PA−RISC/Linux Boot HOWTO

Thomas Marteau

TuxFamily

<marteaut@tuxfamily.org>

Deb RichardsonOriginal author
The Puffin Group

<deb@thepuffingroup.com>

Thibaut VareneContributor of v1.0
ESIEE

<varenet@esiee.fr>

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, we must say that this document keeps some parts from Deb's original one and reveals some of her hidden work.

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

Table of Contents

Table of Contents

Chapter 1. Introduction

1.1. Overview

You just received this HP box you bought online or maybe you got it from your company surplus. 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. Take a look at this Howto and you will see that Linux could be the answer to this question. Anyway, we hope so.

In addition to port Linux, the development team is working on porting the Debian project to PA−RISC. In fact, around 95 % of packages are ported and up−to−date in the repository. The port can be consider as useful now. In Debian3.0r0 called *woody*, the flavour "hppa" has been released for the first time. Some Debian developers non involved in the port but yet curious reported that the port was one of the easiest to install since you feel like installing an i386 version.

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

In a few words, this HOWTO is aimed to anyone looking for some help and information about using Linux on a HP system based on PA−RISC architecture. No special knowledge is necessary but bases about how Debian packages work can be helpful.

If you care about just installing and you do not ask yourself about the way it works, the best advice is to try a Debian release including the PA−RISC port. The *Woody* version is now available for hppa architecture. If you experience any trouble, try the development *netinst* ISO images from the The PA/Linux ESIEE Team.

After listing the supported hardware, this HOWTO explains some commands of the basic console available at boot time. Then, the features of the PA/Linux kernel loader introduce another chapter showing many ways to get your system up and running. At the end, the text goes deep in the kernel compilation and configuration.

1.2. Copyright and Licensing

Copyright 2002 Thibaut Varene.
Copyright 2001−2002 Thomas Marteau.
Copyright 1999 The Puffin Group and Deb Richardson.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front−Cover Texts being 'Copyright and Licensing', and with the Back−Cover Texts being 'HOWTO Contributors'. A copy of the license can be found at http://www.gnu.org/copyleft/fdl.html.

Chapter 2. Supported Hardware

With the release of the 0.9.3 version of PA/Linux, a major improvement was made in term of quantity and quality of hardware support. Since 0.9.3 released, 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`>. The following PA−RISC machines can be booted *almost* like any other box of a different architecture. We must add that this list can change at any time. The best way to get an up to date version is to look at http://pateam.esiee.fr/list.html. There you will know if your hardware is supported and up to what level. For example, if you can run PA/Linux using the *serial console* or the graphic card.

- All 712 models.
- All 715 models including Strider series.
- All 705, 710, 720, 730, 750 models should be running with the latest ISO. It contains some modifications specially for hard disk devices.
- Some 725, 735 (no FWD SCSI), 755 models are running with the latest kernels. 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.
- BXXX models like B132, B160 and B180. These boxes can be used in the framebuffer mode *via* 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 are not yet supported.
- CXXX models like C110, C160, C180L, C240, C360.
- CXXXX models. Indeed, BXXXX and CXXXX are based on Astro/Elroy (aka SBA/LBA) chipsets with varying CPU speeds, number of memory/PCI slots.
- 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.
- E class : Christoph Plattner is working on his E55. E35 and E55 are known to work diskless. The SCSI support is expected soon.
- 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 the ISO images made by the ESIEE team tagged with *″−PDC−″*.
- L class : L1000 and L2000, with *serial console*.
- R class is basically the same as D class.

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

- L3000 – smaller brother of N class – currently only works Uni Processor (UP).
- N class : N4000−55 seems to be supported in UP mode.
- 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.

The following hardware might never work :

- T600.

Chapter 3. Preparing to boot

Like any other system, machines based on PA−RISC processors have to go through several steps in order to have PA/Linux up and running. The next section introduces you to the early boot management of your PA−RISC computer. 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**.

3.1. 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 any hacking on 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*. 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 !

3.1.1. 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 3.2) :
   ```
   Searching for Potential Boot Devices.
   To terminate search, press and hold the ESCAPE key.
   ```
   When this message appears, press and hold the **Esc** key until an option menu appears. This can take a while, be patient.
3. By default, you should enter the **BOOT_ADMIN** console. Though on some 715s and 725s, an option menu looking like this may appear :
   ```
   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:
   ```
   Select 'a) 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 !

3.1.2. BOOT_ADMIN commands

BOOT_ADMIN is an early boot subsystem where you can execute some precise commands. You should find here everything you need to know about them.

All HP−PA systems have **BOOT_ADMIN**. 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 these sections.

Some commands may appear in several different menus, this is normal.

3.1.2.1. The `main` commands

These commands are the basic ones.

- **boot** must 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.
- **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 controler 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 all the 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). You can 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 consulting.
- On almost every systems, you have a **reset** instruction. It makes the box reboot with the latest parameters you have set.

3.1.2.2. 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 mode) sets your display configuration by typing **monitor `<path> <type>`** which indicates your console path and type. If you do not know your monitor type, you can list those available *via* **monitor list**.
- **fastboot** displays or sets the boot tests execution.

3.1.2.3. 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 fimrware revision. You can check if your firmware is up−to−date with this link.
- **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.

3.1.2.4. The `service` commands

It is a PA−RISC guru menu. You will find nothing really interesting for an end−user. 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.

3.2. Consoles

In order to boot your PA−RISC system with the PA/Linux kernel, you must first set up a *console* on it. A *console* is basically the device where the kernel (and the firmware) will display its output, and where your can send your input to control the system at an early boot stage. You can use either *graphic console*, which requires to have a monitor and a keyboard attached to the system, or *serial console*, which allows serial communication between the system and another Linux machine, or any VT system. You should know that the consoles for the firmware and for the kernel can be different. For example, you can have to interact with the **BOOT_ADMIN** mode with a monitor and once PA/Linux is up, you have ttys running on serial ports only.

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 is the actual console of your box, it's quite simple : this is where it will send its first output when switched on (serial line or monitor output, if any).

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*.

3.2.1. 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 differenciate 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 features the *Standard Text Interface* 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 differenciate 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.

3.2.2. Using *serial console*

The *serial console* is a good way to get all console messages handy, including the **BOOT_ADMIN** ones. It is very useful for bug reports, as its output can be easily dumped. Moreover, most of the servers can only be managed with serial console. Anyway, the only cases where you will HAVE TO use serial console is either if you don't have a monitor for your PA−RISC machine, or if your machine doesn't support graphics. It is also possible that the kernel can NOT handle some specific graphic hardware present in your model.

Here follows the procedure to setup *serial console* support.

3.2.2.1. Serial Cable

To connect your PA−RISC machine to your 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 chose 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).

3.2.2.2. Configuring minicom on Linux

In order to communicate with your PA−RISC machine, 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 for all PA−RISC systems. If you are running **minicom** on a PC, you will probably need to change the baud rate.

3.2.3. Switching consoles

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

3.2.3.1. 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.

For some models, you can find slight differences 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.

3.2.3.2. 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 'Console path = rs232_a.9600.8.none', indicating that the system is now set up to boot in *serial console* mode, on RS232 port 'A'. If your machine has only one, this is OK, if not, take care to use the right one. By default, **reset** will reboot your system with the new parameters.

3.2.3.2.1. 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 console, 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 describes how to change the console from an *HP/UX ISL* prompt. You can find a small HP/UX *lifimage* here : http://pateam.esiee.fr/archive/uxbootlf. (See further Section 5.3 to learn how to *netboot* a *lifimage*). In fact, serial console on 712 is only useful if you want to boot the box without any keyboard attached to it, which is otherwise not possible.

Here is the procedure :

1. Turn the box on and when in **BOOT_ADMIN**, boot to HP/UX ISL. For example :
   ```
   BOOT_ADMIN> boot lan isl
   ```
2. Once you get the 'ISL>' prompt, type the following :

   ♦ For switching to serial : **conspath 2/0/4.0x283**
   ♦ For 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.

3.2.3.3. Changing to graphic console mode

It is the opposite operation compared to the previous one. By checking your console path, you should see 'Console path = rs232_a.9600.8.none'. Now, you must set the graphic mode by issuing the following command at 'BOOT_ADMIN>' prompt :

path console graphic_1

You should get the display available on the monitor after a **reset**. If the screen 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. Perhaps you will need to do this operation several times. This is also true when you change your monitor.

Chapter 4. Using PALO, the kernel loader for PA−RISC

4.1. What is PALO?

PALO is a set of two programs, a boot loader, which is loaded by the PA−RISC *firmware* in 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*. 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, though it's worth mentionning that **PALO** doesn't usually need to be called every time you build and install a new kernel, as LILO does. **PALO** is strongly related to PA/Linux development. Thus, several versions has been released. The last changes are explained by the author of **PALO**, *Paul Bame*, in this mail.

4.2. 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 4.4). 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 by itself. It needs **PALO** to be turned into a bootable lifimage for CD or network boot, or to be launched at boot time from a prepared hard disk drive. Have a look at *Glossary* about these words. Quoting *Richard Hirst*, a PA/Linux hacker : "*People often try to put a lifimage in /boot, or a vmlinux on the network*". Which is obviously wrong.

4.3. PALO management tool usage

Here we will show you 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 6.1.

4.3.1. Making a `lifimage` with `RAMDISK`

First things first : when should you go this way ?

At an earlier step 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 HP box *via* the **boot lan** instruction without any further involvement (see Section 5.3). The main advantage of a RAMDISK is that it unpacks its own file system in

RAM, and therefore is completely independant from 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 some customized files. Now, let's see how to obtain a lifimage with RAMDISK.

> (i) If you don't want to mess with building your own RAMDISK, you can use root.bin that can be found on Debian *Boot−Floppies*.

We assume you got the latest source of the PA/Linux kernel. 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 be found at http://www.parisc−linux.org/. Go through the **make menuconfig** step. Then, run **make palo** and if you have the **PALO** installed, you should get this message 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 configuring the palo.conf, you can go back to your **make palo**. The result, a lifimage file, is waiting for you in the linux/ directory.

4.3.2. 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−RISC box will boot *via* the **boot lan** instruction its new kernel.

Getting the NFSROOT support is easier than RAMDISK. You have to edit the palo.conf to specify the boot server IP address instead of the string *HOSTNAME*. In fact, if your server has 10.10.10.2 as its IP adress, 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 configured the `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.

4.3.3. Making a bootable partition

This part is where **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 own hard disk drive. This section is going to explain how to make it work. When installing the **PALO** package, Paul Bame, the author and maintainer, put a copy of the default `/etc/palo.conf` in `/usr/share/doc/palo/palo.conf`. If you want to understand how **PALO** works, you just have to read this file !

To setup a bootable hard disk, you have to partition properly your hard drive (if any, and 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 (*via* CD or network, see Chapter 5), or if you intend to prepare your hard disk using another computer than the target (which can be useful to unpack and setup a downloaded file system for example). The point of this HOWTO is not to teach you how to use **fdisk** or other, so here are the few things you HAVE TO know :

- A partition 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.
- It does not need to be huge. This is were **PALO** will save its configuration, recovery kernel(s) – about 5MB each – and optional ramdisk. 16−32MB seems far sufficient.

Beware ! Your `vmlinux` has also to be located within the first 2GB of the hard disk. We strongly recommend to create a separated `/boot` partition at the front of the disk if your '/' is bigger than that, because if ever your vmlinux goes above the first 2GB of the disk (like when filling up '/' with data), the box won't boot anymore.

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 the path to the kernel that you want to boot within a failsafe session, it will be stored in the 'f0' partition.
- *bootloader* is the path to the iplboot boot loader utility which is created by **PALO** when you issue a **make iplboot**.
- *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 the first octets of the first partition of this device, which partition−type must be 'f0', as shown above.
- *commandline* : the first digit is the number of your ext2 partition where the kernel file is located, as reported by **fdisk**. Logically, the next string is the absolute path to the kernel. The following space separated parameters 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.

  You can also add console=, to force the designation of the output console. You should remind that console=ttyS0 is for a serial console and console=tty0 is for a STI−console. Recently, support for the PDC console (if enabled in the kernel, see Section 6.2.3) has been added, using console=ttyB0. Indeed, the latest versions of **PALO** autodetect the right console path (except for PDC), and can figure out whether a 32bit or 64bit kernel should be used. If not, please mail to the mailing list.

In fact, this third usage of **PALO** is the most common but the default /etc/palo.conf makes it easy to configure. Just choose your root partition. It should the partition containing your root directory. According our **fdisk** example, we want /dev/sda3. Thus, the configuration file should look like that :

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

4.4. How to use PALO at early boot stage ?

4.4.1. 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, we have the solution !

First, you must learn how to interact with **PALO** during the startup sequence. You have to enter **BOOT_ADMIN**, as explained in Section 3.1.1. 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 by a "**y**". You will fall back 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 hiting **ENTER** again. The system will redisplay the new list. This modification is not permanent ! 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 default file, (`/etc/palo.conf`), which you will have properly modified if needed. If you want to add a supplementary parameter, select any one and write yours on the editing line, beginning with a space :

```
Edit which field?
(or 'b' to boot with this command line)? 0
3/boot/vmlinux initrd=root.bin
```

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. This section is mostly inspired from Paul Bame's file as well as the page about **PALO** you can find at http://pateam.esiee.fr/palo.html.

4.4.2. A complete example

This example has been suggested by *Michael Damaschke*. 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 and monitor on, a message on the screen 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. 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, the monitor can be too slow to get 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 blink, this is the right time to press and hold the **Esc** key. If you still have troubles, please refer to the *Consoles* section.

There are few differences about the way to get access to **BOOT_ADMIN** (see Section 3.1.1). If you have an old box, you will get 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 before, you must press the **Esc** key.

You might then get a menu where you must press the **a**–key followed by **ENTER**–key. 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 the **ENTER** key.

After that, 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 5).

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. *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)?>
```

Now, you can manually configure the **PALO** booting parameters. You can see a new menu, 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 :

```
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=vt102
 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 modify 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 have just put '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 do !

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

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

Chapter 5. Available boot solutions

5.1. 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 PA/Linux ISOs as well as recent *Net Install* ISO (see *Glossary*) at The PA/Linux ESIEE Team download page, or at PA−RISC/Linux official website.

   i. start the box and enter the **BOOT_ADMIN** mode. (Section 3.1.1)
   ii. Dispose 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 devices with IPL. You can also specify a **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
```

5.2. 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 4.3.3 to learn how to prepare your hard drive.

   i. start the box and enter the **BOOT_ADMIN** mode. (Section 3.1.1)
   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 devices with IPL. You can also specify a **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 4.3.3). Real life example :
```
boot scsi.6
```

5.3. Booting from network

5.3.1. Preparing to boot from network

This is a very old way to operate but it used to be the only one available for a long time. Usually you won't need to boot from network, except in some very specific cases (*e.g.* unsupported I/O devices). That's why it is detailed here.

You will need a *lifimage* to perform a network boot. See Section 4.2 to learn how to create one. You can also download one at http://pateam.esiee.fr/cd–images/lifimages/

5.3.2. 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 6 for details.

5.3.3. Using rboot

5.3.3.1. Obtaining rboot

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

Old machines, including the Scorpio 715s, require **rbootd**. You can obtain the **rboot** daemon :

- For all distributions but Debian, by getting our archive at http://pateam.esiee.fr/download.html#deb
- For Debian, from http://www.debian.org/Packages/stable/net/rbootd.html, or even more simple :
  ```
  bash# apt-get install rbootd
  ```

5.3.3.2. Configuring rbootd

For instance, to boot a PA–RISC 715 system, you need a Linux system 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, do as follows 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 3.1.2.3).

   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/`. If you use our archive for other distributions, this directory is `/export/hp/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 recompile **rbootd** to use a different directory.

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

5.3.4. Using dhcp/tftp

We will see here how to use a `DHCP` server as a `BOOTP` one.

5.3.4.1. Obtaining dhcp/tftp

Debian users will just have to install the packages *via* these commands as **root** :

```
bash# apt-get install dhcp
bash# apt-get install tftpd
```

If you need rpm packages (for the ISC dhcp server), the best way is to go to http://rpmfind.net/. It seems that Red Hat users need to create the user "nobody" belonging to the group "nogroup". The files present in your `/tftpboot/` directory (see below) should have these user/group privileges.

The **dhcp** package can do far more than a simple **bootp** daemon. Nevertheless, it is also known to be far easier to configure. If you really want to try **bootp**, skip this and go to Section 5.3.5.

5.3.4.2. 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`.

This section is dedicated to Debian users. For others distributions, it should be similar though there can be some differences like default directories. Since Debian maintains a **dhcp** package, we will focus on it only.

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 \
                                                /usr/sbin/in.tftpd /tftpboot
```

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

When this is done, restart **inetd** with : **/etc/init.d/inetd restart**. You 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-server 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).
- ♦ *[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 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";
}
```

5.3.5. Using bootp/tftp

5.3.5.1. Obtaining bootp/tftp

For Debian users, you just have to install the packages *via* these commands as **root** :

```
bash# apt-get install bootp tfptd
```

If you need rpm packages, the best way is to go to http://rpmfind.net/. It seems that Red Hat users need to create the user "nobody" belonging to the group "nogroup". The files present in your /tftpboot/ directory (see below) should have these user/group privileges.

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

5.3.5.2. Configuring bootp/tftp

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

☞ This section is dedicated to Debian users. For others distributions, it should be similar though there can be some differences like default directories. Since Debian maintains a **bootp** package, we will focus on it only.

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/sbin/in.tftpd /tftpboot
   bootps          dgram   udp     wait    root    /usr/sbin/bootpd    \
                                           bootpd -i -t 120
   ```
   Here, /tftpboot/ is being used as tftpd server's root. You can choose another directory if you want. According to **man tftpd**, this is the usual default directory.

   When this is done, restart **inetd** with : **/etc/init.d/inetd restart**. You can also issue a **killall –HUP inetd**.

2. According to **man 5 bootptab**, edit the /etc/bootptab file to contain :
   ```
   [hostname]:hd=/tftpboot:\
           :rp=[root path]:\
           :ht=ethernet:\
           :ha=[mac address]:\
           :ip=[ip address]:\
           :bf=[boot filename]:\
           :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 3.1.2.3).
   - *[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:
   ```

5.3.6. Booting your PA/Linux system from network with serial console

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 5.2.

Here we are. These are just some tips to get the boot for those who tried the network way. We assume that you've done everything outlined above, your development PC is hooked up to your PA−RISC machine *via* a serial line, 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, as well as you, the following procedure will introduce you to the joy of network booting your PA box into Linux.

1. Make sure your development box is properly connected to your PA−RISC machine with a serial cable. Sounds obvious, but check anyway.
2. Start **minicom** on your development box.
3. Fire up your PA−RISC system.
4. Watch your **minicom** console. 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.
   ```
5. If needed, select 'a) Enter Boot Administration mode' from the menu. This brings up the 'BOOT_ADMIN>' prompt.
6. Type the following at the prompt : **boot lan**.
7. 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 6. Building and installing your own PA−RISC/Linux 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 lies on the official CVS site. First, we will discuss about **GCC** compiler. Then, the configuration of your 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.

6.1. GCC compiler

You can compile your kernel with your own PA−RISC box. But on old systems, you may prefer to use another – faster – computer to compile your kernel. We will see the two alternatives. Whichever you choose, you need at least **gcc−3.0**.

☞ By the time this howto is released, only gcc−3.0.X was able to build working kernels. There is a bug in more recent versions that makes the box crash when network activity occurs.

6.1.1. native build

Since Debian is the only distribution supporting 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.

6.1.1.1. apt−get and friends

apt−get is a simple command line utility that manages Debian package system. Gustavo Noronha Silva wrote the APT HOWTO that you should read for sharper details. Actually, we just want to build a kernel, so we will tell you the bare minimum needed to do that. First, setup your `sources.list` in order to fetch the needed archives from the Internet. Here is a sample configuration for `/etc/apt/sources.list`, using a German Debian mirror :

```
# Binary packages
deb http://ftp.de.debian.org/debian unstable main contrib non-free
# non-US packages
deb http://ftp.de.debian.org/debian-non-US unstable/non-US main non-free contrib
# source packages
deb-src http://ftp.de.debian.org/debian unstable main contrib non-free
# non-US source packages
deb-src http://ftp.de.debian.org/debian-non-US unstable/non-US main contrib non-free
```

6.1.1.2. update your gcc

If you are using your own PA−RISC box, you only need the good old **GCC** compiler. We recommend to keep it up−to−date with the latest version uploaded by the developers.

```
bash# apt-get update
```

```
bash# apt-get upgrade
```

If you do not want to upgrade all your system, according to the package description of kernel–source, you need to get those packages updated :

- binutils
- fileutils
- gcc
- libc–dev
- make

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

6.1.2. cross compiled build

In this kernel build method, everything depends on the architecture of your building machine. For x86 computers, you can download a ready–to–use cross compiler archive on the PA/Linux FTP server. You can also find some "exotic" (like for MacOS X) cross–compilers archives on the PA/Linux ESIEE Team website. For other architectures or if you want to compile your own toolchain, please refer to Carlos O'Donell's HOWTO.

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 at ftp://ftp.parisc–linux.org/unofficial–debs/. See PA–RISC Linux Website for details.

6.2. Kernel configuration

If you want to take advantage of the latest kernel improvements, we suggest you to retrieve it from the official PA–RISC/Linux CVS. You can of course either download it from http://www.kernel.org/, or use the Debian package, but 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. 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 correspond to your hardware :

Remenber that **make oldconfig** is a good base to start with, since it works for almost all machines.

- *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, PDC see Section 6.2.3)
- *HIL Support* – useful if you have a HIL controller. See below Section 6.2.1.
- *Console drivers* – is directly related to your console mode (STI console or STI framebuffer)
- *Sound* – enables/disables the Harmony driver

As you 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 functionnalities 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 6.2.4
- *File Systems/Network File Systems* – is where to set EXT3 or NFS support
- *USB support* – If you have enabled *SuperIO* and want USB, look there Section 6.2.2

At the time this HOWTO was written, there was no floppy drive support; and what's more, it was not expected to have one any day.

When you're done with it, save your kernel configuration. Everything is written in the `.config` file. You should back it up because a **make distclean** will remove it. At this very 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.

6.2.1. `HIL` Support

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 what it says :

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 :

```
bash# 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/`.

6.2.2. USB **Support**

USB support on HP−PA is still experimental, therefore it is only proposed 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 spoiled. They do not
behave as expected at all.

(i) 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 :

```
bash# 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, as it is also using the *Linux Input Driver*.

6.2.3. `PDC Console` Support

PDC Console has been improved by *Richard Hirst* in pa37 kernel, though it is still a very *experimental* feature. It is expected to provide adequate PDC 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_PDC_CONSOLE=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 :

```
bash# cd /dev; ./MAKEDEV ttyB0
```

☞ It needs a recent MAKEDEV package to be created this way. By the time this HOWTO was written, such a package could only be found on *netinst* ISO provided at
http://pateam.esiee.fr/download.html.

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

6.2.4. `IDE` Devices Support

There is nothing really special about IDE support. You have to check that your *IDE Chipset* 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 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 :

```
bash# cd dev; ./MAKEDEV hda hdb hdc hdd hde
```

☞ It needs a recent MAKEDEV package to be created this way. By the time this HOWTO was written, such a package could only be found on *netinst* ISO provided at http://pateam.esiee.fr/download.html.

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

6.3. 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 do a :

```
bash# cp vmlinux /boot/vmlinux-[kernelversion]
bash# cp System.map /boot/System.map-[kernelversion]
bash# 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 a **cd /boot**, make sure that vmlinux is a symbolic link to another file, as in the following example :

```
bash# 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 4 for more information). Now do the following :

```
bash# rm -f vmlinux
bash# ln -s vmlinux-[kernelversion] vmlinux
bash# 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 4.3, 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 `.config` in `/boot` as mentionned before. You can use the network (like **ftp**) or a CD to do so, or even direct copy to the hard disk drive.

Chapter 7. 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' version :

- Michael Damaschke <sps01@uni-koeln.de> Thanks for your example about **PALO**
- Helge Deller <deller@gmx.de>
- Grant Grundler <grundler@parisc-linux.org>
- Richard Hirst <rhirst@parisc-linux.org>

For Thibaut's version :

- Matthieu Delahaye <delahaym@esiee.fr>
- Helge Deller <deller@gmx.de>
- Grant Grundler <grundler@parisc-linux.org>
- Richard Hirst <rhirst@parisc-linux.org>
- Clement Moyroud <moyroudc@esiee.fr>
- 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_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.

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)*.

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 thingss about the
> way **PALO** works !

lifimage
> It is the name contraction of 'LIF image', which is indeed a file which respond to '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.

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:* 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 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 for HPMC, LPMC, and TOC's. See PDC_PIM in "PDC Procedures" chapter of PA I/O ACD.

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.

Transfer Of Control (TOC)

Under HP/UX it would make a crash dump and reset the box. It can also be called from the GSP. Under Linux, it will save the registers and reset, saved registers will be accessible through 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 quicly 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 targetting HP 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.

[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.