storing the client boot image. As such, multiple boot images may be stored into the same Flash filesystem and queried with the 'ffsls' (Flash filesystem 'ls' command),
- The user may store client boot images into User Flash (if available) as well as the Boot Flash area.

The main "disadvantage" of using the Flash file system method is that, unlike the Raw Write method, the entire client boot image is loaded into memory and uncompressed. The entire memory-resident root filesystem remains in memory after booting. This attribute of Flash filesystem booting is thus the same as network booting.

The main advantage of using the Raw Write method is that, even though the real physical Flash bank size is 32 MB, the boot image is conceptually divided into a series of 128 KB sized virtual banks or pages by the PowerMAX OS kernel. These virtual Flash pages are copied into memory only on an as needed basis, and the Flash-backed memory area is freed when it is no longer in use.

This Flash paging support provides for better utilization of client memory, as the root filesystem continues to reside in Flash after booting, thus freeing up the memory that would otherwise have been used to implement a memory-resident root filesystem.

The main disadvantage of using the Raw Write method is that when the Raw Write method is used, the boot image object must be written to a specific location within the Boot Flash. Therefore, only one boot image may be stored in Flash at any point in time.

The following sections discuss how to setup a Flash boot image using either the Flash filesystem or the Raw Write methods. The user is free to choose either method, depending upon their needs.

3.3. The Flash Filesystem Method

To convert the netboot client into a Flash filesystem booted client, take the following steps.

NOTE

This procedure assumes that the user has already setup and booted the client successfully as a netboot client, using the procedures described in the “Netboot System Administration” chapter.

1. Reset or power-cycle the client board, and wait for STAR to autoboot into ASTRix.
2. Abort the startup script with a ‘<Ctrl>c’ keyboard sequence, if an automatic netboot startup script has previously been setup to netboot the client.
3. Tftp the netboot image into the client's memory from the File Server SBC:

   ```
   * tftp -g -r <client>.bstrap $HOST <cr>
   ```

   where <client> should be replaced with the name of the client profile file that is located on the File Server SBC in the /etc/profiles directory.
4. Burn the boot image into a Flash filesystem. If you choose to use the Boot Flash for storing the boot image, then Flash filesystem number 3 should be used:

```
* ffsw <client>.bstrap 3 <cr>
```

NOTE

Multiple versions of a client bootimage may be stored into the same Flash filesystem by using a different name. For example, to store a second version of a boot image into Flash filesystem 3 while keeping the previous version:

```
* mv <client>.bstrap <client>.bstrap2 <cr>
* ffsw <client>.bstrap2 3 <cr>
```

Alternatively, if you have User Flash available and wish to store the client boot image into User Flash instead of Boot Flash, then use the ASTRix command:

```
  * ffsls <cr>
```

to view the available Flash filesystem areas, and to pick a Flash filesystem that resides in a User Flash area instead of Boot Flash filesystem area 3.

So for example, to use User Flash filesystem area 5, use the following ASTRix command to burn the boot image to User Flash:

```
* ffsw <client>.bstrap 5
```

5. Change the boot command's configuration slot number 5 so that it boots from the Flash boot image:

   ```
   * boot 5 -n ffs3:<curtis>.bstrap -s -c
   ```

   Or possibly enter something like:

   ```
   * boot 5 -n ffs3:<curtis>.bstrap2 -s -c
   ```

   if you are using a second boot image.

   Alternatively, if you are using User Flash to store the image, then enter the Flash filesystem number that you used in step #4.

   For example, the command:

   ```
   * boot 5 -n ffs5:<curtis>.bstrap -s -c
   ```

   will setup to boot the boot image from User Flash filesystem number 5.

6. If you wish to only manually Flash boot, then remove the startup script:

   ```
   * rm /nvram/startup
   ```

   and proceed to step #7. If you wish to autoboot from Flash, then create a new startup script:

```
* rm /nvram/startup
* vi /nvram/startup
```

and add the following lines to the file:

```
echo “About to Flash boot the client...”
sleep 3
boot 5
```

Use ‘:wq <cr>’ to write the file and exit vi.

The “sleep 3" line above will provide time for you to abort out of the Flash autoboot sequence with a <Ctrl>c keyboard sequence, should you need to make any configuration changes to the client.

7. To test the Flash boot configuration with autoboot configured, issue the ASTRix command:

   ```
   * reboot <cr>
   ```

   The board should reset and enter STAR, and STAR should then autoboot ASTRix; ASTRix should execute the /nvram/startup script, which should boot the client from Flash.

   If there is a problem with the Flash autoboot configuration, then you may abort the ASTRix startup script with a <Ctrl>c keyboard sequence and make any required changes.

   If the board is configured to manually boot the system without using an ASTRix startup script, then test the Flash boot configuration with the following ASTRix command:

   ```
   * boot 5 <cr>
   ```

   ASTRix should load and boot the client boot image from Flash.

3.4. The Raw Write Method

To convert the netboot client into a Flash Raw Write booted client, follow the procedure below. This procedure assumes that the user has already setup and booted the client successfully as a netboot client, using procedures described in the “Netboot System Administration” chapter.

1. Reset or power-cycle the client board, and wait for STAR to autoboot into ASTRix.

2. If an automatic netboot startup script has previously been setup to netboot the client, abort the startup script using a <Ctrl>c keyboard sequence.

3. Copy the netboot image into the client's memory from the File Server SBC:

   ```
   * tftp -g -r <client>.bstrap $HOST <cr>
   ```

where <client> should be replaced with the name of the client profile file that is located on the File Server SBC in the /etc/profiles directory.

4. Burn the boot image into Flash, using the ASTRix Flash write command:

```
* fw 2 0 <client>.bstrap
```

5. fw 0xfa000000 <client>.bstrap

   Replace <client> above with the same name used in step #3.

Note

> Flash area 2 and the flash offset value of 0 MUST be used in the flash write (fw) command above in order for this method of Flash booting to function properly.The above Flash address MUST be used in order for this method of Flash booting to function properly. The 'fw' command will replace/overwrite any filesystem that was previously setup in Flash filesystem 3 in the Boot Flash. Also note that the 'ffsls' ASTRix command will therefore display the #3 BOOT area as 'Unused'.
>
> If Flash Booting is no longer desired, the Flash filesystem 3 may be re-created when a 'ffsw' ASTRix command is issued for Flash filesystem 3. In this case, the first 'ffsw' command issued for Flash filesystem3 will recognize that the Flash filesystem is not present and this command will prompt the user to see if a new filesystem should be created.

6. Place the flashboot program from the File Server into NVRAM. The flashboot program is a small executable that will be booted by ASTRix via the 'boot 5' command. This flashboot program will jump from memory to the start of the Flash executable to begin execution of the client Flash boot image. To store the flashboot program into NVRAM:

```
* cd /nvram <cr>
* tftp -g -r flashboot $HOST <cr>
```

7. Change the boot command configuration slot number 5 so that it boots the flashboot program:

```
* boot 5 -n /nvram/flashboot -s -c
```

8. If you wish to only manually Flash boot, then remove the startup script:

```
* rm /nvram/startup
```

   and proceed to step #8.

If you wish to autoboot from Flash, then create a new startup script:

```
* rm /nvram/startup
* vi /nvram/startup
```

Add the following lines to the file using vi:

```
echo "About to Flash boot the client..."
sleep 3
boot 5
```

Use ':wq <cr>' to write the file and exit vi.

The "sleep 3" line above will provide time for you to abort out of the Flash autoboot sequence with a <Ctrl>c keyboard sequence, should you need to make any configuration changes to the client.

9. To test the Flash boot configuration with autoboot configured issue the ASTRix "*reboot*" command:

```
* reboot <cr>
```

The board should reset and enter STAR. STAR should then autoboot ASTRix, and ASTRix should execute the /nvram/startup script, which will boot the client from Flash. If there is a problem with the Flash autoboot configuration, then you may abort the ASTRix startup script with a <Ctrl>c keyboard sequence and make any required changes.

If the board is configured to manually boot the system without using an ASTRix startup script, then test the Flash boot configuration with the ASTRix "*boot*" command:

```
* boot 5 <cr>
```

ASTRix should load and boot the client boot image from Flash.

4
Debugging Tools

4
Debugging Tools

4.1. System Debugging Tools

This chapter covers the tools available for system debugging on a diskless client. Tools covered in this include the following:

- kdb
- crash

In the loosely-coupled architecture, the only attachment between the file server and the diskless client is via an ethernet network connection. There is no way to remotely access a diskless system’s memory in a loosely-coupled configuration. A client is referred to as a netboot client and is configured via a netboot client profile file in the **`/etc/profile`** directory. For more information on netboot clients, see Chapter 2 “Netboot System Administration”.

The state of a diskless system may be examined as follows:

a. Enter kdb by typing a ~k sequence on the client's console. The client's boot image must have been built with kdb support.

b. Examine the client system locally on the client system using crash.

When **`kdb`** is configured into a client’s kernel, the **`~k`** sequence will cause the system to drop into **`kdb`**. The sequences ~b, ~i, and ~h all cause an immediate reset of the board, where control is subsequently returned to the STAR/ASTRix firmware.

4.2. kdb

The **`kdb`** package is provided with the kernel base package. A client kernel may be configured with **`kdb`**, and its boot image may be configured to enter **`kdb`** early in the boot process. A console terminal must be connected to the client board to interact with **`kdb`**.

On client boards, the console debugger support is not present. However, if system level debugging on a client is desired, then it is possible to use **`kdb`**. To use **`kdb`**, the **`kdb`** and **`kdb_util`** kernel drivers must be configured into the client’s unix kernel. Except for the kdb consdebug and star (or smon) commands, kdb operates without any differences from a normal system when executing on a client. The consdebug command is not available on netboot clients, and the star (or smon) command will reset the board and return control to the STAR/ASTRix firmware.

By default, the client kernel configuration is not built with kdb support. The kdb kernel modules may be configured into the client's kernel via the '-k' option to the mknetbstrap command.

```
#>mknetbstrap -kkdb -kkdb_util
```

The client system may also be programmed to stop during bootup in kdb via the:

```
mknetbstrap -b kdb
```

option. See the **mknetbstrap(1M)** system manual page for more details on these kdb-related options.

When **kdb** is configured into a client's kernel, the **~k** sequence will cause the system to drop into **kdb**.

4.3. crash

The **crash(1M)** utility command may be used to examine the system memory image of a running system by internally formatting and then displaying various control structures, tables, and other information.

The **crash(1M)** utility may be run directly on the client system to examine the current state of the running system. The user may be logged into the system through the attached console terminal or through a networked terminal connection, such as telnet or rlogin.

A
Adding a Local Disk

By default, clients are configured as diskless systems. It may be desirable to connect a local disk drive which is used to store application-specific data. The following example demonstrates how to configure a disk assuming that the disk has been formatted, file systems have been created on the appropriate partitions and the disk has been connected to the client. Refer to the *System Administration* manual (Volume 2) for guidelines on how to accomplish these pre-requisite steps.

The kernel configuration may be modified using the **`config(1M)`** tool and specifying the client's virtual root directory. For example, if the client's virtual root path is **`/vroots/elroy`**:

```
config -r /vroots/elroy
```

1. If necessary, add an entry to the adapters table -

   Adapter information must be added to the adapters table for VME adapters (i.e., via). PCI adapters (i.e., ncr) are auto-configurable and should not be added to the adapters table. If this is a VME adapter, add an entry for it in the adapters table using the Adapters/Add menu option of **`config(1m)`**.

2. Configure kernel modules -

   Use the Modules function of the **`config(1M)`** tool to enable the following modules:

gd	(generic disk driver)
scsi	(device independent SCSI interface support)
ncr	(internal SCSI adapter interface driver)
ufs	(unix file system)
sfs	(unix secure file system)

   If a Resilient File System (XFS) is required for a client, instead of enabling ufs and sfs, enable:

   xfs (resilient file system)

   xfsth (resilient file system threaded)

   Note that the **`kernel.modlist.add`** table in the client's **`custom.conf`** directory (**`/etc/clients/<client_dir>/custom.conf`**) may instead be used to enable kernel modules.

Note:

The procedural steps below differ depending whether the client was configured with NFS support or as embedded.

NFS Clients (steps #3 - #6):

3. Configure Disk Device Files

   Check that an appropriate device node entry (**`Node(4)`**) exists and is uncommented for the disk being added. The following is such an entry from the Node file **`/vroots/elroy/etc/conf/node.d/gd`**:

   ```
   gd  dsk/0  D  ncr  0  0  0  0  3  0640  2
   ```

4. Add mount point directory entries to the memfs root file system via the **`memfs.files.add`** custom file. For example, to add the directories arbitrarily named **`/dsk0s0`** and **`/dsk0s1`**:

   ```
   # cd /etc/clients/<client_dir>/custom.con
   # ./mkprivate memfs.files.add
   # vi memfs.files.add
   ```

   Example entries:

   ```
   d   /dsk0s0     0777
   d   /dsk0s1     0777
   ```

5. Enable Automatic Mounting of a Disk Partition by adding entries to the client's **`vfstab`** file. Note that the mount point directory name must match the directory name specified in the **`memfs.files.add`** file in step 4 above.

   ```
   # cd /etc/clients/<client_dir>/custom.conf
   # ./mkprivate vfstab
   # vi vfstab
   ```

   Example entries:

```
/dev/dsk/0s0   /dev/rdsk/0s0   /dsk0s0  ufs     1    yes   -
/dev/dsk/0s1   /dev/rdsk/0s1   /dsk0s1  ufs     1    yes   -
```

6. Generate a new netboot image:

   ```
   mknetbstrap -r all <client>
   ```

Embedded clients (steps #3 - #5):

3. The disk management tools must be added to the memfs file system. The list of tools is documented in the file **`/usr/etc/diskless.d/sys.conf /memfs.d/add_disk.sh`**. In executing the following commands, we grep the

list of commands from this file and append them to the **`memfs.files.add`** tables.

```
# cd /etc/clients/<client_dir>/custom.conf
# ./mkprivate memfs.files.add
# /sbin/grep "^#f" /usr/etc/diskless.d/sys.conf \
/memfs.d/add_disk.sh | cut -c2- >> ./memfs.files.add
```

Verify that the following entries were appended to **`memfs.files.add`**.

```
f    /sbin/expr          0755
f    /usr/bin/devcfg     0755
f    /usr/bin/cut        0755
f    /sbin/mknod         0755
f    /usr/bin/mkdir      0755
f    /sbin/fsck          0755
f    /etc/fs/ufs/fsck    0755
f    /etc/fs/xfs/mount   0755
f    /etc/fs/ufs/mount   0755
f    /sbin/df            0755
```

4. Embedded client systems do not have access to the kernel configuration directory, which is needed to generate the device node entries. However, the device node must be created on the client system because it's minor number carries information that is unique to the running system. For this reason, special steps must be taken during client boot-up to create the device nodes.

   The sample script below will do the necessary steps to add a local disk on an embedded client. This script may be found on the File Server system under the path **`/usr/etc/diskless.d/sys.conf/memfs.d/add_disk.sh`**. Note that you must set the variables "FSTYPE" and "PARTITIONS" to the appropriate values.

```
# cd /etc/clients/<client_dir>/custom.conf
# ./mkprivate S25client
# cat /usr/etc/diskless.d/sys.conf/memfs.d  \
/add_disk.sh >> ./S25client
# vi ./S25 client
```

   Verify that the script (**illustrated on the next page after step #5**) was appended to **`S25client`** and set the variables FSTYPE and PARTITIONS.

5. Generate a new netboot image:

   mknetbstrap -r all <client>

----------- Beginning of Script --------------------

```
#
#  Start-up script to mount a local disk on a client configured
#  as embedded.
#
#  The variables FSTYPE and PARTITIONS should be set to the
#  appropriate values.
#
#  To be able to run this script, the following binaries must be
#  added to the memfs file system via the memfs.files.add custom
#  table. Example entries follow.
#
#f   /sbin/expr        0755
#f   /usr/bin/devcfg   0755
#f   /usr/bin/cut      0755
#f   /sbin/mknod       0755
#f   /usr/bin/mkdir  0755
#f   /sbin/fsck      0755
#f   /etc/fs/ufs/fsck0755
#f   /etc/fs/xfs/mount0755
#f   /etc/fs/ufs/mount 0755
#f   /sbin/df          0755
#
# Set FSTYPE and PARTITIONS
#
FSTYPE=ufs                    # file system type (ufs, xfs)
PARTITIONS=”0 1 2 3 4 5 6”    # disk partitions to be mounted

#
# Initialize
#
> /etc/mnttab
> /etc/vfstab
disk=0

#
#  Create the device directories
#
/usr/bin/mkdir -p /dev/dsk
/usr/bin/mkdir -p /dev/rdsk

#
#  In this loop, the device nodes are created based on
#  major/minor device information gathered from the call
#  to devcfg.  The fsck(1m) utility is executed for each
#  file system, a mount point directory is created, and
#  the file system is mounted and then verified using df.
#
/usr/bin/devcfg -m disk | /usr/bin/cut -f3 | while read majmin
do
     maj=`echo $majmin | /usr/bin/cut -f1 -d” “`
     min=`echo $majmin | /usr/bin/cut -f2 -d” “`
     #
     # Creates, fsck and mounts the partition.
     #
     for i in $PARTITIONS
     do

          #
          # create the device nodes
          #
          minor=`/sbin/expr $min + $i`
          echo “===>Creating nodes /dev/dsk/${disk}s${i} \c”
          echo “and /dev/rdsk/${disk}s${i}”
          /sbin/mknod /dev/dsk/${disk}s${i} b $maj $minor
          /sbin/mknod /dev/rdsk/${disk}s${i} c $maj $minor
          #  fsck (ufs only) and mount each partitions.
          #
          if [ “$FSTYPE” != “xfs” ]
          then
          echo “===>Fsck’ing partition /dev/rdsk/${disk}s${i}”
               /etc/fs/$FSTYPE/fsck -y /dev/rdsk/${disk}s${i} \
                    > /dev/null
          fi

          #
          # create a mount point directory
          #
          /usr/bin/mkdir /${disk}s${i}

          #
          #  mount the partition
          #
          echo  “===>Mounting  /dev/dsk/${disk}s${i} /${disk}s${i}\n”
          /etc/fs/$FSTYPE/mount /dev/dsk/${disk}s${i} /${disk}s${i}

     done
     break
     disk=`/sbin/expr $disk + 1`
done
#
# verify the partitions are mounted
#
echo “===>Verifying mounted file systems”
/sbin/df -kl

------------- End of Script --------------------
```

B
Make Client System Run in NFS File Server Mode

To become an NFS server, a client system must have an attached local disk. In becoming an NFS server, the client can share the data in the local disk partitions with other systems that have network access to the client. See Appendix A “Adding a Local Disk” for instructions on how to add a local disk to a diskless client.

We will refer to the client with the local disk as the disk_server, and the system(s) that want to access this client disk as disk_clients.

Note

It is important that the hostnames that are used for the disk_server and the disk_clients correspond to network interfaces that are accessible to both the disk_clients and disk_server. The disk_server hostname is the one specified by the disk_clients when they are mounting the disk_client's NFS partition(s), and the various disk_clients hostnames are used on the disk_server for the 'share' command line in the **`/etc/dfstab`** file.

1. Enable the **`nfssrv`** kernel module for the client that has the local disk attached (disk_server). The following commands should be executed on the File Server system of the disk_server client.

   ```
   # cd /etc/clients/<disk_server_dir>/custom.conf
   # ./mkprivate kernel.modlist.add
   # vi kernel.modlist.add
   ```

   Add the following:

   ```
   nfssrv
   ```

2. For each partition to be shared, add an entry similar to the example entry shown below. Note that “`disk_client_1:disk_client_n`” refers to a list of nodes that want to share this partition. See the **`dfstab(4)`** manpage for more information. **`disk_server_vrootpath`** is the path to the virtual root directory of the node with the local disk attached.

   ```
   # vi <disk_server_vrootpath>/etc/dfs/dfstab
   ```

   Example entry:

```
share -F nfs -o rw,root=disk_client_1:disk_client_n -d "/disk0s0" /disk0s0 disk0s0
```

3. Generate a new netboot image for the disk_server client by executing the following command:

   mknetbstrap -r *all* **<disk_server>**

On each disk_client node that wants to share the disk partitions, we need to generate a mount point directory for each partition to be mounted across NFS. These partition can also be automatically mounted and unmounted during the system's boot/shutdown if desired. If the disk_client node is another diskless client, the mount points may be added to the memfs root file system via the **memfs.files.add** table and the automatic mounting may be achieved via the node's **vfstab** file or the **rc** scripts shown below.

1. Add directories to be used for mount points to the memfs filesystem.

```
# cd /etc/clients/<disk_client_dir>/custom.conf
# ./mkprivate  memfs.files.add
# vi memfs.files.add
```

   Example entry:

```
d   /rem_0s0     0755
```

2. Add an entry to the client's **startup** script to automatically mount the partition.

```
# cd /etc/clients/<disk_client_dir>/custom.conf
# ./mkprivate S25client
# vi S25client
```

   Example entry:

```
#
# if the disk_server is up, mount remote file system
# mount point /rem_0s0
#
if ping <disk_server>  > /dev/null
then
/sbin/mount -F nfs  <disk_server>:/disk0s0  /rem_0s0
fi
```

3. Add an entry to the client's **shutdown** script to automatically unmount the partition

```
# cd /etc/clients/<disk_client_dir>/custom.conf
# ./mkprivate  K00client
# vi K00client
```

   Example entry:

```
umount /rem_0s0
```

Glossary

10base-T

See twisted-pair Ethernet (10base-T).

100base-T

See twisted-pair Ethernet (100base-T).

ARP

Address Resolution Protocol as defined in RFC 826. ARP software maintains a table of translation between IP addresses and Ethernet addresses.

AUI

Attachment Unit Interface (available as special order only)

asynchronous

An event occurring in an unpredictable fashion. A signal is an example of an asynchronous event. A signal can occur when something in the system fails, but it is not known when the failure will occur.

asynchronous I/O operation

An I/O operation that does not of itself cause the caller to be blocked from further use of the CPU. This implies that the caller and the I/O operation may be running concurrently.

asynchronous I/O completion

An asynchronous read or write operation is completed when a corresponding synchronous read or write would have completed and any associated status fields have been updated.

block data transfer

The method of transferring data in units (blocks) between a block device such as a magnetic tape drive or disk drive and a user program.

block device

A device, such as a magnetic tape drive or disk drive, that conveys data in blocks through the buffer management code. Compare character device.

block driver

A device driver, such as for a magnetic tape device or disk drive, that conveys data in blocks through the buffer management code (for example, the `buf` structure). One driver is written for each major number employed by block devices.

block I/O

A data transfer method used by drivers for block access devices. Block I/O uses the system buffer cache as an intermediate data storage area between user memory and the device.

block

The basic unit of data for I/O access. A block is measured in bytes. The size of a block differs between computers, file system sizes, or devices.

boot

The process of starting the operating system. The boot process consists of self-configuration and system initialization.

boot device

The device that stores the self-configuration and system initialization code and necessary file systems to start the operating system.

boot image file

A file that can be downloaded to and executed on a client SBC. Usually contains an operating system and root filesystem contents, plus all bootstrap code necessary to start it.

bootstrap

The process of bringing up the operating system by its own action. The first few instructions load the rest of the operating system into the computer.

buffer

A staging area for input-output (I/O) processes where arbitrary-length transactions are collected into convenient units for system operations. A buffer consists of two parts: a memory array that contains data from the disk and a buffer header that identifies the buffer.

cache

A section of computer memory where the most recently used buffers, i-nodes, pages, and so on are stored for quick access.

character device

A device, such as a terminal or printer, that conveys data character by character.

character driver

The driver that conveys data character by character between the device and the user program. Character drivers are usually written for use with terminals, printers, and network devices, although block devices, such as tapes and disks, also support character access.

character I/O

The process of reading and writing to/from a terminal.

client

A SBC board, usually without a disk, running a stripped down version of PowerMAX OS and dedicated to running a single set of applications. Called a client, since the client may maintain an Ethernet connection to its File Server and use that File Server as a type of remote disk device; utilizing it to fetch applications, data, and to swap unused/needed pages to/from memory

controller

The circuit board that connects a device, such as a terminal or disk drive, to a computer. A controller converts software commands from a driver into hardware commands that the device understands. For example, on a disk drive, the controller accepts a request to read a file and converts the request into hardware commands to have the reading apparatus move to the precise location and send the information until a delimiter is reached.

cyclic redundandancy check (CRC)

A way to check the transfer of information over a channel. When the message is received, the computer calculates the remainder and checks it against the transmitted remainder.

datagram

Transmission unit at the IP level.

data structure

The memory storage area that holds data types, such as integers and strings, or an array of integers. The data structures associated with drivers are used as buffers for holding data being moved between user data space and the device, as flags for indicating error device status, as pointers to link buffers together, and so on.

data terminal ready (DTR)

The signal that a terminal device sends to a host computer to indicate that a terminal is ready to receive data.

data transfer

The phase in connection and connection-less modes that supports the transfer of data between two DLS users.

device number

The value used by the operating system to name a device. The device number contains the major number and the minor number.

diagnostic

A software routine for testing, identifying, and isolating a hardware error. A message is generated to notify the tester of the results.

DLM

Dynamically Loadable Modules.

DRAM

Dynamic Random Access Memory.

driver entry points

Driver routines that provide an interface between the kernel and the device driver.

driver

The set of routines and data structures installed in the kernel that provide an interface between the kernel and a device.

embedded

The host system provides a boot image for the client system. The boot image contains a UNIX kernel and a file system image which is configured with one or more embedded applications. The embedded applications execute at the end of the boot sequence.

error correction code (ECC)

A generic term applied to coding schemes that allow for the correction of errors in one or more bits of a word of data.

FDDI

Fiber Distributed Data Interface.

flash autobooting

The process of booting a target from an image in its Flash memory rather than from an image downloaded from a host. Flash booting makes it possible to design targets that can be separated from their hosts when moved from a development to a production environment.

flash booting

See definition for **flash autobooting**.

flash burning

The process of writing a boot or other image into a Flash memory device. On client Power Hawk Series 900 boards, this is usually accomplished with ASTRix 'ffsw' or 'fw' commands.

flash memory

A memory device capable of being occasionally rewritten in its entirety, usually by a special programming sequence. Like ROM, Flash memories do not lose their contents upon power down.

FTP (ftp)

The File Transfer Protocol is used for interactive file transfer.

File Server

The File Server has special significance in that it is the only system with a physically attached disk(s) that contain file systems and directories essential to running the PowerMAX OS. The File Server boots from a locally attached SCSI disk and provides disk storage space for configuration and system files for all clients. All clients depend on the File Server since all the boot images and the system files are stored on the File Server's disk.

function

A kernel utility used in a driver. The term function is used interchangeably with the term kernel function. The use of functions in a driver is analogous to the use of system calls and library routines in a user-level program.

host

A SBC running a full fledged PowerMAX OS system containing disks, networking, and the netboot development environment. Called a File Server since it serves clients with boot images, filesystems, or whatever else they need when they are running.

host board

The single board computer of the File Server.

host name

A name that is assigned to any device that has an IP address.

host system

A term used for the File Server. It refers to the prerequisite Power Hawk system.

interprocess communication (IPC)

A set of software-supported facilities that enable independent processes, running at the same time, to share information through messages, semaphores, or shared memory.

interrupt level

Driver interrupt routines that are started when an interrupt is received from a hardware device. The system accesses the interrupt vector table, determines the major number of the device, and passes control to the appropriate interrupt routine.

interrupt vector

Interrupts from a device are sent to the device's interrupt vector, activating the interrupt entry point for the device.

ICMP

Internet Control Message Protocol, an integral part of IP as defined in RFC 792. This protocol is part of the Internet Layer and uses the IP datagram delivery facility to send its messages.

IP

The Internet Protocol, RFC 791, is the heart of the TCP/IP. IP provides the basic packet delivery service on which TCP/IP networks are built.

ISO

International Organization for Standardization

kernel buffer cache

A set of buffers used to minimize the number of times a block-type device must be accessed.

kdb

Kernel debugger.

loadable module

A kernel module (such as a device driver) that can be added to a running system without rebooting the system or rebuilding the kernel.

MTU

Maximum Transmission Units - the largest packet that a network can transfer.

memory file system image

A cpio archive containing the files which will exist in the root file system of a client system. This file system is memory resident. It is implemented via the existing *memfs* file system kernel module. The kernel unpacks the cpio archive at boot time and populates the root memory file system with the files supplied in the archive.

memory management

The memory management scheme of the UNIX operating system imposes certain restrictions on drivers that transfer data between devices.

modem

A contraction of modulator-demodulator. A modulator converts digital signals from the computer into tones that can be transmitted across phone lines. A demodulator converts the tones received from the phone lines into digital signals so that the computer can process the data.

netboot

The process of a client SBC downloading into its own memory and then executing a boot image file that is retrieved from a File Server SBC by using the TFTP network protocol. On client SBC boards, networking is configured with the STAR 'config' command, and an ASTRix /nvram/startup script may be created and automatically executed after a reset, in order to download and execute a client boot image via TFTP with ASTRix 'tftp' and 'boot' commands.

netload

The process of a target loading a boot image as discussed under netboot, but without subsequently executing it. On Power Hawk Series 900 client boards, netloading is invoked with the ASTRix 'tftp' command.

network boot

See definition for **netboot**.

network load

See definition for **netload**.

netstat

The **`netstat`** command displays the contents of various network-related data structures in various formats, depending on the options selected.

NFS

Network File System. This protocol allows files to be shared by various hosts on the network.

NFS client

In a NFS client configuration, the host system provides UNIX file systems for the client system. A client system operates as a diskless NFS client of a host system.

NIS

Network Information Service (formerly called yellow pages or yp). NIS is an administrative system. It provides central control and automatic dissemination of important administrative files.

NVRAM

Non-**V**olatile **R**andom **A**ccess **M**emory. This type of memory retains its state even after power is removed.

panic

The state where an unrecoverable error has occurred. Usually, when a panic occurs, a message is displayed on the console to indicate the cause of the problem.

PDU

Protocol Data Unit

PowerPC G4

The PowerPC G4 (7450) microprocessor. Part of the PowerPC family of microprocessors; an architecture based on Motorola/IBM's 32-bit RISC design CPU core.

PPP

Point-to-Point protocol is a method for transmitting datagrams over point-to-point serial links

prefix

A character name that uniquely identifies a driver's routines to the kernel. The prefix name starts each routine in a driver. For example, a RAM disk might be given the **`ramd`** prefix. If it is a block driver, the routines are **`ramdopen`**, **`ramdclose`**, **`ramdsize`**, **`ramdstrategy`**, and **`ramdprint.`**

protocol

Rules as they pertain to data communications.

RFS

Remote File Sharing.

random I/O

I/O operations to the same file that specify absolute file offsets.

raw I/O

Movement of data directly between user address spaces and the device. Raw I/O is used primarily for administrative functions where the speed of a specific operation is more important than overall system performance.

raw mode

The method of transmitting data from a terminal to a user without processing. This mode is defined in the line discipline modules.

rcp

Remote copy allows files to be copied from or to remote systems. rcp is often compared to ftp.

read queue

The half of a STREAMS module or driver that passes messages upstream.

rlogin

Remote login provides interactive access to remote hosts. Its function is similar to telnet.

routines

A set of instructions that perform a specific task for a program. Driver code consists of entry-point routines and subordinate routines. Subordinate routines are called by driver entry-point routines. The entry-point routines are accessed through system tables.

rsh

Remote shell passes a command to a remote host for execution.

SBC

Single Board Computer

SCSI driver interface (SDI)

A collection of machine-independent input/output controls, functions, and data structures, that provide a standard interface for writing Small Computer System Interface (SCSI) drivers.

sequential I/O

I/O operations to the same file descriptor that specify that the I/O should begin at the "current" file offset.

SLIP

Serial Line IP. The SLIP protocol defines a simple mechanism for "framing" datagrams for transmission across serial line.

server

See definition for **File Server** and **host**.

STAR/ASTRix

A pair of board-resident Flash monitor utilities that provide a basic I/O system (BIOS), a boot Flash, and system diagnostics for Power Hawk Series 900 single board computers.

ASTRix startup script

As part of the boot process, **`ASTRix`** can automatically perform **`ASTRix`** commands and/or user defined functions written in a startup script that is stored in NVRAM (nonvolatile RAM). A special startup script is used for netbooting client SBCs in loosely-coupled configurations.

SMTP

The Simple Mail Transfer Protocol, delivers electronic mail.

small computer system interface (SCSI)

The American National Standards Institute (ANSI) approved interface for supporting specific peripheral devices.

SNMP

Simple Network Management Protocol

Source Code Control System (SCCS)

A utility for tracking, maintaining, and controlling access to source code files.

special device file

The file that identifies the device's access type (block or character), the external major and minor numbers of the device, the device name used by user-level programs, and security control (owner, group, and access permissions) for the device.

synchronous data link interface (SDLI)

A UN-type circuit board that works subordinately to the input/output accelerator (IOA). The SDLI provides up to eight ports for full-duplex synchronous data communication.

system

A single board computer running its own copy of the operating system, including all resources directly controlled by the operating system (for example, I/O boards, SCSI devices).

system disk

The PowerMAX OS requires a number of system directories to be available on a system disk in order for the operation system to function properly.These directories include: /etc, /sbin, /dev, /usr and /var.

system initialization

The routines from the driver code and the information from the configuration files that initialize the system (including device drivers).

System Run Level

A netboot system is not fully functional until the files residing on the File Server are accessible. **`init(1M)`** ‘**init state 3**’ is the initdefault and the only run level supported for netboot systems. In **init state 3**, remote file sharing processes and daemons are started. Setting initdefault to any other state or changing the run level after the system is up and running, is not supported.

swap space

Swap reservation space, referred to as ‘virtual swap’ space, is made up of the number of real memory pages that may be used for user space translations, plus the amount of secondary storage (disk) swap space available.

target

See definition for **client**.

TELNET

The Network Terminal Protocol, provides remote login over the network.

TCP

Transmission Control Protocol, provides reliable data delivery service with end-to-end error detection and correction.

Trivial File Transfer Protocol(TFTP)

Internet standard protocol for file transfer with minimal capability and minimal overhead. TFTP depends on the connection-less datagram delivery service (UDP).

twisted-pair Ethernet (10base-T)

An Ethernet implementation in which the physical medium is an unshielded pair of entwined wires capable of carrying data at 10 Mbps for a maximum distance of 185 meters.

twisted-pair Ethernet (100base-T)

An Ethernet implementation in which the physical medium is an unshielded pair of entwined wires capable of carrying data at 100 Mbps for a maximum distance of 185 meters.

UDP

User Datagram Protocol, provides low-overhead, connection-less datagram delivery service.

unbuffered I/O

I/O that bypasses the file system cache for the purpose of increasing I/O performance for some applications.

upstream

The direction of STREAMS messages flowing through a read queue from the driver to the user process.

user space

The part of the operating system where programs that do not have direct access to the kernel structures and services execute. The UNIX operating system is divided into two major areas: the user programs and the kernel. Drivers execute in the kernel, and the user programs that interact with drivers execute in the user program area. This space is also referred to as user data area.

yellow pages

See definition for **NIS** (Network Information Services).

Index

Spine for 1" Binder

Product Name: 0.5" from top of spine, Helvetica, 36 pt, Bold

Volume Number (if any): Helvetica, 24 pt, Bold

Volume Name (if any): Helvetica, 18 pt, Bold

Manual Title(s): Helvetica, 10 pt, Bold, centered vertically within space above bar, double space between each title

Bar: 1" x 1/8" beginning 1/4" in from either side

Part Number: Helvetica, 6 pt, centered, 1/8" up

PowerMAX OS

Admin

Power Hawk Series 900 Diskless Systems Administrator's Guide

0891090