The Sparc Linux HOWTO

Donnie Barnes, `djb@redhat.com` V1.1, January 23, 1996

Contents

1 Introduction

This document is in no way complete at this time. It, must like the Sparc port itself, is a work in progress. If you have changes or ammendments you'd like to see made to this document, please contact me directly, `djb@redhat.com`. Much of the information for this document is taken directly from Jim Mintha's WWW pages (`http://www.geog.ubc.ca/sparclinux.html`) and most of the rest of it came from the sparclinux mailing list archive.

2 Other Sources of Information

Probably the best source of information is the SparcLinux WWW site. The URL for it is: `http://www.geog.ubc.ca/sparclinux.html`.

The main FTP site is `ftp://vger.rutgers.edu/pub/linux/Sparc` which is mirrored at `ftp://ftp.cps.cmich.edu` and at `ftp://swell.hampshire.edu/pub/linux/Sparc`. It will also be mirrored at `ftp.redhat.com` *soon*.

Also, if you plan to use or help develop SparcLinux, you should subscribe to the mailing list. To subscribe, send mail to `majordomo@vger.rutgers.edu` with "subscribe sparclinux" in the body and the subject line empty.

3 General Information

This section is basically the equivalent of a FAQ right now. As it grows, a FAQ will become a separate document and only "general" things will stay here.

3.1 What Sparc Machines are Supported?

The current kernels only support Sun4c and Sun4m machines (e.g. Sparc 1, 1+, 2, IPC, IPX, SLC, ELC, Sparc Classic, LX, 5, 10, 20, etc.). SMP Sparcs are also working pretty well, too. Sun4 arch machines will probably not be supported, but anyone with a Sun4 and some time will certainly receive net.help from the community should they choose to try it. The Sun-3 machines are not expected to be supported until someone volunteers their time towards that architecture although most of the underlying generic sparc code is there.

The Sun-3 is based on a completely different processor (the Motorola 680x0) and should most likely be done in concert with the 680x0 Linux port people. The thing to know now is that the linux-m68k project is only working on code for machines that use the Motorola MMU. Most Sun 3 machines (except for sun3x arch machines like the 3/80) have a sparc-like MMU. So, a decent hacker should be able to get the m68k core code and combine it with the MMU stuff from the sparc port.

3.2 Be Careful with `ext2fs` tools!

Because of how large the disk label is on the Sun, the ext2 filesystem on the first partition of a scsi disk will overwrite the label. This is lethal and will probably spam your entire scsi drive. (personally after I format a disk for testing and make out the partitions I do a:

```
dd if=/dev/sd3 of=~davem/diskblah.label bs=1024 count=8192
```

so I can recover if I fall for this problem). So my advice is:

- If you are formatting a disk for the first time, make the first partition start at cylinder 1, this completely avoids this problem.
- If not, whatever you do don't make an ext2 filesystem on partition 1.

You have been warned. When we get linux fdisk working we can forget about this, but for now just be careful.

3.3 MAKEDEV Instructions

There is a MAKEDEV script at `tools/MAKEDEV` on the FTP sites.

While preparing a root filesystem for your system, just place the above script in `dev/MAKEDEV`, make it executable and run it while `dev/` is your current working directory.

I've managed to get it to work under SunOS/Solaris while making nfs root directories, it gives lots of error messages due to unknown groups/users and the fact that it can't look at certain `/proc` files under those OS's. I usually just go:

```
cd /my/nfs/root/directory/dev
MAKEDEV generic
```

That should do the trick. You may want to trim things down if this is for a floppy (anyone who needs 60 or so ptys on a root floppy is crazy) ;-).

3.4 How do I get SunOS shared executables to run under SparcLinux?

Shared SunOS executables work if you put `/usr/lib/ld.so` `/etc/ld.so.cache` and `/usr/lib/lib*.so.*` onto your root partition. For the `getpwd()` library call to work using the sun shared libc, a self contained static `/bin/pwd` is necessary on the system. It is recommended to use the native SparcLinux executable from the sh-utils package:

```
vger.rutgers.edu:/pub/linux/Sparc/userland/sh-*.SPARC.tar.gz
```

3.5 How do I find out Kernel State Information?

A good way to see what the kernel is up to is to type:

- Control-ScrollLock /* A listing of the current processes */
- RightShift-ScrollLock /* Memory Management information */
- RightAltGraph-ScrollLock /* Sparc Register Dump */

3.6 How do I setup a cross compiling environment?

(The following was donated by Jim Nance.)

Right now, there are two bugs in this process. When I receive an updated version of this without the bugs, I will change it. The two bugs are:

- The correct description name for the configure scripts is not sun4
- The latest binutils will not build a bootable kernel

The following instructions apply to building an environment where one can cross compile SparcLinux binaries from an Intel machine running Linux. Currently this method does not include building any libraries, so you can only compile the kernel.

Before you can actually build a cross compiler, you have to first build and install a cross assembler and a cross linker. The source for these programs are in the binutils package. You can obtain the latest binutils via anonymous ftp from prep.ai.mit.edu in the directory /pub/gnu.

To compile, untar the binutils tarfile in a directory, and cd into the top level binutils directory. Now configure, make, and install the binutils packages by typing:

```
./configure --target=sun4
make
su
make install
```

Just to make sure everything worked, look in /usr/local/sun4/bin and see if you have some files in there, particularly ar and ld.

Now that these packages are installed, you can build the actual cross compiler. To do this, get the latest gcc source (its 7 Megs!), and untar it into a directory and then cd into the top level gcc directory. You can configure and build the cross compiler by typing:

```
./configure --target=sun4
make CFLAGS='-O -s' tooldir=/usr/local/sun4
```

This make will actually fail because there is no way to build libgcc.a. This is not a problem. It has actually build all the things we need by the time it fails. You can now su to root, and install the compiler by typing:

```
cp gcc-cross /usr/local/bin/gcc-sparc
make CFLAGS='-O -s' tooldir=/usr/local/sun4 install
```

We now need to make a few symbolic links to make everything run. Assuming that you are compiling gcc-2.7.2, the commands to do this as root are:

```
cd /usr/lib/gcc-lib/sun4/2.7.2
ln -s /usr/local/sun4/bin/ld
ln -s /usr/local/sun4/bin/ar
```

Finally, you need to get a copy of libgcc.a and put it in this directory. If you have access to a machine running SunOS, one way to do this is to build gcc on that machine with the commands:

```
tar xzvf gcc-2.7.2.tar.gz
cd gcc-2.7.2
./configure
make
```

and then copy the libgcc.a file produced into /usr/lib/gcc-lib/sun4/2.7.2 on your linux machine. If you can not do this, there is a libgcc.a file in the sparc-libc package which may work.

At this point you have a functional cross compiler, but no libraries. This means that you should be able to compile the kernel, but no user programs.

4 How do I get SparcLinux running?

First, make sure you are trying this on a supported architecture! See the previous section for more info on which Sun platforms are currently supported.

4.1 Get the Tools in Place

This is a very rough guide to getting things working:

- Get the latest GNU binutils compiled on your machine. (`prep.ai.mit.edu:/pub/gnu`)
- Get GNU awk and GNU bash on your path (available also on prep.ai.mit.edu)
- Get the latest Linux/SPARC kernel snapshot from the FTP site.
- Untar your kernel.
- Do: `make config; make clean; make depend;make vmlinux`
- The resulting vmlinux file is the Linux kernel. Put it in an appropriate place to boot (like `/` :-).
- If you want to net-boot, check the mailing list archives. Miguel wrote a little description (thanks to Theo from OpenBSD) on how to get a working net boot.

- If you lack the network boot facility, you may be interested in Pete's SILO loader. It is available on the FTP site. It is currently undocumented here, however.

4.2 How to build a SparcLinux Kernel

(The following was contributed by David Miller.)

There is only a certain build environment that is guarenteed to work at this point for building the kernel from stock snapshot sources. This is:

- SunOS4.1.3 or higher OS
- Binutils-2.5.2 *only*
- Gcc-2.6.3 (later versions may work)
- Bash installed somewhere convenient (/bin/bash is nice)
- GNU make

This is the environment I stick to, mostly to avoid problems when debugging where I have to consider it a posibility that a compiler and/or linker bug could be causing problems. Binutils-2.6 is known to produce improper kernel images if a bug it tickled in the GNU linker. I am working on helping the binutils people fix this bug. Gcc-2.6.3 is just what I use, it hasn't caused me any problems and I have grown to trust it on the Sparc. The kernel build process will absolutly need the bash shell available for the configuration of the kernel to work at all. GNU Make is also mandatory.

There is some leeway however. It is possible to use the recommended compiler and binutils versions under solaris if one is careful and uses Peter Zaitcev's (zaitcev@ithil.mcst.ru) elf2aout utility to convert the kernel at the end. Please contact Peter with specific questions concerning this setup, he has seen much success with it, I do not use it however so I cannot comment unfortunately.

Some advice on the configuration process. I will go over each section, one at a time and explain things. This information is current as of Jan. 19, 1996.

Type `make config` at the top level of your untarred sources. Note that most of the defaults are very sane for a stock system.

4.2.1 General Setup

- Networking support: You most likely want this enabled, answer 'y'.
- System V IPC: I can't think of any programs which use this yet, but best to be safe and turn it on, answer 'y'.
- Kernel support for ELF binaries: Peter has done some preliminary work attempting to get elf binaries to work, unless you are attempting to help Peter with this work, answer 'n'.

4.2.2 Block Devices

- Normal floppy disk support: If you plan on booting your system using a floppy root disk or similar, answer 'y'. If not, answer 'n'.
- RAM disk support: Unless you have a specific need to have ramdisk capabilities available, answer 'n' it will make your kernel a bit smaller. One reason you may want to enable this along with the floppy driver is if you are attempting to setup a boot/root floppy combination which loads the root filesystem off a floppy into the ramdisk while creating a SparcLinux distribution for others to use.

4.2.3 Network Options

I will be brief here, basically you want to enable (that is, answer 'y') to only two options in this section of questions, the others you want to disable (answer 'n'). Enable, "TCP/IP networking" and "Drop source routed frames". If you have 16mb of ram or more in your sparcstation you may want to enable "Allow large windows" as it will cause the networking to be more efficient in some circumstances. This option is not recommended for small memory configurations (for example my test box has 8mb of ram, so I don't enable this option).

4.2.4 SCSI Support

If you plan on making filesystems, or using this kernel to boot on an already created ext2 sparclinux setup on one of your scsi drives, you will want to enable scsi support. If not, answer 'n' and make your kernel a bit smaller.

- SCSI support: Answer 'y' if you want to use your scsi disks, 'n' if not.

(the following only applies if you answered 'y' to scsi-support)

- SCSI disk support: You almost certainly want to answer 'y' here. SCSI tape support: This is untested, and I haven't heard of anyone trying to use it, play things safe and answer 'n'.
- SCSI CDROM support: There is no direct use for CD roms as of yet, the problems relate to the ISOFS filesystem, more on this on the filesystems section below. Answer 'n'.
- SCSI generic support: Again, nothing uses it yet, answer 'n'.
- Probe all LUNs on each SCSI device: Answer 'n'.
- Verbose SCSI error reporting: Answer 'y', this will make error messages more informative should you encounter any problems with your scsi devices.
- Sparc ESP Scsi Driver: Answer 'y', this is the only SCSI controller supported as of yet on the Sparc.

4.2.5 Network Device Support

- Network device support: Answer 'y'.
- Dummy net driver: Useful for testing things sometimes, answer 'y'.
- SLIP: Answer 'n' it does not work yet.
- PPP: ANswer 'n', again it is non functional at this point.
- Sun LANCE support: Answer 'y', this is the ethernet card that SparcLinux supports.

4.2.6 Filesystems

- Quota support: Answer 'n', nothing uses it.
- Standard (minix) fs support: Answer 'n', not used as of yet.
- Extended fs support: Answer 'n', unused.
- Second extended fs support: This is the filesystem type currently support on harddrives under SparcLinux, so if you plan on creating filesystems and using them for your SparcLinux test setup answer 'y'. If, however, you will only be doing nfs booting answer 'n'.
- Xiafs filesystem support: Answer 'n', unsupported.
- Msdos fs support: Answer 'n', I may get it working in the future.
- /proc filesystem support: Answer 'y', you won't be able to do much of anything unless this is enabled.
- NFS filesystem support: If you plan on accessing nfs partitions from your sparclinux machine at some point, answer 'y', else answer 'n'.
- Root file system on NFS: If you plan on netbooting your sparclinux machine for testing, answer 'y', else 'n'.
- SMB filesystem: Currently unused, answer 'n'.
- ISO9660 cdrom filesystem support: Answer 'n', I plan on investigating why it does not work on the Sparc.
- OS/2 HPFS filesystem support: Answer 'n', unsupported.
- System V and Coherent filesystem support: Unsupported, answer 'n'.

4.2.7 Kernel Hacking

Kernel profiling support: Unless you really know what you are doing, answer 'n' here.

4.2.8 Building it

At this point you are done configuring your SparcLinux kernel. Type 'make dep; make clean', this will build the dependencies for the kernel build. When that finishes type 'make vmlinux' and the kernel will be built for you. It will be left under the name:

```
linux/vmlinux
```

when the build completes. This is the kernel images you will use to boot with. (note, refer to section above concerning elf2aout program written by Peter if you are doing this under Solaris)

4.3 How to Boot with an NFS root filesystem

Since a couple of versions ago, Linux supports an nfs-based root file system. The easiest way to get it working (Alan has been improving this on the past revisions of the kernel) is to type in your prom command line:

```
boot le()vmlinux nfsroot=server-ip-address:/root-nfs-dir
```

Populate your `/root-nfs-dir` with a nice root directory (`/etc`, `/dev`, `/tmp`, `/usr`, `/bin`). A `/usr` collection is also available on the FTP site (including a Linux/SPARC gcc compiler/libc + some nifty tools (setfont :-)).

You will also need to be able to get your networking information via `rarp`. I won't go into too much detail here, but you need to add a rarp entry to a rarp capable machine on your network (Linux on an Intel can do this if the kernel is configured for it). It would look something like:

```
rarp -s IP_OF_SPARC_MACHINE HARDWARE:ADDRESS:OF:SPARC

rarp -s 199.183.24.60 A8:B9:CA:DB:00:50
```

Then you need to be able to get your kernel image via `tftp`. You will need to do this on the machine answering your rarp request. Under Linux, tftp is in one of the NetKit packages. Use `man tftp` to see how to use it. Basically, you need to edit /etc/inetd.conf to enable `in.tftpd` and limit it to a certain directory. It might look something like this:

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

This is my line from a Red Hat Linux machine. This would mean that the kernel needs to go in /tmp/tftp.

Huge problem here. The tftp from the NetKit on a Linux machine will not work. It forces you to include the entire path to the file, even though it will allow you to restrict the paths. So, even if you let tftp have access to `/`, tftp wants you to use `/filename`. You can't request a file this way from the Sun PROM, so this does us no good. To get around it here, we hacked tftpd.c to do a chdir() to the directory specified on the command line. The patch is as follows:

```
--- NetKit-B-0.06/tftpd/tftpd.c.bad     Sat Feb 24 18:12:48 1996
+++ NetKit-B-0.06/tftpd/tftpd.c Sat Feb 24 18:12:51 1996
@@ -270,19 +270,24 @@
        int     fd;
        char *cp, **dirp;

-       if (*filename != '/')
-               return (EACCESS);
+       syslog(LOG_ERR, "tftpd: trying to get file: %s\n", filename);
+
+       if (*filename != '/') {
+               syslog(LOG_ERR, "tftpd: serving file from %s\n", dirs[0]);
+               chdir(dirs[0]);
+       } else {
+               for (dirp = dirs; *dirp; dirp++)
+                       if (strncmp(filename, *dirp, strlen(*dirp)) == 0)
+                               break;
+               if (*dirp==0 && dirp!=dirs)
+                       return (EACCESS);
+       }
        /*
         * prevent tricksters from getting around the directory restrictions
         */
        for (cp = filename + 1; *cp; cp++)
                if(*cp == '.' && strncmp(cp-1, "/../", 4) == 0)
                        return(EACCESS);
-       for (dirp = dirs; *dirp; dirp++)
-               if (strncmp(filename, *dirp, strlen(*dirp)) == 0)
-                       break;
-       if (*dirp==0 && dirp!=dirs)
-               return (EACCESS);
        if (stat(filename, \&stbuf) {\(<\)} 0)
                return (errno == ENOENT ? ENOTFOUND : EACCESS);
        if (mode == RRQ) {
```

Now, what do we name it? The Sun PROM will use rarp to find out it's address, then it will use the machine that answered the rarp request to tftp to and get the kernel. It will look for a file named by a standard convention. The name is the IP address of the sparc in hex with no dots followed by ".SUN4C" and in all caps. An example for a machine with IP 194.77.26.181 would be:

`C24D1AB5.SUN4C`

This is actually just "vmlinux" renamed, or it can be the sun boot loader file. If the latter is the case, you can then use the boot loader to load your kernel. Most people will probably just want to load the kernel, however.

Doing all of this will let you get your kernel booted via the network. You will still need a root directory via NFS that should be populated with a minimal set of utils to let your machine boot to a shell prompt.

5 Credits

The majority of the work is being done by:

- David S. Miller (davem@caip.rutgers.edu) - Written most of the code for the sparc port so far.
- Pete A. Zaitcev (zaitcev@ithil.mcst.ru) - Working on bootblocks
- Miguel de Icaza (miguel@nuclecu.unam.mx) - Working on network stuff

The following have also made great contributions:

- Center for Computer Aids for Industrial Productivity (CAIP), which provided machines and disk space.
- Sun Microsystems, for their cool prom debugger.
- The Free Software Foundation, for their way cool software, especially gcc and binutils.
- Linus Torvalds, for writing Linux in the first place.

6 Copyright

I'm not sure how to Copyright this yet, but this is the current one. If anyone wants to recommend a change, send it to `djb@redhat.com`.