# 06-1 The Kernel

## It all started with...

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix –

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat(same physical layout of the file-system (due to practical reasons)among other things).

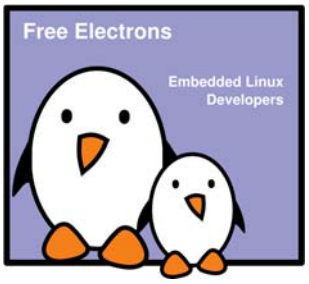
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

## Free Electrons

# Linux kernel introduction

Michael Opdenacker
Thomas Petazzoni
**Free Electrons**

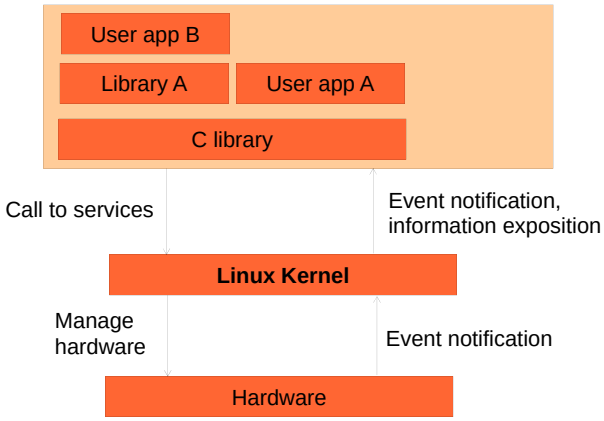





# Embedded Linux driver development

Kernel overview
Linux features

## Linux kernel in the system




## History

- The Linux kernel is one component of a system, which also requires libraries and applications to provide features to end users
- The Linux kernel was created as a hobby in 1991 by a Finnish student, Linus Torvalds
- Linux quickly started to be used as the kernel for free software operating systems
- Linus Torvalds has been able to create a large and dynamic developer and user community around Linux
- Nowadays, hundreds of people contribute to each kernel release, individuals or companies big and small

## Linux license

- The whole Linux sources are Free Software released under the GNU General Public License version 2 (GPL v2).
- For the Linux kernel, this basically implies that:
  - When you receive or buy a device with Linux on it, you should receive the Linux sources, with the right to study, modify and redistribute them.
  - When you produce Linux based devices, you must release the sources to the recipient, with the same rights, with no restriction.
- See our http://free-electrons.com/articles/freesw/ training for exact details about Free Software and its licenses.

## Linux kernel key features

- Portability and hardware support. Runs on most architectures.
- Scalability
  Can run on super computers as well as on tiny devices
  (4 MB of RAM is enough).
- Compliance to standards and interoperability.
- Exhaustive networking support.
- Security
  It can't hide its flaws. Its code is reviewed by many experts.
- Stability and reliability.
- Modularity
  Can include only what a system needs even at run time.
- Easy to program
  You can learn from existing code. Many useful resources on the net.

## Supported hardware architectures

2.6.31 status

What's the current version?

2.6.38

- See the …/arch/ directory in the kernel sources
- Minimum: 32 bit processors, with or without MMU, and gcc support
- 32 bit architectures (…/arch/ subdirectories)
  **arm**, avr32, blackfin, cris, frv, h8300, m32r, m68k, m68knommu, microblaze, mips, mn10300, parisc, s390, sparc, um, xtensa
- 64 bit architectures:
  alpha, ia64, sparc64
- 32/64 bit architectures
  powerpc, x86, sh

How did I find it?

- Find details in kernel sources: …/arch/<arch>/Kconfig or …/Documentation/<arch>/

## Supported hardware architectures

2.6.31 status

What's the current version?

3.5.4

- See the …/arch/ directory in the kernel sources
- Minimum: 32 bit processors, with or without MMU, and gcc support
- 32 bit architectures (…/arch/ subdirectories)
  **arm**, avr32, blackfin, cris, frv, h8300, m32r, m68k, m68knommu, microblaze, mips, mn10300, parisc, s390, sparc, um, xtensa
- 64 bit architectures:
  alpha, ia64, sparc64
- 32/64 bit architectures
  powerpc, x86, sh

How did I find it?

kernel.org

- Find details in kernel sources: …/arch/<arch>/Kconfig, …/arch/<arch>/README, or …/Documentation/<arch>/


## System calls

- The main interface between the kernel and userspace is the set of system calls
- About ~300 system calls that provides the main kernel services
- This interface is stable over time: only new system calls can be added by the kernel developers
- This system call interface is wrapped by the C library, and userspace applications usually never make a system call directly but rather use the corresponding C library function

## System calls

- The main interface between the kernel and userspace is the set of system calls
- About ~300 system calls that provides the main kernel services
- i
  calls can be ad
- This system ca
  library, and use
  make a system
  corresponding

What are examples?

File and device operations, networking operations, inter-process communication, process management, memory mapping, timers, threads, synchronization primitives, etc.

## Virtual filesystems

- Linux makes system and kernel information available in user-space through virtual filesystems (virtual files not existing on any real storage). No need to know kernel programming to access such information!
- Mounting /proc:
  sudo mount -t proc none /proc
- Mounting /sys:
  sudo mount -t sysfs none /sys

Filesystem type — Raw device or filesystem image. In the case of virtual filesystems, any string is fine — Mount point

## /proc details

A few examples:

- /proc/cpuinfo: processor information
- /proc/meminfo: memory status
- /proc/version: kernel version and build information
- /proc/cmdline: kernel command line
- /proc/<pid>/environ: calling environment
- /proc/<pid>/cmdline: process command line

Lots of details about the /proc interface are available in Documentation/filesystems/proc.txt (almost 2000 lines) in the kernel sources.

## ... and many more! See by yourself!

```
beagle$ ls /proc
1/    217/  30/   40/  asound/       fb           misc          sysrq-trigger
10/   218/  31/   41/  buddyinfo     filesystems  modules       sysvipc/
11/   219/  312/  42/  bus/          fs/          mounts@       timer_list
12/   22/   32/   43/  cgroups       interrupts   net@          timer_stats
13/   226/  324/  44/  cmdline       iomem        pagetypeinfo  tty/
14/   228/  33/   45/  config.gz     ioports      partitions    uptime
15/   23/   336/  5/   consoles      irq/         sched_debug   version
16/   232/  34/   53/  cpu/          kallsyms     schedstat     vmallocinfo
17/   235/  35/   6/   cpuinfo       key-users    scsi/         vmstat
18/   236/  351/  62/  crypto        kmsg         self@         zoneinfo
19/   24/   36/   64/  device-tree/  kpagecount   slabinfo
196/  246/  37/   69/  devices       kpageflags   softirqs
199/  254/  390/  7/   diskstats     loadavg      stat
2/    262/  395/  8/   driver/       locks        swaps
21/   3/    4/    9/   execdomains  meminfo      sys/
```

## Embedded Linux usage

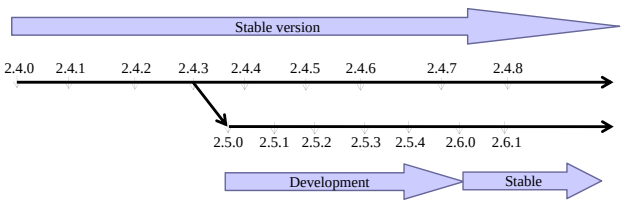
Kernel overview

Linux versioning scheme and development process

## Until 2.6 (1)

- One stable major branch every 2 or 3 years
- Identified by an even middle number
- Examples: 1.0, 2.0, 2.2, 2.4
- One development branch to integrate new functionalities and major changes
- Identified by an odd middle number
- Examples: 2.1, 2.3, 2.5
- After some time, a development version becomes the new base version for the stable branch
- Minor releases once in while: 2.2.23, 2.5.12, etc.

## Until 2.6 (2)




Note: in reality, many more minor versions exist inside the stable and development branches

## Changes since Linux 2.6 (1)

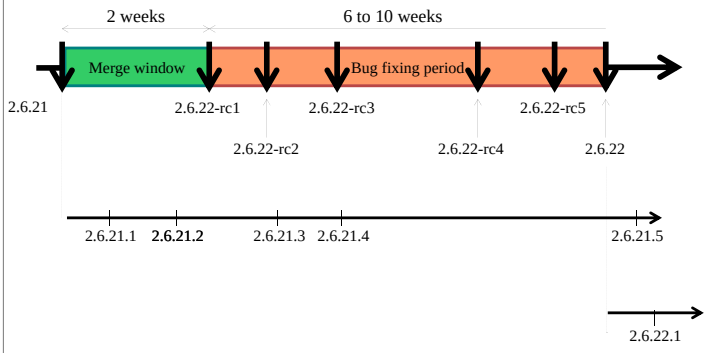
- Since 2.6.0, kernel developers have been able to introduce lots of new features one by one on a steady pace, without having to make major changes in existing subsystems.
- Opening a new Linux 2.7 (or 2.9) development branch will be required only when Linux 2.6 is no longer able to accommodate key features without undergoing traumatic changes.
- Thanks to this, more features are released to users at a faster pace.

## Changes since Linux 2.6 (2)

Since 2.6.14, the kernel developers agreed on the following development model:

- After the release of a 2.6.x version, a two-weeks merge window opens, during which major additions are merged.
- The merge window is closed by the release of test version 2.6.(x+1)-rc1
- The bug fixing period opens, for 6 to 10 weeks.
- At regular intervals during the bug fixing period, 2.6.(x+1)-rcY test versions are released.
- When considered sufficiently stable, kernel 2.6.(x+1) is released, and the process starts again.

## Merge and bug fixing windows




Bug fix updates

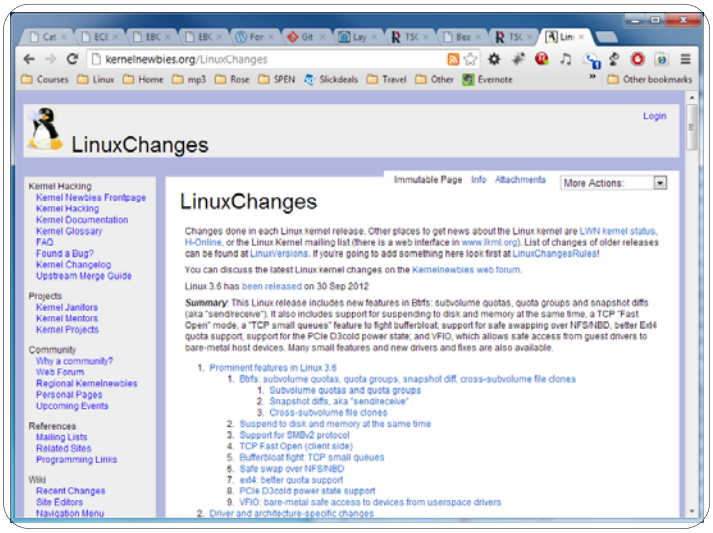
## More stability for the 2.6 kernel tree

- Issue: bug and security fixes only released for last (or last two) stable kernel versions (like 2.6.16 and 2.6.17), and of course by distributions for the exact version that you're using.
- Some people need to have a recent kernel, but with long term support for security updates.
- You could get long term support from a commercial embedded Linux provider.
- You could reuse sources for the kernel used in Ubuntu Long Term Support releases (5 years of free security updates).
- You could choose Linux 2.6.27 for your project, which will be maintained by kernel.org for a long time, unlike other versions.

## What's new in each Linux release?

```
commit 3c92c2ba33cd7d666c5f83cc32aa590e794e91b0
Author: Andi Kleen <ak@suse.de>
Date:   Tue Oct 11 01:28:33 2005 +0200

    [PATCH] i386: Don't discard upper 32bits of HWCR on K8

    Need to use long long, not long when RMWing a MSR. I think
    it's harmless right now, but still should be better fixed
    if AMD adds any bits in the upper 32bit of HWCR.

    Bug was introduced with the TLB flush filter fix for i386

    Signed-off-by: Andi Kleen <ak@suse.de>
    Signed-off-by: Linus Torvalds <torvalds@osdl.org>
...
```


- The official list of changes for each Linux release is just a huge list of individual patches!
- Very difficult to find out the key changes and to get the global picture out of individual changes.
- Fortunately, a summary of key changes with enough details is available on http://wiki.kernelnewbies.org/LinuxChanges




# Embedded Linux kernel usage

## Embedded Linux kernel usage

Michael Opdenacker
Thomas Petazzoni
**Free Electrons**



Latest update: 10/2/2012,
Document sources, updates and translations:
http://free-electrons.com/docs/kernel-usage
Corrections, suggestions, contributions and translations are welcome!

# Contents



# Embedded Linux usage

Compiling and booting Linux

Linux kernel sources

# Location of kernel sources

- The official version of the Linux kernel, as released by Linus Torvalds is available at http://www.kernel.org
  - This version follows the well-defined development model of the kernel
  - However, it may not contain the latest development from a specific area, due to the organization of the development model and because features in development might not be ready for mainline inclusion

Like omap

- Many kernel sub-communities maintain their own kernel, with usually newer but less stable features
  - Architecture communities (ARM, MIPS, PowerPC, etc.), device drivers communities (I2C, SPI, USB, PCI, network, etc.), other communities (real-time, etc.)
  - They generally don't release official versions, only development trees are available

# Linux kernel size (1)

- Linux 2.6.31 sources:
  Raw size: 350 MB (30,900 files, approx 12,000,000 lines)
  gzip compressed tar archive: 75 MB
  bzip2 compressed tar archive: 59 MB (better)
  lzma compressed tar archive: 49 MB (best)
- Minimum Linux 2.6.29 compiled kernel size with CONFIG_EMBEDDED,
  for a kernel that boots a QEMU PC (IDE hard drive, ext2 filesystem, ELF executable support):
  532 KB (compressed), 1325 KB (raw)
- Why are these sources so big?
  Because they include thousands of device drivers, many network protocols, support many architectures and filesystems...
- The Linux core (scheduler, memory management...) is pretty small!

## Linux kernel size (1)

- Linux 2.6.31 sources:
  Raw size: 350 MB (30,900 files, approx 12,000,000 lines)
  gzip compressed tar archive: 75 MB
  bzip2 compressed tar archive: 59 MB (better)
  lzma compressed tar archive: 49 MB (best)
- Linux 2.6.32 sources:

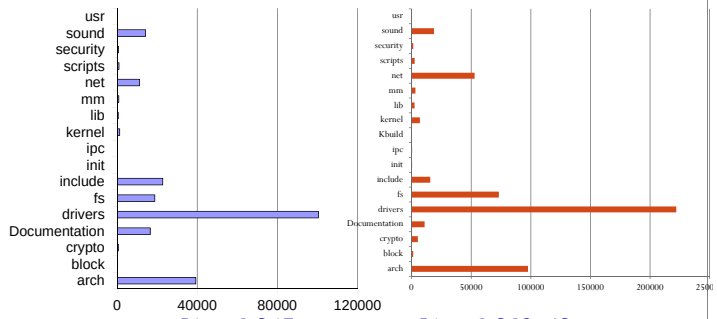
  1.3G
- Linux 3.0.9 sources:

  1.6G
- Linux 3.2.18 sources:

  721M, 48K files

## Linux kernel size (2)

Size of Linux source directories (KB)




Linux 2.6.17
Measured with:
du -s --apparent-size

Linux 2.6.29-r46
Measured with:
du -s --apparent-size

## Getting Linux sources

- Full tarballs
  - Contain the complete kernel sources
  - Long to download and uncompress, but must be done at least once
  - Example:
    http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.7.tar.bz2

## Getting Linux sources

- Incremental patches between versions
  - It assumes you already have a base version and you apply the correct patches in the right order
  - Quick to download and apply
  - Examples
    http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.bz2 (2.6.13 to 2.6.14)
    http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.7.bz2 (2.6.14 to 2.6.14.7)
- All previous kernel versions are available in http://kernel.org/pub/linux/kernel/

## Getting Linux sources

- **`git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git linux-2.6`**

## Getting Linux sources

- bitbake
  - **`cd ${OETREE}/build`**
  - **`bitbake -c clean linux-omap-2.6.28`**
  - **`bitbake -f -c compile linux-omap-2.6.28`**

## Top-Level Source Directory

```
arch/           firmware/   kernel/     samples/
block/          fs/         lib/        scripts/
crypto/         include/    mm/         security/
Documentation/  init/       net/        sound/
drivers/        ipc/        patches/    usr/
                                        virt/
```

## Using the patch command

The patch command applies changes to files in the current directory:

- Making changes to existing files
- Creating or deleting files and directories

patch usage examples:

- patch -p<n> < diff_file
- cat diff_file | patch -p<n>
- bzcat diff_file.bz2 | patch -p<n>
- zcat diff_file.gz | patch -p<n>

n: number of directory levels to skip in the file paths

You can reverse a patch with the -R option

You can test a patch with the --dry-run option

## Anatomy of a patch file

A patch file is the output of the diff command

```
diff -Nru a/Makefile b/Makefile                    <- diff command line
--- a/Makefile  2005-03-04 09:27:15 -08:00         <- File date info
+++ b/Makefile  2005-03-04 09:27:15 -08:00
@@ -1,7 +1,7 @@                                      <- Line numbers in files
 VERSION = 2                                       <- Context info: 3 lines before the change
 PATCHLEVEL = 6                                    <- Useful to apply a patch when line numbers changed
 SUBLEVEL = 11
-EXTRAVERSION =                                    <- Removed line(s) if any
+EXTRAVERSION = .1                                 <- Added line(s) if any
 NAME=Woozy Numbat                                 <- Context info: 3 lines after the change

 # *DOCUMENTATION*
```

## Applying a Linux patch

Linux patches...

- Always to apply to the x.y.<z-1> version
  Downloadable in gzip
  and bzip2 (much smaller) compressed files.
- Always produced for n=1
  (that's what everybody does... do it too!)
- Linux patch command line example:
  cd linux-2.6.13
  bzcat ../patch-2.6.14.bz2 | patch -p1
  bzcat ../patch-2.6.14.7.bz2 | patch -p1
  cd ..; mv linux-2.6.13 linux-2.6.14.7
- Keep patch files compressed: useful to check their signature later.
  You can still view (or even edit) the uncompressed data with vim:
  vim patch-2.6.14.bz2 (on the fly (un)compression)

You can make patch 30% faster by using -sp1 instead of -p1 (silent)

Tested on patch-2.6.23.bz2

## Ketchup - Easy access to kernel sources

http://www.selenic.com/ketchup/

- Makes it easy to download a specific version.
  Takes care of downloading and applying patches
- Example: downloading the latest kernel version
  > mkdir linux-2.6.31
  > cd linux-2.6.31
  > ketchup -G 2.6-tip
  None -> 2.6.31.6
  Downloading linux-2.6.31.6.tar.bz2
  Unpacking linux-2.6.31.6.tar.bz2
- Now getting back to an older version (from the same directory)
  > ketchup -G 2.6.30
  2.6.31.6 -> 2.6.30
  Downloading patch-2.6.31.6.bz2
  Applying patch-2.6.31.bz2 -R
  Downloading patch-2.6.30.bz2
  Applying patch-2.6.30.bz2 -R

The -G option of ketchup disables source signature checking.

See http://kernel.org/signature.html for details about enabling kernel source integrity checking.

## Practical lab – Kernel sources

- Get the sources
- Apply patches

# Embedded Linux usage

Compiling and booting Linux

Kernel configuration

## Kernel configuration

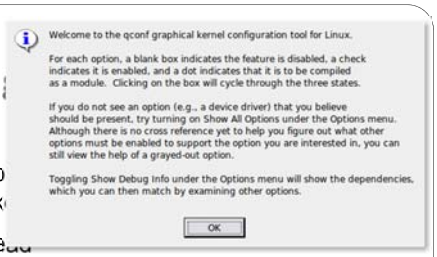
Defines what features to include in the kernel:

- Stored in the `.config` file at the root of kernel sources.
  - Simple text file
- Most useful commands to create this config file:
  `make [xconfig|gconfig|menuconfig|oldconfig]`
- To modify a kernel in a GNU/Linux distribution: the configuration files are usually released in /boot/, together with kernel images: /boot/config-2.6.17-11-generic

```
beagle$ ls -F /boot
uEnv.txt*  uImage@  uImage-3.2.25
```

## make xconfig

`make xconfig`

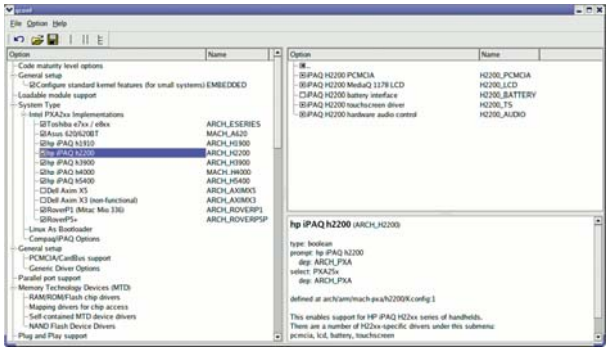



- The most commo
  to configure the k
- Make sure you re
  help -> introduction: useful options!
- File browser: easier to load configuration files
- New search interface to look for parameters
- Required Debian / Ubuntu packages:

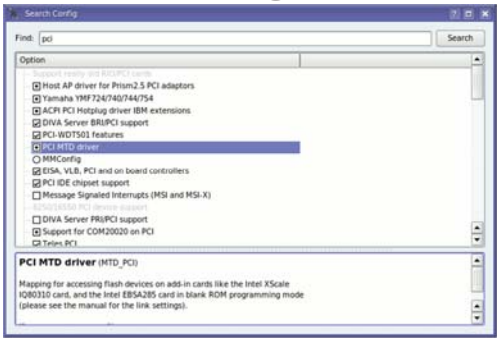
```
host$ sudo apt-get update
host$ sudo apt-get install libqt4-dev
```

libqt3-mt-dev, g++

## make xconfig screenshot



## make xconfig search interface

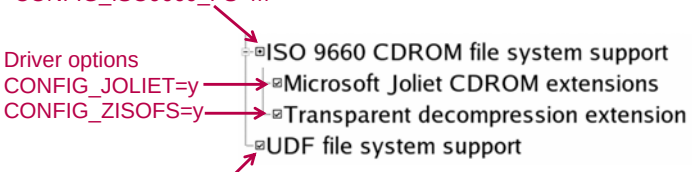



Looks for a keyword in the description string

Allows to select or unselect found parameters.

## Kernel configuration options

Compiled as a module (separate file)
CONFIG_ISO9660_FS=m




Driver options
CONFIG_JOLIET=y
CONFIG_ZISOFS=y

Compiled statically into the kernel
CONFIG_UDF_FS=y

## Corresponding .config file excerpt

```
#
# CD-ROM/DVD Filesystems
#
CONFIG_ISO9660_FS=m
CONFIG_JOLIET=y
CONFIG_ZISOFS=y
CONFIG_UDF_FS=y
CONFIG_UDF_NLS=y

#
# DOS/FAT/NT Filesystems
#
# CONFIG_MSDOS_FS is not set
# CONFIG_VFAT_FS is not set
CONFIG_NTFS_FS=m
# CONFIG_NTFS_DEBUG is not set
CONFIG_NTFS_RW=y
```

Section name (helps to locate settings in the interface)

All parameters are prefixed with CONFIG_

## Kernel option dependencies

- There are dependencies between kernel options
- For example, enabling a network driver requires the network stack to be enabled
- Two types of dependencies
  - *depends on* dependencies. In this case, option A that depends on option B is not visible until option B is enabled
  - *select* dependencies. In this case, with option A depending on option B, when option A is enabled, option B is automatically enabled
  - *make xconfig* allows to see all options, even those that cannot be selected because of missing dependencies. In this case, they are displayed in gray

## make gconfig

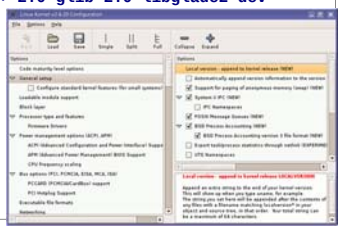
make gconfig

New GTK based graphical configuration interface. Functionality similar to that of make xconfig.

Just lacking a search functionality.

Required Debian packages:

```
host$ sudo apt-get install gtk+-2.0 glib-2.0 libglade2-dev
```



## make menuconfig

make menuconfig

Useful when no graphics are available. Pretty convenient too!

Same interface found in other tools: BusyBox, buildroot...

Required Debian packages: libncurses-dev

## make oldconfig

make oldconfig

- Needed very often!
- Useful to upgrade a .config file from an earlier kernel release
- Issues warnings for configuration parameters that no longer exist in the new kernel.
- Asks for values for new parameters

If you edit a .config file by hand, it's strongly recommended to run make oldconfig afterwards!

## make allnoconfig

make allnoconfig

- Only sets strongly recommended settings to y.
- Sets all other settings to n.
- Very useful in embedded systems to select only the minimum required set of features and drivers.
- Much more convenient than unselecting hundreds of features one by one!

## Undoing configuration changes

A frequent problem:

- After changing several kernel configuration settings, your kernel no longer works.
- If you don't remember all the changes you made, you can get back to your previous configuration:
  > cp .config.old .config
- All the configuration interfaces of the kernel (xconfig, menuconfig, allnoconfig...) keep this .config.old backup copy.

```
host$ git diff .config
host$ git checkout .config
```

## Configuration per architecture

- The set of configuration options is architecture dependent
  - Some configuration options are very architecture-specific
  - Most of the configuration options (global kernel options, network subsystem, filesystems, most of the device drivers) are visible in all-architecture
- By default, the kernel build system assumes that the kernel is being built for the host architecture, i.e native compilation
- The architecture is not defined inside the configuration, but at an higher level
- We will see later how to override this behaviour, to allow the configuration of kernels for a different architecture

## Overview of kernel options (1)

- General setup
  - *Prompt for development/incomplete code* allows to be able to enable drivers or features that are not considered as completely stable yet
  - *Automatically append version information to the version string* allows to concatenate an arbitrary string to the kernel version that an user can get using uname -r. Very useful for support!
  - *Support for swap*, can usually be disabled on most embedded devices
  - *Configure standard kernel features (for small systems)* allows to remove features from the kernel to reduce its size. Powerful, use with care!

## Overview of kernel options (2)

- Loadable module support
  - Allows to enable or completely disable module support. If your system doesn't need kernel modules, best to disable since it saves a significant amount of space and memory
- Enable the block layer
  - If CONFIG_EMBEDDED is enabled, the block layer can be completely removed. Embedded systems using only Flash storage can safely disable the block layer
- Processor type and features (x86) or System type (ARM) or CPU selection (MIPS)
  - Allows to select the CPU or machine for which the kernel must be compiled
  - On x86, only optimization-related, on other architectures very important since there's no compatibility

## Overview of kernel options (3)

- Kernel features
  - Tickless system, which allows to disable the regular timer tick and use on-demand ticks instead. Improves power savings
  - High resolution timer support. By default, the resolution of timer is the tick resolution. With high resolution timers, the resolution is as precise as the hardware can give
  - Preemptible kernel enables the preemption inside the kernel code (the userspace code is always preemptible). See our real-time presentation for details
- Power management
  - Global power management option needed for all power management related features
  - Suspend to RAM, CPU frequency scaling, CPU idle control, suspend to disk

## Overview of kernel options (4)

- Networking support
  - The network stack
  - Networking options
    - Unix sockets, needed for a form of inter-process communication
    - TCP/IP protocol with options for multicast, routing, tunneling, Ipsec, Ipv6, congestion algorithms, etc.
    - Other protocols such as DCCP, SCTP, TIPC, ATM
    - Ethernet bridging, QoS, etc.
  - Support for other types of network
    - CAN bus, Infrared, Bluetooth, Wireless stack, WiMax stack, etc.

## Overview of kernel options (5)

- Device drivers
  - MTD is the subsystem for Flash (NOR, NAND, OneNand, battery-backed memory, etc.)
  - Parallel port support
  - Block devices, a few misc block drivers such as loopback, NBD, etc.
  - ATA/ATAPI, support for IDE disk, CD-ROM and tapes. A new stack exists
  - SCSI
    - The SCSI core, needed not only for SCSI devices but also for USB mass storage devices, SATA and PATA hard drives, etc.
    - SCSI controller drivers

## Overview of kernel options (6)

- Device drivers (cont)
  - SATA and PATA, the new stack for hard disks, relies on SCSI
  - RAID and LVM, to aggregate hard drivers and do replication
  - Network device support, with the network controller drivers. Ethernet, Wireless but also PPP
  - Input device support, for all types of input devices: keyboards, mices, joysticks, touchscreens, tablets, etc.
  - Character devices, contains various device drivers, amongst them
    - serial port controller drivers
    - PTY driver, needed for things like SSH or telnet
  - I2C, SPI, 1-wire, support for the popular embedded buses
  - Hardware monitoring support, infrastructure and drivers for thermal sensors

## Overview of kernel options (7)

- Device drivers (cont)
  - Watchdog support
  - Multifunction drivers are drivers that do not fit in any other category because the device offers multiple functionality at the same time
  - Multimedia support, contains the V4L and DVB subsystems, for video capture, webcams, AM/FM cards, DVB adapters
  - Graphics support, infrastructure and drivers for framebuffers
  - Sound card support, the OSS and ALSA sound infrastructures and the corresponding drivers
  - HID devices, support for the devices that conform to the HID specification (Human Input Devices)

## Overview of kernel options (8)

- Device drivers (cont)
  - USB support
    - Infrastructure
    - Host controller drivers
    - Device drivers, for devices connected to the embedded system
    - Gadget controller drivers
    - Gadget drivers, to let the embedded system act as a mass-storage device, a serial port or an Ethernet adapter
  - MMC/SD/SDIO support
  - LED support
  - Real Time Clock drivers
  - Voltage and current regulators
  - Staging drivers, crappy drivers being cleaned up

## Overview of kernel options (9)

- For some categories of devices the driver is not implemented inside the kernel
  - Printers
  - Scanners
  - Graphics drivers used by X.org
  - Some USB devices
- For these devices, the kernel only provides a mechanism to access the hardware, the driver is implemented in userspace

## Overview of kernel options (10)

- File systems
  - The common Linux filesystems for block devices: ext2, ext3, ext4
  - Less common filesystems: XFS, JFS, ReiserFS, GFS2, OCFS2, Btrfs
  - CD-ROM filesystems: ISO9660, UDF
  - DOS/Windows filesystems: FAT and NTFS
  - Pseudo filesystems: proc and sysfs
  - Miscellanous filesystems, with amongst other Flash filesystems such as JFFS2, UBIFS, SquashFS, cramfs
  - Network filesystems, with mainly NFS and SMB/CIFS
- Kernel hacking
  - Debugging features useful for kernel developers

# make help

make help

- Lists all available make targets
- Useful to get a reminder, or to look for new or advanced options!

# Make help

```
make help
Cleaning targets:
  clean                                   - Remove most generated files but keep the config and
                    enough build support to build external modules
  mrproper                - Remove all generated files + config + various backup files
  distclean               - mrproper + remove editor backup and patch files

Configuration targets:
  config                  - Update current config utilising a line-oriented program
  nconfig                 - Update current config utilising a ncurses menu based program
  menuconfig              - Update current config utilising a menu based program
  xconfig                 - Update current config utilising a QT based front-end
  gconfig                 - Update current config utilising a GTK based front-end
  oldconfig               - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
  localyesconfig  - Update current config converting local mods to core
  silentoldconfig - Same as oldconfig, but quietly, additionally update deps
  defconfig               - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig             - New config where all options are answered with no
  allyesconfig            - New config where all options are accepted with yes
  allmodconfig            - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig              - New config with random answer to all options
  listnewconfig   - List new options
  oldnoconfig     - Same as silentoldconfig but set new symbols to n (unset)

Other generic targets:
  all                                     - Build all targets marked with [*]
* vmlinux                - Build the bare kernel
* modules                 - Build all modules
  modules_install - Install all modules to INSTALL_MOD_PATH (default: /)
  firmware_install- Install all firmware to INSTALL_FW_PATH
                    (default: $(INSTALL_MOD_PATH)/lib/firmware)
  dir/            - Build all files in dir and below
```

# Make help

```
Configuration targets:
  config        - Update current config utilising a line-oriented program
  nconfig       - Update current config utilising a ncurses menu based program
  menuconfig    - Update current config utilising a menu based program
  xconfig       - Update current config utilising a QT based front-end
  gconfig       - Update current config utilising a GTK based front-end
  oldconfig     - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
  localyesconfig  - Update current config converting local mods to core
  silentoldconfig - Same as oldconfig, but quietly, additionally update deps
  defconfig       - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig     - New config where all options are answered with no
  allyesconfig    - New config where all options are accepted with yes
  allmodconfig    - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig      - New config with random answer to all options
  listnewconfig   - List new options
  oldnoconfig     - Same as silentoldconfig but set new symbols to n (unset)
```

# Embedded Linux usage

Compiling and installing the kernel for the host system

# Compiling and installing the kernel

Compiling step

- make

You can speed up compiling by running multiple compile jobs in parallel, especially if you have multiple CPU cores.

Example: make -j 4

MAY1

Slide 71

MAY1 How do you build for the target?

Mark A. Yoder, 12/22/2009

## Compiling and installing the kernel

Compiling step

- make

You can speed up compiling by running multiple compile jobs in parallel, especially if you have multiple CPU cores.

Example: make -j 4

Install steps

Will install the kernel and the modules on your host system

MAY2

- sudo make install
- sudo make modules_install



## Kernel cleanup targets

- Clean-up generated files (to force re-compiling drivers):
  make clean
- Remove all generated files. Needed when switching from one architecture to another
  **Caution: also removes your .config file!**
  make mrproper
- Also remove editor backup and patch reject files:
  (mainly to generate patches):
  make distclean

## Generated files

Created when you run the make command. The kernel is in fact a single binary image, nothing more !

- …/vmlinux
  Raw Linux kernel image, non compressed.
- …/arch/<arch>/boot/zImage (default image on arm)
  zlib compressed kernel image
- …/arch/<arch>/boot/bzImage (default image on x86)
  Also a zlib compressed kernel image.
  Caution: bz means "big zipped" but not "bzip2 compressed"!

News: new compression formats are now available since 2.6.30: lzma and bzip2. Free Electrons also contributed lzo support (very fast decompression).

## Files created by make install

- /boot/vmlinuz-<version>
  Compressed kernel image. Same as the one in /arch/<arch>/boot
- /boot/System.map-<version>
  Stores kernel symbol addresses
- /boot/config-<version>
  Kernel configuration for this version

Don't Use

## Files created by make modules_install

/lib/modules/<version>/: Kernel modules + extras

- kernel/
  Module .ko (Kernel Object) files, in the same directory structure as in the sources.
- modules.alias
  Module aliases for module loading utilities. Example line:
  alias sound-service-?-0 snd_mixer_oss
- modules.dep
  Module dependencies
- modules.symbols
  Tells which module a given symbol belongs to.

Don't Use

All the files in this directory are text files.
Don't hesitate to have a look by yourself!

## The Details

To understand a system one must first understand it parts.

--Chris Hallinan

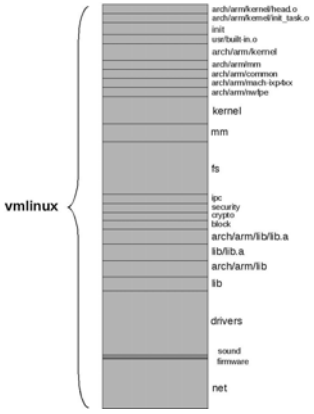
## Link Stage: vmlinux

```
$ arm-angstrom-linux-gnueabi-ld -EB -p --no-undefined -X -o vmlinux
-T arch/arm/kernel/vmlinux.lds \
arch/arm/kernel/head.o \
arch/arm/kernel/init_task.o \
init/built-in.o \
--start-group \
usr/built-in.o \
arch/arm/kernel/built-in.o \
arch/arm/mm/built-in.o \
arch/arm/common/built-in.o \
arch/arm/mach-ixp4xx/built-in.o \
arch/arm/nwfpe/built-in.o \
kernel/built-in.o \
mm/built-in.o \
fs/built-in.o \
ipc/built-in.o \
security/built-in.o \
crypto/built-in.o \
block/built-in.o \
arch/arm/lib/lib.a \
lib/lib.a \
arch/arm/lib/built-in.o \
lib/built-in.o \
drivers/built-in.o \
sound/built-in.o \
firmware/built-in.o \
net/built-in.o \
–end-group \
.tmp_kallsyms2.o \
```

Look in **~/BeagleBoard/oe/build/tmp-angstrom_v2012_05-eglibc/sysroots/*x86_64-linux*/usr/bin/armv7a-angstrom-linux-gnueabi**

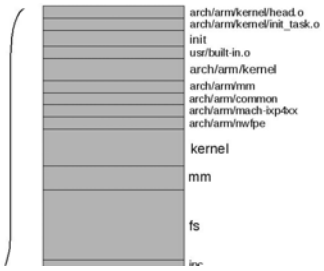
```
host$ cd ~/BeagleBoard/oe/build/tmp-angstrom_v2012_05-eglibc/sysroots/x86_64-linux/usr/bin/armv7a-angstrom-linux-gnueabi
host$ ls
arm-angstrom-linux-gnueabi-addr2line  arm-angstrom-linux-gnueabi-ld
arm-angstrom-linux-gnueabi-ar         arm-angstrom-linux-gnueabi-lsmod
arm-angstrom-linux-gnueabi-as         arm-angstrom-linux-gnueabi-modinfo
arm-angstrom-linux-gnueabi-c++        arm-angstrom-linux-gnueabi-modprobe
arm-angstrom-linux-gnueabi-c++filt    arm-angstrom-linux-gnueabi-nm
arm-angstrom-linux-gnueabi-cpp        arm-angstrom-linux-gnueabi-objcopy
arm-angstrom-linux-gnueabi-depmod     arm-angstrom-linux-gnueabi-objdump
arm-angstrom-linux-gnueabi-g++        arm-angstrom-linux-gnueabi-ranlib
arm-angstrom-linux-gnueabi-gcc        arm-angstrom-linux-gnueabi-readelf
arm-angstrom-linux-gnueabi-gcc-4.5.4  arm-angstrom-linux-gnueabi-rmmod
arm-angstrom-linux-gnueabi-gccbug     arm-angstrom-linux-gnueabi-size
arm-angstrom-linux-gnueabi-gcov       arm-angstrom-linux-gnueabi-strings
arm-angstrom-linux-gnueabi-gprof      arm-angstrom-linux-gnueabi-strip
arm-angstrom-linux-gnueabi-insmod
```

## vmlinux image components




## Compare the two

```
$ arm-linux-ld -EB -p --no-undefined -X -o vmlinux
-T arch/arm/kernel/vmlinux.lds \
arch/arm/kernel/head.o \
arch/arm/kernel/init_task.o \
init/built-in.o \
--start-group \
usr/built-in.o \
arch/arm/kernel/built-in.o \
arch/arm/mm/built-in.o \
arch/arm/common/built-in.o \
arch/arm/mach-ixp4xx/built-in.o \
arch/arm/nwfpe/built-in.o \
kernel/built-in.o \
mm/built-in.o \
fs/built-in.o \
```




## vmlinux Image Components Description

Table 4-1

***vmlinux Image Components Description***

| Component | Description |
|---|---|
| arch/arm/kernel/head.o | Kernel architecture-specific startup code. |
| arch/arm/kernel/init_task.o | Initial thread and task structs required by kernel. |
| init/built-in.o | Main kernel initialization code. See Chapter 5. |
| usr/built-in.o | Built-in initramfs image. See Chapter 6. |
| arch/arm/kernel/built-in.o | Architecture-specific kernel code. |
| arch/arm/mm/built-in.o | Architecture-specific memory-management code. |
| arch/arm/common/built-in.o | Architecture-specific generic code. Varies by architecture |
| arch/arm/mach-ixp4xx/built-in.o | Machine-specific code, usually initialization. |
| arch/arm/nwfpe/built-in.o | Architecture-specific floating point–emulation code. |
| kernel/built-in.o | Common components of the kernel itself. |
| mm/built-in.o | Common components of memory-manage- |