PA–RISC/Linux Boot HOWTO

Thomas Marteau

TuxFamily

<marteaut@tuxfamily.org>

Deb Richardson

The Puffin Group

<deb@thepuffingroup.com>

Thibaut Varène

PA/Team

<T-Bone@parisc-linux.org>

version 1.5 for DocBook V4.2 Edition

Copyright © 1999 The Puffin Group and Deb Richardson.

Copyright © 2001, 2002, 2003 Thomas Marteau.

Copyright © 2002, 2003, 2006 Thibaut Varène.

2006–06–06

Revision History

Revision 1.5	2006–06–06	Revised by: tv
XML cleanup, improved and reorganized contents. Updated to palo 1.5+ and kernel 2.6.		
Revision 1.1	2003–11–01	Revised by: tm
Added Jeremy Drake's Windows" server boot howto.		
Revision 1.0	2002–10–04	Revised by: tm & tv
Content done by Thibaut. Ready for Woody release. Added Glossary and bibliography. XML conversion.		

Revision 0.9	2002−01−15	Revised by: tm
This version brings you some useful advices for compiling your own kernel on hppa systems.		
Revision 0.8	2001−10−17	Revised by: tm
This version takes care of the change of name of the official FTP and CVS sites and modify the license.		
Revision 0.7	2001−10−13	Revised by: tm
This version adds some updates due to the progress of PA/Linux.		
Revision 0.6 draft	2001−09−26	Revised by: tm
This version contains some minor changes and complete the "obtaining bootp/tftpd" section.		
Revision 0.5 draft	2001−07−03	Revised by: tm
This version is a large update from Deb's work.		
Revision 0.3 draft	1999−11−24	Revised by: dlr
The initial and published version of this HOWTO.		

This document outlines the procedures to get the PA−RISC/Linux kernel to boot on your PA−RISC system. It also explains the usage of **PALO**, the kernel loader for PA/Linux. You will find much information on how to compile a kernel from the source available at http://cvs.parisc−linux.org/. Please note that this HOWTO version is newer than Deb Richardson's and includes more accurate information because of the progress of the port. Nevertheless, it's worth mentioning that this document kept parts of Deb's original work and unveiled some remarkable information.

If you are looking for information related to HP hardware but not directly to PA−RISC, please read Bruno Cornec's HP−HOWTO.

Note: by the time this HOWTO was started, Debian was the only Linux distribution available for the PA−RISC platform, hence the "Debian color" of this document. Some times, Debian specific commands will have to be replaced by their equivalent, if any.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 as published by the Free Software Foundation; with no Invariant Sections, with no Front−Cover Texts, and with no Back−Cover Texts. A copy of the license can be found at http://www.gnu.org/copyleft/fdl.html.

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies, that could be damaging to your system. Proceed with caution, and although it is highly unlikely that accidents will happen because of following advice or procedures described in this document, the author(s) do not take any responsibility for any damage claimed to be caused by doing so.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

Table of Contents

Introduction **1**

Chapter 1. Supported Hardware **2**

Chapter 2. Preparing to boot **4**
2.1. BOOT_ADMIN 4
2.1.1. Entering the BOOT_ADMIN interface 4
2.1.2. BOOT_ADMIN commands 5
2.2. Consoles 6
2.2.1. Using graphic console 7
2.2.2. Using serial console 7
2.2.3. Switching consoles 8

Chapter 3. PALO, the PA/Linux kernel loader **10**
3.1. What is PALO? 10
3.2. What does PALO? 10
3.3. PALO management tool usage 10
3.3.1. Making a lifimage with RAMDISK 11
3.3.2. Making a lifimage with NFSROOT 12
3.3.3. Making a bootable partition 12
3.4. How to use PALO at early boot stage? 15
3.4.1. The theory 15
3.4.2. A complete example 16

Chapter 4. Available boot solutions **19**
4.1. Booting from CD 19
4.2. Booting from hard drive 19
4.3. Booting from network 19
4.3.1. Preparing to boot from network 20
4.3.2. rboot or bootp? 20
4.3.3. Using rboot 20
4.3.4. Using dhcp/tftp 21
4.3.5. Using bootp/tftp 23
4.3.6. Effectively booting from network 24

Chapter 5. Building and installing a custom kernel **26**
5.1. GCC compiler 26
5.1.1. Native build 26
5.1.2. Cross compiled build 26
5.2. Kernel configuration 27
5.2.1. Configuring 2.4 kernels 27
5.2.2. Configuring 2.6 kernels 31
5.3. Kernel installation 31

Appendix A. Windows" 2003 boot server howto **33**
A.1. Setup the DHCP service 33
A.2. Get & setup the TFTP server 33
A.3. Attempt to netboot 34

Table of Contents

Appendix B. Older PALO dumps....35
B.1. A500 Session dump using PALO 0.97....35

Appendix C. HOWTO contributors....37

Glossary....38

Bibliography....40
Notes....41

Introduction

You just received this PA−RISC box you bought online, or maybe you got it from your company scrapyard. Anyway, here comes the question of the operating system you are going to use. The PA/Linux project consists in porting Linux to the PA−RISC architecture, and we hope that if you decide to use it on your box, this HOWTO will help you in the process of setting it up.

In addition to porting the Linux kernel, the development team is working on porting the Debian project to PA−RISC. In fact, by the time we wrote this document, over 97% of the package pool is available for the stable Debian release (3.1, aka *Sarge*) on hppa (see the buildd stats for detailed data). Some Debian developers and users reported that the port was one of the easiest to install, as it feels like installing an i386 system.

For more information about the PA−RISC/Linux porting project, please see http://www.parisc−linux.org/, or a mirror site like http://www.fr.parisc−linux.org/. This site deals with kernel development and improvement. For user−space troubles, please refer to Debian hppa port pages.

In a few words, this HOWTO is aimed at anyone looking for some help and information about using Linux on a PA−RISC system. No particular prior knowledge is necessary but bases about how Debian packages work, and general background about Linux can be helpful.

It is also worth mentioning that some sections of the present document aren't Linux−centric, and may be of use to people dealing with other OSes, such as BSDs or even HP−UX.

After listing supported hardware, this HOWTO explains some commands of the Boot Console Handler (BCH) available at boot time. Then, the features of the PA/Linux kernel loader are introduced in another chapter, and finally many ways to get your system up and running are detailed. At the end, the text goes deep in the kernel compilation and configuration, as well as a few appendices giving some extra hints.

Chapter 1. Supported Hardware

With the release of PA−RISC architecture in Debian 3.0 (aka *Woody*), a major improvement was made in term of quantity and quality of hardware support. Since 0.9.3 released[1], the kernel has been greatly improved, so that much unsupported hardware by the time 0.9.3 went out is now handled. That's why even if your model is not listed here, you might give it a try and report your result to the mailing list: <parisc−linux@lists.parisc−linux.org>.

Mind that as of this document's publication date, the 2.4 series of the Linux kernel are deemed obsolete, and no development happens in that branch. Any new comer to the PA−RISC/Linux port should look at the 2.6 kernel series, which supports much more hardware than the 2.4 does. Accordingly, one will not try to use the now aging *Woody* release and will instead focus on its successor: *Sarge*, or even better, the upcoming *Etch*.

The following PA−RISC machines should work just fine, provided that you follow the instructions of the present document. Please note that this list can change at any time. The best way to get an up to date version is to look at http://www.pateam.org/list.html. This is the place to find relevant information about a specific model, including special boot procedures. More hardware information can be found on OpenPA.

SMP machines should work with SMP kernels (and UP ones, of course) unless stated otherwise.

- All 712 models.
- All 715 models including Strider series.
- All 705, 710, 720, 730, 750 models should work.
- Some 725, 735 (no HVD SCSI), 755 models are now working. But since there was not a lot of feedback about these machines, we can not be more explicit.
- The VME−like systems are supported. This includes 742 and 743.
- A180 and similar.
- A500, and similar (rp2400 series).
- BXXX models like B132, B160 and B180. These boxes can be used in the framebuffer mode through the Standard Text Interface.
- BXXXX models like B1000, B2000 and B2600. These boxes can be used with `STI_CONSOLE`, but framebuffer only works with VIS−EG cards. FX adapters are not supported.
- CXXX models like C100, C110, C160, C180, C200, C240, C360.
- CXXXX models. Indeed, BXXXX and CXXXX are the same kind of machines, based on Astro/Elroy (aka SBA/LBA) chipsets with varying CPU speeds, number of memory/PCI slots. C3000, C3600, C3750 are reported to be working.
- D class works unless you have a Remote Management Card installed. Even then, it still kind of works, it's just that ttyS0 gets assigned to the second serial port and you have to switch cables around.
- J class is quite well supported. It has the same split as C class, *i.e.* JXXX and J2240 are U2/Uturn based and JXXXX are Astro/Elroy. It is the SMP version of CXXXX models.
- K class is supported if you are using recent ISO images (*e.g.* Sarge ones).
- L class and similar (rp5400 series).
- N class: N4000 (some rp7400 series are reported to work).
- R class is basically the same as D class.

These are not really working yet but work is being carried on.

- The current 8−way (and bigger) machines using SX1000 chipset and pa8800 (or pa8900) processors do not work yet. Similarly, smaller 2−socket and 4−socket platforms using ZX1 chipset and pa8800 (or pa8900) such as rp3400 series do not work correctly yet either.

No plan to get the following hardware completely supported in the near future:

- E class: E35 and E55 are known to work diskless. The SCSI support is not expected to work anytime soon.
- F,G,H,I classes: Currently not supported.
- SuperDome: It boots "single−cell", multi−IOMMU doesn't work.
- T 5XX and V class: Nobody is working on it at the moment.
- rp8400: these are cell based and probably don't work yet.

The following hardware might never work:

- T600.
- Vis−FX graphic adapters.

Chapter 2. Preparing to boot

Like any other system, machines based on PA−RISC processors have to go through several steps in order to get Linux up and running. The next section introduces you to the early boot management of your PA−RISC computer. (By the way, to be a bit less awkward, we might from time to time call it a *'PA'* box). This chapter will give you some key concepts like **BOOT_ADMIN**.

2.1. BOOT_ADMIN

First of all, you must learn what is and how to use **BOOT_ADMIN** on your PA−RISC box, before thinking about doing anything with it.

BOOT_ADMIN is a *firmware* application, used to manage a PA−RISC machine at an early boot stage, *i.e.* when the box has not yet started its *Operating System*. It is also known as the *Boot Console Handler (BCH)*. Those familiar with the x86 world will probably see it as a kind of *BIOS*, whilst PPC fans might think of it as an equivalent for *Open Firmware*.

We named it **BOOT_ADMIN** throughout this document since it is the name it is the most common prompt it will display on most PA−RISC machines. You will see through this HOWTO that there are many references to it, therefore it's worth saying that minimalistic **BOOT_ADMIN** skills are mandatory!

2.1.1. Entering the BOOT_ADMIN interface

Entering the **BOOT_ADMIN** management tool isn't that awful:

1. Turn your PA−RISC box on.
2. During the boot process, the following message will appear on the current *console* (see Section 2.2):
   ```
   Searching for Potential Boot Devices.
   To terminate search, press and hold the ESCAPE key.
   ```
   When this message is displayed, press and hold the **Esc** key until an option menu appears. This can take a while, be patient. On recent machines, pressing any key interrupts the boot process as well.
3. By default, you should enter the **BOOT_ADMIN** console. Though on some 715s and 725s, an option menu looking like this may be shown:
   ```
   b)    Boot from specified device
   s)    Search for bootable devices
   a)    Enter Boot Administration mode
   x)    Exit and continue boot sequence
   ?)    Help

   Select from menu:
   ```
   Type '**a**' and hit **Enter** to enter Boot Administration mode. This will bring up a '`BOOT_ADMIN>`' prompt.

Once you have the '`BOOT_ADMIN>`' prompt, you can pat yourself on the back: you are in **BOOT_ADMIN** mode!

As it has been said before, the prompt can differ between machines. On recent ones, for instance, it looks like that:

```
Main Menu: Enter command or menu >
```

2.1.2. BOOT_ADMIN commands

BOOT_ADMIN is an early boot subsystem (a Boot Console Handler, as said before) where you can execute a limited set of commands. You should find here everything you need to know about them.

All HP−PA systems have a BCH. The display can be different but the idea remains the same. That's why the following list is not complete but consistent enough. Another important thing is that for each command, you have a shorter way to invoke it. You can see the shortcut shown as uppercase letters in the command name. Full names will be used in this section.

Be cautious when dealing with the BCH, you can harm your system.

Some commands may appear in several different menus, and all commands listed here may not necessarily be available on your particular system, this is normal.

2.1.2.1. The `main` commands

These commands are the basic ones.

- **boot** may be followed by an argument which indicates the path you want to boot. The path should be the definition of a device like for example `FWSCSI.6.0` or `PRI` if you have set this variable correctly. Usually defaults to `PRI`.
- **path** displays or sets the current paths. Invoked with only one argument it will display the current path of the entity passed as argument: **path alt** will display the current alternative boot path. **path pri fwscsi.6.0** will setup the primary boot path as the device attached to Fast and Wide SCSI controller with ID 6 and LUN 0. You can also set and display the paths of console (graphics/serial) and keyboard (ps2/hil/usb).
- **search** is a very useful command. It automagically checks all possible boot devices and displays these bootable paths. In several firmware versions, it links them to a shortcut (like `P0`). It can even search the LAN, if the box is able to boot it. Some firmware revisions allow you to restrain the search path like: **search lan** or **search disk**.
- **display** redisplays the current menu.
- **help** gives you an overview of the available commands and their action. **help *name*** will give you details on command *name*. By default, you can list all `main` commands by typing **help main**.
- **main** will bring you back to the `main` menu, whatever menu you might be currently browsing.
- On almost every systems, there is a **reset** instruction. It restarts the machine with the latest parameters you have set.

2.1.2.2. The `configuration` commands

These commands are available in the `configuration` menu. So, in order to use them, you must enter this menu by typing **configuration** at the '`BOOT_ADMIN>`' prompt.

- **auto** will tell you if the box will automatically start booting when switched on, or will do a search for boot devices, depending on the first argument passed to the command (*boot*, *search*, *start*). You can modify this parameter with the keywords *ON* and *OFF*.
- **default** sets back the factory defaults.
- **monitor** (only in graphic console) sets your display configuration by typing **monitor *<path>* *<type>*** which indicates your console path and type. You can list the available modes by typing **monitor list**.
- **fastboot** displays or sets the boot tests execution.

2.1.2.3. The `information` commands

They give you access to global information about your system. Going into this menu is done by asking for **information**.

- **all** should display everything.
- **bootinfo** lists all the boot parameters of the system.
- **fwrversion** gives your firmware revision. You can check if your firmware is up−to−date at this webpage.
- **lanaddress** shows the MAC (Ethernet) address of the system. On some boxes (especially 712s), two different addresses may appear. The one you are looking for is the first.

2.1.2.4. The `service` commands

It is a PA−RISC guru menu.

You will find nothing really interesting for an end−user here. We recommend you not to play with it unless you *really* know what you are doing.

- **pim [<proc>] [HPMC|LPMC|TOC]** displays the content of a *Processor Internal Memory (PIM)* and Error Log. It is very useful after a *Transfer Of Control (TOC)* to collect debugging information.
- **clearpim** clears *Processor Internal Memory (PIM)* data.
- **scroll** enables or disables the scrolling mode in **BOOT_ADMIN** on recent boxes.

2.2. Consoles

Throughout your PA−RISC experience, you will be often told about *consoles*. This section aims at de−obfuscating what this word means and how to use said *consoles*.

In order to boot your PA−RISC system with the PA/Linux kernel, you must first set up a *console*. A *console* is basically the device where the kernel (and the firmware) will display its output, and where input can by sent to control the system at an early boot stage. You can use either *graphic console*, which requires having a monitor and a keyboard attached to the system, or *serial console*, which allows serial line communication between the system and another Linux machine, or any VT system.

Please note that the firmware console and kernel console are not necessarily the same. For instance, it is possible to interact with **BOOT_ADMIN** using keyboard & monitor, and once PA/Linux is up, to have kernel output sent to serial port only. By default, *PA LOader (PALO)* will try to use firmware console as the kernel one.

Workstations usually boot in graphic mode, whereas servers boot in serial mode. Some boxes will also automatically switch to serial if no keyboard is connected, or if you hold down `TOC` switch while powering the system on.

If you don't know what the actual console of your box is, it's quite simple: find the place where first output is sent when the box is turned on (serial line or monitor output, if any), that is the console.

If you are trying to setup a PA−RISC workstation and have a monitor handy, the easiest method is to use *graphic console*. If you get into troubles, or are trying to configure a server, choose *serial console*.

2.2.1. Using graphic console

To use the graphic console, you must first ensure that the Linux kernel supports your system's graphic card.

There are two ways to deal with the graphic console. If you think about bug−reporting any trouble, you must know how to differentiate both. First, the *STI* console is the classical video text console, like *VGA* on a common PC for example. This name is due to the fact that each PA−RISC box with graphical capabilities features the *Standard Text Interface (STI)* which defines some standardized ways to access the video memory. The other graphic console is the well known *framebuffer* console (which on HP−PA uses STI in a special manner, hence the name *STIfb*). In this case, when booting, you will see a characteristic little penguin appearing on the top−left corner. This is the easiest way to differentiate the two graphic modes.

Obviously, if you can use graphic console, it is the easiest way to proceed. Nevertheless, you must be sure that your hardware is supported.

All HP−provided graphics cards can deal with *Standard Text Interface Console layer (STIcon)*, but not all of them are *Standard Text Interface FrameBuffer layer (STIfb)* supported in Linux. This is especially true for Vis−FX cards that can only be used through STIcon.

2.2.2. Using serial console

The serial console is a good way to obtain all console messages, including the BCH ones. It is very useful for bug reports, as its output can be easily dumped. Moreover, most of the PA−RISC servers can only be managed with serial console.

The only cases where serial console *HAS TO* be used is either if you don't have a monitor handy, or if the machine doesn't support graphics. It is also possible that the kernel can *NOT* handle some specific graphics hardware present in the machine, but that is pretty rare (STIcon should work everywhere).

Here is the procedure to setup serial console support.

2.2.2.1. Serial Cable

To connect a PA−RISC machine to a PC's RS232 port, you need a 9−pin−to−9−pin female plugs null−modem cable. You should be able to obtain such a cable at your local computer hardware reseller. Obviously, you can also choose to connect the other end of the cable to a terminal (in this case it will probably need a 25−pin male plug). Anyway, the most practical method is to connect it to another box running **minicom** or **cu**, which makes all output easily available for further usage (dump report, session log, and so on).

2.2.2.2. Configuring minicom on Linux

In order to communicate with a PA−RISC machine on a serial line, you have to set it up in serial console mode (see below), and configure a serial communication program. We recommend **minicom**, which can be found in most Linux distributions. If you don't have **minicom** on your system, you can find the latest package on any major Linux software website.

Most of the **minicom** configuration is machine dependent. However, you must ensure that:

a. The baud rate is set to 9600
b. Protocol is set to 8−N−1 (8bit data, No parity check, 1 stop bit)

Don't worry too much as these are the default values on PA/Linux. If you are running **minicom** on a PC, you will probably need to change the baud rate.

2.2.3. Switching consoles

It might prove useful that you learn how to manage the console mode on a PA−RISC box. The following section will explain the various operations regarding console modes.

2.2.3.1. Checking current console mode

Type: **path console** to see the current console mode.

If it's *graphic console* mode, it will return something like: '`Console path = graphic_1`'.

If it's *serial console*, it will return: '`Console path = rs232_a.9600.8.none`' or something similar.

On some models, there can be slight differences in the naming, but the idea stays the same. If you want to see more descriptions here, please send us a message describing the box you use and what you get.

2.2.3.2. Changing to serial console mode

To change to serial console mode, type the following command at the '`BOOT_ADMIN>`' command prompt:

path console rs232_a.9600.8.none

or, like on B132L+

path console serial_1

Anyway, on most boxes if you try to setup an invalid path for the console, you will be warned and prompted again for a valid path. To verify that the console path has been correctly set, type **path console**. This should return something like '`Console path = rs232_a.9600.8.none`', indicating that the system is now set up to boot using serial console, on RS232 port 'A'. If your machine has only one, this is OK, if not, take care to use the right one. **reset** will reboot your system with the new parameters.

2.2.3.2.1. How can I change the boot console to serial on a 712?

Unfortunately, it is **normally** not possible. Although 712s are configured for in–house HP development to use serial, this cannot be set in **BOOT_ADMIN**. You will have to use graphic console on 712s. And why the hell would we use this beautiful 712 with serial console when we can have X on it?!

Anyway, if you feel like trying bleeding edge solutions, there is a tip at the PA/Linux mailing list archive. This explains how to change the console from an *HP/UX ISL* prompt. Since you actually *need* HP/UX to be able to do the serial trick, you can find a small HP/UX *lifimage* here: http://www.pateam.org/archive/uxbootlf. (See further Section 4.3 to learn how to *netboot* a lifimage). In fact, serial console on 712 is especially useful if you want to boot the box without having a keyboard attached to it, which is otherwise not possible.

The following takedown is highly unofficial, unsupported and in a general way a *bad idea*, as you can make your 712 unbootable, needing intervention from a HP–techie, if something goes wrong. Beware! Here is the procedure:

1. Turn the box on and when in **BOOT_ADMIN**, boot into HP/UX ISL. For example:
   ```
   BOOT_ADMIN> boot lan isl
   ```
2. Once you get the '`ISL>`' prompt, type the following:
   - Switching to serial: **conspath 2/0/4.0x283**
   - Switching to graphic: **conspath 1/0/0.0**
3. Still at the '`ISL>`' prompt, type **disp**, and check that console path is either '`(hex) 2/0/4.283.0.0.0.0.0`' for serial, or '`(hex) 1/0/0.0.0.0.0.0.0`' for graphic.
4. Power cycle the system to bring it up on the new console.

2.2.3.3. Changing to graphic console mode

This is the reverse of the previous operation. By checking your console path, you should see '`Console path = rs232_a.9600.8.none`'. Now, you can switch to the graphic mode by issuing the following command at '`BOOT_ADMIN>`' prompt:

path console graphic_1

The actual *switch* will happen after a **reset**. If the monitor does not seem to work properly, try to press the **Tab** key (on the keyboard attached to the box of course) at the beginning of the boot sequence to change the resolution of the display. By pressing this key, the monitor resolution cycles from one to another.

Keep that in mind when changing monitors.

Chapter 3. PALO, the PA/Linux kernel loader

3.1. What is PALO?

PALO is a set of two programs, a boot loader, which is loaded by the PA−RISC *firmware* into memory and then executed, and a boot media management tool, which prepares and updates bootable media such as hard disk drives.

The **PALO** boot loader executable is stored in a file called `iplboot`. '`IPL`' is HP jargon for *Initial Program Loader* (See the glossary). The boot media management tool is called **PALO**, which stands for PA/Linux LOader, just as on x86 the boot media management tool is called LILO.

Even though **PALO** is much alike LILO (both have a userland application and a boot loader executable), it's worth mentioning that **PALO** doesn't usually need to be called every time you build and install a new kernel, as LILO does[2].

PALO is strongly related to PA/Linux development. Thus, several versions have been released. Some changes in the way **make palo** operates are explained by the author of **PALO**, *Paul Bame*, in this mail.

3.2. What does PALO?

The main idea is to boot a kernel, passing it all needed parameters. This is what the *boot loader* part of **PALO** does (see Section 3.4). Once it has been called by the *firmware*, it will load the Linux Kernel in memory, passing to it the given arguments, and tell the processor to branch to its entry point. This will begin the execution of the kernel on the PA−RISC computer.

The **PALO** management tool can transform the usual `vmlinux` into a PA−RISC bootable *lifimage*, including or not `RAMDISK` or `NFSROOT` support. However, it can also make a hard disk drive bootable, specifying the console output and the root device. We are going to see all these points precisely.

What must be kept in mind is that `vmlinux` is the kernel alone, which is not bootable as is. It needs **PALO** to be turned into a bootable `lifimage` for CD or network boot, or to be loaded at boot time from a prepared hard disk drive. Have a look at *Glossary* about these words. Quoting a well known PA/Linux hacker:

> *People often try to put a lifimage in /boot, or a vmlinux on the network boot server.*
>
> −−Richard Hirst

Which is obviously wrong.

3.3. PALO management tool usage

Here we will show what can be done with the **PALO** boot media management tool. For in−depth information about **palo** usage, we strongly advise you to take a look at **PALO**'s `README` file, which can be found in `palo/` directory on http://cvs.parisc−linux.org/.

For the next two steps, you will need a compiler toolchain, see Section 5.1.

3.3.1. Making a `lifimage` with `RAMDISK`

First things first: when should you walk this way?

At an earlier stage of the PA/Linux project, the `lifimage` was very useful. In fact, simply putting this file in a boot server tree allows you to boot your PA box using the **boot lan** instruction without any further involvement (see Section 4.3). The main advantage of a `RAMDISK` is that it unpacks its own file system in RAM, and therefore is completely independent of the machine I/O capabilities (hard drives, etc). The main drawback is that you have to build your own `RAMDISK` if you have memory constraints or need some customized files. Now, let's see how to obtain a `lifimage` with `RAMDISK`.

We assume you got the latest source of the PA/Linux kernel tree (to which we'll refer below as the "`linux/` directory"), and that you are somewhat familiar with kernel configuration. Check Section 5.2 for PA−RISC specific options. Mainly, you will need a (cross−)compiler, the `linux/` directory and the **PALO** package installed. If you do not have it, run as *root* **apt−get install palo**. Everything can also be found at http://www.parisc−linux.org/. Go through the kernel configuration step. Then, run **make palo** and if **PALO** is installed, the following message should appear at the end of the compilation:

```
A generic palo config file (./palo.conf) has been created for you.
You should check it and re-run "make palo".
WARNING: the "lifimage" file is now placed in this directory by default!
```

So, edit the `palo.conf` file:

```
# This a generic Palo configuration file.  For more information about how
# it works try 'palo -?'.
#
# Most people using 'make palo' want a bootable file, usable for
# network or tape booting for example.
--init-tape=lifimage
--recoverykernel=vmlinux

########## Pick your ROOT here! ##########
# You need at least one 'root='!
#
# If you want a root ramdisk, use the next 2 lines
#   (Edit the ramdisk image name!!!!)
--ramdisk=ram-disk-image-file
--commandline=0/vmlinux HOME=/ root=/dev/ram initrd=0/ramdisk

# If you want NFS root, use the following command line (Edit the HOSTNAME!!!)
#--commandline=0/vmlinux HOME=/ root=/dev/nfs nfsroot=HOSTNAME ip=bootp

# If you have root on a disk partition, use this (Edit the partition name!!!)
#--commandline=0/vmlinux HOME=/ root=/dev/sda1
```

As you can see, the `RAMDISK` mode is the default. The string *`ram-disk-image-file`* should give to **PALO** the path of your `RAMDISK` file. You shouldn't change anything else to this file. After editing `palo.conf`, you can run **make palo** again. The result, a `lifimage` file, is waiting for you in the `linux/` directory.

3.3.2. Making a `lifimage` with `NFSROOT`

This method is widely used because the kernel and the file system are directly accessible on your boot server. It is also very easy to test a new kernel. You just have to generate the kernel and put it in the correct directory. When starting up, the PA box will boot the new kernel by typing **boot lan** in **BOOT_ADMIN**. Finally, it is the only way to go for systems which I/O devices are not supported (such as E class, by the writing of this document).

Enabling `NFSROOT` support is easier than `RAMDISK`. You have to edit `palo.conf` to specify the boot server IP address instead of the string *`HOSTNAME`*. For instance, if your server has 10.10.10.2 as its IP address, then the `palo.conf` file should contain:

```
# This a generic Palo configuration file.  For more information about how
# it works try 'palo -?'.
#
# Most people using 'make palo' want a bootable file, usable for
# network or tape booting for example.
--init-tape=lifimage
--recoverykernel=vmlinux

########## Pick your ROOT here! ##########
# You need at least one 'root='!
#
# If you want a root ramdisk, use the next 2 lines
#   (Edit the ramdisk image name!!!!)
#--ramdisk=ram-disk-image-file
#--commandline=0/vmlinux HOME=/ root=/dev/ram initrd=0/ramdisk

# If you want NFS root, use the following command line (Edit the HOSTNAME!!!)
--commandline=0/vmlinux HOME=/ root=/dev/nfs nfsroot=10.10.10.2 ip=bootp

# If you have root on a disk partition, use this (Edit the partition name!!!)
#--commandline=0/vmlinux HOME=/ root=/dev/sda1
```

If you have another IP, this field must be filled in with the correct data. You shouldn't change anything else to this file. After having properly configured `palo.conf`, you can go into the `linux/` directory and issue a **make palo**. The result, a `lifimage` file, is as usual waiting for you in the `linux/` directory.

For advanced details on `NFSROOT` management, take a look at *Bibliography* for the appropriate HOWTOs.

3.3.3. Making a bootable partition

In this part, **PALO** can be seen as a LILO clone. **palo** is mainly a program that enables a PA box to boot a kernel present on its hard disk drive. This section is going to explain how to make it work.

After installing the **PALO** package, a copy of the default `palo.conf` can be found at `/usr/share/doc/palo/palo.conf`. We will explain here how to customize it to fit your needs.

To setup a bootable hard disk, you have to partition it properly (that is, if you want to use it as your primary boot device). This implies that this step can only be achieved either if you have already booted a minimal system on your PA–RISC box (from CD or network, see Chapter 4), or if you intend to prepare your hard disk using another computer than the target one (which can be useful to unpack and setup a downloaded file system for a slow box, for example). The point of this HOWTO is not to teach you how to use **fdisk** and

friends, so here are the few things you HAVE TO know:

- A partition entirely contained within the first 2GB of your target device has to be of partition type '`f0`', which is the reserved partition type for **PALO** boot loader.
- There are two ways to use **PALO**: the old scheme (available on all versions), in which that partition will only store configuration and recovery kernel; and the new scheme (available since **PALO** 1.5), in which that partition will be formatted as ext2 or ext3 and mounted in `/boot`.
- In the first case, the partition does not need to be huge. This is were **PALO** will save its configuration, recovery kernel(s) – about 5MB each – and optional recovery ramdisk, so 32MB seem far sufficient.
- Alternatively, in the second case, since you will use it as `/boot`, you should size it decently. 100MB is a good cut.

Beware! The `vmlinux` file that will be actually booted has also to be located within the first 2GB of the hard disk. We strongly recommend to either (in the *old* scheme) create a separate `/boot` partition at the beginning of the disk (unless you plan to boot recovery kernels every time), or use the *new* scheme and mount the **PALO** partition as `/boot`, because if ever your vmlinux binary gets physically stored past the first 2GB of the disk (like when filling up '/' with data), the box won't boot anymore.

In fact, this third usage of **PALO** is the most common as the default `/etc/palo.conf` makes it easy to configure.

3.3.3.1. The old scheme: hidden partition

The hidden partition is deprecated. Don't use this for a new setup. Use the new scheme instead (see Section 3.3.3.2). The hidden partition method is documented for the sake of posterity.

Here is the output of **fdisk** which represents the hard drive of a box with 16MB **PALO** space, 128MB swap space and about 1GB '/' partition:

```
bash# fdisk -l /dev/sda

Disk /dev/sda: 133 heads, 62 sectors, 1017 cylinders
Units = cylinders of 8246 * 512 bytes

         Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *         1         4     16461   f0  Linux/PA-RISC boot
/dev/sda2             5        34    123690   82  Linux swap
/dev/sda3            35       277   1001889   83  Linux
```

Now let's deal with **PALO** configuration. Here are the various parameters you can change:

- *`recoverykernel`* is (as the name suggests) the path to the kernel you want to boot within a failsafe session, it will be stored in the '`f0`–type' partition.
- *`bootloader`* is the path to the `iplboot` boot loader utility which is created by **PALO** when you issue a **make iplboot**.

Usually you don't want to change or even specify this.

- *`init-partitioned`* is used to indicate the pre–partionned device where palo will write its boot parameters. The effect is immediate. It means that **PALO** is going to write on (and erase the content of) the '`f0`' partition of this device, which has to exist.
- *`commandline`*: the first digit is the number of your ext2/ext3 partition where the kernel file is

located, as reported by **fdisk**. Logically, the next string is the absolute path to the kernel *from the root of THIS partition* [3]. The following space separated parameters (do NOT use any quotation mark!) will be passed to the kernel as its arguments. *e.g.*: `HOME=` and `TERM=` are environmental parameters passed to **init** when booting. They are not compulsory but they can be useful. `root=` tells the kernel which partition it must mount as the root file system while booting. It can be tricky when you have more than one disk, and is a mandatory argument. Maximum length for the commandline is 127 characters.

You can also add `console=`, to force the designation of the output console. You should remember that `console=ttyS0` is for a serial console and `console=tty0` is for a STI (graphic) console. Support for the MUX console (if enabled in the kernel, see Section 5.2) has been added, using `console=ttyB0`. Recent versions of **PALO** auto detect the right console path (except for MUX), and can figure out whether a 32bit or 64bit kernel should be used. If not, please mail to the mailing list. Last but not least, if you are using Debian 2.6 kernels, you will also need to add `initrd=X/path/to/initrd`, following the same rules as for the kernel path[3], see above.

According the above **fdisk** example, we want to use `/dev/sda3` as our root partition. Thus, the configuration file should look like that:

```
# The following arguments are set up for booting from /dev/sda, specifically
# mounting partition 3 as root, and using /boot/vmlinux as both the
# recovery kernel, and the default dynamically-booted kernel.
--recoverykernel=/boot/vmlinux
--init-partitioned=/dev/sda
--commandline=3/boot/vmlinux root=/dev/sda3
```

3.3.3.2. The new scheme: mounted partition

Following is a practical example using the new way of doing things, by using a formatted **PALO** partition. That scheme should be the preferred one for new installations.

Looking at the previous example (Section 3.3.3.1), very little things need to be changed. Essentially, if you had the need for a separate `/boot` partition, it will be gone in the scheme detailed below. The configuration for **PALO** will be a little bit different as well, but that's about it.

Here is the output of **fdisk** which represents the hard drive of a box with 100MB **PALO** space (which will be mounted as `/boot`), 128MB swap space and about 1GB '`/`' partition (bear in mind that the `f0` partition must still be wholy contained within the first 2GB of the disk):

```
bash# fdisk -l /dev/sda

Disk /dev/sda: 133 heads, 62 sectors, 1017 cylinders
Units = cylinders of 8246 * 512 bytes

         Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *         1        26    100461   f0  Linux/PA-RISC boot
/dev/sda2            27        36    123690   82  Linux swap
/dev/sda3            37       277    917889   83  Linux
```

Now let's deal with **PALO** configuration. Contrary to the previous example, there are far less options to put in the configuration file, and the setup is a single step operation: the *initialization* step, which needs only to be

done once.

To initialize for the first time the **PALO** partition as something the Linux system can understand (`ext2` or `ext3` filesystem), you need to run the following command (as `root`):

```
[user@machine ~/dir]> palo --format-as=N --init-partitioned=target_disk
```

Where *`N`* is either `2` for `ext2` or `3` for `ext3`, and *`target_disk`* is the device which contains the **PALO** partition, `/dev/sda` in the current example. Continuing with this example, assuming we want an `ext3` partition we would use:

```
[user@machine ~/dir]> palo --format-as=3 --init-partitioned=/dev/sda
```

Do *NOT* use **mkfs** or **mke2fs** to generate the filesystem on this partition. **PALO** marks blocks as used where the boot loader portion of **PALO** is stored on disk. These tools don't know about **PALO** boot loader!

This needs to be run only once for it will erase any existing data on the `f0` partition.

Finally, we need to tell **PALO** about the partition, how we want it to be used, and how we don't want it to be erased everytime **PALO** is being run. Thus, keeping in sync with our current example, the configuration file should look like that:

```
# The following arguments are set up for booting from /dev/sda, specifically
# mounting partition 3 as root and booting the vmlinux file in /dev/sda1, palo
# partition formatted as ext3.
--update-partitioned=/dev/sda --format-as=3
--commandline=1/vmlinux root=/dev/sda3
```

As one can see, since we will be storing our kernels directly on the `f0` partition, we tell **PALO** to load them from it (hence the `1` in the `commandline` argument.

In the previous section (Section 3.3.3.1) we dealt with the *`init-partitioned`* parameter. Here, we use *`update-partitioned`* instead, which, contrary to the former, tells **PALO** to not erase the content of the partition when run[4].

3.4. How to use PALO at early boot stage?

3.4.1. The theory

You have setup everything, rebooted your box, and suddenly you want to change something to the kernel boot arguments, or even boot another kernel. Damn it! How could you, now that the box is booting? Well, stay calm and relax, have a deep breath, we have the solution! Unfortunately, by the time you'll learn about it, your box will have finished booting ;o)

First, you must learn how to interact with **PALO** during the startup sequence. You have to enter **BOOT_ADMIN**, as explained in Section 2.1.1. For some old models (up to 712 or so), you must add the *`ipl`* (or *`isl`*) string to your boot command in the **BOOT_ADMIN** console:

```
BOOT_ADMIN> boot pri ipl
```

On most PA−RISC boxes, the system will ask you if you want to interact with *IPL* anyway. You just have to answer "**y**" and hit **Enter**. You will then end up to **PALO** configuration display, with the list of all parameters and their corresponding numbers.

You just have to enter the number corresponding to the parameter you want to change. Hit **Enter**, modify it and validate the changes by hitting **Enter** again. The system will redisplay the new list. This modification is not permanent[5]! If you want to add a supplementary parameter, select any one and write yours on the editing line, beginning with a space:

```
<#>    edit the numbered field
'b'    boot with this command line
'r'    restore command line
'l'    list dir
? 0
3/boot/vmlinux initrd=3/initrd.img
```

After validation, the list will count one more parameter. If you want to delete one, select it and erase the complete entry. You will see that the list counts one less parameter.

For more informations about **PALO**, please take look at the **PALO** README. You can find a copy of this file after having installed the **palo** package in `/usr/share/doc/palo/README.html`. This HOWTO section is mostly inspired from the above file, written by Paul Bame.

3.4.2. A complete example

This example has been suggested by Michael Damaschke. We will use notions explained in Section 2.1 and Section 3.4.1, and refer to concept such as *console*, seen in Section 2.2. So, let's go for the story of the happy PA/Linux user booting a kernel, also called "*I don't know how to configure my workstation to use the kernel I want during boot sequence!*".

After switching your workstation on, a message on the console will tell you that the workstation is about to start automatically the boot sequence, except if you hold the **Esc** key to stop the auto−booting process. This is a very difficult step: you must hold the **Esc** key down ;o)

Depending on your model, you might need to press this key during a quite long time.

In some cases when using graphic console, the monitor can be too slow to trigger on, and won't allow you to see the warning message. A good workaround is to keep a close eye on the keyboard's lights: when they all blink at once, this is the right time to press and hold the **Esc** key. If you still have troubles, please refer to Section 2.2.

There are a few different ways to get access to **BOOT_ADMIN** (see Section 2.1.1). If you have an old box, you will see an information message displayed, where the workstation's firmware tells you that it will start searching for all bootable devices, or that you can break this by holding down the **Esc** key. This is the same procedure as just mentioned, you must press the **Esc** key.

As usual, on some machines you might then get a menu where you should press the **a** key followed by **Enter**. You are now facing the deadly '`BOOT_ADMIN>`' prompt :^). First, we will turn off `auto boot` process by entering the following lines:

```
```

```
BOOT_ADMIN> auto boot off
```

then hit **Enter** to validate. This will prevent the box from further attempts at auto–booting. In other words, you won't have to stop the boot process with **Esc**, it will stop on its own on subsequent reboots and wait for your instructions.

Now, you must tell the system from which boot device you would like to boot. If it's a hard drive, it must have a '`f0`' partition at the beginning (see Chapter 4).

In this example, the old kernel is `vmlinux` and the new one is `vmlinux-2.4.17-pa3`. The chosen SCSI boot device is designed by: `SCSI.X.0`, where *X* is the SCSI–ID of the disk you want to boot from[6]. *e.g.:*

```
BOOT_ADMIN> boot SCSI.5.0
```

At the end of the previous command line, you must add the *IPL* token if you have a HP 9000/7xx system to specify that you want to interact with IPL. If you have a more recent hardware, the system will ask if you want to interact with IPL anyway:

```
Interact with IPL (Y or N)?>
```

Say **Y** and hit **Enter**. Now, you can manually configure the **PALO** boot parameters. A new menu is displayed, where you can configure on line '`0`' (selected by default) the boot partition number, and the path of your boot kernel.

Here is the complete session log of a A500 serial console output, taken from **PALO** version 1.5. You can find in Section B.1 a session log with an older version of palo, such as the one that can be found on Debian 3.0 install disks.

```
Main Menu: Enter command or menu > bo scsi.5.0
Interact with IPL (Y, N, or Cancel)?> y

Booting...
Boot IO Dependent Code (IODC) revision 1


HARD Booted.
palo ipl 1.5 root@c3k Fri May 14 16:17:38 MDT 2004
Skipping extended partition 6 - beyond reach of IPL

Partition Start(MB) End(MB) Id Type
1               1      31   f0 Palo
2              32     153   83 ext2
3             154    1107   82 swap
5            1108    5875   83 ext2

PALO(F0) partition contains:
    0/vmlinux64 5279419 bytes @ 0x44000

Information: No console specified on kernel command line. This is normal.
PALO will choose the console currently used by firmware (serial).
Current command line:
2/vmlinux root=/dev/sdb5 HOME=/ console=ttyS0 TERM=vt102
 0: 2/vmlinux
 1: root=/dev/sdb5
 2: HOME=/
 3: console=ttyS0
 4: TERM=vt102

<#>    edit the numbered field
```

```
'b'    boot with this command line
'r'    restore command line
'l'    list dir
? 0
2/vmlinux-2.6-cvs initrd=2/initrd.img-cvs
Current command line:
2/vmlinux-2.6-cvs initrd=2/initrd.img-cvs root=/dev/sdb5 HOME=/ console=ttyS0 TERM=vt102
 0: 2/vmlinux-2.6-cvs
 1: initrd=2/initrd.img-cvs
 2: root=/dev/sdb5
 3: HOME=/
 4: console=ttyS0
 5: TERM=vt102

<#>    edit the numbered field
'b'    boot with this command line
'r'    restore command line
'l'    list dir
? 1

Current command line:
2/vmlinux-2.6-cvs  root=/dev/sdb5 HOME=/ console=ttyS0 TERM=vt102
 0: 2/vmlinux-2.6-cvs
 1: root=/dev/sdb5
 2: HOME=/
 3: console=ttyS0
 4: TERM=vt102

<#>    edit the numbered field
'b'    boot with this command line
'r'    restore command line
'l'    list dir
? b
```

PALO was first setup to boot the kernel file `vmlinux` located on the second partition of the SCSI device ID 5 LUN 0. (We know this since we have asked **BOOT_ADMIN** to boot on this device). But we wanted another kernel this time. We have pressed the **Enter** key (to validate the default choice '`0`') and modified the text to match our needs, here `vmlinux-2.6-cvs`. We have also added an `initrd=2/initrd.img-cvs` argument to the command line. We have validated our changes by hitting the **Enter** key. Finally we have decided that we didn't want this additional argument, so we have selected it and erased it. At the end it asked again which field we wanted to edit, we just typed '`b`' instead of any number and hit **Enter** to boot our new kernel.

Please don't change any other parameter unless you really know what you are doing!

That's it! **PALO** has no more secrets for you :–)

As you might have noticed, the **BOOT_ADMIN** interface can take several aspects, so don't be disappointed if yours does not exactly match our examples.

Chapter 4. Available boot solutions

4.1. Booting from CD

Booting from CD is one of the easiest way to start and install your PA−RISC machine; assuming you have a CD drive handy and a bootable CD. You can download official Debian ISOs as well as *Net Install* ISO (see *netinst*) from the Debian Installer pages, or from the PA−RISC/Linux official website.

i. Start the box and enter the **BOOT_ADMIN** mode. (Section 2.1.1)
ii. Place your bootable CD on the CD tray and close it. Sounds obvious, but we know guys who missed that step :)
iii. There are two options from there: either you know the full `PATH` to your CD device, then you can jump to next step, or you don't. In this last case, issue a **search ipl** to list all available bootable devices. You can also specify **search *[PATH]***, which is fastest. For instance if you want to search the SCSI bus:
```
search SCSI
```
On recent boxes, **search disk** is quite helpful. Take a look at **help search** for details specific to your box.
iv. Once you know the full `PATH` to your CD drive, you can issue a **boot *<PATH>***. That's all. If everything goes fine, it will start booting the CD present in the CD reader. Real life example:
```
boot ide
```

4.2. Booting from hard drive

Booting from hard drive is not really more difficult that booting from CD. The only thing really important is that your hard drive has to be correctly prepared. Take a look at Section 3.3.3 to learn how to prepare it.

i. Start the box and enter the **BOOT_ADMIN** mode. (Section 2.1.1)
ii. There are two options from there: either you know the full `PATH` to your hard disk device, then you can jump to next step, or you don't. In this last case, issue a **search ipl** to list all available bootable devices. You can also specify **search *[PATH]***. For instance if you want to search the Single Ended SCSI bus:
```
search SESCSI
```
Take a look at **help search** for details specific to your box.
iii. Once you know the full `PATH` to your hard drive, you can issue a **boot *<PATH>***. That's all. If everything goes fine, it will start booting the kernel as setup by **PALO** (see Section 3.3.3). Real life example:
```
boot scsi.6
```

4.3. Booting from network

Booting from the network is only required in certain cases. Booting from the network is very usefull when you have unsupported I/O devices, diskless systems, or systems with broken hardware. Network booting is detailed below.

4.3.1. Preparing to boot from network

Booting from the network involves two machines: the *boot server* and the *boot client*, the latter being the PA–RISC system you are trying to start up, and the former, the machine that will serve over the network the files which the client needs. The rest of this section will extensively deal with setting up the *boot server* since this is probably the trickiest part.

You will need a *lifimage* to perform a network boot. See Section 3.2 to learn how to create one. You can also use the one from Debian Installer.

Needless to say, all server–side setup is meant to be performed by the *super–user*, also known as `root`.

4.3.2. rboot or bootp?

All 'recent' machines can boot using `BOOTP`, starting from 715/100, 715/120, and 712s. Older ones, mostly early 715s, 710s and 725s need `RBOOT`.

To use `BOOTP` you have to enable the IP: Kernel level autoconfiguration–>IP: BOOTP support within the '`Networking options`' section of the kernel configuration, if you want to use a *home–made* kernel. See Chapter 5 for details.

Please note that though Section 4.3.3 deals with `RBOOT` only, two different implementations of the `BOOTP` protocol are detailed in Section 4.3.4 and Section 4.3.5. We detail these two *because we can*, but if you need to use the `BOOTP` protocol, you will have to choose one.

If you don't know which `BOOTP` implementation to use, go for the **dhcp** one, it is much easier to deal with.

4.3.3. Using rboot

4.3.3.1. Obtaining rbootd

If you have an old machine that requires **rboot** to boot over network, use the following procedure to set up and configure a boot server, and boot using the PA–RISC/Linux kernel.

Old machines, including the Scorpio 715s, use the `RBOOT` protocol. You need **rbootd** to handle their boot requests. Look for it in your favorite distribution archive (assuming you will be servicing boot requests from a Linux box). Here are two ways of getting the **rboot** daemon:

- If you are using a Debian–powered server (which you really should be doing ;o), you're almost done. Run from a command shell:
  ```
  [user@machine ~/dir]> apt-get install rbootd
  ```
- If you can't find any **rbootd** package for your system (which is very possible since it is a very old netboot protocol), you can find its source in the Debian archive: rbootd. You will have to build it from source.

4.3.3.2. Configuring rbootd

As we already said, to boot a RBOOT−aware system, you need a separate machine with **rbootd** installed (this is the 'boot server') on which you will store the PA−RISC/Linux kernel *lifimage* that you want to use to boot your PA−RISC system with.

Once the **rbootd** server software is installed, read the following to configure it to work with your PA−RISC system:

In `/etc/rbootd.conf` you will have to add a line like:

```
ethernet_addr            bootfile
```

1. Replace *bootfile* with the name of your PA−RISC/Linux kernel image, usually '`lifimage`'.
2. Now get the Ethernet address of your PA−RISC system by typing **lanaddress** at the '`BOOT_ADMIN>`' prompt (see Section 2.1.2.3).

   It will return a number like `080009−7004b6`. Take note of this number.
3. In `/etc/rbootd.conf` on your boot server, the Ethernet address has to be colon−delimited. That means you will have to modify the number you just obtained so that every set of two characters (after removing the '−') is separated by a colon. For example: `080009−7004b6` becomes `08:00:09:70:04:b6`. Add the colon delimited Ethernet address to `/etc/rbootd.conf` on your boot server. The resulting file will look something like this:

   ```
   # ethernet addr        boot file           comments

   08:00:09:87:e4:8f      lifimage_715        # PA/Linux kernel for 715/33
   08:00:09:70:04:b6      lifimage_720        # PA/Linux kernel for 720
   ```

   This `rbootd.conf` example contains the Ethernet addresses and boot file names for two different machines.

   Once you have changed the configuration file, restart **rbootd**.

By default, **rbootd** assumes that bootfiles are located in `/var/lib/rbootd/`. Therefore, you will have to put your bootable kernel image in that directory, or, if you really hate that directory for some reason, you can rebuild **rbootd** to use a different directory.

The easiest thing, of course, is just to drop your kernel images in the default directory!

4.3.4. Using dhcp/tftp

We will see here how to setup a DHCP server to handle BOOTP requests (since PA−RISC box use BOOTP, unless they need RBOOT, as mentioned above).

Windows" users might want to look at Appendix A.

4.3.4.1. Obtaining dhcp/tftp

Debian users will just have to install the packages using the following commands, executed as `root`:

```
[user@machine ~/dir]> apt-get install dhcp tftpd
```

If you need rpm packages (for the ISC dhcp server), the best way is to go to http://rpmfind.net/.

The dhcp package can do much more than a simple bootp daemon. Nevertheless, it is also known to be much easier to configure. If you really want to try regular **bootp**, skip this and go to Section 4.3.5.

4.3.4.2. Configuring dhcp/tftp

Here are the instructions to set up **dhcp** on your boot server. To keep this explanation simple, we will assume that you want to assign a fixed IP to your box, without DNS update. Your subnet will be `192.168.1.0/24`, with optional: gateway at `192.168.1.1`, domain name `foo.com` and DNS at `192.168.1.4`. Feel free to replace these values with those which would suit your needs in the next sections.

This section is dedicated to Debian users. For others distributions, it should be similar though there may be some differences like default directories.

1. Edit `/etc/inetd.conf` on your boot server to add the following line, if it doesn't already exist:
   ```
   tftp           dgram   udp     wait    nobody  /usr/sbin/tcpd \
   ```
   Here, `/tftpboot/` is being used as tftpd server's root (this is where you will put the *lifimage* file). You can choose another directory if you want. According to **man tftpd**, this is the usual default directory.

   When this is done, reload **inetd** with: **/etc/init.d/inetd reload**. Non–Debian users can also issue a **killall –HUP inetd**.
2. According to **man 5 dhcpd.conf**, edit the `/etc/dhcpd.conf` file to contain something like:
   ```
   allow bootp;
   default-lease-time 600;
   max-lease-time 7200;
   # This will tell the box its hostname while booting:
   use-host-decl-names on;

   subnet 192.168.1.0 netmask 255.255.255.0 {
                 option routers 192.168.1.1;
                  option domain-name "foo.com";
                  option domain-name-servers 192.168.1.4;
   }

   host [hostname] {
                  hardware ethernet [mac address];
                 fixed-address [ip address];
            filename "[boot filename]";
                  option root-path "[root path]";
   }
   ```
   You have to fill in the *[hostname]*, *[mac address]*, *[ip address]*, *[boot filename]* and *[root path]* fields with the appropriate information, where:

- *[hostname]* is the name of the PA–RISC system.
- *[mac address]* is the colon–delimited ethernet address of the PA–RISC system, which can be obtained by typing **lanaddress** at the 'BOOT_ADMIN>' prompt (see Section 2.1.2.3).
- *[ip address]* is the IP address you wish to assign to the PA–RISC system.
- *[boot filename]* is the name of the bootable kernel image you want to boot your system with.
- *[root path]* is the path to the NFS root filesystem exported by the server.
- Additionally, if the tftp server you want to use is not the same as the one running the dhcp server, you can add `next-server [ip address];`, replacing *[ip address]* with the actual IP of the tftp server, to the dhcp configuration.

You'll end up with something like this for each box you want to *netboot*:

```
host tatooine {
                hardware ethernet 00:40:05:18:0c:dd;
                fixed-address 192.168.1.22;
                filename "lifimage-tatooine";
                option root-path "/exports/tatooineroot";
}
```

4.3.5. Using bootp/tftp

4.3.5.1. Obtaining bootp/tftp

For Debian users, you just have to install the packages by typing these commands as user `root`:

```
[user@machine ~/dir]> apt-get install bootp tftpd
```

If you need rpm packages, the best way is to go to http://rpmfind.net/.

You'll have been warned! This daemon is far more obfuscated in its configuration.

4.3.5.2. Configuring bootp/tftp

Follow these instructions to use the **bootp** daemon on your boot server:

This section is dedicated to Debian users. For others distributions, it should be similar though there may be some differences like default directories.

1. Edit `/etc/inetd.conf` on your boot server to add the following lines, if they don't already exist:

```
tftp            dgram   udp     wait    nobody  /usr/sbin/tcpd      \
                                                                    /usr/
bootps          dgram   udp     wait    root    /usr/sbin/bootpd    \
```

   Here, `/tftpboot/` is being used as tftpd server's root (this is where you will put the *lifimage* file). You can choose another directory if you want. According to **man tftpd**, this is the usual default directory.

   When this is done, reload **inetd** with: **/etc/init.d/inetd reload**. Non–Debian users can also issue a **killall –HUP inetd**.

2. According to **man 5 bootptab**, edit the `/etc/bootptab` file to contain:

```
[hostname]:hd=/tftpboot:\
    [root path]:\       :rp=
                        :ht=ethernet:\
    [mac address]:\     :ha=
    [ip address]:\      :ip=
    [boot filename]:\   :bf=
                        :sm=255.255.255.0:\
                        :to=7200:
```

You have to fill in the *[hostname]*, *[mac address]*, *[ip address]* and *[root path]* fields with the appropriate information, where:

- *[hostname]* is the name of the PA−RISC system.
- *[mac address]* is the NOT−delimited ethernet address of the PA−RISC system, which can be obtained by typing **lanaddress** at the 'BOOT_ADMIN>' prompt (see Section 2.1.2.3).
- *[ip address]* is the IP address you wish to assign to the PA−RISC system.
- *[boot filename]* is the name of the bootable kernel image you want to boot your system with.
- *[root path]* is the path to the NFS root filesystem exported by the server.

You'll end up with something like this:

```
vodka:hd=/tftpboot:\
         :rp=/usr/src/parisc/:\
         :ht=ethernet:\
         :ha=080069088717:\
         :ip=140.244.9.208:\
         :bf=lifimage:\
         :sm=255.255.255.0:\
         :to=7200:
```

4.3.6. Effectively booting from network

To conclude with the developers' way to boot the kernel, this section will tell you how to actually boot your system from a network server. But it tends to be less and less used. Most users will prefer to stick to Section 4.2 once their system is properly setup.

Here we are. We assume that you've done everything outlined above, your network boot server is on the same physical subnet as your PA−RISC machine, you've got a bootable PA/Linux kernel lifimage on your boot server, and you're willing to give it a try. If everything is ready (including you!), the following procedure will introduce you to the joy of network booting your PA box into Linux.

1. Fire up your PA−RISC system.
2. Watch your PA−RISC box starting up. When the following message appears during the PA−RISC machine's boot process, press and hold the **Esc** key:

```
Searching for Potential Boot Devices.
To terminate search, press and hold the ESCAPE key.
```

3. If needed, select 'a) Enter Boot Administration mode' from the menu. This brings up the 'BOOT_ADMIN>' prompt.
4. Type the following at the prompt: **boot lan**.

5. Watch your PA−RISC system magically becoming a PA/Linux system. Ta dah!

Of course your are supposed to run only one boot server at a time on your network, in order to avoid conflicts...

Chapter 5. Building and installing a custom kernel

To build a Linux kernel, you need a compiler and the kernel source. The first element is not a trivial thing to find because it depends on how you want to build your kernel. The second is easier since it can be found at the official CVS site. First, we will discuss about **GCC** compiler. Then, the preparation of the build will be explained. The last paragraph will deal with the installation of this new kernel.

We will deal only with a kernel built without modules, to simplify the explanations.

5.1. GCC compiler

You can build the kernel directly on your own PA−RISC box (*self−hosted* or *native* build). But on old systems, you may prefer to use another – faster – non PA−RISC computer to compile your kernel (*cross−compilation*). We will see the two possibilities.

By the time version 1.0 of this howto was released, only gcc−3.0.X was able to build working kernels. There was a bug in more recent versions that made the box crash when network activity occurs. It should be fixed by now, so using the latest version of gcc should be fine. If ever the above mentioned bug occurs, you'll know what's wrong. Anyway, if you want to build any kernel after 2.6.12−rc3, you will need at least **gcc−3.3**.

5.1.1. Native build

Since Debian was the first distribution to support PA−RISC architecture, if you want to use the *Super Cow* powers, you need to have some basic knowledge about the Debian packaging system. We will explain here how to quickly get a gcc compiler ready on your PA−RISC box. If you are not using Debian, well, we're afraid we can't do much for you: you will have to transpose what is said below to your distribution. We will assume you know how to use

If you are using your own PA−RISC box, you only need the good old **GCC** compiler. You can install the required tools to build a kernel by issuing:

```
[user@machine ~/dir]> apt-get install build-essential libncurses5-dev
```

Essentially, this will install everything you need to build a kernel (and even a bit more). This boils down to **binutils**, **gcc**, **libc−dev**, **make**, **fileutils** and **libncurses5−dev**.

When this is done, you can proceed to the kernel settings.

5.1.2. Cross compiled build

In this kernel build method, everything depends on the architecture of your building machine. If you want to compile your own toolchain, there is a slightly out−of−date HOWTO (O'Donell 2002). Otherwise, we assume you can either find a cross−compiler package for your build host, or make one by yourself.

As there is not yet a 64bit userspace on HP−PA, you have to cross−compile 64bit kernel even if you are building on a 64bit PA−RISC box. You can get unofficial debs for hppa64 compilers and binutils by running for instance:

```
[user@machine ~/dir]> apt-get install gcc-3.3-hppa64 binutils-hppa64
```

See the PA−RISC Linux Website for details.

5.2. Kernel configuration

If you want to take advantage of the latest kernel improvements, we suggest you retrieve it from the official PA−RISC/Linux CVS. Please mind that the *vanilla* kernel that can be found at http://www.kernel.org/ is generally out of sync with the above mentioned CVS kernel, and that snapshots of this kernel are available too, check the download area. In the following, we will focus on a fresh CVS tree.

The best way to obtain appreciable performances is to get a well configured kernel. For the PA−RISC platform, **make oldconfig** is a kind of default setup. If you want to make your own kernel, the first step is to know what hardware you have. The best way to grab useful info is to look at your box and find a maximum of data (model name, partnumber, chipsets, and so on). If you have already booted your box, you can take a look at **dmesg** output. Then, go to the official hardware database or to the HP partsurfer website.

Once you know what is inside your box and what you want to do with it, just run **make menuconfig** or another config command.

5.2.1. Configuring 2.4 kernels

Here is a brief list of architecture dependent menus for 2.4 kernels. You should take a look at them, to see if the values set match your hardware. Mind that 2.4 kernels are now considered *deprecated* anyway: you will not get community support for them.

Remember that **make oldconfig** is a good base to start with, since it works for almost any machine.

- *Processor type* – indicates your CPU model
- *General options* – tells you what is going to be enabled in your kernel (U2/Uturn, USC/GSC/HSC, Lasi, Wax, Dino, LBA/Elroy, SuperIO)
- *Parallel port support* – enables/disables the Lasi/ASP parport
- *SCSI support* – check there for your SCSI chipset (Lasi, Zalon, NCR/SYM53C8XX or other)
- *Network device support* – is used to set your network card (Lasi, Tulip...)
- *Character devices* – defines your I/O capabilities (Lasi, Dino, MUX see Section 5.2.1.3)
- *HIL Support* – useful if you have a HIL controller. See below Section 5.2.1.1.
- *Console drivers* – is directly related to your console mode (STI console or STI framebuffer)
- *Sound* – enables/disables the Harmony driver

As you can see, menus specifically concerned by PA−RISC hardware are not that numerous, but there are lots of dependencies between them. Now, you must configure the kernel accordingly to what you plan to use this box for. Here is a list of some menus you should be going through to configure additional functionalities you might want:

- *General setup* − is responsible for binary formats handled by the kernel. You need ELF, and can try SOM (support for HP/UX binaries. It *might* work with some static executables).
- *Block devices* − sets the ramdisk and loopback support. You probably won't use them.
- *ATA/IDE/MFM/RLL support* − You will need to check this to enable IDE. See Section 5.2.1.4.
- *File Systems/Network File Systems* − is where to set EXT3 or NFS support.
- *USB support* − If you have enabled *SuperIO* and want USB, look here: Section 5.2.1.2.

By the time this HOWTO was written, there was no floppy drive support; and what's more, it is not expected to ever be supported.

When you're done with it, save your kernel configuration. Everything is written in the `.config` file. You should back it up because **make distclean** will remove it. At this stage, you can do **make dep vmlinux** and if everything goes fine, you will have a new kernel in a couple of minutes.

Here follows brief information about specific hardware configurations.

5.2.1.1. HIL Support in 2.4

Since `kernel-2.4.18-pa45`, there is a full HIL support, for mice, tablets and keyboards. It is based on the *Linux Input Driver* model. See the PA−RISC/Linux FAQ and the mail posted on the mailing list by *Helge Deller*. Here is how to configure it:

1. Make sure you have a 2.4.18−pa45 or higher kernel source.
2. Look at your kernel configuration for the following options:
```
CONFIG_INPUT=y
CONFIG_INPUT_KEYBDEV=y
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1024
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=768
CONFIG_INPUT_EVDEV=y

CONFIG_INPUT_SERIO=y

CONFIG_HIL=y
CONFIG_HP_SDC=y
CONFIG_HIL_MLC=y
CONFIG_HP_SDC_MLC=y
CONFIG_HIL_KBD=y
CONFIG_HIL_PTR=y
```

There is no more `CONFIG_HIL_KBD_BASIC`.

3. On your target system, check that the following devices are available:
```
/dev/input/mice
/dev/input/mouseX
/dev/input/eventX
```
If they are not yet present, create them as `root` by running:
```
[user@machine ~/dir]> cd /dev; MAKEDEV input
```
4. Configure **gpm** with the following options in `/etc/gpm.conf`:
```
device=/dev/input/mice
type=imps2
```
5. Here is a sample `/etc/X11/XF86Config-4`:

```
Section "InputDevice"
                Identifier      "HIL Keyboard"
                Driver          "keyboard"
                Option          "CoreKeyboard"
EndSection
Section "InputDevice"
                Identifier      "HIL Mouse"
                Driver          "mouse"
                Option          "CorePointer"
                Option          "Device"                "/dev/input/mice"
                Option          "Protocol"              "ImPS/2"
                Option          "ZAxisMapping"          "4 5"
EndSection
Section "ServerLayout"
                Identifier      "Default Layout"
                Screen          "Default Screen"
                InputDevice     "HIL Keyboard"
                InputDevice     "HIL Mouse"
EndSection
```

You can also download a sample `XF86Config-4` here: ftp://ftp.parisc-linux.org/XFree86/XF86Config-4, adjust color depth and resolution, and put it in your `/etc/X11/`.

5.2.1.2. USB Support in 2.4

USB support on HP-PA is still experimental, therefore it is only configured as modules in default kernel configuration. We have tried to install a B2000 with builtin USB support, both 32 and 64bit, and it worked fine, despite some keyboard problems. Don't worry, nothing critical: the range of keys located between the main part of the keyboard (the letters, backspace, enter...) and the numeric pad are broken. They do not behave at all as expected.

You can use the numeric pad as arrow keys: when **NumLock** is not activated, it behaves as a navigation pad. *e.g.* **8** is **Up Arrow**, **4** is **Left Arrow** and so on.

1. Make sure you have a 2.4.18 or higher kernel source.
2. Look at your kernel configuration for the following options:

```
CONFIG_SUPERIO=y
CONFIG_HOTPLUG=y

CONFIG_INPUT=y
CONFIG_INPUT_KEYBDEV=y
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1024
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=768
CONFIG_INPUT_EVDEV=y

CONFIG_USB=y
CONFIG_USB_DEVICEFS=y
CONFIG_USB_OHCI=y
CONFIG_HID=y
```

3. On your target system, check that the following devices are available:

```
/dev/input/mice
/dev/input/mouseX
/dev/input/eventX
```

If they are not yet present, create them as `root` by running:

```
[user@machine ~/dir]> cd /dev; MAKEDEV input
```

4. Configure **gpm** with the following options in `/etc/gpm.conf`:

```
device=/dev/input/mice
type=imps2
```

5. The XF86−Config−4 is similar to the `HIL` one, as it is also using the *Linux Input Driver*.

5.2.1.3. MUX Console Support in 2.4

`MUX Console` has been improved by *Richard Hirst* in 2.4.18−pa37 kernel, though it is still a very *experimental* feature. It is expected to provide adequate `MUX Console` support to `E−` and `K−Class` machines. Feedback would be really appreciated.

Now follow these steps to get it to work:

1. Make sure you have a 2.4.18−pa37 or higher kernel source.
2. Look at your kernel configuration for the following options:

```
CONFIG_SERIAL_CONSOLE=y

CONFIG_SERIAL_GSC=y

CONFIG_SERIAL_NONSTANDARD=y
CONFIG_SERIAL_MUX=y
```

3. On your target system, check that the following devices are available:

```
/dev/ttyB0
```

If they are not yet present, create them as `root` by running:

```
[user@machine ~/dir]> cd /dev; MAKEDEV ttyB0
```

It needs a recent `MAKEDEV` package to be created this way.

4. Now you can boot your system, taking care that **PALO** uses `console=ttyB0`.

5.2.1.4. IDE Devices Support in 2.4

There is nothing really special about IDE support. You have to check that the *IDE Chipset* in use in your box is supported by the kernel. A common chipset found on PA−RISC hardware is `NS87415`. You can find it on B2000, J5000 and C3000 for instance. You will need IDE support to use some CD−ROM devices.

Here is an example to get IDE to work with this chipset:

1. Make sure you have a recent kernel source.
2. Look at your kernel configuration for the following options:

```
CONFIG_IOMMU_CCIO=y
CONFIG_PCI=y
CONFIG_PCI_LBA=y
CONFIG_IOSAPIC=y
CONFIG_IOMMU_SBA=y
CONFIG_SUPERIO=y
```

```
CONFIG_IDE=y

CONFIG_BLK_DEV_IDE=y

CONFIG_BLK_DEV_IDEPCI=y
CONFIG_BLK_DEV_IDEDMA=y
CONFIG_BLK_DEV_ADMA=y
CONFIG_BLK_DEV_IDEDMA=y
CONFIG_BLK_DEV_NS87415=y
```

3. On your target system, check that the following devices are available:
```
/dev/hd*
```
If they are not yet present, create them as `root` by running:
```
[user@machine ~/dir]> cd /dev; MAKEDEV hda hdb hdc hdd hde
```

Of course we didn't mention much of the architecture independent options. Moreover, the above settings may vary depending on your hardware. This is just an example.

5.2.2. Configuring 2.6 kernels

Here is a brief list of architecture dependent menus for 2.6 kernels. You should take a look at them, to see if the values set match your hardware:

- *Processor type and features* − indicates your CPU model and some specific features such as SMP or Discontigmem support
- *Bus options* − tells you what bus support is going to be enabled in your kernel (U2/Uturn, USC/GSC/HSC, Lasi, Wax, Dino, LBA/Elroy, SuperIO)
- *PA−RISC specific drivers* − enables/disables some PA−RISC specific drivers, such as LED support, GSP and Stable Storage support.

As you can see, menus specifically concerned by PA−RISC hardware are not that numerous, and everything else is much generic by now. Still, you must configure the kernel accordingly to what you plan to use this box for and what features you want supported. Many other drivers are found in their respective submenus, such as SCSI, with the Zalon, Lasi SCSI and SYM2 drivers being there, or the Framebuffer devices (STI) in the Graphics Support menu, or the sound drivers (Harmony and AD1889) in the Sound menu. Help is often provided, feel free to look at it.

Most of what was said for 2.4 is somewhat still applicable to 2.6.

5.3. Kernel installation

If you have made a native build on the box you wish to install, you can setup the new kernel as follows: within the kernel source tree `linux/`, as `root` execute:

```
[user@machine ~/dir]> cp vmlinux /boot/vmlinux-[kernelversion]
[user@machine ~/dir]> cp System.map /boot/System.map-[kernelversion]
[user@machine ~/dir]> cp .config /boot/config-[kernelversion]
```

Though it is not mandatory, we suggest you to replace *[kernelversion]* by the version of the kernel you built, *e.g.*: `vmlinux-2.4.18-pa44`. This will help you dealing with multiple kernel versions on the same machine. The same applies to `.config`. It is not needed to have a working kernel, though it might be very helpful when configuring a new one. Now, do **cd /boot**, make sure that `vmlinux` is a symbolic link to another file, as in the following example:

```
[user@machine ~/dir]> ls -l vmlinux
lrwxrwxrwx  1 root   root   35 Jun 23 01:38 vmlinux -> vmlinux-2.4.18-64-SMP
```

Make sure to remember the name of the kernel actually running on your box if ever the new one won't work properly. You are now able to ask **PALO** to boot on it if needed (see Chapter 3 for more information). Now do the following:

```
[user@machine ~/dir]> rm -f vmlinux
[user@machine ~/dir]> ln -s vmlinux-[kernelversion] vmlinux
[user@machine ~/dir]> sync
```

If you want to boot from network you can forget all this, as you will need to set **PALO** as explained in the Section 3.3, and run **make palo** to create the bootable *lifimage*.

If you have made a cross–compiled build or built a kernel on a PA box which is not the one you wish to install, you have to find a way to put `vmlinux`, `System.map` and eventually `.config` in `/boot/` as mentioned before. You can use the network (like **ftp**) or a CD to do so, or even direct copy to the hard disk drive.

Appendix A. Windows" 2003 boot server howto

This appendix has been greatly contributed by *Jeremy Drake*. It describes the process of setting up a Windows" 2003 Server to serve boot requests for a PA/Linux box.

A.1. Setup the DHCP service

As for the UNIX/Linux based approach (discussed in Section 4.3), you need to collect some information and data before setting everything up. First of all, you need the MAC address of your PA−RISC box. Please check **rboot** preparation for details. You are going to need a *lifimage* file. Please read Section 4.3.1.

Then, you have to enable DHCP service on your Windows" box. You can do that by going into the *Control Panel*, open *Add/Remove Programs*, select *Windows Components* and finally *Networking Services*. There, enable *Dynamic Host Configuration Protocol (DHCP)*.

You need to configure the DHCP service now. Launch the DHCP admin tool by going into the *Control Panel*, open *Admin Tools* and select *DHCP*.

- Expand the server tree.
- Right click on Reservations−>New Reservation...
- In *Reservation name*, put the workstation's host name. Enter an unused IP address. Enter the PA−RISC box' MAC address (no delimiters, just the hex number). Select *Both* for whether it should be bootp or dhcp. Click *Ok* to close this window.
- Look for your newly created reservation at the bottom of the list under *Reservations* and click it.
- Right click on *Configure Options...*
- It should have inherited your server's default options, so we won't cover setting router, dns, wins and lease length.
- Scroll down the list of options to 066: *Boot Server Host Name*. Check the box next to option 066. Enter your tftp server's ip address, because IPL can't resolve hostnames.
- Check option 067: *Bootfile Name* and enter the name of the lifimage. Generally, *lifimage* is a good choice here.
- Click *Ok* and your dhcp server is ready!

A.2. Get & setup the TFTP server

To get the network boot process operational, you need the TFTP service that provides the basic file system at boot time. Get Tftpd from http://tftpd32.jounin.net/. You must download the latest version in zip format. Unzip it and store it in your favorite place. Then, you must setup the monster.

- Run tftpd32.
- Click the *Browse* button.
- Browse to where you put your lifimage, highlight it and click *Ok*.
- Make sure the IP address below the directory is the one you gave to your PA−RISC box.
- Let tftpd32 open. The tftp server only runs when the GUI is displayed.

If you want to run it as a NT service, you have to download a Microsoft" program. Please refer to the \ Tftpd32 FAQ.

A.3. Attempt to netboot

Now, you are fully set up to try to boot your PA−RISC box via network. You can follow the instructions in Section 4.3.6.

If you have any trouble, start by looking at those points and then ask the PA/Linux mailing list.

- Settings on the DHCP server (verify the PA−RISC MAC address is correct).
- Your dhcp server is on the same physical network segment as the PA−RISC box.
- The state of the network connection of the 2 boxes.
- Try to tcpdump while you are booting the PA−RISC box over the lan.

Appendix B. Older PALO dumps

B.1. A500 Session dump using PALO 0.97

```
Main Menu: Enter command or menu > bo scsi.5.0 ipl
Interact with IPL (Y, N, or Cancel)?> y


Booting...
Boot IO Dependent Code (IODC) revision 1


HARD Booted.
palo ipl 0.97 root@c3k Tue Nov 27 14:51:48 MST 2001
Information: Boot device can't seek past 2Gb (ignore next error).
byteio_read: seekread() returned -1 expected 2048

Partition Start(MB) End(MB) Id Type
1               1      15   f0 Palo
2              16     503   82 swap
3             504    2887   83 ext2

PALO(F0) partition contains:
        0/vmlinux64 3990942 bytes @ 0x44000

Information: No console specified on kernel command line. This is normal.
PALO will choose the console currently  used by firmware (serial).
Current command line:
3/boot/vmlinux root=/dev/sda3 HOME=/ console=ttyS0 TERM=vt102
 0: 3/boot/vmlinux
 1: root=/dev/sda3
 2: HOME=/
 3: console=ttyS0
 4: TERM=vt102

Edit which field?
(or 'b' to boot with this command line)? 0
3/boot/vmlinux-2.4.17-pa3 initrd=0/root.bin
Current command line:
3/boot/vmlinux-2.4.17-pa3 initrd=root.bin root=/dev/sda3 HOME=/
                                                                                console=ttyS0 TERM=
 0: 3/boot/vmlinux-2.4.17-pa3
 1: initrd=0/root.bin
 2: root=/dev/sda3
 3: HOME=/
 4: console=ttyS0
 5: TERM=vt102


Edit which field?
(or 'b' to boot with this command line)? 1

Current command line:
3/boot/vmlinux-2.4.17-pa3  root=/dev/sda3 HOME=/ console=ttyS0 TERM=vt102
 0: 3/boot/vmlinux-2.4.17-pa3
 1: root=/dev/sda3
 2: HOME=/
 3: console=ttyS0
 4: TERM=vt102
```

```
Edit which field?
(or 'b' to boot with this command line)? b
```

PALO was first setup to boot the kernel file `vmlinux` located on the third partition of the SCSI device ID 5 LUN 0. (We know this since we have asked **BOOT_ADMIN** to boot on this device). But we wanted another kernel this time. We have pressed the **Enter** key (to validate the default choice '`0`') and modified the text to match our needs, here `vmlinux-2.4.17-pa3`. We have also added an `initrd=0/root.bin` argument to the command line. We have validated our changes by hitting the **Enter** key. Finally we have decided that we didn't want this additional argument, so we have selected it and erased it. At the end it asked again which field we wanted to edit, we just typed '`b`' instead of any number and hit **Enter** to boot our new kernel.

Appendix C. HOWTO contributors

The following people contributed or reviewed this HOWTO in one way or another.

For Deb's version:

- David Alexander deVries <adevries@thepuffingroup.com>
- Philip Imperial Schwan <pschwan@thepuffingroup.com>

For Thomas' versions:

- Michael Damaschke <sps01@uni-koeln.de> Thanks for your example about **PALO**
- Helge Deller <deller@gmx.de>
- Jeremy Drake <jeremyd@apptechsys.com> Thanks for your Windows" boot server howto
- Grant Grundler <grundler@parisc-linux.org>
- Richard Hirst <rhirst@parisc-linux.org>

For Thibaut's versions:

- Matthieu Delahaye <delahaym@esiee.fr>
- Helge Deller <deller@gmx.de>
- Grant Grundler <grundler@parisc-linux.org>
- Richard Hirst <rhirst@parisc-linux.org>
- Kyle McMartin <kyle@mcmartin.ca>
- Clement Moyroud <moyroudc@esiee.fr>
- Carlos O'Donnel <carlos@systemhalted.org>
- Matthew Wilcox <matthew@wil.cx>

Glossary

This is a brief glossary of the PA−RISC specific terminology. You can find a more detailed one at http://www.parisc−linux.org/glossary/.

Boot Console Handler (BCH)
: This is the early boot console available during boot up on most PA−RISC machines, provided by the *Processor−Dependent Code (PDC)*.

  See Also: BOOT_ADMIN.

BOOT_ADMIN
: This a command line utility stored in the boot ROM of the PA box, which is used to configure the computer during early boot sequence. It is a part of the PA−RISC machine's firmware.

  See Also: Boot Console Handler (BCH).

Guardian Service Processor (GSP)
: The GSP is a console subsystem present on certain PA−RISC systems, which provides several features such as remote console, UPS management, system low level control.

High Priority Machine Check (HPMC)
: Fatal system error. *Processor−Dependent Code (PDC)* saves machine state in the *Processor Internal Memory (PIM)*.

Hewlett Packard Precision Architecture (HP−PA)
: 'HP−PA' (sometimes '*hppa*') is the short way to refer to *HP PA−RISC* architecture. It's real meaning is: '*Hewlett Packard Precision Architecture*'. It is used for instance by Debian and OpenBSD to point out their ports.

Initial Program Loader (IPL)
: It is the HP standardized system bootstrap responsible for loading the operating system's kernel on PA−RISC systems. It can be launched from the **BOOT_ADMIN**.

  See Also: BOOT_ADMIN.

Initial System Loader (ISL)
: ISL is the executable that brings you into **BOOT_ADMIN**.

  See Also: Initial Program Loader (IPL).

Logical Interchange Format (LIF)
: This is a HP mass−storage format used for exchanging files among HP computer systems. It basically contains a header (identifying it as a LIF volume) and a directory of fixed size containing the files. The size of the directory is fixed when the volume is created, which explains many things about the way **PALO** works!

lifimage
: It is the name contraction of *LIF image*, which is indeed a file which format respects the `LIF` standard. It can be seen as the equivalent of an `ISO` file, having the `LIF` format instead of `ISO9660`.

  See Also: Logical Interchange Format (LIF).

Low Priority Machine Check (LPMC)
: Generally a recoverable system error.

  See Also: High Priority Machine Check (HPMC).

Management Processor (MP)
: The MP is a newer evolution of the GSP.

See Also: Guardian Service Processor (GSP).

PA−RISC

PA stands for Precision Architecture. It is the name of two generations of HP processors. They are classified as PA−RISC 1.X and PA−RISC 2.0. But a system based on a PA−RISC processor is commonly called a HP−PA box.

See Also: Hewlett Packard Precision Architecture (HP−PA).

PA LOader (PALO)

PALO is the *PA/Linux kernel LOader*. It was designed by Paul Bame as a *LILO* equivalent for the PA−RISC architecture.

Processor−Dependent Code (PDC)

It is the firmware that handles all processor−dependent functionalities, including hardware initialization and self−test procedures. Once it has done this, it passes control to the ISL.

See Also: Initial System Loader (ISL).

Processor Internal Memory (PIM)

Machine state is saved here after *High Priority Machine Check (HPMC)*, *Low Priority Machine Check (LPMC)*, and *Transfer Of Control (TOC)*. See `PDC_PIM` in "PDC Procedures" chapter of PA−RISC I/O ACD (available from http://www.parisc−linux.org/documentation/.

See Also: Initial System Loader (ISL).

netinst

This is not a PA−RISC specific term, though it needs explanations. *Network Install*, also known as *netinst*, are small ISOs containing everything you need to boot a computer and install it from network. They are based on the Debian distribution.

SuckyIO

`(added by special request)` National Semiconductor PC87560UBD, aka *SuperIO*. Provides IDE, USB 1.1, Floppy Disk Controller, parallel port, 2 serial ports, UIR (Infrared), etc. But since National denies the existence of this chip and HP was the only client for this buggy PoS, the name *SuckyIO* has stuck.

SuperIO

Official term for *SuckyIO*.

See Also: SuckyIO.

Standard Text Interface (STI)

It defines a standardized way to access the graphic subsystem on HP−PA.

Standard Text Interface Console layer (STIcon)

It is the basic text−mode console that can run on top of any STI−capable device.

See Also: Standard Text Interface (STI).

Standard Text Interface FrameBuffer layer (STIfb)

It is a superset of STI, providing standard API to access framebuffer devices on HP−PA.

See Also: Standard Text Interface (STI).

Transfer Of Control (TOC)

This acronym can usually be found on some PA−RISC boxes, right near a tiny switch that is not often used (hopefully). On HP/UX it would make a crash dump and reset the box. It can also be requested from the *Guardian Service Processor (GSP)*. On Linux, it will save the registers and reset, saved registers will be accessible in the *Processor−Dependent Code (PDC)*.

Bibliography

These documents might prove helpful to understand the present one, or to open new horizons:

[Raymond 2000] E. S. Raymond, 2000, *Installation−HOWTO* .

How to obtain and install Linux software. It is the first document which a new Linux user should read to get started.

[Maor 1999] O. Maor, 1999, *NFS−Root−Client Mini−HOWTO* .

How to create client root directories on a server that is using NFS Root mounted clients.

[Kostyrka 1997] A. Kostyrka, 1997, *NFS−Root Mini−HOWTO* .

How to setup a 'disk−less' Linux workstation, which mounts its root filesystem *via* NFS.

[Harris et al. 1997] T. Harris and K. Koehntopp, 1997, *Linux Partition HOWTO* .

Teaches you how to plan and layout disk space for your Linux system.

[Dev 1998] A. Dev, 1998, *CVS−RCS−HOWTO* .

This document is a "practical guide" to very quickly setup CVS/RCS source code control system.

[Noronha Silva 2001] G. Noronha Silva, 2001, *APT HOWTO* .

Will help you understand how the Debian package management utility, APT, works.

[O'Donell 2002] C. O'Donell, 2002, *The PARISC−Linux Cross Compiler HOWTO* .

This is a semi−detailed guide for building a cross compiler toolchain targeting PA−RISC systems.

[Cornec 1997] B. Cornec, 1997, *HP HOWTO* .

Describes the use of products available in the Hewlett−Packard (HP) catalog with Linux and some free software.

[Perens et al. 1996] B. Perens, S. Rudolph, I. Grobman, J. Treacy, and A. Di Carlo, 1996, *Debian GNU/Linux 3.0 Installation Documentation Index* .

Will help you to install and configure your Debian GNU/Linux system.

[Debian Installer Team 2005] Debian Installer Team, 2004, 2005, *Debian GNU/Linux 3.1 Installation Documentation* .

Will help you to install and configure your Debian Sarge GNU/Linux system.

[Vermeulen et al. 2006] S. Vermeulen, R. Marples, D. Robbins, C. Houser, and J. Alexandratos, 2006, *Gentoo HPPA Handbook* .

A must read for the wannabe Gentoo user.

[Brouwer 1993] A. Brouwer, 1993, *The Linux keyboard and console HOWTO* .

This note contains some information about the Linux keyboard and console, and the use of non−ASCII characters.

[HP Booting] *HP documentation: Booting Systems* .

This is an HP document detailing booting procedures.

Notes

[1] Before the first release of Debian on hppa, there has been unofficial releases of Woody snapshots, entitled "PA/Linux releases", and numbered *0.x*. Immediately after the release of Woody, the PA/Linux port switched to the normal Debian numbering scheme. In other words, Debian 3.0 is consecutive to PA/Linux 0.9.3.

[2] For the knowledge addict: **PALO** can actually access and read ext2/ext3 filesystem, and therefore follow symlinks, whereas LILO bootloader will only know the physical disk address to access the kernel. See this for further details.

[3] Example: /boot is mounted from a separate partition, which number is, say, 4 according to **fdisk**. From a Linux point of view, the absolute path of the file is `/boot/vmlinux`, but from a *partition* point of view, it is `/vmlinux`. Therefore, the commandline will start with "`4/vmlinux`". We hope that's clear enough!

[4] The *`format-as`* switch is a bit misleading. When used with *`init-partitioned`* it is meant to tell which filesystem to format the new partition, but with *`update-partitioned`*, it is meant to tell **PALO** which filesystem is used on the already formatted partition.

[5] To save your changes, you will have to run **/sbin/palo** when your system will be up and running, and it will write on the disk all the parameters contained in the configuration file, (`/etc/palo.conf`), which you will have properly modified if needed.

[6] For those who wonder what the final 0 means, it's the device LUN. Since most SCSI devices have only one LUN (especially disks), you can safely use 0 as in this example.