05-1 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).

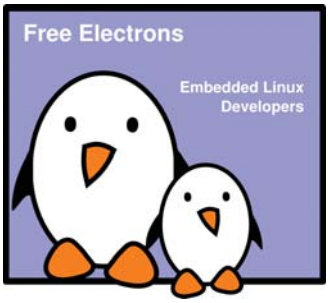
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

Free Electrons

Linux kernel introduction

Michael Opdenacker
Thomas Petazzoni
Free Electrons



Latest update: 10/7/2014,
Document sources, updates and translations:
http://free-electrons.com/docs/kernel-intro
Corrections, suggestions, contributions and translations are welcome!

Embedded Linux driver development

Kernel overview
Linux features

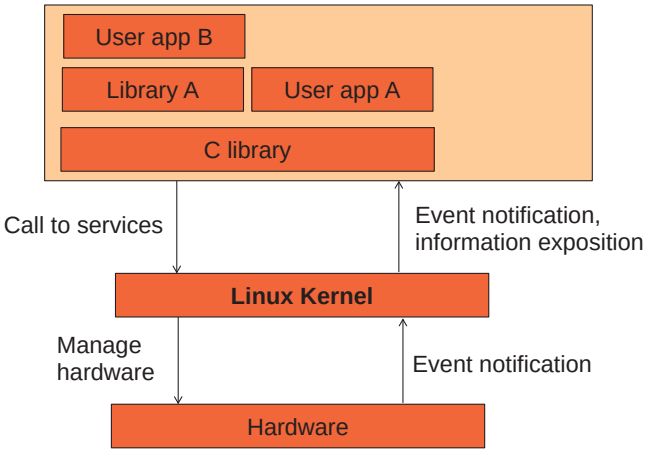
History

- 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 kernel key features

- Portability and hardware support. Runs on most architecture
- 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

Linux kernel in the system



Supported hardware architectures

What's the current version?

3.16.3

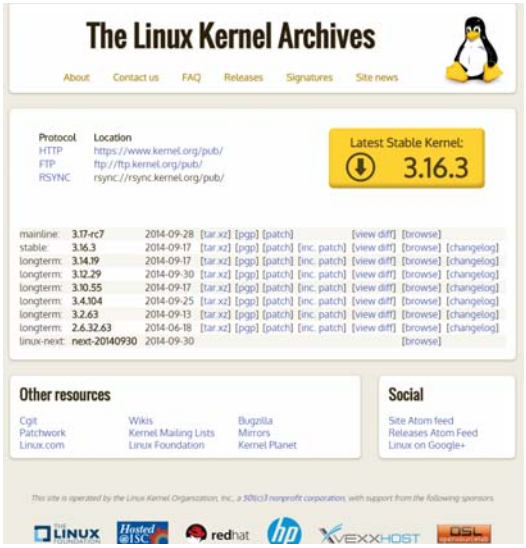
2.6.31 status

- 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

How did I find it?

kernel.org

- 32/64 bit architectures
  powerpc, x86, sh
- Find details in kernel sources: …/arch/<arch>/Kconfig or …/Documentation/<arch>/



System calls

What are examples?

- 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
  calls can be ad
- This system ca
  library, and use
  make a system
  corresponding

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

Pseudo filesystems

- Linux makes system and kernel information available in user space through **pseudo filesystems**, (also called **virtual filesystems**
- Pseudo filesystems allow applications to see directories and files that do not exist on any real storage: they are created and updated on the fly by the kernel
- The two most important pseudo file systems are
  - **`proc`**, usually mounted on **`/proc`**: Operating system related information (processes, memory management parameters...)
  - **`sysfs`**, usually mounted on **`/sys`**: Representation of the system as a set of devices and buses. Information about these devices.

/proc details

A few examples:

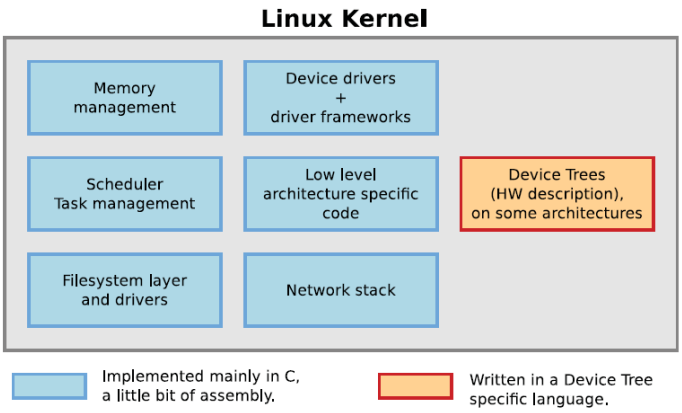
- /proc/cpuinfo: processor information
- /proc/meminfo: memory status
- /proc/version: kernel version and build information
- /proc/cmdline: kernel command line
- /proc/<pid>/environ: calling environment
- /proc/<pid>/cmdline: process command line

Lots of details about the /proc interface are available in Documentation/filesystems/proc.txt (some 1700 lines) in the kernel sources.

... and many more! See by yourself!

```
beagle$ ls -F /proc
1/    16/   36/   45/   75/        cpuinfo       kmsg          slabinfo
10/   17/   38/   46/   76/        crypto        kpagecount    softirqs
101/  18/   39/   5/    79/        device-tree/  kpageflags    stat
11/   19/   40/   53/   8/         devices       loadavg       swaps
12/   2/    41/   530/  80/        diskstats     locks         sys/
127/  20/   412/  531/  81/        dri/          meminfo       sysrq-trigger
129/  21/   418/  533/  87/        driver/       misc          sysvipc/
13/   24/   42/   563/  88/        execdomains   modules       timer_list
138/  243/  429/  564/  9/         fb            mounts@       timer_stats
139/  244/  430/  565/  asound/    filesystems   mtd           tty/
14/   245/  437/  567/  buddyinfo  fs/          net@          uptime
140/  261/  440/  57/   bus/        interrupts    pagetypeinfo  version
142/  268/  442/  6/    cgroups     iomem         partitions    vmallocinfo
144/  27/   443/  69/   cmdline     ioports       sched_debug   vmstat
145/  3/    445/  7/    config.gz   irq/          schedstat     zoneinfo
151/  320/  447/  73/   consoles    kallsyms      scsi/
152/  345/  449/  74/   cpu/        key-users     self@
```

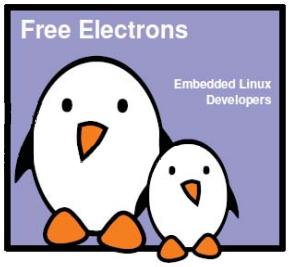
Inside the Linux kernel



Embedded Linux usage

Embedded Linux Kernel Usage

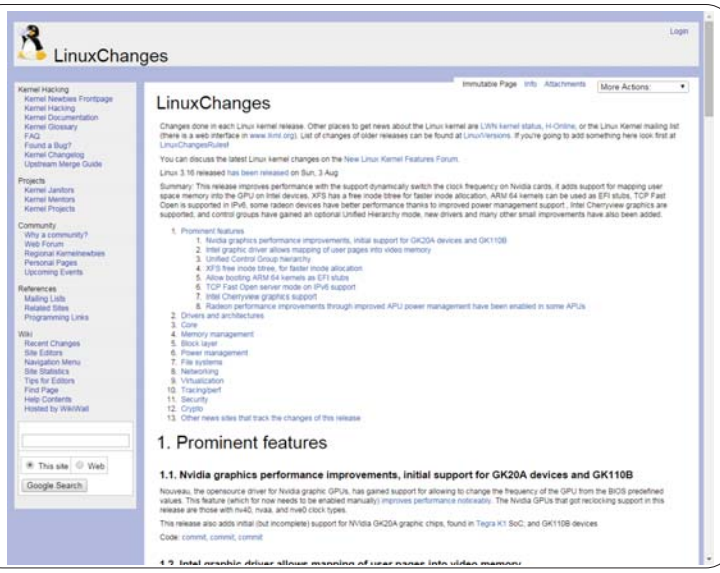
Free Electrons



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



Location of kernel sources

- The official versions of the Linux kernel, as released by Linus Torvalds, are available at **`http://www.kernel.org`**
  - These versions follow the development model of the kernel
  - However, they may not contain the latest development from a specific area yet. Some features in development might not be ready for mainline inclusion yet
- Many chip vendors supply their own kernel sources
  - Focusing on hardware support first
  - Can have a very important delta with mainline Linux
  - Useful only when mainline hasn't caught up yet
- 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.)
  - No official releases, only development trees are available.

Getting Linux sources

- The kernel sources are available from **`http://kernel.org/pub/linux/kernel`** as **full tarballs** (complete kernel sources) and **patches** (differences between two kernel versions).
- However, more and more people use the **`git`** version control system. Absolutely needed for kernel development!
  - Fetch the entire kernel sources and history
    `git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git` (21 minutes)
  - Create a branch that starts at a specific stable version
    `git checkout -b <name-of-branch> v3.11`
  - Web interface available at
    `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/`

The Robert C Nelson BBB Kernel

- http://eewiki.net/display/linuxonarm/BeagleBone+Black
- **`git clone git://github.com/RobertCNelson/bb-kernel.git`**
- `host$` **`cd bb-kernel`**
- `host$` **`git tag`** (This shows what versions can be checked out.)
- `host$` **`git checkout 3.8.13-bone67 -b 3.8.13-bone67`**
- `host$` **`./build_kernel.sh`**

Linux kernel size (1)

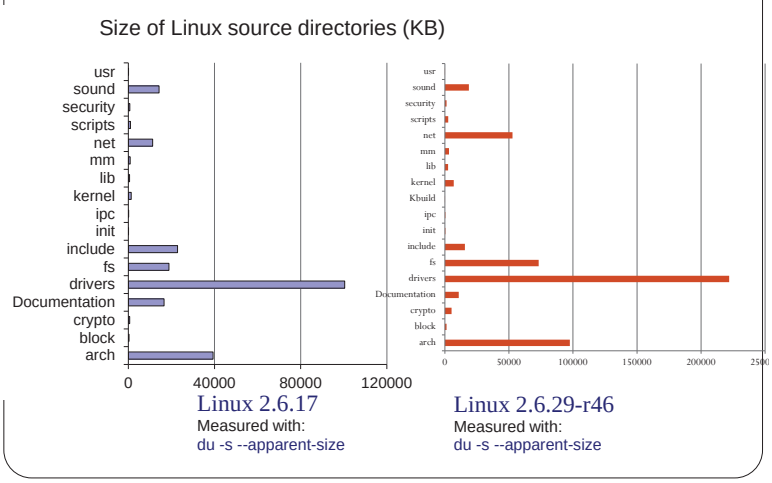
- Linux 3.10 sources:
  Raw size: 573 MB (43,000 files, ~15,800,000 lines)
  `gzip` compressed tar archive: 105 MB
  `bzip2` compressed tar archive: 83 MB (better)
  `xz` compressed tar archive: 69 MB (best)
- Minimum Linux 2.6.29 compiled kernel size with `CONFIG_EMBEDDED`, for a kernel that boots a QEMU PC (IDE hard drive, ext2 lesystem, 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 (2)

As of kernel version 3.10.

- `drivers/`: 49.4%
- `arch/`: 21.9%
- `fs/`: 6.0%
- `include/`: 4.7%
- `sound/`: 4.4%
- `Documentation/`: 4.0%
- `net/`: 3.9%
- `firmware/`: 1.0%
- `kernel/`: 1.0%
- `tools/`: 0.9%
- `scripts/`: 0.5%
- `mm/`: 0.5%
- `crypto/`: 0.4%
- `security/`: 0.4%
- `lib/`: 0.4%
- `block/`: 0.2%
- …

Linux kernel size (3)



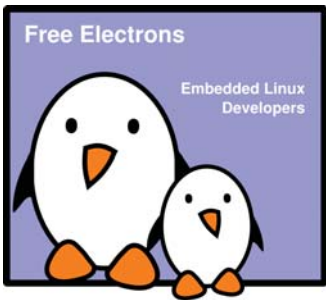
Linux 2.6.17
Measured with:
`du -s --apparent-size`

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

Kernel Source Code

Kernel Source Code

Michael Opdenacker
Thomas Petazzoni
Free Electrons



No C library

- The kernel has to be standalone and can't use user space code
- User space is implemented on top of kernel services, not the opposite
- Kernel code has to supply its own library implementations (string utilities, cryptography, uncompression …)
- So, you can't use standard C library functions in kernel code (`printf()`, `memset()`, `malloc()`,...).
- Fortunately, the kernel provides similar C functions for your convenience, like `printk()`, `memset()`, `kmalloc()`, ...

Kernel memory constraints

- No memory protection
- Accessing illegal memory locations result in (often fatal) kernel oopses
- Fixed size stack (8 or 4 KB). Unlike in user space, there's no way to make it grow
- Kernel memory can't be swapped out (for the same reasons)

Kernel Source Code

```
host$ cd ~/BeagleBoard/bb-kernel/KERNEL
host$ ls -F
arch/            Kbuild            REPORTING-BUGS
block/           Kconfig           samples/
COPYING          kernel/           scripts/
CREDITS          lib/              security/
crypto/          MAINTAINERS       sound/
Documentation/   Makefile          System.map
drivers/         mm/               tools/
firmware/        modules.builtin   usr/
fs/              modules.order     virt/
include/         Module.symvers    vmlinux*
init/            net/              vmlinux.o
ipc/             README
```

Linux sources structure 1/5

- `arch/<ARCH>`
  - Architecture specific code
  - `arch/<ARCH>/mach-<machine>`, machine/board specific code
  - `arch/<ARCH>/include/asm`, architecture-specific headers
  - `arch/<ARCH>/boot/dts`, Device Tree source files, for some architectures
- `block/`
  - Block layer core
- `COPYING`
  - Linux copying conditions (GNU GPL)
- `CREDITS`
  - Linux main contributors
- `crypto/`
  - Cryptographic libraries

Linux sources structure 2/5

- `Documentation/`
  - Kernel documentation. Don't miss it!
- `drivers/`
  - All device drivers except sound ones (usb, pci…)
- `firmware/`
  - Legacy: firmware images extracted from old drivers
- `fs/`
  - Filesystems (fs/ext3/, etc.)
- `include/`
  - Kernel headers
- `include/linux/`
  - Linux kernel core headers
- `include/uapi/`
  - User space API headers
- `init/`
  - Linux initialization (including main.c)
- `ipc/`
  - Code used for process communication

Linux sources structure 3/5

- `Kbuild`
  - Part of the kernel build system
- `Kconfig`
  - Top level description file for configuration parameters
- `kernel/`
  - Linux kernel core (very small!)
- `lib/`
  - Misc library routines (zlib, crc32…)
- `MAINTAINERS`
  - Maintainers of each kernel part. Very useful!
- `Makefile`
  - Top Linux Makele (sets arch and version)
- `mm/`
  - Memory management code (small too!)

Linux sources structure 4/5

- `net/`
  - Network support code (not drivers)
- `README`
  - Overview and building instructions
- `REPORTING-BUGS`
  - Bug report instructions
- `samples/`
  - Sample code (markers, kprobes, kobjects…)
- `scripts/`
  - Scripts for internal or external use
- `security/`
  - Security model implementations (SELinux…)
- `sound/`
  - Sound support code and drivers
- `tools/`
  - Code for various user space tools (mostly C)

Linux sources structure 5/5

- `usr/`
  - Code to generate an initramfs cpio archive
- `virt/`
  - Virtualization support (KVM)

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-3.8.13-bone64
- `beagle$` **`ls -F /boot`**

```
config-3.14.17-bone8      initrd.img-3.8.13-bone64  uboot/
config-3.8.13-bone64      SOC.sh                    uEnv.txt
dtbs/                     System.map-3.14.17-bone8  vmlinuz-3.14.17-bone8*
initrd.img-3.14.17-bone8  System.map-3.8.13-bone64  vmlinuz-3.8.13-bone64*
```

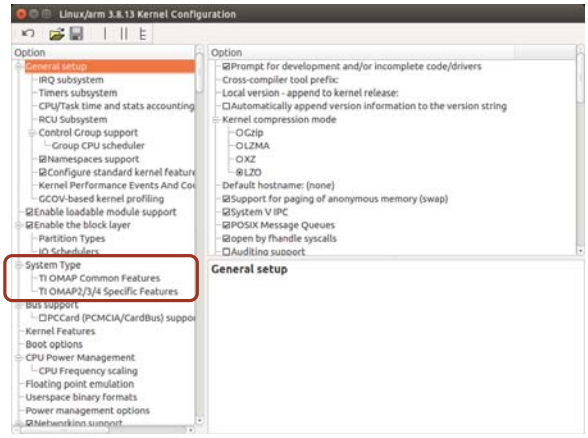
make xconfig

`make xconfig`

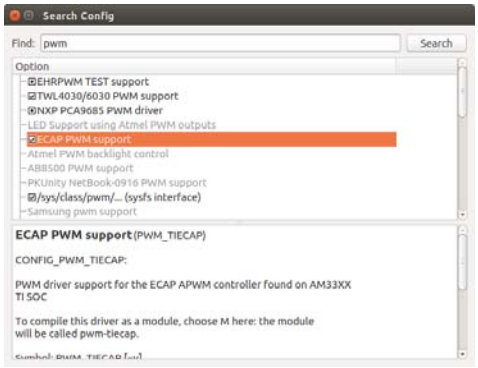
- The most common graphical interface to configure the kernel
- Make sure you read
  help -> introduction: useful options!
- File browser: easier to load configuration files
- New search interface to look for parameters
- Required Debian / Ubuntu packages:

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

make xconfig screenshot



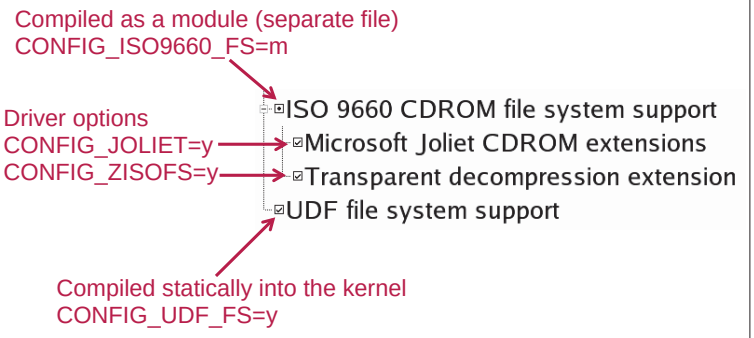
make xconfig search interface



Looks for a keyword in the description string

Allows to select or unselect found parameters.

Kernel configuration options



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_

make gconfig

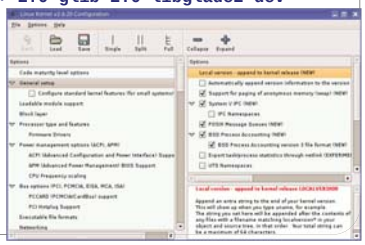
make gconfig

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
```



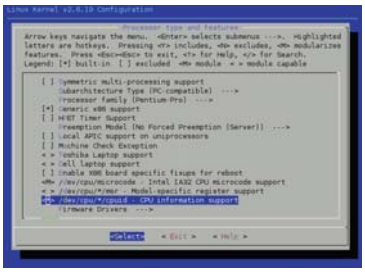
make menuconfig

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
```

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

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.

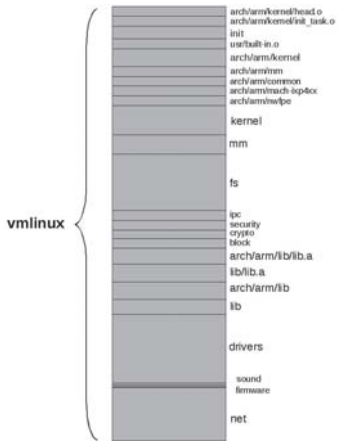
--Chris Hallinan

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               \
```

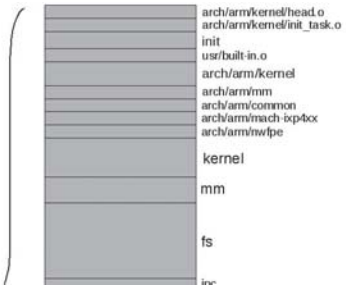
Look in **~/BeagleBoard/bb-kernel/dl/gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/bin/**

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-