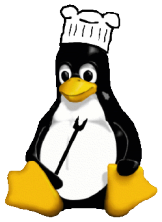
The Open Pitt

What's cooking in Linux and Open Source in Western Pennsylvania



Issue 18 November 2005 www.wplug.org

Building an Embedded Gentoo System *by Tom Watson*

Why would you want to build your own embedded Gentoo system? Firstly, it's not quite as hard as you might think. You may have an old PC with limited resources that you'd like to turn into a web server or NFS server. Or you may have a more traditional embedded system like those found at <http://www.soekris.com/> that you would like to get running.

To follow along with this article you won't need any special hardware. It can all be built and tested on your desktop Linux box. Some helpful files can be found at the URL <http://www.shadysidesupercomputingcenter.com/sywt/gentoo.embedded/>, such as a step-by-step script named "sywt.ge.stepbystep.txt."

The first thing to do is log in as root and create a working directory; I used /opt/embedded. Next we need to download a stage1 tarball and a portage snapshot. These can be found on the Gentoo mirrors (the step-by-step script includes URLs).

The stage1 tarball provides us with a base set of tools we need to build our embedded system. The portage snapshot provides us with the latest package dependency database and tools.

Once these files are downloaded we need to create our arch directory, which is where we will do the real work. Since my target is an i386 machine we do as follows:

```
# mkdir i386
# cd i386
```

Now that we have a working directory we need to unpack our stage1 tarball (working tools) and the portage snapshot:

```
# tar -jxf ../stage1*
# tar -jxf ../portage* -C usr
```

The "-C usr" means change to the usr directory before extracting the files, which puts the portage files in the correct location. Next we need to give our build environment access to the special files in /proc by mounting our proc directory onto the work file system:

```
# mount --bind /proc proc
```

We also need to copy a valid resolv.conf file to the working filesystem so Internet addresses can be looked up for the upcoming install:

```
# cp /etc/resolv.conf
etc/resolv.conf
```

Now we can enter our new environment and start working in our newly-created little Linux box. I generally use the "nice" command to give a lower priority to subsequent operations. This helps prevent the build process from slowing down other activities on the machine:

```
# /bin/nice -n 19 chroot .
/bin/bash --login
# env-update
# source /etc/profile
```

Now our shell is running inside the new environment and anything we do only affects the embedded system. If needed, customize the etc/make.conf file to set compiler options, match your target environment, and include any desired USE flags. For this article we will just stick with the defaults and build for the i386 architecture, so we can skip this step.

At this point we also need to change our build profile to suit the embedded target. We will be using uClibc, a special version of libc designed for microcontrollers and other small systems. Switching is just a matter of softlinking the uClibc profile to /etc/make.profile:

```
# unlink /etc/make.profile
# ln -sf /usr/portage/pro-
files/uclibc/x86/2005.1/
/etc/make.profile
```

See **GENTOO**, *p. 2*

October Roundup

Oct. 15 General User Meeting: **Patrick Wagstrom** spoke about the interactions between weblogs and Open Source. He covered some of the popular weblog packages and described various methods used to aggregate or link together weblogs. Patrick demonstrated how this information can be used to study whether, and to what extent, different Open Source projects interact.

Oct. 22 Tutorials: **Bill Moran** covered the Simple Object Access Protocol (SOAP), which is increasingly being used for distributed applications. After reviewing the basics, he described some of the programming libraries that can provide your applications with SOAP functionality. **Mike O'Connor** followed with a discussion of how and under what conditions Linux can be successfully used within a small organization. As an example, he used First Trinity Evangelical Lutheran Church, which has a mixed Linux, Windows, and Macintosh environment.

Coming Events

Dec. 3: General User Meeting, Topic: Mandriva. 10AM to 2PM, 1507 Newell-Simon Hall, CMU

Dec. 17: Tutorial, Topic: Intro to Objective C. 10AM to 3PM, 1507 Newell-Simon Hall, CMU

Jan. 7: Installfest. 10AM to 5PM, 1507 Newell-Simon Hall, CMU

Jan. 14: General User Meeting. 10AM to 2PM, 1507 Newell-Simon Hall, CMU

The public is welcome at all events

GENTOO, *from p. 1*

Now we are ready to bootstrap the system. This procedure builds our development environment:

```
# cd /usr/portage/scripts
# ./bootstrap.sh
```

You can step away from your machine for a while now as it builds. Once complete, we can then rebuild our base system to match our make flags and the rest of the build environment:

```
# emerge -e system
```

You can step away again for a while now...

OK, now we have a pretty much fully-fledged Gentoo base system. The next thing is to create our rootfs, the directory where we put the files that will end up in actual image. We also need to install a base layout into the directory structure, and emerge the required tools for the system.

You will see "ROOT=/rootfs" in front of most of these commands. This tells emerge to install the results in /rootfs:

```
# mkdir /rootfs
# cd /usr/portage/sys-apps/baselayout-lite/
# ROOT=/rootfs emerge baselayout-lite-1.0_pre1.ebuild
# ROOT=/rootfs emerge uclibc
# ROOT=/rootfs emerge busybox
```

BusyBox is a really neat piece of software. It replaces the standard binutils packages with a single file and a whole lot of symlinks to this file. Based on the command used to run it, BusyBox modifies its behavior. If invoked as "ls" it gives a directory listing, as "cp" it copies files, and so on for many of the standard UNIX commands. This greatly reduces the size of the system which is exactly what we want.

Now that BusyBox is installed, we need to create the symlinks:

```
# mount -o bind /proc/ /rootfs/proc/
# chroot /rootfs /bin/busybox --install -s
# umount /rootfs/proc
```

Next we should probably set the root password on the box:

```
# chroot /rootfs /bin/ash
# passwd
# rm /etc/passwd-
# exit
```

We also need to download, configure, and build a kernel. There is a kernel.config file on the web site configured for this article. It's easier to build the kernel outside of the chroot environment. If you don't want to compile your own, a prebuilt 2.6.14.2 kernel is available on the web site. Copy your kernel to /rootfs/boot.

The last thing to do before we build an image is to edit our /rootfs/etc/fstab file to include the root filesystem. Add a line containing "/dev/hda / reiserfs defaults" to this file.

The /rootfs directory now contains your complete system. For this article, we will use the QEMU emulator to run our little system instead of booting it directly.

Create a tarball of all the files to go into the QEMU image:

```
# cd /rootfs
# rm -R /rootfs/var/lib/portage/
# rm -R /rootfs/var/db/pkg/*
# cd /rootfs
# tar -zcf ../rootfs.tgz .
# exit
```

The last exit command will take you out of the chroot environment and back to the regular shell.

If QEMU is not already installed on your machine, you'll have to do that now. Then you create an empty 64 megabyte image file and format it with the Reiser filesystem:

```
# qemu-img create gentoo.img 64M
# mkfs.reiserfs -q gentoo.img
```

To put your embedded filesystem onto the empty image file, mount it using the loopback device. Here we're mounting it on /mnt/floppy (use a different mount point on your system if you wish):

```
# mount -o loop gentoo.img /mnt/floppy
```

Then you'll untar the root filesystem into the image and unmount it:

```
# tar -zxf rootfs.tgz -C /mnt/floppy
# umount /mnt/floppy
```

Finally, boot the image with QEMU:

```
# qemu -hda gentoo.img -kernel rootfs/boot/kernel-2.6.14.2 -append "root=/dev/hda"
```

Congratulations, you now have a basic running system. There is still a lot of configuration (such as networking) that needs to be done, but I'll leave that up to you.

The Open Pitt is published by the Western Pennsylvania Linux Users Group
<http://www.wplug.org/top/>

Editors: Elwin Green
Vance Kochenderfer

What is Linux?

Linux is a *kernel*, the core of a computer operating system, created by Linus Torvalds. It is typically packaged as a *distribution*, which includes the extra programs necessary to make a computer functional and useful. Since 1991, it has grown from a one-man project which ran on one computer to one with thousands of contributors running on everything from personal organizers to million-dollar supercomputers.

What are Open Source and Free Software?

Open Source and Free Software provide you, the user, with the opportunity to see the source code of the programs you use. You are free to use it, share it with others, and even make changes to it if you wish. While the Free Software and Open Source communities differ in their philosophical approach, in practical terms they share nearly identical goals. Learn more at <http://www.opensource.org/> and <http://www.gnu.org/>.

This newsletter was produced using Open Source and Free Software.

New WPLUG Officers Chosen

Five members of the Board of Directors were elected by the WPLUG membership at the November 5 Annual Meeting. They have taken the following offices:

Chair: Bill Moran
Vice-Chair: Beth Lynn Eicher
Secretary: David Ostroske
Treasurer: Patrick Wagstrom
Director-at-Large: Chris Teodorski

Thanks go to them and to all the candidates who stepped forward to run in WPLUG's first election.

Of course, WPLUG will still need plenty of other volunteers to keep its events going. You should post to the <wplug-plan@wplug.org> mailing list if you have an interest in helping out.