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).

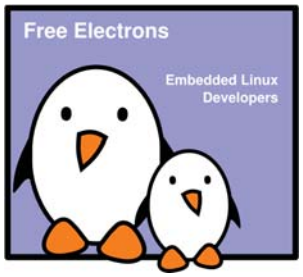
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

Free Electrons

Linux kernel introduction

Michael Opdenacker
Thomas Petazzoni
Free Electrons



Latest update: 10/2/2014,
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

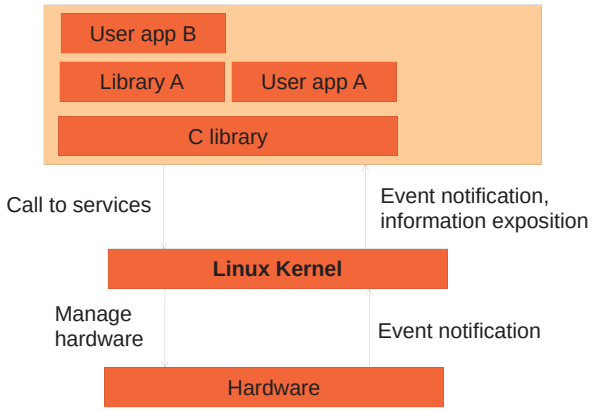
History

- 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 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.

Linux kernel in the system



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.

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

How did I find it?

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

Supported hardware architectures

2.6.31 status

What's the current version?

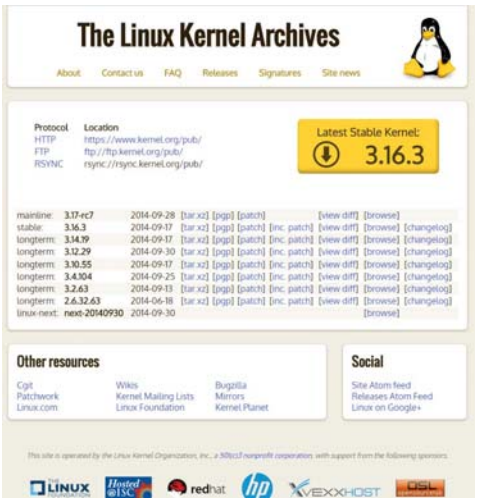
3.16.3

- 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

- The main interface between the kernel and userspace is the set of system calls
- About ~300 system calls that provides the main kernel services
- What are examples?
  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.

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

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:

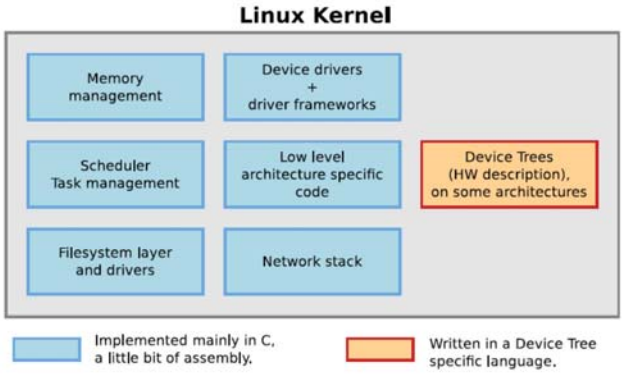
- /proc/cpuinfo: processor information
- /proc/meminfo: memory status
- /proc/version: kernel version and build information
- /proc/cmdline: kernel command line
- /proc/<pid>/environ: calling environment
- /proc/<pid>/cmdline: process command line

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

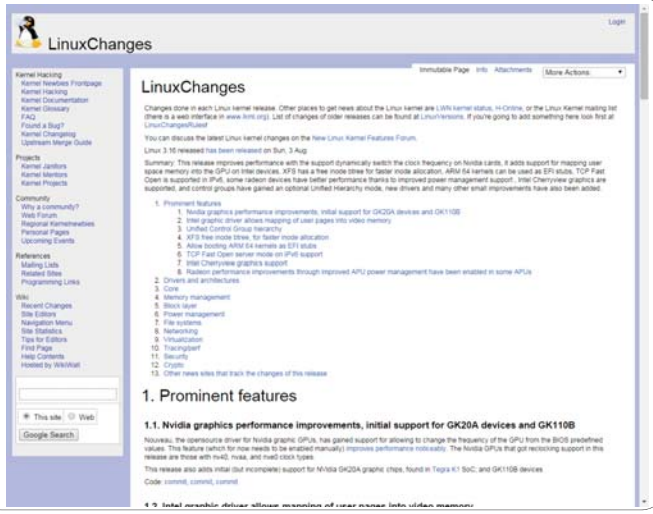
Embedded Linux Kernel Usage

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$` **`git checkout origin/am33x-v3.8 -b tmp`**
- `host$` **`./build_kernel.sh`**

Linux kernel size (1)

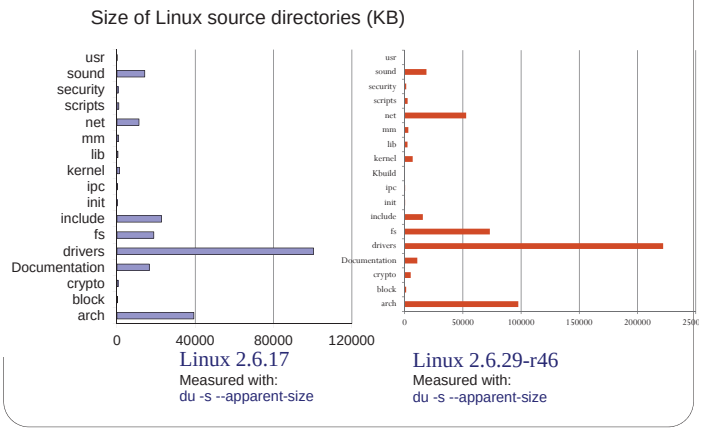
- Linux 3.10 sources:
  Raw size: 573 MB (43,000 files, ~15,800,000 lines)
  `gzip` compressed tar archive: 105 MB
  `bzip2` compressed tar archive: 83 MB (better)
  `xz` compressed tar archive: 69 MB (best)
- Minimum Linux 2.6.29 compiled kernel size with `CONFIG_EMBEDDED`, for a kernel that boots a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



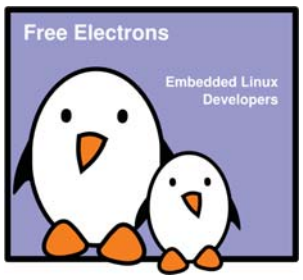
Linux 2.6.17
Measured with:
du -s --apparent-size

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 les, 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: rmware 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 le for conguration 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)