Gentoo Workshop

Mark Platek
Clarkson University
Fall 2010

Resources (a reminder)

- Gentoo Handbook:

  http://www.gentoo.org/doc/en/handbook/handbook-x86.xml

  http://www.gentoo.org/doc/en/handbook/handbook-amd64.xml

- Gentoo Forums

  http://forums.gentoo.org

- Myself – ask if you have questions!

  Email: platekme@clarkson.edu (or platekme89@gmail.com)

  IM: backslash54 v2

Today: Kernel Compilation

- Compiling the kernel is probably the most difficult part of performing a Gentoo installation. But, if you're well-prepared, it's not so bad!
- You should know what sort of hardware you have – then, by deliberately including only drivers for your specific hardware, you can minimize compilation time.
- You can just build every available driver into your kernel, but you'll be here all night while it compiles - this “shotgun” method for driver inclusion will probably work, but it's not practical.
- Use the output of the `lspci` command (available in the 'pciutils' package) to see the hardware you need to find drivers for.
- Depending on how far we get with the kernels, we might stop early.

Now, where were we?

- When we left off last time, we were chrooted into the new installation. We'll have to return to that environment. Do you remember how?
- For purposes of reference, the /boot partition is /dev/sda1, the swap partition is /dev/sda2, and the / partition is /dev/sda3. Your partitions might differ.

```
`mount /dev/sda3 /mnt/gentoo`
`mount /dev/sda1 /mnt/gentoo/boot`
`mkswap /dev/sda2`
`swapon /dev/sda2`
`mount -t proc none /mnt/gentoo/proc`
`mount -o bind /dev /mnt/gentoo/dev`
`chroot /mnt/gentoo /bin/bash`
`env-update && source /etc/profile`
`export PS1="(chroot) $PS1"`
```

Emerging the kernel source

- Before we can build a kernel, we'll have to install the source. Conveniently, there is an ebuild for the kernel, along with a Gentoo-specific patchset.
- If you don't want to use the Gentoo patchset, you can use a vanilla kernel. I'd recommend using the Gentoo kernel, though.
- To install the kernel source:

  `emerge gentoo-sources`
- Or, if you want to use a vanilla kernel:

  `emerge vanilla-sources`

- This will take a few minutes, the kernel source tree is ~400M.

Verify kernel symlink

- You can have multiple versions of the Linux kernel installed at once on a system (though of course you can only run one at once). To select between them, you set the /usr/src/linux symlink.
- Remember the eselect tool? We'll now use it to verify that our kernel symlink has been updated when we installed the kernel sources.
- Run `eselect kernel list` to see a list of possible kernel symlink targets, find the number that corresponds to the version you want to use. In this case, only one kernel is installed, so we want '1'.
- Set the kernel symlink with `eselect kernel set 1`.

News? In my Portage?

- It's more common than you might think. Sometimes, like when major changes are made to key pieces of software (e.g. perl, python, GNOME) a news bulletin will be posted in the Portage tree.
- You can read these news items with eselect as well: `eselect news list` lists all news items, and `eselect news read 1` for example prints the first one.
- In this case, the first news item is about a major upgrade to Python 3. We don't need to worry because nothing was changed by default. (We can switch to the new Python using eselect, by the way.)
- The other item is about a new LDFLAG that has been added to the default profile. It shouldn't cause us any problems.

Oops!

- Unfortunately, I was a bit overzealous with the USE variable I had you all set last time. It's way more than we need right now, and there is really no reason to compile most of the packages that it ends up pulling in as dependencies (we'll want them later, but not now).
- So, please edit your /etc/make.conf:

  Comment out the long USE variable.

  Replace it with `USE=”mms sse sse2”`
- Also take this time to install a good text editor if you haven't already - nano isn't going to cut it anymore.

  \`emerge vim\` or \`emerge emacs\`
- Now, back to our regularly-scheduled program...

Updating configuration files

- In many cases, a program's behavior is determined by configuration file(s). But, what happens when the program is updated to a new version? Will the old config file still work?
- Generally, the answer is 'no'. You have to make whatever modifications are necessary to get the configuration file working with the new version. Thankfully, tools exist to perform this task is a semi-automated way: we will be using cfg-update.
- Install cfg-update: `emerge cfg-update`.
- Now, we can get rid of that “config files need updating” notice. Run `cfg-update -u` and it will look for old config files that have not been merged with replacement new ones.
- It's important to note that a default “reference” config file is kept with each version of a package. The old config file is simply replaced with this new reference one if you have not changed the old one, but you must merge the two manually if you have changed the old config file from its reference. cfg-update helps you do this.

Updating config files (cont'd)

- Such is the case with our update! We have to use cfg-update to merge the old config file (which apparently is different from its reference) and the new reference file.
- cfg-update in text mode uses sdiff to perform the merges – it will print the old file on the left and the new one on the right, and show differences between them in standard “diff” notation.
- In this case, we will use everything from the new file (enter 'r' at every prompt).
- Finally, hit 'v' to check out the merged file, then hit '1' to replace the old file with the merged file.

- It is possible to control the behavior of most Linux programs with config files, so it's worth taking the time to check them out – most are well-commented. Good examples include /etc/sudoers and /etc/cfg-update.conf

It's kernel time!

- Prepare yourselves, it's time for the real deal. I will be doing this with you, but my kernel <u>will</u> be different from yours! So, don't use my configuration as a reference.
- To begin building your kernel, go to the source directory:

  `cd /usr/src/linux`
- Then, start the curses-based configuration system:

  `make menuconfig`
- We are greeted with a list of components to build with the kernel. Components with an '*' will be built-in, whereas components represented by an 'M' will be built as modules. Built-in components are available at boot time, so it is necessary to build in things like drivers. However, components that you don't need immediately (e.g. extra filesystems) can be built as modules, and you can shave a few seconds off of your boot time.
- If you aren't sure about whether to include an item, you are probably safe to leave the default configuration. Be very careful adding features marked “experimental”.

Kernel: General Setup

- For the most part, we can leave this section alone – the default configuration is sufficient.
- Have a look at the things you can change, though. Note that you can create a ramdisk on boot from the contents of a file – this might be convenient later on.

Kernel: Module Support / Block Layer

- Once again, you can probably use the default configuration for both of these. I like to add forced module loading, though I've never had cause to use it.
- Have a look at the available I/O schedulers, though – the Deadline scheduler was used up until 2.6.18 when CFQ (Completely Fair Queuing) became the default.
- The scheduler deals with I/O requests. CFQ tries to minimize costly hard disk seeks as much as possible. But, what if you have an SSD and seeks are of no concern? Then, you might want to use the No-Op I/O scheduler – it does not waste CPU time computing the best seek pattern.

Kernel: Processor

- Here's an important one! You will define kernel components to do with the processor.
- Be sure to set the processor family!
- You can set the preemption model for the kernel as well – it, like the scheduler, has a a profound effect on the kernel's behavior.
  - The safest is no preemption, but the kernel will be slower to react.
  - Full preemption means that the kernel can always interrupt any process. This decreases the kernel's reaction time to user events, but can cause the kernel to get bogged down under very heavy loads.
  - Voluntary preemption is a decent compromise between the two, but full preemption generally works very well.
- You can safely disable features that don't apply to your processor (e.g. AMD microcode update support if you have an Intel processor).

Kernel: PM and ACPI

- ACPI is one way for hardware to report its state to the OS. For example, laptop batteries generally report their charge state through ACPI.
- Support for CPU frequency scaling allows the OS to decrease the CPU's clock multiplier on the fly, causing the CPU to run at a much lower frequency and consume much less power. Laptop users will want to make sure this is enabled.
- Also, if you leave the governor in userspace, you can use the Gnome frequency scaling applet.

Kernel: Bus Options / Executable Support

- In this section, we'll set components to define the behavior of the PCI bus.
- Not much to say about this, you can probably leave the defaults.
- I recommend that you include IA32 a.out support, though. Otherwise, there are some 32-bit binaries that you might not be able to run.

Kernel: Networking Support

- This is another big one – in this section we define the behavior of the network subsytem.
- In Network Options, we can include support for different network protocols implemented in the kernel (IP, TCP, UDP, etc).
- If your machine has bluetooth, you'll need to check the configuration.

Kernel: Device Drivers

- This section is the most important to get right! If you don't include drivers for your hardware, you can't use it – this gets important when we're talking about HDD drivers.
- Everyone will have a different configuration here – the best way to approach this section is to methodically go through each option and subsection. If you have two drivers that you can't choose between, include them both to be safe.
- You can configure ramdisks in “Block devices”.
- Be sure to include support for your SATA (or IDE/PATA) controller in “Serial ATA and Parallel ATA Drivers”. Intel SCH family SATA controllers are very common, they are supported by default.
- You can include support for md raid (software RAID) devices in “Multiple devices driver support”. If you want to use software RAID, you'll want to include this.
- If you have firewire, you will need to enable the firewire stack.
- In “Network device support”, try to find your NIC. It will probably be listed under “Ethernet (1000Mbit)”.

Kernel: Device Drivers (cont'd)

- If you have Wifi, don't forget to include a driver for it!
- If you have an Intel Core-series processor, don't forget to include the temperature sensor in “Hardware monitoring support”.
- In “Graphics support” we need to add support for either Uvesafb or VESA. Include support for both if you want to try them out and find your preference. DON'T include support for the nvidia framebuffer, even if you have an Nvidia graphics card – it messes up the framebuffer when you install the Nvidia driver.
- You'll want to include support for your sound card, too – it is probably listed under “PCI sound devices”.
- Some extra nice things can be found in “x86 Platform Specific Device Drivers”.

Kernel: Firmware Drivers

- This section can be left in the default configuration, unless you have a Dell laptop.

Kernel: File Systems

- Here we will include support for different filesystems.
- At the very least we must include support for ext2 (since /boot is formatted with it) and xfs or ext4 (whichever / is formatted with). Generally, it is better to include many filesystems, though you may wish to build them as modules.
- We cannot build ext2 and xfs/ext4 as modules – if we do that, they will not be loaded until after they are needed. This results in an unbootable kernel!
- But, we can build reiserfs support as a module, since we don't use it but may want to in the future.
- It is a good idea to enable UDF support, so that you can read all DVDs/CDs.
- Don't forget to include NTFS support! We'll probably need to read an NTFS partition at some point.
- You can build in support for AFS if you want to play with it :).

Kernel: Hacking

- These options are most useful for kernel developers. So, there is no reason to enable most of them. We'll just stick to the defaults.
- You can omit frame pointers for the kernel if you wish.

Kernel: Security / Crypto

- Set security components (e.g. SELinux) in “Security Options”. The defaults are OK, but I generally remove SELinux, since I never use it.
- Add any cryptographic algorithms you feel like including. All commonly-used ones are selected by default, but you can add other ones if you wish.
- At the very bottom are compression algorithms, you should include all three.
    - A bit of trivia: Gzip uses DEFLATE, it's based on Huffman coding.

Kernel: Virtualization / Library Routines

- If you want to run virtual machines, you should enable all of the virtualization options.
- You can leave the library routines section at its default configuration.

Kernel: compilation

- Save your configuration – it will be saved to the file /usr/src/linux/.config.
- We would execute `make && make modules_install` to build the kernel now. But, let's see just how fast our machines are!

  `time make && make modules_install`
- `make` builds all built-in components
- `make modules_install` build all modules, them installs the binaries to /lib/modules/<kernel version>
- `time` keeps track of how long it takes to execute make

- The kernel will almost always compile – but will it boot? Did you include all device drivers? Filesystems?
- We'll find out later, there are still things to do before we can try to boot the machine.

Copyright