SRM Firmware Howto

Rich Payne, and David Huggins−Daines

v0.8, 09 November 2000

This document describes how to boot Linux/Alpha using the SRM console, which is the console firmware also used to boot Compaq Tru64 Unix (also known as Digital Unix and OSF/1) and OpenVMS.

Table of Contents

Table of Contents

1. About this manual

1.1. Who should read this manual

You should read this manual if you are installing Linux on a new Alpha system that can only boot from the SRM console, or if you are installing Linux on an older Alpha system that can use the SRM console and wish to use SRM to boot your Linux installation.

Because SRM is the only way to boot Linux on modern Alpha systems, and because it provides the proper operating environment for Unix and Unix−like operating systems (such as Linux), it is the recommended way of booting Linux on Alpha when available.

Sometimes, it is preferable to use the ARC, ARCSBIOS, or AlphaBIOS console, such as if you have a machine for which SRM is not available, if you wish to dual−boot with Windows NT without switching consoles, or if you have hardware that is not supported by SRM. On these machines, you will typically use MILO to boot Linux. For more information, refer to the MILO Howto, available from http://www.alphalinux.org/faq/milo.html.

1.2. Conventions

Throughout this manual, we will use the following conventions for commands to be entered by the user:

SRM console commands will be shown with the characteristic SRM '>>>' prompt, like this: [1]

```
>>> boot dva0 −fi linux.gz −fl "root=/dev/fd0 load_ramdisk=1"
```

Unix commands will be shown with the '#' command prompt if they are to be run as `root`, or '$' if they are to be run by a normal user, like this:

```
# swriteboot −f3 /dev/sda /boot/bootlx
```

Aboot commands will be shown with the 'aboot>' command prompt, like this:

```
aboot> b 6/boot/vmlinuz root=/dev/hda6
```

2. What is SRM?

SRM console is used by Alpha systems as Unix−style boot firmware. Tru64 Unix and OpenVMS depend on it and Linux can boot from it. You can recognize SRM console as a blue screen with a prompt that is presented to you on power−up.

2.1. Getting to SRM

Most Alpha systems have both the SRM and ARC/AlphaBIOS console in their firmware. On one of these machines, if your machine starts up with ARC/AlphaBIOS by default, you can switch to SRM through the "Console Selection" option in the Advanced CMOS Setup menu. To make the change permanent, you should set the `os_type` environment variable in SRM to "OpenVMS" or "Unix", like this:

```
>>> set os_type Unix
```

Either one will work to boot Linux. However, if you intend to dual−boot OpenVMS on this machine, you must set `os_type` to "OpenVMS". Conversely, to return to ARC/AlphaBIOS, you can set `os_type` to "NT".

Some older systems may not have both SRM and ARC in firmware as shipped. On these systems, you will have to upgrade your firmware. See http://ftp.digital.com/pub/DEC/Alpha/firmware for the latest firmware updates and instructions.

A few older systems (primarily evaluation boards such as the 164SX and 164LX) are "half−flash" systems, whose firmware can hold SRM or AlphaBIOS, but not both. If you have one of these machines, you will have to reflash your firmware with the SRM console using the AlphaBIOS firmware update utility. Again, see http://ftp.digital.com/pub/DEC/Alpha/firmware for firmware images and instructions. If you wish to return to AlphaBIOS on these machines, you may rerun the firmware update utility from a floppy in SRM using the `fwupdate` command. You can also start AlphaBIOS from a floppy using the `arc` command.

2.2. Using the SRM console

The SRM console works very much like a Unix or OpenVMS shell. It views your NVRAM and devices as a pseudo−filesystem. You can see this if you use the `ls` command. Also, it contains a fairly large set of diagnostic, setup, and debugging utilities, the details of which are beyond the scope of this document. As in the Unix shell, you can pipe the output of one command to the input of another, and there is a `more` command that works not unlike the Unix one. To get a full listing of available commands, run:

```
>>> help | more
```

As well, SRM has environment variables, a number of which are pre−defined and correspond to locations in NVRAM. You can view the entire list of environment variables and their values with the `show` command (there are quite a few of them, so you will probably want to pipe its output to `more`). You can also show variables matching a "glob" pattern – for example, `show boot*` will show all the variables starting in "boot".

Environment variables are categorized as either *read−only*, *warm non−volatile*, or *cold non−volatile*. The full listing of pre−defined variables is detailed in the Alpha Architecture Reference Manual. The most useful pre−defined environment variables for the purposes of booting Linux are `bootdef_dev`, `boot_file`, `boot_flags`, and `auto_action`, all of which are cold non−volatile.

To set environment variables, use the `set` command, like this:

```
>>> set bootdef_def dka0
```

If you set an undefined variable, it will be created for you, however it will not persist across reboots.

The `bootdef_dev` variable specifies the device (using VMS naming conventions – see Section 5.6.1 for an explanation of these) which will be booted from if no device is specified on the `boot` command line, or in an automatic boot. The `boot_file` variable contains the filename to be loaded by the secondary bootloader, while `boot_flags` contains any extra flags. `auto_action` specifies the action which the console should take on power−up. By default, it is set to `HALT`, meaning that the machine will start up in the SRM console. Once you have configured your bootloader and the boot−related variables, you can set it to `BOOT` in order to boot automatically on power−up.

Finally, two helpful console keystrokes you should know are Control−C, which, as in the shell, halts a command in progress (such as an automatic boot), and Control−P, which if issued from the aboot prompt (or other secondary bootloader) will halt the bootloader and return you to the SRM console.

2.3. How Does SRM Boot an OS?

All versions of SRM can boot from SCSI disks and the versions for recent platforms, such as the Noname or AlphaStations can boot from floppy disks as well. Network booting via `bootp` is supported. Note that older SRM versions (notably the one for the Jensen) cannot boot from floppy disks. Booting from IDE devices is supported on newer platforms (164SX, 164LX, 164UX, DS20, DS10, DP264, UP2000(+), UP1000, UP1100 etc..).

Booting Linux with SRM is a two step process: first, SRM loads and transfers control to the secondary bootstrap loader. Then the secondary bootstrap loader sets up the environment for Linux, reads the kernel image from a disk filesystem and finally transfers control to Linux.

Currently, there are two secondary bootstrap loaders for Linux: the *raw* loader that comes with the Linux kernel and `aboot` which is distributed separately. These two loaders are described in more detail below.

2.4. Loading The Secondary Bootstrap Loader

SRM knows nothing about filesystems or disk−partitions. It simply expects that the secondary bootstrap loader occupies a consecutive range of physical disk sector, starting from a given offset. The information on the size of the secondary bootstrap loader and the offset of its first disk sector is stored in the first 512 byte sector. Specifically, the long integer at offset 480 stores the *size* of the secondary bootstrap loader (in 512−byte blocks) and the long at offset 488 gives the *sector number* at which the secondary bootstrap loader starts. The first sector also stores a flag−word at offset 496 which is always 0 and a checksum at offset 504. The checksum is simply the sum of the first 63 long integers in the first sector.

If the checksum in the first sector is correct, SRM goes ahead and reads the *size* sectors starting from the sector given in the *sector number* field and places them in *virtual* memory at address `0x20000000`. If the reading completes successfully, SRM performs a jump to address `0x20000000`.

3. SRM Device Naming

3.1. The First Two Letter

The following is based on the example device dkb1.2.3.4.5 taken from a Digital Server 3300 (Whitebox version of an AS800).

Two letter port or class driver designator:

- DR: RAID set device
- DV: Floppy Drive
- EW: Ethernet port (TULIP, DEC 21040)
- EI: Ethernet port (Intel 82557 or 82559)
- PK: SCSI port (controller)
- DK: SCSI disk
- MK: SCSI tape
- PU: DSSI port
- DU: DSSI disk
- MU: DSSI tape
- JK: SCSI monitor (or robot)
- DQ: (E)IDE Device (disk or CD−ROM)

3.2. The Rest Of The Device Name

- b−> adapter ID (one letter adapter designator)
- 1−>Device number (SCSI unit numbers are forced to 100x Node ID)
- 2−>Bus Node ID
- 3−>Channel Number
- 4−>Channel Number (used for multi−channel devices)
- 5−>Logical Slot number
    - ♦ EISA: they correspond to the physical slot numbers (1−3)
    - ♦ PCI:
        - ◊ slot 5= SCSI controller on system backplane (DS3300)
        - ◊ slot 6= On board VGA (DS3300)
        - ◊ slot 7= PCI to EISA bridge chip (DS3300)
        - ◊ slots 11 − 14 = Correspond to Physical PCI option slots: PCI11, PCI12, PCI13 and PCI14 (64bit) (DS3300)
- 6−>Hose number: 0 PCI_0 (32bit PCI); 1 EISA (DS3300)

---

4. The Raw Loader

The sources for this loader can be found in directory `arch/alpha/boot` of the Linux kernel source distribution. It loads the Linux kernel by reading `START_SIZE` bytes starting at disk offset `BOOT_SIZE+512` (also in bytes). The constants `START_SIZE` and `BOOT_SIZE` are defined in `linux/include/asm-alpha/system.h`. `START_SIZE` must be at least as big as the kernel image (i.e., the size of the `.text`, `.data`, and `.bss` segments). Similarly, `BOOT_SIZE` must be at least as big as the image of the raw bootstrap loader. Both constants should be an integer multiple of the sector size, which is 512 bytes. The default values are currently 2MB for `START_SIZE` and 16KB for `BOOT_SIZE`. Note that if you want to boot from a 1.44MB floppy disk, you have to reduce `START_SIZE` to 1400KB and make sure that the kernel you want to boot is no bigger than that.

To build a raw loader, simply type `make rawboot` in the top directory of your linux source tree (typically `/usr/src/linux`). This should produce the following files in `arch/alpha/boot`:

`tools/lxboot`:

> The first sector on the disk. It contains the offset and size of the next file in the format described above.

`tools/bootlx`:

> The raw boot loader that will load the file below.

`vmlinux.nh`:

> The raw kernel image consisting of the `.text`, `.data`, and `.bss` segments of the object file in `/usr/src/linux/vmlinux`. The extension `.nh` indicates that this file has no object−file header.

The concatenation of these three files should be written to the disk from which you want to boot. For example, to boot from a floppy, insert an empty floppy disk in, say, `/dev/fd0` and then type:

```
# cat tools/lxboot tools/bootlx vmlinux >/dev/fd0
```

You can then shutdown the system and boot from the floppy by issuing the command `boot dva0`.

5. The aboot Loader

When using the SRM firmware, `aboot` is the preferred way of booting Linux. It supports:

- direct booting from various filesystems (`ext2`, `ISO9660`, and `UFS`, the DEC Unix filesystem)
- listing directories and following symbolic links on ext2 (version 0.6 and later)
- booting of executable object files (both ELF and ECOFF)
- booting compressed kernels
- network booting (using bootp)
- partition tables in DEC Unix format (which is compatible with BSD Unix partition tables)
- interactive booting and default configurations for SRM consoles that cannot pass long option strings
- load initrd images to load modules at boot time (0.7 and later)

5.1. Getting and Building aboot

The latest sources for `aboot` are available from alphalinux.org and alphalinux.org mirrors. They can also be obtained via CVS from www.alphalinux.org, to get the latest version from CVS use these commands:

```
bash$ export CVSROOT=':pserver:anonymous@www.alphalinux.org:/home/axplinux/cvs/development'
bash$ cvs login
bash# cvs -z3 co aboot
```

(Note there is no password for the CVS login, just press enter)

The description in this manual applies to `aboot` version 0.6 or newer. Please note that many distributions ship aboot with them so downloading aboot from this directory is probably not neccesary.

Once you downloaded and extracted the latest tar file, take a look at the `README` and `INSTALL` files for installation hints. In particular, be sure to adjust the variables in `Makefile` and in `include/config.h` to match your environment. Normally, you won't need to change anything when building under Linux, but it is always a good idea to double check. If you're satisfied with the configuration, simply type `make` to build it (if you're not building under Linux, be advised that `aboot` requires GNU `make`).

After running `make`, the `aboot` directory should contain the following files:

aboot

  This is the actual `aboot` executable (either an ECOFF or ELF object file).

bootlx

  Same as above, but it contains only the text, data and bss segments−−−that is, this file is not an object file.

sdisklabel/writeboot

Utility to install `aboot` on a hard disk.

tools/e2writeboot

Utility to install `aboot` on an ext2 filesystem (usually used for floppies only).

tools/isomarkboot

Utility to install `aboot` on a iso9660 filesystem (used by CD−ROM distributors).

tools/abootconf

Utility to configure an installed `aboot`.

5.2. Floppy Installation

The bootloader can be installed on a floppy using the `e2writeboot` command (note: this can't be done on a Jensen since its firmware does *not* support booting from floppy). This command requires that the disk is not overly fragmented as it needs to find enough contiguous file blocks to store the entire `aboot` image (currently about 90KB). If `e2writeboot` fails because of this, reformat the floppy and try again (e.g., with `fdformat(1)`). For example, the following steps install `aboot` on floppy disk assuming the floppy is in drive `/dev/fd0`:

```
# fdformat /dev/fd0
# mke2fs /dev/fd0
# e2writeboot /dev/fd0 bootlx
```

5.3. Harddisk Installation

Since the `e2writeboot` command may fail on highly fragmented disks and since reformatting a harddisk is not without pain, it is generally safer to install `aboot` on a harddisk using the `swriteboot` command. `swriteboot` requires that the first few sectors are reserved for booting purposes. We suggest that the disk be partitioned such that the first partition starts at an offset of 2048 sectors. This leaves 1MB of space for storing `aboot`. On a properly partitioned disk, it is then possible to install `aboot` as follows (assuming the disk is `/dev/sda`):

```
# swriteboot /dev/sda bootlx
```

On systems where partition `c` in the entire disk it will be necessary to 'force' the write of aboot. In this case use the `−f` flag followed by the partition number (in the case of partition `c` this is 3):

```
# swriteboot /dev/sda bootlx −f3
```

On a Jensen, you will want to leave some more space, since you need to write a kernel to this place, too−−−2MB should be sufficient when using compressed kernels. Use `swriteboot` as described in Section Section 5.6 to write `bootlx` together with the Linux kernel.

5.4. CD−ROM Installation

To make a CD−ROM bootable by SRM, simply build `aboot` as described above. Then, make sure that the `bootlx` file is present on the iso9660 filesystem (e.g., copy `bootlx` to the directory that is the filesystem master, then run `mkisofs` on that directory). After that, all that remains to be done is to mark the filesystem as SRM bootable. This is achieved with a command of the form:

```
# isomarkboot filesystem bootlx
```

The command above assumes that `filesystem` is a file containing the iso9660 filesystem and that `bootlx` has been copied into the root directory of that filesystem. That's it!

5.5. Building the Linux Kernel

A bootable Linux kernel can be built with the following steps. During the `make config`, be sure to answer "yes" to the question whether you want to boot the kernel via SRM (for certain platforms this is automatically selected). Note that if you build a generic kernel (by selecting "Generic" as the alpha system type), the kernel is able to guess whether it is running under SRM or not.

```
# cd /usr/src/linux
# make config
# make dep
# make boot
# make modules (if applicable)
# make modules_install (if applicable)
```

The last command will build the file `arch/alpha/boot/vmlinux.gz` which can then be copied to the disk from which you want to boot from. In our floppy disk example above, this would entail:

```
# mount /dev/fd0 /mnt
# cp arch/alpha/boot/vmlinux.gz /mnt
# umount /mnt
```

5.6. Booting Linux

With the SRM firmware and `aboot` installed, Linux is generally booted with a command of the form:

```
boot devicename -fi filename
-fl flags
```

The *filename* and *flags* arguments are optional. If they are not specified, SRM uses the default values stored in environment variables `BOOTDEF_DEV` , `BOOT_OSFILE` and `BOOT_OSFLAGS`. The syntax and meaning of these two arguments is described in more detail below. To list the current values of these variables type `show boot*` at the SRM command prompt. This will also show a boot_dev variable (among others), this variable is read only and needs to be changed via the bootdef_dev variable.

5.6.1. Device Naming

This corresponds to the device from which SRM will attempt to boot. Examples include:

dva0

– First floppy drive, `/dev/fd0` under Linux

dqa0

– Primary IDE cdrom or hard disk as Master, `/dev/hda` under Linux

dqa1

– Primary IDE cdrom or hard disk as Slave, `/dev/hdb` under Linux

dka0

– SCSI disk on first bus, Device 0, `/dev/sda` under Linux

ewa0

– First Ethernet Device, `/dev/eth0` under Linux

For example to boot from the disk at SCSI id 6, you would enter:

```
>>> boot dka600
```

To list the devices currently installed in the system type `show dev` at the SRM command line. In contrast to Linux device naming, the partition number on a disk device is *not* given as part of the device name (you may see extra numbers after the device names when running `show dev` – these correspond to things like PCI bus and device numbers and are not useful to the user). Remember, as mentioned in Section 2.3, that SRM knows *nothing* about partitions or disklabels – it merely reads a boot block and secondary bootstrap from sectors on a disk. Therefore, the partition number is given as part of the boot filename.

5.6.2. Boot Filename

The filename argument takes the form: "[*n*/]*filename*"

n is a single digit in the range 1..8 that gives the partition number from which to boot from. *filename* is the path of the file you want boot. For example to boot a kernel named vmlinux.gz from the second partition of SCSI device 6, you would enter:

```
>>> boot dka600 -file 2/vmlinux.gz
```

Or to boot from floppy drive 0, you'd enter:

```
>>> boot dva0 -file vmlinux.gz
```

If a disk has no partition table, `aboot` pretends the disk contains one `ext2` partition starting at the first diskblock. This allows booting from floppy disks.

As a special case, partition number 0 is used to request booting from a disk that does not (yet) contain a file system. When specifying "partition" number 0, `aboot` assumes that the Linux kernel is stored right behind the `aboot` image. Such a layout can be achieved with the `swriteboot` command. For example, to setup a filesystem−less boot from `/dev/sda`, one could use the command:

```
# swriteboot /dev/sda bootlx vmlinux.gz
```

Booting a system in this way is not normally necessary. The reason this feature exists is to make it possible to get Linux installed on a systems that can't boot from a floppy disk (e.g., the Jensen).

5.6.3. Boot Flags

A number of bootflags can be specified. The syntax is:

```
-flags "options..."
```

Where "options..." is any combination the following options (separated by blanks). There are many more bootoptions, depending on what drivers your kernel has installed. The options listed below are therefore just examples to illustrate the general idea:

load_ramdisk=1

> Copy root file system from a (floppy) disk to the RAM disk before starting the system. The RAM disk will be used in lieu of the root device. This is useful to bootstrap Linux on a system with only one floppy drive.

floppy=str

Sets floppy configuration to *str*.

root=dev

Select device *dev* as the root−file system. The device can be specified as a major/minor hex number (e.g., 0x802 for /dev/sda2) or one of a few canonical names (e.g., `/dev/fd0`, `/dev/sda2`).

single

Boot system in single user mode.

kgdb

Enable kernel−gdb (works only if `CONFIG_KGDB` is enabled; a second Alpha system needs to be connected over the serial port in order to make this work)

Some SRM implementations (e.g., the one for the Jensen) are handicapped and allow only short option strings (e.g., at most 8 characters). In such a case, `aboot` can be booted with the single−character boot flag "i". With this flag, `aboot` will enter interactive mode

5.6.4. Using aboot interactively

As of version 0.6, `aboot` supports a simple command−oriented interactive mode. Note that this is *different* from the prompt which previous versions issued when booted with the "i" flag, or after failing to load a kernel. You can get a summary of the available commands by typing "h" or "?" at the prompt:

```
>>> boot dka0 -fl i
aboot> ?
 h, ?                   Display this message
 q                      Halt the system and return to SRM
 p 1-8                  Look in partition <num> for configuration/kernel
 l                      List pre-configured kernels
 d <dir>                List directory <dir> in current filesystem
 b <file> <args>        Boot kernel in <file> (- for raw boot)
                        with arguments <args>
 0-9                    Boot pre-configuration 0-9 (list with 'l')
aboot> b 3/vmlinux.gz root=/dev/sda3 single
```

5.6.5. The aboot.conf configuration file

Since booting in that manner quickly becomes tedious, `aboot` allows to define short−hands for frequently used command lines. In particular, a single digit option (0−9) requests that `aboot` uses the corresponding option string stored in file `/etc/aboot.conf`. A sample `aboot.conf` is shown below:

```
#
# aboot default configurations
```

```
#
0:3/vmlinux.gz root=/dev/sda3
1:3/vmlinux.gz root=/dev/sda3 single
2:3/vmlinux.new.gz root=/dev/sda3
3:3/vmlinux root=/dev/sda3
8:- root=/dev/sda3            # fs-less boot of raw kernel
9:0/vmlinux.gz root=/dev/sda3 # fs-less boot of (compressed) ECOFF kernel
-
```

With this configuration file, the command

```
>>> boot dka0 -fl 1
```

corresponds exactly to the boot command shown above.

Finally, at the `aboot` prompt, it is possible to enter one of the single character flags ("0"−"9") to get the same effect as if that flag had been specified in the boot command line. As noted in the help text cited above, you can also list the available default configurations with the "l" command.

5.6.5.1. Selecting the Partition of /etc/aboot.conf

When installed on a harddisk, `aboot` needs to know what partition to search for the `/etc/aboot.conf` file. A newly compiled `aboot` will search the *second* partition (e.g., `/dev/sda2`). Since it would be inconvenient to have to recompile `aboot` just to change the partition number, `abootconf` allows to directly modify an installed `aboot`. Specifically, if you want to change `aboot` to use the *third* partition on disk `/dev/sda`, you'd use the command:

```
# abootconf /dev/sda 3
```

You can verify the current setting by simply omitting the partition number. That is: `abootconf /dev/sda` will print the currently selected partition number. Note that `aboot` does have to be installed already for this command to succeed. As of version 0.6, `swriteboot` it will preserve the existing configuration when installing a new `aboot` on a hard disk.

Since `aboot` version 0.5, it is also possible to select the `aboot.conf` partition via the boot command line. This can be done with a command line of the form *a*`:`*b* where *a* is the partition that holds `/etc/aboot.conf` and *b* is a single−letter option as described above (`0-9`, `i`, or `h`). For example, if you type `boot -fl "3:h" dka100` the system boots from SCSI ID 1, loads `/etc/aboot.conf` from the third partition, prints its contents on the screen and waits for you to enter the boot options.

5.7. Setting up a BOOTP capable server using DHCP

The following configuration assumes that the server is running RH−6.2. Prerequisites packages are,

- dhcp−2.0.5
- tftp−server−0.16.5

5.7.1. DHCP & BOOTP configuation

Once those packages are installed there are a few setup issues to take care of.

Create the default directory to which files will be pulled from using tftp.

```
# mkdir /tftpboot
```

Create the dhcp.leases file which is not create per default (though it should be) when you install the dhcp package so the dhcp server may start.

```
# mkdir -p /var/state/dhcp
# touch /var/state/dhcp/dhcpd.leases
```

Configure the inetd to accept the tftp service. Edit your /etc/inetd.conf file and locate the following line. Then uncomment it and save the file.

```
#tftp   dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd
```

Create the /etc/dhcp.conf configuation file. An example config is provided below with the directives which allow BOOTP.

```
subnet 192.168.1.0 netmask 255.255.255.0 {
       option routers                         192.168.1.1;
       option subnet-mask                   255.255.255.0;
       option nis-domain                 "alphalinux.org";
       option domain-name                "alphalinux.org";
       option domain-name-servers             192.168.1.2;
       range                 192.168.1.3   192.168.1.254;
       range dynamic-bootp   192.168.1.3   192.168.1.254;
       default-lease-time                           21600;
       max-lease-time                               43200;
       allow bootp;
       allow booting;
       filename "/tftpboot/vmlinux.bootp";
}
```

5.7.1.1. Examination of /etc/dhcp.conf

There are four directives that you should be concerned with.

- `range dynamic-bootp 192.168.1.3 192.168.1.254;` which defines the range of ip's available for bootp.
- `allow bootp;` which tells the dhcp server to allow the bootp protocol..
- `allow booting;` which tells the dhcp server to allow the transfer of the file specified either in the the "filename" directive or passed in the "−file" flag in SRM.
- `filename "/tftpboot/vmlinux.bootp";` which is the default file which is transferred and executed when no filename specified in SRM as an argument.

Lastly, Restart the inetd daemon so that the changes we made can take effect

```
# service inet restart
```

You should now have a DHCP server that is capable of BOOTP.

5.7.2. bootpd configuration

The bootpd is the older way of making a bootp server and for the most part is not used anymore in lieu of more modern DHCP servers that are capable of handling the protocol with minimal configuration and more flexibility. This style of setup does not allow just any client to be granted a BOOTP request. Instead you must specify the ip address and MAC address of the allowed clients. Naturally this could get quite tedious if you where say administrating more than a few machines.

bootpd rpms can be found on older versions of RedHat's distributions like version 5.2 and below. Note: the rpm itself is named bootp though the package does contain the bootpd filename. It is available for download at your favorite RedHat mirror. The bootp package requires the tftp−server just as before and the location to where the files are grabbed from is the same.

Once installed you must configure your inetd service to talk to the bootpd daemon. Uncomment the following line in your /etc/inetd.conf .

```
#bootps dgram   udp     wait    root    /usr/sbin/tcpd  bootpd
```

Then restart the inetd.

```
# service inet restart
```

Configuring the `/etc/bootptab` file. The `bootptab` file has one entry describing each client that is allowed to boot from the server. For example, if you want to boot the machine `voodoo.alphalinux.org`, then an entry of the following form would be needed:

```
voodoo.alphalinux.org:\
        :hd=/tftpboot/:bf=vmlinux.bootp:\
        :ht=ethernet:ha=08012B1C51F8:hn:vm=rfc1048:\
        :ip=192.12.69.254:bs=auto:
```

This entry assumes that the machine's Ethernet address is `08012B1C51F8` and that its IP address is 192.12.69.254. The Ethernet address can be found with the `show device` command of the SRM console or, if Linux is running, with the `ifconfig` command. The entry also defines that if the client does not specify otherwise, the file that will be booted is `vmlinux.bootp` in directory `/tftpboot`. For more information on configuring `bootpd`, please refer to its man page.

5.8. Booting Over the Network

Three steps are necessary before Linux can be booted via a network. First you need an Ethernet adapter that is supported by SRM. Most version of SRM support the DE500 series of cards, with newer versions (5.6 and later) also supporting the Intel EtherExpress/Pro series of cards. Second, you need to set the SRM environment variables to enable booting via the bootp protocol and third you need to setup another machine as the your boot server. Enabling bootp in SRM is usually done by setting the ewa0_protocol (DE500 cards) or eia0_protocol (Intel cards) variable to bootp.

```
>>> set ewa0_protocol bootp
```

Also check to see that your ethernet device has a link light to whatever hub or switch it is connected to. If you do not see a link light try forcing the negotiation of the ethernet device. For example:

```
>>> set ewa0_mode FastFD
```

Would set the DE500 ethernet card to fast full duplex operation. To see a list of the available modes

```
>>> set ewa0_mode
```

Netboot using the aboot sources is currently broken though for the curious the steps needed are further below. Instead use the directions for netbooting using the kernel sources.

5.8.1. Netboot using the kernel sources

1. Make sure the kernel you want to boot has already been built
2. Execute the following while in the linux source dir:

    - `make bootimage`
    - `make bootpfile`

    This creates a uncompressed kernel named 'bootpfile' located in arch/alpha/boot/ . Note that this kernel is significantly larger than that produced by the aboot sources.
3. Copy bootpfile to the bootp server's directory. With a default setup the tftp server would look in /tftpboot so copy bootpfile into /tftpboot .

5.8.2. Netboot using the aboot sources

1. Build aboot with with the command `make netboot`.
2. Make sure the kernel that you want to boot has been built already. By default, the aboot Makefile uses the kernel in /usr/src/linux/arch/alpha/boot/vmlinux.gz (edit the Makefile if you want to use a different path). The result of make netboot is a file called vmlinux.bootp which contains aboot and the Linux kernel, ready for network booting.
3. Copy vmlinux.bootp to the bootp server's directory. In the example above, you'd copy it into /tftpboot/vmlinux.bootp.

Next, power up the client machine and boot it, specifying the Ethernet adapter as the boot device. Typically, SRM calls the DEC based Ethernet adapter ewa0 and the Intel based adapter eia0, so to boot from that device, you'd use the command:

```
      >>> boot ewa0
```

The −fi and −fl options can be used as usual. For example,

```
      >>> boot ewa0 -fi  bootpfile -fl "root=/dev/hda2"
```

In particular, you can ask aboot to prompt for Linux kernel arguments by specifying the option −fl i .

5.8.3. Updating the SRM console through BOOTP

Updating your SRM console over the network through BOOTP is just as easy as booting the Linux kernel in the same manner. The hardware prerequisites are the same as netbooting Linux.

First you have to obtain an SRM image that is able to BOOTP over the network. These images normally have a .exe extension. For DEC/Compaq Alpha products these images can be found at ftp://gatekeeper.dec.com/pub/DEC/Alpha/firmware/v5.8/. You can also find these files on the Alpha Systems Firmware Update CD−ROM. API NetWorks does not offer net bootable SRM images at this time though that may change in the near future.

For example say you had a DS20 and wanted to update it's firmware over the network using BOOTP. You would have to,

1. Get the correct firmware image for the DS20 that supported BOOTP execution which in this case the filename is `ds20_v5_8.exe` from ftp://gatekeeper.dec.com/pub/DEC/Alpha/firmware/v5.8/.
2. Copy the file to the /tftpboot folder located on the BOOTP server.

To execute the update from SRM you would do the following:

```
>>> b ewa0 -fi ds20_v5_8.exe
```

SRM would then proceed to upgrade the firmware in the same fashion as if you had done the firmware update from a CD.

5.9. Partitioning Disks

5.9.1. What is a disklabel?

A disk label is a partition table. Unfortunately, there are several formats the partition table can take, depending on the operating system.

DOS partition tables are the standard used by Linux and Windows. AlphaBIOS systems and every Linux kernel can read DOS partition tables. Unfortunately, the SRM console's boot sector format overlaps with parts of the DOS partition table on disk, and therefore DOS partition tables cannot be used with SRM.

BSD disklabels are used by several variants of Unix, including Tru64. SRM's boot block does not conflict with the BSD disklabel (in fact, the BSD disklabel resides entirely within "reserved" areas of the first sector), and Linux can use a BSD disklabel, provided that support for BSD disklabels has been compiled into the kernel.

To boot from a disk using SRM, a BSD disklabel is required. If the disk is not a boot disk, the BSD disklabel is not required. A BSD disklabel can be created using fdisk, the standard Linux disk partitioning tool.

5.9.2. Partitioning the Easy Way: a DOS Disklabel

The simplest way to partition your disk is to let your Linux installer do it for you, for example by using Red Hat's disk druid or fdisk. On Red Hat 6.1, this will produce a valid BSD disklabel, but *only* if the disk in question previously contained one. In most cases, this will produce a DOS disklabel. It will be readable by Linux, but you will not be able to boot from it via SRM. For this reason, you will probably want to create a BSD disklabel manually in order to boot Linux

5.9.3. Partitioning with a BSD Disklabel

1. Start fdisk on the disk you're configuring
2. Choose to make a BSD disklabel − option 'b' (newer versions of fdisk will detect existing BSD disklabels and automatically enter disklabel mode)
3. You'll notice some things: Partitions are letters instead of numbers, from a−h Partition 'c' covers the whole of the disk. This is the convention, don't touch it. While you can see it, note down the disk parameters as you'll use them more often than with the DOS−disklabel approach
4. Creating a new partition uses the same procedure as the DOS−disklabel approach, except that the partitions are referred to by letter instead of number. That is, 'n' to make a new partition followed by the partition letter followed by the starting block followed by the end block
5. Setting partition type is slightly different, because the numbering scheme is different (1 is swap, 8 is ext2).
6. When you are finished, write ('w') and quit ('q') as normal.

There are some important catches that you must be aware of when partitioning using a BSD disklabel:

- Partition 'a' should start about 1M into the disk: don't start it at sector 1, try starting at sector 10 (for example). This leaves plenty of space for writing the boot block (see below)
- There is a bug in some versions of fdisk which makes the disk look one sector bigger than it actually is. The listing when you create the BSD disklabel is correct. The last sector of partition 'c' is correct. The default last sector when creating a new partition is 1 sector too big
- Always adjust for this extra sector. This bug exists in the version of fdisk shipped with Red Hat 6.0. Not making an adjustment for this problem almost always leads to "Access beyond end of device" errors from the Linux kernel.

Once you have made a BSD disklabel, continue the installation. After installation, you can write a boot block to your disk to make it bootable from SRM.

6. Sharing a Disk With DEC Unix

Unfortunately, DEC Unix doesn't know anything about Linux, so sharing a single disk between the two OSes is not entirely trivial. However, it is not a difficult task if you heed the tips in this section. The section assumes you are using `aboot` version 0.5 or newer.

6.1. Partitioning the disk

First and foremost: *never* use any of the Linux partitioning programs (`minlabel` or `fdisk`) on a disk that is also used by DEC Unix. The Linux `minlabel` program uses the same partition table format as DEC Unix `disklabel`, but there are some incompatibilities in the data that `minlabel` fills in, so DEC Unix will simply refuse to accept a partition table generated by `minlabel`. To setup a Linux `ext2` partition under DEC Unix, you'll have to change the disktab entry for your disk. For the purpose of this discussion, let's assume that you have an rz26 disk (a common 1GB drive) on which you want to install Linux. The disktab entry under DEC Unix v3.2 looks like this (see file `/etc/disktab`):

```
rz26|RZ26|DEC RZ26 Winchester:\
        :ty=winchester:dt=SCSI:ns#57:nt#14:nc#2570:\
        :oa#0:pa#131072:ba#8192:fa#1024:\
        :ob#131072:pb#262144:bb#8192:fb#1024:\
        :oc#0:pc#2050860:bc#8192:fc#1024:\
        :od#393216:pd#552548:bd#8192:fd#1024:\
        :oe#945764:pe#552548:be#8192:fe#1024:\
        :of#1498312:pf#552548:bf#8192:ff#1024:\
        :og#393216:pg#819200:bg#8192:fg#1024:\
        :oh#1212416:ph#838444:bh#8192:fh#1024:
```

The interesting fields here are `o`*?*, and `p`*?*, where *?* is a letter in the range `a-h` (first through 8-th partition). The `o` value gives the starting offset of the partition (in sectors) and the `p` value gives the size of the partition (also in sectors). See `disktab(4)` for more info. Note that DEC Unix likes to define overlapping partitions. For the entry above, the partition layout looks like this (you can verify this by adding up the various `o` and `p` values):

```
  a     b         d           e           f
|---|-------|-----------|-----------|-----------|

                             c
|-----------------------------------------------|

                          g                 h
            |-----------------|-----------------|
```

DEC Unix insists that partition `a` starts at offset 0 and that partition `c` spans the entire disk. Other than that, you can setup the partition table any way you like.

Let's suppose you have DEC Unix using partition `g` and want to install Linux on partition `h` with partition `b` being a (largish) swap partition. To get this layout without destroying the existing DEC Unix partition, you need to set the partition types explicitly. You can do this by adding a t field for each partition. In our case, we add the following line to the above disktab entry.

```
        :ta=unused:tb=swap:tg=4.2BSD:th=resrvd8:
```

Now why do we mark partition h as "reservd8" instead of "ext2"? Well, DEC Unix doesn't know about Linux. It so happens that partition type "ext2" corresponds to a numeric value of 8, and DEC Unix uses the string "reservd8" for that value. Thus, in DEC Unix speak, "reservd8" means "ext2". OK, this was the hard part. Now we just need to install the updated disktab entry on the disk. Let's assume the disk has SCSI id 5. In this case, we'd do:

```
# disklabel −rw /dev/rrz5c rz26
```

You can verify that everything is all right by reading back the disklabel with disklabel −r /dev/rrz5c. At this point, you may want to reboot DEC Unix and make sure the existing DEC Unix partition is still alive and well. If that is the case, you can shut down the machine and start with the Linux installation. Be sure to skip the disk partitioning step during the install. Since we already installed a good partition table, you should be able to proceed and select the 8th partition as the Linux root partition and the 2nd partition as the swap partition. If the disk is, say, the second SCSI disk in the machine, then the device name for these partitions would be /dev/sdb8 and /dev/sdb2, respectively (note that Linux uses letters to name the drives and numbers to name the partitions, which is exactly reversed from what DEC Unix does; the Linux scheme makes more sense, of course ;−).

6.2. Installing aboot

First big caveat: with the SRM firmware, you can boot one and only one operating system per disk. For this reason, it is generally best to have at least two SCSI disks in a machine that you want to dual−boot between Linux and DEC Unix. Of course, you could also boot Linux from a floppy if speed doesn't matter or over the network, if you have a bootp−capable server. But in this section we assume you want to boot Linux from a disk that contains one or more DEC Unix partitions.

Second big caveat: installing aboot on a disk shared with DEC Unix renders the first and third partition unusable (since those *must* have a starting offset of 0). For this reason, we recommend that you change the size of partition a to something that is just big enough to hold aboot (1MB should be plenty).

Once these two caveats are taken care of, installing aboot is almost as easy as usual: since partition a and c will overlap with aboot, we need to tell swriteboot that this is indeed OK. We can do this under Linux with a command line of the following form (again, assuming we're trying to install aboot on the second SCSI disk):

```
# swriteboot −f1 −f3 /dev/sdb bootlx
```

The −f1 means that we want to force writing bootlx even though it overlaps with partition 1. The corresponding applies for partition 3.

This is it. You should now be able to shutdown the system and boot Linux from the harddisk. In our example, the SRM command line to do this would be:

```
>>> boot dka5 -fi 8/vmlinux.gz -fl root=/dev/sdb8
```

7. Installation of Distributions

7.1. RedHat 6.0, 6.1 and 6.2

7.1.1. Installation from the Red Hat 6.0, 6.1 or 6.2 CD

Red Hat have made their distribution CD bootable from SRM console [2] To start an installation, put the CD in and type the following:

```
>>> boot srm-device -file kernels/generic.gz -flags root=linux-device
```

In the above, the SRM device name and Linux device name for your CD−ROM drive are needed. For Example if the machine had an IDE cdrom installed as primary master the command would look like this:

```
>>> boot dqa0 -file kernels/generic.gz -flags "root=/dev/hda"
```

See the section on Section 5.6.1 conventions if you don't know what these are.

7.2. SuSE 6.1

7.2.1. Installation from the SuSE 6.1 CD

The SuSE 6.1 CD is not bootable from SRM console. SuSE have an alternative approach which involves creating two boot floppies, the images of which are included on the CD. The boot disks can be created in various ways, depending on the systems you have available

Writing the boot disks from a linux system The command to use is dd. From the mount−point of SuSE CD 1, the commands are:

```
# dd if=disks/aboot of=/dev/fd0
# dd if=disks/install of=/dev/fd0
```

For writing the boot disks from a windows system, the command to use is rawrite. It is available on the CD.

```
 D:\tools\> rawrite
```

The program then prompts for input disk image and output disk drive. Run this command once for each of the disk images as shown above.

Starting the SuSE installer from the boot disks With the floppy disk made from the aboot image in place, type:

```
>>> boot dva0 -file vmlinux.gz -flags "root=/dev/fd0 load_ramdisk=1"
```

This will start the kernel, prompt you for the second boot disk, and start the installer

7.3. SuSE 6.3

7.3.1. Installation from the SuSE 6.3 CD

The SuSE 6.3 CD−ROM is SRM bootable much like the RedHat 6.0 and 6.1 CD−ROMs. The best way to start the install from SRM is to use the following command:

```
>>> boot srm-device -flags 0
```

In the above, the SRM device names for your CD−ROM drive is needed. For Example if the machine had an IDE cdrom installed as primary master the command would look like this:

```
>>> boot dqa0 -flags 0
```

SuSE has added support to aboot to allow it to load initrd files. The above command will from the CD−ROM drive and use config number 0 from the /etc/aboot.conf file. For other variations on this refer to the SuSE installation guide.

8. Document History

v0.8 9th November 2000 Changed from Rich Payne <rdp@alphalinux.org>

- Added section on SRM Device names
- Many spelling/grammer fixes.

v0.7.1 6th November 2000 Changes from Peter Petrakis <ppetrakis@alphalinux.org>

- Cleaned up netbooting section. Avoid duplicate information.
- Added DHCP/BOOTP server configuration section.
- Added SRM netbooting section.
- Put the older bootpd configuration in it's own section and elaborated on it.

v0.7 10th July 2000 Changes from Rich Payne <rdp@alphalinux.org>

- Updated for RedHat 6.2
- Fixed aboot link for alphalinux.org and added CVS information.
- Added additional netboot information from Peter Petrakis <ppetrakis@alphalinux.org>

v0.6.1 21 March 2000 Changes from Rich Payne <rdp@alphalinux.org>

- Made the installation hints a new chapter
- Added information on Netbooting
- Added to the new section on RedHat 6.1 and BSD disklabels
- Removed David Mosberger−Tang's name from the authors list
- Marked a few of the feature as being in 0.6 only
- Added info for SuSE 6.3 and RedHat 6.1

v0.6 3 March 2000 Changes and information from David Huggins−Daines <dhd@linuxcare.com>

- Moved the notes on MILO vs. SRM to an "About this document" section
- Added sections on switching to SRM, and basic SRM usage
- Added section on the new interactive use of aboot
- Updated the note on DOS partition tables to mention the Red Hat 6.1 installer's behavior.
- Normalized the markup, and codified the conventions used for user−entered commands.
- Corrected the notes on BSD disklabels (SRM does *not* read BSD disklabels, it's just that they don't conflict with the boot block).

v0.5.2 5 December 1999 Added comments and information from Stig Telfer (stig @ alpha−processor.com).

- Added chart on SRM to Linux name mappings
- Added RedHat 6.0 and SuSE 6.1 installation information
- Added Disk Partitioning Information

v0.5.1 (Not Released) 13 November 1999 Took the original 0.5 document and updated several parts:

- Update information on SRM booting from IDE devices
- Fixed URL to aboot source
- Update toc page to reflect MILO's future
- Included information on bootdef_dev and boot_dev to chapter 3
- Added this section

v0.5 17 August 1996 – Original Document by David Mosberger−Tang

Notes

[1]
: On multiprocessor machines, you will see 'P00>>' instead, or possibly some other number depending on which processor SRM is running.

[2]
: Please note that through the official RedHat CD−ROM is SRM bootable, copies made by various other companies may not be bootable.