

Configuring and Building The Linux Kernel

Sunil Beta <betasam@gmail.com>

Duration: (45 +/- 15) mins

Prerequisites

- C programming/development basics
- *nix usage and building packages (shellscripts, make, ...)
- Basic OS course, monolithic, modular kernels

Target Audience

- Those who are new to linux (the kernel)
- Those who would like to experiment with linux (the kernel)
- People who have used Unix/*nix systems
- Wannabe Hackers
- Those who have built the 2.4 kernels and are looking for a bit of help on the 2.6 kernels

Building the Linux kernel is easy

- so long as you know what exactly to do
- if you have a PC running Linux, that's all you need

What is a Kernel?

(detailed definitions in OS theory books)

- The core of the Operating System
- Resource Organizer
- Resource Scheduler

Design Classification

- Monolithic Kernels
- Modular Kernels
- Micro Kernels
- Nano Kernels

Trivia about the Linux Kernel

Linux is classified as a

- Monolithic Kernel
- Modular Kernel

Current Versions of the Linux Kernel in use

- 2.4.x
- 2.6.x

2.[odd number].subversion

- Development Kernel [not for production use]

2.[even number].subversion.?interimversion?

- Stable Kernel [for production use]

Why [Re]build the Kernel?

- New/Non-standard Hardware
- Get access to the features of the newest kernel
- Optimize the kernel
 - for size
 - to extract maximum performance
- Hacking the Kernel

Hardware Requirements

The Development Machine

- 64MB-128MB RAM (recommended)
- Linux Distribution (2GB)
- 100MB + 400MB - Sources + Build files
- 4MB - 40MB - Built Kernel

Software Requirements

Building 2.4.x

■Gnu C	2.91.66	# gcc --version
■Gnu make	3.77	# make --version
■binutils	2.9.1.0.25	# ld -v
■util-linux	2.10o	# fdformat --version
■modutils	2.4.2	# insmod -V
■e2fsprogs	1.19	# tune2fs
■reiserfsprogs	3.x.0b	# reiserfsck 2>&1 grep reiserfsprogs
■pcmcia-cs	3.1.21	# cardmgr -V
■PPP	2.4.0	# pppd --version
■isdn4k-utils	3.1pre1	# isdnctrl 2>&1 grep version

Building 2.6.x

■Gnu C	2.95.3	# gcc --version
■Gnu make	3.78	# make --version
■binutils	2.12	# ld -v
■util-linux	2.10o	# fdformat --version
■module-init-tools	0.9.10	# depmod -V
■e2fsprogs	1.29	# tune2fs
■jfsutils	1.1.3	# fsck.jfs -V
■reiserfsprogs	3.6.3	# reiserfsck -V 2>&1 grep reiserfsprogs
■xfsprogs	2.1.0	# xfs_db -V
■pcmcia-cs	3.1.21	# cardmgr -V
■quota-tools	3.09	# quota -V
■PPP	2.4.0	# pppd --version
■isdn4k-utils	3.1pre1	# isdnctrl 2>&1 grep version
■nfs-utils	1.0.5	# showmount --version
■procps	3.1.13	# ps --version
■oprofile	0.5.3	# oprofiled --version

Software Requirements

Building 2.6.x

■Gnu C	2.95.3	# gcc --version
■Gnu make	3.78	# make --version
■binutils	2.12	# ld -v
■util-linux	2.10o	# fdformat --version
■module-init-tools	0.9.10	# depmod -V
■e2fsprogs	1.29	# tune2fs
■jfsutils	1.1.3	# fsck.jfs -V
■reiserfsprogs	3.6.3	# reiserfsck -V 2>&1 grep reiserfsprogs
■xfsprogs	2.1.0	# xfs_db -V
■pcmcia-cs	3.1.21	# cardmgr -V
■quota-tools	3.09	# quota -V
■PPP	2.4.0	# pppd --version
■isdn4k-utils	3.1pre1	# isdnctrl 2>&1 grep version
■nfs-utils	1.0.5	# showmount --version
■procps	3.1.13	# ps --version
■oprofile	0.5.3	# oprofiled --version

Determining current hardware

```
$ /sbin/lspci -v
```

```
00:00.0 Host bridge: Silicon Integrated Systems [SiS] 630 Host (rev 31)
00:00.1 IDE interface: Silicon Integrated Systems [SiS] 5513 [IDE] (rev d0)
00:01.0 ISA bridge: Silicon Integrated Systems [SiS] SiS85C503/5513 (LPC Bridge)
00:01.1 Ethernet controller: Silicon Integrated Systems [SiS] SiS900 10/100 Ethernet
(rev 82)
00:01.2 USB Controller: Silicon Integrated Systems [SiS] USB 1.0 Controller (rev 07)
00:01.3 USB Controller: Silicon Integrated Systems [SiS] USB 1.0 Controller (rev 07)
00:01.4 Multimedia audio controller: Silicon Integrated Systems [SiS] SiS PCI Audio
Accelerator (rev 02)
00:01.6 Modem: Silicon Integrated Systems [SiS] Intel 537 [56k Winmodem] (rev a0)
00:02.0 PCI bridge: Silicon Integrated Systems [SiS] SiS 530 Virtual PCI-to-PCI bridge
(AGP)
00:03.0 CardBus bridge: O2 Micro, Inc. OZ6912 Cardbus Controller
00:0a.0 FireWire (IEEE 1394): Lucent Microelectronics FW323 (rev 04)
01:00.0 VGA compatible controller: Silicon Integrated Systems [SiS] SiS630 GUI
Accelerator+3D (rev 31)
```

Determining current hardware

```
$ cat /proc/cpuinfo
```

```
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model    : 11
model name : Intel(R) Pentium(R) III Mobile CPU          1200MHz
stepping : 1
cpu MHz   : 1200.112
cache size : 512 KB
fdiv_bug : no
hlt_bug  : no
f00f_bug : no
coma_bug : no
fpu      : yes
fpu_exception : yes
cpuid level : 2
wp       : yes
flags    : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx fxsr sse
bogomips : 2375.68
```

Acquiring the Sources

1. Download Pristine Sources from Linus

`ftp://ftp.kernel.org/pub/linux/kernel/v2.4/`

`ftp://ftp.kernel.org/pub/linux/kernel/v2.6/`

visit `http://kernel.org/` for http, rsync details

2. Acquire sources for your distribution

- Redhat: source RPMs
- SuSE: SRPMs
- Debian: apt-source

Vendor Kernels and Pristine Kernels

- Vendors apply several patches to the kernel
 - to support additional hardware
 - to fix problems reported by distro users
 - offer support exclusively to their trees
 - learn more from www.kernelnewbies.org

The Sources

```
linux-2.6.0.tar.bz2  
patch-2.6.1.bz2  
patch-2.6.2.bz2  
...  
patch-2.6.9.bz2
```

```
$ tar jxvf /path/to/kernel-sources/linux-2.6.0.tar.bz2  
$ cd linux*/scripts/  
$ ./patch-kernel .. /path/to/kernel-sources/
```

[OR]

```
$ rpm -ivh /path/to/rpm/kernel-src-{version}.rpm  
look in /usr/src/packages/RPM for linux-{version}.tar.bz2
```

Assigning a Unique Name

not a "requirement", just for safety

linux/Makefile:

```
VERSION = 2  
PATCHLEVEL = 6  
SUBLEVEL = 9  
EXTRAVERSION =  
NAME=Zonked Quokka
```

Change EXTRAVERSION= to
EXTRAVERSION=-mybuild

This is to

- avoid overwriting existing kernel modules

More Safety

```
$ cd /path/to/linux
```

■ Backup .config

- \$ cp .config ~/somewhere/config.save

■ Wipe out any existing builds/objects

- \$ make mrproper

Configuring the Kernel

(for 2.6 kernels)

```
$ make config
```

Or

```
$ make oldconfig
```

Or

```
$ make menuconfig
```

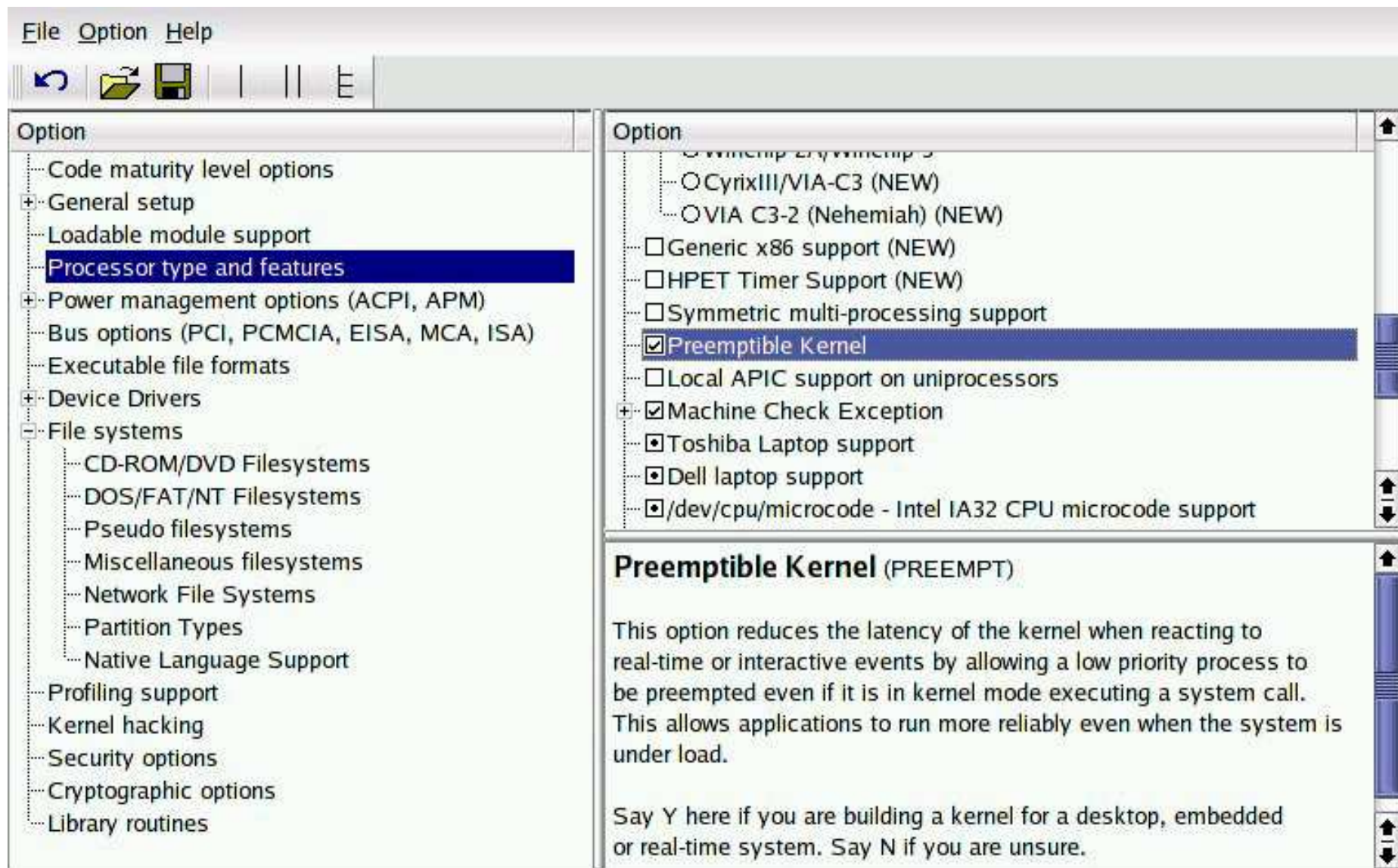
Or

```
$ make xconfig
```

Or

```
$ make gconfig
```

The Configuration Screen



```
$ make xconfig
```

The Configuration Options

- **Code Maturity Level**
 - Select "Alpha" quality software
- **General Setup**
 - sysctl?
- **Loadable Modules Support**
 - Modular kernel support
 - Module version information on symbols
- **Processor Type and Features**
 - Select specific Processor extensions
 - SMP, High Memory, Pre-emptible Kernel

The Configuration Options (...)

■ Bus Options

- PCI, PCMCIA, EISA, ISA

■ Executable File Formats

■ Device Drivers

- I/O hardware, SCSI cards, USB, HID ...

■ File Systems

- ext2, ext3, reiserfs, ...

■ Security architecture

■ Kernel Hacking

- debug symbols, kgdb, magicysrq

■ Kernel Libraries

Building the kernel

```
$ cd /path/to/linux  
$ make
```

With 2.6 the build system is simplified

- presents fewer messages on-screen
- can change verbosity with `make V=<verbosity>`

Takes between 10-45 minutes (P4,P3,Celeron)

- depending on the host processor
- the configuration (selected drivers, etc.)

Installing the Kernel

■ Installing Modules

```
$ su -C "make modules_install"
```

■ Creating the Initial Ramdisk

- (On Debian)

- \$ mkinitrd /boot/initrd-2.6.x.img 2.6.x

- please read mkinitrd manuals if you use a different distribution

■ Installing the kernel image

- \$ cp /path/to/linux/arch/i386/boot/bzImage
/boot/vmlinuz-2.6.x

■ Copying the System.map

- \$ cp /path/to/linux/System.map /boot/System.map-2.6.x

Configuring your Bootloader

- You probably use
 - GRUB (Grand Unified Bootloader)
 - LILO (Linux Loader)
- The Bootloader needs to know
 - where your kernel resides
 - where your initrd resides
 - what options need to be passed to the kernel

Configuring GRUB

```
default=0
timeout=10
title Red Hat Linux (2.4.20-24.9)
    root (hd0,1)
    kernel /boot/vmlinuz-2.4.20-24.9 ro root=LABEL=/
    initrd /boot/initrd-2.4.20-24.9.img
title Red Hat Linux (2.4.20-20.9)
    root (hd0,1)
    kernel /boot/vmlinuz-2.4.20-20.9 ro root=LABEL=/
    initrd /boot/initrd-2.4.20-20.9.img
title Test Kernel (2.6.0)
    root (hd0,1)
    kernel /boot/bzImage-2.6.0 ro root=LABEL=/
    initrd /boot/initrd-2.6.0.img
```

typical configuration listing of /boot/grub/menu.lst

Configuring LILO

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
default=test-2.6.0
keytable=/boot/us.klt
lba32
prompt
timeout=50
message=/boot/message
menu-scheme=wb:bw:wb:bw
image=/boot/vmlinuz
    label=linux
    root=/dev/hda3
    append=" idel=autotune ide0=autotune"
    read-only
image=/boot/bzImage-2.6.0
    label=test-2.6.0
    root=/dev/hda2
    read-only
```

typical configuration listing of /etc/lilo.conf
after configuring lilo, run:

```
$ /sbin/lilo
```

Helper Utilities & Tips

- **module-init-tools**
 - makes it possible to "modprobe module-name"
- **udev (deprecates devfs)**
 - makes hotplugging possible
 - alternate utilities supporting udev also exist
- **ensure essential drivers are loaded before apps start**
 - like DRM (for most X11 installation)
 - ethernet/network device driver (for most network apps)
- **make sure you run `$ depmod -a`**
- **avoid building anything (including the kernel) as root**

When Things go wrong ...

■ Kernel refuses to build

- rebuild with make V=3
- redirect output to a file
 - `$ make V=3 1>&2 | tee ~/somewhere/build.log`
- study/report message on a mailing list

■ Kernel refuses to boot

- note kernel boot messages
 - at least the last line
- study/report message on a mailing list

■ X refuses to load, applications refuse to work

- report on the application mailing list

Essential Reading

This Presentation is Derived from:

- <http://www.linux.org/docs/ldp/howto/Kernel-HOWTO/>
<http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html>

Before building the 2.6 kernel, read this:

- <http://kerneltrap.org/node/view/799>

If you want to write your own device drivers, you **MUST** read:

- Linux Device Drivers, 2nd Edition
- (this still doesn't cover the 2.6 kernel)

Thankyou!

This Presentation

- is created with "magicpoint"
 - <http://mew.org/mgp/>
- was edited in emacs
 - <http://www.emacs.org/>
- visit <http://www.codito.com/~beta/kernelbuild/>
- or email "betasam[AT]gmail[DOT]com"

Writing your own Drivers / Modules

- You have read "Linux Device Drivers" (Alessandro Rubini)
- You want to
 - Fix a Kernel Bug
 - Report Bugs and check Mailing lists first
 - Support New Hardware
 - Use Customized I/O Hardware
 - Create a New Kernel Feature
 - Customize the kernel to support your application*
- Read path/to/linux/Documentation/*

First Steps

- Rebuild your kernel with some third party patch
 - like <http://bootsplash.de/>
- Source code comprehension
 - visit <http://lxr.linux.no/>
- understand kernel driver layout
- understand the basics of ELF
 - ELF sections
 - the New .ko and its ELF sections
- understand the linux boot process
- understand the linux init procedure
 - visit <http://www.tldp.org/> for both

Starting to Code

- Read a driver in the same class as the one you're about to write
- The basic classes of drivers are:
 - block
 - character
 - network
- some special extensions are used for:
 - framebuffer
 - USB client hardware
 - IEEE1394 clients
 - SCSI hardware
 - ...
- Join a relevant mailing list
 - like lkml (warning: heavy traffic)
 - visit www.vger.org for more details

Starting to Code ...

- Try using a different directory structure for your driver
 - better to stay out of the kernel tree
 - unless you expect your code to get included into the kernel
- Many hardware companies provide modules/drivers
 - that exist outside the kernel
- Understand the kernel licensing framework
 - in-kernel code NEEDS to be GPL
 - modular code can be GPL or LGPL or GPL-compatible

Tips for Driver Writing

- One Driver class at a time
- Do not write in-kernel code unless it's impossible to make the code modular
- use CVS or Arch or Bitkeeper for versioning
- Follow the kernel coding standards - avoid inventing your own
- If you can write a user app (and/or scripts) to test your driver, Great!