Pocket Linux Guide

David Horton

<dhorton<AT>NOSPAM.member.fsf.org>

Revision History

Revision 3.1	2005−04−09	Revised by: DH
Minor clarifications and spelling corrections.		
Revision 3.0	2004−11−02	Revised by: DH
Changed bootloader to GRUB rather than LILO. Updated versions on all source code packages. Made minor clarifications to some shell commands and scripts.		
Revision 2.1	2004−02−18	Revised by: DH
Corrected typos. Changed resource site hosting to SourceForge. Added appendix B to include the GNU Free Documentation License as part of this document.		
Revision 2.0	2003−11−08	Revised by: DH
Updated to use GNU coreutils in place of fileutils, sh−utils and textutils. Updated version numbers on many source code packages. Introduced Freshmeat as a resource for finding source code. Changed /etc/mtab to a real file rather than using a symlink to /proc/mounts. Corrected local_fs script errors. Updated email address.		
Revision 1.2	2003−05−31	Revised by: DH
Corrected errors in "strip −o library" commands.		
Revision 1.1	2003−05−21	Revised by: DH
Bug fixes, typo corrections and improved XML markup.		
Revision 1.0	2003−02−17	Revised by: DH
Initial Release, reviewed by LDP.		

The Pocket Linux Guide is for anyone interested in learning the techniques of building a GNU/Linux system from source code. The guide is structured as a project that builds a small diskette−based GNU/Linux system called Pocket Linux. Each chapter explores a small piece of the overall system explaining how it works, why it is needed and how to build it. After completing the Pocket Linux project, readers should possess an enhanced knowledge of what makes GNU/Linux systems work as well as the confidence to explore larger, more complex source−code−only projects.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Legal Information

1. Copyright and License

---

2. Disclaimer

This documentation is provided as−is with no warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Use the concepts, examples and information at your own risk. The author(s) do not take any responsibility for damages that may arise from the use of this document.

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

Introduction

1. About Pocket Linux

The Pocket Linux Guide demonstrates how to build a small console−based GNU/Linux system using only source code and a couple of diskettes. It is intended for Linux users who would like to gain a deeper understanding about how their system works beneath the shroud of distribution specific features and tools.

2. Prerequisite Skills

This guide is intended for intermediate to advanced Linux users. It is not intentionally obscure, but certain assumptions about the readers skill level are made. Success with this guide depends in part on being able to perform the following tasks:

- Use basic shell commands
- Reference man and info pages
- Build a custom Linux kernel
- Compile source code using make and related tools

3. Project Format

The Pocket Linux Guide takes a hands−on approach to learning. The guide is written with each chapter building a piece of an overall project. Chapters are further broken into sections of Analysis, Design, Construction and Implementation. This format is derived from Rapid Application Development (RAD) methodology. Without going into detail about design methodologies, the sections may be summed up as follows.

- The Analysis section gives a high−level overview of what is to be accomplished in each chapter. It will introduce the tasks that need to be completed and why they are important to the overall system.
- The Design section defines the source code packages, files and configuration necessary to address the requirements set forth in the Analysis section. Much of the theory of why certain system files exist and what their purpose is can be found here.
- The Construction section is where all the hands−on action takes place. This section goes into detail about building source code and configuring the system files.
- The Implementation section will test the proper operation of the project at the end of each chapter. Often there are a few shell commands to perform and samples of expected screen outputs are given.

Readers interested in learning more about RAD may want to consult a textbook covering systems analysis and design or visit the following University of California, Davis website on the subject: http://sysdev.ucdavis.edu/WEBADM/document/rad−stages.htm.

4. Help & Support

Readers are encouraged to visit the Pocket Linux Resource Site at http://pocket−linux.sourceforge.net/. The resource site is home to:

- Information about the Pocket Linux mailing list.

- A web−based troubleshooting forum where readers can ask questions and give tips to others.
- A collection of diskette images for various chapters.
- Additional projects that may be of interest to Pocket Linux Guide readers.

5. Feedback

For technical questions about Pocket Linux please use the mailing list or the troubleshooting forum on the resource site. General comments and suggestions may be sent to the mailing list or emailed to the author directly.

Chapter 1. Project Initiation

1.1. A Brief History of GNU/Linux

In the early 90's GNU/Linux systems consisted of little more than a beta−quality Linux kernel and a small collection of software ported from the GNU project. It was a true hacker's operating system. There were no CD−ROM's or GUI installation tools; everything had to be compiled and configured by the end user. Being a Linux Expert meant knowing your system inside and out.

Toward the middle of the decade several GNU/Linux distributions began appearing. One of the first was Slackware in 1993 and since then there have been many others. Even though there are many "flavors" of Linux today, the main purpose of the distribution remains the same. The distribution automates many of the tasks involved in GNU/Linux installation and configuration taking the burden off of the system administrator. Being a Linux Expert now means knowing which button to click in the GUI administration tool.

Recently there has been a yearn for a return to the "good old days" of Linux when men were men, sysadmins were hardcore geeks and everything was compiled from source code. A notable indication of this movement was the publication of the Linux−From−Scratch−HOWTO version 1.0 by Gerard Beekmans in 1999. Being a Linux Expert once again means knowing how to do it yourself.

For more historical information, see Ragib Hasan's "History of Linux" at http://netfiles.uiuc.edu/rhasan/linux

1.2. The Goal of Pocket Linux

The purpose of Pocket Linux is to support and encourage people who wish to explore Linux by building a GNU/Linux system from nothing but source code. Pocket Linux is not intended to be a full featured system, but rather to give the reader a taste of what is involved in building an operating system from source code. After completing the Pocket Linux system the reader should have enough knowledge to confidently build almost any project using only source code. Given this direction we can put a few constraints on the project.

- The main focus should be learning. The project should not just describe how to do something, it should also describe why it should be done.
- The required time commitment should be minimal and manageable.
- The project should not require any investment in additional hardware or reconfiguration of existing hardware to set up a lab environment.
- Readers should not need to know any programming languages in order to complete the project.
- To remain true to the spirit of GNU/Linux, all software used in the project should be covered under the GNU/GPL or another, similarly liberal, open−source license.

1.3. Working Within The Constraints

The Pocket Linux project gets its name from the fact that the bulk of the project fits onto two diskettes making it possible to carry the entire, working system around in one's pocket. This has the advantage of not requiring any additional hardware since any PC can be booted from the diskettes without disrupting any OS that exists on the hard drive. Using diskettes also partially addresses the aspect of time commitment, because the project size and complexity is necessarily limited by the 1.44 Megabyte size of the installation media.

To further reduce the time commitment, the Pocket Linux project is divided into several phases, each one chapter in length. Each phase builds only a small piece of the overall project, but at the same time the conclusion of each chapter results in a self−contained, working system. This step−by−step approach should allow readers to pace themselves and not feel the need to rush to see results.

Chapters are further subdivided into four sections. The first two sections, analysis and design, focus on the theory of what is to be accomplished in each phase and why. The last two sections, construction and implementation, detail the steps needed to do the actual building. Advanced readers, who may be familiar with the theories laid out in a particular chapter are encouraged to gloss over the analysis and design sections in the interest of time. The separation of theory from hands−on exercises should allow readers of all skill levels to complete the project without feeling either completely lost or mired in too much detail.

Finally, the Pocket Linux project will strive to use GNU/GPL software when possible and other open−source licensed software when there is no GNU/GPL alternative. Also, Pocket Linux will never require any programming more complex than a BASH shell script.

Chapter 2. A Simple Prototype

2.1. Analysis

Since this is the first phase of the project it will be kept very simple. The goal here is not to create the ultimate GNU/Linux system on the first try. Instead, we will be building a very minimal, working system to be used as a building block in subsequent phases of the project. Keeping this in mind, we can list a few goals for phase one.

- Keep it simple to avoid stressing out.
- Build something that works for instant gratification.
- Make something that it is useful in later phases of the project.

2.2. Design

2.2.1. Simplification

Take a moment to skim through the Bootdisk−HOWTO or the From−PowerUp−to−BASH−Prompt−HOWTO. These HOWTO documents can be found online at http://www.tldp.org/docs.html#howto. Both documents offer an excellent view of what it takes to get a GNU/Linux system up and running. There is also a lot of information to digest. Remember that one of our goals is, "keep it simple to avoid stressing out," so we want to ignore everything but the absolutely critical pieces of a boot / root diskset.

Basically it boils down to the following required items:

- A boot loader
- The Linux kernel
- A shell
- Some `/dev` files

We don't even need an init daemon. The kernel can be told to run the shell directly by passing it an option through the boot loader.

For easy construction we will build a two−disk boot / root set rather than trying to get everything onto a single diskette. The boot loader and kernel will go on the boot disk and the shell will reside on the root disk.

2.2.2. Boot Disk

For the boot disk we simply need to install the GRUB bootloader and a Linux kernel. We will need to use a kernel that does not require modules for the hardware we need to access. Mainly, it should have compiled−in support for the floppy drive, ram disk, second extended filesystem, proc filesystem, ELF binaries, and a text−based console. If such a kernel is not available, it will need to be built from source code. Kwan Lowe's Kernel Rebuild Guide is a good reference for this task, however we can ignore the sections that deal with modules and the initial ramdisk.

2.2.3. Root Disk

For the root disk we will need a floppy that has been prepared with a filesystem. We will also need a BASH shell that is statically–linked so we can avoid the additional complexities of shared libraries. The **configure** program in the BASH source code recognizes the `--enable-static-link` option for this feature. We will also be using the `--enable-minimal-config` option to keep the BASH binary down to a manageable size. Additional requirements for the root disk are a `/dev` directory and a device file for the console. The `console` device is required for BASH to be able to communicate with the keyboard and video display.

2.2.4. CPU Compatibility

There is one other, less obvious requirement to keep in mind and that is CPU compatibility. Each generation of CPU features a more complex architecture than its predecessor. Late generation chips have additional registers and instructions when compared to an older 486 or 386. So a kernel optimized for a new, fast 6x86 machine will not run on an older box. (See the `README` file in the Linux kernel source code for details.) A BASH shell built for a 6x86 will probably not run on an older processor either. To avoid this problem, we can choose the 386 as a lowest common denominator CPU and build all the code for that architecture.

2.3. Construction

In this section, we will be building the actual boot disk and root disk floppies. Lines preceded by `bash#` indicate a shell command and lines starting with `grub>` indicate a command typed within the grub shell.

2.3.1. Prepare the boot disk media

Insert a blank diskette labeled "boot disk".

It may be necessary to erase the "blank" diskette if it comes factory pre–formatted for another, non–Linux operating system. This can be done using the command **dd if=/dev/zero of=/dev/fd0 bs=1k count=1440**

```
bash# mke2fs -m0 /dev/fd0
bash# mount /dev/fd0 /mnt
```

2.3.2. Build the GRUB bootloader

Get the GRUB source code from ftp://alpha.gnu.org/gnu/grub/ and unpack it into the `/usr/src` directory.

Configure and build the GRUB source code for an i386 processor by using the following commands:

```
bash# cd /usr/src/grub-0.95
bash# export CC="gcc -mcpu=i386"
bash# ./configure --host=i386-pc-linux-gnu --without-curses
bash# make
```

2.3.3. Copy the bootloader files to diskette

Normally, after compiling source code, one would use the command **make install** to copy the finished files to their proper destinations in the filesystem. However, using **make install** does not work well with small media like the floppy disks we are using. The problem is that there are many files in a package besides the actual binaries that get the job done. For example, there are often man or info pages that provide documentation. These extra files can take up more space than we can spare on the diskette. We can work around this limitation by copying essential files manually rather than using **make install**.

For GRUB to boot we will need to copy the stage1 and stage2 bootloader files to the `/boot/grub` directory on the boot floppy.

```
bash# mkdir -p /mnt/boot/grub
bash# cp /usr/src/grub-0.95/stage1/stage1 /mnt/boot/grub
bash# cp /usr/src/grub-0.95/stage2/stage2 /mnt/boot/grub
```

2.3.4. Finish bootloader installation

Once the bootloader's files are copied to the boot disk we can enter the grub shell to finish the installation.

```
bash# /usr/src/grub-0.95/grub/grub
grub> root (fd0)
grub> setup (fd0)
grub> quit
```

2.3.5. Build the Linux kernel

The steps for building the kernel were tested using Linux kernel version 2.4.26 and should work any 2.4.x or 2.6.x kernel. The latest version of the kernel source code may be downloaded from http://www.kernel.org/ or one of its mirrors.

The instructions below are very brief and are intended for someone who has previous experience building custom kernels. A more detailed explanation of the kernel building process can be found in the Kernel Rebuild Guide by Kwan Lowe.

```
bash# cd /usr/src/linux
bash# make menuconfig
```

Be sure to configure support for the following:

- 386 processor
- Console on virtual terminal (2.4.x kernels only)
- ELF binaries
- Floppy disk
- proc filesystem
- RAM disk with a default size of 4096K
- Second extended (ext2) filesystem
- VGA console

```
bash# make dep
bash# make clean
bash# make bzImage
```

2.3.6. Copy the kernel to diskette

```
bash# cp /usr/src/linux/arch/i386/boot/bzImage /mnt/boot/vmlinuz
```

2.3.7. Unmount the boot disk

```
bash# cd /
bash# umount /mnt
```

2.3.8. Prepare the root disk media

Insert a blank diskette labeled "root disk".

```
bash# mke2fs -m0 /dev/fd0
bash# mount /dev/fd0 /mnt
```

2.3.9. Build BASH

Get the bash–3.0 source code package from ftp://ftp.gnu.org/gnu/bash/ and untar it into the `/usr/src` directory.

Build BASH for an i386 CPU with the following commands:

```
bash# cd /usr/src/bash-3.0
bash# export CC="gcc -mcpu=i386"
bash# ./configure --enable-static-link \
  --enable-minimal-config --host=i386-pc-linux-gnu
bash# make
bash# strip bash
```

2.3.10. Copy BASH to the root disk

```
bash# mkdir /mnt/bin
bash# cp bash /mnt/bin/bash
bash# ln -s bash /mnt/bin/sh
```

2.3.11. Create device files that BASH needs

```
bash# mkdir /mnt/dev
bash# mknod /mnt/dev/console c 5 1
```

2.3.12. Unmount the root disk

```
bash# cd /
bash# umount /mnt
```

2.4. Implementation

2.4.1. System startup

Follow these steps to boot the system:

- Restart the PC with the boot disk in the floppy drive.
- When the `grub>` prompt appears, type **`kernel (fd0)/boot/vmlinuz init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1`** and press **Enter**.
- After the kernel loads, type **`boot`** and press **Enter**.
- Insert the root disk when prompted.

If all goes well the screen should look something like the example shown below.

```
GNU GRUB version 0.95

grub> kernel (fd0)/boot/vmlinuz init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1
   [Linux-bzImage, setup=0xc00, size=0xce29b]

grub> boot

Linux version 2.4.26
..
.. [various kernel messages]
..
VFS: Insert root floppy disk to be loaded into RAM disk and press ENTER
RAMDISK: ext2 filesystem found at block 0
RAMDISK: Loading 1440 blocks [1 disk] into ram disk... done.
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 178k freed
# _
```

2.4.2. Testing what works

Try out a few of BASH's built–in commands to see if things are working properly.

```
bash# echo "Hello World"
bash# cd /
bash# pwd
bash# echo *
```

2.4.3. Noting what does not work

Try out a few other familiar commands.

```
bash# ls /var
bash# mkdir /var/tmp
```

Notice that only commands internal to BASH actually work and that external commands like **ls** and **mkdir** do not work at all. This shortcoming is something that can be addressed in a future phase of the project. For now we should just enjoy the fact that our prototype boot / root diskset works and that it was not all that hard to build.

2.4.4. System shutdown

Remove the diskette from fd0 and restart the system using **CTRL−ALT−DELETE**.

Chapter 3. Saving Space

3.1. Analysis

One of the drawbacks in the prototype phase of the project was that the diskset was not all that useful. The only commands that worked were the ones built into the BASH shell. We could improve our root disk by installing commands like **cat**, **ls**, **mv**, **rm** and so on. Unfortunately, we are short on space. The current root disk has no shared libraries so each utility would have to be statically−linked just like the BASH shell. A lot of big binaries together with a static shell will rapidly exceed the tiny 1.44M of available disk space. So our main goal in this phase should be to maximize space savings on the root disk and pave the way for expanded functionality in the next phase.

3.2. Design

Take another look at the Bootdisk−HOWTO and notice how many utilities can be squeezed onto a 1.44M floppy. There are three things that make this possible. One is the use of shared libraries. The second is stripped binaries. And the third is the use of a compressed filesystem. We can use all of these techniques to save space on our root disk.

3.2.1. Shared Libraries

First, in order to use shared libraries we will need to rebuild the BASH shell. This time we will configure it without using the `--enable-static-link` option. Once BASH is rebuilt we need to figure out which libraries it is linked with and be sure to include them on the root disk. The **ldd** command makes this job easy. By typing **ldd bash** on the command−line we can see a list of all the shared libraries that BASH uses. As long as all these libraries are copied to the root disk, the new BASH build should work fine.

3.2.2. Stripped Binaries

Next, we should strip any binaries that get copied to the root disk. The manpage for **strip** does not give much description of what it does other than to say, "strip discards all symbols from the object files." It seems like removing pieces of a binary would render it useless, but this is not the case. The reason it works is because a large number of these discarded symbols are used for debugging. While debugging symbols are very helpful to programmers working to improve the code, they do not do much for the average end−user other than take up more disk space. And since space is at a premium, we should definitely remove as many symbols as possible from BASH and any other binaries before we copy over them to the ramdisk.

The process of stripping files to save space also works with shared library files. But when stripping libraries it is important to use the `--strip-unneeded` option so as not to break them. Using `--strip-unneeded` shrinks the file size, but leaves the symbols needed for relocation intact which is something that shared libraries need to function properly.

3.2.3. Compressed Root Filesystem

Finally, we can tackle the problem of how to build a compressed root filesystem. The Bootdisk−HOWTO suggests three ways of constructing a compressed root filesystem using either a ramdisk, a spare hard drive partition or a loopback device. This project will concentrate on using the ramdisk approach. It seems logical that if the root filesystem is going to be run from a ramdisk, it may as well be built on a ramdisk. All we have to do is create a second extended filesystem on a ramdisk device, mount it and copy files to it. Once the filesystem is populated with all the files that the root disk needs, we simply unmount it, compress it and write it out to floppy.

For this to work, we need to make sure the system used for building has ramdisk support. If ramdisk is not available it is also possible to use a loopback device. See the Bootdisk–HOWTO for more information on using loopback devices.

3.3. Construction

This section is written using ramdisk seven (`/dev/ram7`) to build the root image. There is nothing particularly special about ramdisk seven and it is possible to use any of the other available ramdisks provided they are not already in use.

3.3.1. Create a ramdisk

```
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7 4096
bash# mount /dev/ram7 /mnt
```

3.3.2. Rebuild the BASH shell

```
bash# cd /usr/src/bash-3.0
bash# make distclean
bash# export CC="gcc -mcpu=i386"
bash# ./configure --enable-minimal-config --host=i386-pc-linux-gnu
bash# make
bash# strip bash
```

3.3.3. Determine which libraries are required

```
bash# ldd bash
```

View the output from the **ldd** command. It should look similar to the example below.

```
bash# ldd bash
  libdl.so.2 => /lib/libdl.so.2 (0x4001d000)
  libc.so.6 => /lib/libc.so.6 (0x40020000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Some systems may have a slightly different library set up. For example, you may see `libc.so.6 => /lib/tls/libc.so.6` rather than `libc.so.6 => /lib/libc.so.6` as shown in the example. If your **ldd** output does not match the example then use the path given by your **ldd** command when completing the next step.

3.3.4. Copy BASH and its libraries to the ramdisk

```
bash# mkdir /mnt/bin
bash# cp bash /mnt/bin
bash# ln -s bash /mnt/bin/sh
```

```
bash# mkdir /mnt/lib
bash# strip --strip-unneeded -o /mnt/lib/libdl.so.2 /lib/libdl.so.2
bash# strip --strip-unneeded -o /mnt/lib/libc.so.6 /lib/libc.so.6
bash# strip --strip-unneeded -o /mnt/lib/ld-linux.so.2 /lib/ld-linux.so.2
bash# chmod +x /mnt/lib/ld-linux.so.2
```

Using **strip -o** might seem an odd way to copy library files from the development system to the ramdisk. What it does is strip the symbols while the file is in transit from the source location to the destination. This has the effect of stripping symbols from the library on the ramdisk without altering the libraries on the development system. Unfortunately file permissions are lost when copying libraries this way which is why the **chmod +x** command is then used to set the execute flag for the rootdisk's dynamic loader.

3.3.5. Create a console device

```
bash# mkdir /mnt/dev
bash# mknod /mnt/dev/console c 5 1
```

3.3.6. Compress the ramdisk image

```
bash# cd /
bash# umount /dev/ram7
bash# dd if=/dev/ram7 of=~/phase2-image bs=1k count=4096
bash# gzip -9 ~/phase2-image
```

3.3.7. Copy the compressed image to diskette

Insert the floppy labeled "root disk" into drive fd0.

```
bash# dd if=~/phase2-image.gz of=/dev/fd0 bs=1k
```

3.4. Implementation

Successful implementation of this phase is probably the most difficult part of the Pocket Linux Guide. If you need help getting things to work please visit the Pocket Linux Guide Resource Site to browse the troubleshooting forum and subscribe to the mailing list.

3.4.1. System startup

Follow these steps to boot:

- Restart the PC using the boot disk from the previous chapter.
- At the `grub>` prompt, type **`kernel (fd0)/boot/vmlinuz init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1`** and press **Enter**.
- Type **`boot`** at the `grub>` prompt and press **Enter**.
- Insert the new, compressed root disk when prompted.

The screen output should be similar to the following example:

```
GNU GRUB version 0.95
```

```
grub> kernel (fd0)/boot/vmlinuz init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1
   [Linux-bzImage, setup=0xc00, size=0xce29b]

grub> boot

Linux version 2.4.26
..
.. [various kernel messages]
..
VFS: Insert root floppy disk to be loaded into RAM disk and press ENTER
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 178k freed
# _
```

3.4.2. Verify results

If the implementation was successful, this new root disk should behave exactly like the root disk from the previous chapter. The key difference is that this compressed root disk has much more room to grow and we will put this extra space to good use in the next phase of the project.

3.4.3. System shutdown

Remove the diskette from fd0 and restart the system using **CTRL−ALT−DELETE**.

Chapter 4. Some Basic Utilities

4.1. Analysis

In the previous chapter it might seem like we did not accomplish very much. A lot of energy was expended redesigning the root disk, but the functionality is basically the same as in the initial prototype phase. The root disk still does not do very much. But we did make significant improvements when it comes to space savings. In this chapter we will put that extra space to good use and start cramming the root disk with as many utilities as it can hold.

The first two root disks we built only had shell built–in commands like **echo** and **pwd**. This time it would be nice to have some of the commonly used external commands like **cat**, **ls**, **mkdir**, **rm** and such on the root disk. Keeping this in mind we can define the goals for this phase as follows:

- Retain all of the functionality from the previous root disk.
- Add some of the commonly used external commands.

4.2. Design

4.2.1. Determining Required Commands

The first question that might come to mind is, "How do we know which commands are needed?" It is possible to just start with **cat** and **ls** then install other commands as we discover a need for them. But this is terribly inefficient. We need a plan or a blueprint to work from. For this we can turn to the Filesystem Hierarchy Standard (FHS) available from http://www.pathname.com/fhs/. The FHS dictates which commands should be present on a Linux system and where they should be placed in the directory structure.

4.2.2. Locating Source Code

The next logical question is, "Now that we know what we need, where do we get the source code?" One way to find the answer to this question is to check the manpages. We can either search the manpages included with one of the popular GNU/Linux distributions or use one of the manpage search engines listed at http://www.tldp.org/docs.html#man. One thing that should tip us off as to where to find the source code for a particular command is the email address listed for reporting bugs. For example the cat manpage lists bug–textutils@gnu.org. From this email address we can deduce that **cat** is part of the textutils package from GNU.

4.2.3. Leveraging FHS

So let's look at the FHS requirements for the `/bin` directory. The first few commands in the list are **cat**, **chgrp**, **chmod**, **chown** and **cp**. We already know that **cat** is part of GNU's textutils. Using the next few commands as keywords in a manpage search we discover that we need GNU's fileutils package for **chmod**, **chgrp**, **chown** and **cp**. In fact quite a few of the commands in `/bin` come from GNU's fileutils. The **date** command also comes from a GNU package called sh–utils. So a good way to tackle the problem of finding source code might be to group the commands together by package as shown below.

- The BASH shell –– **echo**, **false**, **pwd**, **sh**, **true**
- GNU textutils –– **cat**

* GNU fileutils −− **chgrp**, **chmod**, **chown**, **cp**, **dd**, **df**, **ln**, **ls**, **mkdir**, **mknod**, **mv**, **rm**, **rmdir**, **sync**
* GNU sh−utils −− **date**, **hostname**, **stty**, **su**, **uname**

These four packages do not contain all of the commands in the `/bin` directory, but they do represent of over 70% of them. That should be enough to accomplish our goal of adding some of the commonly used external commands. We can worry about the other commands in later phases of the project.

4.2.4. Downloading Source Code

To fetch the source code we simply need to connect to GNU's FTP site and navigate to the appropriate package directory.

When we get to the directory for textutils there are several versions available. There is also a note informing us that the package has been renamed to coreutils. The same message about coreutils appears in the fileutils and sh−utils directories as well. So instead of downloading three separate packages we can get everything in one convenient bundle in the coreutils directory.

4.3. Construction

Rather than copying files directly to the ramdisk, we can make things easier by setting up a staging area. The staging area will give us room to work without worrying about the space constraints of the ramdisk. It will also provide a way to save our work and make it easier to enhance the rootdisk in later phases of the project.

The staging procedure will work like this:

1. Create a directory structure as defined in the FHS.
2. Copy in the files from phase 2's root disk.
3. Build the new package from source code.
4. Install files into the correct FHS directories.
5. Strip the binaries to save space.
6. Check library dependencies.
7. Copy to the whole directory structure to the ramdisk.
8. Compress the ramdisk and write it out to floppy.

4.3.1. Create a staging area

```
bash# mkdir ~/staging
bash# cd ~/staging
bash# mkdir bin boot dev etc home lib mnt opt proc root sbin tmp usr var
bash# mkdir var/log var/run
```

4.3.2. Copy contents of phase 2 rootdisk

```
bash# dd if=~/phase2-image.gz | gunzip -c > /dev/ram7
bash# mount /dev/ram7 /mnt
bash# cp -dpR /mnt/* ~/staging
bash# umount /dev/ram7
bash# rmdir ~/staging/lost+found
```

4.3.3. Install binaries from GNU coreutils

Download a recent version of coreutils from ftp://ftp.gnu.org/gnu/coreutils/

```
bash# cd /usr/src/coreutils-5.2.1
bash# export CC="gcc -mcpu=i386"
bash# ./configure --host=i386-pc-linux-gnu
bash# make
bash# cd src
bash# cp cat chgrp chmod chown cp date dd df ~/staging/bin
bash# cp hostname ln ls mkdir mkfifo mknod ~/staging/bin
bash# cp mv rm rmdir stty su sync uname ~/staging/bin
```

4.3.4. Copy additional libraries

Check library requirements by using **ldd** on some of the new binaries.

```
bash# ldd ~/staging/bin/cat
bash# ldd ~/staging/bin/ls
bash# ldd ~/staging/bin/su
bash# ls ~/staging/lib
```

Note the differences in the required libraries, as shown by the **ldd** command, and the libraries present in the staging area, as shown by the **ls** command, then copy any missing libraries to the staging area.

```
bash# cp /lib/librt.so.1 ~/staging/lib
bash# cp /lib/libpthread.so.0 ~/staging/lib
bash# cp /lib/libcrypt.so.1 ~/staging/lib
```

4.3.5. Strip binaries and libraries

```
bash# strip ~/staging/bin/*
bash# strip --strip-unneeded ~/staging/lib/*
```

4.3.6. Create a compressed root disk image

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7 4096
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/staging/* /mnt
bash# umount /dev/ram7
bash# dd if=/dev/ram7 of=~/phase3-image bs=1k count=4096
bash# gzip -9 ~/phase3-image
```

The process for creating the compressed root disk image will change very little throughout the remaining chapters. Writing a small script to handle this function can be a great time saver.

4.3.7. Write the root disk image to floppy

Insert the diskette labeled "root disk" into drive fd0.

```
bash# dd if=~/phase3-image.gz of=/dev/fd0 bs=1k
```

4.4. Implementation

We will need to have a read−write filesystem in order for some of the commands to work. The kernel's normal behavior is to mount root as read−only, but we can change this using a kernel option. By passing the kernel the *rw* option before *init=/bin/sh* we will get a read−write root filesystem.

4.4.1. System startup

Follow these steps to get the system running.

- Boot the PC from using the GRUB boot disk.
- At the grub> prompt, type **kernel (fd0)/boot/vmlinuz rw init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1**.
- Verify that you remembered to add the *rw* parameter and press **Enter**.
- Type boot and press **Enter**.
- Insert the recently created root disk when prompted.

The terminal display should look similar to the example below.

```
GNU GRUB version 0.95

grub> kernel (fd0)/boot/vmlinuz rw init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1
   [Linux-bzImage, setup=0xc00, size=0xce29b]

grub> boot

Linux version 2.4.26
..
.. [various kernel messages]
..
VFS: Insert root floppy disk to be loaded into RAM disk and press ENTER
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem) read-write.
Freeing unused kernel memory: 178k freed
# _
```

4.4.2. Testing new commands

Now that the system is up and running, try using some of the new commands.

```
bash# uname -a
bash# ls /etc
bash# echo "PocketLinux" > /etc/hostname
bash# hostname $(cat /etc/hostname)
bash# uname -n
bash# mkdir /home/stuff
bash# cd /home/stuff
```

If everything goes well the commands like **cat**, **ls** and **hostname** should work now. Even **mkdir** should work since the root filesystem is mounted read−write. Of course since we are using a ramdisk, any changes will be lost once the PC is reset.

4.4.3. System shutdown

Remove the diskette from fd0 and restart the system using **CTRL−ALT−DELETE**.

Chapter 5. Checking and Mounting Disks

5.1. Analysis

In the previous chapter we added many new commands by installing coreutils and as a result the root disk has a lot more functionality. But there are still a few things lacking. One thing that really stands out is that there was no way to mount disks. In order to get a read−write root filesystem we had to resort to passing the *rw* kernel parameter at the `grub>` prompt. This is fine for an emergency situation, but a normal system boot process should do things differently.

Most GNU/Linux distributions take several steps to mount filesystems. Watching the boot process or digging into the startup scripts on one of the popular Linux distributions reveals the following sequence of events:

1. The kernel automatically mounts the root filesystem as read−only.
2. All local filesystems are checked for errors.
3. If filesystems are clean, root is remounted as read−write.
4. The rest of the local filesystems are mounted.
5. Network filesystems are mounted.

So far our Pocket Linux system can do step one and that is it. If we want to have a professional looking boot / root diskset we will have to do better than one out of five. In this phase of the project we will work on steps two and three. Steps four and five can wait. Since this is a diskette−based system, there really are no other filesystems to mount besides root.

Taking into account all of the above information, the goals for this phase are defined as follows:

- A way to check filesystem integrity.
- The ability to mount filesystems.
- A script to automate checking and mounting of local filesystems.

5.2. Design

5.2.1. Determining necessary utilities.

We can use the Filesystem Hierarchy Standard (FHS) document to help find the names of utilities we need and where they reside in the directory structure. The FHS `/sbin` directory lists **fsck** and something called **fsck.*** for checking filesystems. Since we are using a Second Extended (ext2) filesystem the **fsck.*** becomes **fsck.ext2** for our purposes. Mounting filesystems is done using the commands **mount** and **umount** in the `/bin` directory. However, the name of a script to automatically mount local filesystems cannot be found. On most systems this type of script is in the `/etc` directory, but while FHS does list requirements for `/etc`, it does not currently make recommendations for startup scripts. Several GNU/Linux distributions use `/etc/init.d` as the place to hold startup scripts so we will put our filesystem mounting script there.

5.2.2. Finding source code

In the previous chapter we used manpages to help us find source code. In this chapter we will use a tool called the Linux Software Map (LSM). LSM is a database of GNU/Linux software that tracks such things as package name, author, names of binaries that make up the package and download sites. Using an LSM search engine we can locate packages using command names as keywords.

If we search Ibiblio's Linux Software Map (LSM) at http://www.ibiblio.org/pub/Linux/ for the keyword "fsck" we get a large number of matches. Since we are using a Second Extended filesystem, called ext2 for short, we can refine the search using "ext2" as a keyword. Supplying both keywords to the LSM search engine comes up with a package called e2fsprogs. Looking at the LSM entry for e2fsprogs we find out that this package contains the utilities **e2fsck**, **mke2fs**, **dumpe2fs**, **fsck** and more. We also find out that the LSM entry for e2fsprogs has not been updated for a while. There is almost certainly a newer version out there somewhere. Another good Internet resource for source code is SourceForge at http://sourceforge.net/. Using the keyword "e2fsprogs" in the SourceForge search engine results in a much newer version of e2fsprogs.

Finding **fsck** was quite an adventure, but now we can move on to finding **mount** and **umount**. A search on LSM comes up with a number of matches, but most of them point to various versions of a package called util−linux. All we have to do is scroll through and pick the most recent release. The LSM entry for util−linux lists a lot of utilities besides just mount and umount. We should definitely scan through the list to see if any of the other util−linux commands show up in the FHS requirements for `/bin` and `/sbin`.

Below is a list of packages we have gathered so far and the utilities that match up with FHS.

- e2fsprogs −− **fsck**, **fsck.ext2** (**e2fsck**), **mkfs.ext2** (**mke2fs**)
- util−linux −− **dmesg**, **getty** (**agetty**), **kill**, **login**, **mount**, **swapon**, **umount**

5.2.3. Automating fsck and mount

Now that we have **fsck** and **mount** commands we need to come up with a shell script to automate checking and mounting the local filesystems. An easy way to do this would be to write a short, two line script that calls **fsck** and then **mount**. But, what if the filesystems are not clean? The system should definitely not try to mount a corrupted filesystem. Therefore we need to devise a way of determining the status of the filesystems before mounting them. The manpage for **fsck** gives some insight into how this can be accomplished using return codes. Basically, if **fsck** returns a code of zero or one it means the filesystem is okay and a return code of two or greater means some kind of manual intervention is needed. A simple if−then statement could evaluate the **fsck** return code to determine whether or not the filesystem should be mounted. For help on writing shell scripts we can turn to the BASH(1) manpage and the Advanced−BASH−Scripting−Guide. Both references are freely available from the Linux Documentation Project web site at http://www.tldp.org/.

5.2.4. File dependencies

The last thing to do is to figure out if any other files besides the binaries are needed. We learned about using **ldd** to check for library dependencies in the last phase of the project and we will use it to check the utilities in this phase too. There are also some other files that **fsck** and **mount** will need and the fsck(8) and mount(8) manpages give some insight into what those files are. There is `/etc/fstab` that lists devices and their mount points, `/etc/mtab` that keeps track of what is mounted, and a number of `/dev` files that represent the various disks. We will need to include all of these to have everything work right.

5.2.4.1. /etc/fstab

The `/etc/fstab` file is just a simple text file that can be created with any editor. We will need an entry for the root filesystem and for the proc filesystem. Information about the format of this file can be found in the fstab(5) manpage or by looking at the `/etc/fstab` file on any of the popular GNU/Linux distributions.

5.2.4.2. /etc/mtab

The /etc/mtab file presents a unique challenge, because it does not contain static information like fstab. The mtab file tracks mounted filesystems and therefore its contents change from time to time. We are particularly interested in the state of mtab when the system first starts up, before any filesystems are mounted. At this point /etc/mtab should be empty so we will need to configure a startup script to create an empty /etc/mtab before any filesystems are mounted. But it is not possible to create any files in the /etc directory because / is read−only at startup. This creates a paradox. We cannot create an empty mtab, because the / filesystem is not mounted as writable and we should not mount any filesystems until we have created an empty mtab. In order to sidestep this problem we need to do the following:

1. Remount / as read−write, but use the −n option so that **mount** does not attempt to write an entry to /etc/mtab which is read−only at this point.
2. Create an empty /etc/mtab file now that the filesystem is writable.
3. Remount / as read−write again, this time using the −f option so that an entry is written into /etc/mtab, but / is not actually mounted a second time.

5.2.4.3. Device files

The only thing left to do is to create device files. We will need /dev/ram0, because that is where the root filesystem is located. We also need /dev/fd0 to mount other floppy disks and /dev/null for use by some of the system commands.

5.3. Construction

5.3.1. Install utilities from e2fsprogs

Download the e2fsprogs source code package from http://sourceforge.net/projects/e2fsprogs/

```
bash# cd /usr/src/e2fsprogs-1.35
bash# export CC="gcc -mcpu=i386"
bash# ./configure --host=i386-pc-linux-gnu
bash# make
bash# cd e2fsck
bash# cp e2fsck.shared ~/staging/sbin/e2fsck
bash# ln -s e2fsck ~/staging/sbin/fsck.ext2
bash# cd ../misc
bash# cp fsck mke2fs ~/staging/sbin
bash# ln -s mke2fs ~/staging/sbin/mkfs.ext2
```

5.3.2. Install utilities from util−linux

Get the latest util−linux source from ftp://ftp.win.tue.nl/pub/linux−local/utils/util−linux/

```
bash# cd /usr/src/util-linux-2.12h
```

Use a text editor to make the following changes to MCONFIG:

- Change "CPU=$(shell uname −m)" to "CPU=i386"
- Change "HAVE_SHADOW=yes" to "HAVE_SHADOW=no"

```
bash# ./configure
```

```
bash# make
bash# cp disk-utils/mkfs ~/staging/sbin
bash# cp fdisk/fdisk ~/staging/sbin
bash# cp login-utils/agetty ~/staging/sbin
bash# ln -s agetty ~/staging/sbin/getty
bash# cp login-utils/login ~/staging/bin
bash# cp misc-utils/kill ~/staging/bin
bash# cp mount/mount ~/staging/bin
bash# cp mount/umount ~/staging/bin
bash# cp mount/swapon ~/staging/sbin
bash# cp sys-utils/dmesg ~/staging/bin
```

5.3.3. Check library requirements

```
bash# ldd ~/staging/bin/* | more
bash# ldd ~/staging/sbin/* | more
bash# ls ~/staging/lib
```

All of the dependencies revealed by the **ldd** command are for libraries already present in the staging area so there is no need to copy anything new.

5.3.4. Strip binaries to save space

```
bash# strip ~/staging/bin/*
bash# strip ~/staging/sbin/*
```

5.3.5. Create additional device files

```
bash# mknod ~/staging/dev/ram0 b 1 0
bash# mknod ~/staging/dev/fd0 b 2 0
bash# mknod ~/staging/dev/null c 1 3
```

5.3.6. Create the fstab and mtab files

```
bash# cd ~/staging/etc
```

Use an editor like vi, emacs or pico to create the following file and save it as `~/staging/etc/fstab`.

```
proc        /proc   proc   defaults   0   0
/dev/ram0   /       ext2   defaults   1   1
```

Create an empty mtab file.

```
bash# echo -n >mtab
```

5.3.7. Write a script to check and mount local filesystems

Use an editor to create the following shell script and save it as `~/staging/etc/init.d/local_fs`:

```
#!/bin/sh
#
# local_fs - check and mount local filesystems
#
PATH=/sbin:/bin ; export PATH
```

```
fsck -ATCp
if [ $? -gt 1 ]; then
  echo "Filesystem errors still exist!  Manual intervention required."
  /bin/sh
else
  echo "Remounting / as read-write."
  mount -n -o remount,rw /
  echo -n >/etc/mtab
  mount -f -o remount,rw /
  echo "Mounting local filesystems."
  mount -a -t nonfs,nosmbfs
fi
#
# end of local_fs
```

Set execute permissions on the script.

```
bash# chmod +x local_fs
```

5.3.8. Create a compressed root disk image

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7 4096
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/staging/* /mnt
bash# umount /dev/ram7
bash# dd if=/dev/ram7 of=~/phase4-image bs=1k count=4096
bash# gzip -9 ~/phase4-image
```

5.3.9. Write the root disk image to floppy

Insert the diskette labeled "root disk" into drive fd0.

```
bash# dd if=~/phase4-image.gz of=/dev/fd0 bs=1k
```

5.4. Implementation

5.4.1. System startup

Start the system using the following procedure:

- Boot the PC using the floppy labeled "boot disk".
- At the `grub>` prompt, type the usual kernel and boot commands, but without the *rw* parameter this time. In other words, type **`kernel (fd0)/boot/vmlinuz init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1`**, press **Enter** then type **`boot`** and press **Enter**.
- Put in the recently created root disk when prompted.

The output should resemble the example below:

```
GNU GRUB version 0.95

grub> kernel (fd0)/boot/vmlinuz init=/bin/sh root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1
```

```
   [Linux-bzImage, setup=0xc00, size=0xce29b]

grub> boot

Linux version 2.4.26
..
.. [various kernel messages]
..
VFS: Insert root floppy disk to be loaded into RAM disk and press ENTER
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 178k freed
# _
```

5.4.2. Test the local_fs script

Run the script by typing the following commands at the shell prompt:

```
bash# PATH=/sbin:/bin:/etc/init.d ; export PATH
bash# cat /etc/mtab
bash# local_fs
bash# cat /etc/mtab
bash# df
```

If everything is working properly, then the screen output should look something like the example below.

```
bash# PATH=/sbin:/bin:/etc/init.d ; export PATH
bash# cat /etc/mtab
bash# local_fs
/dev/ram0: clean 74/1024 files 3178/4096 blocks
Remounting / as read-write.
Mounting local filesystems.
bash# cat /etc/mtab
/dev/ram0 / ext2 rw 0 0
proc /proc proc rw 0 0
bash# df
Filesystem      1k-blocks       Used Available Use% Mounted on
/dev/ram0       3963            3045 918        77% /
```

5.4.3. Create and mount additional filesystems

Procure a blank floppy disk and label it as "home". Remove the root disk floppy and insert the "home" diskette. Type the following commands:

```
bash# mkfs -t ext2 /dev/fd0
bash# fsck /dev/fd0
bash# mount /dev/fd0 /home
bash# mkdir /home/floyd
bash# cd /home/floyd
bash# echo "Goodbye cruel world." > goodbye.txt
bash# cat goodbye.txt
```

5.4.4. System shutdown

```
bash# cd /
bash# umount /home
```

Remove the diskette from fd0 and restart the system using **CTRL−ALT−DELETE**.

Chapter 6. Automating Startup & Shutdown

6.1. Analysis

The root disk from the last chapter is looking pretty good. It has about seventy percent of the commands that the Filesystem Hierarchy Standard (FHS) document requires for the root filesystem. Plus it has commands for checking and mounting filesystems. But even with all of this the root disk is far from perfect. The list below outlines three things that could use some improvement if the Pocket Linux system is to stand up next to the more professional looking distributions.

1. The system currently requires the kernel parameters to be typed at the `grub>` prompt in order to start properly. On any other GNU/Linux system this is only done in an emergency situation when the system is corrupted.
2. Checking and mounting the root filesystem has to be done manually by running a script at a shell prompt. On most modern operating systems this function is handled automatically as part of the system start−up process.
3. Using **CTRL−ALT−DELETE** for system shutdown is not very graceful. Filesystems should be unmounted and cached information should be flushed prior to shutdown. Again, this is something that most operating systems handle automatically.

Taking the above list into consideration, the goals for this phase are defined as follows:

- Kernel loads without manual intervention.
- Automated system start−up sequence.
- Graceful shutdown capability.

6.2. Design

6.2.1. Determining necessary utilities

Loading the kernel without manually typing parameters is easy to do if we read the grub info page. According to the section entitled "configuration" all of the commands used for booting can be put in a file called `menu.lst` and placed in the `/boot/grub` directory.

Be sure to type the `menu.lst` filename correctly with a lowercase L after the dot and not a number one.

To automate system start−up we will need an init daemon. We know this because the Bootdisk−HOWTO and From−Powerup−To−BASH−Prompt−HOWTO both make mention of **init** as the first program to start after the kernel loads. The latter HOWTO also goes into some detail about the `/etc/inittab` file and the organization of startup scripts. This could be helpful since FHS, the blueprint we have used so far, makes no recommendation for init scripts.

We will also need to find the **shutdown** command to fulfill the second goal of graceful shutdown capability.

6.2.2. Obtaining source code

Searching the Linux Software Map on Ibiblio for the keyword "init" gives a large number of results. From reading the From−Powerup−To−BASH−Prompt−HOWTO however, we know that most Linux systems use a

System V style init daemon. Narrowing the search with the additional key phrase of "System V" gives much better results. The sysvinit package contains **init**, **shutdown**, **halt** and **reboot** which is everything we need. The version listed in the LSM entry looks to be pretty old, but there is a primary−site URL that will probably lead to the latest version.

6.2.3. Checking dependencies

The manpage for **init** mentions a FIFO called `/dev/initctl` that is required for **init** to communicate with other programs in the sysvinit package. We will have to create this file for **init** to function properly.

6.2.4. Designing a simple GRUB configuration file.

Using a GRUB configuration file is slightly more complex than specifying the bootloader commands manually. There are directives for features like menus, default selections and timeouts that need to be specified in the configuration file as well as the familiar kernel loading command. The info page for GRUB gives much of the necessary information. We may also be able to use the GRUB configuration file on the development system as a template. However, there is some inconsistency between vendors as to the name and location of the file. Regardless of what the path is on the development system it should be `/boot/grub/menu.lst` on the Pocket Linux System.

6.2.5. Outlining start−up scripts

Many of the popular GNU/Linux distributions use System V style init scripts. Since we are using a "sysvinit" daemon it makes sense to use System V style scripts as well. The following documents all touch upon the System V style init scripts in some way and will serve as references when building the scripts for this project:

- The Debian Policy Manual −− available online at http://www.debian.org/doc/debian−policy.
- The Linux Standard Base specification −− downloadable in many formats from http://www.linuxbase.org/spec/index.shtml.
- Essential System Administration, 3rd Edition by Aeleen Frisch −− available at libraries, bookstores or directly from O'Reilly Publishing at http://www.oreilly.com/.

After glancing at one or two of the above references we should have a pretty good idea of how the System V style system initialization process works. We should also know what it takes to create System V style init scripts for the Pocket Linux project. Below is a brief list of what needs to be done:

- Create an `inittab` file to call an `rc` script with a numerical argument giving the runlevel.
- Write an `rc` script that uses the runlevel argument to execute the appropriate "K" and "S" scripts.
- Modify the previously built `local_fs` script to take *`start`* and *`stop`* arguments.
- Create new scripts for `shutdown` and `reboot`.
- Set up `/etc/rcN.d` directories and links to scripts in `/etc/init.d`.

As always, the BASH(1) manpage and the Advanced BASH Scripting Guide are very helpful for writing and understanding shell scripts.

6.3. Construction

There is a lot of typing to do in this section because of all of the start–up scripts that need to be created. Using a mouse to copy the text from this guide and paste it into a text editor can be a great time saving tool.

6.3.1. Create a GRUB configuration file

Insert and mount the floppy labeled "boot disk".

```
bash# mount /dev/fd0 /mnt
bash# cd /mnt/boot/grub
```

Use your favorite text editor to create the following file and save it as /mnt/boot/grub/menu.lst:

```
default 0
timeout 3
title Pocket Linux Boot Disk
kernel (fd0)/boot/vmlinuz root=/dev/fd0 load_ramdisk=1 prompt_ramdisk=1
```

6.3.2. Install sysvinit utilities

Download the latest sysvinit source from ftp://ftp.cistron.nl/pub/people/miquels/software/

```
bash# cd /usr/src/sysvinit-2.85/src
bash# make CC="gcc -mcpu=i386"
bash# cp halt init shutdown ~/staging/sbin
bash# ln -s halt ~/staging/sbin/reboot
bash# ln -s init ~/staging/sbin/telinit
bash# mknod ~/staging/dev/initctl p
```

In the interest of speed we are skipping the steps for checking libraries and stripping binaries. The library requirements for sysvinit are very basic and the Makefile is configured to automatically strip the binaries.

6.3.3. Create /etc/inittab file

Use a text editor to create the following file and save it as `~/staging/etc/inittab`

```
# /etc/inittab - init daemon configuration file
#
# Default runlevel
id:1:initdefault:
#
# System initialization
si:S:sysinit:/etc/init.d/rc S
#
# Runlevel scripts
r0:0:wait:/etc/init.d/rc 0
r1:1:respawn:/bin/sh
r2:2:wait:/etc/init.d/rc 2
r3:3:wait:/etc/init.d/rc 3
r4:4:wait:/etc/init.d/rc 4
r5:5:wait:/etc/init.d/rc 5
r6:6:wait:/etc/init.d/rc 6
```

```
#
# end of /etc/inittab
```

6.3.4. Create /etc/init.d/rc script

Use a text editor to create the following file and save it as `~/staging/etc/init.d/rc`

```
#!/bin/sh
#
# /etc/init.d/rc - runlevel change script
#
PATH=/sbin:/bin
SCRIPT_DIR="/etc/rc$1.d"
#
# Check that the rcN.d directory really exists.
if [ -d $SCRIPT_DIR ]; then
#
# Execute the kill scripts first.
  for SCRIPT in $SCRIPT_DIR/K*; do
    if [ -x $SCRIPT ]; then
      $SCRIPT stop;
    fi;
  done;
#
# Do the Start scripts last.
  for SCRIPT in $SCRIPT_DIR/S*; do
    if [ -x $SCRIPT ]; then
      $SCRIPT start;
    fi;
  done;
fi
#
# end of /etc/init.d/rc
```

Make the file executable.

```
bash# chmod +x ~/staging/etc/init.d/rc
```

6.3.5. Modify /etc/init.d/local_fs script

A case statement is added to allow the script to either mount or unmount local filesystems depending on the command−line argument given. The original script is contained inside the "start" portion of the case statement. The "stop" portion is new.

```
#!/bin/sh
#
# local_fs - check and mount local filesystems
#
PATH=/sbin:/bin ; export PATH

case $1 in

start)
  echo "Checking local filesystem integrity."
  fsck -ATCp
  if [ $? -gt 1 ]; then
    echo "Filesystem errors still exist!  Manual intervention required."
    /bin/sh
```

```
  else
    echo "Remounting / as read-write."
    mount -n -o remount,rw /
    echo -n > /etc/mtab
    mount -f -o remount,rw /
    echo "Mounting local filesystems."
    mount -a -t nonfs,smbfs
  fi
;;

stop)
  echo "Unmounting local filesystems."
  umount -a -r
;;

*)
  echo "usage: $0 start|stop";
;;

esac
#
# end of local_fs
```

6.3.6. Create a hostname script

Use a text editor to create the following script and save it as `~/staging/etc/init.d/hostname`

```
#!/bin/sh
#
# hostname - set the system name to the name stored in /etc/hostname
#
PATH=/sbin:/bin ; export PATH

echo "Setting hostname."
if [ -f /etc/hostname ]; then
  hostname $(cat /etc/hostname)
else
  hostname gnu-linux
fi
#
# end of hostname
```

6.3.7. Create halt & reboot scripts

Use a text editor to create `~/staging/etc/init.d/halt` as shown below.

```
#!/bin/sh
#
# halt - halt the system
#
PATH=/sbin:/bin ; export PATH

echo "Initiating system halt."
halt
#
# end of /etc/init.d/halt
```

Create the following script and save it as `~/staging/etc/init.d/reboot`

```
#!/bin/sh
#
# reboot - reboot the system
#
PATH=/sbin:/bin ; export PATH

echo "Initiating system reboot."
reboot
#
# end of /etc/init.d/reboot
```

Flag all script files as executable.

```
bash# chmod +x ~/staging/etc/init.d/*
```

6.3.8. Create rcN.d directories and links

```
bash# cd ~/staging/etc
bash# mkdir rc0.d rc1.d rc2.d rc3.d rc4.d rc5.d rc6.d rcS.d
bash# cd ~/staging/etc/rcS.d
bash# ln -s ../init.d/local_fs S20local_fs
bash# ln -s ../init.d/hostname S30hostname
bash# cd ~/staging/etc/rc0.d
bash# ln -s ../init.d/local_fs K10local_fs
bash# ln -s ../init.d/halt K90halt
bash# cd ~/staging/etc/rc6.d
bash# ln -s ../init.d/local_fs K10local_fs
bash# ln -s ../init.d/reboot K90reboot
```

6.3.9. Create the root disk image

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7 4096
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/staging/* /mnt
bash# umount /dev/ram7
bash# dd if=/dev/ram7 of=~/phase5-image bs=1k
bash# gzip -9 ~/phase5-image
```

6.3.10. Copy the image to diskette

Insert the diskette labeled "root disk" into drive fd0.

```
bash# dd if=~/phase5-image.gz of=/dev/fd0 bs=1k
```

6.4. Implementation

6.4.1. System Startup

Boot the PC using the floppy labeled "boot disk". Place the recently created root disk in fd0 when prompted. The output should resemble the example below:

```
GNU GRUB version 0.95
```

```
Uncompressing Linux... Ok, booting kernel.
..
.. [various kernel messages]
..
VFS: Insert root floppy to be loaded into RAM disk and press ENTER
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 178k freed
Checking local filesystem integrity.
/dev/ram0: clean 105/1024 files 2842/4096 blocks
Remounting / as read-write.
Mounting local filesystems.
Setting the hostname.
INIT: Entering runlevel: 1
# _
```

6.4.2. Verify success of startup scripts

Use the **mount** command to check that local filesystems are mounted as read−write. The output should look like the example below.

```
bash# mount
/dev/root on / type ext2 (rw)
proc on /proc type proc (rw)
```

Check the hostname.

```
bash# uname −n
gnu−linux
```

6.4.3. System shutdown

Bring the system down gracefully with the **shutdown** command.

```
bash# shutdown −h now
```

We should see the following output from **init** and the shutdown scripts:

```
INIT: Switching to runlevel: 0
INIT: Sending processes the TERM signal
Terminated
INIT: Sending processes the KILL signal
Unmounting local filesystems.
Initiating system halt.
System halted.
```

Chapter 7. Enabling Multiple Users

7.1. Analysis

Up to now the system has been operating in single−user mode. There is no login process and anyone who boots the system goes straight into a shell with root privileges. Obviously, this is not the normal operating mode for most GNU/Linux distributions. Most systems feature multi−user capability where many users can access the system simultaneously with different privilege levels. These multi−user systems also support virtual consoles so that the keyboard and video display can be multiplexed between several terminal sessions. So in this phase we would like to add the following enhancements to the system:

- Enable multi−user capability.
- Create multiple, virtual consoles.

7.2. Design

7.2.1. The login process

The From−Powerup−To−BASH−Prompt−HOWTO does a good job of outlining the steps in the login process. Basically it works like this.

1. The **init** daemon starts a **getty** process on the terminal.
2. The **getty** program displays the contents of `/etc/issue` and prompts for a user name.
3. When the user name is entered, control is handed off to the **login** program.
4. The **login** program asks for a password and verifies the credentials using `/etc/passwd`, `/etc/group` and possibly `/etc/shadow`.
5. If everything is okay the user's shell is started.

7.2.2. Obtaining source code

The **getty** and **login** programs were already installed as part of the util−linux package so there is no need to download any new source code.

7.2.3. Creating support files

7.2.3.1. Device nodes

Details about virtual console device files can be found in the Linux kernel source code file called `devices.txt` in the `Documentation` directory. We will need to create `tty1` through `tty6` for each of the virtual consoles as well as `tty0` and `tty` to represent the current virtual console.

7.2.3.2. /etc/issue

The `/etc/issue` file is pretty easy to construct. It can contain any text we want displayed on the screen prior to the login prompt. It could be something friendly like "Welcome to Pocket Linux", something menacing like "Authorized users only!" or something informational like "Connected to tty1 at 9600bps". The agetty(8) manpage explains how to display information like tty line and baud rate using escape codes.

7.2.3.3. /etc/passwd

The format of `/etc/passwd` can be obtained by reading the passwd(5) manpage. We can easily create a user account by adding a line like "root::0:0:superuser:/root:/bin/sh" to the file.

Maintaining passwords will be somewhat challenging because of the system being loaded into ramdisk. Any changes to `/etc/passwd` will be lost when the system is shutdown. So to make things easy, we will create all users with null passwords.

7.2.3.4. /etc/group

The structure of `/etc/group` is available from the group(5) manpage. A line of "root::0:root" would define a group called "root" with no password, a group id of zero and the user root assigned to it as the only member.

7.2.3.5. Conventions

User and group names and id's are generally not chosen at random. Most Linux systems have very similar looking `/etc/passwd` and `/etc/group` files. Definitions for commonly used user id and group id assignments may be found in one of several places:

- The `/etc/passwd` and `/etc/group` files on any popular GNU/Linux distribution.
- The Debian Policy Manual −− available online at http://www.debian.org/doc/debian−policy.
- The Linux Standard Base specification −− downloadable in many formats from http://www.linuxbase.org/spec/index.shtml.
- Essential System Administration, 3rd Edition by Aeleen Frisch −− available at libraries, bookstores or directly from O'Reilly Publishing at http://www.oreilly.com/.

7.2.4. Dependencies

Running **ldd** on the `login` program from util−linux will reveal that it is linked to the libraries `libcrypt.so.1`, `libc.so.6` and `ld-linux.so.2`. In addition to these libraries there is another, unseen dependency on `libnss_files.so.2` and the configuration file `/etc/nsswitch.conf`.

The name service switch library `libnss_files.so.2` and `nsswitch.conf` are required for `libc.so.6`, and consequently the `login` program, to access the `/etc/passwd` file. Without libnss and its configuration file, all logins will mysteriously fail. More information about glibc's use of the name service switch libraries can be found at http://www.gnu.org/software/libc/manual/html_node/Name−Service−Switch.html.

7.2.5. Assigning ownership and permissions

Previously, with the single user system, there was no need to worry about permissions when installing directories, files and device nodes. The shell was effectively operating as root, so everything was accessible. Things become more complex with the addition of multiple user capability. Now we need to make sure that every user has access to what they need and at the same time gets blocked from what they do not need.

A good guideline for assigning ownership and permissions would be to give the minimum level of access required. Take the `/bin` directory as an example. The Filesystem Hierarchy (FHS) document says, "`/bin` contains commands that may be used by both the system administrator and by users". From that statement we can infer that `/bin` should have read and execute permission for everyone. On the other hand, the `/boot` directory contains files for the boot loader. Chances are good that regular users will not need to access anything in the `/boot` directory. So the minimum level of access would be read permission for the root user and other administrators who are members of the root group. Normal users would have no permissions assigned on the `/boot` directory.

Most of the time we can assign similar permissions to all the commands in a directory, but there are some programs that prove to be exceptions to the rule. The **su** command is a good example. Other commands in the /bin directory have a minimum requirement of read and execute, but the **su** command needs to be setuid root in order to run correctly. Since it is a setuid binary, it might not be a good idea to allow just anyone to run it. Ownership of 0:0 (root user, root group) and permissions of rwsr−x−−− (octal 4750) would be a good fit for **su**.

The same logic can be applied to other directories and files in the root filesystem using the following steps:

1. Assign ownership to the root user and root group.
2. Set the most restrictive permissions possible.
3. Adjust ownership and permissions on an "as needed" basis.

7.3. Construction

7.3.1. Verify presence of getty and login

```
bash# ls ~/staging/sbin/getty
bash# ls ~/staging/bin/login
```

7.3.2. Modify inittab for multi−user mode

Modify `~/staging/etc/inittab` by changing the default runlevel and adding **getty** entries as shown below.

```
# /etc/inittab - init daemon configuration file
#
# Default runlevel
id:2:initdefault:
#
# System initialization
si:S:sysinit:/etc/init.d/rc S
#
# Runlevel scripts
r0:0:wait:/etc/init.d/rc 0
r1:1:respawn:/bin/sh
r2:2:wait:/etc/init.d/rc 2
r3:3:wait:/etc/init.d/rc 3
r4:4:wait:/etc/init.d/rc 4
r5:5:wait:/etc/init.d/rc 5
r6:6:wait:/etc/init.d/rc 6
#
# Spawn virtual terminals
1:235:respawn:/sbin/getty 38400 tty1 linux
2:235:respawn:/sbin/getty 38400 tty2 linux
3:235:respawn:/sbin/getty 38400 tty3 linux
```

```
4:235:respawn:/sbin/getty 38400 tty4 linux
5:235:respawn:/sbin/getty 38400 tty5 linux
6:2345:respawn:/sbin/getty 38400 tty6 linux
#
# end of /etc/inittab
```

7.3.3. Create tty devices

```
bash# cd ~/staging/dev
bash# mknod ~/staging/dev/tty0 c 4 0
bash# mknod ~/staging/dev/tty1 c 4 1
bash# mknod ~/staging/dev/tty2 c 4 2
bash# mknod ~/staging/dev/tty3 c 4 3
bash# mknod ~/staging/dev/tty4 c 4 4
bash# mknod ~/staging/dev/tty5 c 4 5
bash# mknod ~/staging/dev/tty6 c 4 6
bash# mknod ~/staging/dev/tty c 5 0
```

7.3.4. Create support files in /etc

7.3.4.1. /etc/issue

Create the file `~/staging/etc/issue` using the example below or design a customized message.

```
Connected to \l at \b bps.
```

Be sure that "\l" is a lowercase letter L and not the number one.

7.3.4.2. /etc/passwd

Use a text editor to create a minimal passwd file conforming to the Linux Standards Base (LSB) document. Save the file as `~/staging/etc/passwd`

```
root::0:0:Super User:/root:/bin/sh
bin:x:1:1:Legacy UID:/bin:/bin/false
daemon:x:2:2:Legacy UID:/sbin:/bin/false
```

7.3.4.3. /etc/group

Use a text editor to create an LSB conforming group file and save it as `~/staging/etc/group`

```
root::0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
```

7.3.4.4. /etc/nsswitch.conf

Create the following file and save it as `~/staging/etc/nsswitch.conf`

```
passwd: files
group:  files
```

7.3.5. Copy required libraries

```
bash# cp /lib/libnss_files.so.2 ~/staging/lib
bash# strip --strip-unneeded ~/staging/lib/*
```

7.3.6. Set directory and file permissions

Set minimal privileges on all files and directories under `~/staging`. Everything is owned by the root user and the root group. Permissions are read−write for the owner and read−only for the group. Exceptions to the blanket permissions are handled case by case.

```
bash# cd ~/staging
bash# chown -R 0:0 ~/staging/*
bash# chmod -R 640 ~/staging/*
```

Set execute permission on all directories. (Note the capital "X")

```
bash# chmod -R +X ~/staging/*
```

Files in `/bin` are read and execute for all, but `su` is an exception.

```
bash# chmod 755 ~/staging/bin/*
bash# chmod 4750 ~/staging/bin/su
```

Files in `/dev` have various permissions. Disk devices should be accessible to administrators only. Other files like `/dev/null` should have full privileges granted to everyone.

```
bash# chmod 660 ~/staging/dev/fd0 dev/ram0
bash# chmod 666 ~/staging/dev/null
bash# chmod 622 ~/staging/dev/console
bash# chmod 600 ~/staging/dev/initctl
bash# chmod 622 ~/staging/dev/tty
bash# chmod 622 ~/staging/dev/tty?
```

The `passwd` and `group` files must be world readable.

```
bash# chmod 644 ~/staging/etc/passwd
bash# chmod 644 ~/staging/etc/group
```

The scripts in `/etc/init.d` are read and execute for administrators.

```
bash# chmod 750 ~/staging/etc/init.d/*
```

Libraries need read and execute permissions for everyone.

```
bash# chmod 755 ~/staging/lib/*
```

Only root should have access to the `/root` directory.

```
bash# chmod 700 ~/staging/root
```

Make files in `/sbin` read and execute for administrators.

```
bash# chmod 750 ~/staging/sbin/*
```

Temp should be read−write for all with the sticky bit set.

```
bash# chmod 1777 ~/staging/tmp
```

7.3.7. Create the root disk image

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7 4096
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/staging/* /mnt
bash# umount /dev/ram7
bash# dd if=/dev/ram7 of=~/phase6-image bs=1k count=4096
bash# gzip -9 ~/phase6-image
```

7.3.8. Copy the image to diskette

Insert the diskette labeled "root disk" into drive fd0.

```
bash# dd if=~/phase6-image.gz of=/dev/fd0 bs=1k
```

7.4. Implementation

7.4.1. System Startup

If everything goes well, the virtual console display should look similar to the following example:

```
Connected to tty1 at 38400 bps.
gnu-linux login:
```

7.4.2. Add a new user to the system

Log in as root.

Create a new, unprivileged user and new group by appending a line to the `/etc/passwd` and `/etc/group` files, respectively. Be sure to use a double greater−than (>>) to avoid accidentally overwriting the files.

```
bash# echo "floyd::501:500:User:/home/floyd:/bin/sh" >>/etc/passwd
bash# echo "users::500:" >>/etc/group
bash# mkdir /home/floyd
bash# chown floyd.users /home/floyd
bash# chmod 700 /home/floyd
```

7.4.3. Test the new user's ability to use the system

Switch to virtual terminal tty2 by pressing **ALT+F2**.

Log in as floyd.

Try the following commands and verify that they work.

```
bash$ pwd
bash$ ls -l /
bash$ cat /etc/passwd
```

Try the following commands and verify that they do not work.

```
bash$ ls /root
bash$ /sbin/shutdown -h now
bash$ su -
```

7.4.4. System shutdown

Switch back to tty1 where root is logged in.

```
bash# shutdown -h now
```

Chapter 8. Filling in the Gaps

8.1. Analysis

The root disk has come a long way since its humble beginnings as a statically−linked shell. It now shares many features with the popular, ready−made distributions. For example it has:

* Several common utilities like **cat**, **ls** and so on.
* Startup scripts that automatically check and mount filesystems.
* Graceful shutdown capability.
* Support for multiple users and virtual terminals.

As a final test, we can put the root disk up against the Filesystem Hierarchy Standard (FHS) requirements for the root filesystem. (We will ignore anything in the `/usr` hierarchy because of space constraints.) Compared to FHS requirement, the only files missing are a few commands in the `/bin` directory. Specifically, the root disk lacks the following commands:

* **more**
* **ps**
* **sed**

In addition to the required commands, it might be nice to include the "ed" editor listed as an option by the FHS. It is not as robust as vi or emacs, but it works and it should fit onto the tiny root filesystem.

So in order to finish up this phase of the project, we need to accomplish the following goals:

* Add the **more**, **ps** and **sed** commands.
* Install the optional **ed** editor.

8.2. Design

8.2.1. more

There is a **more** command that comes with util−linux, but it will not work for this project. The reason is because of library dependencies and space constraints. The util−linux supplied **more** needs either the libncurses or libtermcap to work and there just is not enough space on the root disk floppy to fit everything in. So, in order to have a **more** command we will have to get creative.

The **more** command is used to display a file page by page. It's a little like having a **cat** command that pauses every twenty−five lines. The basic logic is outlined below.

* Read one line of the file.
* Display the line on the screen.
* If 25 lines have been displayed, pause.
* Loop and do it again.

Of course there are some details left out like what to do if the screen dimensions are not what we anticipated, but overall it is a fair representation of what **more** does. Given this simple program logic, it should not be hard to put together a short shell script that emulates the basic functionality of **more**. The BASH(1) manpage and Adv−BASH−Scripting−Guide will serve as references.

8.2.2. More device files

The **more** script will need access to device files that are not on the root disk yet. Specifically **more** needs to have `stdin`, `stdout` and `stderr`, but while we are at it we should check for any other missing `/dev` files. The Linux Standard Base requires `null`, `zero` and `tty` to be present in the `/dev` directory. Files for `null` and `tty` already exist from previous phases of the project, but we still need `/dev/zero`. We can refer to `devices.txt` in the Linux source code `Documentation` directory for major and minor numbers.

8.2.3. ps, sed & ed

These three packages can be found by using the Internet resources we have used before plus one new site. The "sed" and "ed" packages can be found at the same place we found BASH, on the GNU FTP server. The procps package shows up in an Ibiblio LSM search, but it is an old version. In order to find the latest version we can go to the Freshmeat website at http://freshmeat.net and search for "procps" in projects.

Both "sed" and "ed" packages feature GNU's familiar **configure** script and are therefore very easy to build. There is no **configure** script for "procps" but this does not make things too difficult. We can just read the package's `README` file to find out about how to set various configuration options. We can use one of these options to avoid the complexity of using and installing libproc. Setting *`SHARED=0`* makes `libproc` an integrated part of **ps** rather than a separate, shared library.

8.3. Construction

8.3.1. Write a "more" script

Create the following script with a text editor and save it as `~/staging/bin/more.sh`

```
#!/bin/sh
#
# more.sh - emulates the basic functions of the "more" binary without
#           requiring ncurses or termcap libraries.
#
# Assume input is coming from STDIN unless a valid file is given as
# a command-line argument.
if [ -f $1 ]; then
  INPUT="$1"
else
  INPUT="/dev/stdin"
fi
#
# Set IFS to newline only. See BASH(1) manpage for details on IFS.
IFS=$'\n'
#
# If terminal dimensions are not already set as shell variables, take
# a guess of 80x25.
if [ "$COLUMNS" = "" ]; then
  let COLUMNS=80;
fi
if [ "$LINES" = "" ]; then
  let LINES=25;
```

```
fi
#
# Initialize line counter variable
let LINE_COUNTER=$LINES
#
# Read the input file one line at a time and display on STDOUT until
# the page fills up. Display "Press <Enter>" message on STDERR and wait
# for keypress from STDERR.  Continue until the end of the input file.
# Any input line greater than $COLUMNS characters in length is wrapped
# and counts as multiple lines.
#
while read -n $COLUMNS LINE_BUFFER; do
  echo "$LINE_BUFFER"
  let LINE_COUNTER=$LINE_COUNTER-1
  if [ $LINE_COUNTER -le 1 ]; then
    echo "Press <ENTER> for next page or <CTRL>+C to quit.">/dev/stderr
    read</dev/stderr
    let LINE_COUNTER=$LINES
  fi
done<$INPUT
#
# end of more.sh
```

Create a symbolic link for `more`

```
bash# ln -s more.sh ~/staging/bin/more
```

8.3.2. Create additional device files

```
bash# ln -s /proc/self/fd ~/staging/dev/fd
bash# ln -s fd/0 ~/staging/dev/stdin
bash# ln -s fd/1 ~/staging/dev/stdout
bash# ln -s fd/2 ~/staging/dev/stderr
bash# mknod -m644 ~/staging/dev/zero c 1 5
```

8.3.3. Install ps

Get the latest procps source package from http://procps.sourceforge.net/

```
bash# cd /usr/src/procps-3.2.3
bash# make SHARED=0 CC="gcc -mcpu=i386"
bash# cd ps
bash# cp ps ~/staging/bin
```

8.3.4. Install sed

Download GNU's sed from ftp://ftp.gnu.org/gnu/sed/

```
bash# cd /usr/src/sed-4.1.2
bash# export CC="gcc -mcpu=i386"
bash# ./configure --host=i386-pc-linux-gnu
bash# make
bash# cd sed
bash# cp sed ~/staging/bin
```

8.3.5. Install ed

The ed package also comes from GNU at ftp://ftp.gnu.org/gnu/ed/

```
bash# cd /usr/src/ed-0.2
bash# ./configure --host=i386-pc-linux-gnu
bash# make
bash# cp ed ~/staging/bin
```

8.3.6. Strip binaries to save space

```
bash# strip ~/staging/bin/*
```

8.3.7. Ensure proper permissions

```
bash# chown 0:0 ~/staging/bin/*
bash# chmod -R 755 ~/staging/bin
bash# chmod 4750 ~/staging/bin/su
```

8.3.8. Create the root disk image

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7 4096
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/staging/* /mnt
bash# umount /dev/ram7
bash# dd if=/dev/ram7 of=~/phase7-image bs=1k
bash# gzip -9 ~/phase7-image
```

8.3.9. Copy the image to diskette

Insert the diskette labeled "root disk" into drive fd0.

```
bash# dd if=~/phase7-image.gz of=/dev/fd0 bs=1k
```

8.4. Implementation

8.4.1. System startup

Boot from the diskset in the usual way and log in as root.

8.4.2. Test the "more" script

Display kernel messages by piping the output of **dmesg** to **more**.

```
bash# dmesg | more
```

Examine the `local_fs` script by using **more** with a command–line argument.

```
bash# more /etc/init.d/local_fs
```

8.4.3. Use ps to show running processes

Display processes for the user currently logged in.

```
bash# ps
```

Display all available information about all running processes.

```
bash# ps -ef
```

8.4.4. Run a simple sed script

Use **sed** to display an alternate version of `/etc/passwd`.

```
bash# sed -e "s/Legacy/Old School/" /etc/passwd
```

Verify that sed did not make the changes permanent.

```
bash# cat /etc/passwd
```

8.4.5. Test the "ed" editor

Use **ed** to change properties on the "daemon" user.

```
bash# ed -p*
ed* r /etc/passwd
ed* %p
ed* /daemon/s/Legacy/Old School/
ed* %p
ed* w
ed* q
```

Verify that the changes are permanent (at least until the system is restarted.)

```
bash# cat /etc/passwd
```

8.4.6. System shutdown

Bring the system down gracefully with the **shutdown** command.

Chapter 9. Project Wrap Up

9.1. Celebrating Accomplishments

As the Pocket Linux Project draws to a close we should take a moment to celebrate all of our accomplishments. Some of the highlights are listed below:

- We have built a system, from source code only, that fully implements all of the commands described in the Filesystem Hierarchy Standard requirements for a root filesystem.
- We have learned how to use Internet resources to locate and download the source code needed to build a GNU/Linux system.
- We have written basic system startup and shutdown scripts and configured them to execute in the proper runlevels.
- We have included support for multiple users on virtual consoles and implemented permissions on system files.
- But most importantly, we have learned some good design techniques and project management skills that will enable us to tackle any future projects with ease and confidence.

9.2. Planning Next Steps

The Pocket Linux system is nearly overflowing, so there really is no more room to expand the current root diskette to support any additional commands and features. This leaves us with a few choices of where to go next. We can:

- Find a way to expand the current system just enough to host a small application. (For more information about hosting applications with Pocket Linux, see Appendix A)
- Remove multi−user capability and some of the less often used commands from the root disk, replacing them with utilities like tar and gzip that would be useful for a rescue/restore diskset.
- Use the techniques we have learned to design and build an entire GNU/Linux system and install it on a more spacious hard disk partition. (For more infomation about building a larger system, check out the GNU/Linux System Architect Toolkit at: http://architect.sourceforge.net/.)

Which ever path is chosen, we can move forward confidently, armed with the knowledge we need to be successful in our endeavors.

Appendix A. Hosting Applications

A.1. Analysis

An operating system by itself is not much fun. What makes an OS great is the applications that can be run on top of it. Unfortunately, Pocket Linux currently does not have much room for anything other than system programs. Still, it would be nice to expand the system just enough to host some cool applications. Obviously a full−blown X−Windows GUI is out of the question, but running a small console based program should be within our reach.

Rather than doing a typical "hello world" program as an example, application hosting will be demonstrated using a console based audio player called mp3blaster. Building mp3blaster offers more technical challenge than "hello world" and the finished product should be a lot more fun. However, it should not be construed that a console−based jukebox is the only application for Pocket Linux. On the contrary, after completing this phase the reader should have the knowledge and tools to build almost any console−based program he or she desires.

So what will it take to turn a pocket−sized GNU/Linux system into a pocket−sized mp3 player? A few things are listed below.

- Add support for audio hardware.
- Create space for the mp3blaster program.
- Provide a convenient way to access audio files.

A.2. Design

A.2.1. Support for audio hardware

There is a vast proliferation of audio hardware on the market and each sound card has its own particular configuration. For details on how to set up a particular sound card we can turn to the Sound−HOWTO available from The Linux Documentation Project. In a broader sense, however, we can treat a sound card like any other piece of new hardware. To add new hardware to a GNU/Linux system we will need configure the kernel to recognize it and configure `/dev` files on the root disk to access it.

A.2.1.1. Kernel support for audio

In order to support sound cards, a new kernel will have to be built. It is very important that audio hardware support be configured as built−in, because Pocket Linux is not set up to handle kernel modules.

A.2.1.2. Root disk support for audio

Searching `devices.txt` for the keyword "sound" will list quite a few possible audio devices, but usually only `/dev/dsp` and `/dev/mixer` are required to get sound from a PC. These two files control the digital audio output and mixer controls, respectively.

A.2.2. Creating space for the program

Probably the easiest way to create more space for the mp3blaster program is to mount an additional storage device. There are several choices for mount points. So far `/usr`, `/home` and `/opt` are all empty directories and any one of them could be used to mount a floppy, CD−ROM or additional compressed ramdisk image. The `/usr` directory is a logical choice for a place to put an application, but what about the choice of media? Mp3blaster and its required libraries are too big to fit on a 1.44M floppy and burning a CD−ROM seems like a lot of work for one little program. So given these constraints, the best choice would be to put the program on a compressed floppy.

A.2.2.1. Mounting additional compressed floppies

Mounting CDs and uncompressed diskettes is easy, but what about loading compressed images from floppy into ramdisk? It will have to be done manually, because automatic mounting of compressed floppies only works for the root diskette. And using **mount /dev/fd0** will not work because there is no filesystem on the diskette, there are only the contents of a gzip file. The actual filesystem is contained inside the gzip file. So how can we mount the filesystem buried beneath the gzip file? This puzzle can be solved by examining at the steps used to create the familiar compressed root disk floppy.

1. A ramdisk is created, mounted and filled with files.
2. The ramdisk device is unmounted.
3. The contents of the ramdisk are dumped to an image file using **dd**.
4. The image file is compressed with **gzip**.
5. The compressed image file is written to floppy with **dd**.

If that is how the compressed image makes its way from ramdisk to compressed floppy, then going from compressed floppy to ramdisk should be as simple as running through the steps in reverse.

1. The compressed image file is read from floppy with **dd**.
2. The image file is uncompressed with **gunzip**.
3. The contents of the image file are dumped into ramdisk using **dd**.
4. The ramdisk device is mounted.
5. The files are available.

We can cut out the intermediate image file by using a pipe to combine **dd** and **gunzip** like this: **dd if=/dev/fd0 | gunzip −cq > /dev/ram1**. Now the compressed floppy goes straight into ramdisk, decompressing on the fly.

A.2.2.2. Root disk support for additional ramdisks

We already have kernel support for ramdisks, because we are using a compressed root disk, but we will need to create more ramdisks in `/dev`. Typically the kernel supports eight ramdisks on `/dev/ram0` through `/dev/ram7` with `ram0` being used for the rootdisk. The `devices.txt` file included in the Linux source code documentation will be helpful for matching devices to their major and minor numbers.

A.2.3. Accessing audio files

The sample mp3 file that we will be using in our example is small enough to fit on an uncompressed floppy disk so that there is no need to burn a CD. However, serious music lovers may want to have the capability to mount a custom CD−ROM full of tunes and that option will require support for additional hardware.

A.2.3.1. CD−ROM hardware support

Most modern CD−ROM drives will use IDE devices like `/dev/hdc` or `/dev/hdd`. To support these CD−ROM drives we will have to configure IDE support in the kernel and create the appropriate device files on the root disk.

A.2.3.2. CD−ROM filesystem support

CD−ROMs have different filesystems than hard disks and floppies. Most CD burning applications use a filesystem called ISO−9660 and have the capability to support Joliet or Rockridge extensions. We will have to include support for these filesystems in the kernel in order to mount CD−ROMs.

A.2.4. Other required files

We will want to have all of mp3blaster's required libraries and other supporting files available as part of the compressed `/usr` image so that mp3blaster can run correctly. The familiar **ldd** command can be used to determine which libraries mp3blaster requires. Any additional libraries can be placed in `/usr/lib`. Even though some of the libraries may appear in `/lib` on the development system, they can still go in `/usr/lib` on the Pocket Linux system. The dynamic linker, `ld-linux.so`, is smart enough to look in both places when loading libraries.

Because mp3blaster uses the curses (or ncurses) screen control library there is one additional file we need. The curses library needs to know the characteristics of the terminal it is controlling and it gets that information from the terminfo database. The terminfo database consists of all the files under the `/usr/share/terminfo` directory and is very large compared to our available disk space. But, since Pocket Linux only supports the PC console, we only have one terminal type to worry about and therefore need only one file. The piece of the terminfo database we need is the file `/usr/share/terminfo/l/linux`, because we are using a "Linux" terminal. For more information about the subject of curses, see John Strang's book entitled "Programming with Curses" available from O'Reilly publishing.

A.2.5. Summary of tasks

Between sound cards, ramdisks, CD−ROMs and terminfo there is quite a bit to keep track of. So let's take a moment to organize and summarize the tasks necessary to make the pocket jukebox a reality.

- Create a new kernel disk that includes built−in support for audio hardware, IDE devices and CD−ROM filesystems.
- Create the appropriate `/dev` files on the root disk to support audio hardware, additional ramdisks and IDE CD−ROMs.
- Install the **gunzip** utility to enable decompression of the usr image.
- Create a startup script to load a compressed image from floppy into a ramdisk and mount the ramdisk on `/usr`.
- Create a compressed floppy that holds the mp3blaster program, its required libraries and terminfo files.

A.3. Construction

A.3.1. Create an enhanced boot disk

A.3.1.1. Build a new kernel

```
bash# cd /usr/src/linux
bash# make menuconfig
```

Be sure to configure support for the following:

- 386 processor
- Floppy disk
- RAM disk
- Second extended (ext2) filesystem
- Virtual console
- Audio hardware
- CD−ROM hardware
- ISO−9660 and Joliet filesystems

```
bash# make dep
bash# make clean
bash# make bzImage
```

A.3.1.2. Copy the kernel to diskette

Place the boot disk in drive fd0

```
bash# mount /dev/fd0 /mnt
bash# cp /usr/src/linux/arch/i386/boot/bzImage /mnt/boot/vmlinuz
```

A.3.1.3. Unmount the boot disk

```
bash# cd /
bash# umount /mnt
```

A.3.2. Create an enhanced root disk

A.3.2.1. Create additional device files

A.3.2.1.1. IDE CD−ROM

```
bash# mknod −m640 ~/staging/dev/hdc b 22 0
bash# mknod −m640 ~/staging/dev/hdd b 22 64
```

Optionally create additional IDE devices.

A.3.2.1.2. Ramdisk

```
bash# mknod −m 640 ~/staging/dev/ram1 b 1 1
bash# mknod −m 640 ~/staging/dev/ram2 b 1 2
bash# mknod −m 640 ~/staging/dev/ram3 b 1 3
bash# mknod −m 640 ~/staging/dev/ram4 b 1 4
bash# mknod −m 640 ~/staging/dev/ram5 b 1 5
bash# mknod −m 640 ~/staging/dev/ram6 b 1 6
bash# mknod −m 640 ~/staging/dev/ram7 b 1 7
```

A.3.2.1.3. Audio

```
bash# mknod -m664 ~/staging/dev/dsp c 14 3
bash# mknod -m664 ~/staging/dev/mixer c 14 0
```

A.3.2.2. Install the gunzip binary

```
bash# cd /usr/src/gzip-1.2.4a
bash# export CC="gcc -mcpu=i386"
bash# ./configure --host=i386-pc-linux-gnu
bash# make
bash# strip gzip
bash# cp gzip ~/staging/bin
bash# ln -s gzip ~/staging/bin/gunzip
```

Don't forget to verify library requirements, check the ownership and check permissions on the gzip binary.

A.3.2.3. Write a startup script to mount a compressed floppy

Use a text editor to create the following script and save it as `~/staging/etc/init.d/usr_image`

```
#!/bin/sh
#
# usr_image - load compressed images from floppy into ramdisk and
#             mount on /usr.
#
echo -n "Is there a compressed diskette to load for /usr [y/N]? "
read REPLY
if [ "$REPLY" = "y" ] || [ "$REPLY" = "Y" ]; then
  echo -n "Please insert the /usr floppy into fd0 and press <ENTER>."
  read REPLY
  echo "Clearing /dev/ram1."
  dd if=/dev/zero of=/dev/ram1 bs=1k count=4096
  echo "Loading compressed image from /dev/fd0 into /dev/ram1..."
  (dd if=/dev/fd0 bs=1k | gunzip -cq) >/dev/ram1 2>/dev/null
  fsck -fp /dev/ram1
  if [ $? -gt 1 ]; then
    echo "Filesystem errors on /dev/ram1!  Manual intervention required."
  else
    echo "Mounting /usr."
    mount /dev/ram1 /usr
  fi
fi
#
# end of usr_image
```

Configure the script to run right after root is mounted.

```
bash# ln -s ../init.d/usr_image ~/staging/etc/rcS.d/S21usr_image
```

A.3.2.4. Create a compressed root disk

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/staging/* /mnt
bash# umount /dev/ram7
```

```
bash# dd if=/dev/ram7 of=~/phase8-image bs=1k
bash# gzip -9 ~/phase8-image
```

Insert the diskette labeled "root disk" into drive fd0.

```
bash# dd if=~/phase8-image.gz of=/dev/fd0 bs=1k
```

A.3.3. Create a compressed /usr disk for mp3blaster

The compressed /usr diskette will be created in using the same process that is used to create the compressed root disk. We will copy files to a staging area, copy the staging area to ramdisk, compress the ramdisk and write it to diskette.

A.3.3.1. Create a staging area

```
bash# mkdir ~/usr-staging
bash# cd ~/usr-staging
bash# mkdir bin lib
bash# mkdir -p share/terminfo/l
```

A.3.3.2. Install the mp3blaster program

Download the latest version of mp3blaster source code from its home at http://www.stack.nl/~brama/mp3blaster/.

```
bash# cd ~/usr/src/mp3blaster-3.2.0
bash# ./configure
bash# make
bash# cp src/mp3blaster ~/usr-staging/bin
```

A.3.3.3. Copy additional libraries and terminfo

Use **ldd** to find out which libraries are needed for mp3blaster.

The following is an example from the author's development system. It is possible that different systems may yield slightly different results in terms of library requirements.

```
bash# cd ~/usr-staging/lib
bash# ldd ~/usr-staging/bin/mp3blaster
bash# cp /usr/lib/ncurses.so.5.0  .
bash# cp /usr/lib/stdc++.so.3 .
bash# cp /lib/libm.so.6 .
bash# cp /usr/lib/libgcc_s.so.1 .
bash# cd ~/usr-staging/share/terminfo/l
bash# cp /usr/share/terminfo/l/linux .
```

A.3.3.4. Make a compressed image and copy it to diskette

```
bash# cd /
bash# dd if=/dev/zero of=/dev/ram7 bs=1k count=4096
bash# mke2fs -m0 /dev/ram7
bash# mount /dev/ram7 /mnt
bash# cp -dpR ~/usr-staging/* /mnt
bash# umount /dev/ram7
```

```
bash# dd if=/dev/ram7 of=~/mp3blaster-image bs=1k
bash# gzip -9 ~/mp3blaster-image
```

Insert the diskette labeled "mp3blaster" into drive fd0.

```
bash# dd if=~/mp3blaster-image.gz of=/dev/fd0 bs=1k
```

A.3.4. Create a data diskette for testing

Go to the Internet site http://www.paul.sladen.org and download the mp3 file of Linus Torvalds pronouncing "Linux." The direct link is: http://www.paul.sladen.org/pronunciation/torvalds−says−linux.mp3. Create a Second Extended (ext2) filesystem on a floppy and copy the mp3 file onto the diskette.

A.4. Implementation

A.4.1. System Startup

1. Boot from the kernel diskette.
2. Insert the root floppy when prompted.
3. When prompted for a /usr diskette, say 'Y'.
4. Insert the mp3blaster diskette and press **Enter**.

A.4.2. Verify that the /usr diskette loaded properly

```
bash# mount
bash# ls -lR /usr
```

A.4.3. Check the audio device initialization

```
bash# dmesg | more
```

If everything worked there should be a line or two indicating that the kernel found the audio hardware. The example below shows how the kernel might report a Yamaha integrated sound system.

```
ymfpci: YMF740C at 0xf4000000 IRQ 10
ac97_codec: AC97 Audio codec, id: 0x4144:0x5303 (Analog Devices AD1819)
```

A.4.4. Test audio output

```
bash# echo "Garbage" > /dev/dsp
```

A short burst of static coming from the PC speakers indicates that sound is working.

A.4.5. Play a sample file

Insert the diskette containing the sample audio file.

```
mount /dev/fd0 /home
bash# /usr/bin/mp3blaster
```

Use mp3blaster to select and play the file `/home/torvalds-says-linux.mp3`. Use mp3blaster's mixer controls to adjust the volume as needed.

A.4.6. System shutdown

Bring the system down gracefully with the **shutdown** command.

Appendix B. GNU Free Documentation License

Version 1.2, November 2002

B.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

B.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world−wide, royalty−free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or Back−Cover Texts, in the notice that says that the Document is released under this License. A Front−Cover Text may be at most 5 words, and a Back−Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

B.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

B.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer−network location from which the general network−using public has access to download using public−standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

B.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

B.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

B.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

B.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

B.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

B.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

B.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.12. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front−Cover Texts, and no Back−Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front−Cover Texts and Back−Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.