

HP World 2002

Hewlett Packard Virtual Partitions (vPars)

Marty Poniatowski

Hewlett-Packard Company

Introduction

This paper is based on the Text *HP Virtual Partitions*, published by Prentice Hall PTR ISBN # 0-13-065419-1. All of the topics in this paper are covered in much more detail in the book.

I refer to chapters in the vPars book many times in this paper. There is a lot of good vPars background and many useful examples in this paper; however, vPars are a complex topic on which I based a separate book so there is no way I could cover all aspects of vPars in one paper. I refer to the chapters in the vPars book as a source of more detail on a given topic.

About Virtual Partitions

With Virtual Partitions (vPars) you can take almost any HP 9000 server and turn it into many "virtual" computers. These virtual computers can each be running their own instance of HP-UX and associated applications. The virtual computers are isolated from one another at the software level. Software running on one Virtual

Partition will not affect software running in any other Virtual Partition. In the Virtual Partitions you can run different revisions of HP-UX, different patch levels of HP-UX, different applications, or any software you want and not affect other partitions.

There are some base requirements that must be met in order to run vPars on your system. At the time of this writing, the following minimum requirements must be met for each vPar on your system:

- Minimum of one CPU.
- Sufficient memory to run HP-UX and any other software that will be present in the vPar.
- A boot disk off of which HP-UX can be booted. At the time of this writing it is not possible to share bus adapters between vPars. Therefore, a separate bus adapter is required for each of the vPars. This requirement may have been removed by the time you read this book.
- A console for managing the system. The console can be either physical or virtual. We'll cover the console in detail in the book.
- An HP 9000 system supported by HP-UX 11i. At the time of this writing only HP-UX 11i is supported in vPars. With systems based on Itanium Processor Family (IPF) processors, there are plans to support numerous operating systems in vPars in the future.

The system we'll use in most of the examples throughout this paper is an rp 54xx (formerly know as L-Class) system that meets all of the requirements in the previous list. You may also want to have additional disks and a separate LAN card in your vPars. I strongly recommend the LAN card so that you can establish TELNET, or other, sessions to your vPars rather than connect to them only from the console. The LAN card is also required to perform backup and Ignite-UX related work.

If you have Instant Capacity on Demand (iCOD) employed on your server, all CPUs must be activated in order for vPars to work. When employing Processor Sets (psets) in a vPar, use only bound CPUs.

There is a vPars product bundled with HP-UX 11i as well as a full, or add-on product. There are very few limitations with the add-on product. The bundled

product has a limitation of a maximum of two vPars and one of the vPars can have only one CPU.

This paper was written with Virtual Partitions software that had recently been released for the first time. There have been many enhancements to Virtual Partitions since the writing of this paper. There is a Graphical User Interface being considered for vPars that I haven't covered in this paper. There is something to be said for working with a product when it is new. You really get a good understanding of the functionality of the product by using the command line only and performing a lot of manual procedures. In addition, Superdome vPars software is in covered in an Appendix in the vPars book but not in this paper.

Virtual Partitions Background

HP-UX Virtual Partitions (vPars) allow you to run multiple instances of HP-UX on the same HP 9000 server. From a hardware perspective a vPar consists of CPU, memory, and I/O that is a subset of the overall hardware on the computer. From a software perspective a vPar consists of the HP-UX 11i Operating Environment and all application-related software to successfully run your workload. Figure 1-1 shows a conceptual diagram of the way in which HP 9000 computer system resources can be allocated to support multiple vPars.

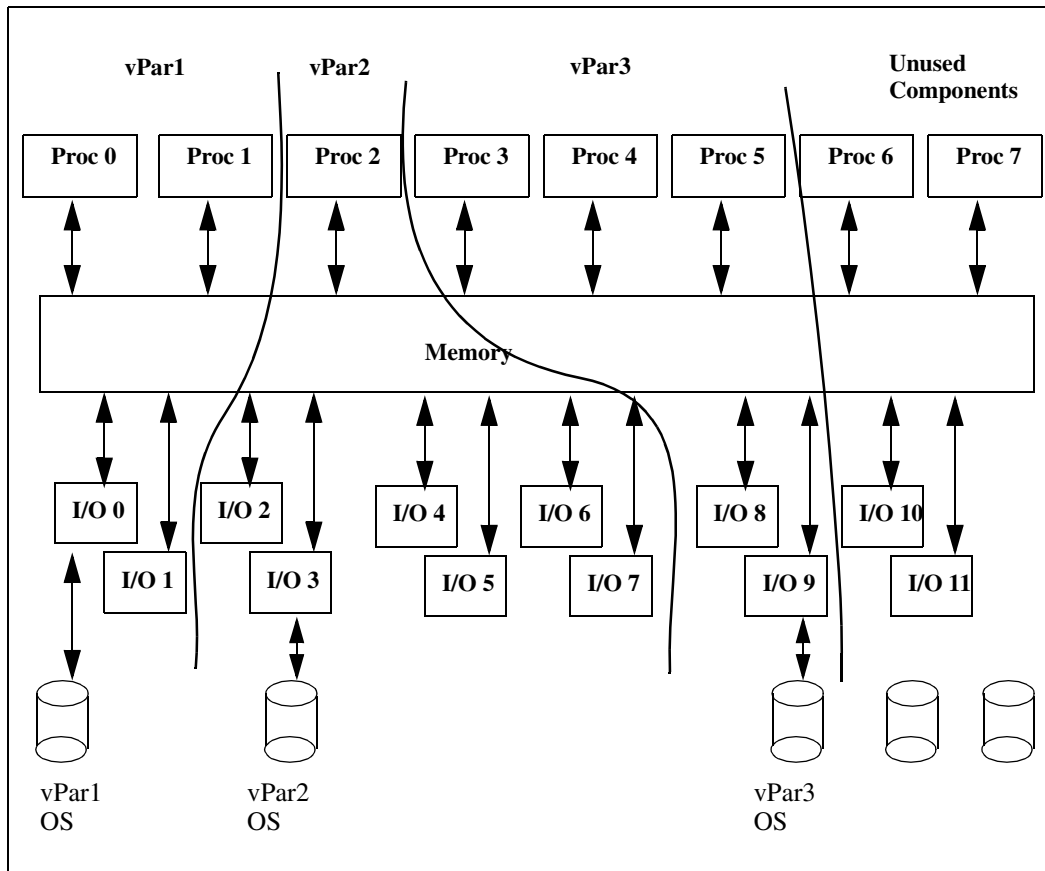


Figure 1-1 Example of HP System Resource Allocation with vPars

The components of which your HP 9000 is comprised can be allocated in a variety of ways. You can see that the eight-way system shown has a different number of processors, different amount of memory, and different number of I/O cards allocated to each vPar. The unused components can be added to any of the

vPars or be the basis for yet another vPar. In addition, components can be moved from one vPar to another (with some restrictions described later in the book).

Uses of Virtual Partitions

I have worked on many vPars installations that have a variety of uses for vPars. The following are a sampling of the reasons to use vPars:

- Increased System Utilization - Many servers are underutilized. With vPars you can devote a subset of system resources to each vPar. With each vPar running its own instance of HP-UX 11i and associated applications, you'll get higher overall system utilization.
- Flexibility - Many applications have resource needs that change. With vPars you can devote fewer system components when application needs are low and additional resources when an application needs them. An increased end-of-the-month workload, for instance, can be given more system resources to complete faster.
- Server Consolidation - Running multiple instances of HP-UX 11i and their associated applications on one HP server reduces the overall number of servers required. Web servers that had run on different servers can now be run in different vPars on the same computer.
- Application Isolation - HP vPars are fully software-isolated from one another. A software failure in one vPar does not affect other vPars.
- Mixed Production, Test, and Development - Production and testing can take place on the same server with vPars. When testing is complete, the test vPar can become the production vPar. Similarly, development usually takes place on a separate system. With the software isolation of vPars, however; development can take place on the same system with other applications.

These are just a sampling of the uses I've seen for vPars. Many others will emerge as vPars become widely used and systems experts implement them in more computing environments.

Preparing to Create Virtual Partitions

After having loaded vPars software we can create partitions. Chapter 1 of the vPars book covers loading HP-UX 11i and vPars software in detail. At the time of this writing vPars software comes on a separate CD-ROM. Now we're ready to create Virtual Partitions. After we create Virtual Partitions in this paper we have to boot the partitions we've created. Although we will boot the partitions in this paper, the details of booting Virtual Partitions, and HP 9000 booting in general, are covered in detail in Chapter 3 of the vPars book.

With both HP-UX 11i and the Virtual Partitions software on our disk, we can begin the process of creating partitions. Our goal is to have a system that looks like that in Figure 1-2:

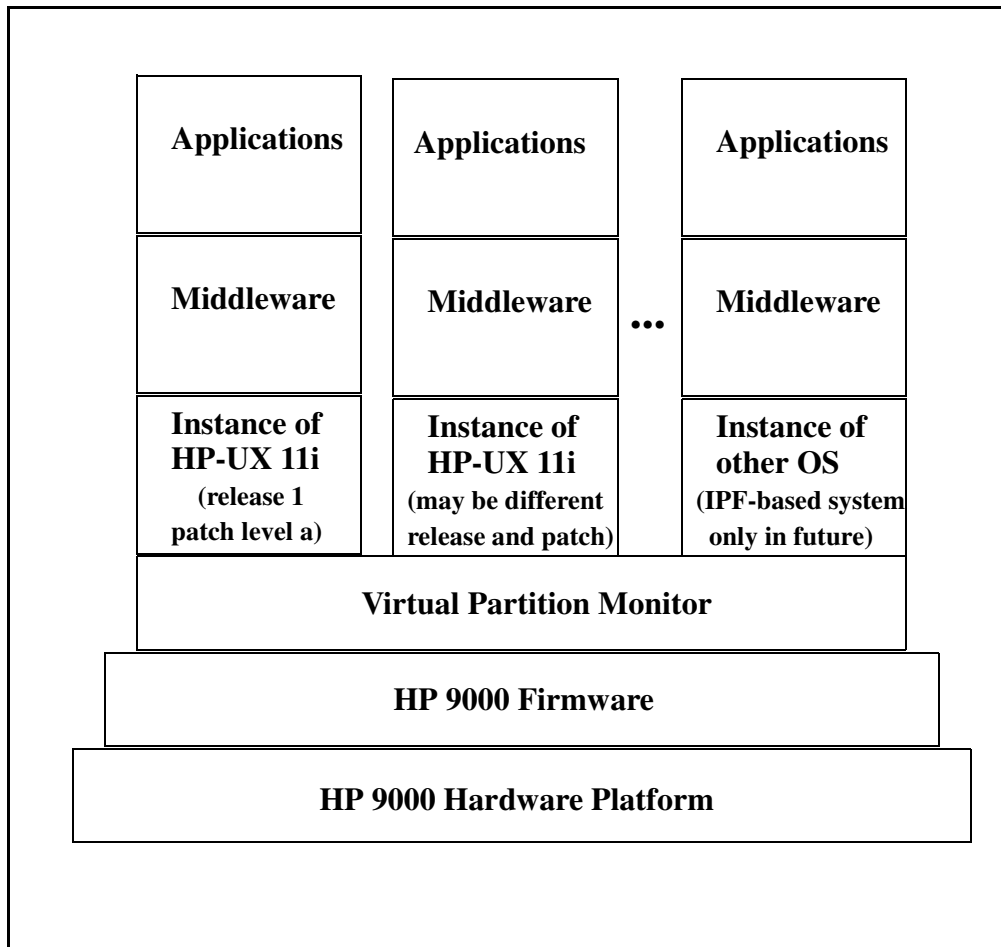


Figure 1-2 Virtual Partitions Software Stack

There are many components in Figure 1-2. We already have many of the components in this diagram on our system. Starting from the bottom we have the hardware, firmware, Virtual Partition Monitor (covered as part of the boot process in Chapter 3 of the vPars book), and HP-UX 11i installed on two different disks. There are two HP-UX 11i instances shown in the left-most two stacks of Figure 1-2. These are the operating systems we already have loaded.

The two HP-UX 11i instances can't run simultaneously on our rp 54xx (formerly know as L-Class) system because we have not yet created our Vir-

tual Partitions. Without Virtual Partitions created, we can boot HP-UX off of one *or* the other of these disks but we can't run both. Let's now create our Virtual Partitions so we can indeed have two instances of HP-UX 11i running simultaneously. After Virtual Partitions have been created, you can proceed to load the middleware and applications shown on top of HP-UX 11i in Figure 1-2.

Although we're covering running multiple instances of HP-UX 11i in the Virtual Partitions we create, we won't be restricted to only HP-UX 11i in the future. On systems based on the Itanium Processor Family (IPF) CPUs, we'll be able to run additional operating systems. The rightmost Virtual Partition in Figure 1-2 depicts this future capability. This capability does not exist at the time of this writing, so it is not covered. Look for an updated revision of the vPars book in the future to cover this capability once it exists and keep an eye on www.hp.com for both IPF and Virtual Partition enhancements.

Although the examples in this paper take place on four-way (four-processor) rp 54xx (formerly know as L-Class) systems, vPars run on most HP servers. The more components of which your HP server is comprised, the more vPars you can have on the server and the more options you have for crafting vPars. In the examples in this paper we'll end up with a four-way rp 54xx (formerly know as L-Class) with two fully configured Virtual Partitions. Next we'll cover loading software.

Loading the Software Required for Virtual Partitions

We cover installing Virtual Partitions software in this section. I assume that you already have HP-UX 11i installed on your system or know how to do so.

Keep in mind that HP-UX must be loaded for each Virtual Partition you wish to run. If, for instance, you want to run two Virtual Partitions, as we do in our examples in this book, HP-UX 11i will need to be loaded for both Virtual Partitions. The procedure covered for loading HP-UX 11i needs to be performed for every Virtual Partition you want to run. HP-UX 11i can be loaded from media, such as your HP-UX 11i distribution on a CD-ROM or from an Ignite/UX server. You can use any method to load HP-UX 11i and the Virtual Partitions software for every Virtual Partition you want to run.

At the time of this writing there are two vPars products: a product with the full functionality covered throughout the book and a free product that has a subset of the full product. The free product has a limitation of a maximum of two vPars, and one of the vPars can have only one CPU.

When you buy the full product, product number T1335AC, a CD-ROM is provided that has on it the following components:

- vPars software
- patches
- vPars administration guide
- booklet
- WINSTALL files used for booting

At some point the full product will be on standard distribution rather than a separate CD-ROM. The free product, called VPARSBASE at the time of this writing, can be downloaded from www.software.hp.com.

Figure 1-3 shows an example of the software components that appear below the full product T1335AC.

```

Terminal
Window Edit Options Help
=== SD Install - Software Selection (sfamwpl) (1) Help
File View Options Actions
Press CTRL-K for keyboard help.
Source: sfamwpl:/var/marty/vpar_ic10.tar
Target: sfamwpl:/
Only software contained in the parent bundle is shown.
All software on the source is available for selection.
-----
Subproducts or Filesets:T1335AC.VirtualPartition 0 of 3 selected
-----
Marked? Name Revision Information
-----
..(go up)
VPAR-KRN -> A.01.00.10 Virtual Partition Kernel
VPAR-MON2 -> A.01.00.10 Virtual Partition Monito
VPAR-RUN -> A.01.00.10 Virtual Partition User S

```

Figure 1-3 Example of Loading vPars Software

Figure 1-3 shows the three components of which T1355AC is comprised: the kernel, monitor, and run environment. The additional components listed earlier, patches and WINSTALL, will also have to be loaded for vPars to be fully operational.

After loading this software, a reboot takes place to build the kernel. This is done for you automatically; however, vPars kernel-related background is covered in both Chapters 2 and 4 of the vPars book.

All of the vPars software must be loaded on every HP-UX 11i volume that will be used on your vPars server. The loading of this software will take place for every HP-UX 11i instance that you wish to run simultaneously on your vPars server. There are two ways to load the HP-UX 11i operating system and vPars software on all of the volumes used for vPars. The first, which is the method used throughout this book, is to load HP-UX 11i and vPars software on all vPars volumes prior to creating Virtual Partitions. The second is to load only the volume of the first vPar with all software, create as many vPars as you want, and then use `vparboot -p vp_name -I ignite_kernel` to boot and load HP-UX 11i on the other disks. In this paper, I first load HP-UX 11i and vPars software on all disks before creating vPars.

A lot of software has been loaded as a result of loading the vPars software. The `/sbin` directory has in it the `vpar` commands we'll use in upcoming sections. The following is a long listing of the `vpar` commands in `/sbin`.

```
# ll /sbin/vpar*
-r-xr-xr-x 1 bin bin 128760 Oct 18 22:08 /sbin/vparboot
-r-xr-xr-x 1 bin bin 161216 Oct 18 22:07 /sbin/vparcreate
-r-xr-xr-x 1 bin bin 101040 Oct 18 22:04 /sbin/vpard
-r-xr-xr-x 1 bin bin 59592 Oct 18 22:04 /sbin/vpardump
-r-xr-xr-x 1 bin bin 30520 Oct 18 22:04 /sbin/vparextract
-r-xr-xr-x 1 bin bin 140072 Oct 18 22:08 /sbin/vparmodify
-r-xr-xr-x 1 bin bin 47232 Oct 18 22:04 /sbin/vparreloc
-r-xr-xr-x 1 bin bin 127808 Oct 18 22:08 /sbin/vparremove
-r-xr-xr-x 1 bin bin 132008 Oct 18 22:08 /sbin/vparreset
-r-xr-xr-x 1 bin bin 152040 Oct 18 22:08 /sbin/vparstatus
-r-xr-xr-x 1 bin bin 26152 Oct 18 22:04 /sbin/vparutil
#
```

These are the commands that you'll use to create, view, modify, and work with vPars in general. Chapter 2 of the vPars book is devoted to describing these commands and giving examples of using most of them in their various forms. In addition, the tear-out card included with the book summarizes many of these commands.

There are several files in `/stand` related to the vPars kernel. The following listing shows some of these:

```
# ll /stand/vp*
-rw----- 1 root      root      8232 Nov 16 07:09 /stand/vpdb
-r-xr-xr-x 1 bin       bin       849992 Oct 18 22:02 /stand/vpmon
#
```

vpmon is loaded at the time of system startup and is the basis for running vPars. Chapter 3 of the vPars book covers booting in detail, including bringing up **vpmon**. **vpdb** is the vPars database that contains all information related to all of the vPars running on your system. This file is automatically synchronized by the vPars application to ensure that all vPars have the same information about all vPars on your system.

There are several startup-related files, including those shown below, which are covered in more detail in Chapter 8 of the vPars book covering startup.

```
/etc/rc.config.d/vpard
/etc/rc.config.d/vparhb
/etc/rc.config.d/vparinit
```

```
/sbin/init.d/vpard
/sbin/init.d/vparhb
/sbin/init.d/vparinit
```

Of particular interest is **vparhp**, which is the *heartbeat* daemon related to keeping **vpdb** synchronized on all of your vPars.

Very important to your work related to vPars are the online man pages. The following listing shows the man pages loaded on my system at the time of this writing.

```
# man -k vpar
vparboot(1M)      - boot a virtual partition
vparcreate(1M)   - create a virtual partition
vpardump(1M)     - manage monitor dump files
vparextract(1M)  - extract memory images from a running virtual
                  partition system
vparmodify(1M)   - modify the attributes of a virtual partition
vparreloc(1M)    - relocate the load address of a vmunix file,
                  determine if a vmunix file is relocatable,
                  or promote the scope of symbols in a
                  relocatable vmunix file
vparremove(1M)   - remove a virtual partition
vparreset(1M)    - reset a virtual partition
vparresources(5) - description of virtual partition resources
```

```

and their requirements
vparstatus(1M) - display information about one or more
                virtual partitions
vpartition(1)  - display information about the Virtual
                Partition Command Line Interface
vparutil(1M)  - get and set SCSI parameters for SCSI
                controllers from a virtual partition
#

```

Many of these man pages appear in Appendix A of the vPars book.

At this point we have HP-UX 11i and the Virtual Partitions software loaded on the system.

Virtual Partitions Command Summary

There are several commands used to create and work with Virtual Partitions. A table in the vPars book as well as a tear-out card provide an overview of many commonly used Virtual Partitions-related commands. Table 1-1 is an abbreviated version of command summary:

Table 1-1 Virtual Partition Commands

Command	Description
<i>ISL</i> > Initial System Load prompt.	Virtual Partitions Monitor is loaded from <i>ISL</i> > with: <i>ISL</i> > hpux /stand/vpmon MON> To load Virtual Partitions directly from <i>ISL</i> >, use: <i>ISL</i> > hpux /stand/vpmon vparload -p vPar_name

Command	Description
<p><i>MON</i>></p> <p>Virtual Partitions Monitor prompt. (Also see vparload command.)</p>	<p>This is loaded from <i>ISL</i> with:</p> <pre>ISL> hpux /stand/vpmon</pre> <p><i>MON</i>></p> <p>To load an alternate database from <i>ISL</i>, use:</p> <pre>ISL> hpux /stand/vpmon -D db_file</pre> <p>To load one vPar from <i>MON</i>, use:</p> <pre>MON> vparload vPar_name</pre> <p>Many other commands can be issued from <i>MON</i>. Type help or ? to list. (Commands include: scan, vparinfo, ls, log, getauto, lifls, cbuf, cat.)</p>
<p>vparload</p> <p>Load Virtual Partitions from <i>MON</i>> prompt only.</p>	<p>To boot a Virtual Partition from <i>MON</i>>:</p> <pre>MON> vparload -p vPar_name</pre>
<p>vparboot</p> <p>Boot a Virtual Partition from the command line only.</p>	<p>To boot a Virtual Partition from the command line:</p> <pre># vparboot -p vPar_name</pre>
<p>vparcreate</p> <p>Create a Virtual Partition.</p>	<p>To create a Virtual Partition with three processors (<i>num</i>) total, two bound (<i>min</i>), 2048MB RAM, all components on 0/0, boot disk at 0/0/1/1.2.0, with a kernel of <i>/stand/vmunix</i>, autoboot on, and console at 0/0/4/0:</p> <pre># vparcreate -p vPar_name -a cpu::3 -a cpu:::2:4 -a mem::2048 -a io:0/0 -a io:0/0/1/1.2.0:boot -b /stand/vmunix -B auto</pre>
<p>vparmodify</p> <p>Modify the attributes of a Virtual Partition.</p>	<p>To add processor at path <i>109</i> (adds this proc to those already assigned):</p> <pre># vparmodify -p vPar_name -a cpu:109</pre>
<p>vparremove</p> <p>Delete a Virtual Partition.</p>	<p>To delete a Virtual Partition in the currently running database:</p> <pre># vparremove -p vPar_name</pre>
<p>vparreset</p> <p>Reset a Virtual Partition.</p>	<p>To reset a Virtual Partition without TOC (t), hard (h), bypassing display of PIM data (q), or forcing (f):</p> <pre># vparreset -p vPar_name</pre>

Command	Description
vparresources(5) man page Provides description of Virtual Partitions and their resources.	This is a manual page that describes Virtual Partition resources in general and how resources are specified in other commands, such as vparmodify .
vparstatus Display the status of Virtual Partitions.	To display the status of a Virtual Partition in verbose mode: # vparstatus -v -p vPar_name
vpartition man page Display information about the Virtual Partition Command Line Interface.	Provides the following brief description of Virtual Partitions commands: vparboot Boot (start) a virtual partition. vparcreate Create a new virtual partition. vparmodify Modify an existing virtual partition. vparremove Remove (delete) an existing virtual partition. vparreset Simulate a TOC or hard reset to a virtual partition. vparstatus Display virtual partition and available resources information.
Specify CPU Resources by:	Number of bound and unbound CPUs: <i>cpu::num</i> CPU hardware path(s): <i>cpu:path</i> Minimum and maximum number: <i>cpu::[min]:[max]</i>
Specify Memory by:	Size <i>mem::size</i> Base and range: <i>mem::base:range</i> combination of both above.
Specify I/O:	Use path: <i>io:path[:attr1[,attr2[...]]]</i> (see man page vparresources for details).
To add resources use: (This adds component relative to what already exists if running vparmodify .)	<i>-a cpu:path</i> <i>-a cpu:num</i> (can be done with vPar running) <i>[-a cpu::num] [-a cpu::[min]:[max]] [-a cpu:path] (::: is vparcreate only)</i> <i>-a io:path[:attr1[,attr2[...]]]</i> <i>-a mem::size</i> <i>-a mem::base:range</i>
To delete resources use (This deletes component relative to what already exists if running vparmodify .)	<i>-d cpu:path</i> <i>-d cpu:num</i> (can be done with vPar running) <i>-d io:path[:attr1[,attr2[...]]]</i> <i>-d mem::size</i> <i>-d mem::base:range</i>

Command	Description
To modify resources use: (This modifies to absolute number rather than relative.)	<i>-m cpu::num</i> (can be done with vPar running) <i>-m cpu:::[min][:max]]</i> <i>-m io:path[:attr1[,attr2[...]]]</i> <i>-m mem::size</i>
vPars setboot Options: <i>-a</i> <i>-b</i> <i>-p</i> <i>-s</i> no options	Changes the alternate boot path of the Virtual Partition. Sets the autoboot attribute of the Virtual Partition. Changes the primary boot path of the Virtual Partition. No effect. Displays information about boot attributes. To set Autoboot <i>on</i> : # setboot -b on
vPars States: <i>load</i> <i>boot</i> <i>up</i> <i>shut</i> <i>down</i> <i>crash</i> <i>hung</i>	The kernel image of a Virtual Partition is being loaded into memory. This is done by the Virtual Partition monitor. The Virtual Partition is in the process of booting. The kernel image has been successfully loaded by the Virtual Partition monitor. The Virtual Partition has been successfully booted and is running. The Virtual Partition is in the process of shutting down. The Virtual Partition is not running and is down. The Virtual Partition has experienced a panic and is crashing. The Virtual Partition is not responding and is hung.

We'll use some of the commands shown in Table 1-1 in the upcoming section on creating virtual partitions. There is more detail on the Virtual Partition commands in Appendix A, which contains the online manual pages for the commands.

Let's now move on to creating our virtual partitions.

Steps to Create Virtual Partitions

In this section we'll cover the steps to create Virtual Partitions. These are steps I performed while working with vPars with some of the very first installations. This list should serve as a framework for working with vPars.

You may chose not to perform some of the steps and to add others. It is only a framework for getting vPars working on your system.

In our upcoming examples to create our Virtual Partitions, we'll execute the steps shown in Figure 1-4.

- 1) Load HP-UX 11i onto the disks on which you want to run a Virtual Partition* (media or Ignite/UX server.)
- 2) Load Virtual Partitions software onto the disk(s) on which you want to run a Virtual Partition.
- 3) Gather information on system components and hardware paths using **ioscan**, **dmesg**, and other commands.
- 4) List components of which Virtual Partitions will be comprised.
- 5) Build new HP-UX 11i kernels on all volumes on which Virtual Partitions will run* with **mk_kernel** and **kmupdate** (done automatically, done for all 11i instances that will run vPars.)
- 6) Create first Virtual Partition with **vparcreate**.
- 7) Boot first Virtual Partition with **vparload** at *MON*> prompt. Use **vparstatus -v** to see running vPar and **vparstatus -A** to see available components.
- 8) Create second Virtual Partition with **vparcreate** (can also be done before booting any vPars.)
- 9) Boot second Virtual Partition with **vparboot** from the first vPar and monitor it booting with **vparstatus**. After it has booted, view remaining available components with **vparstatus -A**.
- 10) Modify Virtual Partition(s) as required with **vparmodify** and view modifications with **vparstatus -v** and **vparstatus -A**. Modifications can be any type, such as adding CPUs, changing attributes, and so on.

Other tasks and comments:

- Many other tasks can be performed. Commands **vparremove** and **vparreset** were not used in steps above.

- A typical list of components of which a vPar would be comprised looks like the following:

```

name          cable1
processors    min of 1 (bound) max of 3 (1 bound 2 unbound) with num equal to 1
memory        1024 MB
LAN           0/0/0/0 (not specified explicitly)
boot disk     0/0/1/1.2.0
kernel        /stand/vmunix
autoboot      off (manual)
console       0/0/4/0 (not specified explicitly)

```

* HP-UX 11i must be loaded on volume before or after Virtual Partition is created. If HP-UX 11i is loaded after vPar is created, then **vparboot -p vp_name -I ignite_kernel** is used to load 11i.

Figure 1-4 Steps to Create Virtual Partitions

1) Load HP-UX 11i

HP-UX 11i must be loaded on the volumes that will be used to host all vPars. The method you use to install 11i, whether media, Ignite-UX, or some other technique, are all acceptable provided that HP-UX 11i is present on all of the disks. HP-UX 11i must be present on the first disk before you begin the vPar creation. You can create vPars on other disks before HP-UX 11i is loaded on them and then use `vparboot -p vp_name -I ignite_kernel` to boot and load HP-UX 11i on the other disks. In this paper I first load HP-UX 11i on all disks before creating vPars. In our upcoming example the first Virtual Partition will be created on the internal disk on an rp 54xx (formerly know as L-Class) system. The second Virtual Partition will be created on an external disk. After loading HP-UX 11i on one of the root volumes, I issued `uname`, which resulted in the following output:

```
# uname -a
HP-UX cvhdcon3 B.11.11 U 9000/800 136414696 unlimited-user license
#
```

The hostname of *cvhdcon3* and HP-UX revision *11.11*, which is the latest HP-UX 11i available at the time of this writing, are shown.

An interesting nuance to working with vPars is the naming of hosts and vPars. In a nutshell, you supply hostnames when installing 11i and Virtual Partition names when creating vPars. It reduces confusion if both the hostname and vPar name are the same for an instance of HP-UX 11i. In some cases, however, organizations require hostnames to conform to conventions that result in names that are difficult to remember. In this case some system administrators pick easy-to-remember vPar names. In the upcoming example we used different names for the vPars and hostnames.

Our upcoming examples have hostnames of *cvhdcon3* and *cvhdcon4*. The respective vPar names used are *cable1* and *cable2*.

2) Load the Virtual Partitions Application Software

The Virtual Partitions software must also be loaded on the volumes that will be used to host all vPars. At the time of this writing there are *base* and *full* versions of the software. The restrictions of the *base* software are a maximum of two vPars and a maximum of one CPU in one of the vPars. After loading the vPars software on one of the root volumes I ran **swlist** which resulted in the following output:

```
# swlist
# Initializing...
# Contacting target "cvhdcon3"...
#
# Target: cvhdcon3:/
#
#
# Bundle(s) :
#
BUNDLE11i          B.11.11.0102.2 Required Patch Bundle for HP-UX 11i, Febr
uary 2001
CDE-English        B.11.11          English CDE Environment
FDDI-00            B.11.11.01      PCI FDDI;Supptd HW=A3739A/A3739B;SW=J3626
AA
FibrChanl-00      B.11.11.06      PCI/HSC FibreChannel;Supptd HW=A6684A,A66
85A,A5158A
GigEther-00       B.11.11.14      PCI/HSC GigEther;Supptd HW=A4926A/A4929A/
A4924A/A4925A;SW=J1642AA
HPUX11i-OE-MC     B.11.11.0106    HP-UX Mission Critical Operating Environm
ent Component
HPUXBase64        B.11.11          HP-UX 64-bit Base OS
HPUXBaseAux       B.11.11.0106    HP-UX Base OS Auxiliary
HWEEnable11i     B.11.11.0106.8  Hardware Enablement Patches for HP-UX 11i
, June 2001
OnlineDiag        B.11.11.03.08   HPUX 11.11 Support Tools Bundle, Jun 2001

RAID-00           B.11.11.01      PCI RAID; Supptd HW=A5856A
VPARSBASE         A.01.00.03      HP-UX Virtual Partitions
#
```

The *HP-UX Virtual Partitions* software is the last entry shown in this listing.

3) Gather the System Component and Hardware Paths

You get to know your hardware at an intimate level when working with vPars. You not only need to know the components of which your system is comprised, you also need to know the paths of much of the hardware. Some system components, such as *System Bus Adapters* and the *memory controller*, are shared among vPars, so you don't specify those components as part of individual Virtual Partitions. Most other components in your system, such as processors, I/O cards, disks, and others, are fixed to specific vPars.

In order to see the components of which our example system is comprised, we'll run **ioscan -f** and **dmesg** in the following listing:

```
# ioscan -f
Class      I  H/W Path      Driver      S/W State  H/W Type    Description
-----
root       0
ioa        0  0              sba         CLAIMED    BUS_NEXUS   System Bus
Adapter (803)
ba         0  0/0           lba         CLAIMED    BUS_NEXUS   Local PCI Bus
Adapter (782)
lan        0  0/0/0/0       btlan       CLAIMED    INTERFACE   HP PCI 10/100
Base-TX Core
ext_bus    0  0/0/1/0       c720        CLAIMED    INTERFACE   SCSI C896
Ultra Wide Single-Ended
target     0  0/0/1/0.1     tgt         CLAIMED    DEVICE
disk       0  0/0/1/0.1.0   sdisk       CLAIMED    DEVICE      HP DVD-ROM 304
target     1  0/0/1/0.3     tgt         CLAIMED    DEVICE
tape       0  0/0/1/0.3.0   stape       CLAIMED    DEVICE      HP          C1537A
target     2  0/0/1/0.7     tgt         CLAIMED    DEVICE
ctl        0  0/0/1/0.7.0   sctl        CLAIMED    DEVICE      Initiator
ext_bus    1  0/0/1/1       c720        CLAIMED    INTERFACE   SCSI C896
Ultra Wide Single-Ended
target     3  0/0/1/1.0     tgt         CLAIMED    DEVICE
disk       1  0/0/1/1.0.0   sdisk       CLAIMED    DEVICE      SEAGATE ST17340 4LC
target     4  0/0/1/1.2     tgt         CLAIMED    DEVICE
disk       2  0/0/1/1.2.0   sdisk       CLAIMED    DEVICE      SEAGATE ST17340 4LC
target     5  0/0/1/1.7     tgt         CLAIMED    DEVICE
ctl        1  0/0/1/1.7.0   sctl        CLAIMED    DEVICE      Initiator
ext_bus    2  0/0/2/0       c720        CLAIMED    INTERFACE   SCSI C87x
Ultra Wide Single-Ended
target     6  0/0/2/0.0     tgt         CLAIMED    DEVICE
disk       3  0/0/2/0.0.0   sdisk       CLAIMED    DEVICE      SEAGATE ST17340 4LC
target     7  0/0/2/0.2     tgt         CLAIMED    DEVICE
disk       4  0/0/2/0.2.0   sdisk       CLAIMED    DEVICE      SEAGATE ST17340 4lc
target     8  0/0/2/0.7     tgt         CLAIMED    DEVICE
ctl        2  0/0/2/0.7.0   sctl        CLAIMED    DEVICE      Initiator
ext_bus    3  0/0/2/1       c720        CLAIMED    INTERFACE   SCSI C87x
Ultra Wide Single-Ended
target     9  0/0/2/1.7     tgt         CLAIMED    DEVICE
ctl        3  0/0/2/1.7.0   sctl        CLAIMED    DEVICE      Initiator
tty        0  0/0/4/0       asio0       CLAIMED    INTERFACE   PCI Serial
tty        1  0/0/5/0       asio0       CLAIMED    INTERFACE   PCI Serial
ba         1  0/1           lba         CLAIMED    BUS_NEXUS   Local PCI Bus
Adapter (782)
ba         2  0/2           lba         CLAIMED    BUS_NEXUS   Local PCI Bus
Adapter (782)
ba         3  0/3           lba         CLAIMED    BUS_NEXUS   Local PCI Bus
Adapter (782)
ba         4  0/4           lba         CLAIMED    BUS_NEXUS   Local PCI Bus
```

ba	5	0/5	lba	CLAIMED	BUS_NEXUS	Adapter (782) Local PCI Bus
ba	6	0/8	lba	CLAIMED	BUS_NEXUS	Adapter (782) Local PCI Bus
fc	0	0/8/0/0	td	CLAIMED	INTERFACE	HP Tachyon TL/TS Fibre Channel Mass Storage Adapter
fcp	0	0/8/0/0.8	fcp	CLAIMED	INTERFACE	FCP Protocol Adapter
ext_bus	4	0/8/0/0.8.0.5.0	fcparray	CLAIMED	INTERFACE	FCP Array Interface
target	10	0/8/0/0.8.0.5.0.0	tgt	CLAIMED	DEVICE	
disk	5	0/8/0/0.8.0.5.0.0.0	sdisk	CLAIMED	DEVICE	HP A5277A
disk	6	0/8/0/0.8.0.5.0.0.1	sdisk	CLAIMED	DEVICE	HP A5277A
disk	7	0/8/0/0.8.0.5.0.0.2	sdisk	CLAIMED	DEVICE	HP A5277A
disk	8	0/8/0/0.8.0.5.0.0.3	sdisk	CLAIMED	DEVICE	HP A5277A
target	11	0/8/0/0.8.0.5.0.1	tgt	CLAIMED	DEVICE	
disk	9	0/8/0/0.8.0.5.0.1.0	sdisk	CLAIMED	DEVICE	HP A5277A
target	12	0/8/0/0.8.0.5.0.2	tgt	CLAIMED	DEVICE	
disk	10	0/8/0/0.8.0.5.0.2.0	sdisk	CLAIMED	DEVICE	HP A5277A
target	13	0/8/0/0.8.0.5.0.3	tgt	CLAIMED	DEVICE	
disk	11	0/8/0/0.8.0.5.0.3.0	sdisk	CLAIMED	DEVICE	HP A5277A
ext_bus	5	0/8/0/0.8.0.255.0	fcpdev	CLAIMED	INTERFACE	FCP Device Interface
target	14	0/8/0/0.8.0.255.0.5	tgt	CLAIMED	DEVICE	
ctl	4	0/8/0/0.8.0.255.0.5.0	sctl	CLAIMED	DEVICE	HP A5277A
ba	7	0/9	lba	CLAIMED	BUS_NEXUS	Local PCI Bus Adapter (782)
fc	1	0/9/0/0	td	CLAIMED	INTERFACE	HP Tachyon TL/TS Fibre Channel Mass Storage Adapter
fcp	1	0/9/0/0.8	fcp	CLAIMED	INTERFACE	FCP Protocol Adapter
ext_bus	6	0/9/0/0.8.0.4.0	fcparray	CLAIMED	INTERFACE	FCP Array Interface
target	15	0/9/0/0.8.0.4.0.0	tgt	CLAIMED	DEVICE	
disk	12	0/9/0/0.8.0.4.0.0.0	sdisk	CLAIMED	DEVICE	HP A5277A
disk	13	0/9/0/0.8.0.4.0.0.1	sdisk	CLAIMED	DEVICE	HP A5277A
disk	14	0/9/0/0.8.0.4.0.0.2	sdisk	CLAIMED	DEVICE	HP A5277A
disk	15	0/9/0/0.8.0.4.0.0.3	sdisk	CLAIMED	DEVICE	HP A5277A
target	16	0/9/0/0.8.0.4.0.1	tgt	CLAIMED	DEVICE	
disk	16	0/9/0/0.8.0.4.0.1.0	sdisk	CLAIMED	DEVICE	HP A5277A
target	17	0/9/0/0.8.0.4.0.2	tgt	CLAIMED	DEVICE	
disk	17	0/9/0/0.8.0.4.0.2.0	sdisk	CLAIMED	DEVICE	HP A5277A
target	18	0/9/0/0.8.0.4.0.3	tgt	CLAIMED	DEVICE	
disk	18	0/9/0/0.8.0.4.0.3.0	sdisk	CLAIMED	DEVICE	HP A5277A
ext_bus	7	0/9/0/0.8.0.255.0	fcpdev	CLAIMED	INTERFACE	FCP Device Interface
target	19	0/9/0/0.8.0.255.0.4	tgt	CLAIMED	DEVICE	
ctl	5	0/9/0/0.8.0.255.0.4.0	sctl	CLAIMED	DEVICE	HP A5277A
ba	8	0/10	lba	CLAIMED	BUS_NEXUS	Local PCI Bus Adapter (782)
lan	1	0/10/0/0	btlan	CLAIMED	INTERFACE	HP A5230A/ B5509BA PCI 10/100Base-TX Addon
ba	9	0/12	lba	CLAIMED	BUS_NEXUS	Local PCI Bus Adapter (782)
lan	2	0/12/0/0	btlan	CLAIMED	INTERFACE	HP A5230A/ B5509BA PCI 10/100Base-TX Addon
psc	0	32	psc	CLAIMED	BUS_NEXUS	Bus Converter
processor	0	33	processor	CLAIMED	PROCESSOR	Processor
psc	1	36	psc	CLAIMED	BUS_NEXUS	Bus Converter
processor	1	37	processor	CLAIMED	PROCESSOR	Processor
psc	2	96	psc	CLAIMED	BUS_NEXUS	Bus Converter
processor	2	97	processor	CLAIMED	PROCESSOR	Processor
psc	3	100	psc	CLAIMED	BUS_NEXUS	Bus Converter
processor	3	101	processor	CLAIMED	PROCESSOR	Processor
memory	0	192	memory	CLAIMED	MEMORY	Memory
#						

```
# dmesg

Jul 31 20:03
gate64: sysvec vaddr = 0xc0002000 for 2 pages
NOTICE: nfs3_link(): File system was registered at index 3.
NOTICE: autofs_link(): File system was registered at index 6.
NOTICE: cachefs_link(): File system was registered at index 7.
0 sba
0/0 lba
0/0/0/0 btlan
0/0/1/0 c720
0/0/1/0.1 tgt
0/0/1/0.1.0 sdisk
0/0/1/0.3 tgt
0/0/1/0.3.0 stape
0/0/1/0.7 tgt
0/0/1/0.7.0 sctl
0/0/1/1 c720
0/0/1/1.0 tgt
0/0/1/1.0.0 sdisk
0/0/1/1.2 tgt
0/0/1/1.2.0 sdisk
0/0/1/1.7 tgt
0/0/1/1.7.0 sctl
0/0/2/0 c720
0/0/2/0.0 tgt
0/0/2/0.0.0 sdisk
0/0/2/0.2 tgt
0/0/2/0.2.0 sdisk
0/0/2/0.7 tgt
0/0/2/0.7.0 sctl
0/0/2/1 c720
0/0/2/1.7 tgt
0/0/2/1.7.0 sctl
0/0/4/0 asio0
0/0/5/0 asio0
0/1 lba
0/2 lba
0/3 lba
0/4 lba
0/5 lba
0/8 lba
0/8/0/0 td
td: claimed Tachyon TL/TS Fibre Channel Mass Storage card at 0/8/0/0
0/8/0/0.8 fcp
0/8/0/0.8.0.5.0 fcparray
0/8/0/0.8.0.5.0.0 tgt
0/8/0/0.8.0.5.0.0.0 sdisk
0/8/0/0.8.0.5.0.0.1 sdisk
0/8/0/0.8.0.5.0.0.2 sdisk
0/8/0/0.8.0.5.0.0.3 sdisk
0/8/0/0.8.0.5.0.1 tgt
0/8/0/0.8.0.5.0.1.0 sdisk
0/8/0/0.8.0.5.0.2 tgt
0/8/0/0.8.0.5.0.2.0 sdisk
0/8/0/0.8.0.5.0.3 tgt
0/8/0/0.8.0.5.0.3.0 sdisk
0/8/0/0.8.0.255.0 fcpdev
0/8/0/0.8.0.255.0.5 tgt
0/8/0/0.8.0.255.0.5.0 sctl
0/9 lba
0/9/0/0 td
td: claimed Tachyon TL/TS Fibre Channel Mass Storage card at 0/9/0/0
0/9/0/0.8 fcp
0/9/0/0.8.0.4.0 fcparray
0/9/0/0.8.0.4.0.0 tgt
0/9/0/0.8.0.4.0.0.0 sdisk
0/9/0/0.8.0.4.0.0.1 sdisk
0/9/0/0.8.0.4.0.0.2 sdisk
0/9/0/0.8.0.4.0.0.3 sdisk
0/9/0/0.8.0.4.0.1 tgt
0/9/0/0.8.0.4.0.1.0 sdisk
0/9/0/0.8.0.4.0.2 tgt
0/9/0/0.8.0.4.0.2.0 sdisk
0/9/0/0.8.0.4.0.3 tgt
```

```

0/9/0/0.8.0.4.0.3.0 sdisk
0/9/0/0.8.0.255.0 fcpdev
0/9/0/0.8.0.255.0.4 tgt
0/9/0/0.8.0.255.0.4.0 sctl
0/10 lba
0/10/0/0 btlan
0/12 lba
0/12/0/0 btlan
32 pbc
33 processor
36 pbc
37 processor
96 pbc
97 processor
100 pbc
101 processor
192 memory
btlan: Initializing 10/100BASE-TX card at 0/0/0/0....

System Console is on the Built-In Serial Interface
btlan: Initializing 10/100BASE-TX card at 0/10/0/0....
btlan: Initializing 10/100BASE-TX card at 0/12/0/0....
Entering cifs_init...
Initialization finished successfully... slot is 9
Logical volume 64, 0x3 configured as ROOT
Logical volume 64, 0x2 configured as SWAP
Logical volume 64, 0x2 configured as DUMP
Swap device table: (start & size given in 512-byte blocks)
  entry 0 - major is 64, minor is 0x2; start = 0, size = 8388608
Dump device table: (start & size given in 1-Kbyte blocks)
  entry 0000000000000000 - major is 31, minor is 0x12000; start = 117600,
size = 4194304
Starting the STREAMS daemons-phase 1
Create STCP device files
Starting the STREAMS daemons-phase 2
$Revision: vmunix: vw: -proj selectors: CUPI80_BL2000_1108 -c 'Vw
for CUPI80_BL2000_1108 build' -- cupi80_bl2000_1108 'CUPI80_BL2000_1108' Wed
Nov 8 19:24:56 PST 2000 $
Memory Information:
  physical page size = 4096 bytes, logical page size = 4096 bytes
  Physical: 4194304 Kbytes, lockable: 3231756 Kbytes, available: 3711728 Kbytes
#

```

The output of **ioscan -f** and **dmesg** provide a lot of useful information about our system. We'll use the components and paths in **ioscan** output and the memory information in **dmesg** to create a list of components for the respective vPars in the upcoming step. We now know, for instance, that the paths of two of the LAN cards are at *0/0/0/0* and *0/10/0/0*. We know the paths of all four processors of *33*, *37*, *97*, and *101*. The console is located at *0/0/4/0*. From the **dmesg** output we know that we have a total of four GBytes of RAM that can be spread among the vPars.

From these two outputs we have the information we need to create the Virtual Partitions in the next step.

4) List the Components of the Virtual Partitions

From the **ioscan** and **dmesg** messages we can select the components of our first Virtual Partition. The following is a list of components we'll include in this partition:

First vPar *cable1*

```

name          cable1
processors    min of one (bound) max of three (two unbound)
              with num (bound + unbound) equal to one
memory        1024 MB
LBA Core I/O  0/0 (all components on 0/0 are implied)
LAN           0/0/0/0 (not specified explicitly, on 0/0)
boot disk     0/0/1/1.2.0
kernel        /stand/vmunix (this is default)
autoboot      off (manual)
console       0/0/4/0 (not specified explicitly, on 0/0)

```

You may want to set *autoboot* to *auto* during installation and set to *manual* after installation. This makes booting easier during installation.

Some of the components require some explanation concerning the way in which they are implemented with vPars. The following is a more detailed discussion of some of these components, including CPU, memory, and LAN, bootdisk, setboot, kernel, and console.

CPU

The CPUs used in both this partition (*cable1*) and the one we will define shortly (*cable2*) are specified with *min*, *max*, and *num*. We will have *min bound* CPUs that have I/O interrupts assigned to them and are therefore ideal for I/O-intensive applications. The additional CPUs assigned to the vPars are *unbound*, which do not process I/O interrupts. Therefore, *unbound* CPUs are ideal for processor-intensive applications as opposed to I/O-intensive applications. *unbound* CPUs can be freely moved from one vPar to another while vPars are running, so having *min* bound CPUs gives us the freedom to move

around the *unbound* CPUs. *Bound* CPUs can also be added to and deleted from Virtual Partitions only when the partition is down.

On machines that employ Non-Uniform Memory Access (NUMA) you would use hardware paths (*path*) to specify CPUs. This is to ensure that you minimize the distance between CPUs and memory. On systems such as the rp 7400 (formerly know as N-Class) and rp 54xx (formerly know as L-Class) that do not employ NUMA, *min* is recommended to define *bound* CPUs.

For our work on *cable1* and *cable2* and for my work with vPars in general, the most common desire is to have a *min* number of *bound* CPUs in all vPars and then move around *unbound* CPUs as the applications in vPars need them. For instance, when **vparcreate** is run we would specify the following:

```
vparcreate -p cable1 -a cpu:::1:3 -a cpu:::1
```

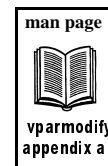
At the time of creation *cable1* will have one *bound* CPU only because we specified a *min* of one and a *num* of one. *num* is the total *bound* + *unbound* CPUs, and since we specified one for *num*, we'll get one *bound* CPU (I have seen some vPars material use *num* and some use *total* so *num* and *total* are interchangeable in this book.) Since *max* is three we have left the door open to add as many as two additional *unbound* CPUs. If we have two *unbound* CPUs on our system, we can move them among the vPars as required with **vparmodify**. To remove the two *unbound* CPUs from *cable2* and add them to *cable1*, we would issue the two following **vparmodify** commands:

```
vparmodify -p cable2 -m cpu:::1    <-- reduces cable2 from 3 to 1
```

```
vparmodify -p cable1 -m cpu:::3    <-- increases cable1 from 1 to 3
```

We first removed the two *unbound* CPUs from *cable2* and then added them to *cable1*. If the two *unbound* CPUs were not assigned to a vPar, we would not have to remove them from *cable2* prior to adding them to *cable1*.

There are many ways to work with CPUs, so by characterizing your applications and understanding the options for using *bound* and *unbound* CPUs, you can use the processor mix that best meets your needs.



Memory

We have identified one GByte of memory for *cab1e1*. Memory can be specified by *range* or *size*.

To add one GByte of memory to *cab1e1* using *size* we would use the following **vparcreate** command:

```
vparcreate -p cab1e1 -a mem::1024
```



This **vparcreate** command specifies only the memory for use in *cab1e1*. The full **vparcreate** command for creating *cab1e1* will be shown in an upcoming section.

The memory is specified in MBytes (1024 MBytes = 1 GByte) in multiples of 64 MBytes. At the time of this writing, the Virtual Partition Monitor consumes roughly 128 MBytes of RAM, so this will not be available to allocate to a Virtual Partition. Modifying memory allocation requires that the Virtual Partition be down, at the time of this writing.

On machines that employ Non-Uniform Memory Access (NUMA) you would use the *range*. *Range* is a subset of *size*. On systems such as the rp 7400 (formerly know as N-Class) and rp 54xx (formerly know as L-Class) that do not employ NUMA, the *size* is recommended to define memory. The syntax for specifying memory by range is as follows:

```
mem:::base:range
```

None of the examples in this book were prepared using NUMA systems so you won't see any examples using the *range* syntax; all examples use the *size* syntax.

LAN

The LAN interface used for this first Virtual Partition is on Local Bus Adapter (LBA) zero. This means that any other components on LBA zero would have to be in this Virtual Partition as well. At the time of this writing, components on an LBA can't be shared between vPars.

Note that we have decided not to use the hostname as our Virtual Partition name. As mentioned earlier, it is desirable to use the same name for the hostname and vPar. Because our hostnames are a little hard to remember the system administrator decided to use simple vPar names. When we loaded HP-UX 11i on the system we selected the hostnames (you can also run **set_parms** after loading 11i to set the system name and other parameters) of *cvhdcon3* and *cvhdcon4*. We then chose the simple vPar names of *cable1* and *cable2*, respectively.

Boot Disk

The **ioscan** issued earlier in this paper showed many disk devices. The boot device for our first Virtual Partition is the internal disk with the hardware path *0/0/1/1.2.0*.

At the time of this writing, components that are at or below the Local Bus Adapter (LBA) level are devoted to a single Virtual Partition. This means that although the output of our earlier **ioscan** command shows four internal disks, all four of these disks must be in the same Virtual Partition because they are on the same LBA.

Kernel

We'll use the default HP-UX kernel of */stand/vmunix* for the kernel in this Virtual Partition. Since we're using the default kernel, we don't have to specify this as part of the **vparcreate** command; however, we'll include it in the **vparcreate** command for completeness purposes.



setboot Command

In our example we have *autoboot* set to *off* for our Virtual Partition. The **setboot** command on a non-vPars system reads from and writes to stable storage. On a vPars system the **setboot** command interacts with the Virtual Partition database. In our upcoming example we'll set the *autoboot* to *off* when we create *cable1* with **vparcreate**. Running **setboot** on a vPars system has the effects shown in Table 1-2:

Table 1-2 `setboot` and Virtual Partitions

vPars <code>setboot</code> Option	Description
<code>-a</code>	Changes the alternate boot path of the Virtual Partition. To set the alternate boot path: # <code>setboot -a 0/8/0/0.8.0.5.0.0.0</code>
<code>-b</code>	Sets the autoboot attribute of the Virtual Partition. To set Autoboot <i>on</i> : # <code>setboot -b on</code>
<code>-p</code>	Changes the primary boot path of the Virtual Partition. To set the primary boot path: # <code>setboot -p 0/0/1/1.2.0</code>
<code>-s</code>	Has no effect.
no options	Displays information about boot attributes.

The `setboot` command is one of the aspects of working with vPars that is different from a non-vPars system.

Console

Chapter 5 of the vPars book contains detailed information on the way in which the console operates in a vPars environment. In our first partition we have specified LBA `0/0` as a component of vPar `cabl1`. Since the physical console at `0/0/4/0` is on the Core I/O card at `0/0`, it is an implied component of `cabl1` and we do not have to specify the physical console in this partition. The other Virtual Partitions on this system will use the virtual console functionality of vPars whereby issuing `Ctrl-A` cycles between virtual console displays.

Database

The Virtual Partition database that contains all vPar-related information is `/stand/vpdb`. This is managed, and synchronized for you, so you don't need to pay too much attention to it if you don't want to. You can, however; create

an alternate database if you wish. You may want to do this in order to create a completely different Virtual Partition configuration for your system without affecting your currently running database.

When creating Virtual Partitions with **vparcreate** you can use the **-D** option and specify an alternate database name that is a file in the **/stand** directory, such as **/stand/vpdb.app2**. When you boot vPars from this database (with `ISL> hpux /stand/vpmon -D db_file`) it is the default, so all modifications made to vPars defined in this database are made to it rather than the default.



Second vPar *cable2*

We'll list the same categories of components for *cable2* as we did for *cable1* in the following list:

```

name          cable2
processors    min of one (bound) max of three (two unbound)
              with num (bound + unbound) equal to one
memory        1024 MB
LAN           0/10/0/0
boot disk     0/8/0/0.8.0.5.0.0.0
kernel        /stand/vmunix (this is the default)
autoboot      off (manual)
console       virtual console to be created

```

We now have a list of components for two vPars. The result is that our rp 54xx (formerly know as L-Class) system has been divided into two vPars that look like Figure 1-5:

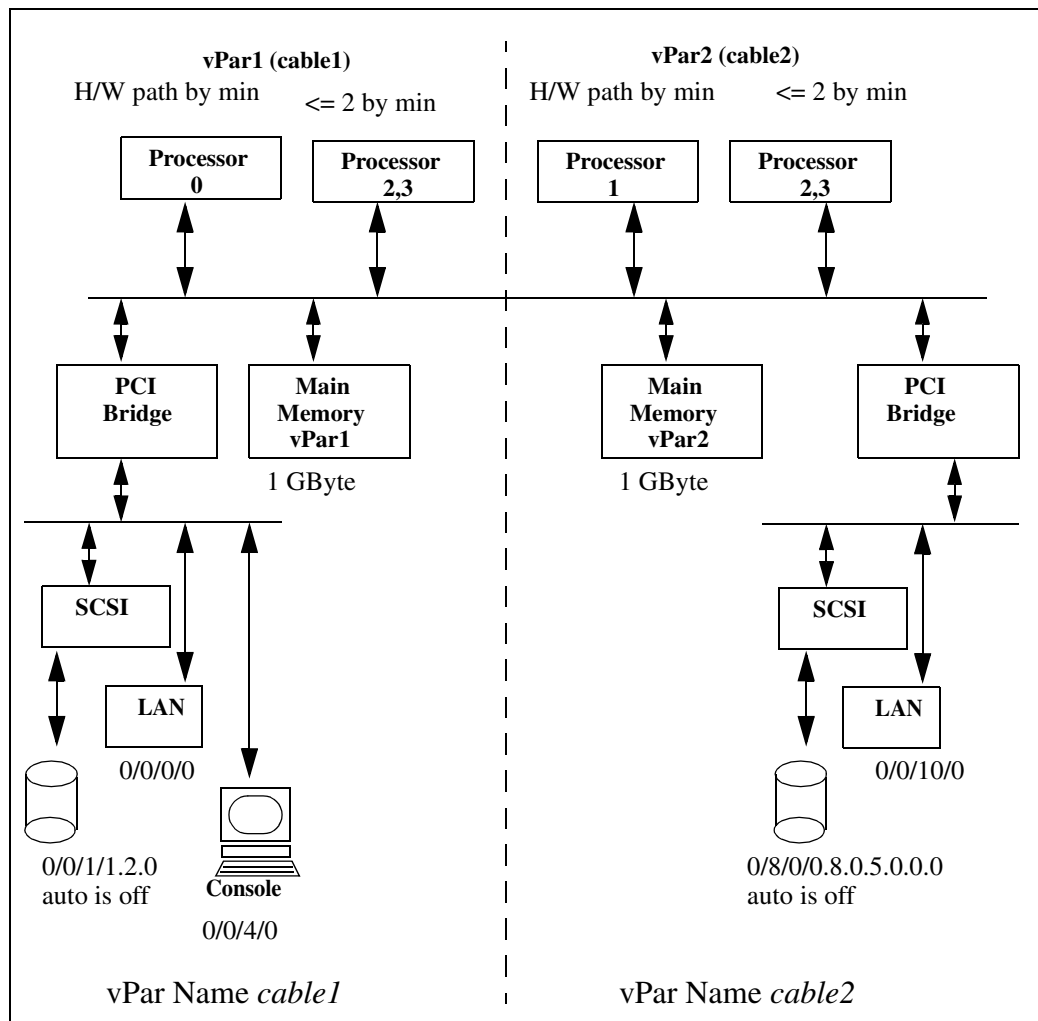


Figure 1-5 rp 54xx (formerly know as L-Class) with Two vPars (unused components not shown)

Figure 1-5 reflects what our system will look like when we perform the upcoming steps to create our two vPars. Note that two *unbound* processors, shown as 2,3 in Figure 1-5, can be assigned to *cable1* and *cable2* as required.

5) Virtual Partition Kernel-Related Work

Each Virtual Partition has its own instance of HP-UX 11i, which has its own HP-UX kernel. It is likely that you'll customize these kernels in a variety of ways to suit the applications you have running in the respective vPars. When you install the vPars software, it automatically reconfigures the kernel to include the vPar drivers and make the kernel relocatable. You do not have to perform the kernel-related steps in this section because they are performed for you when vPars software is loaded. It is still informative; however, to see the steps that were manually performed in this section to get better insight concerning the way vPars operate. In this step we'll investigate the files that have been updated by the vPars application and build the new kernel. Keep in mind that the new kernel needs to be built on every volume that has HP-UX 11i on it and will run a vPar.

Because memory is shared among multiple vPars, the kernel must be relocatable in memory. At the time of this writing, there are patches that allow the kernel to be built as a relocatable kernel. We won't perform any checks related to patches.

The file **/sbin/vecheck** is a vPar file that is required on the system. The following listing is a portion of **/usr/conf/gen/config.sys** that checks to see if **/sbin/vecheck** has been loaded on the system:

```
# Determine whether the linker supports kernel relocation.  If it does,
# link the kernel using the relocation options.
LOADOPTS_ADDL=~ \
    if [ -f /sbin/vecheck ]; then \
        ${WHAT} ${LD} | \
        ${AWK} '$$0 ~ /92453-07 linker/ { \
            split($$7, vers, "."); \
            if ( vers[1] == "B" && \
                ( vers[2] == 11 && vers[3] >= 25 ) || vers[2] > 11 ) \
                print "${LOADOPTS_RELOC}"; \
            else print "${LOADOPTS_STATIC}"; }'; \
    else \
        echo "${LOADOPTS_STATIC}"; \
    fi; \
\
```

The following is a long listing of **/sbin/vecheck** which was loaded with the vPar software:

```
# ll /sbin/vecheck
-r-xr-xr-x 1 bin bin 20533 Mar 5 19:01 /sbin/vecheck
#
```

Next let's take a look at `/stand/system` to see the `vpar` driver that has been added to the file:

```
# cat /stand/system
*****
* Source: /ux/core/kern/filesets.info/CORE-KRN/generic
* @(#)B.11.11_LR
*
*****
* Additional drivers required in every machine-type to
* create a complete
* system file during cold install. This list is every driver that the
* master.d/ files do not force on the system or is not identifiable by
* ioscan.
* Other CPU-type specific files can exist for their special cases.
* see create_sysfile (1m).
*****
*
* Drivers/Subsystems
sba
lba
c720
sctl
sdisk
asio0
cdfb
cxperf
olar_psm
olar_psm_if
dev_olar
diag0
diag1
diag2
dmem
dev_config
iomem
nfs_core
nfs_client
nfs_server
btlan
maclan
dlpi
token_arp
inet
uipc
tun
telm
tels
netdiag1
nms
```



```

hpstreams
clone
strlog
sad
echo
sc
timod
tirdwr
pipedev
pipemod
ffs
ldterm
ptem
pts
ptm
pckt
td
fddi4
gelan
GSctoPCI
iop_drv
bs_ism
vxfs
vxportal
lvm
lv
nfsm
rpcmod
autofsc
cachefsc
cifs
prm
vpar          <--- vpar driver added here
STRMSGSZ      65535
nstrpty       60
dump lvvol
maxswapchunks 2048
#

```

The *vpar* driver is a master driver described in `/usr/conf/master.d` as shown below:

```

# pwd
/usr/conf/master.d
# cat vpar
$CDIO
vpar          0
$$$

$DRIVER_INSTALL
vcn           -1          209
vcs           -1          -1

```

```

vpar_driver      -1          -1
$$$

$DRIVER_DEPENDENCY
vcn              vpar
vcs              vpar
vpar             vcs vcn vpar_driver
vpar_driver     vpar
$$$

$DRIVER_LIBRARY
*
* The driver/library table. This table defines which libraries a given
* driver depends on. If the driver is included in the dfile, then the
* libraries that driver depends on will be included on the ld(1) command
* line. Only optional libraries *need* to be specified in this table,
* (but required ones can be included, as well).
*
* Driver handle      <libraries>
*
* subsystems first
vcn                  libvpar-pdk.a
vcs                  libvpar-pdk.a
vpar                 libvpar-pdk.a
vpar_driver         libvpar-pdk.a
$$$

$LIBRARY
*
* The library table. Each element in the library table describes
* one unique library. The flag member is a boolean value, it is
* initialized to 1 if the library should *always* be included on
* the ld(1) command line, or 0 if the library is optional (i.e. it
* is only included when one or more drivers require it). The order
* of the library table determines the order of the libraries on the
* ld(1) command line, (i.e. defines an implicit load order). New
* libraries must be added to this table.
* Note: libhp-ux.a must be the last entry, do not place
* anything after it.
*
* Library      <required>
*
libvpar-pdk.a      0
$$$
#

```

You can see in this file there are multiple drivers present. The *vcn* and *vcs* drivers are used to support the console in a vPars environment. Since you'll probably only have one physical console for multiple partitions, you need a way to share the physical device. The use of these drivers is described in Chapter 4 in the vPars book in which kernel configuration is covered. For now it is sufficient to know that these drivers exist as part of the vPars installation and must be built into the kernel.

Now that the kernel has what it needs to be built relocatable and the drivers are present for vPars, we can run **mk_kernel** to build the new kernel and **kmupdate** to move the new kernel-related files into place. This is done automatically for you, but the following commands show how you would perform this process:

```
# mk_kernel
Generating module: krm...
Compiling /stand/build/conf.c...
Loading the kernel...
Generating kernel symbol table...
# kmupdate

Kernel update request is scheduled.

Default kernel /stand/vmunix will be updated by
newly built kernel /stand/build/vmunix_test
at next system shutdown or startup time.

#
```

Keep in mind that this procedure needs to be performed for all HP-UX 11i operating systems that will run a Virtual Partition.

6) Create the First Virtual Partition

The **vparcreate** command is used to create a vPar. The summary of this command is shown in Table 1-1 and its man page appears in Appendix A. The general form of the command is as follows:



```
vparcreate -p vp_name [-B boot_attr] [-D db_file] [-S static_attr]
[-b kernel_path] [-o boot_opts] [-a rsrc] [-a...]
```

When creating this vPar, I placed the **vparcreate** command in a file so that I could modify it for the second vPar and execute it. The **vparcreate** command is shown below:

```
# cat /tmp/cable1
vparcreate -p cable1 -B manual -b /stand/vmunix -a cpu::1
-a cpu:::1:3 -a mem::1024 -a io:0/0 -a io:0/0/1/1.2.0:boot
#
```

After changing the permissions on this file and running it, the vPar *cable1* was successfully created. Next we'll boot the vPar we just created.

7) Boot the First Virtual Partition

Now that the first vPar has been created and the kernel automatically rebuilt to support vPars, we can boot the first vPar which we named *cable1*.

We'll both boot off the first vPar and check its status. We need to load the Virtual Partition Monitor (**vpmon**) at the *ISL>* prompt. **vpmon** is a *ram-disk* kernel, similar to *vmunix*, that needs to be loaded at the time of boot. From the *ISL>* prompt we are going to run **vpmon** to get the *MON>* prompt. From the *MON>* prompt we boot our Virtual Partition with **vparload**, as shown in the following example:

```
ISL> hpux /stand/vpmon

Boot
: disk(0/0/1/1.2.0.0.0.0.0;0)/stand/vpmon
421888 + 142056 + 4247112 start 0x23000
cable1: WARNING: No boot device specified

Welcome to VPMON (type '?' for a list of commands)

MON> vparload -p cable1

[MON] Console client set to cable1

[MON] cable1 loaded
```

·
·

You may see messages different from those shown in the example after the **vparload** command was issued. In any event, the system progressed through the remainder of the boot process and booted the one Virtual Partition *cab1e1* that we created. We now have a subset of the system components dedicated to this Virtual Partition.

The **vparload** command has the following three forms:

form1: `vparload -all`

form2: `vparload -auto`

form3: `vparload -p vp_name [-b kernelpath] [-o boot_options]
[-B hardware_path]`

We issued the third form shown above.

Now that the partition has booted, let's first obtain the status of the one Virtual Partition we created, called *cab1e1*, that we have running:

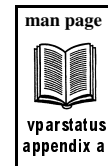
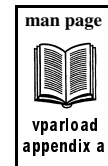
```
# vparstatus -p cab1e1 -v

[Virtual Partition Details]
Name:          cab1e1
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
Min/Max:       1/3
Bound by User [Path]:
Bound by Monitor [Path]: 33
Unbound [Path]:

[IO Details]
0.0
0.0.1.1.2.0 BOOT

[Memory Details]
Specified [Base /Range]:
              (bytes) (MB)
Total Memory (MB): 1024
#
```





The output of **vparstatus** shows that *cabl1* is *up*. The *-v* option is used to obtain a verbose output. You can see from this listing that the bound CPU at hardware path *33* (the *bound* CPU we specified with the *min*) is part of the partition, that there is one GByte of memory in the partition, and that the I/O components we specified are in the partition. Had there been other partitions configured, we would have seen their output as well.

Note that the console at *0/0/4/0* is an implied component of this vPar. So too is the LAN interface at *0/0/0/0*. Both of these components are part of the Core I/O card that we specified as part of *cabl1* with the *-a io:0/0* argument to the **vparcreate** command.

We can now run **vparstatus -A** to view the available components of our system. Since we created a first partition with only one CPU, we should see three CPUs and many other system components available, as shown in the following listing:



```
# vparstatus -A

[Unbound CPUs (path)]:  37
                        97
                        101

[Available CPUs]:      3

[Available I/O devices (path)]:  0.1
                                   0.2
                                   0.3
                                   0.4
                                   0.5
                                   0.8
                                   0.9
                                   0.10
                                   0.12
                                   32
                                   36
                                   96
                                   100

[Unbound memory (Base /Range)]:  0x0/128
                                   (bytes) (MB)  0xc000000/1856
                                                0x180000000/1088

[Available memory (MB)]:  3072
#
```

This output shows many components available for our second partition. Based on our earlier planning exercise, we know the components that we wish to include in the second vPar, and this **vparstatus -A** command confirms that they are indeed available.

For *cable2* we want one CPU initially, and there are three available. We want the I/O cards for boot and LAN at *0/8* and *0/10* respectively. We want one GByte of memory and there are now roughly three GBytes available. We have all of the components we need to proceed to our next step of creating *cable2*.

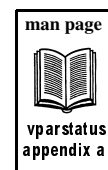
8) Create the Second Virtual Partition

We earlier listed all of the components of which our second partition is to be comprised and confirmed that these components are still available with the **vparstatus -A** command. HP-UX 11i has already been loaded on a second disk on the same system used to create our first Virtual Partition *cable1*. We can create our second Virtual Partition, which we'll call *cable2*. We'll create the second while the first is running and boot the second vPar from the first.

Here are the components we earlier listed for our second vPar:

```
name          cable2
processors    min of one (bound) max of three (two unbound)
              with num (bound + unbound) equal to one
memory        1024 MB
LAN           0/10
LBA           0/8
boot disk     0/8/0/0.8.0.5.0.0.0
kernel        /stand/vmunix (this is the default)
autoboot      off (manual)
console       virtual console to be created
```

There are several differences between the list of components for the two vPars. We have devoted a different LAN card and boot disk. We have specified the CPUs in the same manner in both vPars with *min* (this will be *bound*) of one and *max* of three, *num* of one, and let the vPars software identify the one *bound* processor. These two Virtual Partitions will use different





I/O paths for their devices. Let's now run the **vparcreate** command to create *cable2*:

We can now proceed to create the second partition with the command shown in the following file:

```
# cat /tmp/cable2
vparcreate -p cable2 -B manual -b /stand/vmunix -a cpu::1
-a cpu:::1:3 -a mem::1024 -a io:0/8
-a io:0/8/0/0.8.0.5.0.0.0:boot -a io:0/10
#
```



After executing this file we can determine if the second vPar has been created and the components of which it is comprised by running **vparstatus**:

```
# vparstatus -v

[Virtual Partition Details]
Name:          cable1
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
Min/Max:      1/3
Bound by User [Path]:
Bound by Monitor [Path]: 33
Unbound [Path]:

[IO Details]
0.0
0.0.1.1.2.0 BOOT

[Memory Details]
Specified [Base /Range]:
              (bytes) (MB)
Total Memory (MB): 1024

[Virtual Partition Details]
Name:          cable2
State:         Down
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
```



```
Min/Max: 1/3
Bound by User [Path]:
Bound by Monitor [Path]: 37
Unbound [Path]:

[IO Details]
 0.8
 0.8.0.0.8.0.5.0.0.0, BOOT
 0.10

[Memory Details]
Specified [Base /Range]:
      (bytes) (MB)
Total Memory (MB): 1024
#
```

This output shows that the first vPar is intact and that the second has been successfully created with the name, kernel file, CPU, I/O, and memory components we specified. Note that each vPar has one bound CPU assigned to it. The LAN card assigned to *cable2* appears in the output because we specified LBA *0/10* as one of the components of *cable2*. The console at *0/0/4/0* and the LAN interface at *0/0/0/0* are implied components of *cable1* and do not appear in the **vparstatus -v** output.

With the second vPar created, we can proceed to the next step and boot it.



9) Boot the Second Virtual Partition

Since we already have the first vPar running, called *cable1*, and the second vPar created, called *cable2*, we can boot the second vPar from the first. There are many options to boot vPars. Since we already have the first vPar running, we'll simply boot the second from the first with **vparboot** and then run **vparstatus -v** as shown in the following example. If we type subsequent **vparstatus** commands we can see the status of vPar *cable2* progress from *Load*, to *Boot* in the next output, and finally *Up* when the vPar is running, as shown in the following listing:



```

# vparboot -p cable2
vparboot: Booting cable2. Please wait...

# vparstatus
[Virtual Partition]

Virtual Partition Name      State Attributes Kernel Path      Boot
=====
cable1                      Up    Dyn,Manl  /stand/vmunix
cable2                      Load  Dyn,Manl  /stand/vmunix
=====

[Virtual Partition Resource Summary]

Virtual Partition Name      CPU      CPU      Num      Memory (MB)
                          Min/Max  Bound/  IO      # Ranges/
                          =====  =====  =====  =====
cable1                      1/ 3    1 0    4      0/ 0
cable2                      1/ 3    1 0    4      0/ 0

# vparstatus
[Virtual Partition]

Virtual Partition Name      State Attributes Kernel Path      Boot
=====
cable1                      Up    Dyn,Manl  /stand/vmunix
cable2                      Boot  Dyn,Manl  /stand/vmunix
=====

[Virtual Partition Resource Summary]

Virtual Partition Name      CPU      CPU      Num      Memory (MB)
                          Min/Max  Bound/  IO      # Ranges/
                          =====  =====  =====  =====
cable1                      1/ 3    1 0    4      0/ 0
cable2                      1/ 3    1 0    4      0/ 0

# vparstatus
[Virtual Partition]

Virtual Partition Name      State Attributes Kernel Path      Boot
=====
cable1                      Up    Dyn,Manl  /stand/vmunix
cable2                      Up    Dyn,Manl  /stand/vmunix
=====

[Virtual Partition Resource Summary]

Virtual Partition Name      CPU      CPU      Num      Memory (MB)
                          Min/Max  Bound/  IO      # Ranges/
                          =====  =====  =====  =====
cable1                      1/ 3    1 0    4      0/ 0
cable2                      1/ 3    1 0    4      0/ 0

#

```

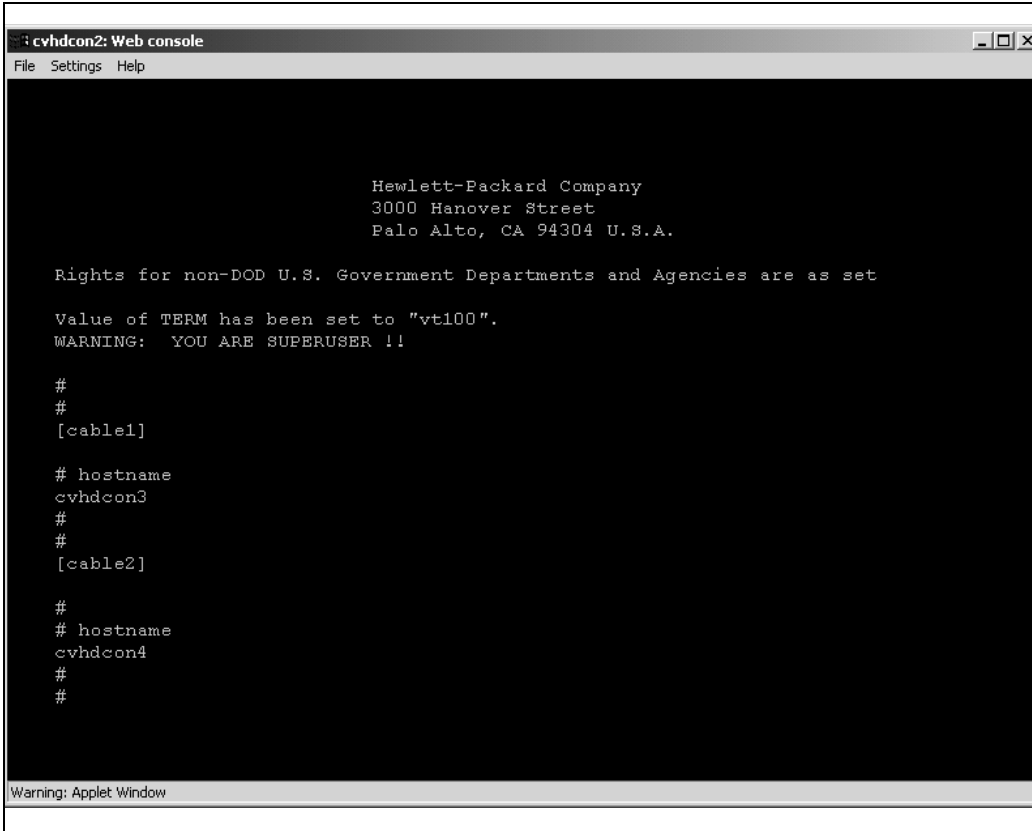
This progression of states of *cable2* reflects the time it takes to boot the operating system from the second volume on which this vPar is run.

In addition to *load*, *boot*, and *up*, there are other states in which you may find a Virtual Partition as well. Table 1-3 summarizes the states of Virtual Partitions at the time of this writing:

Table 1-3 Virtual Partitions States

vPars State	Description
<i>load</i>	The kernel image of a Virtual Partition is being loaded into memory. This is done by the Virtual Partition monitor.
<i>boot</i>	The Virtual Partition is in the process of booting. The kernel image has been successfully loaded by the Virtual Partition monitor.
<i>up</i>	The Virtual Partition has been successfully booted and is running.
<i>shut</i>	The Virtual Partition is in the process of shutting down.
<i>down</i>	The Virtual Partition is not running and is down.
<i>crash</i>	The Virtual Partition has experienced a panic and is crashing.
<i>hung</i>	The Virtual Partition is not responding and is hung.

With more than one vPar running, you would use the built-in vPars drivers to toggle the console between any number of Virtual Partitions using *Ctrl-A*. Figure 1-6 shows using the console to view *cable1* with a hostname of *cvhdcon3*. Issuing *Ctrl-A* connects to vPar *cable2* with a hostname of *cvhdcon4*. When you issue *Ctrl-A* to switch to the next vPar in the console you are supplied with the name of the vPar to which you have connected in brackets, such as [*cable1*].



```
cvhdcon2: Web console
File Settings Help

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set

Value of TERM has been set to "vt100".
WARNING: YOU ARE SUPERUSER !!

#
#
[cable1]

# hostname
cvhdcon3
#
#
[cable2]

#
# hostname
cvhdcon4
#
#

Warning: Applet Window
```

Figure 1-6 Console Shown Using *Ctrl-A* to Toggle Between vPars

In addition to using the console to switch between vPars, you can also use the LAN cards configured into the respective vPars to open a *TELNET* or other type of session to the vPars. This is the same technique you would use to connect to any system over the network and is one of the primary reasons you always want to have a LAN card configured as part of every vPar.

We did not cover the configuration of the two LAN cards, one in each vPar, in this paper. The LAN configuration would have to be completed for both vPars in order to use the networking cards for such operations as a *TELNET* session. Chapter 13 of the vPars book covers many networking topics, including the */etc/hosts* file; */etc/rc.config.d/netconf* file, which must be configured on each vPar; and many others.

10) Modify the Virtual Partition

It is likely that you'll want to modify your Virtual Partitions in a variety of ways. You may want to add or remove a CPU, for instance. Let's take a look at an example of adding a CPU to a Virtual Partition.

In the upcoming example there is a four-processor system on which there are the two Virtual Partitions we just created: *cabl1* and *cabl2*. Each vPar has one *bound* CPU that was assigned by *min* when the vPars were created. Let's run **vparstatus** to see the components of which these two Virtual Partitions are comprised and confirm that each has one bound CPU:



```
# vparstatus -p cabl1 -v

[Virtual Partition Details]
Name:          cabl1
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
Min/Max:       1/3
Bound by User [Path]:
Bound by Monitor [Path]: 33      <-- one bound CPU @ 33
Unbound [Path]:

[IO Details]
0.0
0.0.1.1.2.0  BOOT

[Memory Details]
Specified [Base /Range]:
              (bytes) (MB)
Total Memory (MB): 1024
#

# vparstatus -p cabl2 -v

[Virtual Partition Details]
Name:          cabl2
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
```

```

Boot Opts:

[CPU Details]
Min/Max: 1/3
Bound by User [Path]:
Bound by Monitor [Path]: 37    <-- one CPU in use at 37
Unbound [Path]:

[IO Details]
0.8.0.0.8.0.5.0.0.0 BOOT
0.10.0.0

[Memory Details]
Specified [Base /Range]:
      (bytes) (MB)
Total Memory (MB): 1024
#

```



The output of these two **vparstatus** commands shows that *cabl1* has one *bound* CPU and *cabl2* has one *bound* CPU. On the rp 54xx (formerly know as L-Class) system on which these vPars were created there are a total of four CPUs. This means that two CPUs should be available. Let's run **vparstatus -A** to view the available components on a system:

```

# vparstatus -A

[Unbound CPUs (path)]: 97
                        101
[Available CPUs]: 2

[Available I/O devices (path)]: 0.1
                                0.2
                                0.3
                                0.4
                                0.5
                                0.9
                                0.12
                                32
                                36
                                96
                                100

[Unbound memory (Base /Range)]: 0x0/64
      (bytes) (MB)                0xc000000/1856
                                0x180000000/128

[Available memory (MB)]: 2048
#

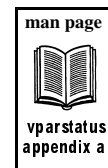
```

This output confirms that there are two CPUs available at hardware paths *97* and *101*. We can add these CPUs in a variety of ways. Let's use the **vparmodify** command to change the *num* of CPUs in *cabl1* to two CPUs. We do this by *adding* one to the current number of CPUs with *-a*. This is a relative operation in that one CPU will be added to the current number of CPUs. You can use **vparmodify -m** if you want to specify the absolute number of CPUs for the vPar rather than the relative number. The following shows this **vparmodify** command:



```
# vparmodify -p cabl1 -a cpu::1
#
```

We can now run **vparstatus -p cabl1 -v** to confirm that the CPU has been added, shown in the following listing:



```
# vparstatus -p cabl1 -v

[Virtual Partition Details]
Name:          cabl1
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
Min/Max:  1/3
Bound by User [Path]:
Bound by Monitor [Path]:  33          <-- original CPU @ 33
Unbound [Path]:  97                 <-- unbound CPU @ 97

[IO Details]
  0.0
  0.0.1.1.2.0  BOOT

[Memory Details]
Specified [Base /Range]:
      (bytes) (MB)
Total Memory (MB):  1024
#
```

The **vparstatus** output shows that the CPU at hardware path *97* has indeed been added to *cabl1* with the **vparmodify** command as *unbound*.

In addition, we can run **GlancePlus** or **top** to confirm that there are two CPUs in use on *cable2*. The following is a **top** output run on *cable2*:

```
# top
System: cvhdcon3                               Thu Oct  4 15:30:42 2001
Load averages: 0.19, 0.51, 0.62
124 processes: 110 sleeping, 14 running
Cpu states:
CPU  LOAD  USER  NICE  SYS  IDLE  BLOCK  SWAIT  INTR  SSYS
0    0.37  0.0%  0.2%  0.0% 99.8%  0.0%  0.0%  0.0%  0.0%
1    0.02  0.0%  0.0%  0.8% 99.2%  0.0%  0.0%  0.0%  0.0%
---  ---
avg  0.19  0.0%  0.2%  0.4% 99.4%  0.0%  0.0%  0.0%  0.0%

Memory: 93636K (57816K) real, 322124K (239536K) virtual, 746284K free Page# 1/4

CPU TTY          PID USERNAME PRI NI  SIZE  RES STATE  TIME %WCPU %CPU COMMAND
0 ?             36 root      152 20   0K    832K run   0:00  0.33  0.33 vxfsd
1 ?            1342 root      158 10   80K   212K sleep 0:10  0.28  0.28 cclugd
0 ?            1149 root      152 20 4644K 7260K run   0:06  0.21  0.21 prm3d
1 ?             922 root      154 24  540K  808K sleep 0:00  0.15  0.15 hpterm
0 pty/ttypl    3114 root      186 24  596K  528K run   0:00  0.17  0.15 top
1 ?            1146 root      -16 20 7788K 7240K run   0:03  0.14  0.13 midaemon
1 ?             3 root       128 20   0K    32K sleep 0:04  0.11  0.11 statdaemon
0 ?            2018 root      154 20 3908K 1908K sleep 0:00  0.05  0.04 alarmgen
1 ?            1272 root      152 20  856K  960K run   0:00  0.04  0.04 opcmona
1 ?            1372 root      152 20 1076K 2356K run   0:00  0.04  0.04 samd
0 ?             0 root       128 20   0K    0K sleep  0:11  0.02  0.02 swapper
1 ?             1 root       168 20  448K  204K sleep 0:00  0.02  0.02 init
0 ?             2 root       128 20   0K    32K sleep 0:00  0.02  0.02 vhand
0 ?             4 root       128 20   0K    32K sleep 0:00  0.02  0.02 unhashdaemo
1 ?             20 root      147 20   0K    32K sleep 0:00  0.02  0.02 lvmkd
0 ?             22 root      147 20   0K    32K sleep 0:00  0.02  0.02 lvmkd
1 ?             24 root      147 20   0K    32K sleep 0:00  0.02  0.02 lvmkd
0 ?            339 root      154 20  152K  204K sleep 0:00  0.02  0.02 syncer
0 ?            342 root      168 20   76K   192K sleep 0:00  0.02  0.02 vphbd
0 ?            345 root      168 20  156K  216K sleep 0:00  0.02  0.02 vpard
0 ?            410 root      154 20   80K   224K sleep 0:00  0.02  0.02 syslogd
0 ?            446 root      127 20  156K  424K sleep 0:00  0.02  0.02 netfmt
0 ?            552 root      154 20  740K  816K sleep 0:00  0.02  0.02 rpc.statd
0 ?            558 root      154 20 1004K 1032K sleep 0:00  0.02  0.02 rpc.lockd
0 ?            586 root      154 20  180K  316K sleep 0:00  0.02  0.02 inetd
0 ?            855 root      154 20 1064K  472K sleep 0:00  0.02  0.02 sendmail:
0 ?            863 root      154 20  772K  712K sleep 0:00  0.02  0.02 snmpdm
0 ?            896 root      154 20  620K  552K sleep 0:00  0.02  0.02 mib2agt
0 ?            914 root      154 20 1332K  444K sleep 0:00  0.02  0.02 cmsnmpd
1 ?            951 root      154 20 4044K 1840K sleep 0:00  0.02  0.02 rpcd
1 pty/ttypl    952 root      158 24  512K  180K sleep 0:00  0.02  0.02 sh
0 ?            974 root      168 20  152K  304K sleep 0:04  0.02  0.02 scrdaemon
0 ?            996 root      154 20  200K  336K sleep 0:00  0.02  0.02 pwgrd
0 ?           1039 root      154 10  308K  428K sleep 0:00  0.02  0.02 diagmond
0 ?           1093 root      154 20 1224K  816K sleep 0:00  0.02  0.02 ttd
1 ?           1135 root      154 20 2588K 1624K sleep 0:00  0.02  0.02 perlbd
0 ?           1156 root      154 20 2952K 1572K sleep 0:00  0.02  0.02 swagentd
0 ?           1167 root      154 20  224K  252K sleep 0:00  0.02  0.02 emsagent
0 ?           1168 root      127 20 2380K 2204K sleep 0:00  0.02  0.02 scopeux
```

This output shows two CPUs, labeled *0* and *1*, in *cable1*. The *System* name of *cvhdcon3* is shown at the output because the hostname for *cable1* is *cvhdcon3*.

Although this is a simple example showing how a Virtual Partition can be modified, it also demonstrates the power of vPars. While both vPars on the system are running, a processor can be added to one or both without interruption of the programs running in the vPars.

Note that the *-a* option to **vparmodify** changed the number of CPUs *relative* to the current number. In our case the current number of CPUs was one and using *-a cpu::1* added one CPU to the current number of one resulting in two CPUs. This is true also when we use the *-d* option to **vparmodify** to remove processors. The following example shows running **vparstatus** to see the two CPUs, using **vparmodify** to change the number of CPUs back to one (this is also relative to the current number of CPUs, which is two,) and a **vparstatus** to confirm that this change has taken place:

```
# vparstatus -p cable1 -v

# vparstatus -p cable1 -v

[Virtual Partition Details]
Name:          cable1
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
Min/Max:      1/3
Bound by User [Path]:
Bound by Monitor [Path]: 33
Unbound [Path]: 97

[IO Details]
0.0
0.0.0.0
0.0.1.1.2.0  BOOT
0.0.4.0  CONSOLE

[Memory Details]
Specified [Base /Range]:
      (bytes) (MB)
Total Memory (MB): 1024
# vparmodify -p cable1 -d cpu::1
# vparstatus -p cable1 -v

[Virtual Partition Details]
Name:          cable1
State:         Up
Attributes:    Dynamic,Manual
Kernel Path:   /stand/vmunix
Boot Opts:

[CPU Details]
Min/Max:      1/3
```



```
<-- bound CPU @ 33
<-- unbound CPU @ 97
```

```

Bound by User [Path]:
Bound by Monitor [Path]: 33          <-- original CPU @ 33
Unbound [Path]:                    <-- no unbound CPUs

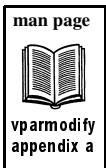
[IO Details]
  0.0
  0.0.0.0
  0.0.1.1.2.0 BOOT
  0.0.4.0  CONSOLE

[Memory Details]
Specified [Base /Range]:
          (bytes) (MB)
Total Memory (MB): 1024
#

```

We could perform many other modifications to the vPars with the two *unbound* CPUs that are available, such as adding two CPUs to one of the vPars or one CPU to each vPar.

Please keep in mind the relative nature of components when using **vparmodify** and that some changes, such as modifying memory or adding I/O components, require the vPar to be down at the time of this writing.



Virtual Partition Dump Files

When a Virtual Partition crashes, a dump file is created in **/stand/vmpon.dmp**. When the Virtual Partition boots, files are created in **/var/adm/crash/vpar**. The files have an extension with a number indicating the number of the dump that occurred. For instance, **vpmon.1**, **vpmon.dmp.1**, and **summary.1** indicate the first set of files that are saved in **/var/adm/crash/vpar**.

An example of what you might see in **/stand** and **/var/adm/crash/vpar** related to dumps are shown in the following listing:

```

VPARNAME = extraq1

# ll /stand
total 100400
-rw-r--r-- 1 root    sys           19 Jul 13 15:04 bootconf
drwxr-xr-x 4 root    sys        2048 Oct 18 11:43 build
drwxrwxrwx 5 root    sys        1024 Oct 18 13:06 dlkm
drwxrwxrwx 5 root    root        1024 Oct 18 11:21 dlkm.vmunix.prev
-rw-r--r-- 1 root    sys        3388 Oct 18 13:16 ioconfig
-r--r--r-- 1 root    sys           82 Jul 13 15:34 kernrel
drwxr-xr-x 2 root    sys        1024 Oct 18 13:18 krs
drwxr-xr-x 2 root    root        1024 Oct 18 13:16 krs_lkg

```

```

drwxr-xr-x  2 root    root      1024 Oct 18 13:18 krs_tmp
drwxr-xr-x  2 root    root      8192 Jul 13 15:04 lost+found
-rw-----  1 root    root        12 Oct 18 13:16 rootconf
-r--r--r--  1 root    sys      2035 Oct 18 11:42 system
-r--r--r--  1 root    sys      994 Jul 13 15:28 system.01
-r--r--r--  1 root    sys      999 Jul 13 15:56 system.02
-r--r--r--  1 root    sys      994 Jul 13 15:28 system.base
drwxr-xr-x  2 root    sys      1024 Jul 13 15:37 system.d
-r--r--r--  1 root    sys      2035 Oct 18 10:55 system.prev
-rwxr-xr-x  1 root    root    22682568 Oct 18 13:16 vmunix
-rwxr-xr-x  1 root    root    22682568 Oct 18 11:04 vmunix.prev
-rw-----  1 root    sys      8232 Oct 18 13:36 vpdb
-rw-----  1 root    root     8232 Jul 17 14:11 vpdb.OLD
-r-xr-xr-x  1 bin     bin     837616 Aug 31 18:59 vpmon
-rw-----  1 root    root    5078504 Oct 10 10:43 vpmon.dmp <-- vPar dump

# ll /var/adm/crash/vpar <-- vPar dump directory
total 46464
-rw-r--r--  1 root    root        2 Oct 10 10:43 count
-rw-r--r--  1 root    root     16794 Jul 17 13:26 summary.0
-rw-r--r--  1 root    root     17953 Jul 18 10:35 summary.1
-rw-r--r--  1 root    root     19538 Jul 18 11:36 summary.2
-rw-r--r--  1 root    root     10012 Oct 10 10:43 summary.3
-r-xr-xr-x  1 root    root    855928 Jul 17 13:26 vpmon.0
-r-xr-xr-x  1 root    root    855928 Jul 18 10:35 vpmon.1
-r-xr-xr-x  1 root    root    855928 Jul 18 11:36 vpmon.2
-r-xr-xr-x  1 root    root     837616 Oct 10 10:43 vpmon.3
-rw-----  1 root    root    5078504 Jul 17 13:26 vpmon.dmp.0
-rw-----  1 root    root    5078504 Jul 18 10:35 vpmon.dmp.1
-rw-----  1 root    root    5078504 Jul 18 11:36 vpmon.dmp.2
-rw-----  1 root    root    5078504 Oct 10 10:43 vpmon.dmp.3

```

The `/var/adm/crash/vpar` directory has in it the vPar dump-related files for four (0-3) crashes.

The dump file created in `/stand` is saved in `/var/adm/crash/vpar` and extended with the crash number. The dump file in `/stand` is overwritten with each crash, but you have a history with all of the dump files and related information in `/var/adm/crash/vpar`. Please leave in place the `/stand/vpmon.dmp` file.

vparmgr GUI

There is always the age-old question when working with UNIX whether a task should be done at the command line or using a Graphical User Interface (GUI). The `vparmgr` GUI many of the tasks described in this paper.

Figure 1-7 shows the main Virtual Partition Manager page that the you first see when the application is started that provides status. It gives basic information about the number of vPars and the resources on the system.

From this screen you can perform a variety of tasks including creating new vPars, modifying existing vPars, booting vPars, and getting more detailed information about the system's vPar configuration.

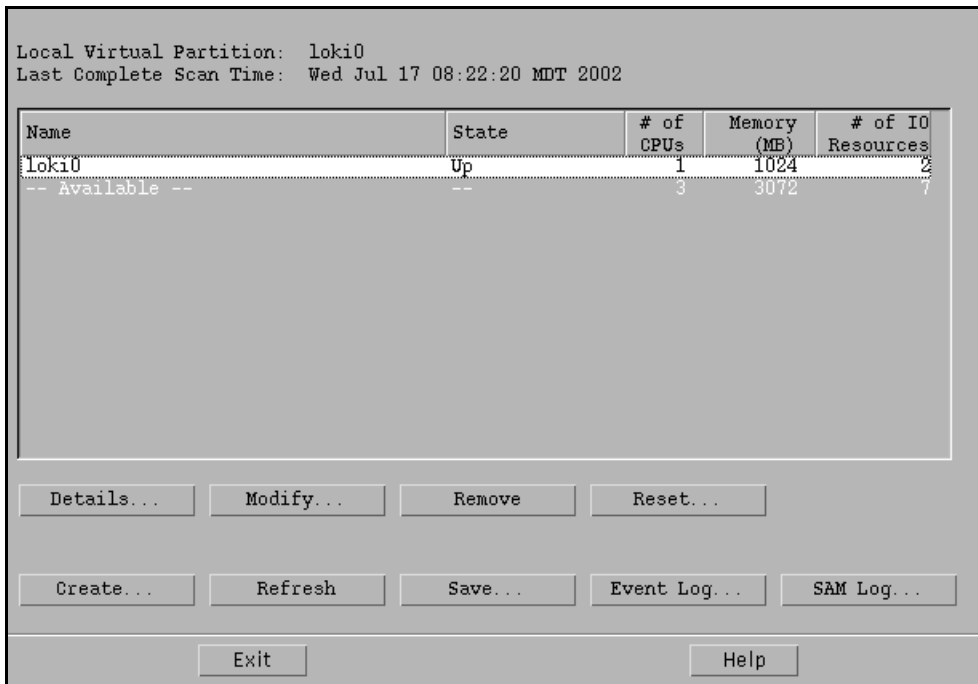


Figure 1-7 Main **vparmgr** Screen

Figure 1-8 is the vPar modify screen which is displayed when a user selects an existing vPar and click modify. The user can then change the configuration of this vPar. The actions available through the modify screen are similar to what is available in the create wizard.

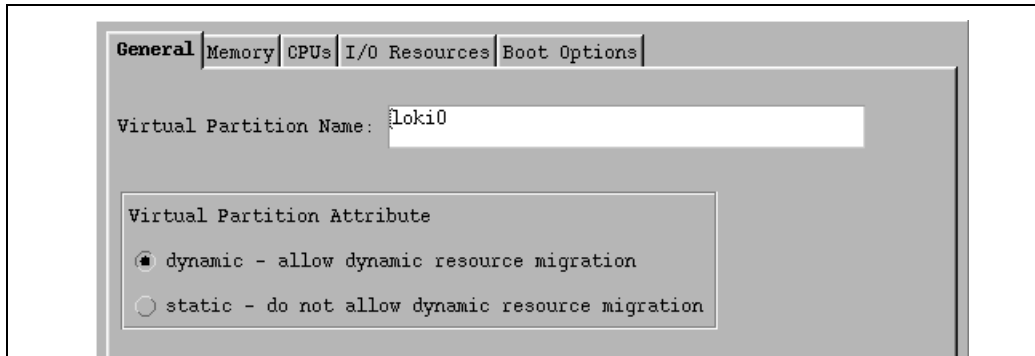


Figure 1-8 **vparmgr** *Modify* Screen

Figure 1-9 is the screen displayed when the user selects the available row on the status screen and clicks details. It shows more detailed information about the available resources on the system.

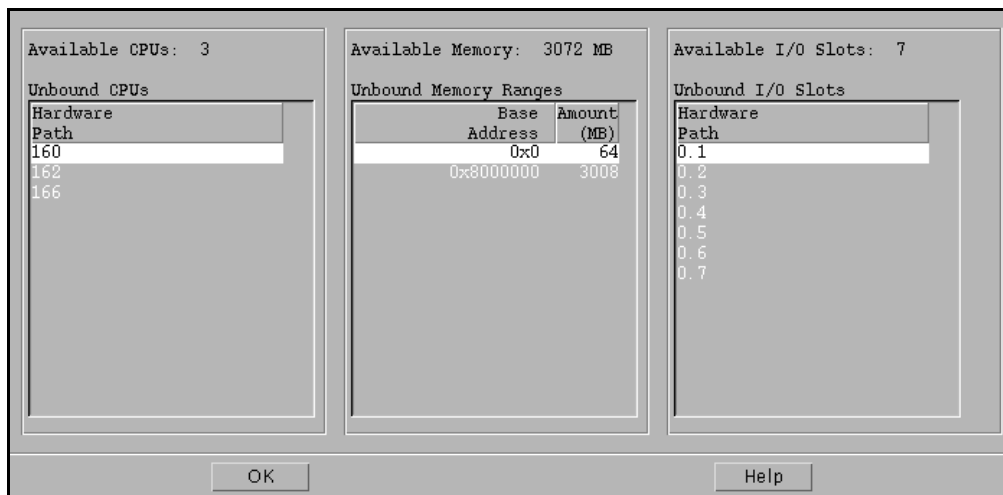


Figure 1-9 **vparmgr** *Available* Screen

With **vparmgr** you have the flexibility of using the graphical interface or command line when working with vPars.



Summary

There were some vPars-related commands in this paper that were not used. The **vparreset** and **vparremove** commands summarized in Table 1-1 were not issued at all for instance. The **vparremove** command can be run on any vPar provided that it is in the *down* state. The general steps to get vPars up and running and to perform some modification were covered to give you a simple framework from which to work. There are some other commands that were not covered, for which there are manual pages in Appendix A as well. More detail in specific areas appears in other chapters of the vPars book and I encourage you to review the online man pages for all of the vPars in Appendix A.

There are also some considerations related to server technology that we did not cover. If you have Instant Capacity on Demand (iCOD) employed on your server, all CPUs must be activated in order for vPars to work. When employing Processor Sets (psets) in a vPar, use only bound CPUs.

