SRM Firmware Howto

David Mosberger <mailto:davidm@azstarnet.com> v0.5, 17 August 1996

This document describes how to boot Linux/Alpha using the SRM firmware, which is the firmware normally used to boot DEC Unix. Generally, it is preferable to use MILO instead of aboot since MILO is perfectly adapted to the needs of Linux. However, MILO is not always available for a particular system and MILO does not presently have the ability to boot over the network. In either case, using the SRM console may be the right solution.

Contents

Unless you're interested in technical details, you may want to skip right to Section 3.

1 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. Also, booting from IDE disk drives is unsupported.

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.

1.1 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`.

2 The Raw Loader

The sources for this loader can be found in directory

```
linux/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 `/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 issueing the command `boot dva0`.

3 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)
- 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

3.1 Getting and Building aboot

The latest sources for `aboot` are available in *this ftp directory* `<ftp://ftp.azstarnet.com/pub/linux/axp/aboot>`. The description in this manual applies to `aboot` version 0.5 or newer.

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`.

3.2 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
```

3.3 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 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 3.6 to write `bootlx` together with the Linux kernel.

3.4 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!

3.5 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.

```
cd /usr/src/linux
make config
make dep
make boot
```

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
```

3.6 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 `BOOT_OSFILE` and `BOOT_OSFLAGS`. The syntax and meaning of these two arguments is described in more detail below.

3.6.1 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 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).

3.6.2 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 prompt the user to interacively enter a boot option string of up to 256 characters. For example:

```
boot dka0 -fl i
aboot> 3/vmlinux.gz root=/dev/sda3 single
```

Since booting in that manner quickly becomes tedious, `aboot` allows to define short-hands for frequently used commandlines. 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. It is quite easy to forget what number corresponds to what option string. To alleviate this problem, boot with option "h" and `aboot` will print the contents of `/etc/aboot.conf` before issueing the prompt for the full option string.

Finally, whenever `aboot` prompts for an option string, it is possible to enter one of the single character flags ("i", "h", or "0"-"9") to get the same effect as if that flag had been specified in the boot command line. For example, you could boot with flag "i" and then type "h" (followed by return) to remind yourself of the contents of `/etc/aboot.conf`

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. Also, when installing a new `aboot`, the partition number will fall back to the default (i.e., it will be necessary to rerun `abootconf`).

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.

3.7 Booting Over the Network

Two prelimenary steps are necessary before Linux can be booted via a network. First, you need to set the SRM environment variables to enable booting via the bootp protocol and second you need to setup another machine as the your boot server. Please refer to the SRM documentation that came with your machine for information on how to enable bootp. Setting up the boot server is obviously dependent on what operating system that machine is running, but typically it involves starting the program `bootpd` in the background after 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 `myhost.cs.arizona.edu`, then an entry of the following form would be needed:

```
myhost.cs.arizona.edu:\
        :hd=/remote/: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 `/remote`. For more information on configuring `bootpd`, please refer to its man page.

Next, build `aboot` with with the command `make netboot`. 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.

Finally, copy `vmlinux.bootp` to the bootsever's directory. In the example above, you'd copy it into `/remote/vmlinux.bootp`. Next, power up the client machine and boot it, specifying the Ethernet adapter as the boot device. Typically, SRM calls the first Ethernet adapter `ewa0`, so to boot from that device, you'd use the command:

```
boot ewa0
```

The `-fi` and `-fl` options can be used as usual. In particular, you can ask `aboot` to prompt for Linux kernel arguments by specifying the option `-fl i`.

4 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.

4.1 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 ;-).

4.2 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 dualboot 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
```