Power Hawk Series 700
Diskless Systems Administrator's Guide

0891086-040
November 2004

Copyright 2004 by Concurrent Computer Corporation. All rights reserved. This publication or any part thereof is intended for use with Concurrent Computer Corporation products by Concurrent Computer Corporation personnel, customers, and end–users. It may not be reproduced in any form without the written permission of the publisher.

The information contained in this document is believed to be correct at the time of publication. It is subject to change without notice. Concurrent Computer Corporation makes no warranties, expressed or implied, concerning the information contained in this document.

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Computer Corporation, 2881 Gateway Drive Pompano Beach, FL 33069. Mark the envelope **"Attention: Publications Department**."
This publication may not be reproduced for any other reason in any form without written permission of the publisher.

UNIX is a registered trademark of the Open Group.
Ethernet is a trademark of Xerox Corporation.
PowerMAX OS is a registered trademark of Concurrent Computer Corporation.
Power Hawk and PowerStack II/III are trademarks of Concurrent Computer Corporation.

Other products mentioned in this document are trademarks, registered trademarks, or trade names of the manufactures or marketers of the product with which the marks or names are associated.

Printed in U. S. A.

Revision History:		Level:	Effective With:
Original issue	June 2001	000	PowerMAX OS Release 5.1
Previous issue	August 2001	020	General Update
Previous issue	September 2001	030	Release 5.1SR3 Update
Current issue	November 2004	040	Release 6.2 Update

Preface

Scope of Manual

Intended for system administrators responsible for configuring and administering diskless system configurations. A companion manual, the *Power Hawk Series 700 Closely-Coupled Programming Guide*, is intended for programmers writing applications which are distributed across multiple single board computers (SBCs).

Structure of Manual

This manual consists of a title page, this preface, a master table of contents, nine chapters, local tables of contents for the chapters, three 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, *SBC Hardware Considerations*, provides equipment specifications, hardware preparation, installation instruction and general operating data.
- Chapter 3, *Netboot System Administration*, provides an overview of the steps that must be followed in configuring a loosely-coupled system (LCS) configuration.
- Chapter 4, *VME Boot System Administration*, provides an overview of the steps that must be followed in configuring a closely-coupled system (CCS) configuration.
- Chapter 5, *Flash Boot System Administration*, This chapter is a guide to configuring a diskless single board computer (SBC) to boot PowerMAX OS from flash memory.
- Chapter 6, *Modifying VME Space Allocation*. describes how a system administrator can modify the default VME space configuration on Closely-Coupled systems (CCS).
- Chapter 7, *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, *Backplane P0 Bridge Board Cluster Configuration*, describes the various cluster configurations that are supported when a Backplane P0 (BPP0) Bridge Board is present. This Appendix also describes the additional limitations that are associated with some of the BPP0 configurations.

- Appendix B, *Adding a Local Disk*, provides instructions on how to add a local disk to a client.
- Appendix C, *Make Client System Run in NFS File Server Mode*, 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 700 Closely-Coupled Programming Guide	0891087
Power Hawk Series 700 PowerMAX OS Version x.x Release Notes	0891084-*reln* (*reln* = release number)

Vendor Manuals

Title	Synergy Document Number	VGM5 Single Processor SBC	VGM5 Dual Processor SBC	VSS4 Quad Processor SBC
VGM5 VMEbus Dual G3/G4 PowerPC Single Board Computer User Guide	98-0317/UG-VGM5-01	X	X	-
VSS4 Quad 750 PowerPC VMEbus Single Board Computer for DSP User Guide	99-0062/UG-VSS4-01	-	-	X
SMon PowerPC Series SBCs Developers Application & Debugger User Guide	99-0041/UG-PPSM-01	X	X	X

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

Preface iii

Chapter 1 Introduction

Overview 1-1
Diskless Topography 1-1
Diskless Boot Basics 1-3
Configuration Toolsets 1-5
Definitions 1-6
Hardware Overview 1-9
Series 700 Hardware Features 1-9
Diskless Implementation 1-10
Virtual Root 1-10
Boot Image Creation and Characteristics 1-10
MEMFS Root Filesystem 1-11
Booting 1-12
VME Boot 1-13
Net Boot 1-13
Flash Boot 1-14
P0Bus Networking 1-15
Remote File Sharing 1-17
Shared Memory 1-20
Swap Space 1-20
Configuring Diskless Systems 1-22
Closely-Coupled System Hardware Prerequisites 1-22
Loosely-Coupled System Hardware Prerequisites 1-23
Disk Space Requirements 1-23
Software Prerequisites 1-24
Licensing Information 1-24

Chapter 2 SBC Hardware Considerations

Introduction 2-1
Unpacking Instructions 2-2
Board Jumpers 2-2
VGM5 Reset/SMI Toggle Switch 2-3
VSS4 Reset/SMI Toggle Switch 2-4

Chapter 3 Netboot System Administration

Configuration Overview 3-1
Installing a Loosely-Coupled System 3-1
Installing Additional Boards 3-3
SBC Client Board Configuration 3-3
Client Configuration 3-8
The Client Profile File 3-8

Required Parameters 3-9
Required NFS-Related Parameters 3-9
Hosts Tables 3-11
Configuring Clients Using netbootconfig 3-11
Creating and Removing a Client Configuration 3-11
Subsystem Support 3-13
Customizing the Basic Client Configuration 3-13
Modifying the Kernel Configuration 3-14
kernel.modlist.add 3-14
mknetbstrap 3-15
config utility 3-15
idtuneobj 3-15
Custom Configuration Files 3-16
S25client and K00client rc Scripts 3-18
memfs.inittab and inittab Tables 3-19
vfstab Table 3-20
kernel.modlist.add Table 3-20
memfs.files.add Table 3-21
vroot.files.add Table 3-22
Modifying the Client Profile Parameters 3-24
Launching Applications 3-25
Launching an Application for Embedded Clients 3-25
Launching an Application for NFS Clients 3-25
Booting and Shutdown 3-26
The Boot Image 3-27
Creating the Boot Image 3-28
Examples on Creating the Boot Image 3-28
Net Booting 3-28
Netboot Using SMon 3-29
Verifying Boot Status 3-30
Shutting Down the Client 3-30

Chapter 4 VME Boot System Administration

Overview 4-1
Cluster Configuration Overview 4-1
Installing the Cluster 4-2
How To Boot the Cluster 4-4
Installing Additional Boards in a Cluster 4-5
SBC Cluster Configuration 4-6
Board Jumpers 4-6
Installing the P0Bus Overlay 4-7
File Server Board Configuration 4-7
Client Board Configuration 4-11
Cluster Configuration 4-18
The Profile Files 4-18
The cluster.profile File 4-19
The Client Profile File 4-26
Networking Hostname Naming Conventions 4-31
Node Configuration 4-33
Creating and Removing a Client 4-34
Subsystem Support 4-35
Slave Shared Memory Support 4-36

System Tunables Modified . . . 4-39
Customizing the Basic Configuration . . . 4-40
Modifying the Kernel Configuration . . . 4-40
kernel.modlist.add . . . 4-41
mkvmebstrap . . . 4-42
config Utility . . . 4-42
idtuneobj . . . 4-42
Custom Configuration Files . . . 4-43
S25client and K00client rc Scripts . . . 4-45
Memfs.inittab and Inittab Tables . . . 4-46
vfstab Table . . . 4-47
kernel.modlist.add Table . . . 4-47
memfs.files.add Table . . . 4-48
vroot.files.add Table . . . 4-49
Modifying Profile Parameters . . . 4-51
Cluster.profile File . . . 4-51
Modifying Client Profile Settings . . . 4-54
Launching Applications . . . 4-55
Launching an Application (Embedded Client) . . . 4-55
Launching an Application (NFS Client) . . . 4-55
Booting and Shutdown . . . 4-56
The Boot Image . . . 4-57
Booting Options . . . 4-58
Creating the Boot Image . . . 4-60
VME Booting . . . 4-61
Net Booting . . . 4-62
Flash Booting . . . 4-62
Verifying Boot Status . . . 4-62
Shutting Down the Client . . . 4-63

Chapter 5 Flash Boot System Administration

Introduction . . . 5-1
User Flash Hardware Characteristics . . . 5-2
Booting a Netbootable Client from Flash . . . 5-2
Burning a Netboot Client’s User Flash . . . 5-3
Burning and Booting from Flash for VMEBus Bootable Clients . . . 5-4

Chapter 6 Modifying VME Space Allocation

Overview . . . 6-1
Default VME Configuration . . . 6-1
Reasons to Modify Defaults . . . 6-2
Limitations . . . 6-3
Changing The Default VME Configuration . . . 6-3
VME A32 Window . . . 6-3
Closely-Coupled VME A32 Window Considerations . . . 6-4
Example Configuration . . . 6-4

Chapter 7 Debugging Tools

System Debugging Tools . . . 7-1
kdb . . . 7-2

crash . . . 7-2
savecore . . . 7-3
sbcmon . . . 7-3

Appendix A Backplane P0 Bridge Board Cluster Configuration . . . A-1

Appendix B Adding a Local Disk . . . B-1

Appendix C Make Client System Run in NFS File Server Mode . . . C-1

Glossary . . . Glossary-1

Index . . . Index-1

List of Illustrations

Figure 1-1. Loosely-Coupled System Configuration . . . 1-2
Figure 1-2. Closely-Coupled Cluster of Single Board Computers . . . 1-3
Figure 1-3. Power Hawk Networking Structure . . . 1-16
Figure 2-1. VMG5 Motherboard RESET and SMI Toggle Switch . . . 2-3
Figure 2-2. VSS4 Motherboard Reset and SMI Toggle Switch . . . 2-4

List of Tables

Table 3-1. Boot Image Dependencies . . . 3-27
Table 4-1. Boot Image Dependencies . . . 4-58
Table 6-1. Default Processor/PCI/VME Configuration . . . 6-2

1 Introduction

1.1. Overview 1-1
1.1.1. Diskless Topography 1-1
1.1.2. Diskless Boot Basics 1-3
1.1.3. Configuration Toolsets 1-5
1.2. Definitions 1-6
1.3. Hardware Overview 1-9
1.3.1. Series 700 Hardware Features 1-9
1.4. Diskless Implementation 1-10
1.4.1. Virtual Root 1-10
1.4.2. Boot Image Creation and Characteristics 1-10
1.4.3. MEMFS Root Filesystem 1-11
1.4.4. Booting 1-12
1.4.4.1 VME Boot 1-13
1.4.4.2 Net Boot 1-13
1.4.4.3 Flash Boot 1-14
1.4.5. P0Bus Networking 1-15
1.4.6. Remote File Sharing 1-17
1.4.7. Shared Memory 1-20
1.4.8. Swap Space 1-20
1.5. Configuring Diskless Systems 1-22
1.5.1. Closely-Coupled System Hardware Prerequisites 1-22
1.5.2. Loosely-Coupled System Hardware Prerequisites 1-23
1.5.3. Disk Space Requirements 1-23
1.5.4. Software Prerequisites 1-24
1.6. Licensing Information 1-24

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. The *Power Hawk Series 700 Closely-Coupled Programming Guide* is a companion to this manual and contains information on the programming interfaces for inter-process communication between processes that are resident on separate single board computers (SBCs) in diskless configurations where all SBCs share a single VME backplane.

1.1.1. Diskless Topography

There are two basic topographies for configuring a set of single board computers for diskless operation. The topography defines the way that the File Server and diskless client SBCs are connected. The fileserver is an SBC which has attached a system disk that stores the boot images that define the software that is downloaded and runs on a diskless system.

The two basic diskless topologies, Loosely-Coupled Systems (LCS) and Closely-Coupled Systems (CCS), are described below:

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

Closely-Coupled - This configuration (see Figure 1-2) is supported when the fileserver and the diskless SBC share the same VMEbus and PCI-to-PCI (P0) bus (hereafter referred to as a cluster). Often multiple diskless clients will be loaded into the same cluster. This configuration makes use of the P0Bus to emulate the system bus of a symmetric multiprocessing system. Many forms of inter-process communication between processes that are running on separate single board computers are provided. See the *Power Hawk Series 700 Closely-Coupled Programming Guide* for detailed information on these interfaces.

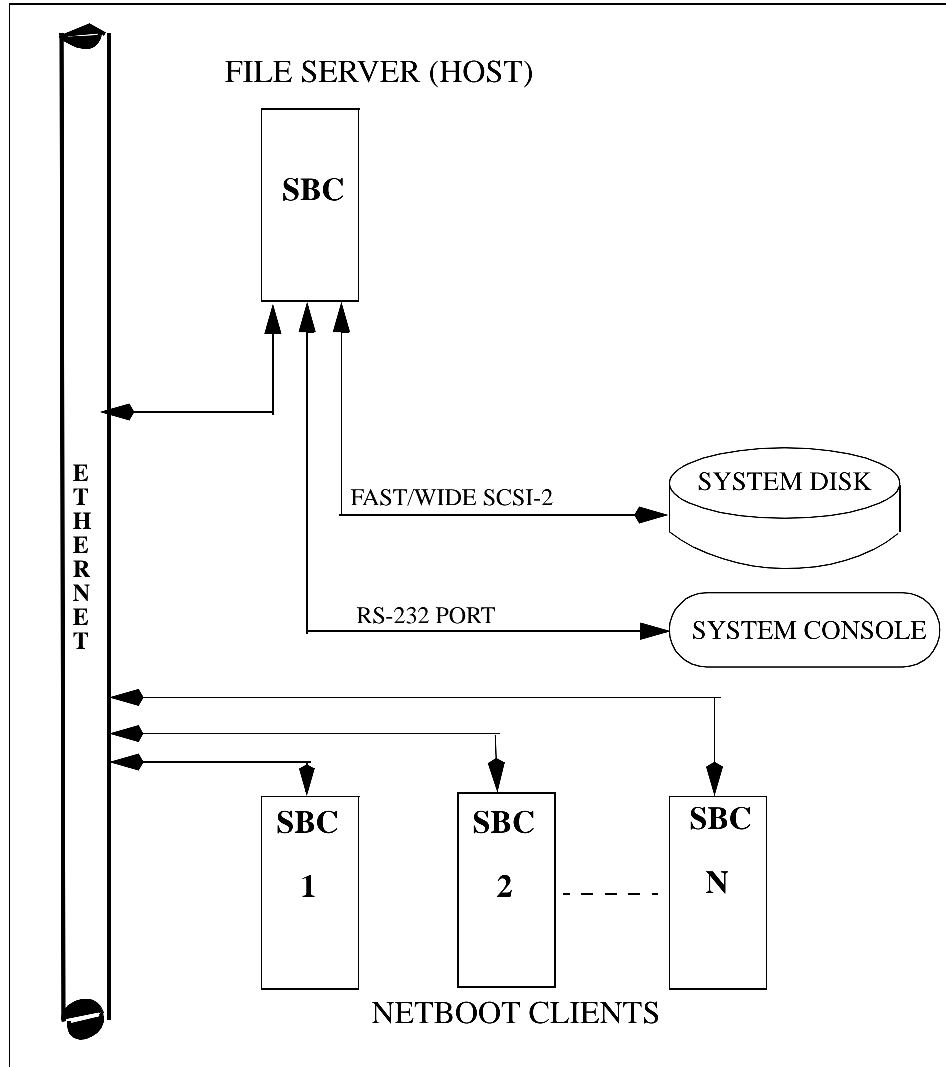


Figure 1-1. Loosely-Coupled System Configuration

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 applica-

tions 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.

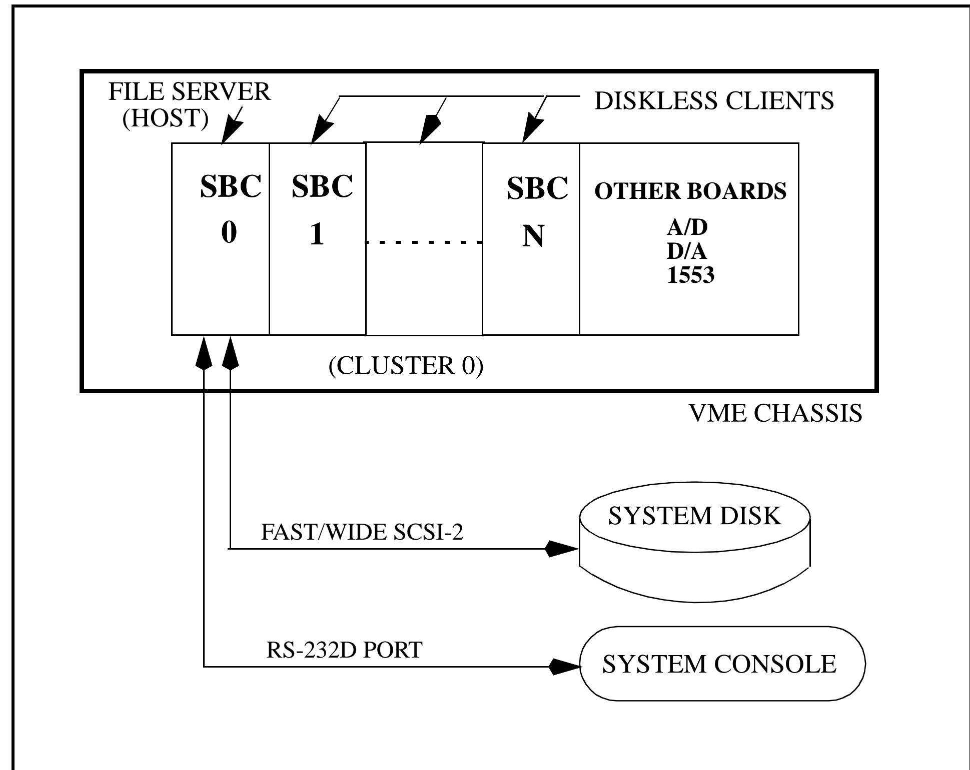


Figure 1-2. Closely-Coupled Cluster of Single Board Computers

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.

Note that it is possible to combine the above topographies and configurations in various ways. For example, one could have a Closely-Coupled system where some of the client SBCs are embedded clients and some are NFS clients.

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 environ-

ment used to build the boot image and it is also mounted by diskless systems that maintain an NFS connection to the File Server. Note that 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 three 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. When the diskless system shares the VMEbus with the File Server, the boot image can be downloaded from the File Server, across the VMEbus, directly into the memory of the diskless system.
3. The boot image may have already been burned into flash ROM. In this case, the board's firmware (**SMon**) 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 four techniques for initiating the boot sequence on a diskless system. Note that 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 (**SMon**) must be configured to boot from the network. The boot sequence is initiated either by resetting the board, by cycling the power on the board, or by manually issuing the **SMon** command to execute a TFTP boot. Note that the manual **SMon** method is only available when a console terminal is connected to the diskless system.
2. To boot over the VMEbus, the boot sequence is initiated by executing the **sbcboot** command on the File Server. This command causes the diskless system to be reset, the boot image downloaded over the VMEbus into the diskless system's memory, and execution of the downloaded boot image is then initiated.
3. To boot from flash ROM, the board's firmware (**SMon**) must be configured to boot from flash, through the use of a **SMon** startup script. The boot sequence will be initiated whenever the board is reset, cycling the power on the board or manually executing the **SMon** startup script or flash boot commands. Note that the manual **SMon** method is only available when a console terminal is connected to the diskless system.
4. If a diskless system with a flash ROM boot image shares the same VMEbus with the File Server, then a **sbcboot** command can be executed on the File Server that will initiate the boot process for the diskless system, if the diskless system is properly configured via a **SMon** startup script to execute a flash ROM boot after a reset. Note that in this case, the bootserver can initiate the boot sequence because the client system can be remotely reset by the File Server from across the VMEbus.

1.1.3. Configuration Toolsets

Two sets of tools are 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 tools that comprise both toolsets are executed on the File Server. One toolset is used for configuring closely-coupled systems while the other toolset is used for configuring loosely-coupled systems.

The closely-coupled toolset consists of the tools **`vmebootconfig`** and **`mkvmebstrap`**. The closely-coupled or VME toolset must be used if the single board computers in the configuration share a VMEbus and that VMEbus is going to be used for any type of inter-SBC communication. There are instances where clients in a closely-coupled VME configuration may wish to boot from an ethernet connection to the File Server or from flash ROM. The VME toolset provides support for such booting. Because these clients are part of a VME cluster, the VME toolset must be used to configure them.

The Net Boot toolset consists of the tools **`netbootconfig`** and **`mknetbstrap`**. These tools handle the simpler case of loosely-coupled systems that boot via an ethernet network or from flash ROM, where no VMEbus-based communication will be utilized on the client system.

The **`netbootconfig`** and **`vmebootconfig`** tools are used to create the diskless configuration environment for a diskless client. The **`mknetbstrap`** and **`mkvmebstrap`** tools are used for creating a diskless client's boot image. More information is provided on these tools in Chapter 3, “Netboot System Administration” and in Chapter 4, “VME Boot 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.
Closely-Coupled System (CCS)	A Closely-Coupled System (CCS) is a set of Single Board Computers (SBCs) which share the same VMEbus and also additionally share a common PCI-to-PCI (P0) bus. The first board must have an attached system disk and acts as the File Server for the other boards in the rack. The VMEbus can be used to download boot images for the diskless SBCs from the File Server SBC, and the P0Bus can be used for various types of inter-board communications.
Cluster	A cluster is one or more SBC(s) which reside on the same VMEbus and also share a common P0Bus. In general, a cluster may be viewed as a number of SBCs which reside in the same VME chassis. Note that "cluster" and "Closely-Coupled system" are synonymous.
Board ID (BID)	All SBCs in the cluster are assigned a unique board identifier or BID. The BIDs range from 0 to 7 in any given cluster. Every cluster must have the server SBC as BID 0. Additional SBCs installed in a cluster may use any of the remaining unused BID. By convention, BIDs are usually allocated sequentially [1,2,3] but this is not mandatory. There are additional considerations and restrictions that apply to BID assignments when a Backplane P0 Bridge Board (BPP0) is used to connect two P0Bus overlays together in a given cluster. See Appendix A "Backplane P0 Bridge Board Cluster ConfigurationBackplane P0 Bridge Board Cluster Configuration" for more details.
Host	Generic term used to describe Board ID 0 in the cluster (see definition of File Server immediately below).
File Server	The File Server has special significance in Loosely-Coupled and Closely-Coupled systems 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. In a Closely-Coupled System it is the SBC that downloads a boot image to all other clients in the same cluster across the VMEbus. There is only one File Server in a Loosely-coupled or Closely-Coupled system. The File Server must be configured as BID 0 for Closely-Coupled Systems. 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").

VME Boot — A master/slave kernel boot method by which the File Server resets, downloads and starts an operating system kernel on a client which is attached to the same VMEbus. Note that the client does not initiate the boot sequence.

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 (**`SMon`**) — A board-resident ROM monitor utility that provides a basic I/O system (BIOS), a boot ROM, and system diagnostics for Power Hawk Series 700 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.
P0Bus Networking	In a Closely-Coupled system, all SBCs within the same cluster may be transparently networked together using a P0Bus-based point-to-point network. This inter-SBC network provides a high speed TCP/IP connection between SBCs which reside on the same P0Bus (i.e. are in the same cluster).
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

Diskless capabilities are available on all Power Hawk Series 700 platforms.

The Power Hawk Model 710 platform is supported only as a loosely-coupled client.

The Power Hawk Model 720 and 740 platforms may be used in either loosely-coupled or closely-coupled configurations.

Motherboard Designation	System Platform	Number of CPUs	Form Factor	Netboot	VMEboot	Flashboot
VGM5	Power Hawk 710	1	VME 6U	yes	no	yes
VGM5	Power Hawk 720	2	VME 6U	yes	yes	yes
VSS4	Power Hawk 740	4	VME 6U	yes	yes	yes

1.3.1. Series 700 Hardware Features

Features	Power Hawk 710	Power Hawk 720	Power Hawk 740
SBC Processor	Single G4 PowerPC Processor	Dual G4 PowerPC Processors	Four G4 PowerPC Processors
Ports	Two RS-232D Serial Ports	Two RS-232D Serial Ports	Four RS-232D Serial Ports
CPU Interrupts	One 8-bit CPU Mailbox	Two 8-bit CPU Mailboxes	Four 8-bit CPU Mailboxes
Flash Memory	4/8/16/32/64 MB 8-bit wide user flash memory	4/8/16/32/64MB 8-bit wide user flash memory	4/16 MB 8-bit wide user flash memory
Cache Memory	2 MB L2 Backside Cache		
DRAM	64 MB to 512 MB SDRAM with Parity		
Ethernet	Fast Ethernet 10Base-T/100Base-T		
SCSI-2	Fast-20 SCSI (8/16-bit wide)		
Onboard Real-Time Clocks (RTCs)	Three 32-bit Counter/Timers		
I/O Buses	A32/D32/BLT64 VMEbus with Master/Slave controller functions. PMC Compliant slot. Optional 6U expansion board provides up to three additional PMC slots. A 64 bit wide PCI-to-PCI (P0) bus for inter-SBC communications.		

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. Note that 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 filesystems that are available to it, perhaps those on local disks or from across the network.
These other filesystems 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 filesystems 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 when the boot image is loaded into Flash. When the boot image resides in Flash, only those files actually in use will reside in (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 filesystems stored in Flash will be in a compressed format, in order to make maximum use of this relatively tiny device.

1.4.4. Booting

Once a bootstrap image is generated, it must be loaded and started on the client for which it was built. Three methods of booting are provided: VME Boot, Net Boot and Flash Boot. All three methods are supported in closely-coupled configurations while only net boot and flash boot are supported in loosely-coupled configurations. (These methods are explained in more detail in the following sub-sections.)

In configuring a diskless configuration, the user must decide which method of booting will be used for each diskless client. Booting of a diskless client consists of two distinct operations. First, the boot image must be downloaded to the diskless client. Downloading can be performed over the VMEbus (supported only in closely-coupled configurations) or across an ethernet network (supported in both closely-coupled and loosely-coupled configurations). Downloading using these two mechanisms differs in that a VME download is a 'push' operation, while an ethernet download is a 'pull' operation. That is, the VME download is initiated by commands that are executed on the File Server, which cause the boot image to be written to the client's DRAM. An ethernet download is initiated by a **SMon** command, either automatically, via a **SMon** startup script, or manually at the attached console terminal. Once executed on the diskless client. this **SMon** command will cause the boot image to be read from the File Server and downloaded into the client's DRAM.

The second phase of the boot operation is to initiate execution within the downloaded image. Like the download of the boot image, initiation of execution may be controlled either remotely from the VMEBus, or locally by **SMon** commands that are executed on the client systems. In managing a diskless configuration, the user will generally be unaware of the distinction between the two phases of the boot process. This is because both the VME download and the ethernet download methods actually perform both phases of the boot operation. The distinction between these two phases of booting is made here to better understand the third boot mechanism: booting from flash.

The boot image can be burned into flash on the diskless client. Once burned into flash, the complete image no longer needs to be downloaded into DRAM. The operating system is copied from flash into DRAM, the compressed **cpio** file system is left in flash and only copied into memory as needed. This means that booting from flash doesn't require phase one of the boot process - the download of the boot image. The method used to initiate execution within the boot image for a flash boot system depends upon the connection between the File Server and the client. If the diskless client and the File Server are located in the same VMEbus, then a remote reset of the diskless client (from across the VMEbus) can be issued on the File Server which will cause the flash boot sequence on the client to occur, if the client is properly configured to execute a flash boot **SMon** startup script after a reset. If the client has only a network connection to the File Server, or has no connection at all the File Server, then the boot sequence must be initiated by using a **SMon** startup script, which will be executed after any reset or power cycle.

Booting from flash is significantly faster than other boot mechanisms and is recommended for final deployed environments. While actively testing and modifying the application and the boot image, downloading the boot image can be performed via the VMEbus or via an ethernet connection. The final version of the boot image can then be burned into flash when the application is deployed.

1.4.4.1. VME Boot

In a Closely-Coupled configuration, the File Server (board ID 0 of cluster) is capable of downloading a boot image to all other clients in the same cluster across the VMEbus. VME booting (or VMEbus booting) uses the VMEbus to transfer the PowerMAX OS™ bootstrap image from the File Server's disk to the client's DRAM (memory).

A VME boot will result in the hardware reset of the boot client. This stops execution of any code running on the boot client. It is suggested that when rebooting any client that is currently running a PowerMAX OS™ bootstrap image, that it be shutdown (e.g. using **`shutdown(1M)`**) prior to rebooting (if possible).

Once the client is reset, the File Server's on-board DMA Controller transfers the bootstrap image from the File Server's disk to the client. A command, sent over the VMEbus to a special memory location on the client, starts execution of the bootstrap image.

1.4.4.2. Net Boot

Net Boot (or Network booting) is an alternative method of loading and executing a kernel image to a client over Ethernet. Note that the **`SMon`** resident monitor supports booting via ethernet but <u>not</u> other networking media. This method is distinguished by the fact that the host cannot actively force a client to accept and boot an image; rather, the client must initiate the transfer with the host and cooperate with the host to complete the transfer. This client initiation may take one of two forms: **`SMon`** command(s) executed on the client console by an operator, or an automatic execution of the same **`SMon`** command(s) by the client SBC whenever it is powered up, or reset through the use of a **`SMon`** startup script.

Net booting is performed by **`SMon`** using the TFTP (Trivial File Transfer Protocol, RFC783). This is a standard protocol that is supported by the PowerMAX OS™ and is explained in more detail in the *Network Administration* manual (Pubs No. 0890432).

Any client can be net booted as long as the SBC is physically connected to the same Ethernet as the File Server.

Whether manually booting or autobooting a client, that client must first be set up with the information it needs to do net booting. This is accomplished with the **`SMon smonconfig`** command. Once this is done, the client should not ever need to be reconfigured unless one or more of these parameters change, as the information is saved in NVRAM and thus will be preserved across reboots and power down cycles.

The following information is required by the **`SMon smonconfig`** command to configure each client that will be performing a net boot:

What should the Ethernet host address be?

> IP address for the File Server which should have already been defined in the **`/etc/hosts`** file. You must use the address which defines the Ethernet interface, not the P0Bus network interface.

What should the Ethernet target address be?

The local client SBC's IP address which **SMon** will use as the return address for TFTP data transfer. For NFS clients, this is the Ethernet IP Address of this client SBC. 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, and does not conflict with any other IP addresses used in the network.

What should the Ethernet mask be?

Subnet mask used for this interface. It is usually 255.255.255.0.

What should the Ethernet gateway address be?

If the client SBC's access to the File Server is through a gateway, then that system's gateway address should be entered here.

After the **SMon** has been initialized with the network parameters, a net boot may be initiated via the **SMon TftpBoot** command. Refer to Chapter 3, "Netboot System Administration", for more information on net booting.

1.4.4.3. Flash Boot

Flash Boot or flash booting, is a method of loading and executing a kernel image from Flash ROM. Flash booting is the preferred method of booting a diskless client in the production or deployed phase of an application. There are two advantages in booting from flash. First, flash boot allows very fast boot times because there are no rotational delays that would normally be associated with reading from disk. Second, the root file system is maintained as a read-only image in flash freezing memory that would otherwise have to be used to maintain an in-memory root filesystem, and thus provide 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 downloaded into flash is the same boot image that can be downloaded via the VME backplane or via an ethernet network connection. Therefore a developer can use one of the other download/boot techniques while actively modifying the boot image during the development phase, and then use the same final boot image in the flash when in the deployed phase of the application. There are no tools specifically targeted towards creating boot images for flash booting. Instead, either the loosely-coupled or closely-coupled configuration tools are used for building the image, depending on whether the client system will be used in a shared VME backplane configuration.

The first step in preparing an SBC for flash boot is to load the boot image into flash ROM on the board. Once the boot image is burned into Flash and a SMon startup script has been configured to execute after reset on the netboot client, then the same utilities can be used to initiate a download of the boot image from Flash to memory and begin execution within the downloaded boot image.

If the File Server is only connected to the client system via an ethernet network, then 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 above for net booting, by configuring the networking addresses with the **SMon smonconfig** command. Then the **SMon**

load command is used to load the boot image into memory without executing that image. The boot image is then burned into Flash from memory with the **SMon fp uf** command.

Once the image is burned into User Flash, then all subsequent booting can be done from the image stored in User Flash by setting up a **SMon** startup script to read from User Flash and execute the Flash boot image. Refer to Chapter 5, "Flash Boot System Administration", for more information.

1.4.5. P0Bus Networking

For closely-coupled configurations, the cluster software package provides the capability of networking between clients and the File Server, utilizing the P0Bus for data transfers. Figure 1-3 illustrates the streams networking model, which includes interface modules to enable standard networking across the P0Bus. This P0Bus network connection is a point-to-point link and is analogous to a SLIP or PPP connection. A point-to-point P0Bus connection is established via the **busdlpiattach(1M)** command. This command is executed at boot time by the File Server and clients to establish connections with the other Power Hawk boards in the same cluster. The **netstat(1M)** command shows the P0Bus point-to-point connections:

```
netstat -i
```

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Collis
lo0	8231	127.0.0.0	localhost	78	0	78	0	0
sym0	1500	129.1.34.32	astro	461	0	28	0	0
bus0	8320	129.1.34.0	astro-p0	8764	0	9364	0	0

The interface bus0 represents one P0Bus point-to-point connection with another Power Hawk board.

When configuring a cluster, the IP addresses of the Power Hawk systems that are used on the P0Bus are defined based upon the **busdlpi/BUSNET** IP Base Address of the cluster, and the appropriate entries must be added to the **/etc/hosts** file. Required kernel device drivers for P0Bus networking are configured, and applicable P0Bus network related tunable parameters are initialized to reflect the configured IP address during **vmebootconfig(1M)** processing.

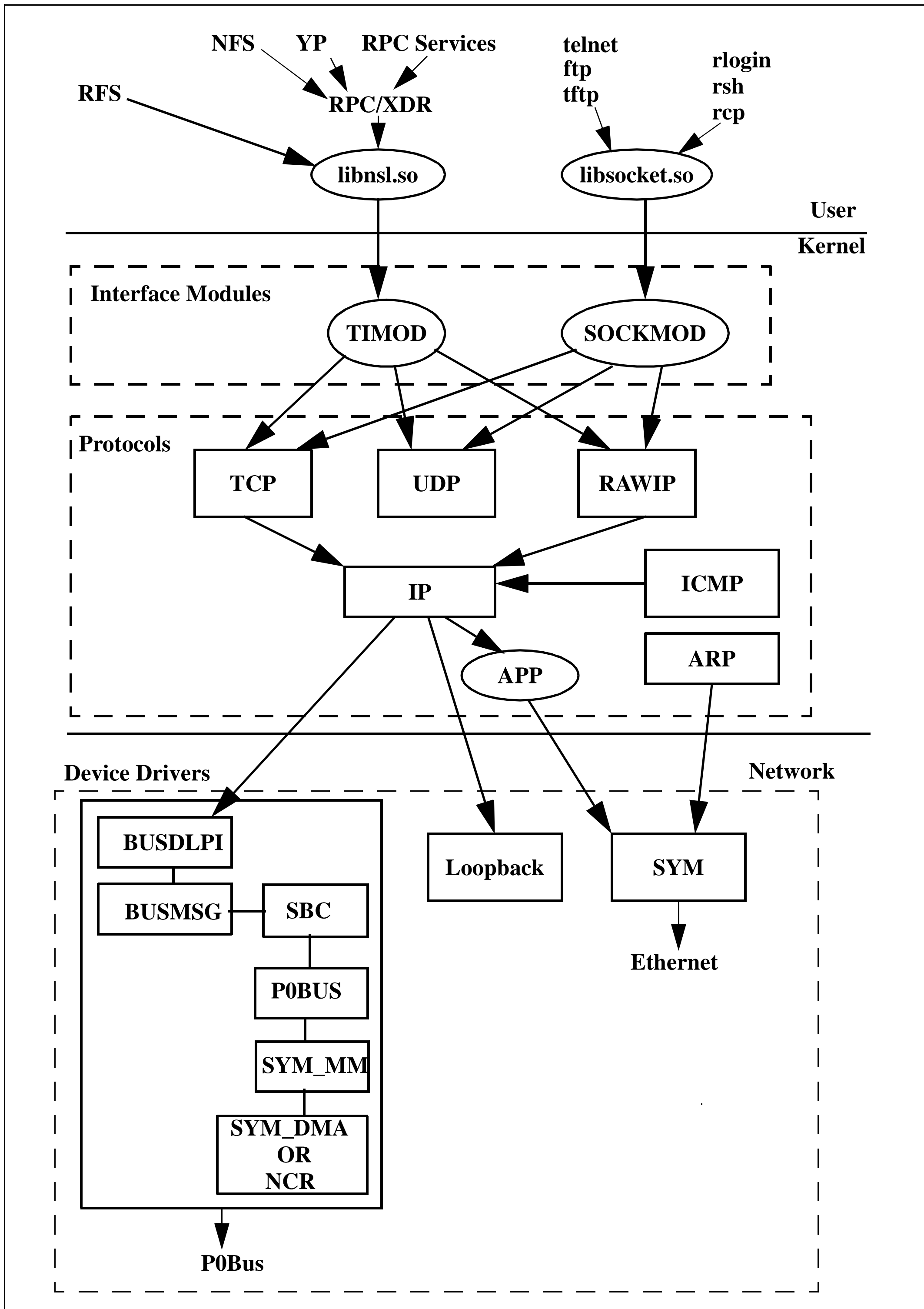


Figure 1-3. Power Hawk Networking Structure

1.4.6. 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:

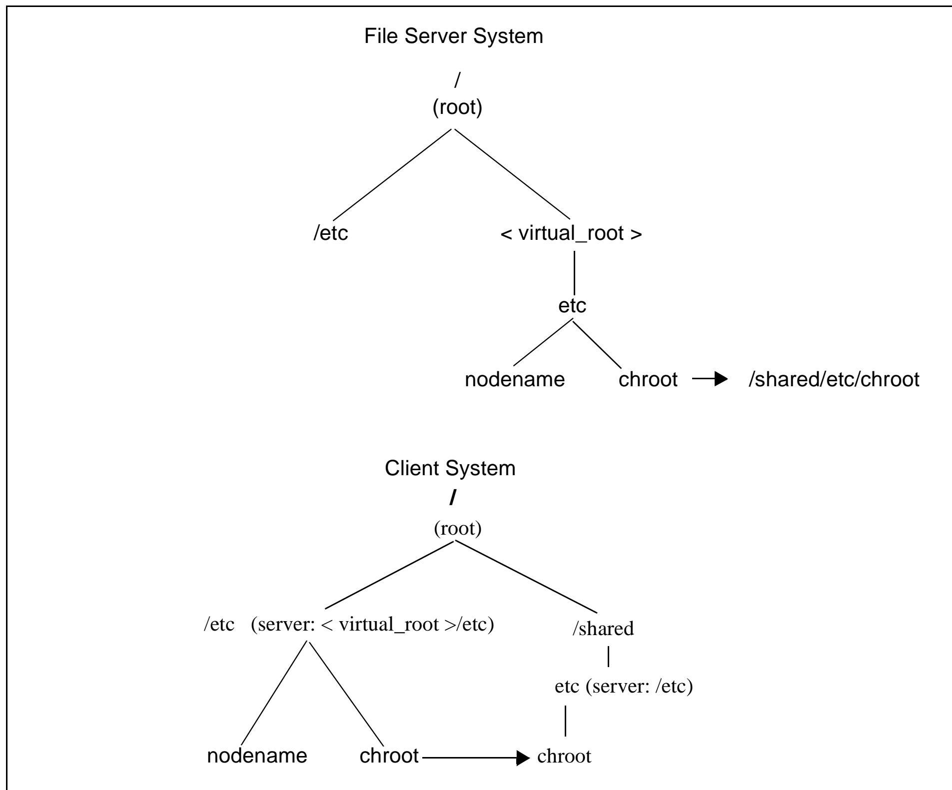
Path on File Server	Mount Point on Client
/usr	/usr
/sbin	/sbin
/opt	/opt
<virtual_rootpath>/etc	/etc
<virtual_rootpath>/var	/var
<virtual_rootpath>/dev	/dev
<virtual_rootpath>/tmp	/tmp
virtual_rootpath>/users	/users
/etc	/shared/etc
/var	/shared/var
/dev	/shared/dev

The **/etc** and **/var** directories under the client's virtual root contain some files that are client-specific and therefore these files cannot be shared. These directories also contain files which have the same content for the File Server and all client virtual roots on the File Server, these files are shared. Because the **/etc** and **/var** directories contain both shared and non-shared files, yet all files must reside in this directory (because the standard utilities expect them to be in those directories), these directories require special treatment in 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.



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 utilities **mkvmebstrap** and **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**, whose P0Bus networking nodenames are **client1-p0** and **client2-p0**, respectively, the following line appears in the **dfstab** table.

```
CLIENTS=client1:client1-p0:client2:client2-p0 /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, assuming that the P0Bus network nodename for **client1** is **client1-p0**.

from:

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

to:

```
/usr/sbin/share -F nfs -o root=client1:client1-p0 -d \
"/home/vroots" /home/vroots vroots
```

Note that 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, assuming that the P0Bus networking nodename for **client1** is **client1-p0**.

/usr/sbin/share -F nfs -o rw=client1:client1-p0,root=client1:client1-p0 \ -
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.

1.4.7. Shared Memory

Closely-Coupled systems can be configured such that SBCs within the same cluster can share memory with each other. This capability, called Slave MMap shared memory, is explained briefly below. Refer to Chapter 3 Shared Memory, located in the *Power Hawk Series 700 Closely-Coupled Programmers Guide*, for additional information.

The Slave MMap interface provides **mmap(2)**, **shmbind(2)**, **read(2)** and **write(2**) access to a configurable user-accessible shared memory area on each SBC.

Each SBC maps its own contiguous Slave MMap memory area into a predefined P0Bus physical address range with a downstream P0Bus-to-PCI window. All memory mapped in this way is remotely accessible simultaneously by all other SBCs in the cluster from across the P0Bus. Local SBC access out to the P0Bus is provided through a local upstream PCI-to-P0Bus window, which passes local PCI bus access requests through the upstream window and out onto the P0Bus.

1.4.8. 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. Note that 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 overcommitting 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. Closely-Coupled System Hardware Prerequisites

A Closely-Coupled VME cluster requires the following hardware:

- VME card chassis
- One Power Hawk 720 or 740 single board computer, with a minimum of 64 MB of DRAM for use as the server SBC.
- One Power Hawk 720 or 740 single board computer, with a minimum of 64 MB of DRAM for each client SBC.
- All SBCs must be connected to either a single P0Bus with a single P0Bus overlay board, or to one of two P0Bus overlay boards, where the two P0Bus overlays are connected together with one Backplane P0 (BPP0) bridge board. (See the *VGM5* or *VSS4 User Guide* for a more detailed description of the P0Bus Overlay board.)
- One SCSI 2 GB (4 GB or higher is preferred) disk drive for PowerMAX OS™ software installation, connected to the internal NCR/Symbios SCSI controller on the server SBC.
- One supported SCSI CD-ROM device, connected to the internal NCR/Symbios SCSI controller, 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.

1.5.2. Loosely-Coupled System Hardware Prerequisites

A loosely-coupled configuration requires the following hardware:

- At least one VME card chassis.
- One Power Hawk 710, 720 or 740 single board computer, with a minimum of 64 MB of DRAM, for use as the server SBC.
- One Power Hawk 710, 720 or 740 single board computer, with a minimum of 64 MB of DRAM, for each client SBC.
- One SCSI 2 GB (4 GB or higher is preferred) disk drive for PowerMAX OS™ software installation, connected to the internal NCR/Symbios SCSI controller on the server SBC.
- One supported SCSI CD-ROM device, connected to the internal NCR/Symbios SCSI controller 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. The on-board Symbios Ethernet controller (Symbios SYM53C885) should be used for this connection.

1.5.3. 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

1.5.4. 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 5.1 or later)	
cmds	Advanced Commands	lp, 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
SBC Hardware Considerations

2.1. Introduction 2-1
2.2. Unpacking Instructions 2-2
2.3. Board Jumpers 2-2
2.4. VGM5 Reset/SMI Toggle Switch 2-3
2.5. VSS4 Reset/SMI Toggle Switch 2-4

2
SBC Hardware Considerations

2.1. Introduction

This chapter provides hardware preparation, installation instructions and general operating information. The Single Board Computers (SBCs) including other VME modules, can be packaged in various VME chassis configurations depending on end-user application requirements. The chassis can vary in the number of slots available, and also, may be either rack-mount or desk top versions.

Refer to either the *Synergy Microsystems VSS4 VMEbus Quad G3/G4 PowerPC Single Board Computer for DSP User Guide*, or the *Synergy Microsystems VGM5 Dual G3/G4 PowerPC Single Board Computer User Guide* for more detailed information on hardware considerations that may be applicable to your particular hardware configuration. Refer to the Preface of this manual for specific manual titles and document numbers.

NOTE

Unless otherwise stated, this chapter applies to both the VGM5 (single and dual CPU models) and VSS4 SBCs.

CAUTION

Avoid touching areas of integrated circuitry; static discharge can damage circuits.

Concurrent strongly recommends that you use an antistatic wrist strap and a conductive foam pad when installing or upgrading a system. Electronic components, such as disk drive, computer boards, and memory modules, can be extremely sensitive to Electrostatic Discharge (ESD). After removing the component from the system or its protective wrapper, place the component flat on a grounded, static-free surface (and in the case of a board, component side up). Do not slide the component over any surface.

If an ESD station is not available, you can avoid damage resulting from ESD by wearing an antistatic strap (available at electronic stores) that is attached to an unpainted metal part of the system chassis.

2.2. Unpacking Instructions

NOTE

If the shipping container is damaged upon receipt, request that the carrier's agent be present during unpacking and inspection of the equipment.

Unpack the equipment from the shipping container. Refer to the packing list and verify that all items are present. Save the packing material for storing and reshipping of the equipment.

2.3. Board Jumpers

There are no jumper setup requirements for SBC boards whether they are used in closely-coupled, loosely-coupled or flash boot configurations.

However, the user should verify that the following jumpers, located on connector J02N (VGM5) or J02L (VSS4), are not installed:

Jumper Pins	Function	Remarks
5 & 6	VME System Controller.	No jumper should be installed.
7 & 8	VME64 Auto-System Controller Disable	No jumper should be installed.
9 & 10	User Defined Slot Number	No jumper should be installed.

Note that the Slot number jumper (9 & 10) is not used; in closely-coupled systems, the SBC board ID of the server SBC is always 0, and the SBC board ID of a client SBC is determined from a client board's **`SMon startup`** script, and not from its slot number value.

Note

For more information on the J02N/J02L jumpers, see the subsection entitled "**Jumpers"** in "**Section 2, Getting Started**" of the VGM5 or VSS4 SBC User Guide.

2.4. VGM5 Reset/SMI Toggle Switch

The VMG5 motherboard has a RESET and SMI toggle switch for each CPU. See Figure 2-1.

RESET	Assert either a **CPU** or board-level **RESET** as described below. Pushing a switch to the **right** asserts a CPU-level RESET to the corresponding CPU. The **CPU-X** (top) switch asserts a reset to the CPU on single CPU models and to **CPU-X** on the dual CPU models. The **CPU-Y** switch (bottom) asserts a reset to **CPU-Y** which has an effect only on dual CPU models. Pushing **both** switches to the right at the same time asserts a board-level reset on all VGM Series models: • Resets the CPU(s). • Resets all on-board components that have such a function and clears all on-board control registers. • Asserts a VME RESET if the board is serving as the System Controller.
SMI	Pushing a switch to the **left** asserts an SMI interrupt to the respective CPU. Pushing the bottom switch to the left has no effect on single processor boards.

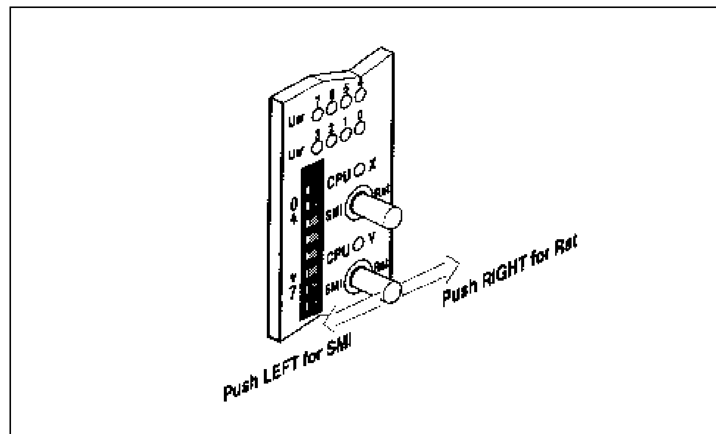


Figure 2-1. VMG5 Motherboard RESET and SMI Toggle Switch

2.5. VSS4 Reset/SMI Toggle Switch

The VSS4 motherboard is provided with a toggle switch for RESET and SMI interrupts. See Figure 2-2.

RESET	Pushing a switch to the **right** asserts a board-level RESET which: • Resets the CPUs. • Resets all on-board components that have such a function and clears all on-board control registers. • Asserts a VME RESET if the board is serving as the System Controller.
SMI	Pushing the toggle switch to the **left** asserts an SMI interrupt to all CPUs on the board.

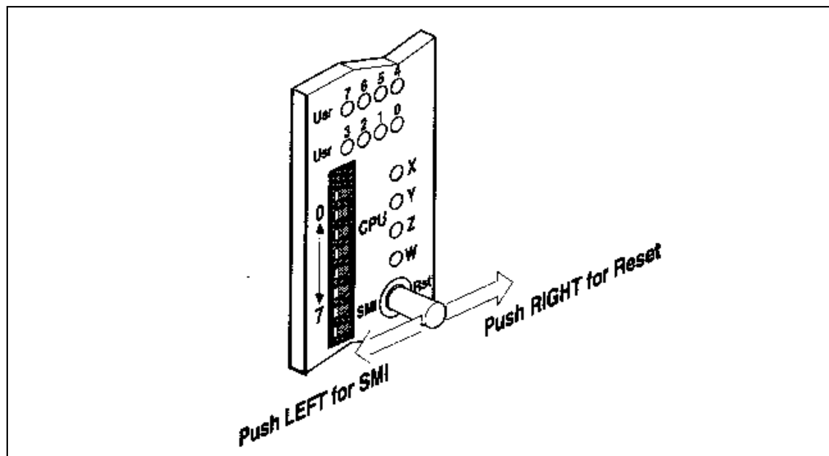


Figure 2-2. VSS4 Motherboard Reset and SMI Toggle Switch

3
Netboot System Administration

3.1. Configuration Overview 3-1
3.1.1. Installing a Loosely-Coupled System. 3-1
3.1.2. Installing Additional Boards 3-3
3.2. SBC Client Board Configuration 3-3
3.3. Client Configuration 3-8
3.3.1. The Client Profile File 3-8
3.3.1.1 Required Parameters 3-9
3.3.1.2 Required NFS-Related Parameters 3-9
3.3.1.3 Hosts Tables 3-11
3.3.2. Configuring Clients Using netbootconfig 3-11
3.3.2.1 Creating and Removing a Client Configuration 3-11
3.3.2.2 Subsystem Support 3-13
3.4. Customizing the Basic Client Configuration 3-13
3.4.1. Modifying the Kernel Configuration 3-14
3.4.1.1 kernel.modlist.add 3-14
3.4.1.2 mknetbstrap 3-15
3.4.1.3 config utility 3-15
3.4.1.4 idtuneobj 3-15
3.4.2. Custom Configuration Files 3-16
3.4.2.1 S25client and K00client rc Scripts 3-18
3.4.2.2 memfs.inittab and inittab Tables 3-19
3.4.2.3 vfstab Table 3-20
3.4.2.4 kernel.modlist.add Table 3-20
3.4.2.5 memfs.files.add Table 3-21
3.4.2.6 vroot.files.add Table 3-22
3.4.3. Modifying the Client Profile Parameters 3-24
3.4.4. Launching Applications 3-25
3.4.4.1 Launching an Application for Embedded Clients 3-25
3.4.4.2 Launching an Application for NFS Clients. 3-25
3.5. Booting and Shutdown 3-26
3.5.1. The Boot Image 3-27
3.5.2. Creating the Boot Image. 3-28
3.5.2.1 Examples on Creating the Boot Image 3-28
3.5.3. Net Booting 3-28
3.5.3.1 Netboot Using SMon. 3-29
3.5.4. Verifying Boot Status 3-30
3.5.5. Shutting Down the Client 3-30

3
Netboot System Administration

3.1. 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 or a P0Bus in this configuration.

The following instructions assume that all the prerequisite hardware has been installed and each netboot client's on-board Symbios 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 1-23, and refer to Chapter 2 for hardware setup considerations.

NOTE

Loosely-coupled diskless clients may be Power Hawk Model 710, 720, 740, 910, 920, or 940 systems. However, note that the Model 710 system and the Power Hawk 900 Series are not supported as a Closely-coupled diskless client. (See Chapter 4, "VME Boot System Administration" for more information about closely-coupled diskless clients.) Refer to Power Hawk 900 Series Diskless System Administrators Manual for Series 900 information.

3.1.1. 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" section on page 1-24 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.

Note that 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 filesystem 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 1-23 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 section “The Client Profile File” on page 3-8, and there is also an online description of these parameters in the file **/usr/etc /diskless.d/profiles.conf/netboot.client.profile.README**.

NOTE

Be careful NOT to use **vmebootconfig(1M)** to create the initial client profile file. The **vmebootconfig(1M) -P** client option will output the contents for a closely-coupled client profile file, and this CCS version of a client profile file is NOT suitable for configuring netboot clients.

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 their embedded Symbios Ethernet controller.

5. On the File Server system, execute **netbootconfig(1M)** to configure the build environment of each diskless client to be configured. See “Configuring Clients Using netbootconfig” on page 3-11" for more information.

6. On the File Server system, execute **mknetbstrap(1m)** to create the boot images of each diskless client. See “Booting and Shutdown” on page 3-26 for more information.

7. On each client, connect a console terminal and power up the netboot client. Use **SMon** to set the hardware clock, to setup the **SMon** networking configuration, and to optionally configure a **SMon netboot startup** script. See “SBC Client Board Configuration” on page 3-3 for more details.

8. On each client, either:

- manually execute a **SMon tftpboot** command to boot the client, or
- reset or power-cycle the client board, if the client is configured with an **SMon netboot/tftpboot startup** script.

   See “Net Booting” on page 3-28 for more details on booting up the client.

9. 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 3-26 for more information.

3.1.2. Installing Additional Boards

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

3.2. 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 *SMon PowerPC Series SBCs Application Developer & Debugger User Guide* for additional information regarding **SMon** 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 client SBC board must be setup to execute **SMon** after power-on or after reset. However, if the board is setup to execute the **fdiag** diagnostic program after power-up or reset, then this must be modified as follows:

NOTE

In the following discussion, **<cr>** stands for hitting the Return/Enter key on the keyboard.

Power-up or reset the board and watch for the resulting input prompt:

a. If the:

   `SMon0>`

   prompt appears, skip the rest of this step and go to step #3 below.

b. If the:

   `To change any of this, hit any key...`

   text appears, then hit any key, and then type **`X<cr>`** to enter **`SMon`**. Skip the rest of this step and go to step #3 below.

c. If the:

   `fdiag0>`

   prompt appears after the initial power-on/reset information is displayed, then the board is currently setup to execute the **`fdiag`** diagnostic program instead of **`SMon`**. To change this, enter:

   `config<cr>`, and then enter **`<cr>`**

   to step through this command prompts/questions until the:

   `SMon boot enabled [Y]:`

   prompt appears. Answer **`Y<cr>`** to this prompt. Enter **`<cr>`** until the rest of the prompts/questions are completed for the **`config`** command. Now enter **`reboot<cr>`** to reboot the board into **`SMon`**.

   If the:

   `To change any of this, hit any key...`

   text appears, then hit any key, and then type **`X<cr>`** to enter **`SMon`**.

3. Now that the SBC board executes **`SMon`** by default, make sure that the post script is not enabled. To disable execution of the post script,

   enter:

   `config <cr>`

   and enter **`<cr>`** to step through this command's prompts/questions until the:

   `post script enabled [N]:`

   prompt appears. Answer **`N <cr>`** to this prompt.

   Enter **`<cr>`** until the rest of the prompts/questions are completed for the **`config`** command.

Now enter **`reboot<cr>`** to reboot the board into **`SMon`**.

If the:

```
To change any of this, hit any key...
```

text appears, then hit any key, and then type **`X<cr>`** to enter **`SMon`**.

4. This step describes how to configure the IP address networking parameters that will be used by the embedded Symbios Ethernet controller when executing under **`SMon`**.

   To set up these parameters, type **`smonconfig<cr>`** and then hit **`<cr>`** to step through the other prompts until the Ethernet parameters appear:

   ```
   ETHERNET PARAMETERS:
   What is the board’s serial number? [1014042]
   What should the Ethernet host address be? [129.134.30.26]
   What should the Ethernet target address be?
   [129.134.32.79]
   What should the Ethernet mask be? [255.255.255.0]
   What should the Ethernet gateway address be?
   [129.134.32.196]
   ```

   The ‘`Ethernet host 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.

   The ‘`Ethernet target address`’ should be set to the IP address of this client SBC. This IP address should be the Symbios Ethernet address of this client SBC that is already in the File Server SBC's **`/etc/hosts`** file.

   If the File Server and this client SBC are not connected to the same ethernet subnetwork, then the ‘`Ethernet gateway address`’ address must be set to a gateway system that provides access to the File Server’s subnetwork.

   Hit **`<cr>`** to step through all the other **`smonconfig`** parameter prompts until this command completes.

5. The netboot client's hardware clock must be updated to match the date on the system designated as the File Server.

   Use the ‘**`SMon`** '**`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:

   ```
   SMon0> date<cr>
   ```

   ```
   Mar 7 01:02:05 1996
   ```

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

```
date month day hour min sec year
```

You must use a '**#**' prefix when entering decimal values. For example, to set the date to August 22, 1996 and the time to 14:55:30:

```
SMon0> date 8 #22 #14 #55 #30 #96
```

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

   a. If you want to always manually boot the client, then configure **SMon** so that it will NOT execute the startup script after power-on or reset: type **smonconfig<cr>** and answer **1<cr>** at the prompt below to change the configuration to **SMon** without a **startup** script.

   ```
   The SMon ROM can be used in several ways:
   (1) ROM-boot SMon Stand-alone
   (2) ROM-boot SMon with startup script
   Which one do you want? [1]:1<cr>
   ```

   Then hit **<cr>** to step through other prompts.

   b. If you want the board to automatically attempt a **tftp** download and boot from the File Server after the client is powered-cycled or reset, then first create the following **SMon startup** script:

   ```
   vi "startup" 1
   ```

NOTE

While **SMon** provides a subset of **vi** commands for editing the **startup** script, be careful when exiting the **vi** editor. Be sure to type **:q** when exiting **vi**, and not **:w**. Typing **:w** will cause the contents of the startup script to be lost and for all **SMon** configuration settings to be reset back to their factory default values.

The required contents of the client netboot **startup** script is shown below:

------------------------Startup Script ----------------------------

```
tftpboot "<client>.bstrap"
```

-----------------------End of Startup Script ---------------------

where the **<client>** value above should be replaced by the

actual hostname of this client system. For example, if this client's hostname is '*wilma*', then the following one line **startup** script will download and boot this client from the File Server:

tftpboot "wilma.bstrap"

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.

After exiting the **vi** editor with the **:q** command, then **SMon** must be configured to execute this startup script after a power-cycle or reset. Type **smonconfig<cr>** and answer **2<cr>** at the prompt below to change the configuration to **SMon** with a **startup** script.

```
The SMon ROM can be used in several ways:
(1) ROM-boot SMon Stand-alone
(2) ROM-boot SMon with startup script
Which one do you want? [1]:2<cr>
```

Then hit **<cr>** to step through some of the other **smonconfig** command prompts, until the following prompt appears:

How long (in seconds) should CPU delay before starting up? [5]

Note that in order to ensure that a client can be successfully download its boot image via **tftpboot**, the server system MUST be completely booted into **run level 3** before attempting to **tftpboot** any clients.

Normally, the default value of 5 seconds for the CPU delay time parameter in the line above is a sufficient delay time if the netboot client is to be manually reset or power cycled after the server system is known to have completely finished its boot up sequence into **run level 3**.

However, if the netboot clients are located in the same rack as the server, and if it is desirable to have the netboot clients boot themselves up after a power cycle sequence without the need for any additional manual resets, then the default **SMon** delay of 5 seconds must be significantly increased. In this case, the **SMon** CPU delay time must be increased to a value that will allow enough time for the server to power up and completely finish its own boot up sequence into **run level 3**.

To come up with a reasonable delay time value, it is suggested that you power cycle the rack and measure how long it takes the server to boot up into **run level 3** (until you get a login prompt at the server's console terminal). It is recommended that you add approximately 30 additional seconds to this amount of time, as a safety margin, and then use this value as the SMon startup delay time value.

Note that this **SMon** delay before startup parameter value applies to all startup sequences, including manual resets as well as power cycle situations. However, the user may

manually interrupt the startup delay processing by touching any character on the keyboard, if a console terminal is connected to the client board.

After either entering **<cr>** for using a default value of 5 seconds, or after entering a new value such as 90 seconds followed by a **<cr>**, the user should then hit **<cr>** to step through the remaining prompts until the **smonconfig** command has been completed.

3.3. 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 3-11)

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.

3.3.1. 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. Note that 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. Note that the client profile file name should be equivalent to the client's hostname.

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.

3.3.1.1. 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. (This parameter is used by the diskless tools as a method for differentiating between loosely-coupled netboot and closely-coupled client profile files.)

3.3.1.2. Required NFS-Related Parameters

The following netboot client profile parameters are required only if the client is a NFS client (SYS_CONFIG=nfs). Note that 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

3.3.1.3. Hosts Tables

For each loosely-coupled client profile file created in the **/etc/profiles** directory, an entry for that client's embedded Symbios 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. Note that this IP address should match the value entered for this client SBC under the **SMon smonconfig** command's "**Ethernet target address**" parameter, as described under step #4 in the section "SBC Client Board Configuration", beginning on page 3-3.

3.3.2. 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**).

3.3.2.1. Creating and Removing a Client Configuration

When creating new client configurations, the client profile parameters must already be setup in the **/etc/profiles** client profile files (see "The Client Profile File" on page 3-8) 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 3-11 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*

3.3.2.2. 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**).

Note that 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*

3.4. Customizing the Basic Client Configuration

This section contains information on the following major topics:

- Modifying the Kernel Configuration (page 3-14)
- Custom Configuration Files (page 3-16)
- Modifying the Client Profile Parameters ()
- Launching Applications (page 3-25)
  - Launching an Application (Embedded Clients) (page 3-25)
  - Launching an Application (NFS clients) (page 3-25)

3.4.1. 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.

Note that 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. Note that 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.

3.4.1.1. 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 3-16.

3.4.1.2. 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.

3.4.1.3. 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.

Note that 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**).

3.4.1.4. 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.

Note that 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.

3.4.2. 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 netbooted 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 also supporting 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.

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.

3.4.2.1. 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. Note that 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.

3.4.2.2. 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.

Note that 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.

3.4.2.3. 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.

Note that 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.

3.4.2.4. 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 3-14.

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.

3.4.2.5. 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, then 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.

Note that, 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.

Also note that 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, since pages are brought into DRAM memory from flash only when referenced.

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>
```

3.4.2.6. 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**. Note that 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>**).

3.4.3. 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. Note that these parameters only apply to NFS netboot clients. Also note that for both of 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. Note that 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 3-8 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.

3.4.4. Launching Applications

Following are descriptions on how to launch applications for:

- Embedded Clients
- NFS Clients

3.4.4.1. 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 3-21, 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 3-16 for more information about the **memfs.files.add** table and the **S25client rc** script.

3.4.4.2. 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 ROM, this is not the case, since the memfs file system remains in flash ROM 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 3-16 for more information about editing the **vfstab** file.

3.5. Booting and Shutdown

This section describes the following major topics:

- The Boot Image (page 3-27)
- Creating the Boot Image (page 3-28)
- Net Booting (page 3-28)
- Verifying Boot Status (page 3-30)
- Shutting Down the Client (page 3-30)

3.5.1. 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 3-1. These files are explained in detail under section "Customizing the Basic Client Configuration" on page 3-13.

Table 3-1. 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

3.5.2. 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. Note that 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.

3.5.2.1. 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**

3.5.3. 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 **SMon**

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 5, "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 **SMon** networking parameters have been setup on the client board with the **SMon smonconfig** command.

   Additionally, if the client is to automatically boot up after a reset or power-cycle, then verify that the **SMon startup** script has been setup to **tftpboot** from the File Server, and verify that **SMon** is configured to execute the startup script as part of the **SMon** boot procedure. See "SBC Client Board Configuration" on page 3-3 for details.

2. Verify that the boot image has been created. (See "Creating the Boot Image" on page 3-28.)

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

3.5.3.1. Netboot Using SMon

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 via a **SMon tftpboot startup** script, 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 **SMon** command at the client's SMon console terminal:

  tftpboot <client_profile_filename>.bstrap

  where **<client_profile_filename>** is equal to the client profile filename of the client on the File Server system. This command will download and execute the specified bootstrap image from the File Server system's **/tftpboot** directory and then execute the image.

3.5.4. 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. Note, however, that 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.

3.5.5. 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.

4
VME Boot System Administration

4.1. Overview . . . 1-1
4.2. Cluster Configuration Overview . . . 1-1
4.2.1. Installing the Cluster . . . 1-2
4.2.2. How To Boot the Cluster . . . 1-4
4.2.3. Installing Additional Boards in a Cluster . . . 1-5
4.3. SBC Cluster Configuration . . . 1-6
4.3.1. Board Jumpers . . . 1-6
4.3.2. Installing the P0Bus Overlay . . . 1-7
4.3.3. File Server Board Configuration . . . 1-7
4.3.4. Client Board Configuration . . . 1-11
4.4. Cluster Configuration . . . 1-18
4.4.1. The Profile Files . . . 1-18
4.4.1.1 The cluster.profile File . . . 1-19
Cluster-wide Parameters . . . 1-19
File Server SBC Parameters . . . 1-24
4.4.1.2 The Client Profile File . . . 1-26
Required Parameters . . . 1-26
NFS Related Parameters . . . 1-29
Shared Memory Parameters . . . 1-29
4.4.1.3 Networking Hostname Naming Conventions . . . 1-31
4.4.2. Node Configuration . . . 1-33
4.4.2.1 Creating and Removing a Client . . . 1-34
4.4.2.2 Subsystem Support . . . 1-35
4.4.2.3 Slave Shared Memory Support . . . 1-36
Static Memory Allocations . . . 1-37
Dynamic Memory Allocations . . . 1-38
4.4.2.4 System Tunables Modified . . . 1-39
4.5. Customizing the Basic Configuration . . . 1-40
4.5.1. Modifying the Kernel Configuration . . . 1-40
4.5.1.1 kernel.modlist.add . . . 1-41
4.5.1.2 mkvmebstrap . . . 1-42
4.5.1.3 config Utility . . . 1-42
4.5.1.4 idtuneobj . . . 1-42
4.5.2. Custom Configuration Files . . . 1-43
4.5.2.1 S25client and K00client rc Scripts . . . 1-45
4.5.2.2 Memfs.inittab and Inittab Tables . . . 1-46
4.5.2.3 vfstab Table . . . 1-47
4.5.2.4 kernel.modlist.add Table . . . 1-47
4.5.2.5 memfs.files.add Table . . . 1-48
4.5.2.6 vroot.files.add Table . . . 1-49
4.5.3. Modifying Profile Parameters . . . 1-51
4.5.3.1 Cluster.profile File . . . 1-51
4.5.3.2 Modifying Client Profile Settings . . . 1-54
4.5.4. Launching Applications . . . 1-55
4.5.4.1 Launching an Application (Embedded Client) . . . 1-55
4.5.4.2 Launching an Application (NFS Client) . . . 1-55
4.6. Booting and Shutdown . . . 1-56

4.6.1. The Boot Image 1-57
4.6.2. Booting Options 1-58
4.6.3. Creating the Boot Image 1-60
4.6.4. VME Booting 1-61
4.6.5. Net Booting 1-62
4.6.6. Flash Booting 1-62
4.6.7. Verifying Boot Status 1-62
4.6.8. Shutting Down the Client 1-63

4
VME Boot System Administration

4.1. Overview

This chapter describes in detail the following operations and/or procedures:

- Cluster Configuration Overview (below)
- Installing the Cluster (page 4-2)
- Cluster Configuration (page 4-18)
- Customizing the Basic Configuration (page 4-40)
- Booting and Shutdown (page 4-56)

4.2. Cluster Configuration Overview

This is an overview of the steps that must be followed in configuring a closely-coupled configuration. Each of these steps is described in more detail in the sections that follow.

This section covers the following topics:

- Installing the Cluster (page 4-2)
- How to Boot the Cluster (page 4-4)
- Installing Additional Boards in the Cluster (page 4-5)

A closely-coupled cluster consists of one File Server SBC and one or more diskless client SBCs.

The standard way of booting clients in a cluster is over the VMEbus. Client SBCs in the cluster are VME booted from SBC 0, which acts as a boot and File Server for the entire cluster. The actual commands that cause clients to be booted are run on the File Server SBC.

Any client within a cluster could be configured to network boot via an ethernet network connection. This is normally accomplished by using an appropriate Ethernet Multi-Port Hub (AUI or 10BaseT), or connecting to a 10Base2 (BNC) subnet using an AUI-to-BNC transceiver. However, when a diskless client is within a cluster configuration, VME booting is the preferred booting method since it does not require additional hardware setup.

4.2.1. Installing the Cluster

Follow these instructions to install a cluster.

Note that symbolic links have been added to allow more convenient access to the configuration files. The file **/etc/profiles** is a link to the directory **/usr/etc/diskless.d/profiles.conf** and **/etc/clients** is a link to **/usr/etc/diskless.d/client.conf**.

1. Install all SBC boards and setup the SMon configuration parameters for all boards, and set up the **SMon startup** scripts for each client board. (See "Client Board Configuration" on page 4-11 for information about setting up the **SMon startup** script.)

2. Install the File Server (SBC 0 in the local cluster) with the prerequisite software packages, the diskless package, and all patches. Refer to "Chapter 1 Software Prerequisites" and the applicable system release notes for more information.

3. Configure and mount filesystem(s) other than **/** (root) or **/var** that can be used to store the virtual root directories for each client. Each **vroot** directory requires approximately 15-25MB of disk space (see "Disk Space Requirements" on page 1-23 for more details). This value is for the default shipped configuration. You must also consider swap space which is implemented as a file under the vroot directory. Adding applications to the vroot and configuring kernel modules will also increase the disk space requirements.

   If not already present, an entry for this filesystem must be added to **/etc/vfstab**. An existing filesystem can be used, but there must be sufficient file space to hold the client's virtual root files.

4. Edit the **/etc/hosts** file on the File Server SBC so that it contains the networking information for each diskless NFS client SBC. For information on client hostnames, see "Networking Hostname Naming Conventions" on page 4-31.

5. Create the **cluster.profile** file in the **/etc/profiles** directory. There are two sections to the **cluster.profile** configuration file. The first section describes a closely-coupled cluster as a whole. The second section has optional parameters that apply to the File Server. A sample cluster profile can be created with the following command:

   vmebootconfig -P *cluster* **>** /**etc/profiles/cluster.profile**

   You may then edit this cluster.profile file to suit the particular needs of the cluster and File Server SBC. See the section "The cluster.profile File" on page 4-19 for explanations on the parameters located in this file.

NOTE

If shared memory is to be configured for the cluster, the parameters SBC_SLAVE_MMAP_MAXSZ and VME_DRAM_WINDOW should be assigned appropriately before configuring the SBCs. These parameters cannot be easily modified after clients are configured. See the section "Cluster-wide Parameters" on page 4-19 for more information on these parameters.

6. For each client SBC installed, add a client profile file in the same **/etc/profiles** directory. You can use **vmebootconfig(1M)** to print out a template of a VME boot client profile. The client profile file name must be equivalent to the client's networking hostname. For example, to create a client profile for a client with hostname wilma, do the following:

   vmebootconfig -P *client* **> /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 section "The Client Profile File" on page 4-26.

7. At this time, the **/etc/profiles** directory should contain the file **cluster.profile** and a client profile file for each of the clients in the cluster.

   The **vmebootconfig(1M)** command is used to modify the File Server's kernel configuration to run in closely-coupled mode and to configure the build environment for each client's boot object. For example, to create a new build environment for two client SBCs with client profile filenames *wilma* and *fred*, execute the following:

   /usr/sbin/vmebootconfig -C *wilma fred*

   The File Server's kernel configuration, if not already modified in a previous invocation of the tool, will also be modified at this time. See the **vmebootconfig(1M)** online man page for details on creating, updating, and removing client build environments.

8. Build the boot image objects for the client SBCs with **mkvmebstrap(1M)**. For example, to build the boot objects for the clients *wilma* and *fred*, execute the following:

   /usr/sbin/mkvmebstrap *wilma fred*

   See the **mkvmebstrap(1M)** man page for details on building a boot image object.

9. Rebuild the File Server's kernel using **idbuild(1M)** and then reboot the File Server SBC.

10. First boot the default client configuration before attempting to customize any client configurations. For example, to boot clients *wilma* and *fred*, use

the command:

/usr/sbin/mkvmebstrap -B *wilma fred*

See the section "How To Boot the Cluster" on page 4-4 for more information about various types of client booting scenarios.

As needed, you may customize the client configuration, rebuild boot images and reboot a client using **mkvmebstrap(1M)**. See Section "Customizing the Basic Configuration" on page 4-40 for more information.

NOTE

It is strongly advised that you first boot the default client configuration before attempting to customize any client configuration.

4.2.2. How To Boot the Cluster

The method for booting a client system depends on whether the client boot is initiated via the VME backplane (VME boot) or is initiated from the client system (self boot). Note that systems which boot from a boot image that is loaded in flash fall into either the VME boot or self boot categories depending on whether or not the boot process is initiated over the VME bus. A client that is booted over ethernet is always self booted. A client that downloads its boot image over the VME bus is always VME booted.

When a client boot is initialized via the VME backplane, there is the option of automatically booting client systems as part of the File Server's initialization. Listed below are three possible scenarios for booting client systems. For full details of the interaction of the boot interface and automatic booting, see "Booting Options" on page 4-58.

a. VME boot clients with autoboot

When the File Server is booted, the client systems that have autoboot configured will be automatically booted as part of the File Server system initialization processing. When a client's AUTOBOOT parameter is set to '**y**' in its profile file, a hidden file named **.autoboot** will be created under the client's virtual root directory. This file will serve to indicate that the client SBC should be automatically booted or shutdown by the File Server SBC whenever the File Server is booted or shutdown. Note that client booting in this case is performed by background processes. If a manual boot of the client is initiated while the background processes are actively booting the client, then the boot up results will be undefined.

b. VME boot clients with no autoboot

When no hidden **.autoboot** file exists in a client's virtual root directory, then this client SBC will not be automatically booted during the File Server SBC's initialization processing. In this case, the client SBC must be manually booted by executing the **mkvmebstrap(1M)** utility with the proper options (in particular, the **-B** option which will cause the client SBC to be booted).

See the **mkvmebstrap(1M)** man page for details on all the options available for rebuilding client boot images and for booting client SBCs.

c. Netboot clients

These clients are booted via **SMon** commands. The **SMon smonconfig** networking parameters must be setup for netbooting. See "Net Boot" on page 1-13 for a description of these networking parameters. Use **mkvmebstrap(1M)** to create the boot image for the client SBC. Execute the **TftpBoot SMon** command manually on the client system's console, or automatically at reset or power-cycle time, using a **SMon startup** script, to Net boot the client from an ethernet connection.

d. Flashboot clients

These clients are booted at reset time from the client's Flash memory through the use of a **Smon startup** script, or manually, on the client system's console.

4.2.3. Installing Additional Boards in a Cluster

Use these instructions when adding additional boards in a cluster, after the initial configuration of the closely-coupled system.

1. If the board(s) have not yet been installed:

   a. Perform an orderly system shutdown of each client running in the cluster and then shutdown the File Server and power down the entire cluster.

   b. Install all additional SBC boards and initialize the **SMon** parameters and **SMon startup** script on each board. See Section "Client Board Configuration" on page 4-11 for more information.

2. For each additional client board, add a VME boot client profile file in the **/etc/profiles** directory. Use **vmebootconfig(1M)** to print out a template of a client profile. For example, to create a client profile for a client with hostname *wilma*, do the following:

   vmebootconfig -P *client* **> /etc/profiles/wilma**
   vi /etc/profiles/wilma

   Edit the resulting client profile file to be relevant to the specific characteristics of the new client board. The parameters listed in the client profile are described in **/etc/profiles/client.profile.README**, and are discussed in section "The Client Profile File" on page 4-26.

3. Execute **vmebootconfig(1M)** to configure the build environment for the new client(s), specifying each new client's profile filename on the invocation line.

4. Build the boot image objects and boot the new client SBCs with **mkvmebstrap(1M)**. For example, to build the boot objects and boot

two new clients *barney* and *dino*, execute the following:

/usr/sbin/mkvmebstrap -b *barney dino*

See the **mkvmebstrap(1M)** man page for details on building a boot image object. As previously mentioned, you are highly encouraged to first successfully boot a new client board before attempting to customize any client configuration.

5. As needed, customize the configuration of the new clients and reboot new clients using **mkvmebstrap**. See Section "Customizing the Basic Configuration" on page 4-40 for more information.

4.3. SBC Cluster Configuration

This sections covers the Power Hawk 720 and 740 SBC board installation in a closely-coupled cluster system.

Note

Power Hawk 710 SBCs are NOT supported in a closely-coupled configuration. However, Power Hawk 710 SBCs are supported in a loosely-coupled configuration. See Chapter 3 "Netboot System Administration" for details on loosely-coupled system configuration.

Major topics covered in this section are:

- Board Jumpers (below)
- Installing the P0Bus Overlay (page 4-7)
- File Server Board Configuration (page 4-7)
- Client Board Configuration (page 4-11)

4.3.1. Board Jumpers

There are no jumper setup requirements for Power Hawk Series 700 boards, whether they are used in closely-coupled, loosely-coupled, or flash boot configurations. However, the

user should verify that the following jumpers, located on connector J02N (VGM5) or J02L (VSS4), are not installed.

Jumper Pins	Function	Remarks
5 & 6	VME System Controller.	No jumper should be installed.
7 & 8	VME64 Auto-System Controller Disable	No jumper should be installed.
9 & 10	User Defined Slot Number	No jumper should be installed.

Note that the Slot number jumper (9 & 10) is not used; in closely-coupled systems, the SBC board ID of the File Server SBC is always 0, and the SBC board ID of a client SBC is determined from a client board's **`SMon startup`** script, and not from its slot number value.

4.3.2. Installing the P0Bus Overlay

A Power Hawk Series 700 closely-coupled system requires a P0Bus overlay, which is used to connect all SBCs in a cluster. The P0Bus is utilized for inter-SBC communications within the cluster.

The P0Bus is available in either a 4-slot or a 5-slot model. The P0Bus overlay is attached to the VMEBus backplane and thus each slot of the P0Bus corresponds to a VMEBus slot. If some VMEBus slots may not be utilized due to attached PMC cards of excess width, then there will also be corresponding slots on the P0Bus that also may not be utilized.

Additionally, PowerMAX OS also provides support for connecting two P0Bus overlays together to form a larger P0Bus, by using a single Backplane P0 (BPP0) Bridge board. A configuration which uses the BPP0 board to attach two P0Bus overlays will have a total of 8, 9, or 10 available P0Bus slots. A maximum of 8 processor boards can be installed in a single cluster, regardless of whether more P0Bus slots are available. Because of hardware limitations on the sizes and physical address alignments of the windows that are created for allowing data to be passed through a BPP0 bridge board, there are some restrictions on the placement and numbering of SBCs in a cluster that has a P0Bus BPP0 board in it. These restrictions are described in Appendix A, "Backplane P0 Bridge Board Cluster ConfigurationBackplane P0 Bridge Board Cluster Configuration".

For instructions on installing the P0Bus overlay and the Backplane P0 Bridge board, see the subsection entitled "Installing the P0 overlay" located in Section 2, "Getting Started", of the *VGM5* or *VSS4 SBC User Guide*.

4.3.3. File Server Board Configuration

This section describes the procedure for configuring a SBC board as a File Server SBC in a closely-coupled system. In the event your SBC board was not already configured as a File Server SBC, or the configuration of the File Server SBC board appears to be cor-

rupted or lost, follow the procedures provided in this section to configure a SBC board as a File Server SBC.

Note that the board jumper settings have already been discussed in a previous section, **4.3.1. Board Jumpers on page 4-6**.

The user should also refer to the *SMon PowerPC Series SBCs Application Developer & Debugger User Guide* for additional information regarding **SMon** commands and features.

1. The File Server SBC has an implicit SBC Board ID value of 0, and this SBC must be installed in VME slot 1 in the rack.

2. Connect a terminal to Serial UART Port A/Console if one is not already connected.

3. The File Server SBC board should be set up to execute the SMon monitor after power-on or after reset. If the board is configured to execute the **fdiag** diagnostic program after power-up or reset, then this must first be modified as follows:

NOTE

In the following discussion, **<cr>** stands for hitting the **Return/Enter** key on the keyboard.

Power-up or reset the board and watch for the resulting input prompt:

a. If the:

SMon0>

prompt appears, skip the rest of this step and go to step #4 below.

b. If the:

```
To change any of this, hit any key...
```

text appears, then hit any key, and then type **X<cr>** to enter **SMon**. Skip the rest of this step and go to step #4 below.

c. If the:

fdiag0>

prompt appears after the initial power-on/reset information is displayed, then the board is currently setup to execute the **fdiag** diagnostic program instead of **SMon**. To change this, enter **config<cr>**, and then enter **<cr>** to step through this command's prompts/questions until the:

```
SMon boot enabled [Y]:
```

prompt appears.

Answer **Y<cr>** to this prompt.

Enter **<cr>** until the rest of the prompts/questions are completed for the

config command.

Now enter **reboot<cr>** to reboot the board into **SMon**.

d. If the:

```
To change any of this, hit any key...
```

text appears, then hit any key, and then type **X<cr>** to enter **SMon**.

4. Now that the SBC board executes **SMon** by default, make sure that the post script is not enabled. To disable execution of the post script, enter:

   config <cr>

   and enter **<cr>** to step through this command's prompts/questions until the:

   ```
   post script enabled [Y]:
   ```

   prompt appears. Answer **N <cr>** to this prompt.

   Enter **<cr>** until the rest of the prompts/questions are completed for the **config** command. Now enter **reboot<cr>** to reboot the board into **SMon**.

   If the:

   ```
   To change any of this, hit any key...
   ```

   text appears, then hit any key, and then type **X<cr>** to enter **SMon**.

5. The next step is to set up the networking addresses that the embedded Symbios Ethernet controller will use. To set up these parameters, type:

   smonconfig<cr>

   and then enter **<cr>** to step through other prompts until the Ethernet parameters appear:

   ```
   ETHERNET PARAMETERS:
   What is the board's serial number? [1014042]
   What should the Ethernet host address be? [129.134.30.26]
   What should the Ethernet target address be?
   [129.134.32.79]
   What should the Ethernet mask be? [255.255.255.0]
   What should the Ethernet gateway address be?
   [129.134.32.196]
   ```

   The `Ethernet target address` should be set to the IP address of this File Server SBC, as specified in the **/etc/hosts** file for the Symbios Ethernet interface.

   The `Ethernet host address` can be optionally set to the IP address of a host that might be used as a tftp host for this File Server SBC.

The `Ethernet gateway address` may be optionally set to a gateway system/node that is accessible from this File Server's Ethernet controller.

Note that the two remote IP addresses above should also be contained in the **/etc/hosts** file located on this File Server SBC.

Hit **<cr>** to step through all the other **smonconfig** command prompts until this command completes.

6. The File Server SBC may be optionally setup with a **SMon startup** script to load and execute the console processor, or the user may prefer to manually load and execute the console processor.

   a. If the user does NOT want to use a **SMon startup** script to automatically enter the console processor after reset/power-up, then enter **smonconfig<cr>** and answer **1<cr>** to the following prompt.

   b. If a **SMon startup** script is desired for entering the console processor after reset/power-up, then answer **2<cr>** to the prompt below:

```
The SMon ROM can be used in several ways:
(1) ROM-boot SMon Stand-alone
(2) ROM-boot SMon with startup script
Which one do you want? [1]: 2
```

   Continue to enter **<cr>** until the prompts for this command is completed. If you entered 1**<cr>** to the above prompt, then skip ahead to step #7 below.

   To create the **startup** script that will load and execute the console processor, enter **vi "startup"<cr>** and enter the following line as the only line in the **startup** script:

   scsidiskboot *n*   <== where '*n*' is equal to the SCSI ID of the boot disk device

NOTE

While **SMon** provides a subset of **vi** commands for editing the **startup** script, be careful when exiting the **vi** editor. Be sure to type **:q** when exiting **vi**, and not **:w**. Typing **:w** will cause the contents of the **startup** script to be lost and for all **SMon** configuration settings to be reset back to their factory default values.

For more background information on the **SMon startup** script, and on using the **SMon** built-in editor, see the *SMon PowerPC Series SBCs Application Developer & Debugger User Guide*, and also the *Power Hawk Series 700 Console Reference Manual*.

7. The procedure for setting up the File Server SBC is complete. Enter the **`SMon reboot<cr>`** command to reset the SBC board and have **`SMon`** initialize the SBC board and optionally enter the console processor mode.

4.3.4. Client Board Configuration

This section describes the procedure for configuring a SBC board as a client SBC in a closely-coupled system. In the event your SBC board was not already configured as a client SBC, or the configuration of the client board appears to be corrupted or lost, then follow the procedures provided in this section to configure the board as a client SBC.

The user should also refer to the *SMon PowerPC Series SBCs Application Developer & Debugger User Guide* for additional information regarding **`SMon`** commands and features.

Note that the board jumper settings have already been discussed in a previous section, **4.3.1. Board Jumpers on page 4-6**.

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 client SBC board should usually be set up to execute **`SMon`** after power-on or after reset. However, if the board was to execute the **`fdiag`** diagnostic program after power-up or reset, then this must first be modified as follows:

NOTE

In the following discussion, **`<cr>`** stands for hitting the **Return/Enter** key on the keyboard.

Power-up or reset the board and watch for the resulting input prompt:

a. If the:

`SMon0>`

prompt appears, skip the rest of this step and go to step #3 below.

b. If the:

`To change any of this, hit any key...`

text appears, then hit any key, and then type **`X<cr>`** to enter **`SMon`**. Skip the rest of this step and go to step #3 below.

c. If the:

`fdiag0>`

prompt appears after the initial power-on/reset information is displayed, then the board is currently setup to execute the **fdiag** diagnostic program instead of **SMon**. To change this, enter:

config<cr>, and then enter **<cr>**

to step through this command's prompts/questions until the:

```
SMon boot enabled [Y]:
```

prompt appears. Answer **Y<cr>** to this prompt. Enter **<cr>** until the rest of the prompts/questions are completed for the **config** command.

Now enter **reboot<cr>** to reboot the board into **SMon**.

If the:

```
To change any of this, hit any key...
```

text appears, then hit any key, and then type **X<cr>** to enter **SMon**.

3. Now that the SBC board executes **SMon** by default, make sure that the post script is not enabled. To disable execution of the post script, enter:

   config <cr>

   and enter **<cr>** to step through this command's prompts/questions until the:

   ```
   post script enabled [N]:
   ```

   prompt appears. Answer **N <cr>** to this prompt.

   Enter **<cr>** until the rest of the prompts/questions are completed for the **config** command.

   Now enter **reboot<cr>** to reboot the board into **SMon**.

   If the:

   ```
   To change any of this, hit any key...
   ```

   text appears, then hit any key, and then type **X<cr>** to enter **SMon**.

4. This step describes how to configure **SMon** so that it always executes the **startup** script after reset/power-up and how to set up the networking addresses that the embedded Symbios Ethernet controller will use. To set up these parameters, type **smonconfig<cr>** and answer **2<cr>** at the prompt below to change it to **SMon** with **startup** script.

   ```
   The SMon ROM can be used in several ways:
   (1) ROM-boot SMon Stand-alone
   (2) ROM-boot SMon with startup script
   Which one do you want? [1]:2<cr>
   ```

   Then hit **<cr>** to step through other prompts until the Ethernet parameters appear:

```
ETHERNET PARAMETERS:
What is the board's serial number? [1014042]
What should the Ethernet host address be? [129.134.30.26]
What should the Ethernet target address be? [129.134.32.79]
What should the Ethernet mask be? [255.255.255.0]
What should the Ethernet gateway address be? [129.134.32.196]
```

The `Ethernet host address` should be set to the IP address of the File Server SBC within this cluster. This IP address should be the same Ethernet address of the File Server that is located in the File Server's **/etc/hosts** file.

The `Ethernet target address` should be set to the IP address of this client SBC. This IP address should be the Symbios Ethernet address of this client SBC that is already in the File Server SBC's **/etc/hosts** file.

A gateway address MUST be specified if the File Server's ethernet interface resides on a different subnet from this client's ethernet interface.

Hit **<cr>** to step through all the other **smonconfig** parameter prompts until this command completes.

NOTE

If this client SBC is going to be configured as an embedded client, or if this client is going to be configured as a NFS client with only the P0Bus networking interface and no Symbios Ethernet networking interface, then setting up the ETHERNET PARAMETERS above is not required. However, even in these cases, these parameters should still be setup if the user plans to use a temporary Ethernet connection to use **tftp** to download the initial **SMon startup** script from the File Server SBC (see the next step #5 below for more details).

5. The client SBC's VME boot SMon **startup** script must be setup. This script provides the File Server SBC with access to this client SBC for the VME boot support. This script must get executed after every reset or power-up. To check to see if the **startup** script is already setup, enter:

   vi "startup"<cr>

   to examine the contents of the **startup** script.

NOTE

While **SMon** provides a subset of **vi** commands for editing the **startup** script, be careful when exiting the **vi** editor. Be sure to type **:q** when exiting **vi**, and not **:w**. Typing **:w** will cause the contents of the **startup** script to be lost and for all **SMon** configuration settings to be reset back to their factory default values.

The required contents of the client SBC **startup** script is shown below. A copy of this **startup** script may also be printed to stdout by using **vmebootconfig(1M)** with the **-P** option: "**vmebootconfig -P** *smon*".

```
------------------Startup Script --------------------

int sbc_id = REPLACE_ME;
wlle(fec00000, 80008810); wlle(fee00000, 801001);
wlle(fe801348, 0);  wlle(fe80134c, getmemsize());
wlle(fe801f74, (effc0000 + (sbc_id * 1000)));
wlle(fe801f70, 80f20000); wl(ffffe4, 0); wl(ffffe8, 0);
printf("Waiting for server download; polling at ffffc0\n");
wl(ffffc0, feedbeef); while (rl(ffffc0) == feedbeef) ;
if (rl(ffffe4) != 0 && rl(ffffe8) != 0)
 printf("Load image at %x:%x\n", rl(ffffe4), rl(ffffe8));
wb(ffeffe50, 80); netstart; if (rl(ffffc0) == beeffedd) {
 printf("Jumping to %x\n", rl(ffffe0)); g rl(ffffe0);
} if (rl(ffffc0) == beefc0de) {
 fp uf erase; fp uf rl(ffffe4) rl(ffffe8); wb(ffeffe50, 80);
} else if (rl(ffffc0) != deadbeef)
 printf("ERROR: unknown command: %x\n", rl(ffffc0));
wl(ffffc0, feedbeef);

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

NOTE

If the VME_VRAI_BASEADDR **cluster.profile** parameter is modified from its default value, then the 0xeffc0000 value in the fourth line of the above **Smon startup** script must be modified so that is matches the new VME_VRAI_BASEADDR parameter value. For example, if the VME_VRAI_BASEADDR is set to a non-default value of 0xa000, then the 0xeffc0000 value in the fourth line of the above script must be set to a value of 0xa0000000. See the section **Cluster-wide Parameters on page 4-19** for details on the VME_VRAI_BASEADDR parameter.

If the **startup** script appears as it does above, with REPLACE_ME set to the desired *sbc_id* value that is to be used for this client board, then type **:q<cr>** to exit the **vi** editor and go to step #6 below.

5a. Setting the *sbc_id* value -

Each client board must have a unique *sbc_id* value. This is the unique software SBC ID value that identifies this board within the cluster, and this is the SBC ID value that the File Server SBC uses to configure, build, probe and boot client SBCs. In general, the *sbc_id* value may be any number from 1 to 7 (the File Server SBC is

always 0), but based on the maximum amount of memory on any SBC in the cluster, this value may need to be reduced in range. The table below shows the valid *sbc_id* values for each VME_DRAM_WINDOW tunable value:

VME_DRAM_WINDOW	Maximum DRAM Size	sbc_id Values
1	64MB	1-7
2	128MB	1-7
3	256MB	1-7
4	512MB	1-3
5	1GB	1

NOTE

> If the cluster contains a Backplane P0 (BPP0) Bridge board with two P0Bus Overlay boards connected together, then the *sbc_id* value may need to be further restricted. If your cluster contains a BPP0 Bridge board, then refer to Appendix ABackplane P0 Bridge Board Cluster Configuration "Backplane P0 Bridge Board Cluster ConfigurationBackplane P0 Bridge Board Cluster Configuration" for information on additional *sbc_id* value restrictions.

If the *sbc_id* value needs to be modified, or if the **`startup`** script below is not set up or is incorrect, then type:

```
vi "startup" <cr>
```

in order to enter or modify the script as shown below. Note that usually only the REPLACE_ME field needs to be changed in the **`startup`** script previously shown above.

NOTE

> Be sure to enter "**`:q<cr>`**" to exit **`vi`** after this script has been edited.

For more background information on the **`SMon startup`** script, and on using the **`SMon`** built-in vi editor, see the *SMon PowerPC Series SBCs Application Developer & Debugger User Guide*, and the *Power Hawk Series 700 Console Reference Manual.*

5b. Downloading the **`SMon Startup`** Script -

The CCS **startup** script is somewhat lengthy, and it can be somewhat difficult to manually type in, especially if multiple client SBC boards need to be setup.

Therefore, as an alternative to manually typing in the entire **startup** script, this script may instead be downloaded from the File Server SBC and loaded into the **startup** script buffer via **tftp**. This method requires a Symbios Ethernet connection on the client SBC that will provide **tftp** access to the File Server SBC. Note that this Ethernet connection need only be temporarily installed, if this client SBC is not using the Symbios Ethernet interface after the client board has been configured. This method also requires the that client SBC's Ethernet host and target addresses have already been properly initialized back in step #4 (page 4-12).

To download the **startup** script contents from the File Server SBC, follow these steps:

a. If the **/tftpboot** directory does not already exist on the File Server SBC, generate it by:

   mkdir /tftpboot<cr>

b. On the File Server SBC, place a copy of the **Smon startup** script in the **/tftpboot** directory as follows:

   vmebootconfig -P *smon* **> /tftpboot/startup**

   (Note that this step is only required once, since all clients being initialized may use this same **startup** script file.)

c. The File Server's **tftp** server (**tftpd(1M)**) is not enabled by default. To enable the **tftp** service, run one of the following commands. Note that it is harmless to run this command even if the daemon is already enabled. Refer to the **tftpd(1M)** man page for more information on the **-s** secure option.

   To start **tftpd** in non-secure mode:

   /usr/etc/diskless.d/sys.conf/bin.d/enable_tftp

   To start **tftpd** in secure mode:

   /usr/etc/diskless.d/sys.conf/bin.d/enable_tftp -s

d. On the client SBC, issue the following **SMon** command to download the **startup** script from the File Server SBC via **tftp**, and to load the contents of the script into the **SMon startup** script buffer:

   loadEB "startup"<cr>

e. At this point, only the "REPLACE_ME" *sbc_id* value needs to be modified in the **startup** script. On the client SBC, enter:

   vi "startup"<cr>

   and change just the *sbc_id* value on the first line of the script,

and then type:

:q<cr>

to update the script and exit the **vi** editor.

6. This step describes how to configure **SMon** so that it always executes the **startup** script after reset or power-up. Type:

   smonconfig<cr>

   and answer: **2<cr>**

   at the prompt below so that **SMon** will execute the **startup** script after each power-cycle or reset:

   ```
   The SMon ROM can be used in several ways:
    (1) ROM-boot SMon Stand-alone
    (2) ROM-boot SMon with startup script
   Which one do you want? [1]: 2<cr>
   ```

7. The vmeboot client's hardware clock must be updated to match the date on the system designated as the File Server.

   Use the **SMon** '**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:

   ```
   SMon0> date<cr>
   Mar 7 01:02:05 1996
   ```

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

   date month day hour min sec year

   You must use a '**#**' prefix when entering decimal values. For example, to set the date to August 22, 1996 and the time to 14:55:30:

   ```
   SMon0> date 8 #22 #14 #55 #30 #96
   ```

8. Enter the **reboot** command to reset the SBC board and have **SMon** initialize and then execute the new **startup** script.

   reboot<cr>

   Then, the

   ```
   Waiting for server download; polling at 0xffffc0
   ```

   message should appear on the terminal screen if the **startup** script has been

set up properly. If the script has not been entered correctly, it may be necessary to reset the board and go back and re-edit the **startup** script using the **vi** editor.

9. After the File Server successfully configures, downloads and boots this client SBC, then this SBC's serial terminal should no longer be needed and it may be removed if the user so desires, or it may alternatively be used as the client board's console terminal when running under the PowerMAX OS.

   Similarly, the Symbios Ethernet connection may also be removed at this point, if the **SMon startup** script has been successfully downloaded, and if the Ethernet connection is no longer required.

4.4. Cluster Configuration

This section describes the steps for creating the environment on the File Server that is necessary for support of diskless client SBCs in greater detail than the "Installing the Cluster section on page 4-2.

The major topics described in this section are:

- Configuring the cluster.profile file (page 4-19) and client profile file (page 4-26)
- Creating and removing cluster configurations (page 4-34)

4.4.1. The Profile Files

Information about the File Server and each client in the cluster is specified in profile files. The administrator creates and updates these profile files and invokes **vmebootconfig(1M)** to create on the File Server the environment necessary for supporting a private virtual root directory, where a private boot image for each client may be configured, customized and built.

The **cluster.profile** file and **vmebootconfig(1M)** are also used to modify the File Server's kernel configuration so that it is configured to operate as a File Server in a closely-coupled system (CCS) environment. After the kernel configuration is modified, and prior to booting a diskless client SBC, the File Server's kernel MUST be rebuilt and the system must be rebooted. After the File Server is rebooted, it is then able to support the booting of diskless client SBCs that reside in the same cluster.

NOTE

When **vmebootconfig(1M)** initially modifies the File Server's kernel configuration, it will enable the '**sym_dma**' kernel module if the '**ncr**' kernel module is not currently enabled. These two modules are mutually exclusive, and the '**sym_dma**' module is not required if the '**ncr**' kernel module is already enabled. (The '**ncr**' or the '**sym_dma**' kernel modules are used for DMAing data across the P0Bus.)

Therefore, if the '**sym_dma**' kernel module is enabled in the File Server's initial closely-coupled kernel configuration and the system administrator decides to enable the '**ncr**' kernel module at a later point in time, then the '**sym_dma**' kernel module should also be disabled.

4.4.1.1. The cluster.profile File

If the **cluster.profile** file does not already exist in the **/etc/profiles** directory, then it must be created. A **cluster.profile** file can be created with the following command:

vmebootconfig -P *cluster* **> /etc/profiles/cluster.profile**

You should then edit this **cluster.profile** file to suit the particular needs of the cluster and File Server SBC. The file
"**/etc/profiles/cluster.profile.README**"
contains explanations of the various parameters set in the **cluster.profile** file.

There are two sections to the **cluster.profile** configuration file. The first section describes a closely-coupled cluster as a whole, and the second section has optional parameters that apply specifically to the File Server SBC.

Some parameters are optional, and some parameters are already set to the recommended default value.

4.4.1.1.1. Cluster-wide Parameters

The first section of the **cluster.profile** file contain the following cluster-wide related parameters.

VME_DRAM_WINDOW=2

This parameter specifies the maximum memory size that can be supported on any SBC in the cluster. It is used to determine the base I/O addresses used by VMEbus and P0 Bus masters to remotely access each board's DRAM (on-board memory). The window size must be greater than or equal to the largest DRAM installed on any single board computer in the cluster. This parameter is used to set the kernel tunable of the same name, and this corresponding tunable must have the same value on each SBC in the cluster. The value of this parameter affects the number of boards that can be supported in a cluster.

The valid values for this parameter and the corresponding sizes and number of boards (clients + File Server) supported are shown in the table below.

Parameter Value	Maximum DRAM Window Size	Maximum Number of Boards (Including File Server)
1	64MB	8
2	128MB	8
3	256MB	8
4	512MB	4
5	1GB	2
Note that care should be taken to set this parameter to an accurate value; this parameter cannot be easily modified after the client SBCs are configured.		

SBC_SLAVE_MMAP_MAXSZ=1

This parameter defines the largest possible Slave Mmap memory area that can be setup on any SBC in the cluster. This value is used to set the kernel tunable of the same name and this tunable value must be the same for each SBC in the cluster.

Note that this parameter defines the maximum size limit of the Slave Mmap area on any SBC in the cluster; the actual size for each SBC's Slave Mmap area is defined with the SBC_SLAVE_MMAP_SIZE parameter (see below for a description of this parameter).

The maximum Slave Mmap area size is always at least 4KB in size, where the first 4KB of memory is reserved for system-only usage.

NOTE: If this SBC_SLAVE_MMAP_MAXSZ parameter is left set to the default minimum a value of 1 (4KB), then no user-accessible Slave Mmap shared memory areas may be configured on ANY SBC in the cluster.

When the maximum Slave Mmap memory area is defined to be larger than 4KB, then this additional memory space may be optionally allocated on a per-SBC basis for shared memory usage.

The Slave Mmap area size defined may range from 4KB to 256MB, but the size may never be larger than one fourth of the amount of memory defined by the VME_DRAM_WINDOW parameter. When there is a Backplane P0 (BPP0) Bridge board installed in the cluster, then the value of this SBC_SLAVE_MMAP_MAXSZ parameter may need to be further reduced in size (refer to the Note below for information on the additional BPP0 restrictions).

This value is used to set the system tunable of the same name and must be the same value on each SBC in the cluster. The valid values for this tunable and the corresponding sizes (in parentheses) are shown in the table below.

1 (4KB)	5 (64KB)	9 (1Mb)	13 (16MB)	17 (256MB)
2 (8KB)	6 (128KB)	10 (2MB)	14 (32MB)	
3 (16KB)	7 (256KB)	11 (4MB)	15 (64MB)	
4 (32KB)	8 (512KB)	12 (8MB	16 (128MB)	

If Slave Mmap shared memory is to be configured on some or all of the SBCs in the cluster, then this parameter should be assigned to an appropriate value BEFORE configuring the client profile files; this parameter cannot be easily modified after the client SBCs are configured.

NOTE: When a Backplane P0 (BPP0) Bridge board is installed (see the P0_BPP0_SBC_ID parameter below), additional size restrictions may apply. In the following two configurations, the size defined by this SLAVE_MMAP_MAXSZ parameter may NOT be larger than 1/8 (normally it is 1/4) of the amount of memory defined by VME_DRAM_WINDOW:

Case 1: VME_DRAM_WINDOW = 3 (256B) and P0_BPP0_SBC_ID = 2

Case 2: VME_DRAM_WINDOW = 4 (512 MB) and P0_BPP0_SBC_ID = 1

VME_VRAI_BASEADDR=0xeffc

This parameter defines the upper 16 bits of the base address location on the VMEbus where a VME Remote Access Image (VRAI) of the PCI-to-VME64 Universe II bridge registers for each SBC (including the File Server SBC) are configured to reside.

This value is used to set the kernel tunable of the same name and tunable must be the same value for each SBC in the cluster.

The default value of this parameter is 0xeffc, the minimum value is 0xa000, and the maximum value is 0xfafc; however, the value for this parameter MUST fall within the range of the VME_A32_START and VME_A32_END kernel tunables. (The VME_A32_START kernel tunable has a default value of 0xc000, and the VME_A32_END kernel tunable has a default value of 0xfaff.)

The default value of 0xeffc (which translates to a VMEbus address value of 0xeffc0000) should not usually need modification, unless there is a conflict in VMEbus address usage with another VMEbus I/O device that must be loaded in this address range.

The VRAI area size for each SBC is 4KB. Since each SBC's VRAI area is placed in this range of VMEbus addresses, the total amount of VMEbus address space that is used for this purpose depends upon the maximum possible number of boards in the cluster.

For example, when a possible maximum of eight boards may be placed into the cluster, then:

8 * 4KB

bytes of VMEbus address space are reserved, and should not be used for any other purpose. For example, using the default VME_VRAI_BASEADDR parameter value of 0xeffc, the total VRAI image space for eight boards would occupy the following VMEbus address range:

0xeffc0000 - 0xeffc7fff

NOTE: The value assigned here must also match the value assigned in each client's **`SMon startup`** script. If VME_VRAI_BASEADDR is modified from its default value, then the **`SMON startup`** script for EACH client SBC must also be modified. See section "Client Board Configuration" (page 4-11) for information on modifying the **`SMon startup`** script due to a change in this parameter from its default value.

P0_BPP0_SBC_ID=0

This parameter MUST be set to zero unless the P0Bus is composed of two P0Bus overlays joined by a Backplane P0 (BPP0) Bridge board. (Run-time checks during system initialization time will be done to validate that no BPP0 bridge board is present in the cluster.)

This parameter is used to set the kernel tunable of the same name and this tunable value must be the same for each SBC in the cluster.

When a BPP0 board is present in the system, then this parameter MUST be set to a non-zero value of the smallest logical SBC board id that is located on the second P0Bus overlay. All boards located on the second P0Bus overlay must have logical SBC board id values that are equal to or greater than this tunable value, and all boards located on the first P0Bus overlay must have logical SBC board id values that are less than this tunable value. (Note that the lower slots in the VME cardcage are located on the first P0Bus overlay, and the higher slots in the VME cardcage are located on the second P0Bus overlay.)

The following are some of the guidelines for setting this parameter. For a more complete explanation on the setting of this parameter, refer to Appendix A, "Backplane P0 Bridge Board Cluster ConfigurationBackplane P0 Bridge Board Cluster Configuration.

When non-zero, the valid values for this tunable are determined by the setting of the VME_DRAM_WINDOW parameter that was previously described:

When VME_DRAM_WINDOW is set to value of 1, 2 or 3 (256MB or less), then the valid values for this tunable are: 2, 4 or 6. Note that If P0_BPP0_SBC_ID is set to 6, then the logical SBC id value of 4 cannot be used for any client SBC in the cluster.

When VME_DRAM_WINDOW is set to 4 (512MB), then the valid values for this tunable are: 1, 2 or 3.

When VME_DRAM_WINDOW is set to 5 (1GB), then the only valid value for this tunable is 1.

VME_IRQ1=n	VME_IRQ2=n
VME_IRQ3=n	VME_IRQ4=n
VME_IRQ5=n	VME_IRQ6=n
VME_IRQ7=n	

These 7 parameters are used to specify on which logical SBC ID the corresponding VME interrupt request level (IRQ) is enabled. Each parameter must be assigned a valid logical board id number.

For example, the assignment VME_IRQ3=1 specifies that VMEbus IRQ 3 is to be reserved and used exclusively by the client SBC whose logical board id is 1. (Note that the logical SBC ID value of 0 is always reserved for the File Server.)

These settings are used to set corresponding the kernel tunables VME_IRQ[1-7]_ENABLE, and the values for all of these tunables must be the same for all SBCs in the cluster.

Only one board in the cluster may enable/use a particular IRQ level, but any one board may enable/use from none, up to all of the possible VME IRQ levels.

CCS_NET=p0

This parameter defines the networking interface to be used to mount, across NFS, the diskless client's file systems that reside on the File Server. In addition, this interface may also be used for general purpose networking. It is required that each client be assigned an IP address in the **`/etc/hosts`** file for the chosen interface. The valid values are '**`p0`**' for the P0Bus networking interface and '**`eth`**' for Symbios ethernet networking interface.

See **Networking Hostname Naming Conventions on page 4-31** for guidelines in generating networking hostnames and see the P0_NET_IPADDR parameter description below for information on P0Bus IP address formulation.

OPT_NET=eth

This parameter is optional and defines a second network interface in addition to the CCS_NET interface.

This interface will be free of the NFS traffic inherent in the diskless implementation that is handled by the network assigned in CCS_NET above.

The valid choices are '**`p0`**' for the p0 bus networking interface if CCS_NET=eth; and '**`eth`**' for ethernet if CCS_NET=p0.

When the optional network is the P0Bus network ('**`p0`**'), it is required that each client is assigned an IP address for the p0bus interface in the **`/etc/hosts`** file. All the point-to-point connections will be created during the node system's boot.

When the optional network is Symbios Ethernet (**`eth`**), then each client node may independently chose to configure ethernet by adding an entry for the interface in the **`/etc/hosts`** file and assigning ETHER_SUBNETMASK in the client's profile file (see section "NFS Related Parameters" on page 4-29). If an entry does not exist in

/etc/hosts for the client's ethernet networking interface then it will not be configured, and a warning message will be output to inform the system administrator that this interface lacks a corresponding hostname entry.

See **Networking Hostname Naming Conventions on page 4-31** for guidelines in generating networking hostnames and see the P0_NET_IPADDR parameter description below for information on P0Bus IP address formulation.

P0_NET_IPADDR=192.168.1.1

When either of the parameters CCS_NET or OPT_NET are assigned the value **p0**, a P0Bus networking IP address must be defined. This should be an internet address, in decimal dot notation, of a unique subnet to be used by the cluster for P0 networking. The P0Bus internet address for each client in the cluster will be generated by adding the client's board id number to this address. This parameter is used to set the kernel tunables BUSNET_IPADDR_HI and BUSNET_IPADDR_LO.

P0Bus IP Address Formulation

When either of the parameters CCS_NET or OPT_NET are set to '**p0**'in the **cluster.profile** file, then the cluster will be configured with P0Bus networking.

In this case, a SBC's P0Bus networking IP address is formulated by adding the SBC's logical SBC ID value to the address defined for this P0_NET_IPADDR parameter.

For example, if P0_NET_IPADDR is set to a value of 192.168.1.1, the following IP addresses correspond to the File Server and clients that are assigned logical SBC ID values 1 and 2. (Note that the File Server's logical SBC ID is always 0.)

Logical SBC ID	P0Bus IP Address
0	192.168.1.1
1	192.168.1.2
2	192.168.1.3

4.4.1.1.2. File Server SBC Parameters

The second section of the **cluster.profile** file contains the following parameters that relate specifically to the File Server SBC, and NOT to the whole cluster. Note that these same parameters are also contained in the client profile file for defining these same parameter values for each client SBC (see the section "The Client Profile File" on page 4-26).

SBC_SLAVE_MMAP_SIZE=

This optional parameter is an index value, which defines the amount of local DRAM memory that will be mapped onto the P0bus for use by the Slave Mmap interface. Note that the first 4KB is always reserved for internal kernel use. This index parameter specifies the amount of space (MINUS 4KB) that may be used as the user-accessible Slave Mmap memory area.

For example, an index value of 5 reserves 64KB of physical memory, 4KB is for kernel use and 60KB may be used for shared memory.

Note that if this parameter is left unspecified (left blank), then a default 4KB size kernel-accessible Slave Mmap area will be setup. In this case, no user-accessible Slave Mmap memory area will be available on the File Server SBC.

This parameter value must be less than or equal to the value assigned to the parameter SBC_SLAVE_MMAP_MAXSZ in this **`cluster.profile`** file. This parameter value will be used to set the kernel tunable of the same name for the File Server's kernel.

The index values and corresponding sizes, in parenthesis, are shown below. Note that the specified size is the sum of the kernel (4KB) space, PLUS the shared memory space:

2 (8KB)	6 (128KB)	10 (2MB)	14 (32MB)
3 (16KB)	7 (256KB)	11 (4MB)	15 (64MB)
4 (32KB)	8 (512KB)	12 (8MB)	16 (128MB)
5 (64KB)	9 (1MB)	13 (16MB)	17 (256MB)

SBC_SLAVE_MMAP_START=

The optional SBC_SLAVE_MMAP_START parameter determines whether the Slave Mmap area is statically or dynamically allocated. When SBC_SLAVE_MMAP_START is set to zero or is not specified (left blank), then the Slave Mmap memory area is dynamically allocated during system initialization. This is the preferred allocation setting, unless a particular application requires **`shmbind(2)`** support for locally accessing this Slave Mmap area from the same SBC.

When SBC_SLAVE_MMAP_START is non-zero, then this indicates that the Slave MMAP area is statically allocated. In this case, SBC_SLAVE_MMAP_START must be set to a physical DRAM address value that is aligned on a boundary that is a multiple of the SBC_SLAVE_MMAP_SIZE parameter value.

This parameter value will be used to set the kernel tunable of the same name for the File Server's kernel.

When SBC_SLAVE_MMAP_START is non-zero, then the kernel will attempt to use the specified reserved memory area that must also be defined in the *res_sects[]* array of that SBC. During system initialization, the kernel will search the *res_sects[]* array and try to locate an entry that starts at the SBC_SLAVE_MMAP_START value, with a length equal to the SBC_SLAVE_MMAP_SIZE tunable value.

For example, if SBC_SLAVE_MMAP_START is set to 0x1400000 and SBC_SLAVE_MMAP_SIZE is set to a value of 9 (for a 1MB size), then the following *res_sects[]* entry would reserve that range of physical DRAM memory:

```
--------------------------------------------------------------
struct res_sect res_sects[] = {
   /* r_start, r_len, r_flags */
   { 0x1400000, 0x100000, 0 },  /* Slave Mmap area */
   { 0, 0, 0 }  /* This must be the last line, DO NOT change
it. */
};
--------------------------------------------------------------
```

NOTE On the first invocation of **vmebootconfig(1M)**, the File Server's **/etc/conf/pack.d/mm/space.c** *res_sects[]* array will be automatically updated with the appropriate Slave Mmap entry if this parameter is set to a non-zero value. After that point, the **mkvmebstrap(1M) -m** option may be used to modify the File Server's SBC_SLAVE_MMAP_SIZE and SBC_SLAVE_MMAP_START tunables. This **-m** option will also automatically modify the *res_sects[]* array, if needed.

4.4.1.2. 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. Note that all of the parameters located in a client profile file are specific to that one client SBC; all cluster-wide parameters are defined in the cluster.profile file.

You can use **vmebootconfig(1M)** to print out a starting template of a client profile with the "**-P** client" option. Note that the client profile file name should be equivalent to the client's hostname.

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

```
vmebootconfig -P client > /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 client SBC.

The parameters contained in a client profile file are described below. Some parameters are required, and some parameters are optional, depending upon the type of networking interface(s) being configured.

4.4.1.2.1. Required Parameters

The following parameters in the client profile file are required for all types of client SBCs.

BOARD_ID=

This parameter is the logical SBC ID for this client SBC. (Board id zero is reserved for the File Server.)

NOTE This parameter value MUST match the value assigned to the variable '*sbc_id*' in the board's **SMon startup** script. See the section "Client Board Configuration" on page 4-11 for details on the **SMon startup** script.

The *sbc_id* value allowed for this parameter usually ranges from 1 to 7. However, this range may be reduced due to the maximum DRAM memory size that is defined by the VME_DRAM_WINDOW parameter, which is defined in the **cluster.profile** file:

VME_DRAM_WINDOW	Maximum DRAM Size	*sbc_id* Values
1	64MB	1-7
2	128MB	1-7
3	256MB	1-7
4	512MB	1-3
5	1GB	1

NOTE When a P0Bus bridge board is installed, additional board id restrictions may apply. For more information, refer to the **P0_BPP0_SBC_ID** parameter documentation in the previous section "The cluster.profile File" on page 4-19, and also refer to Appendix A, "Backplane P0 Bridge Board Cluster ConfigurationBackplane P0 Bridge Board Cluster Configuration".

VROOT=

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

AUTOBOOT=

This parameter indicates whether or not this client should be booted/shutdown whenever the File Server is booted/shutdown.

This parameter may be set to '**y**' for yes. In this case this embedded or NFS client will be automatically booted when the File Server SBC boots up. If the client is a NFS client, then this client will also be automatically shutdown when ever the File Server SBC executes a shutdown operation. Note that the automatic shutdown support does not apply to embedded clients.

This parameter may be also set to '**n**' for no. In this case, this client SBC will not be automatically booted when the File Server SBC boots, nor will this client SBC be automatically shutdown when the File Server SBC executes a shutdown operation.

NOTE When this parameter is set to '**y**', then a hidden file named **.autoboot** will be created by **vmebootconfig(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 booted or shutdown by the File Server SBC whenever the File Server is booted or shutdown. This **.autoboot** file may be manually removed or created in the client's virtual root directory, as needed.

FLASHBOOT=

This parameter, which must be set to either 'y' or 'n', indicates whether or not this client should be booted from a boot image that resides in the client's User Flash, or from a boot image that resides on the File Server and is downloaded across the VMEBus into the client board's memory.

When this parameters is set to 'n' for no, the client will be booted from the client's boot image that resides on the File Server by downloading the client's boot image across the VMEBus.

When this parameter is set to 'y' for yes, then this indicates that the client should be booted from the boot image that resides in the client's User Flash.

It is recommended that this parameter be initially set to 'n'. After a working client vmeboot configuration and boot image have been created on the File Server and tested on the client, the user may then optionally choose to set this parameter to 'y' in order to enable User Flash booting.

Note that once this parameter is set to 'y', then the AUTOFLASH parameter must also be set to a valid value (see the AUTOFLASH parameter discussion below).

See Chapter 5 "Flash Boot System Administration" for more information about burning and booting from User Flash.

AUTOFLASH=

This parameter is ignored unless the FLASHBOOT parameter is set to 'y'. When FLASHBOOT=y, then this parameter becomes a required parameter that must be set to either 'y' or 'n'.

When FLASHBOOT=y and this parameter is also set to 'y', then the client's boot image that resides on the File Server will be automatically burned into the client SBC's User Flash whenever a new client boot image is generated.

When FLASHBOOT=y and this parameter is set to 'n', then the client's boot image will not ever be automatically burned into the client's User Flash.

Note that in this case, the user must manually cause the initial or new versions of the client's boot image to be burned into the client board's User Flash by using the appropriate **`mkvmebstrap(1M)`** or **`sbcboot(1M)`** options.

See Chapter 5 "Flash Boot System Administration" for more information about burning and booting from User Flash.

SYS_CONFIG=

This parameter defines the client SBC's system configuration. The valid values for this parameter are:

`emb` Embedded client. This configuration operates only in singleuser mode, has no networking support, and no swap space.

`nfs` NFS client. This configuration operates in multiuser mode, contains networking support, supports remote NFS swap space and remotely mounted (NFS) directories, including the File Server system directories (such as **`/bin`** and **`/etc`**).

BOOT_IFACE=vme

This parameter is set to '**`vme`**' for all client profile files that are created with the '**`vmebootconfig -P client`**' method. This parameter indicates that this client is a VME boot client, and as such, this parameter should NOT be modified by the user.

NOTE In order to create a Net boot client profile file, the **netbootconfig(1M) -P** option should be used to create a Net boot client profile file. See the "Netboot System Administration" chapter for details on configuring netboot clients.

4.4.1.2.2. NFS Related Parameters

The following client profile file parameters are only required for NFS clients (those clients with SYS_CONFIG set to '**nfs**'); clients with SYS_CONFIG set to '**emb**' may leave these parameters blank.

SWAP_SIZE=

This parameter defines the size, in megabytes, of a remote NFS swap space area. This swap space is implemented as a file (**dev/swap_file**) residing in the client's virtual root and accessed over NFS.

This parameter is recommended to be 1.5 times the size of the physical memory (DRAM) located on this client SBC.

ETHER_SUBNETMASK=

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

For example: 255.255.255.0.

For Symbios Ethernet networking to be configured on a client node, the following must be true:

- CCS_NET or OPT_NET must be set to '**eth**' in the **cluster.profile** file.
- This parameter must be set to a valid Ethernet subnet mask value.
- An entry in File Server's **/etc/hosts** file must exist with a hostname of **<client_profile_filename>-eth**. For example, if a client's client profile filename is '*wilma*', then a hostname of '*wilma-eth*' must exist in the **/etc/hosts file**. See section **Networking Hostname Naming Conventions on page 4-31** for a more complete description of CCS networking hostnames.

4.4.1.2.3. Shared Memory Parameters

The following two optional client profile parameters define the Slave Mmap memory area configuration for this client SBC.

If no user-accessible Slave Mmap shared memory area is required on this client SBC, then these two parameters may be left blank

SBC_SLAVE_MMAP_SIZE=

This optional parameter is an index value that defines the amount of local DRAM memory that will be mapped onto the P0bus for use by the Slave Mmap interface.

Note that the first 4KB is always reserved for internal kernel use. This index parameter specifies the amount of space (MINUS 4KB) that may be used as the user-accessible Slave Mmap memory area.

For example, an index value of 5 reserves 64KB of physical memory, 4KB is for kernel use and 60KB may be used for shared memory.

Note that if this optional parameter is left unspecified (left blank), then a default 4KB size kernel-accessible Slave Mmap area will be setup. In this case, no user-accessible Slave Mmap memory area will be available on this client SBC.

This parameter value must be less than or equal to the value assigned to the parameter SBC_SLAVE_MMAP_MAXSZ, which is defined in the **cluster.profile** file (in section "Cluster-wide Parameters" on page 4-19). This parameter value will be used to set the kernel tunable of the same name for the client's kernel.

The index values and corresponding sizes, in parenthesis, are shown below. Note that the specified size is the sum of the kernel (4KB) space, PLUS the shared memory space:

2 (8KB)	6 (128KB)	10 (2MB)	14 (32MB)
3 (16KB)	7 (256KB)	11 (4MB)	15 (64MB)
4 (32KB)	8 (512KB)	12 (8MB)	16 (128MB)
5 (64KB)	9 (1MB)	13 (16MB)	17 (256MB)

SBC_SLAVE_MMAP_START=

The SBC_SLAVE_MMAP_START parameter determines whether the Slave Mmap area is statically or dynamically allocated. When SBC_SLAVE_MMAP_START is set to zero, then the Slave Mmap memory area is dynamically allocated during system initialization. This is the preferred allocation setting, unless a particular application requires **shmbind(2)** support for accessing this Slave Mmap area on the local SBC.

When SBC_SLAVE_MMAP_START is non-zero, then this indicates that the Slave MMAP area is statically allocated. In this case, SBC_SLAVE_MMAP_START must be set to a physical DRAM address value that is aligned on a boundary that is a multiple of the SBC_SLAVE_MMAP_SIZE parameter value.

This parameter value will be used to set the kernel tunable of the same name for the client's kernel.

When SBC_SLAVE_MMAP_START is non-zero, then the kernel will attempt to use the reserved memory area that must also be defined in the *res_sects[]* array of that SBC. At system initialization time, the kernel will search the *res_sects[]* array and try to locate an entry that starts at the SBC_SLAVE_MMAP_START value, with a length equal to the SBC_SLAVE_MMAP_SIZE tunable value.

For example, is SBC_SLAVE_MMAP_START is set to 0x1400000 and SBC_SLAVE_MMAP_SIZE is set to a value of 9 (for a 1MB size), then the following *res_sects[]* entry would reserve that range of physical DRAM memory:

```
-------------------------------------------------------------------------------------------------

struct res_sect res_sects[] = {
    /* r_start, r_len, r_flags */
   { 0x1400000, 0x100000, 0 },  /* Slave Mmap area */
   { 0, 0, 0 }  /* This must be the last line, DO NOT change
it. */
};

-------------------------------------------------------------------------------------------------
```

NOTE On the first invocation of **vmebootconfig(1M)**, the client's **<virtual_root>/etc/conf/pack.d/mm/space.c** *res_sects[]* array will be automatically updated with the appropriate Slave Mmap entry if this parameter is set to a non-zero value.

After that point, the **mkvmebstrap(1M) -m** option may be used to modify the client's SBC_SLAVE_MMAP_SIZE and SBC_SLAVE_MMAP_START tunables. This **-m** option will also automatically modify the *res_sects[]* array, if needed.

4.4.1.3. Networking Hostname Naming Conventions

This section discusses the hostname name format that must be followed when adding P0Bus or Symbios Ethernet networking hostnames to the File Server's **/etc/hosts** file for boards that are within a CCS cluster.

Note that this hostname naming convention is required by the diskless utilities, and thus this convention must be followed in order to successfully configure networking support within a cluster.

Symbios Ethernet Hostnames

If either the CCS_NET or OPT_NET parameters in the **cluster.profile** file are set to '**eth**', then the hostname entries for the boards in the cluster should be one of the following formats:

If the board is a client SBC, then the hostname should be of the format:

<client_profile_filename>-eth

For example, if a client's profile filename is '*wilma*', then the corresponding Symbios Ethernet hostname that should be added to the **/etc/hosts** file should be:

wilma-eth

Note that the Symbios Ethernet hostnames that are added to the **/etc/hosts** file may also contain additional alias hostnames, if desired.

If the board is the file server SBC, then the ethernet hostname entry was added to the **/etc/hosts** file during the software installation phase, therefore, the File Server does not require any additional entries for its ethernet network interface. It is assumed that the File Server's nodename is the same as the Symbios ethernet networking interface name.

P0Bus Ethernet Hostnames

If either the CCS_NET or OPT_NET parameters in the **cluster.profile** file are set to '**p0**', then the hostname entries for the boards in the cluster should be of the format:

If the board is a client SBC, then the hostname should be of the format:

<client_profile_filename>-p0

For example, if a client's profile filename is '*wilma*', then the corresponding P0Bus networking hostname that should be added to the **/etc/hosts** file should be:

wilma-p0

If the board is the File Server SBC, then the hostname that should be added to the **/etc/hosts** file should be of the format:

<server_nodename>-p0

So for example, if the File Server's nodename is '*fred*', then the following entry should be added to the **/etc/hosts** file:

fred-p0

Note that the corresponding IP address that should be used for P0Bus networking entries in the **/etc/hosts** file depends upon the board's logical SBC ID value and the P0_NET_IPADDR **cluster.profile** parameter value. See "Cluster-wide Parameters" on page 4-19 for details on the P0Bus networking IP address values.

As is the case for Symbios Ethernet hostnames, the P0Bus networking hostnames that are added to the **/etc/hosts** file may also contain additional alias hostnames, if desired.

System Nodename:

In addition to the network interface entries described above, the system nodename must be added as an alias to the appropriate network interface. When both network interfaces are configured, it is the administrator's choice to which one should be used for the client's system nodename. The system nodename must be the same name used for the client profile filename.

Examples:

The examples entries below assume that the File Server named '*barney*' and the client named '*fred*' are members of a cluster with the following settings in the **/etc/profiles/cluster.profile** file:

```
CCS_NET=p0
OPT_NET=eth
```

1. Example File Server entries:

```
192.168.1.1      barney-p0
129.134.32.74    barney
```

   The entry for *barney* is assumed to have been added during the system installation phase. The barney-p0 entry must be added prior to invoking **vmebootconfig**.

2. Example client entries:

   Assuming client *fred's* profile (**/etc/profiles/fred**) contains the following setting:

   ```
   ETHERNET_SUBNETMASK=255.255.255.0
   ```

   Then, one of the two sets of example entries below must be added to the **/etc/hosts** file:

   ```
   192.168.1.2      fred-p0
   129.134.32.76    fred-eth fred
   ```

   In this example it is desired that the client's system nodename is assigned to the Symbios ethernet network interface, therefore, the system nodename (and client profile name) is added as an alias to the ethernet network interface entry.

   ```
   192.168.1.2      fred-p0  fred
   129.134.32.76    fred-eth
   ```

   In this example, it is desired that the client's system nodename is assigned to the p0Bus network interface, therefore, the system nodename (and client profile name) is added as an alias to the p0Bus network interface entry.

4.4.2. Node Configuration

The **vmebootconfig(1M)** tool is used to create, remove or update the diskless client configuration located on the File Server SBC. It is also used to modify the File Server SBC's kernel configuration to run in closely-coupled mode.

Prior to running this tool, configuration information must be specified in the **cluster.profile** and client profile file(s). (See "The cluster.profile File" (page 4-19) and "The Client Profile File" (page 4-26) for more information about setting up these profile files.)

For more details on running the **vmebootconfig(1M)** tool, see the **vmebootconfig(1M)** manual page available online.

Vmebootconfig(1M) gathers information from the various tables and stores this information into a ksh-loadable file, named **.client_profile**, under the client's virtual root directory. The **.client_profile** is used by **vmebootconfig(1M)**, by other configuration tools and by the client during system startup. It is accessible on the client SBC at **/.client_profile**.

Most of the tasks performed by **vmebootconfig(1M)** are geared toward configuring a diskless client; however, some configuration is also done for the File Server. When the File Server system is specified in the node list argument, options that are not applicable to the File Server are silently ignored.

Vmebootconfig(1M) appends a process progress report and run-time errors to the client-private log file,
"**/etc/clients/<client_profile_filename>.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 (**-R**) and update client (**-U**).

4.4.2.1. Creating and Removing a Client

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

For a diskless client:

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(4)** 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.

Modifies the client's kernel configuration.

For the File Server:

Modifies the File Server's kernel configuration to run in closely-coupled mode. (The File Server's kernel must be rebuilt and rebooted.)

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

For a diskless client:

Removes the virtual root directory.

Removes client's name from the **dfstab(4)** tables and executes an **unshare(1M)** of the virtual root directory.

For the File Server:

Removes the closely-coupled tunables from the File Server's kernel configuration. (The File Server's kernel must be rebuilt and rebooted.)

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 or add in support for one or more subsystems (see "Subsystem Support" on page 4-35 for more information).

Examples:

Create the diskless client configuration of all vmeboot clients with client profile files located in the **/etc/profiles** directory. Process at most three clients at the same time:

vmebootconfig -C -p3 *all*

Remove the configuration of client ‘*rosie*’. Send the output to stdout instead of to the client's log file.

vmebootconfig -R -t *rosie*

Update the virtual root directories of all the clients with vmeboot client profile files in the **/etc/profiles** directory. Process one client at a time:

vmebootconfig -U -v -p1 *all*

4.4.2.2. Subsystem Support

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

Subsystem support is added to a diskless client configuration using **vmebootconfig(1M)** options, when invoked in either the create or update mode.

Subsystem support is added to a client configuration via the **-a** option and removed using the **-r** option. For a list of the current subsystems supported see the **vmebootconfig(1M)** manual page or invoke **vmebootconfig(1M)** with the help option (**-h**).

Note that 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 using the **-v** option of **vmebootconfig(1M)**.

Example 1:

Create diskless client wilma's configuration and add subsystem support for remote message queues and remote semaphores (CCS_IPC) and for the frequency based scheduler closely-coupled support (CCS_FBS):

vmebootconfig -C -a CCS_FBS -a CCS_IPC *wilma*

Example 2:

Update the virtual roots of all the clients with client profile files in the /etc/profiles directory, and add the frequency based scheduler closely-coupled support. Process one client at a time:

vmebootconfig -U -v -p1 -a CCS_FBS *all*

Example 3:

Remove the frequency based scheduler closely-coupled support from the clients *wilma* and *fred*:

vmebootconfig -U -r CCS_FBS *wilma fred*

4.4.2.3. Slave Shared Memory Support

The File Server's and client's initial Slave Mmap shared memory configuration are configured through proper setup of the SLAVE_MMAP_SIZE and SLAVE_MMAP_START profile parameters, which are located in both the **/etc/profiles** client profile file(s), and the **cluster.profile** file (for the File Server).

However, the Slave Mmap shared memory configuration of the File Server or any client may be modified after the SBC's configuration has been initially created with the **vmebootconfig(1M)** create (**-C**) option, through the use of the **mkvmebstrap(1M)** utility.

See section "The cluster.profile File" on page 4-19 and section "The Client Profile File" on page 4-26 for more information on these parameters.

The **mkvmebstrap(1M)** is a utility that is generally used to generate the bootstrap image that is used in booting a diskless client in a CCS configuration, and to subsequently boot the client SBC from the File Server.

However, **mkvmebstrap(1M)** may also be used to modify some of the attributes of a client's or File Server's kernel configuration. This second capability of **mkvmebstrap(1M)** may be used to modify an already existing client's or the File Server's Slave Mmap shared memory kernel configuration.

Note that in order to use **mkvmebstrap(1M)** to modify a client's Slave Mmap shared memory configuration, the client's virtual root image must already have been created with a **vmebootconfig(1M)** create (**-C**) invocation.

The **-m** option of **mkvmebstrap(1M)** may be used to configure a physical memory area on one or more SBCs which can then be accessed by the other members of the cluster via the Slave Mmap shared memory interface.

The Slave Mmap memory area may be configured to be dynamically or statically allocated. A dynamic allocation is one in which the kernel dynamically allocates, at system initialization time, the physical memory for the Slave Mmap memory area. In a static memory allocation, the administrator specifies the starting address of the statically allocated physical Slave Mmap memory area.

The Slave Mmap memory areas of both the remote and local SBCs may be accessed through a process's address space via the **mmap(2)** interface. The **shmbind(2)** interface may also always be used to access a remote SBC's Slave Mmap memory area through the process's address space. However, note that **shmbind(2)** may only be used to access the local SBC's Slave Mmap memory area if it was statically allocated. See the *Power Hawk Series 700 Closely-Coupled Programming Guide* for more information on the Slave Mmap Shared memory (SMAP) interface.

When configuring a statically allocated Slave Mmap memory area, the starting physical memory address of the area and a size index value are specified as one comma-separated argument on the **-m** option:

mkvmebstrap ... -m *addr, size_index . . .*

The *size_index* of the Slave Mmap shared memory area is specified in a size index value, and this index value must be less than or equal to the index value that was specified for the SLAVE_MMAP_MAXSZ parameter, which is located in the **/etc/profiles**

/cluster.profile file. See section "Cluster-wide Parameters" on page 4-19 for a discussion of this parameter.

The valid size_index values are shown below:

2 (8KB)	6 (128KB)	10 (2MB)	14 (32MB)
3 (16KB)	7 (256KB)	11 (4MB)	15 (64MB)
4 (32KB)	8 (512KB)	12 (8MB)	16 (128MB)
5 (64KB)	9 (1MB)	13 (16MB)	17 (256MB)

Note that a Slave Mmap shared memory area index size value of 1 (4KB) is not valid. This is due to fact that the kernel requires and makes use of the first 4KB of the Slave Mmap area for internal system purposes. Therefore, no user-accessible shared memory area would exist for a 4KB Slave Mmap memory area size. Due to the kernel's use of the first 4KB of the memory area, note that the amount of user-accessible Slave Mmap shared memory is always the specified size MINUS 4KB. So for example, a size_index value of 8 (512KB), would yield an actual user-accessible Slave Mmap shared memory area size of 512KB - 4KB, or 508KB.

The statically allocated starting physical address, addr, is specified as a hexadecimal physical address value. Note that this address must be aligned on a boundary that is a multiple of the actual size that is indicated with the *size_index* value.

For example, to specify a physical memory area starting at address 0x01000000 and of size 64KB, one would specify:

```
mkvmebstrap ... -m 0x01000000,5 ...
```

When configuring a dynamically allocated Slave Mmap memory area, the starting address is specified as zero. For example, to specify a dynamically allocated Slave Mmap shared memory area of size 128KB, one would specify:

```
mkvmebstrap ... -m 0,6 ...
```

Only one Slave Mmap shared memory area is allowed per node; therefore, if there is an existing area allocated when a new area is requested, the existing configuration is first removed. To remove a reserved memory segment that was previously configured without adding another, the address and the size value fields are specified as zero. For example, to remove the regions reserved in the previous examples above:

```
-m 0,0
```

4.4.2.3.1. Static Memory Allocations

The allocated memory area is considered statically allocated when the administrator specifies the starting address of the contiguous physical memory (DRAM) that is to be reserved. A reasonable start address value to use is 0x02000000 (32MB boundary), if the board DRAM size is greater than or equal to 64MB.

Note that the starting physical address must be aligned on a boundary that is a multiple of the total size of the shared memory area (the size that includes the user-accessible Slave Mmap shared memory size, plus the 4KB kernel area).

Other reserved memory areas that are contained in the *res_sects[]* array of the SBC's **etc/conf/pack.d/mm/space.c** file must also be taken into account when selecting the start address, so that no overlapping areas are configured.

When the shared memory is configured to be statically allocated, then both **vmebootconfig(1M)** with the create (**-C**) option and **mkvmebstrap(1M)** with the **-m** option will automatically update the **<client_virtual_root>/etc/conf/mm/pack.d/space.c** *res_sects[]* array as needed.

Both **mmap(2)** and **shmbind(2)** operations are supported on remote and local Slave Mmap shared memory accesses when the memory is allocated statically.

Examples:

Replace the Slave Mmap shared memory area with a statically allocated area starting at 32MB and of total size 64KB (a user-accessible size of 60KB):

mkvmebstrap -m 0x02000000,5 *rosie*

Remove the currently configured statically allocated Slave Mmap shared memory area by specifying zero for the size index:

mkvmebstrap -m 0,0 *rosie*

4.4.2.3.2. Dynamic Memory Allocations

A dynamic memory allocation lets the operating system choose the starting address for the Slave Mmap memory area. Dynamic allocation is indicated by specifying zero for the starting address.

When DRAM is dynamically allocated, the **mmap(2)** interface may be used for accessing the local SBC's Slave Mmap shared memory area, as well as any remote SBC's Slave Mmap shared memory area. However, the **shmbind(2)** interface may only be used for accessing remote SBC Slave Mmap shared memory areas; the local SBC's shared memory area may NOT be accessed with **shmbind(2)**.

Examples:

Update client fred with a dynamically allocated Slave Mmap shared memory area of size 128KB (and a user-accessible shared memory area size of 124KB):

mkvmebstrap -m 0,6 *fred*

Remove the previously configured dynamically allocated Slave Mmap shared memory area on *fred* (by using a 0 size_index value):

mkvmebstrap -m 0,0 *fred*

4.4.2.4. System Tunables Modified

The following tunables may be set by **vmebootconfig(1M)**:

- using predefined settings,
- based on information provided by the user on the invocation line,
- based on information contained in the **/etc/profiles/cluster.profile** and vmeboot client profile files.

The current settings for these tunables may be displayed using the **-d** option of **mkvmebstrap**.

1. The following cluster-wide tunables are set to the same value for all SBCs in the cluster (except for VME_UNIV_SYSCON):

Tunable Name	Value
VME_CLOSELY_COUPLED	1
IGNORE_BUS_TIMEOUTS	1
VME_DRAM_WINDOW	<user-defined>
VME_VRAI_BASEADDR	<user-defined>
P0_BPP0_SBC_ID	<user-defined>
SBC_SLAVE_MMAP_MAXSZ	<user-defined>
VME_UNIV_SYSCON	0 (for all clients)
VME_UNIV_SYSCON	1 (only on File Server)

2. The following VME Interrupt Request Level (IRQ) tunables are set to different value for each SBC's kernel, with only one SBC's kernel enabling any single one of these tunables:

Tunable Name	Value
VME_IRQ1_ENABLE	<user-defined>
VME_IRQ2_ENABLE	<user-defined>
VME_IRQ3_ENABLE	<user-defined>
VME_IRQ4_ENABLE	<user-defined>
VME_IRQ5_ENABLE	<user-defined>
VME_IRQ6_ENABLE	<user-defined>
VME_IRQ7_ENABLE	<user-defined>

3. System tunables to support Slave Mmap Shared Memory. These values are unique to each SBC. Note that these Slave Mmap shared memory tunables may also be modified by **mkvmebstrap(1M)** during **-m** option processing:

Tunable Name	Value
SBC_SLAVE_MMAP_START	<user-defined>
SBC_SLAVE_MMAP_SIZE	<user-defined>

4. System tunables that are related to P0Bus Networking. If the P0Bus networking interface is not configured to be enabled in the cluster, then these tunables are not used. When the P0Bus networking interface is configured to be enabled for the cluster, then these tunables apply to all SBCs in the cluster.

   The following tunables are set to the same value for all SBC’s kernel configurations:

Tunable Name	Value
BUSNET_IPADDR_LO	<user-defined>
BUSNET_IPADDR_HI	<user-defined>

4.5. Customizing the Basic Configuration

This section discusses the following major topics dealing with customizing the basic client configuration:

- Modifying the Kernel Configuration (page 4-40)
- Custom Configuration Files (page 4-43)
- Modifying Profile Parameters (page 4-51
- Launching Applications (page 4-55)
  - Embedded Client (page 4-55)
  - NFS Client (page 4-55)

4.5.1. 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.vmeboot** and **modlist.emb.vmeboot** respectively, under the directory path **/usr/etc/diskless.d/sys.conf/kernel.d**. These template files should not be modified.

Note that, 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. Note that all the commands referenced below should be executed on the File Server system.

a. 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 **mkvmebstrap(1m)**. The advantage of this method is that the client's kernel configuration is recorded in a file that is utilized by **mkvmebstrap(1m)**. This allows the kernel to be easily re-created if there is a need to remove and recreate the client configuration.

b. Kernel modules may be manually configured or de-configured using options to **mkvmebstrap(1m)**.

c. All kernel configuration can be done using the **config(1M)** utility and then rebuilding the unix kernel.

d. 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. This method is recommended when modifying cluster configuration tunables that would otherwise require the rebuild of all of the kernels for the clients in a given cluster.

4.5.1.1. kernel.modlist.add

The **kernel.modlist.add** custom table is used by the boot image creating tool, **mkvmebstrap(1m)** for adding user-defined extensions to the standard kernel configuration of a client system. When **mkvmebstrap(1m)** is run, it compares the modification date of this file with that of the unix kernel. If **mkvmebstrap(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 the section on "Custom Configuration Files" on page 4-43.

4.5.1.2. mkvmebstrap

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

4.5.1.3. 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. Note that 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:

a. Use the Rebuild/Static menu from within **config(1m)** to build a new unix kernel and then invoke **mkvmebstrap(1m)**. **mkvmebstrap(1m)** will find the boot image out-of-date compared to the newly built unix file and will automatically build a new boot image.

b. Use **mkvmebstrap(1m)** and specify "unix" on the rebuild option (**-r**).

4.5.1.4. 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.

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

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

4.5.2. 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.

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** path. 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 customization 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 copy non-system files to a client's virtual root directory (nfs clients only)

The files installed under the **custom.conf** directory in a client's configuration 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 **vmebootconfig(1M)** must be specified when the client configuration is being removed to prevent these files from being deleted.

When a client is configured using **vmebootconfig(1M)**, a directory is created specifically for that client under the **/etc/clients** directory. The client's custom configuration files are installed under a 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 VME booted embedded or NFS client, then the **<client_dir>** directory name will be of the format:

<client_profile_filename>_<board_id>.vme

For example, if the VME boot client's profile name was *wilma*, and the BOARD_ID parameter for *wilma* is set to 2, then the **<client_dir>** directory name, located in the **/etc/clients** directory would be:

wilma_2.vme

The files in these client-private directories are initially shared such that a change to one of these files will affect all the clients in the cluster.

NOTE

Note that if the server is also supporting loosely-coupled clients, then changes to the **custom.conf** files also affect the server's loosely-coupled clients that may also be sharing these files.

The tools, **mkprivate** and **mkshared**, under each client's private **custom.conf** directory are available to change the state of a custom file from shared to private, or from private to shared, respectively. Before creating a new version, **mkshared** will save the current version to a file named **<customfile>.old** and **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>_<boardid>.vme/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>_<boardid>.vme \
   /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 VME boot client named '*wilma*' (with a BOARD_ID of 1):

```
cd /etc/clients/wilma_1.vme/custom.conf
./mkprivate K00client
vi K00client
```

And to share the **K00client** script previously made private:

```
cd /etc/clients/wilma_1.vme/custom.conf
./mkshared K00client
vi K00client
```

Changes to the customization files are processed the next time the boot image generating utility, **mkvmebstrap(1m)**, is invoked. If **mkvmebstrap(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.

Since a full set of the client custom configuration files exist in each client's private **custom.conf** directory, either as a private file or as a link to the shared client configuration file, **mkvmebstrap(1M)** uses the custom configuration files in each client's private directory when building and configuring a specific client's boot image object.

Note that when a subsystem is configured via the node configuration tool **vmebootconfig(1m)**, the tool may generate a client-private version of the customization files to add support required for that subsystem. Before modifying the client-shared versions, verify that a client-private version does not already exist. If a client-private version already exists, make the changes to that file, as the client-shared versions will be ignored for this client.

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

4.5.2.1. 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 **mkvmebstrap(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 edited under the **/usr/etc/diskless.d /cluster.conf/custom.conf/nfs|emb** directories if the changes are to apply globally to all clients. If the changes are to apply to only one client, then the scripts should be edited under the client's private **/etc/clients/<client_dir>/custom.conf** directory, using the **mkprivate** tool to create a private copy of the file, if it is still linked to the shared version of the file.

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.

4.5.2.2. 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 on-line 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_rootpath>/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.

Note that 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 **inittab** 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 edited under the **/usr/etc/diskless.d/cluster.conf/custom.conf/nfs|emb** directory if the changes apply globally to all clients. If the changes are to apply to only one client, then the these tables should be edited under the client's private **/etc/clients/<client_dir>/custom.conf** directory, using the **mkprivate** tool to create a private copy of the file, if it is still linked to the shared version of the file.

4.5.2.3. 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)** manual page for rules on modifying this table.

Note that 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 the other customization files, these tables may be edited under the **/usr/etc/diskless.d/cluster.conf/custom.conf/nfs** directory if the changes apply globally to all clients. If the changes are to apply to only one client, then the these tables should be edited under the client's private **/etc/clients/<client_dir>/custom.conf** directory, using the **mkprivate** tool to create a private version of the file, if it is still linked to the shared version of the file.

4.5.2.4. 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 4-40.

Changes to this file are processed the next time **mkvmebstrap(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 **mkvmebstrap(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 edited under the **/usr/etc/diskless.d/cluster.conf/custom.conf/nfs|emb** directory if the changes apply globally to all clients. If the changes are to apply to only one client, then the these tables should be edited under the client's private **/etc/clients/<client_dir>/custom.conf** directory, using the **mkprivate** tool to create a private version of the file, if it is still linked to the shared version of the file.

4.5.2.5. memfs.files.add Table

When the **mkvmebstrap(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.vmeboot** and **filelist.nfs.vmeboot** for NFS clients and **devlist.emb.vmeboot** and **filelist.emb.vmeboot** for embedded clients under the **/usr/etc /diskless.d/sys.conf/memfs.d** directory. This set of files <u>should</u> <u>not</u> be modified by the user.

Note that additional files may be added to the memory-based file system via the **memfs.files.add** table located under the **/usr/etc/diskless.d /custom.conf/nfs|emb** directory. Guidelines for adding entries to this table are included as comments at the top 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.

Note that, 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 <u>should</u> <u>not</u> 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.

Also note that 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, since pages are brought into DRAM memory from flash only when referenced.

Changes to the **memfs.files.add** file are processed the next time **mkvmebstrap(1m)** is invoked. A new **memfs.cpio** 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 a 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>
```

4.5.2.6. vroot.files.add Table

The **vroot.files.add** 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 **mkvmebstrap(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 clients (the embedded clients are unable to access their virtual root on the File Server system). This table is processed by **mkvmebstrap(1M)** whenever this table has been modified since the last invocation of **mkvmebstrap(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 **vmebootconfig(1M)**.
- This file table can be setup to have **mkvmebstrap(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' vroot: **/users**, **/dev**, **/etc**, **/tmp**, or **/var**. Note that

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.

Examples:

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 **mkvmebstrap(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
```

every time that **mkvmebstrap(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 **vroots.files.add** table file may be edited under the **/usr/etc/diskless.d/cluster.conf/custom.conf/nfs** directory if the changes are to apply globally to all NFS clients. If the changes are to apply to only one client, then the scripts should be edited under the client's private **/etc/clients/<client_dir>/custom.conf** directory, using the **mkprivate** tool to create a private copy of the file, if it is still linked to the shared version of the file.

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 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 -
(**mkvmebstrap -B -r unix <client_profile_filename>**).

4.5.3. Modifying Profile Parameters

Once a diskless configuration has been established, it is often necessary to modify some aspect of the configuration. A client's configuration is generated based on the parameter settings in both the cluster.profile and a client's profile file.

Most parameters in the **cluster.profile** file may be changed via the **-u** option of **mkvmebstrap**. For those parameters not supported to be modified via this option, all the client configurations must be removed, the change applied to the **cluster.profile** and the configurations then recreated.

Most parameters in a client profile may not be changed after a client is configured. With few exceptions, the client configuration must be removed, the client profile modified and then the client configuration recreated.

The next two sections explain in detail how to modify parameters in the cluster and client profile files.

4.5.3.1. Cluster.profile File

Changes to the **/etc/profiles/cluster.profile** file may affect every member of the cluster and require that system tunables and the client's profile file(s) be modified. Changes must be applied to all the members of the cluster at the same time. Booting a client when these parameters are not identical for all clients in a given cluster, may result in bad system behavior, including panics and hangs on the File Server.

Refer to the section "Node Configuration" using **vmebootconfig(1M)** on page 4-33 for more information on creating and removing a cluster and the online manual page for **vmebootconfig(1M)**.

The following techniques can be used to modify settings in the cluster.profile file after the cluster has been configured:

1. Use the **mkvmebstrap -u** option to modify the cluster-wide parameters contained in the **cluster.profile** file.

   In this method, **mkvmebstrap(1M)** will update the profiles and the

kernel tunables and then build a new bootstrap for all clients. In the case of the VME_IRQ tunables, only the affected board's kernels are rebuilt.

Except for VME_IRQ parameter changes that do not affect the File Server, the File Server's kernel must also be manually rebuilt and subsequently rebooted before the **cluster.profile** modifications will take affect. Note that in these cases, the File Server's kernel should be rebuilt and rebooted BEFORE attempting to reboot the clients with their newly updated bootstrap images.

The **cluster.profile** parameter name and the new value, separated by an equal sign (with no spaces), should be specified as the value on the **-u** option. For example:

```
mkvmebstrap -u VME_VRAI_BASEADDR=0x20000
```

The following parameters may be specified with the **-u** option:

VME_VRAI_BASEADDR

Changing this parameter also requires modifying the **SMon startup** script. The startup script MUST be modified before any client may be booted with a new updated bootstrap image. See section "Client Board Configuration" on page 4-11 for details on modifying the **SMon startup** script.

Note that the File Server's kernel must also be rebuilt and rebooted AFTER the client's **SMon startup** scripts have been modified, and subsequently executed by **SMon**, but before any clients have been booted with the new bootstrap image.

VME_DRAM_WINDOW
SBC_SLAVE_MMAP_MAXSZ

These two parameters apply to all SBCs in the cluster. The File Server's kernel must also be rebuilt and rebooted. Note that modifying the SBC_SLAVE_MMAP_MAXSZ parameter changes the values that are allowed for the SBC_SLAVE_MMAP_SIZE parameter. Therefore, it may be necessary to modify the SBC_SLAVE_MMAP_SIZE parameter in some client profile files, and possibly in the **cluster.profile** for the File Server. Note that this modification may be done with the **mkvmebstrap -m** option. See section "Slave Shared Memory Support" on page 4-36 for details on the use of this option.

P0_BPP0_SBC_ID

This parameter applies to all SBCs in the cluster. The File Server's kernel must also be rebuilt and rebooted, before any client is rebooted with its new bootstrap image. Note that modifying this parameter may also require changing one or more client's logical *SBC_ID* parameters in their client profile file(s), and the logical SBC_ID in their corresponding **SMon startup** script(s).

Also note that modifying this parameter may reduce or increase the allowed maximum value for the SBC_SLAVE_MMAP_MAXSZ parameter.

P0_NET_IPADDR

This parameter applies to all SBCs in the cluster. The File Server's kernel must also be rebuilt and rebooted, before any client is rebooted with its new bootstrap image. Note that modifying this parameter will require also changing all the <node>-p0 entries in the **/etc/hosts** file.

VME_IRQ [1-7]

A change to one of the seven IRQ tunables affects at most two systems; the system that previously had the IRQ enabled and the system that is now enabling the IRQ. For these parameters, only the systems affected need to be updated and rebooted. If it is necessary to reboot the File Server system, it should be rebooted first, before rebooting the clients.

2. Remove the configuration of all the clients in the cluster and recreate the cluster environment:

   a. Perform a shutdown on all the diskless client systems in the cluster.

   b. Remove the configuration of all the clients in the cluster using **vmebootconfig(1M)** and specifying the all argument and the **-R** and **-s** options, for example:

      vmebootconfig -R -s *all*

   c. Change the desired parameter(s) in the **/etc/profiles/cluster.profile**.

   d. Recreate all the client configurations using **vmebootconfig(1M)**. Specify the all argument and the **-C** options to create all the configurations in the cluster, for example:

      vmebootconfig -C -p2 *all*

   e. Rebuild the unix kernel and boot images of each client using **mkvmebstrap(1M)**.

   f. Rebuild the File Server's kernel using **idbuild(1M)**.

   g. Reboot the File Server of the cluster.

   h. Boot the clients in the cluster.

3. Use the **mkvmebstrap(1M) -m** option to modify the File Server's Slave Mmap parameters, SBC_SLAVE_MMAP_START and SBC_SLAVE_MMAP_SIZE.

   These parameters are used to configure Slave Mmap shared memory specifically for the File Server SBC. These shared memory parameters are modifiable with the **-m** option of **mkvmebstrap**. See section "Slave Shared Memory Support" on page 4-36 for details on the use of this **mkvmebstrap** option.

Note that the rest of the parameters located in the **cluster.profile** file may NOT be modified with the **mkvmebstrap -u** option. For these other parameters, the entire cluster should be removed and reconfigured with the new parameter values. See the previous method #2 in this section for details on this approach.

4.5.3.2. Modifying Client Profile Settings

To modify most of the parameter values in a client profile, a client configuration must be removed and reconfigured. The client should always first be shutdown before modifying any of the parameters discussed below.

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

1. Remove the current client configuration for *wilma*:

   `vmebootconfig -R -s` *wilma*

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

   `vi /etc/profiles/wilma`

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

   `vmebootconfig -C` *wilma*

The client profile file parameters that MAY be modified without the need to remove and reconfigure the client are described below:

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 boot up and shutdown support. See the section "The Profile Files" on page 4-18 for more details on the AUTOBOOT parameter.

SBC_SLAVE_MMAP_START
SBC_SLAVE_MMAP_SIZE

These parameters are used to configure Slave Mmap shared memory. These shared memory parameters may be modified with the **`-m`** option of **`mkvmebstrap`**. See section "Slave Shared Memory Support" on page 4-36 for details on the use of this mkvmebstrap option.

SWAP_SIZE

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 client is currently up and running.

FLASHBOOT
AUTOFLASH

These two parameters may be directly modified in a client' client profile file. The new values for these parameters will be used on the very next invocation

of **mkvmebstrap(1M)** that involves the booting of this client. These parameters control the automatic burning and booting from a client's User Flash. See the section "The Client Profile File" on page 4-26 and also Chapter 5, "Flash Boot System Administration" for more information about burning and booting from User Flash.

4.5.4. Launching Applications

Following are descriptions of launching applications for:

- Embedded Client
- NFS Client

4.5.4.1. Launching an Application (Embedded Client)

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 "memfs.files.add Table" on page 4-48 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 command 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 4-43 above for more information about the **memfs.files.add** table and the **S25client rc** script.

4.5.4.2. Launching an Application (NFS Client)

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 ROM, this is not the case, since the memfs file system remains in flash ROM 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_rootpath>/etc	**/etc**
<virtual_rootpath>/var	**/var**
<virtual_rootpath>/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 added to the virtual root via the **vroot.files.add** custom file.

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 4-43 above for more information about editing the **vfstab** file.

4.6. Booting and Shutdown

This section deals with the following major topics pertaining to booting and shutdown:

- "The Boot Image" on page 4-57
- "Booting Options" on page 4-58
- "Creating the Boot Image" on page 4-60
- "VME Booting" on page 4-61

- "Net Booting" on page 4-62
- "Flash Booting" on page 4-62
- "Verifying Boot Status" on page 4-62
- "Shutting Down the Client" on page 4-63

4.6.1. 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, **/usr/etc/diskless.d/sys.conf/bin.d/bstrap.makefile**, is used by **mkvmebstrap(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 **mkvmebstrap(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 **mkvmebstrap** 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 listed below in Table 4-1. These files are explained in detail in "Custom Configuration Files" on page 4-43.

.

Table 4-1. 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)
	S25client (embedded client configurations only)

A diskless client's boot image is created under **etc/conf/cf.d** in the client's virtual root directory.

4.6.2. Booting Options

The booting method of a diskless client is determined by parameters in the **/etc/profiles** client profile file. This section discusses three of the client profile parameters, BOOT_IFACE, AUTOBOOT and FLASHBOOT.

The BOOT_IFACE parameter specifies the interface used to download the boot image and initiate the boot sequence on the diskless client:

- When the client profile file has initially created with the **vmebootconfig(1M) -P** client method, then the BOOT_IFACE parameter in the resulting client's profile file will be set to a value of '**vme**'.
- When the client profile file has initially created with the **netbootconfig(1M) -P** method, then the BOOT_IFACE parameter in the resulting client's profile file will be set to a value of '**net**'.

The setting of the BOOT_IFACE parameter also affects whether the client can be automatically restarted from the File Server. When BOOT_IFACE is set to, '**vme**' then the use of the VMEbus as the boot interface facilitates the autoboot capability by allowing the File Server to remotely reset the client SBC from across the VMEbus, and to subsequently initiate a new download and boot sequence.

For Net boot clients, the AUTOBOOT client profile parameter determines whether the client should be automatically shutdown when the File Server is shutdown.

For VME boot clients, the AUTOBOOT client profile parameter determines whether the client should be automatically booted/shutdown when the File Server is booted/shutdown.

The following is a description of how the BOOT_IFACE, AUTOBOOT and FLASHBOOT parameters specified in the **/etc/profiles** client profile file affect the boot process. Note that the FLASHBOOT parameter is only present in the vmeboot client profile file.

BOOT_IFACE=net, AUTOBOOT=n

The ethernet interface is used to download the boot image. The boot sequence will not be initiated from the File Server SBC, because there is no means to access the control registers of the client from the File Server. Instead, **SMon** command(s) must be executed on the client in order to download the client's boot image.

After the download is completed, the client may either a) boot from DRAM or b) burn the boot image into flash and then boot from flash.

Because the File Server cannot initiate a boot directly, the AUTOBOOT parameter has no effect on the booting of a netboot client. Also, the File Server SBC will not attempt to shutdown the netboot client during File Server shutdown processing since the AUTOBOOT parameter is set to '**n**'.

BOOT_IFACE=net, AUTOBOOT=y

This case is the same as the previous description, except for the fact that if the SYS_CONFIG client profile parameter is set to '**nfs**', then when the File Server SBC is in the process of shutting down, it will attempt to remotely shutdown the netboot client.

As was previously mentioned, because the File Server cannot initiate a boot sequence directly, the AUTOBOOT parameter has no effect on the booting of netboot client, even when AUTOBOOT is set to '**y**'.

BOOT_IFACE=vme, AUTOBOOT=y, FLASHBOOT=n

The VMEbus is used to download the image and initiate the boot sequence. Because the AUTOBOOT parameter set to '**y**', the client is automatically downloaded and booted when the File Server SBC is booted. The client may also be manually booted from either DRAM or FLASH by invoking **mkvmebstrap(1m)** on the File Server SBC with the appropriate boot options.

If the SYS_CONFIG client profile parameter is set to '**nfs**', then when the File Server SBC is in the process of shutting down, it will attempt to remotely shutdown the vmeboot client.

BOOT_IFACE=vme, AUTOBOOT=n, FLASHBOOT=n

The VMEbus is used to download the image and initiate the boot sequence. Because the AUTOBOOT parameter is set to '**n**', the client must be manually booted from either DRAM or User Flash by invoking **mkvmebstrap(1m)** on the File Server SBC with the appropriate boot options.

The File Server SBC will not attempt to shutdown the client SBC when the File Server SBC is shutting itself down.

BOOT_IFACE=vme, AUTOBOOT=y, FLASHBOOT=y

The boot sequence is initiated over the VMEbus. The boot loader is executed from User Flash and causes the kernel to be decompressed and loaded into DRAM.

Execution then is initiated in that kernel. Because the AUTOBOOT parameter is set to '**y**', the client is automatically booted from its User Flash when the File Server SBC is booted.

Additionally, if the AUTOFLASH client parameter is set to 'y', then any newly created client boot images that are created on the File Server, will be automatically downloaded into the client's memory and burned into the client's User Flash, before that client is subsequently booted from User Flash.

The client may also be manually booted from User Flash by invoking **mkvmebstrap(1m)** on the File Server with the appropriate boot options.

BOOT_IFACE=vme, AUTOBOOT=n, FLASHBOOT=y

The boot sequence is initiated over the VMEbus. The boot loader is executed from the client's board User Flash which causes the kernel to be decompressed and loaded into DRAM. Execution is then initiated in that kernel. Because the AUTOBOOT parameter is set to '**n**', the client must be manually booted from User Flash by invoking **mkvmebstrap(1m)** on the File Server SBC with the appropriate boot options. If the AUTOFLASH client parameter is set to 'y', then any newly created client boot images on the File Server will be automatically downloaded into the client's memory and burned into the client's User Flash before that client is subsequently booted from User Flash.

If there are client systems that are configured to autoboot, a part of the File Server's system initialization is to start a background boot process for each client system that is configured to autoboot. An administrator should not issue commands that would initiate a manual boot for clients that are in the process of autobooting. It is likely that this would cause a failure of the client boot process.

The autoboot process may also cause the boot image to be rebuilt prior to downloading of the client SBC. The boot image is rebuilt only when one of the dependencies of the boot image cause the current boot image to be out of date, or if the boot image does not currently exist.

In the case where the AUTOBOOT=y and FLASHBOOT=y if the boot image is found to be out of date when the File Server is booted, then the boot image will be automatically rebuilt on the File Server and then the client will be automatically booted from its own User Flash. However, the File Server will also automatically download the newly created boot image and burn this new image into the client's User Flash before the client is booted, only if the AUTOFLASH parameter is set to 'y'. If the AUTOFLASH parameter is set to 'n', then the client will still be automatically booted, but it will be booted by using an older version of the boot image that was not updated in the client's User Flash.

Any client which is configured to autoboot and which is also an NFS client, will be automatically shutdown when the File Server is shutdown.

4.6.3. Creating the Boot Image

The **mkvmebstrap(1m)** tool is used to build the boot image. This tool gathers information from the client's profile file(s) located in the **/etc/profiles** directory. See the online manual page for this command. Some example uses follow. Note that building a boot image is resource-intensive. When creating the boot image of multiple cli-

ents in the same invocation, use the **-p** option of **mkvmebstrap(1m)** to limit the number of client boot images which are simultaneously processed.

Examples:

Example 1.

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

mkvmebstrap -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.

mkvmebstrap -r unix -b kdb *wilma fred*

Example 3.

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

mkvmebstrap -p 3 -k kdb -k -rtc *all*

4.6.4. VME Booting

VME booting is enabled when the client is configured with the BOOT_IFACE client profile parameter set to '**vme**' (see "Booting Options" on page 4-58). In this case, **mkvmebstrap(1M)** may be used to VME boot a diskless client.

Invoked with the **-B** option, **mkvmebstrap(1M)** can be used to both update/create the boot image and boot the client(s) in the cluster. **Mkvmebstrap(1M)** must be executed on the File Server SBC. It calls on **sbcboot(1M)** to perform the boot operations.

Once the boot image has been created, the **sbcboot(1M)** tool may be executed on the File Server SBC of the cluster to boot a single client. To boot a client, **sbcboot(1M)** may be executed on the File Server SBC where the **boot_image_file sbcboot(1M)** parameter specifies the client's boot image (i.e **<vroot_dir>/etc/conf/cf.d/unix.bstrap**).

VME boot clients depend on the File Server for the creation and storage of the boot image. Once booted, clients configured with NFS support continue to rely on the File Server for accessing their system files across NFS. Clients configured as embedded, once up and running, do not depend on any other system.

Examples:

Example 1.

Update the boot image and boot all the clients in the cluster. Limit the number of clients processed in parallel to 2.

mkvmebstrap -B -p2 *all*

Example 2.

Update the boot image and boot the one client with a client profile filename of *betty*.

mkvmebstrap -B *betty*

4.6.5. Net Booting

Net booting is supported when the client is configured with "BOOT_IFACE=net" as the boot interface client profile parameter (see"Booting Options" on page 4-58).

For more information about netboot clients and loosely-coupled configurations, see Chapter 3 "Netboot System Administration.

4.6.6. Flash Booting

Flash booting is supported in both netboot client (loosely-coupled) and vmeboot client (closely-coupled) configurations.

For more information about Flash Booting, see Chapter 5, "Flash Boot System Administration".

4.6.7. 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:

a. **rlogin(1)** or **telnet(1)** from the File Server or remote File Server specifying the client's hostname.

b. attach a terminal to the console serial port and login.

You can also use the **ping(1m)** command and specify the client's hostname to verify that the network interface is running. Note, however, that 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 follows:

```
# 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.

4.6.8. Shutting Down the Client

If a client is configured with NFS and the **.autoboot** hidden file exists in the client's virtual root directory, then the client will automatically be shutdown whenever the File Server of the cluster is brought down.

A client configured with NFS can also be manually shutdown from the File Server using the **rsh(1)** command and specifying the client's hostname.

For example, if the hostname used for the client is *wilma*, the following **shutdown(1M)** command would bring the system wilma to **init state 0** immediately.

rsh *wilma* **/sbin/shutdown -g0 -y -i0**

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

5
Flash Boot System Administration

5.1. Introduction 1-1
5.2. User Flash Hardware Characteristics 1-2
5.3. Booting a Netbootable Client from Flash. 1-2
5.4. Burning a Netboot Client's User Flash. 1-3
5.5. Burning and Booting from Flash for VMEBus Bootable Clients 1-4

5
Flash Boot System Administration

5.1. Introduction

Power Hawk Series 700 systems support an optional User Flash memory device. This is a read-mostly memory-mapped device that behaves like normal memory, at least for reads, and whose contents are not lost across system power cycles. At the time of this writing, the available User Flash memory sizes for series 700 systems range from 8 to 64 megabytes. On the write function side, all Power Hawk Series 700 systems support a set of **SMon** commands that implement bulk erasure and reprogramming of User Flash with any desired set of user data that can be made to fit onto the device.

For clients with the User Flash option, the client's boot image can be stored into its User Flash, and then the client can be configured to be booted henceforth from that device. This change 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 development, each client would continue to be booted up using the netboot or the VMEBus boot sequence that is appropriate for that client. When converted to User Flash booting, the netboot or VMEBus bootup sequence will become slightly modified to avoid the actual copying of a boot image down from a server, replacing that step with booting the boot image that is already on that client's User Flash. All the other steps of the netboot or VMEBus boot sequence are performed as before. Therefore, User Flash booting should not be thought of as a totally separate function from netbooting or VMEBus booting, but thought of instead as a minor parameter or adjustment to the standard netboot or VMEBus boot sequence that must always occur.

Any boot image which is netbootable or VMEbus bootable can be put into User Flash and booted from there. No special preparation or treatment of the boot image is needed. Any technique that the user finds satisfactory for copying a working boot image into the User Flash will work.

There are several advantages to converting a project over to User Flash. First, it gives a client some independence from a server. For embedded clients especially, it is possible to develop a boot image that makes no reference to a server at all, resulting in a client that can be placed into a standalone configuration during the deployed phase. Second, bootup times are faster when booting from User Flash as there is no boot image download occurring. 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 server. Third, a Flashed boot image makes better use of client memory, as the root filesystem continues to reside in User Flash after booting, thus freeing up the memory that would otherwise have had to be used to implement a memory-resident root filesystem.

5.2. User Flash Hardware Characteristics

The User Flash on Power Hawk Series 700 boards is what is called a paged flash or a banked flash. The typical User Flash is 64 megabytes. However, the processor physical address space has only a 128 kilobyte wide window through which the User Flash contents can been seen or modified. Therefore the User Flash is conceptually broken up into a series of 128 kilobyte-sized 'banks' or 'page'; only one of these banks can be mapped into the physical address space at any one time.

The User Flash 128 kilobyte window begins at physical address **0xfff8000**.

The byte-wide register that selects which page of the User Flash is to be mapped is located at physical address **0xffeffe50**. The sign bit of the value written to this register must be set to enable the mapping; the lower seven bits is the numerical id of the User Flash page that is to be mapped in.

5.3. Booting a Netbootable Client from Flash

Assume at this point that a client and a server have been completely set up and that a useful boot image has somehow already been burned into the client's User Flash. Further assume that the client is a netboot client, because the hand-boot technique shown here is most useful for that configuration. The example below shows how that client is booted via **SMon** commands entered in at the **SMon** prompt:

```
SMon> wb ffeffe50 80
SMon> g fff8000
```

The first command **wb**, forces the first page of the User Flash to be mapped into the physical address space. The second command jumps to the first byte of that mapped page. This jump begins the PowerMAX OS boot sequence.

If these two commands are put into the netboot client's **SMon startup** script, then this client will autoboot from User Flash every time the client is reset or is powered up.

If a client is an embedded system, that is, one that after booting does not attempt to communicate with the server across the network or VMEBus, then moving its boot image to User Flash cuts the only required connection the client had with its server - that of fetching a boot image at boot time. That client now can be disassociated from its server and moved to a remote site.

For VMEBus bootable clients, it is possible to do these same steps by hand and it is possible to replace the **SMon startup** script that VMEBus bootable clients use as well, but none of this is necessary to do nor even desirable. The standard **SMon** VMEBus "**startup**" script already has these two commands embedded in it, executed as desired on command from the server.

5.4. Burning a Netboot Client's User Flash

A netbootable client, before being converted to flashboot, typically has a **SMon startup** script which is executed whenever that client powers up or is reset. This script will contain **SMon** commands that look similar to the following example and whose job is to netboot that client. Note that the user, at an **SMon** prompt on some client, can enter these commands anytime to initiate a normal netboot - a user does not need to power cycle a client and have them run from a **startup** script in order to be able to use them:

```
SMon> load "miles.bstrap"
SMon> g
```

or,

```
SMon> tftpboot "miles.bstrap"
```

These commands load a boot image from the server across the Ethernet cable via the **tftp** protocol, then jumps to the first byte of that boot image to begin the PowerMAX OS boot sequence. The above example assumes that the client's name is '*miles*' and that this client's boot image has already been built and installed on the server system under the **/tftpboot** directory with the **mknetbstrap(1M)** utility.

To instead burn a boot image into User Flash, the operator has to type in modified versions of the above **SMon** commands. Below is an example of such a command sequence, intermixed with the screen output that they produce:

```
SMon> fp uf erase
erasing USER flash: /

SMon> load "miles.bstrap" 1400000
Received 1713842 bytes in 12.2 seconds.
loaded miles.bstrap at 1400000

SMon> fp uf 1400000 #1713842
```

The **SMon** output shown above displays two numbers - **1400000** and **17132842**. These two numbers must be specified as arguments to the third **SMon** command shown above. Note the pound sign (**#**) typed as part of the final argument; this tells **SMon** that the argument is in decimal notation.

To summarize, the process of burning User Flash consists of 1) loading the client's boot image into the client's global memory, 2) erasing the User Flash, and 3) copying the boot image in memory to the User Flash.

Note that, when loading up a boot image into memory, the user must take care that the image is not loaded into any memory location that **SMon** uses. The load address shown above, **1400000**, is believed to be safe.

Hand burning works best for converting netbootable clients to User Flash boot. For VME-Bus bootable clients, the **SMon** commands described here are already embedded in the standard **SMon** "**startup**" script for VMEBus clients that should be already running on the client. This burn command sequence is triggered from the server, by the user or automatically by software running on the server, whenever that is desired or necessary.

5.5. Burning and Booting from Flash for VMEBus Bootable Clients

Given a working VMEBus bootable client configuration, the user can convert that configuration to boot from User Flash by setting to 'y' the following parameters in the client's profile file:

```
FLASHBOOT=y
AUTOFLASH=y
```

(**/etc/profiles/miles** would be the name of the profile file on the server for a client named '*miles*').

With these lines in place, flashburning and flashbooting will occur automatically as part of the normal sequence of operations. Whenever the server tells the client to boot, it will order that client to boot from its User Flash. Whenever the client is told to boot and the copy of the boot image in its User Flash is outdated, then the server will burn the latest boot image into the client's User Flash before ordering it to reboot.

If the user does not want a client's User Flash to be automatically burned with new boot images as needed, then that user should set AUTOFLASH=n in the client's profile. In that case, reboot commands issued to the client will continue to use whatever boot image was last placed into the User Flash.

To force a new boot image to be burned into client '*miles*':

mkvmebstrap -F *miles*

To force client '*miles*' to execute the boot image that is in its User Flash, without consideration as to whether or not what is in the User Flash is the latest and the greatest:

mkvmebstrap -X *miles*

And finally, the normal client boot command sequence, shown below, will boot the client with whatever method is specified by the FLASHBOOT and AUTOFLASH variable values that are in its profile. If FLASHBOOT=n, then this sequence does a normal VMEBus download and boot. If FLASHBOOT=y and AUTOFLASH=n, then this sequence will force the client to boot from whatever boot image is already in its User Flash - irrespective of whether that image is the latest and greatest or not. If FLASHBOOT=y and AUTOFLASH=y, then every attempt to boot the client will first burn, if necessary, a fresh boot image into the client's User Flash before the client is made to execute the image.

mkvmebstrap -B *miles*

The **sbcboot** command is the actual low level command that burns flashes or causes clients to execute what is in their User Flash. All the above methods eventually result in calls to **sbcboot**, executed at the appropriate times with the appropriate options.

To burn the User Flash of the client in **vme bus slot #2**, execute something like the following commands on the server:

```
TARGET=/dev/vmebus/target2
FILE=/vroots/miles/etc/conf/cf.d/unix.bstrap
sbcboot -F $TARGET $FILE
```

The **/dev/vmebus/target2** argument must be replaced with the device name of the slot that the desired client is in. The '**/vroots/...**' argument must be replaced with the filename containing the boot image that is to be downloaded and burned into that client.

To force the client in **vme bus slot #2** to boot itself from User Flash, enter on the server command line:

```
sbcboot -X /dev/vmebus/target2
```

NOTE

Burning User Flash of necessity causes the targeted client to go through a hard reset as part of the burn process. Therefore, a user should not attempt to burn the User Flash while a client is busy doing useful work.

6
Modifying VME Space Allocation

6.1. Overview 1-1
6.2. Default VME Configuration 1-1
6.3. Reasons to Modify Defaults 1-2
6.4. Limitations 1-3
6.5. Changing The Default VME Configuration 1-3
6.5.1. VME A32 Window 1-3
6.5.2. Closely-Coupled VME A32 Window Considerations 1-4
6.6. Example Configuration 1-4

6
Modifying VME Space Allocation

6.1. Overview

Material is this chapter is applicable to closely-coupled systems only.

On Power Hawk Series 700 platforms, accesses to VME space that are issued by the processor are accomplished through a special range of processor physical addresses. The hardware on these Power Hawk platforms translates this range of processor physical addresses into PCI bus addresses that fall into the PCI Memory Space range on the PCI Bus. Additional hardware on Power Hawk Series 700 platforms is set up to translate these PCI Memory Space addresses into VMEbus addresses which the hardware will place upon the VMEbus.

6.2. Default VME Configuration

The Power Hawk Series 700 platforms define a 2GB minus 48MB space that maps processor bus read and write cycles into PCI memory space. This area is used to allow master programmed I/O access to

- PMC/PCI devices
- VME A32 devices
- the PCI-to-VME Universe VME Remote Access Image (VRAI) area
- the PCI-to-PCI (P0) upstream window areas for accessing the P0Bus

Note that the VRAI area and the upstream P0Bus windows are both used specifically for closely-coupled configurations.

The user configurable VME A32 memory space area resides between processor addresses 0xA0000000 and 0xFCAFFFFF, which map directly to the same PCI memory space addresses ranges, 0xA0000000 to 0xFCAFFFFF.

One set of kernel tunables define this range of PCI Memory space addresses that may be used for VMEbus A32 space accesses.

Any PCI memory space not mapped to the VMEbus is available for use by PMC/PCI add-on cards which may be attached to the Power Hawk's PMC expansion slot, and also by the upstream P0Bus window areas.

The default configuration for processor to PCI to VME address translations on Power Hawk Series 700 platforms is shown in Table 6-1.

Table 6-1. Default Processor/PCI/VME Configuration

Processor Address	PCI Address	VME Address	VME Type	Window Size	Description/Tunables
0x80000000 0xBFFFFFFF	0x80000000 0xBFFFFFFF			0x40000000 1GB	PCI/PMC devices and P0Bus upstream windows - No VME Tunables
0xC0000000 0xFCAFFFFF	0xC0000000 0xFCAFFFFF	0xC0000000 0xFCAFFFFF	A32	0x3CB00000 971MB	VME_A32_START VME_A32_END
0xFCC00000 0xFCFEFFFF	0xFCC00000 0xFCFEFFFF	0xC00000 0xFEFFFF	A24	0x003F0000 4MB minus 64KB	`Fixed Mapping` No Tunables
0xFCFF0000 0xFCFFFFFF	0xFCFF0000 0xFCFFFFFF	0x0000 0xFFFF	A16	0x00010000 64KB	Fixed Mapping No Tunables

6.3. Reasons to Modify Defaults

There are several reasons why a system administrator may want to modify the default VME A32 address space configuration on Power Hawk Series 700 platforms. Some possible reasons are listed below:

- There are one or more VME devices configured in the system that have a large amount of on-board memory and/or require that their VME space be configured on certain address boundaries. For example, an A32 VME device might contain 1GB of on-board memory. It would not be possible to configure this device into the system using the default VME configuration address space (since it has a size of 971MB).

- Although PCI devices can be configured to respond to PCI I/O Space addresses, PCI devices can also be configured to respond to PCI Memory Space addresses. Therefore, it is possible that there may be a mix of PCI and VME devices in the system that both want to share the available PCI Memory Space.

  The P0Bus upstream windows also reside in PCI Memory space, and the size of these upstream windows can be rather large if the SBC_SLAVE_MMAP_MAXSZ parameter is set to one of the larger index values. (See "Cluster-wide Parameters" on page 4-19 for details on this **`cluster.profile`** parameter.)

  In these situations, it may be desirable to reduce the amount of VME A32 address space within PCI Memory Space in order to allow more of the PCI Memory Space to be used for PCI upstream windows and other PCI devices.

6.4. Limitations

The Power Hawk Series 700 platform hardware and PowerMAX OS software place certain restrictions on the configuration of VME address space and its accessibility from the processor's point of view. The restrictions on configuring VME space on Power Hawk Series 700 platforms under PowerMAX OS are:

- The total range of PCI Memory space is 0x80000000 to 0xFCFFFFFF, for a size of 0x7D000000 (2GB minus 48MB).
- The location and size of the A16 and A24 VME windows shown in Table 6-1 may not be modified.
- The allowed range of VME A32 addresses is from 0xA0000000 to 0xFCAFFFFF, for a total size of 0x5CB00000 (approximately 1.45 GB).
- If one or more PCI devices have been configured to respond to addresses within the PCI Memory Space range, then accesses to this PCI address space must be coordinated between VME and PCI devices, such that the VME windows and PCI device ranges do not overlap. Note that this configuration coordination is handled automatically by the kernel at system initialization time.

6.5. Changing The Default VME Configuration

Within the bounds of the restrictions that were mentioned in the previous section on Limitations, the system administrator may modify certain system defaults involving the VME to PCI configuration, and the placement of VME A32 window for A32 devices. This configuration of VME A32 space on Power Hawk platform may be accomplished by modifying the following tunables via the **config(1M)** utility:

VME_A32_START
VME_A32_END

As stated earlier, any available PCI Memory space not mapped to the VMEbus is available for add-on PMC/PCI devices.

6.5.1. VME A32 Window

The VME_A32_START and VME_A32_END tunables define the standard programmed I/O master window interface to VME A32 space. These tunables define processor local bus addresses which, when accessed through read/write cycles on the local processor, generate corresponding read/write cycles on the VMEbus in VME A32 space.

Note that this window cannot be disabled.

6.5.2. Closely-Coupled VME A32 Window Considerations

When configuring your VME A32 space, you must define the VME_VRAI_BASEADDR cluster.profile parameter so that it falls completely within the VME A32 window space.

This space is used to hold a set of 4KB sized PCI-to-VME64 Universe bridge hardware register images, where there is one image for each SBC in the cluster. Therefore, the total amount of VRAI space taken up by all of the SBC's VRAI images depends upon the maximum number of SBCs allowed in the cluster.

For example, when a possible maximum of eight boards may be placed into the cluster, then:

8 * 4KB

bytes of VMEbus address space are reserved, and this space should not be used for any other purpose, even when less than eight SBCs are physically present in the rack, since the kernel initialization code uses this area to 'probe' for the existence of other SBCs in the cluster.

Note that if the VME A32 window is moved such that the VME_VRAI_BASEADDR parameter also requires modification, then the **`SMon startup`** script for each client SBC must also be modified. (See "Cluster-wide Parameters" on page 4-19 for details on this **`cluster.profile`** parameter.)

6.6. Example Configuration

This section provides an example of a non-default VME A32 space configuration. In this example, an A32 VME memory module with 1GB of on-board memory is being installed. The module is strapped to respond to addresses `0xA0000000` to `0xDFFFFFF`. This example further assumes that there are no other VME devices on the VMEbus.

By modifying the tunable:

VME_A32_START to 0xA000

then the new VME A32 window area is now large enough to accommodate the 1GB memory module.

The default value for the VME_A32_END tunable (0xFCAF) does not need to change, and therefore the closely-coupled VRAI images that must reside in this A32 region (see "Closely-Coupled VME A32 Window Considerations" section above) may remain set to their usual default location.

The new PCI and VME A32 space configuration is shown below:

Processor Address	PCI Address	VME Address	VME Type	Window Size	Description/Tunables
0x80000000 0x9FFFFFFF	0x80000000 0x9FFFFFFF			0x20000000 512MB	PCI/PMC devices No VME Tunables
0xA0000000 0xFCAFFFFF	0xA0000000 0xFCAFFFFF	0xA0000000 0xFCAFFFFF	A32	0x5CB00000 1.448GB	VME_A32_START VME_A32_END
0xFCC00000 0xFCFEFFFF	0xFCC00000 0xFCFEFFFF	0xC00000 0xFEFFFF	A24	0x003F0000 4MB - 64KB	`Fixed Mapping` No Tunables
0xFCFF0000 0xFCFFFFFF	0xFCFF0000 0xFCFFFFFF	0x0000 0xFFFF	A16	0x00010000 64KB	Fixed Mapping No Tunables

7
Debugging Tools

7.1. System Debugging Tools 1-1
7.2. kdb 1-2
7.3. crash 1-2
7.4. savecore 1-3
7.5. sbcmon 1-3

7
Debugging Tools

7.1. System Debugging Tools

This chapter 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. Tools covered in this include the following:

- kdb
- crash
- savecore
- sbcmon

In a closely-coupled system architecture, members of a cluster, referred to as vmeboot clients, are connected to a common VME backplane and PCI-to-PCI (P0) bus. Each client's memory is accessible to the File Server and to other clients via hardware mappings in both PCI-to-PCI (P0) and VME space. This inter-SBC remote memory access is provided through the sbc device driver interface (**sbc(7)**). A vmeboot client is configured via a vmeboot client profile in the **/etc/profiles** directory. For more information on vmeboot clients, see Chapter 4 "VME Boot System Administration".

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 3 "Netboot System Administration".

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

vmeboot and netboot clients:

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.

vmeboot clients only:

b. create a system dump using **savecore(1m)** or **sbcmon(1m)** and then examine the system dump with **crash(1m)**.

c. examine the client system directly from the File Server 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 the system to drop into the board's **SMon** firmware.

7.2. 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** command, which is not available on clients, **kdb** operates without any differences from a normal system when executing on a client. A terminal should be connected to the console terminal port on the client being debugged in order to use **kdb**.

The default client kernel configuration does not have **kdb** support. The **kdb** kernel modules may be configured into the client's kernel via the **-k** option and the system may be programmed to stop in **kdb** via the **-b** option. Both these options are supported by the boot image generating tools **mkvmebstrap(1m)** for closely-coupled and **mknetbstrap(1m)** for loosely-coupled.

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

7.3. crash

The **crash(1m)** utility may be run from the file server system to examine the memory of a client board in the local cluster. **Crash(1m)** may also be invoked on a diskless client to examine the memory of another member of the cluster. In the case where **crash** is invoked on a diskless client, rather than on the file server, steps must be taken to gain read access to the unix file of the client whose memory is to be probed.

The **crash** utility may be run from the file server system to examine the memory of a client board in the local cluster as follows:

```
crash -d /dev/target<1-7> -n <VROOT>/etc/conf/cf.d/unix
```

For example, the following command may be used to run **crash** on a vmeboot client that has a **BOARD_ID** client profile parameter value of 1, and whose **VROOT** client profile parameter is set to **/vroots/wilma**:

```
crash -d /dev/target1 -n /vroots/wilma/etc/conf/cf.d/unix
```

7.4. savecore

The **savecore(1m)** system utility is used to save a memory image of the system after the system has crashed. **Savecore(1m)** supports the '**-t**' option to identify the client system that should be saved. This option allows **savecore(1m)** to be run on the file server, to create a crash file from the memory of a diskless client. **Savecore(1m)** saves the core image in the file **vmcore.n** and the namelist in **unix.n** under the specified directory. The trailing ``**.n**" in the pathnames is replaced by a number which is incremented every time **savecore(1m)** is run in that directory.

The **savecore(1m)** utility may be run from the file server system to create a system memory dump of a vmeboot client board in the same cluster. The following series of commands would be used to save a memory image and then analyze that image. The system dump created by **savecore(1m)** under the directory **<dirname>** and given the numerical suffix **<n>** may then be examined using crash.

```
savecore -f -t <board_id> <dirname> <VROOT>/etc/conf/cf.d/unix

cd <dirname>

crash -c <n>
```

For example, if vmeboot client '*wilma*' has a **BOARD_ID** parameter value of 2 and a **VROOT** parameter value of **/vroot/wilma** in it's client profile file, then the following commands would be used, assuming that no system dump already exits in the '**client_savecores**' directory:

```
savecore -f -t 2 /client_savecores /vroot/wilma/etc/conf/cf.d/unix
cd /client_savecores
crash -c1
```

7.5. sbcmon

sbcmon(1m) is a utility provided by the diskless package. It enables the host system to detect when another board in the cluster panics. When a panic is discovered, the **savecore(1m)** utility may be used to capture a dump file for future crash analysis or to shutdown the local system. This utility is documented in the **sbcmon(1m)** online man page provided in the diskless package.

A
Backplane P0 Bridge Board Cluster Configuration

This Appendix describes the various cluster configurations that are supported when a Backplane P0 (BPP0) Bridge Board is present. This Appendix also describes the additional limitations that are associated with some of the BPP0 configurations.

As is the case for clusters configured without a BPP0 bridge board, the following points are also true for BPP0 clusters:

- The server SBC board must be placed in the first (lowest numbered) slot in the rack, and the server's logical SBC board ID is always 0.
- For client SBCs, the logical SBC ID value does NOT indicate the physical slot number in the rack where that board is located. The local SBC ID value may be selected independently of the slot where the board is placed. However, the SBC ID value MUST follow the P0 overlay restrictions that are discussed below under the description of the P0_BPP0_SBC_ID parameter.

The are two **`/etc/profiles/cluster/cluster.profile`** parameters that determine which BPP0 cluster configurations are possible. These two parameters are also discussed in detail in section "Cluster-wide Parameters" on page 4-19.

P0_BPP0_SBC_ID

This parameter defines the lowest possible logical SBC ID value that may be used on the second (higher numbered slots) P0 overlay. All SBCs located in the second P0 overlay must have a logical SBC ID value that is greater than or equal to this parameter. All SBCs located in the first (lower numbered slots) P0 overlay must have logical SBC ID values that are less than this tunable.

A **P0_BPP0_SBC_ID** parameter value of zero indicates that the system has **NOT** been configured to accommodate a BPP0 bridge board. If a BPP0 is probed for and found on the P0Bus by the server's kernel during system initialization, then warning messages will be output to the console and the BPP0 bridge and closely-coupled support will **NOT** be enabled. Therefore, when a BPP0 is physically present in the cluster, this **P0_BPP0_SBC_ID** parameter **MUST** be set to a non-zero value even when there are

currently no SBCs located on the second overlay.

VME_DRAM_WINDOW

This parameter is an index value that defines the largest amount of DRAM that is located on any SBC in the cluster.

Given the above parameters, there are basically three types of valid BPP0 cluster configurations. These are:

1. **VME_DRAM_WINDOW** size <= 256MB
2. **VME_DRAM_WINDOW** size = 512MB
3. **VME_DRAM_WINDOW** size = 1GB

These three configuration types are described in more detail below.

1. Configurations with a VME_DRAM_WINDOW size <= 256MB

When **VME_DRAM_WINDOW** is set to an index value of 1, 2 or 3, then all of the SBCs in the cluster may contain no more that 256MB of DRAM. In this case, a maximum of 7 or 8 SBCs may be located in a cluster, depending upon the BPP0 cluster configuration being used.

In this type of BPP0 configuration, the valid values for the **P0_BPP0_SBC_ID** tunable are: 2, 4 or 6.

The following table shows the valid logical SBC ID values for each of the two P0 overlays for a given valid **P0_BPP0_SBC_ID** value. It should be noted that when there are less than the maximum number of boards in the configuration, any subset of the valid SBC IDs specified below for each of the P0 overlays may be assigned.

Note that the local SBC ID values do not denote specific slot locations in the rack or specific P0 overlay positions:

P0_BPP0_SBC_ID	First P0 overlay logical SBC IDs	Second P0 overlay logical SBC IDs
2	0 1	2 3 4 5 6 7
4	0 1 2 3	4 5 6 7
6	0 1 2 3 5	6 7

The following list contains the additional restrictions for BPP0 cluster configurations with DRAM sizes <= 256MB:

- Since the largest P0 overlay accommodates a maximum of 5 slots/ boards, the maximum number of SBCs in the cluster is therefore 7 when the **P0_BPP0_SBC_ID** parameter is set to either 2 or 6 (5 boards on one overlay + 2 boards on the other overlay). Only when

P0_BPP0_SBC_ID is set to 4 will a maximum number of 8 SBCs be configurable in the cluster (with 4 SBCs located on each P0 overlay).

- When **P0_BPP0_SBC_ID** is set to 6, then the logical SBC ID value of 4 may not be used on the first overlay. However, it is still possible to place 5 SBCs on the first overlay in this configuration when a 5-slot overlay is used as the first P0 overlay.

When **P0_BPP0_SBC_ID** is set to 2 and **VME_DRAM_WINDOW** is set to 3 (256MB) then the largest value allowed for the **SBC_SLAVE_MMAP_MAXSZ** diskless `cluster.profile` parameter is one-eighth the amount of memory defined by the **VME_DRAM_WINDOW** parameter. (Usually, up to one-fourth the amount of memory defined by the **VME_DRAM_WINDOW** parameter may be used for the **SBC_SLAVE_MMAP_MAXSZ** parameter. See section "Cluster-wide Parameters" on page 4-19 for more information about the **SBC_SLAVE_MMAP_MAXSZ** parameter.)

2. Configurations with a VME_DRAM_WINDOW size = 512MB

When **VME_DRAM_WINDOW** is set to a value of 4, then all of the SBCs in the cluster contain 512MB of DRAM or less. In this case, a maximum of 4 SBCs may be located in the cluster.

In this BPP0 configuration, the valid values for the **P0_BPP0_SBC_ID** tunable are: 1, 2 or 3.

The following table shows the valid logical SBC ID values for each of the two P0 overlays for a given valid **P0_BPP0_SBC_ID** value. It should be noted that when there are less than the maximum number of boards in the configuration, any subset of the valid SBC IDs specified below for each of the P0 overlays may be assigned.

Note that the logical SBC ID values do not denote specific slot locations in the rack or specific P0 overlay positions:

P0_BPP0_SBC_ID	First P0 overlay logical SBC IDs	Second P0 overlay logical SBC IDs
1	0	1 2 3
2	0 1	2 3
3	0 1 2	3

The only additional restriction for this type of BPP0 cluster configuration is:

- When **P0_BPP0_SBC_ID** is set to 1, then the largest value allowed for the **SBC_SLAVE_MMAP_MAXSZ** diskless `cluster.profile` parameter is one-eighth the amount of memory defined by the **VME_DRAM_WINDOW** parameter. (Usually, up to one-fourth the amount of memory defined by the **VME_DRAM_WINDOW** parameter may be used for the **SBC_SLAVE_MMAP_MAXSZ** parameter. See section "Cluster-wide Parameters" on page 4-19 for more information about the **SBC_SLAVE_MMAP_MAXSZ** parameter.)

3. Configurations with a VME_DRAM_WINDOW size = 1GB

   When **VME_DRAM_WINDOW** is set to a value of 5, then the two SBCs in the cluster contain 1GB of DRAM or less. In this case, a maximum of 2 SBCs may be located in the cluster.

   In this BPP0 configuration, the only valid value for the **P0_BPP0_SBC_ID** tunable is 1.

   The following table shows the valid logical SBC ID values for each of the two P0 overlays for the one valid **P0_BPP0_SBC_ID** value.

   Note that the logical SBC ID values do not denote specific slot locations in the rack or specific P0 overlay positions:

P0_BPP0_SBC_ID	First P0 overlay logical SBC IDs	Second P0 overlay logical SBC IDs
1	0	1

   There are no additional restrictions for this type of BPP0 cluster configuration.

B
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. Note that the instructions provided in this appendix apply to both loosely/closely-coupled systems. 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 the client is a closely-coupled client and the ncr module is being enabled, then use the Modules function of the **config(1M)** tool to make sure that the following kernel module is disabled:

   sym_dma (P0Bus DMA module - Used only when the ncr module is not configured into the client's kernel. The ncr and sym_dma modules are mutually exclusive.

   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 boot image.

   For a closely-coupled system you can optionally use the **-B** option to also boot

the client:

mkvmebstrap -B -r *all <client>*

For a loosely-coupled system:

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 boot image.

For a closely-coupled system you can optionally use the **-B** option to also boot the client:

mkvmebstrap -B -r *all <client>*

For a loosely-coupled system:

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 --------------------
```

C
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 nodes in the cluster. Note that these instructions apply to both loosely-coupled and closely-coupled systems. See Appendix B "Adding a Local Disk" for instructions on how to add a local disk to a diskless client.

We will refer to the client that has a local disk attached as the disk_server and the node(s) that want to share this disk as disk_clients.

Note

Note that on systems where both network interfaces (p0 and eth) are enabled, each node has multiple hostnames - one for each enabled network interface. Diskless clients follow the rule of <nodename>-p0 and <nodename>-eth, respectively, for the p0bus and the symbios ethernet networks. However, the node configured as the File Server of a closely-coupled system uses <nodename> for its ethernet network hostname and <nodename>-p0 for its p0bus interface. It is important that the hostname specified is attached to the same interface on the disk_clients and the disk_server. A hostname is specified on the disk_clients when mounting the partitions and on the disk_server in the share command line in the **dfstab** table.

First we must enable the **nfssrv** kernel module which, by default, is disabled in a diskless client configuration. We must also give the disk_client node(s) permission to access the partitions. The following commands are to be run on the node configured as the File Server of the loosely or closely-coupled system.

1. Enable the **nfssrvr** kernel module for the client that has the local disk attached (disk_server). This may be accomplished in several ways, either by using **config(1M)** and using the **-r** option to specify the virtual root directory or by adding it to the **kernel.modlist.add** custom file.

```
# 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 boot image for the disk_server node by executing the command:

   If this is a closely-coupled system you may use the **-B** option to also boot the disk_server node:

   mkvmebstrap -B -r *all* **<disk_server>**

   If this is a loosely-coupled system:

   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.

Backplane P0 Bridge Board (BPP0)

A P0*PCI bridge board, which may be used to connect two P0Bus Overlay boards together in order to create a larger common P0Bus in a closely-coupled system configuration. See definitions for **P0Bus Overlay** and **P0*PCI (P0Bus)**.

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 if the client maintains a P0Bus or Ethernet connection to its File Server, it may use that File Server as a kind of remote disk device, utilizing it to fetch applications, data, and to swap unused pages to.

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.

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 SBC boards, this is usually accomplished with **`SMon fp uf`** command.

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 **`SMon smonconfig`** command, and a **`SMon`** startup script may be created and configured to automatically execute after a reset, in order to download and execute a boot image via TFTP with the **`SMon tftpboot`** command.

netload

The process of a target loading a boot image as discussed under netboot, but without subsequently executing it. On SBC boards, netloading is invoked with the **`Smon load`** 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.

P0*PCI (P0Bus)

The PCI-to-PCI (P0) hardware bus interface that provides improved SBC board-to-board performance. The P0Bus is 64 bits wide and operates at 33 MHz, for a theoretical maximum of 264 MB/sec. This is more than three times the theoretical maximum of the standard VME64 bus of 80 MB/sec. The Power Hawk Series 700 P0Bus interface is based on the Intel 21554 64-bit PCI-to-PCI bridge chip.

The P0Bus hardware is required for Power Hawk Series 720/740 Closely Coupled configurations, where the P0Bus is used for Closely Coupled inter-SBC communications.

P0*PCI (P0Bus) Overlay

A P0*PCI connector board, which is used to connect multiple SBCs in the same cardcage (cluster) to a common P0Bus.

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 (7400) 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**.

SMon

A board-resident ROM monitor utility that provides a basic I/O system (BIOS), a boot ROM, and system diagnostics for Power Hawk Series 700 single board computers (SBCs).

SMon startup script

As part of the boot process, **`SMon`** can automatically perform **`SMon`** commands and/or user defined functions written in a startup script that is stored in NVRAM (nonvolatile RAM). Special startup scripts are used for booting client SBCs in closely-coupled configurations, and also 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.

SYM (sym)

Internal Symbios Logic SYM53C885 PCI-SCSI/Fast Ethernet Multifunction Controller.

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 in order for the operation system to function properly. In a closely-coupled cluster, 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

Numerics

100base-T Glossary-1
10base-T Glossary-1

A

ARP Glossary-1

B

banked flash 5-2
Basic Client
  customizing the configuration 3-13
Basic Configuration
  Customizing 4-40
Block
  device Glossary-2
  driver Glossary-1
Board
  Jumpers 4-6
Boot
  device Glossary-2
Boot Image 3-27, 4-57
  Characteristics of 1-10
  Creation of 1-10
  how to create 4-60
Boot Status
  how to verify 3-30, 4-62
Bootable object file Glossary-2
Booting 1-12, 4-56
  a netbootable client from flash 5-2
  Flash 4-62
  the boot image 3-26
booting
  Net 4-62
  VME 4-61
Booting Options 4-58
BPP0 configurations A-1
Bridge Board A-1
Burn
  a boot image into User Flash 5-3
burn process 5-5
Burning 5-3
  a Netboot Client's User Flash 5-3
Burning and Booting
  for VMEBus Bootable Clients 5-4
Burning and booting from Flash 5-4

C

Cache Glossary-2
Changing
  Default VME Configuration 6-3
Character
  driver Glossary-3
  I/O schemes Glossary-3
Client
  how to shutdown 4-63
client Glossary-3
Client Board Configuration 4-11
Client Configuration 3-8
client kernel 7-2
Client Profile File 3-8
Client Profile Parameters 3-24
Client profile parameters
  how to modify 3-24
client's boot image 5-3
client's global memory 5-3
Closely-Coupled
  VME A32 window considerations 6-4
Closely-Coupled System Hardware Prerequisites 1-22
Cluster
  Configuration 4-18
Cluster Configuration 4-18
cluster members 7-1
Cluster-wide Parameters 4-19
config utility 3-15, 4-42
Configuration Files
  Custom 4-43
Configuration Toolsets 1-5
Configuring
  Clients Using netbootconfig 3-11

Diskless Systems 1-22
console debugger 7-2
console terminal 7-2
copying the boot image 5-3
crash 7-1, 7-2
crash analysis 7-3
Creating
a Client Configuration 3-11
the Boot Image 3-28
Critical code Glossary-3
Custom Configuration Files 3-16
Customizing
the Basic Client Configuration 3-13, 4-40

D

default client kernel 7-2
Default VME Configuration 6-1
Definitions
of terms 1-6
device switch table Glossary-4
Disk Space Requirements 1-23
disk_clients C-1
disk_server C-1
Diskless Boot Basics 1-3
diskless client 7-1
Diskless Implementation 1-10
Diskless Topography 1-1
Driver routines Glossary-4
Dynamic Memory Allocations 4-38

E

Embedded Client 1-2
launching an application 4-55
Embedded Clients
launching an application 3-25
ENV
Set Environment Command Glossary-4
erasing the User Flash 5-3
Ethernet LAN 1-23
Example
of non-default VME A32 space configuration 6-4
Examples
on Creating the Boot Image 3-28

F

Features
Series 700 SBC Hardware 1-9
File Server Glossary-5
File Server Board Configuration 4-7
file server system 7-2
flash autobooting Glossary-4
Flash Boot 1-7, 1-14
flash booting Glossary-4
flash burning Glossary-5
flash memory Glossary-5
flashboot 5-3
flashbooting 5-4
flashburning 5-4
Functions Glossary-5

G

GEV Glossary-5

H

hard reset 5-5
Hardware Characteristics
User Flash 5-2
Hardware Overview 1-9
Hardware Prerequisites
Closely-Coupled System 1-22
Loosely-Coupled System 1-23
host Glossary-5
How To
Boot the Cluster 4-4

I

idtuneobj 3-15, 4-42
Illustrations
Closely-Coupled Cluster of Single Board Computers 1-3
Loosely-Coupled System Configuration 1-2
Init Level 1-8
Inittab Table 4-46
Installing
Additional Boards 3-1
Loosely-Coupled System 3-1
the Cluster 4-2

Installing Additional Boards 3-3
Integrated
Disk File Controller (IDFC) Glossary-6
Interrupt level Glossary-5
Interrupt priority level (IPL) Glossary-6

K

K00client rc script 3-18, 4-45
kdb 7-1, 7-2
Kernel buffer cache Glossary-6
Kernel Configuration
Modifying 4-40
kernel.modlist.add 3-14, 4-41
kernel.modlist.add Table 3-20, 4-47
keyadm 1-24

L

Launching an Appication for NFS Clients 3-25
Launching an Appilcation for Embedded Clients 3-25
Launching an Application (Embedded Client) 4-55
Launching an Application (NFS Clients) 4-55
Launching Applications 3-25, 4-55
Licensing Information 1-24
Limitations
on configuration of VME address space 6-3
local cluster 7-2
Local Disk B-1
Loosely-Coupled System Hardware Prequisites 1-23

M

Make Client System Run in NFS File Server Mode C-1
mapping
P0Bus and VME space 7-1
MEMFS Root Filesystem 1-11
memfs.files.add 3-21
memfs.files.add Table 3-21, 4-48
memfs.inittab and inittab Tables 3-19
Memfs.inittab Table 4-46
Memory
Shared 1-20
mknetbstrap 3-15
Modifying
client profile parameters 3-24
Modifying Profile Parameters 4-51
Modifying the Kernel Configuration 3-14, 4-40

N

NCR/Symbios SCSI 1-23
Net Boot 1-13
Netboot
using SMon 3-29
netboot Glossary-7
netbootable client 5-3
netload Glossary-7
network boot Glossary-7
Network Information Services Glossary-12
network load Glossary-7
NFS Client
launching an application 4-55
NFS Clients 1-3
launching an application 3-25
NFS Related Parameters 4-29
Node Configuration 4-33

O

Overview
of Hardware 1-9

P

P0 Bridge A-1
P0 overlay positions A-3
P0 overlay restrictions A-1
P0Bus
Networking 1-15
paged flash 5-2
panics 7-3
Parameters
modifying profile 4-51
Point-to-Point protocol Glossary-8
Portable device interface (PDI) Glossary-8
Power Hawk
Networking Structure 1-16
PPP Glossary-8
Prerequisites
Hardware, Closely-Coupled System 1-22
Hardware, Loosely-Coupled System 1-23
Software 1-24
processor and user limits 1-24
profile
modifying parameters 4-51

R

random I/O Glossary-9
Raw I/O Glossary-9
rc scripts
    K00client 4-45
    S25client 4-45
rcp Glossary-9
read queue Glossary-9
Reasons to Modify Defaults 6-2
Remote
    File Sharing 1-17, Glossary-9
Removing
    a Client Configuration 3-11
Required Parameters 3-9, 4-26
RFS Glossary-9
rlogin Glossary-9
rsh Glossary-9

S

S25client rc script 3-18, 4-45
savecore 7-1, 7-3
SBC
    Cluster Configuration 4-6
    Configuration, client board 3-3
SBC Configuration 3-3
sbc device driver 7-1
SBC ID value A-1
sbcboot command 5-4
sbcmon 7-1, 7-3
SCSI
    driver interface (SDI) Glossary-10
SCSI CD-ROM 1-23
sequential I/O Glossary-10
Serial Port A 1-22, 1-23
Series 700 Hardware Features 1-9
server Glossary-10
Server SBC Parameters 4-24
Shared Memory 1-20
Shutdown 3-26, 4-56
Slave Shared Memory Support 4-36
SLIP Glossary-10
Small Computer System Interface (SCSI) Glossary-10
SMTP Glossary-10
SNMP Glossary-10
Source Code Control System (SCCS) Glossary-10
Space
    Swap 1-20
Static Memory Allocations 4-37
Subsystem Support 3-13, 4-35
Swap Space 1-20
swap space 1-8, Glossary-11
Symbios SYM53C885 1-23
Synergy Monitor (SMon) 1-7
system console 1-22, 1-23
system debugging 7-1
system dump 7-1, 7-3
System initialization Glossary-11
System Run Level 1-8, Glossary-11

T

Tables 3-21
    Configuration 3-8
    Hosts 3-11
    kernel.modlist.add 3-20, 4-47
    memfs.files.add 4-48
    memfs.inittab & inittab 4-46
    memfs.inittab and inittab 3-19
    vfstab 3-20, 4-47
target Glossary-11
target system Glossary-11
TCP Glossary-12
TELNET Glossary-11
TFTP Glossary-11, Glossary-12
tftp protocol 5-3
The cluster.profile File 4-19
Tools 7-1
Toolset
    Configuration 1-5
Topography
    Diskless 1-1
Trivial File Transfer Protocol Glossary-12
Trivial File Transfer Protocol(TFTP) 1-8
tunables
    modifying system 4-39

U

UDP Glossary-12
Unpacking
    Instructions 2-2
Upstream Glossary-12
User Flash 5-2
User Flash Hardware 5-2
user limits 1-24
Using netbootconfig 3-11
Utilities
    kdb, crash, savecore, sbcmon 7-1

V

vfstab Table 4-47
VGM5 Reset Switch 2-3
VGM5 SMI Switch 2-3
Virtual Root 1-10
VME
    A32 Window 6-3
    Booting 1-13
VMEBus bootable clients 5-2
vroot.files.add Table 3-22
VSS4 Reset Switch 2-4
VSS4 SMI Switch 2-4

Y

yellow pages Glossary-12

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 700 Diskless Systems Administrator’s Guide

0891086