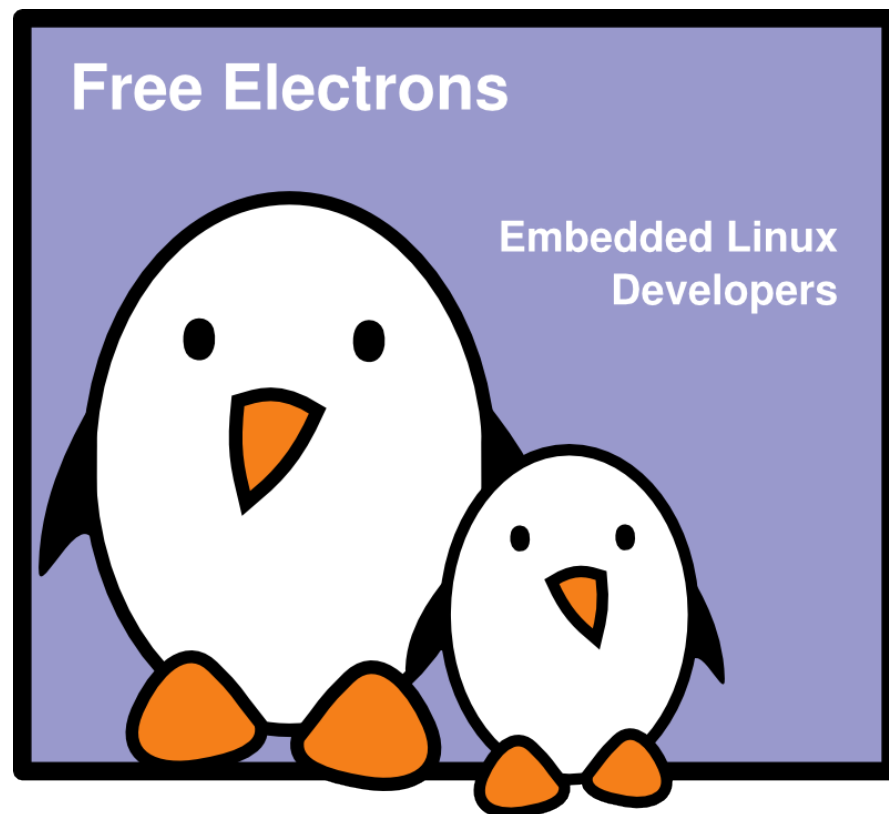




Embedded Linux system development training

Thomas Petazzoni
Florent Peyraud
Michael Opdenacker



Linux kernel

- Linux device drivers
- Board support code
- Mainstreaming kernel code
- Kernel debugging

Embedded Linux Training

All materials released with a free license!

- Unix and GNU/Linux basics
- Linux kernel and drivers development
- Real-time Linux, uClinux
- Development and profiling tools
- Lightweight tools for embedded systems
- Root filesystem creation
- Audio and multimedia
- System optimization

Free Electrons

Our services

Custom Development

- System integration
- Embedded Linux demos and prototypes
- System optimization
- Application and interface development

Consulting and technical support

- Help in decision making
- System architecture
- System design and performance review
- Development tool and application support
- Investigating issues and fixing tool bugs





Rights to copy

© Copyright 2004-2009, Free Electrons
feedback@free-electrons.com

Document updates available on
<http://free-electrons.com/doc/training/embedded-linux/>

Corrections, suggestions,
contributions and translations are welcome!

Latest update: May 21, 2009



Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions



Attribution. You must give the original author credit.



Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>



Electronic copies of this document

- ▶ All these documents are available under a free documentation license:
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>
- ▶ Electronic versions of this training book are available here:
<http://free-electrons.com/doc/training/embedded-linux/>
- ▶ You can use them to display the presentations on your own screen.
- ▶ These documents will only remain on our website for a few months. You should get updates from
<http://free-electrons.com/doc/training/embedded-linux/>



Hyperlinks in this document

- ▶ Links to external sites
Example: <http://kernel.org/>

Usable in the PDF and ODP formats
Try them on this page!

- ▶ Kernel source files
Our links let you view them in your browser.
Example: [kernel/sched.c](#)

- ▶ Kernel source code:
Identifiers: functions, macros, type definitions...
You get access to their definition, implementation and where they are used. This invites you to explore the source by yourself!

click → [wait_queue_head_t queue;](#)
→ [init_waitqueue_head\(&queue\);](#)

- ▶ Table of contents
Directly jump to the corresponding sections.
Example: [Kernel configuration](#)



Cooperate!

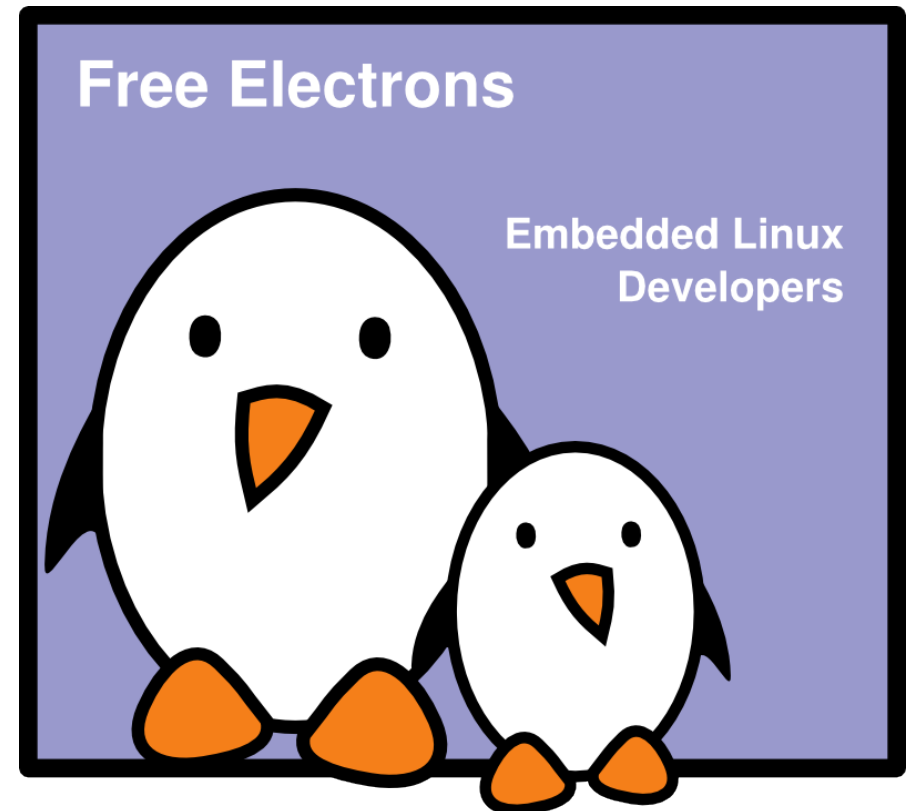
As in the Free Software and Open Source community, cooperation during practical labs is valuable in this training session:

- ▶ Don't hesitate to share your questions with other participants. They can make some points easier to understand, by using their own words.
- ▶ Explaining what you understood to other participants also helps to consolidate your knowledge.
- ▶ If you complete your labs before other people, don't hesitate to help other people and investigate the issues they face. The faster we progress as a group, the more time we have to explore extra topics.
- ▶ Don't hesitate to report potential bugs to your instructor.
- ▶ Don't hesitate to look for solutions on the Internet as well.



Introduction to embedded Linux

Michael Opdenacker
Thomas Petazzoni
Free Electrons





Embedded system?

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems control many of the common devices in use today.

Wikipedia, http://en.wikipedia.org/wiki/Embedded_system



Many different systems

- ▶ A very generic definition
 - ▶ Covers very different types of systems
 - ▶ Fuzzy border with “standard” systems.
- ▶ Consumer electronics (CE) products
 - ▶ Home routers, DVD players, TV sets, digital cameras, GPS, camcorders, mobile phones, microwave ovens...
- ▶ Industrial products
 - ▶ Machine control, alarms, surveillance systems, automotive, rail, aircraft, satellite...





Embedded Linux

- ▶ The Free Software and Open Source world offers a broad range of tools to develop embedded systems.
- ▶ Advantages
 - ▶ Reuse of existing components for the base system. Allows to focus on added its value.
 - ▶ High quality, proven components (Linux kernel, C libraries...)
 - ▶ Complete control on the choice of components. Modifications possible without external constraints.
 - ▶ Community support: tutorials, mailing lists...
 - ▶ Low cost, in particular no per-unit royalties.
 - ▶ Potentially less legal issues.
 - ▶ Easier access to software and tools.



Market share survey

- ▶ Operating system used in the previous project
 - ▶ Proprietary OS: 39%
 - ▶ Free of cost embedded Linux: 29%
 - ▶ Embedded Linux with commercial support: 11%
 - ▶ Home grown OS: 7%
 - ▶ No OS: 11%
- ▶ Operating system planned for the next project
 - ▶ Free of cost embedded Linux: 71%
 - ▶ Embedded Linux with commercial support: 16%
 - ▶ Proprietary OS: 12%
 - ▶ Home grown OS: 1%
- ▶ Source: Venture Development Corp, October 2007

http://www.vdc-corp.com/_documents/pressrelease/press-attachment-1394.pdf



Device examples

- ▶ GPS: TomTom and Garmin
- ▶ Home network routers: Linksys, Netgear
- ▶ PDA: Zaurus, Nokia N8x0
- ▶ TVs, camcorders, DVD players: Sony, Philips
- ▶ Mobile phones: Motorola, Android, OpenMoko
- ▶ Industrial machinery
- ▶ And many other products you don't even imagine...



Quiz



It works with Linux, but what is it for?



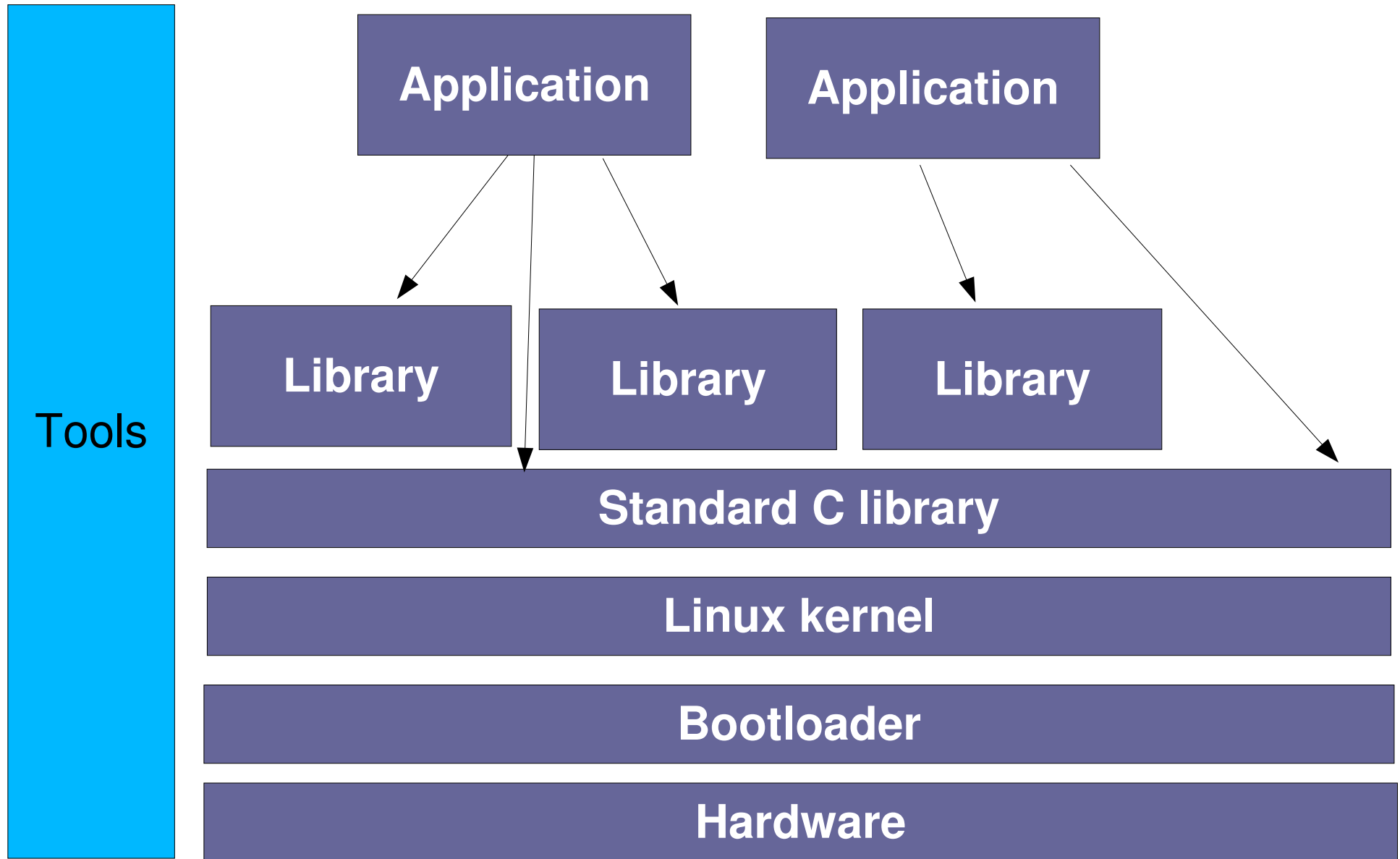
Answer



To milk cows!



Basic architecture





Embedded hardware

- ▶ Hardware for embedded systems is often different from hardware for classical systems.
 - ▶ Often a different CPU architecture: often ARM, MIPS or PowerPC. x86 is also used.
 - ▶ Storage on flash storage, NOR or NAND type, often with limited capacity (from a few MB to hundreds of MB)
 - ▶ Limited RAM capacity (from a few MB to several tens of MB)
 - ▶ Many interconnect bus not often found on the desktop: I2C, SPI, SSP, CAN, etc.
- ▶ Development boards starting from a few hundreds of EUR / USD
 - ▶ Often used as a basis for the final board design.



Examples

▶ Picotux 100

- ▶ ARM7 55 MHz, Netsilicon NS7520
- ▶ 2 MB of flash
- ▶ 8 MB of RAM
- ▶ Ethernet
- ▶ 5 GPIOs
- ▶ Serial



▶ OpenMoko

- ▶ ARM 920T 400 MHz, Samsung 2442B
- ▶ 2 MB of NOR flash
- ▶ 128 MB of RAM
- ▶ 256 MB of NAND flash
- ▶ 640x480 touchscreen , Bluetooth, GSM, serial, GPS, sound, 2 buttons, Wifi, USB, etc.





Emulation

- ▶ QEMU allows to emulate many CPU architectures
 - ▶ X86, of course, but also PowerPC, ARM, MIPS, SPARC, etc.
 - ▶ Command: `qemu-system-ARCH`
- ▶ Full system emulation: CPU, RAM and devices
- ▶ For each architecture, several platforms are proposed:
 - ▶ For ARM: Integrator, Versatile, PDA Sharp, Nokia N8x0, Gumstix, etc.
 - ▶ `qemu-system-arm -M ?`
- ▶ Website: <http://bellard.org/qemu/>



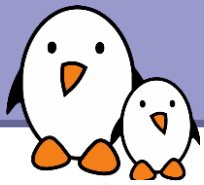
Minimum requirements

- ▶ A CPU supported by gcc and the Linux kernel
 - ▶ 32 bit CPU
 - ▶ MMU-less CPUs are also supported, through the uClinux project.
- ▶ A few MB of RAM, from 4 MB.
8 MB are needed to do really do something.
- ▶ A few MB of storage, from 2 MB.
4 MB to really do something.
- ▶ Linux isn't designed for small microcontrollers that just have a few tens or hundreds of flash and RAM.
 - ▶ Base metal, no OS
 - ▶ Reduced systems, such as FreeRTOS



Cross-compiling toolchains (1)

- ▶ Essential tool for embedded development on non-x86 architectures.
- ▶ Contains tools
 - ▶ Executing on the host machine (the developer's machine, usually x86)
 - ▶ Generating / handling code for the target machine (usually non x86)
- ▶ Binary tools
 - ▶ Binutils
 - ▶ Ld, as, nm, readelf, objdump, etc.
- ▶ Standard C library
 - ▶ glibc, uClibc or eglibc
- ▶ C/C++ compiler
 - ▶ gcc
- ▶ Math libraries
 - ▶ gmp, mpfr
- ▶ Debugger
 - ▶ gdb



Cross-compiling toolchains (2)

▶ Hand made

- ▶ Requires to configure and compile all the components in the right order.
- ▶ Gcc and binutils are not always bug free on non x86 architectures. Need to apply patches.

▶ Pre-compiled

- ▶ The simplest solution
- ▶ Code Sourcery is a renowned supplier, under contract with ARM, MIPS and WindRiver

▶ Generated by scripts

- ▶ Crosstool-ng,
<http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool>
- ▶ Buildroot, <http://buildroot.uclibc.org>



Bootloader

- ▶ On PCs: LILO or GRUB
 - ▶ The BIOS does a big part of the job, supplying the bootloader with routines to load data from the disk.
- ▶ On embedded architectures: no BIOS
 - ▶ The bootloader must do everything, including the initialization of the RAM controller.
 - ▶ At power-up, the CPU starts to execute at a fixed address.
 - ▶ Hardware is designed in a way that part of flash storage is mapped at this address.
 - ▶ The bootloader entry point is stored at that address in flash. It takes control on the hardware right from power-up.



Bootloader: U-Boot

- ▶ Many bootloaders exist for non-x86 architectures, some more or less architecture or platform specific.
- ▶ The most popular free software bootloader is **Das U-Boot**
- ▶ It supports a wide number of architectures, and is easy to configure and modify
- ▶ Basic features:
 - ▶ Kernel and filesystem download through the network (tftp protocol)
 - ▶ Flash protection, erasing and writing
 - ▶ Kernel execution
 - ▶ Diagnostic tools: reading / writing memory, device tests, etc.
- ▶ <http://www.denx.de/wiki/U-Boot>



Linux kernel (1)

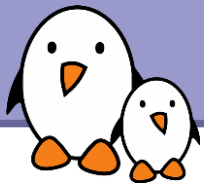
- ▶ Essential part of an embedded system.
- ▶ Basic system components: managing processes, memory, filesystems, protocols, networking, etc.
- ▶ Contains drivers for most devices:
- ▶ The kernel distinguishes 3 levels for embedded hardware support:
 - ▶ CPU architecture: ARM, MIPS, PowerPC
 - ▶ CPU model: Samsung SC2442 for example
 - ▶ Machine: OpenMoko Freerunner



Features brought by Linux

Changes when you come from a traditional embedded OS (RTOS)

- ▶ Virtual memory: each program runs with virtual addresses
No need for address relocation to avoid conflicts.
- ▶ Memory protection: processes can only access their own address space. Errors are immediately detected, killing the process.
Processes can neither corrupt other processes or the kernel.
Processes need to go through dedicated APIs to access hardware addresses.
- ▶ On-demand paging (thanks to the MMU): the kernel only loads the parts of executables and libraries which are actually accessed. This increases application startup time, performance and optimizes memory consumption.
- ▶ And many others!



Linux kernel (2)

- ▶ The kernel is often ported on a board by its manufacturer, otherwise you can have it done by an specialized company, or get your hands dirty.
- ▶ A configuration file is supplied with each machine. It can be customized.



Linux kernel (3)

- ▶ Get kernel sources
- ▶ Apply patches if needed
- ▶ Configure the architecture and cross-compiling toolchain in the Makefile.
- ▶ Use a ready-made configuration
 - ▶ `make mymachine_defconfig`
- ▶ Compile
 - ▶ `make`
- ▶ Result: 1 file
 - ▶ `arch/arm/boot/zImage` on ARM



Standard C library

- ▶ The base library between all other libraries on one side, and the kernel on the other side.
- ▶ It is part of the cross-compiling toolchain.
- ▶ Three solutions
 - ▶ GNU Libc, the standard release used in all desktop and server systems. Full features, but big.
 - ▶ uClibc, a complete rewrite of a simpler libc, optimized for size and with configurable features.
 - ▶ eglibc, a derivative of GNU Libc adding more configuration flexibility.
- ▶ With the C library in an embedded system, you already have a feature-rich API to program non-graphical applications.



BusyBox

- ▶ Need a basic set of utilities for the target system (in particular for shell scripts)
- ▶ cp, ls, mv, mkdir, rm, tar, mknod, wget, grep, sed...
- ▶ A solution: use standard GNU tools
 - ▶ fileutils, coreutils, tar, wget, etc.
 - ▶ Drawback: too many utilities, not designed for embedded.
- ▶ A better solution: BusyBox
 - ▶ All the utilities in a single binary program.
 - ▶ Utilities with reduced features... and size
 - ▶ Extremely configurable
 - ▶ Symbolic links to use them as usual
 - ▶ <http://www.busybox.net>
 - ▶ Used in many devices on the market.



Graphical libraries

- ▶ Low-level graphical solutions
 - ▶ The framebuffer managed by the Linux kernel.
 - ▶ DirectFB, offering a more convenient programming interface.
 - ▶ X.org Kdrive, a simplified X server
 - ▶ Nano-X
- ▶ Higher-level graphical solutions
 - ▶ Qt, which can directly work on top of the kernel framebuffer, or using an X server
 - ▶ GTK, which can work on top of DirectFB or using an X server.
 - ▶ WxEmbedded, on top of X, DirectFB or Nano-X



Libraries and tools

- ▶ In theory, all the free software tools and libraries can be cross-compiled and used on an embedded platform.
 - ▶ Once the system is in place, it's just Linux!
- ▶ In practice, cross-compiling is often difficult, because not anticipated by original developers.
 - ▶ Though they have many shortcomings, well used autotools are the best way to make software cross-compiling aware.
- ▶ Dedicated tools for platforms with limited resources
 - ▶ Dropbear replacing OpenSSH as ssh server and client.
 - ▶ Several reduced HTTP servers instead of Apache
 - ▶ Text editors



Building tools

- ▶ Several ways of building an embedded system
 - ▶ By hand
 - ▶ Painful, with little reproducibility, difficult to find the right options and apply the right patches, etc.
 - ▶ Using building tools
 - ▶ Buildroot
 - ▶ OpenEmbedded
 - ▶ PTXdist
 - ▶ Using distributions
 - ▶ Gentoo Embedded
 - ▶ Debian Embedded

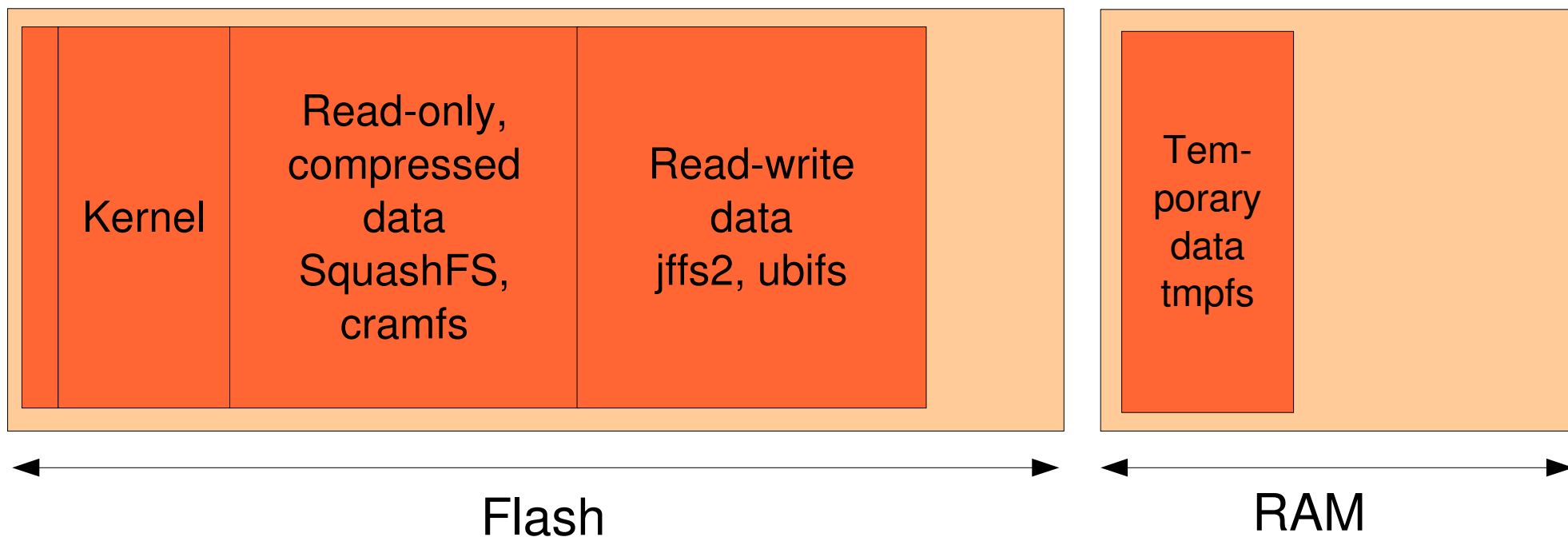


Storage (1)

- ▶ Flash storage in embedded system is usually raw flash.
- ▶ No hardware takes care of “wear leveling”: equally distributing writes across flash sectors.
- ▶ These flash devices are considered by the kernel as Memory Technology Devices (MTD)
- ▶ Specific filesystems must be used
- ▶ Today: JFFS2, UBIFS
- ▶ Tomorrow: LogFS, AXFS
- ▶ Other filesystems for embedded systems: SquashFS and cramfs, for read-only parts, tmpfs for temporary data.



Storage (2)





Real-time

- ▶ One constraint in some embedded systems is a predictable response time.
 - ▶ Collecting data or reacting to an event, which must happen within bounded time.
- ▶ Requires a “real-time” system.
- ▶ Real-time has **nothing to do with performance**
 - ▶ The time limits to guarantee can be wider, but must be guaranteed.
 - ▶ A real-time system is sometimes globally slower than a classical time-sharing system.



Linux and real-time

- ▶ Linux was not originally designed as a real-time system, but as a time-sharing system.
 - ▶ Goal: maximize the use of resources to maximize the global throughput of applications.
- ▶ Two approaches to add real-time support to the kernel:
 - ▶ An in-kernel approach. Goal: progressively reduce the sections during which the kernel doesn't react to an external event.
 - ▶ Linux-rt, patch maintained by Ingo Molnar, Thomas Gleixner...
 - ▶ A dual-kernel approach, where a real-time microkernel is in charge of processing important events, guaranteeing response time. The Linux kernel runs as a background task when there's no real-time task to run.
 - ▶ RTAI, Xenomai



Debugging

- ▶ For low level parts, bootloader and kernel, using JTAG
 - ▶ Bus allowing to directly control the CPU
 - ▶ Need for a probe connecting to the board
 - ▶ On the host machine, generally used through gdb (compiled to support the target CPU)
- ▶ For user applications
 - ▶ Using gdbserver on the target board
 - ▶ And gdb on the development host (compiled to support the target CPU), controlling the application execution remotely, by connecting to gdbserver.



Going further

- ▶ Training materials from Free-Electrons, available under a free documentation license
 - ▶ Integrating embedded Linux systems
 - ▶ Kernel code and device driver development
- ▶ Building Embedded Linux Systems, O'Reilly, 2008
- ▶ Embedded Linux Primer, Prentice Hall, 2006
- ▶ Linux Devices, <http://www.linuxdevices.com>
- ▶ ELC et ELCE conference videos, released by Free Electrons
<http://free-electrons.com/community/videos/conferences/>



The Unix and GNU / Linux command line

System administration basics



Network setup (1)

- ▶ `ifconfig -a`
Prints details about all the network interfaces available on your system.
- ▶ `ifconfig eth0`
Lists details about the `eth0` interface
- ▶ `ifconfig eth0 192.168.0.100`
Assigns the `192.168.0.100` IP address to `eth0` (1 IP address per interface).
- ▶ `ifconfig eth0 down`
Shuts down the `eth0` interface (frees its IP address).





Network setup (2)

▶ `route add default gw 192.168.0.1`

Sets the default route for packets outside the local network. The gateway (here `192.168.0.1`) is responsible for sending them to the next gateway, etc., until the final destination.

▶ `route -n`

Lists the existing routes

`-n` option: immediately displays ip addresses instead of trying to find their domain names

▶ `route del default`

or `route del <IP>`

Deletes the given route

Useful to redefine a new route.



Network testing

► `ping freshmeat.net`
`ping 192.168.1.1`

Tries to send packets to the given machine and get acknowledgment packets in return.

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.  
64 bytes from 192.168.1.1: icmp_seq=0 ttl=150 time=2.51 ms  
64 bytes from 192.168.1.1: icmp_seq=1 ttl=150 time=3.16 ms  
64 bytes from 192.168.1.1: icmp_seq=2 ttl=150 time=2.71 ms  
64 bytes from 192.168.1.1: icmp_seq=3 ttl=150 time=2.67 ms
```

- When you can ping your gateway, your network interface works fine.
- When you can ping an external IP address, your network settings are correct!



Network setup summary

Only for simple cases with 1 interface, no dhcp server...

- ▶ Connect to the network
(cable, wireless card or device...)
- ▶ Identify your network interface:
`ifconfig -a`
- ▶ Assign an IP address to your interface (assuming `eth0`)
`ifconfig eth0 192.168.0.100` (example)
- ▶ Add a route to your gateway (assuming `192.168.0.1`)
for packets outside the network:
`route add default gw 192.168.0.1`



Name resolution

- ▶ Your programs need to know what IP address corresponds to a given host name (such as `kernel.org`)
- ▶ Domain Name Servers (DNS) take care of this.
- ▶ You just have to specify the IP address of 1 or more DNS servers in your `/etc/resolv.conf` file:
`nameserver 217.19.192.132`
`nameserver 212.27.32.177`
- ▶ The changes take effect immediately!



Creating filesystems

Examples

▶ `mkfs.ext2 /dev/sda1`

Formats your USB key (`/dev/sda1`: 1st partition raw data) in `ext2` format.

▶ `mkfs.ext2 -F disk.img`

Formats a disk image file in `ext2` format

`-F`: force. Execute even if not a real device file.

▶ `mkfs.vfat -v -F 32 /dev/sda1 (-v: verbose)`

Formats your USB key back to `FAT32` format.

▶ `mkfs.vfat -v -F 32 disk.img`

Formats a disk image file in `FAT32` format.

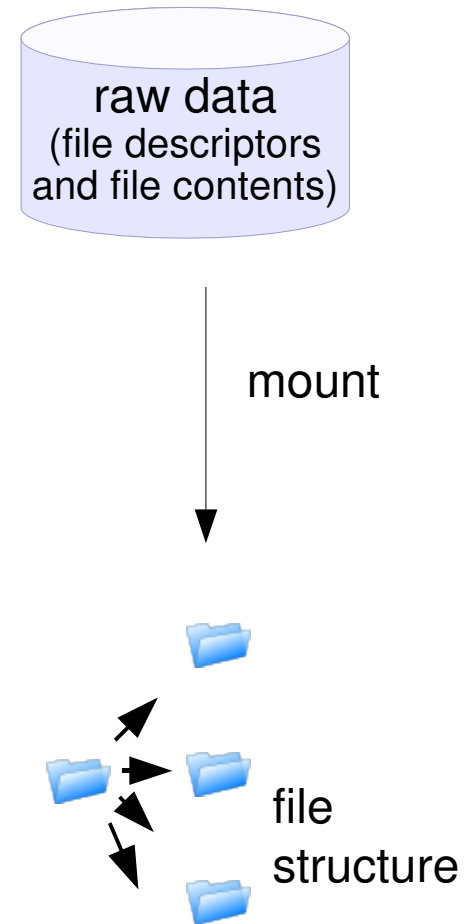
Blank disk images can be created as in the below example:

```
dd if=/dev/zero of=disk.img bs=1024 count=65536
```



Mounting devices (1)

- ▶ To make filesystems on any device (internal or external storage) visible on your system, you have to *mount* them.
- ▶ The first time, create a mount point in your system:
`mkdir /mnt/usbdisk` (example)
- ▶ Now, mount it:
`mount -t vfat /dev/sda1 /mnt/usbdisk`
/dev/sda1: physical device
-t: specifies the filesystem (format) type
(`ext2`, `ext3`, `vfat`, `reiserfs`, `iso9660`...)





Mounting devices (2)

- ▶ Lots of `mount` options are available, in particular to choose permissions or the file owner and group... See the `mount` manual page for details.
- ▶ Mount options for each device can be stored in the `/etc/fstab` file. Thanks to this file, you just need to state the mount point:

```
# /etc/fstab: static file system information.
# <file system> <mount point>    <type>    <options>                <dump> <pass>
proc          /proc          proc      defaults                  0       0
/dev/hda3     /              ext3      defaults,errors=remount-ro 0       1
/dev/hda4     /home          ext3      defaults                  0       2
/dev/hda2     /root2         ext3      defaults                  0       2
/dev/hda1     none           swap      sw                        0       0
/dev/hdc      /media/cdrom0  udf,iso9660 user,noauto              0       0
```

- ▶ `mount` examples with `/etc/fstab`:
`mount /proc`
`mount /media/cdrom0`



Mounting devices (3)

You can also mount a filesystem image stored in a regular file (*loop devices*)

- ▶ Useful to develop filesystems for another machine
- ▶ Useful to access the contents of an ISO cdrom image without having to burn it.
- ▶ Useful to have a Linux filesystem inside a file in a Windows partition.

```
cp /dev/sda1 usbkey.img  
mount -o loop -t vfat usbkey.img /mnt/usbdisk
```



Listing mounted filesystems

Just use the `mount` command with no argument:

```
/dev/hda6 on / type ext3 (rw,noatime)
none on /proc type proc (rw,noatime)
none on /sys type sysfs (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda4 on /data type ext3 (rw,noatime)
none on /dev/shm type tmpfs (rw)
/dev/hda1 on /win type vfat (rw,uid=501,gid=501)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
```



Unmounting devices

- ▶ `umount /mnt/usbdisk`

Commits all pending writes and unmounts the given device, which can then be removed in a safe way.

- ▶ To be able to unmount a device, you have to close all the open files in it:

- ▶ Close applications opening data in the mounted partition

- ▶ Make sure that none of your shells have a working directory in this mount point.

- ▶ You can run the `lsof <mount point>` command to view which processes still have open files in the mounted partition.



Shutting down

- ▶ `halt`

Immediately halts the system.

- ▶ `reboot`

Immediately reboots the system.

- ▶ `[Ctrl][Alt][Del]`

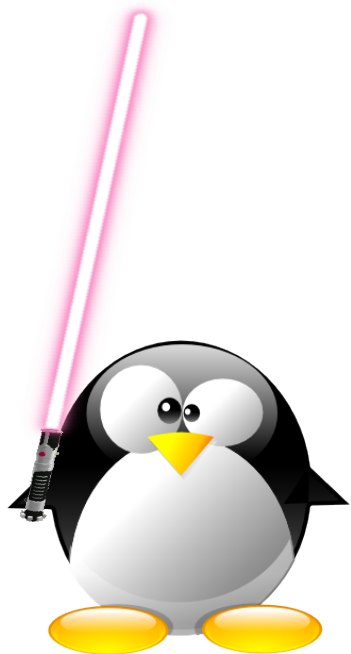
Also works on GNU/Linux to reboot.

Embedded systems: you must use an implementation of `init` and can specify any key combination in `/etc/inittab`.



Beware of the dark side of root

- ▶ `root` user privileges are only needed for very specific tasks with security risks: mounting, creating device files, loading drivers, starting networking, changing file ownership, package upgrades...
- ▶ Even if you have the `root` password, your regular account should be sufficient for 99.9 % of your tasks (unless you are a system administrator).
- ▶ In a training session, it is acceptable to use `root`. In real life, you may not even have access to this account, or put your systems and data at risk if you do.





Using the root account

In case you really want to use `root`...

- ▶ If you have the `root` password:

`su` – (**s**witch **u**ser)

- ▶ In modern distributions, the `sudo` command gives you access to some `root` privileges with your own user password.

Example: `sudo mount /dev/hda4 /home`



Managing software packages (1)

Instructions for Debian based GNU/Linux systems
(Debian, Ubuntu...)

- ▶ Package repositories are specified in
`/etc/apt/sources.list`
- ▶ To update package repository lists:
`sudo apt-get update`
- ▶ To find the name of a package to install, the best is to use the
search engine on <http://packages.debian.org> or on
<http://packages.ubuntu.com>. You may also use:
`apt-cache search <keyword>`



Managing software packages (2)

- ▶ To install a given package:
`sudo apt-get install <package>`
- ▶ To remove a given package:
`sudo apt-get remove <package>`
- ▶ To install all available package updates:
`sudo apt-get dist-upgrade`

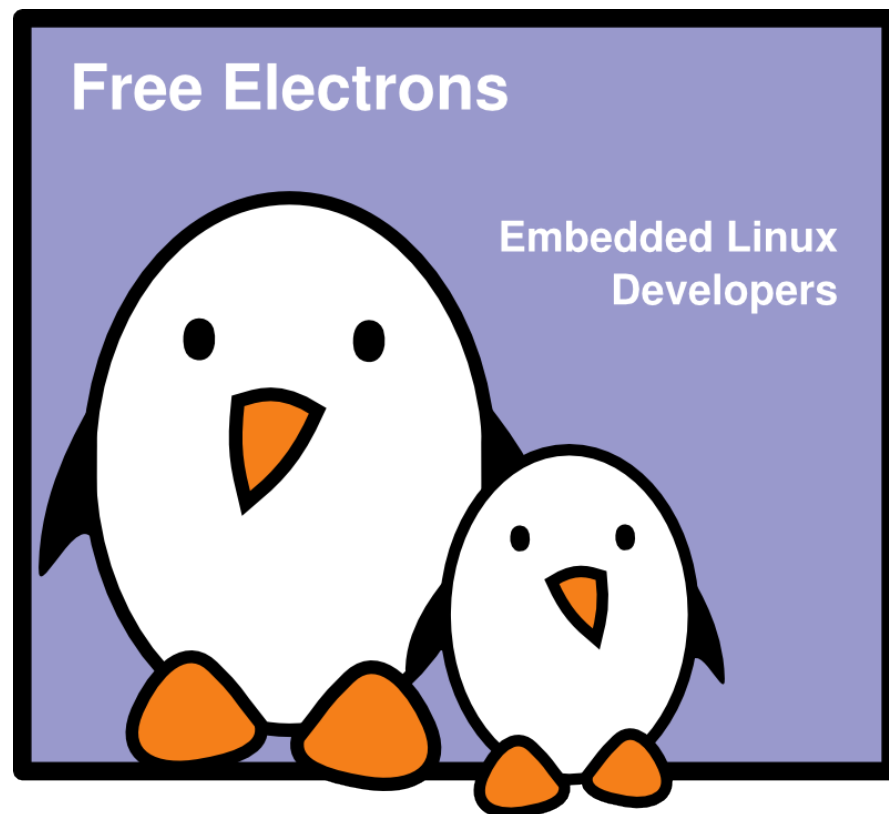
Further details on package management:

<http://www.debian.org/doc/manuals/apt-howto/>



Cross-compiling toolchains

Thomas Petazzoni
Michael Opdenacker
Free Electrons



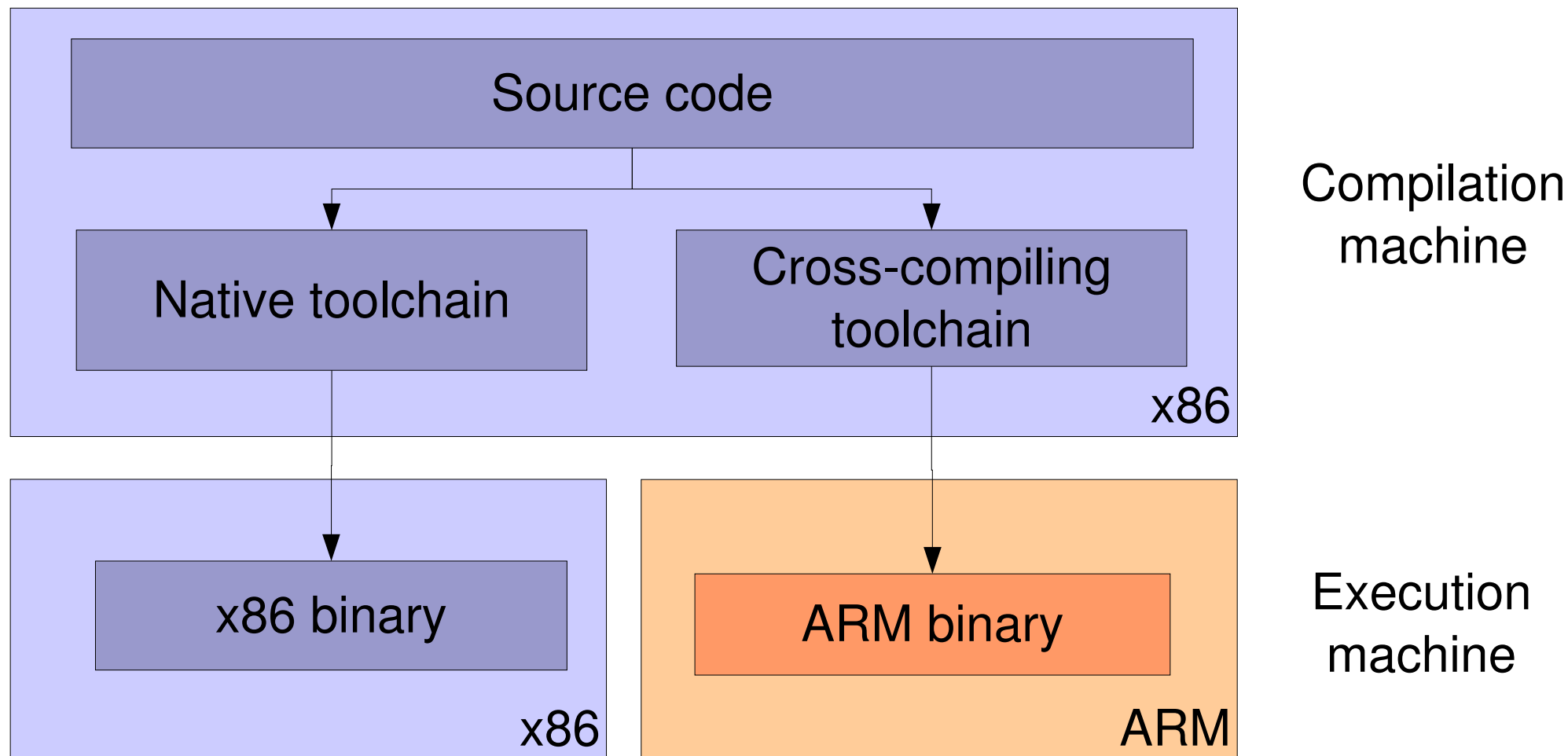


Definition (1)

- ▶ The usual development tools available on a GNU/Linux workstation is a **native toolchain**
- ▶ This toolchain runs on your workstation and generates code for your workstation, usually x86
- ▶ For embedded system development, it is usually impossible or not interesting to use a native toolchain
 - ▶ The target is too restricted in terms of storage and/or memory
 - ▶ The target is very slow compared to your workstation
 - ▶ You may not want to install all development tools on your target.
- ▶ Therefore, **cross-compiling toolchains** are generally used. They run on your workstation but generate code for your target.

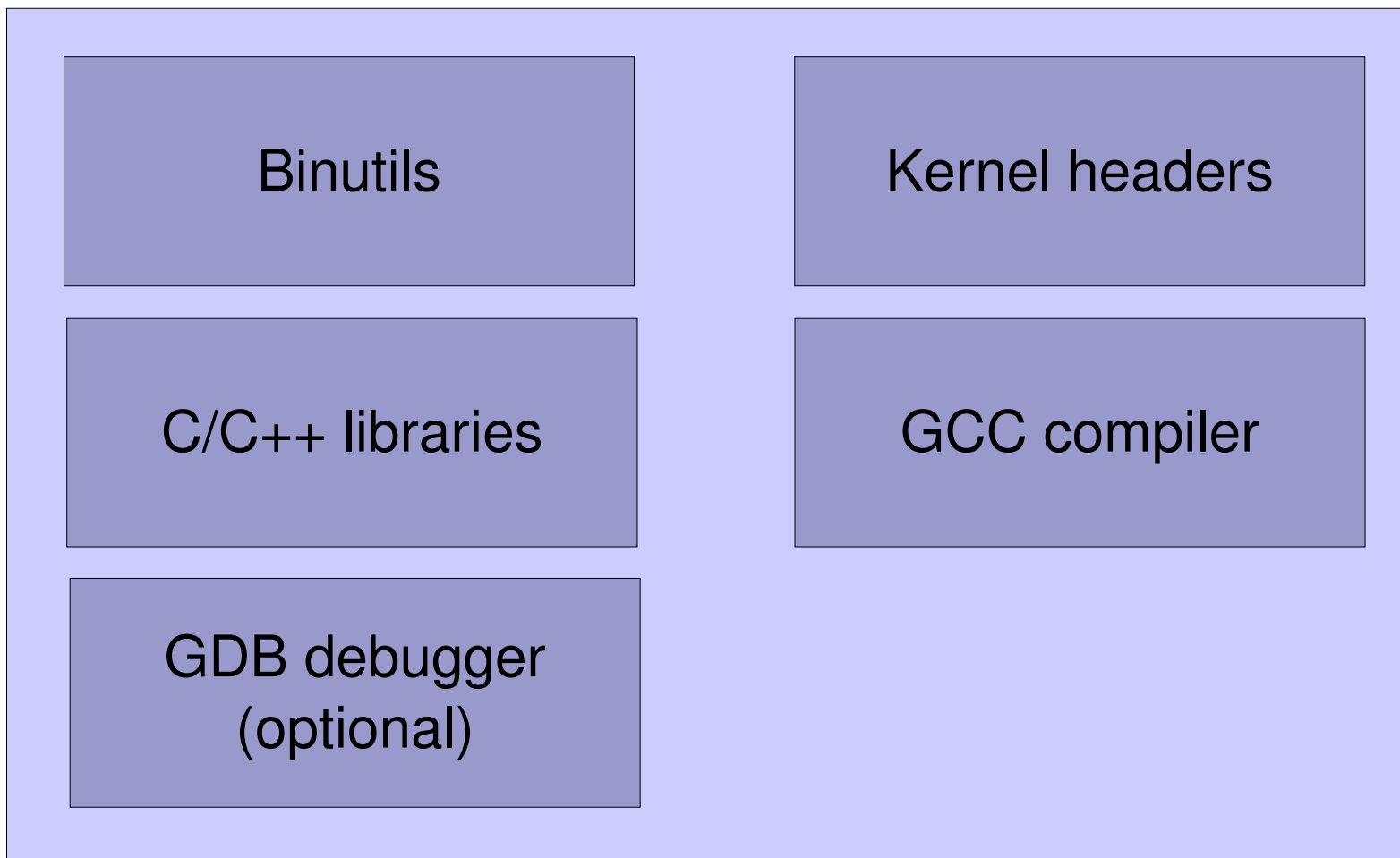


Definition (2)





Components





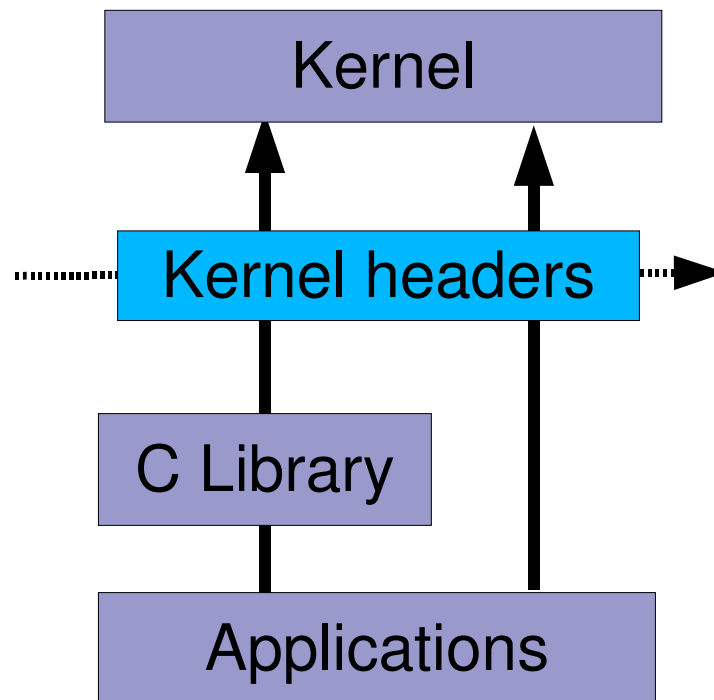
Binutils

- ▶ **Binutils** is a set of tools to generate and manipulate binaries for a given CPU architecture
 - ▶ `as`, the assembler, that generates binary code from assembler source code
 - ▶ `ld`, the linker
 - ▶ `ar`, `ranlib`, to generate `.a` archives, used for libraries
 - ▶ `objdump`, `readelf`, `size`, `nm`, `strings`, to inspect binaries. Very useful analysis tools !
 - ▶ `strip`, to strip useless parts of binaries in order to reduce their size
- ▶ <http://www.gnu.org/software/binutils/>
- ▶ GPL license



Kernel headers (1)

- ▶ The C library and compiled programs needs to interact with the kernel
 - ▶ Available system calls and their numbers
 - ▶ Constant definitions
 - ▶ Data structures, etc.
- ▶ Therefore, compiling the C library requires kernel headers, and many applications also require them.
- ▶ Available in `<linux/...>` and `<asm/...>` and a few other directories corresponding to the ones visible in `include/` in the kernel sources





Kernel headers (2)

- ▶ System call numbers, in `<asm/unistd.h>`

```
#define __NR_exit      1
#define __NR_fork      2
#define __NR_read      3
```

- ▶ Constant definitions, here in `<asm-generic/fcntl.h>`, included from `<asm/fcntl.h>`, included from `<linux/fcntl.h>`

```
#define O_RDWR        00000002
```

- ▶ Data structures, here in `<asm/stat.h>`

```
struct stat {
    unsigned long    st_dev;
    unsigned long    st_ino;
    [...]
};
```



Kernel headers (3)

- ▶ The kernel-to-userspace ABI is backward compatible
 - ▶ Binaries generated with a toolchain using kernel headers older than the running kernel will work without problem, but won't be able to use the new system calls, data structures, etc.
 - ▶ Binaries generated with a toolchain using kernel headers newer than the running kernel might work on if they don't use the recent features, otherwise they will break
 - ▶ Using the latest kernel headers is not necessary, unless access to the new kernel features is needed
- ▶ The kernel headers are extracted from the kernel sources using the `headers_install` kernel `Makefile` target.



GCC compiler

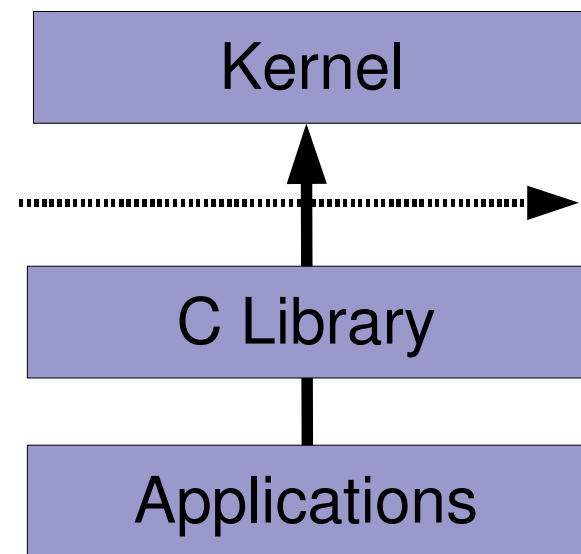
- ▶ GNU C Compiler, the famous free software compiler
- ▶ Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa, etc.
- ▶ <http://gcc.gnu.org/>
- ▶ Available under the GPL license, libraries under the LGPL.





C library

- ▶ The C library is an essential component of a Linux system
 - ▶ Interface between the applications and the kernel
 - ▶ Provides the well-known standard C API to ease application development
- ▶ Several C libraries are available: `glibc`, `uClibc`, `eglibc`, `dietlibc`, `newlib`, etc.
- ▶ The choice of the C library must be made at the time of the cross-compiling toolchain generation, as the GCC compiler is compiled against a specific C library.





glibc



<http://www.gnu.org/software/libc/>

- ▶ License: LGPL
- ▶ C library from the **GNU** project
- ▶ Designed for performance, standards compliance and portability
- ▶ Found on all **GNU / Linux** host systems
- ▶ Of course, actively maintained
- ▶ Quite big for small embedded systems: about 1.7 MB on **arm** (**Familiar Linux**: **libc**: 1.2 MB, **libm**: 500 KB)



<http://www.uclibc.org/> from CodePoet Consulting

- ▶ License: LGPL
- ▶ Lightweight C library for small embedded systems
 - ▶ High configurability: many features can be enabled or disabled through a `menuconfig` interface
 - ▶ Works only with Linux/uClinux, works on most embedded architectures
 - ▶ No stable ABI, different ABI depending on the library configuration
 - ▶ Focus on size rather than performance
 - ▶ Small compile time



uClibc (2)

- ▶ Most of the applications compile with uClibc
 - ▶ This applies to all applications used in embedded applications
 - ▶ The whole **Debian Woody** was ported to it...
You can assume it satisfied most needs!
- ▶ Size (**arm**): 4 times smaller than **glibc**!
uClibc: approx. 400 KB (**libuClibc**: 300 KB, **libm**: 55KB)
glibc: approx 1700 KB (**libc**: 1.2 MB, **libm**: 500 KB)
- ▶ Used on a large number of production embedded products, including consumer electronic devices
- ▶ Actively maintained, large developer and user base
- ▶ Now supported by **MontaVista**, **TimeSys** and **Wind River**.



uClibc (3)

After compilation and installation, the following components are available

- ▶ Standard headers, `stdio.h`, `stdlib.h`, `unistd.h` and others, and Linux kernel headers, integrated with the C library headers.
- ▶ The libraries themselves, with mainly
 - ▶ `libuClibc`, the C library itself
 - ▶ `ld-uClibc`, the dynamic loader, responsible for loading the shared libraries at the beginning of a program's execution
 - ▶ `librt`, the library implementing the real-time related functions
 - ▶ `libstdc++`, the C++ standard library
 - ▶ `libpthread`, the threads library
 - ▶ `libm`, the mathematic library



Honey, I shrunk the programs!

<i>C program</i>	<i>Compiled with shared libraries</i>		<i>Compiled statically</i>	
	<i>glibc</i>	<i>uClibc</i>	<i>glibc</i>	<i>uClibc</i>
Plain “hello world”	4.6 K	4.4 K	475 K	25 K
Busybox	245 K	231 K	843 K	311 K



eglibc

« Embedded glibc », under the LGPL

- ▶ Variant of the GNU C Library (GLIBC) designed to work well on embedded systems
- ▶ Strives to be source and binary compatible with GLIBC
- ▶ eglibc's goals include reduced footprint, configurable components, better support for cross-compilation and cross-testing.
- ▶ Can be built without support for NIS, locales, IPv6, and many other features.
- ▶ Supported by a consortium, with Freescale, MIPS, MontaVista and Wind River as members.
- ▶ <http://www.eglibc.org>





Other smaller C libraries

- ▶ Several other smaller C libraries have been developed, but none of them have the goal of allowing the compilation of large existing applications
- ▶ They need specially written programs and applications
- ▶ Choices :
 - ▶ Dietlibc, <http://www.fefe.de/dietlibc/>. Approximately 70 KB.
 - ▶ Newlib, <http://sources.redhat.com/newlib/>
 - ▶ Klibc, <http://www.kernel.org/pub/linux/libs/klibc/>, designed for use in an initramfs or initrd at boot time.

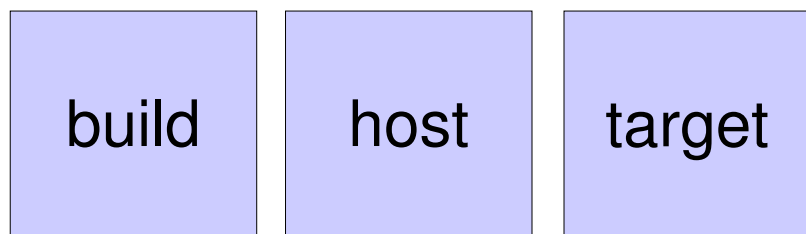


Building a toolchain

- ▶ Three machines must be distinguished when discussing toolchain creation
 - ▶ The build machine, where the toolchain is built.
 - ▶ The host machine, where the toolchain will be executed.
 - ▶ The target machine, where the binaries created by the toolchain will be executed.
- ▶ Four build types are possible

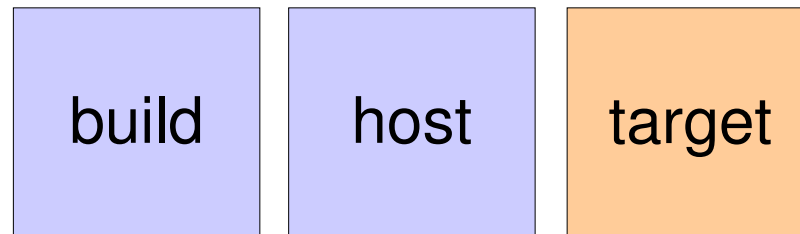


Building a toolchain (2)



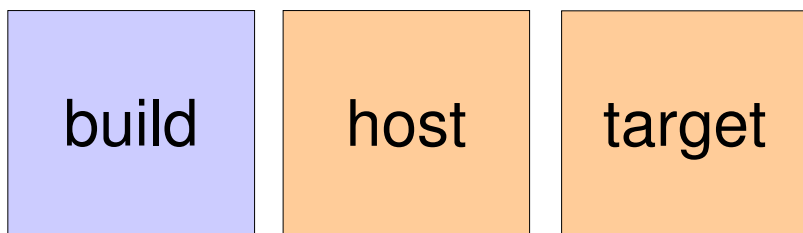
Native build

used to build the normal gcc of a workstation



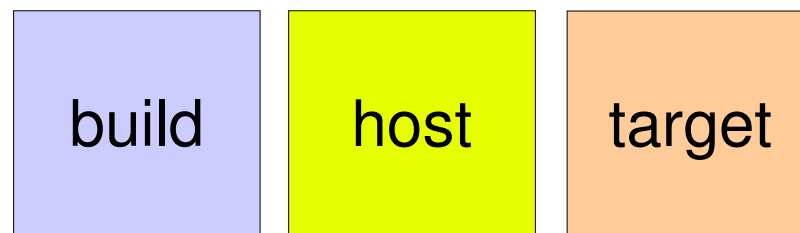
Cross build

used to build a toolchain that runs on your workstation but generates binaries for the target



Cross-native build

used to build a toolchain that runs on your target and generates binaries for the target



Canadian build

used to build on architecture A a toolchain that runs on architecture B and generates binaries for architecture C



Building a toolchain (3)

- ▶ Many decisions must be made when building a toolchain
- ▶ Choosing the C library
- ▶ Choosing the version of the different components
- ▶ Choosing the configuration of the toolchain
 - ▶ Which ABI should be used ? Toolchains for the ARM architecture for example, can generate binaries using the OABI (Old ABI) or the EABI (Embedded ABI), that are incompatible
 - ▶ Should the toolchain support software floating point, or does the hardware support floating point operations ?
 - ▶ Should the toolchain support locales, IPv6, or other specific features ?



Building a toolchain (4)

Crosstool build reports: <http://kegel.com/crosstool/crosstool-0.43/buildlogs>

gcc-4.1-20050709	gcc-4.1-20050709	gcc-4.0.1	gcc-4.0.1	gcc-4.0.1	gcc-4.0.1	gcc-4.0.1	gcc-4.0.1	gcc-4.0.1	gcc-4.1-20050716	gcc-4.1-20050716	gcc-4.1-20050716	gcc-4.1-20050716	gcc-4.1-20050716	gcc-4.1-20050716
cgcc-3.3.6	cgcc-3.3.6	cgcc-2.95.3	cgcc-2.95.3	cgcc-2.95.3	cgcc-3.3.6	cgcc-3.3.6	cgcc-3.3.6	cgcc-3.3.6	cgcc-2.95.3	cgcc-2.95.3	cgcc-2.95.3	cgcc-3.3.6	cgcc-3.3.6	cgcc-3.3.6
glibc-2.3.2	glibc-2.3.2	glibc-2.2.2	glibc-2.2.2	glibc-2.2.2	glibc-2.3.2	glibc-2.3.2	glibc-2.3.2	glibc-2.3.2	glibc-2.2.2	glibc-2.2.2	glibc-2.2.2	glibc-2.3.2	glibc-2.3.2	glibc-2.3.2
binutils-2.15	binutils-2.15	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1	binutils-2.16.1
linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3	linux-2.6.11.3
hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2	hdrs-2.6.11.2
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok
ICE gdb FAIL	ICE gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok

alpha

arm

arm9tdmi

arm-iwmmxt

arm-softfloat

arm-xscale

armeb

armv5b-softfloat

i686

ia64

m68k

mips

mipsel

powerpc-405

powerpc-750

powerpc-860

powerpc-970

s390



Basic steps

- ▶ Extract and install kernel headers
- ▶ Extract, configure, compile and install binutils
- ▶ Extract, configure and compile a first version gcc that generates binaries for the target. It will be used to cross-compile the C library.
- ▶ Extract, configure and compile the C library using the previously generated compiler.
- ▶ Re-configure and compile the final gcc cross-compiler.



Homebuilt toolchains

Building a cross-compiling toolchain by yourself is a difficult and painful task! Can take days or weeks!

- ▶ Lots of details to learn. Several components to build.
- ▶ Lots of decisions to make
(such as C library version and configuration for your platform)
- ▶ Need kernel headers and C library sources
- ▶ Need to be familiar with current `gcc` issues and patches on your platform
- ▶ Useful to be familiar with building and configuring tools
- ▶ <http://www.aleph1.co.uk/armlinux/docs/toolchain/toolchHOWTO.pdf>
can show you how fun it can be!



Get a precompiled toolchain

- ▶ Solution that most people choose, because it is the simplest and most convenient solution
- ▶ First, determine what toolchain you need: CPU, endianism, C library, component versions, ABI, soft float or hard float, etc.
- ▶ Many toolchains are freely available pre-compiled on the Web
 - ▶ CodeSourcery, <http://www.codesourcery.com>, is a reference in that area, but they only provide glibc toolchains.
 - ▶ Free-Electrons provides uClibc toolchains, <http://free-electrons.com/community/tools/uclibc>
 - ▶ See also <http://elinux.org/Toolchains>



Installing and using a precompiled toolchain

- ▶ Follow the installation procedure proposed by the vendor
- ▶ Usually, it is simply a matter of extracting a tarball at the proper place
 - ▶ Toolchains used not to be relocatable!
You must install them in the location they were built for.
 - ▶ This is no longer true with gcc 4.x, thanks to sysroot support, but it is still more convenient to install them at the proper place.
- ▶ Then, add the path to toolchain binaries in your **PATH**:
`export PATH=/path/to/toolchain/bin/:$PATH`



Toolchain building utilities

Another solution is to use utilities that automate the process of building the toolchain

- ▶ Same advantage as the pre-compiled toolchains: you don't need to mess up with all the details of the build process
- ▶ But also offers more flexibility in terms of toolchain configuration, component version selection, etc.
- ▶ They also usually contain several patches that fix known issues with the different components on some architectures
- ▶ Identical principle: shell scripts or Makefile that automatically fetch, extract, configure, compile and install the different components



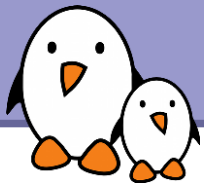
Toolchain building utilities (2)

▶ Crosstool

- ▶ The precursor, written by Dan Kegel
- ▶ Set of scripts and patches, glibc-only
- ▶ Not really maintained anymore
- ▶ <http://www.kegel.com/crosstool>

▶ Crosstool-ng

- ▶ Rewrite of [Crosstool](#), with a [menuconfig](#)-like configuration system
- ▶ Feature-full: supports [uClibc](#), [glibc](#), [eglibc](#), hard and soft float, many architectures
- ▶ Actively maintained
- ▶ <http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool>



Toolchain building utilities (3)

Many root filesystem building systems also allow the construction of cross-compiling toolchain

▶ Buildroot

- ▶ Makefile-based, [uClibc](#) only, maintained by the community

- ▶ <http://buildroot.uclibc.org>

▶ PTXdist

- ▶ Makefile-based, [uClibc](#) or [glibc](#), maintained mainly by Pengutronix

- ▶ http://www.pengutronix.de/software/ptxdist/index_en.html

▶ OpenEmbedded

- ▶ The feature-full, but complex building system

- ▶ <http://www.openembedded.org/>

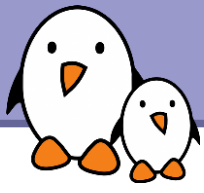


Practical lab – Using Crosstool-NG

Time to build your toolchain

- ▶ Configure **Crosstool-NG**
- ▶ Run it to build your own cross-compiling toolchain





Bootloaders



Bootloaders

- ▶ The bootloader is a piece of code responsible for
 - ▶ Basic hardware initialization
 - ▶ Loading of an application binary, usually an operating system kernel, from the Flash, from the network, or from another type of non-volatile storage.
 - ▶ Possibly uncompression of the application binary
 - ▶ Execution of the application
- ▶ Besides these basic functions, most bootloaders provide a shell with various commands implementing different operations.
 - ▶ Loading of data from storage or network, memory inspection, hardware diagnostics and testing, etc.



Bootloaders on x86 (1)

- ▶ The x86 processors are typically bundled on a board with a non-volatile memory containing a program, the BIOS.
- ▶ This program gets executed by the CPU after reset, and is responsible for basic hardware initialization and loading of a small piece of code from a non-volatile storage.
 - ▶ This piece of code is usually the first 512 bytes of an hard disk
- ▶ This piece of code is usually a 1st stage bootloader, which will load the full bootloader itself.
- ▶ The bootloader can then offer all its features. It typically understands filesystem formats so that the kernel file can be loaded directly from a normal filesystem.



Bootloaders on x86 (2)

- ▶ GRUB, Grand Unified Bootloader, the most powerful one.
<http://www.gnu.org/software/grub/>
 - ▶ Can read many filesystem formats to load the kernel image and the configuration, provides a powerful shell with various commands, can load kernel images over the network, etc.
- ▶ LILO, the original Linux Loader
<http://freshmeat.net/projects/lilo/>
- ▶ Syslinux, for network and removable media booting
<http://syslinux.zytor.com>



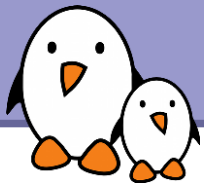
Bootloaders on embedded architectures (1)

- ▶ On embedded architectures, the low-level booting process is very CPU and board dependent
 - ▶ Some boards have a NOR flash from which the CPU starts executing instructions after reset. In that case, the bootloader must directly be flashed inside the NOR at the proper location
 - ▶ Some CPUs have an integrated bootcode in ROM that automatically loads a small portion of a DataFlash or NAND flash, usually to a static RAM. In that case, a minimal first stage bootloader is required, that will load the main bootloader (BootROM on AT91SAM CPUs, Steppingstone on S3C24xx CPUs, etc.).
- ▶ The bootloader on embedded architectures starts right after CPU reset, so it must initialize all the devices, including the memory controller in order to access the DRAM.
- ▶ As the boot process is very CPU and board dependent, refer to the vendor documentation.



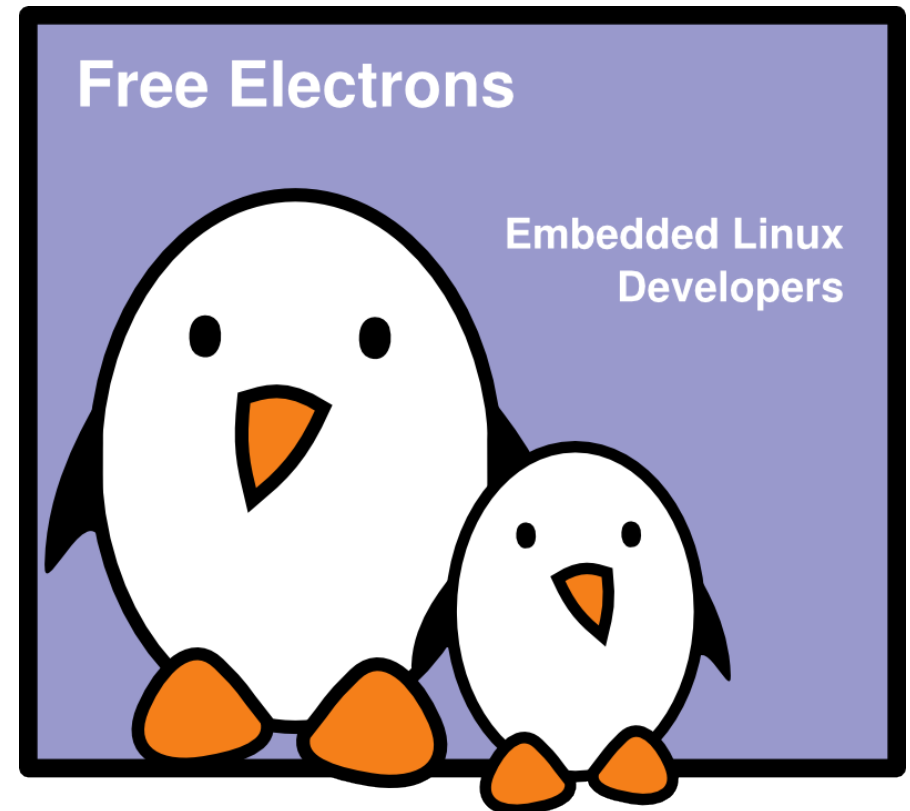
Bootloaders on embedded architectures (2)

- ▶ We will focus on the generic part, the main bootloader, offering the most important features.
- ▶ There are several open-source generic bootloaders
 - ▶ U-Boot, the universal bootloader by Denx
The most used on ARM, also used on PPC, MIPS, x86, m68k, NIOS, etc. Clearly the most widely used community solution.
<http://www.denx.de/wiki/U-Boot>
 - ▶ RedBoot, based on RedHat eCos
<http://sources.redhat.com/redboot/>
 - ▶ uMon: MicroMonitor general purpose, multi-OS bootloader
<http://microcross.com/html/micromonitor.html>
- ▶ There are also a lot of other open-source or proprietary bootloaders, often architecture-specific



The U-boot bootloader

Michael Opdenacker
Thomas Petazzoni
Free Electrons





U-Boot

U-Boot is a typical free software project

- ▶ Freely available at <http://www.denx.de/wiki/U-Boot>
- ▶ Documentation available at <http://www.denx.de/wiki/U-Boot/Documentation>
- ▶ The latest development source code is available in a Git repository:
<http://git.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary>
- ▶ Development and discussions happen around an open mailing-list <http://lists.denx.de/pipermail/u-boot/>
- ▶ Since the end of 2008, it follows a fixed-interval release schedule. Every two months, a new version is released. Versions are named YYYY.MM.



Compiling U-Boot (1)

- ▶ Get the source code from the website, and uncompress it
- ▶ The `include/configs/` directory contains one configuration file for each supported board
 - ▶ It defines the CPU type, the peripherals and their configuration, the memory mapping, the U-Boot features that should be compiled in, etc.
 - ▶ It is a simple `.h` file that sets pre-processor constants. See the [README](#) file for the documentation of these constants.
- ▶ Assuming that your board is already supported by U-Boot, there should be one file corresponding to your board, for example `include/configs/omap2420h4.h`.



Compiling U-Boot (2)

- ▶ U-Boot must be configured before being compiled
 - ▶ `make BOARDNAME_config`
 - ▶ Where `BOARDNAME` is the name of the configuration file in `include/configs/`, without the `.h`
- ▶ Make sure that the cross-compiler is available in `PATH`
`export PATH=/usr/local/uclibc-0.9.29-2/arm/bin/:$PATH`
- ▶ Compile U-Boot, by specifying the cross-compiler prefix
`make CROSS_COMPILE=arm-linux-`



Installing U-Boot

- ▶ U-Boot must usually be installed in a Flash memory to be executed by the hardware. Depending on the hardware, the installation of U-Boot is done in a different way
 - ▶ The board provides some kind of specific boot monitor, which allows to flash the second stage bootloader. In this case, refer to the board documentation and tools
 - ▶ U-Boot is already installed, and can be used to Flash a new version of U-Boot. However, be careful: if the new version of U-Boot doesn't work, the board is unusable
 - ▶ The board provides a JTAG interface, which allows to write to the Flash memory remotely, without any system running on the board. It also allows to rescue a board if the bootloader doesn't work.



U-boot prompt

- ▶ Connect the target to the host through a serial console
- ▶ Power-up the board. On the serial console, you will see something like:

```
U-Boot 1.1.2 (Aug  3 2004 - 17:31:20)
RAM Configuration:
Bank #0: 00000000  8 MB
Flash:  2 MB
In:     serial
Out:    serial
Err:    serial
u-boot #
```

- ▶ The U-Boot shell offers a set of commands. We will study the most important ones, see the documentation for a complete reference or the `help` command.



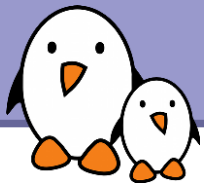
Information commands

Board
information

```
u-boot # bdfinfo
DRAM bank = 0x00000000
-> start = 0x00000000
-> size = 0x00800000
ethaddr = 00:40:95:36:35:33
ip_addr = 10.0.0.11
baudrate = 19200 bps
```

Flash
information

```
u-boot # flinfo
Bank # 1: AMD Am29LV160DB 16KB,2x8KB,32KB,31x64KB
Size: 2048 KB in 35 Sectors
Sector Start Addresses:
S00 @ 0x01000000 ! S01 @ 0x01004000 !
S02 @ 0x01006000 ! S03 @ 0x01008000 !
S04 @ 0x01010000 ! S05 @ 0x01020000 !
S06 @ 0x01030000 S07 @ 0x01040000
...
S32 @ 0x011D0000 S33 @ 0x011E0000
S34 @ 0x011F0000
```



Environment variables (1)

- ▶ U-Boot can be configured through environment variables, which affect the behavior of the different commands.
- ▶ See the documentation for the complete list of environment variables.
- ▶ The `printenv` command also to display all variables or one :

```
u-boot # printenv
```

```
baudrate=19200
```

```
ethaddr=00:40:95:36:35:33
```

```
netmask=255.255.255.0
```

```
ipaddr=10.0.0.11
```

```
serverip=10.0.0.1
```

```
stdin=serial
```

```
stdout=serial
```

```
stderr=serial
```



Network configuration

```
u-boot # printenv serverip
```

```
serverip=10.0.0.2
```



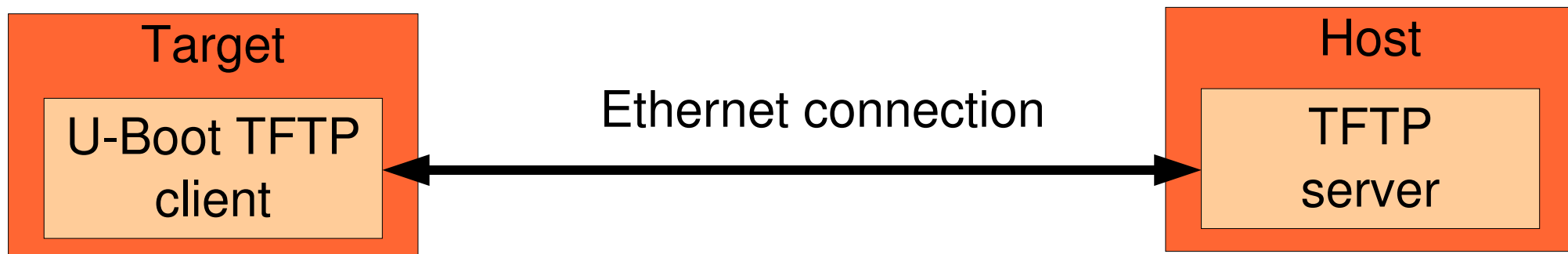
Environment variables (2)

- ▶ The value of the environment variables can be changed using the `setenv` command :
`u-boot # setenv serverip 10.0.0.2`
- ▶ Environment variable changes can be stored to flash using the `saveenv` command. The location in flash is defined at compile time in the U-Boot configuration file.
- ▶ You can even create small shell scripts stored in environment variables:
`setenv myscript 'tftp 0x21400000 uImage ;
bootm 0x21400000'`
- ▶ You can then execute the script:
`run myscript`



Transferring files to the target

- ▶ U-Boot is mostly used to load and boot a kernel image, but it also allows to change the kernel image and the root filesystem stored in flash.
- ▶ Files must be exchanged between the target and the development workstation. This is possible :
 - ▶ Through the network if the target has an Ethernet connection, and U-Boot contains a driver for the Ethernet chip. If so, the TFTP protocol can be used to exchange files
 - ▶ Through the serial line if no Ethernet connection is available.





Configuring tftp (1)

U-Boot is able to download files (kernel images, etc.) from a remote TFTP server.

Instructions for `xinetd` based systems (Fedora Core, Red Hat...)

- ▶ Install the `tftp-server` package if needed
- ▶ Remove `disable = yes` in `/etc/xinetd.d/tftp`
- ▶ Copy your image files to the `/tftpboot/` directory (or to the location specified in `/etc/xinetd.d/tftp`)
- ▶ You may have to disable `SELinux` in `/etc/selinux/config`
- ▶ Restart `xinetd`:
`sudo /etc/init.d/xinetd restart`



Configuring tftp (2)

On GNU/Linux systems based on **Debian: Ubuntu, Knoppix**

- ▶ Install the `tftpd-hpa` package if needed
- ▶ Set `RUN_DAEMON="yes"`
in `/etc/default/tftpd-hpa`
- ▶ Copy your images to `/var/lib/tftpboot`
- ▶ `/etc/hosts.allow:`
Replace `ALL : ALL@ALL : DENY`
by: `ALL : ALL@ALL : ALLOW`
- ▶ `/etc/hosts.deny:`
Comment out `ALL: PARANOID`
- ▶ Restart the server:
`sudo /etc/init.d/tftpd-hpa restart`



Testing tftp

- ▶ Strongly recommended to test your tftp server from your workstation itself before trying from your target.
- ▶ First get a tftp client if you don't have any yet:
`apt-get install tftp-hpa` (Debian / Ubuntu example)
- ▶ Now, try to download a file from your server:
`tftp localhost`
`> get uImage`
- ▶ If the download doesn't succeed immediately, fix your server configuration.



U-boot mkimage

- ▶ The kernel image that U-Boot loads and boots must be prepared, so that an U-Boot specific header is added in front of the image
- ▶ This is done with a tool that comes in U-Boot, `mkimage`
- ▶ Debian / Ubuntu: just install the `uboot-mkimage` package.
- ▶ Or, compile it by yourself: simply configure U-Boot for any board of any architecture and compile it. Then install `mkimage`:
`cp tools/mkimage /usr/local/bin/`
- ▶ The special target `uImage` of the kernel Makefile can then be used to generate a kernel image suitable for U-Boot.



Flashing a kernel image

- ▶ Compile your kernel and generate the U-Boot header using the `uImage` target, or directly using `mkimage`
- ▶ Copy the kernel image to the directory exported by the TFTP server
- ▶ On the board, in U-Boot, download the kernel image to memory :
`u-boot # tftp 8000 uImage`
- ▶ Unprotect NOR flash
`u-boot # protect off 1:0-4`
- ▶ Erase NOR flash
`u-boot # erase 1:0-4`
- ▶ Copy to NOR flash (`0x01000000`: first sector)
`u-boot # cp.b ${fileaddr} 1000000 ${filesize}`
- ▶ Restore NOR flash sector protection:
`u-boot # protect on 1:0-4`

See our practical labs for details handling NAND flash.



boot commands

- Specify kernel boot parameters:

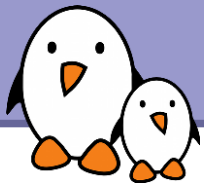
```
u-boot # setenv bootargs mem=64M \  
console=ttyS0,115200 init=/sbin/init \  
root=/dev/mtdblock0
```

Continues on
the same line

- Execute the kernel from a given physical address

(RAM or flash):

```
bootm 0x01030000
```



Practical lab – U-Boot

Time to start the practical lab !

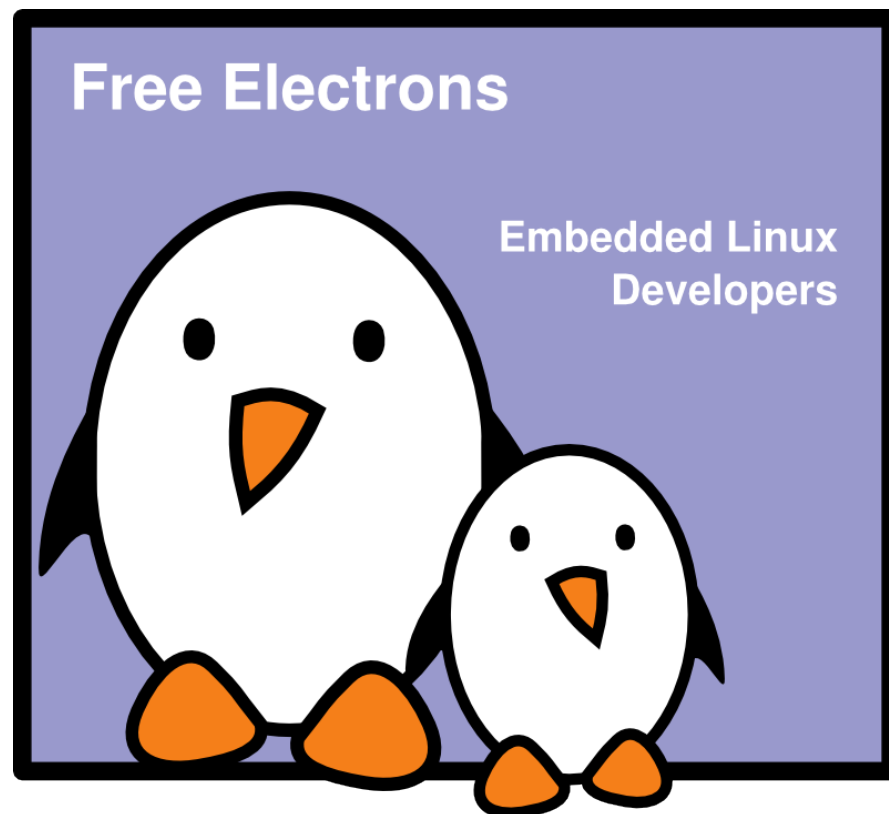
- ▶ Configure, compile and install U-Boot on a board
- ▶ Use basic U-Boot commands to download and run a sample program





Embedded Linux kernel usage

Michael Opdenacker
Thomas Petazzoni
Free Electrons





Contents

Kernel overview

- ▶ Linux features
- ▶ Linux versioning scheme and development process

Compiling and booting

- ▶ Linux kernel sources
- ▶ Kernel configuration
- ▶ Compiling the kernel
- ▶ Overall system startup
- ▶ Linux device files
- ▶ Cross-compiling the kernel

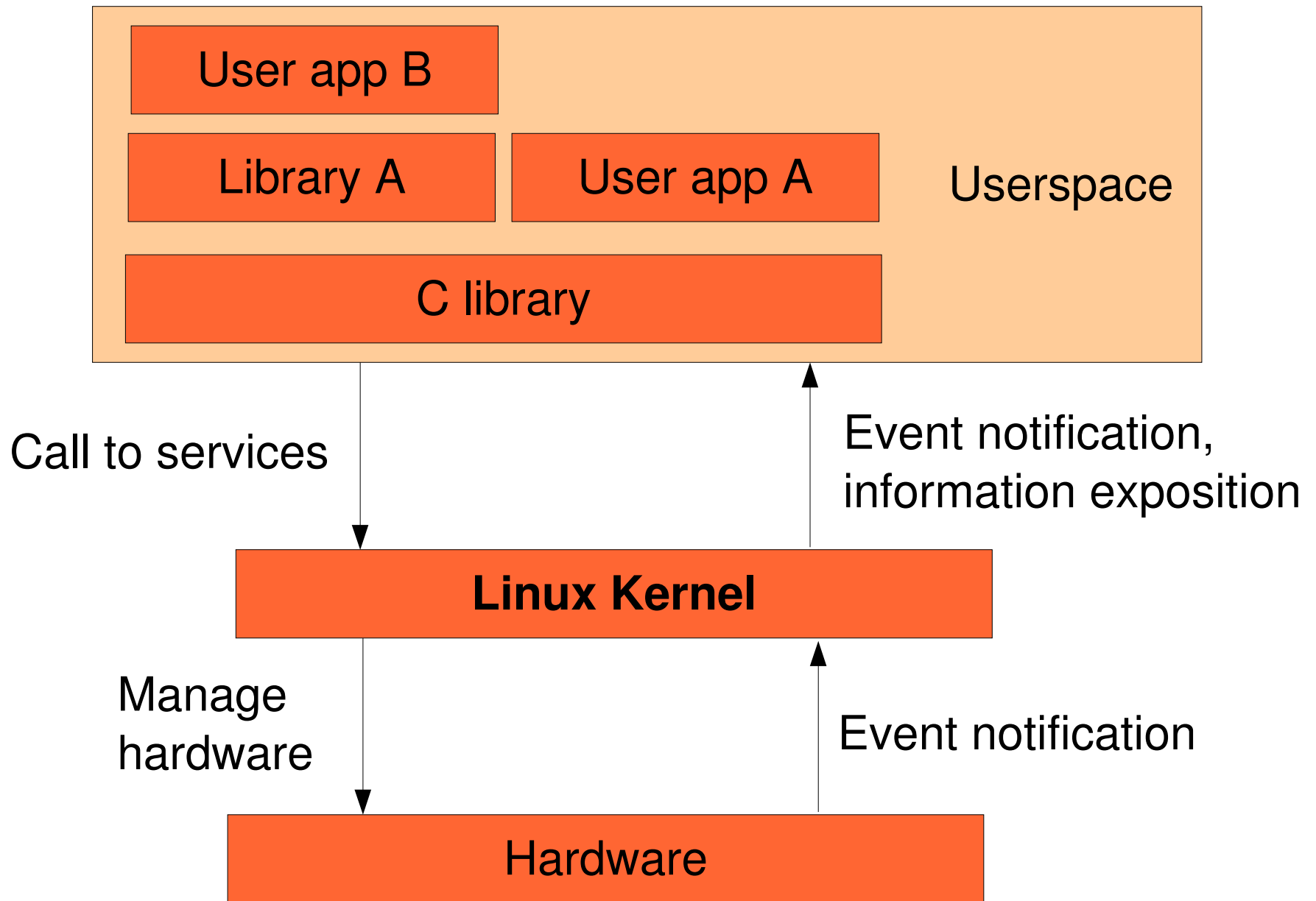


Kernel overview

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.

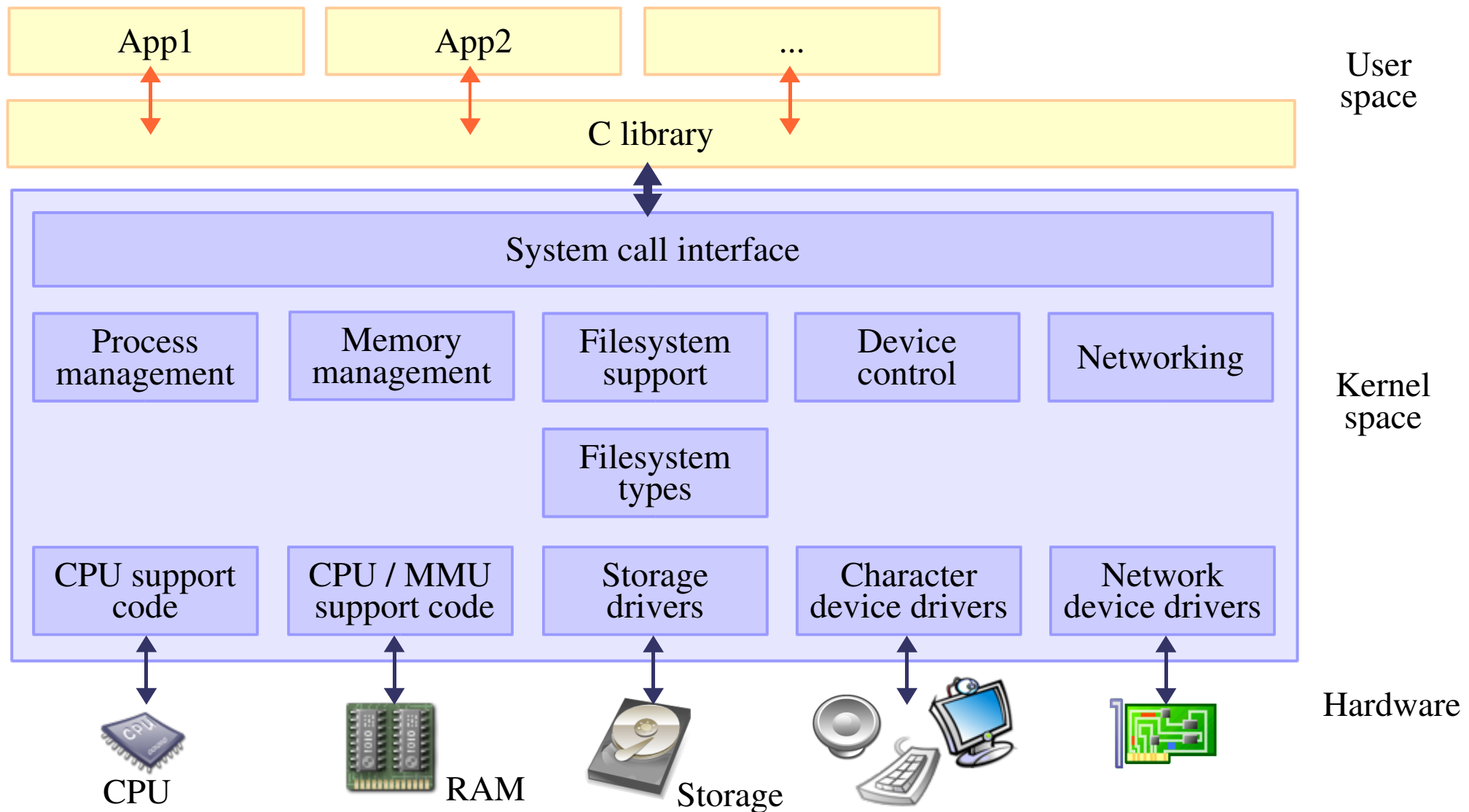


Supported hardware architectures

- ▶ 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`, `mips`, `mn10300`, `parisc`, `s390`, `sparc`, `um`,
`xtensa`
- ▶ 64 bit architectures:
`alpha`, `ia64`, `sparc64`
- ▶ 32/64 bit architectures
`powerpc`, `x86`, `sh`
- ▶ Find details in kernel sources: `arch/<arch>/Kconfig`, `arch/<arch>/README`, or `Documentation/<arch>/`



Kernel architecture





Mounting 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

... and many more! See by yourself!

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



Kernel overview

Linux versioning scheme and development process

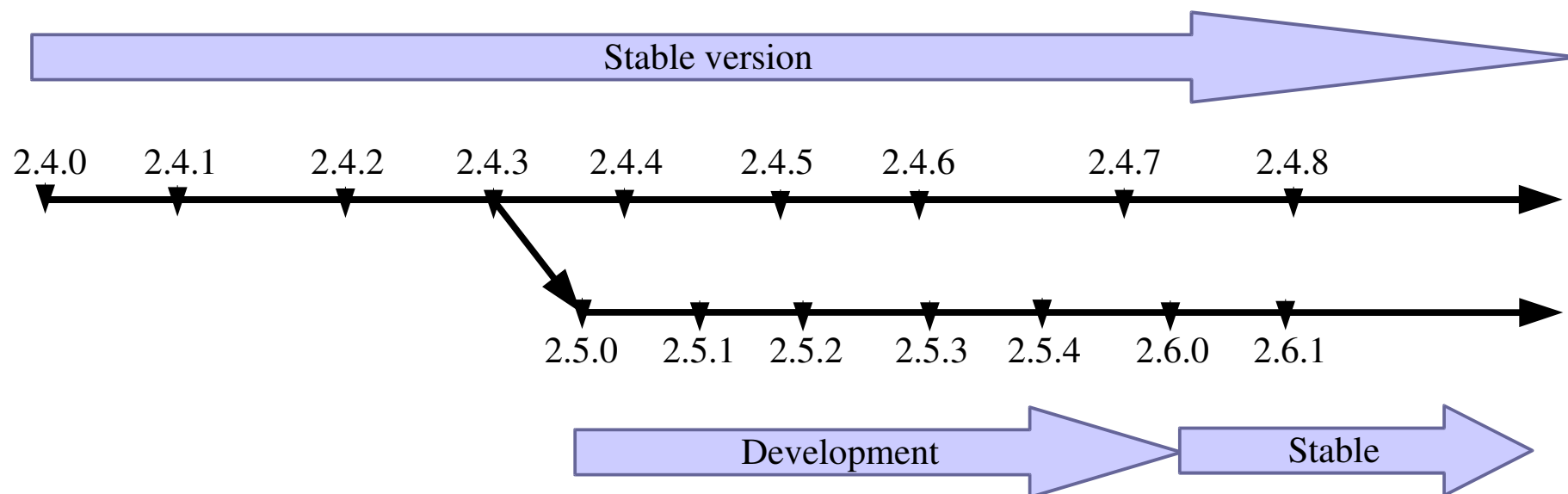


Until 2.6 (1)

- ▶ 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)

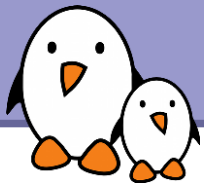


Note: in reality, many more minor versions exist inside the stable and development branches



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.



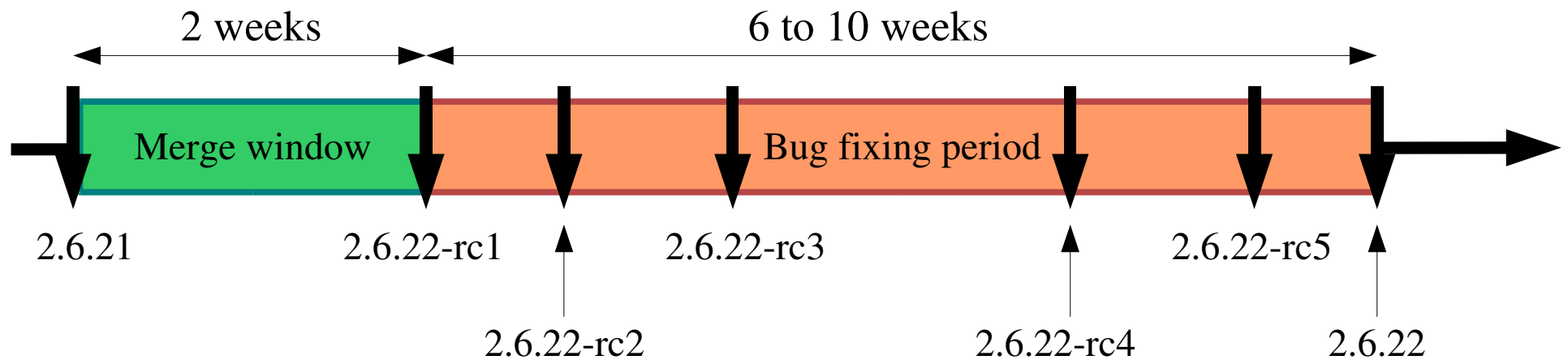
Changes since Linux 2.6 (2)

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





More stability for the 2.6 kernel tree

- ▶ Issue: 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.
- ▶ That's why Adrian Bunk proposed to maintain a 2.6.16 stable tree (March 2006), for as long as needed (years!). Latest update: July 2008 (checked in March, 2009).



What's new in each Linux release? (1)

```
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>

...

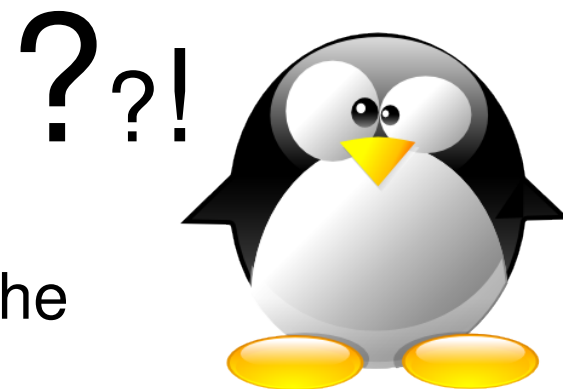


- ▶ The official list of changes for each Linux release is just a huge list of individual patches!
- ▶ Very difficult to find out the key changes and to get the global picture out of individual changes.



What's new in each Linux release? (2)

- ▶ Fortunately, a summary of key changes with enough details is available on <http://wiki.kernelnewbies.org/LinuxChanges>
- ▶ For each new kernel release, you can also get the changes in the kernel internal API: <http://lwn.net/Articles/2.6-kernel-api/>
- ▶ What's next?
[Documentation/feature-removal-schedule.txt](#) lists the features, subsystems and APIs that are planned for removal (announced 1 year in advance).






Compiling and booting Linux Linux kernel sources





The Linux Kernel Archives - Mozilla Firefox <2>

File Edit View History Bookmarks Tools Help

http://kernel.org/

 **The Linux Kernel Archives**

Welcome to the Linux Kernel Archives. This is the primary site for the Linux kernel source, but it has much more than just Linux kernels. [Frequently Asked Questions](#)

			
Protocol	Location	Protocol	Location
HTTP	http://www.kernel.org/pub/	HTTP	http://www.eu.kernel.org/pub/
FTP	ftp://ftp.kernel.org/pub/	FTP	ftp://ftp.eu.kernel.org/pub/
RSYNC	rsync://rsync.kernel.org/pub/	RSYNC	rsync://rsync.eu.kernel.org/pub/

The latest stable version of the Linux kernel is: [2.6.22.6](#) 2007-08-31 06:25 UTC [F](#) [V](#) [VI](#) [C](#) [Changelog](#)

The latest [prepatch](#) for the stable Linux kernel tree is: [2.6.23-rc5](#) 2007-09-01 06:35 UTC [B](#) [V](#) [VI](#) [C](#) [Changelog](#)

The latest [snapshot](#) for the stable Linux kernel tree is: [2.6.23-rc5-git1](#) 2007-09-04 19:01 UTC [B](#) [V](#) [C](#)

The latest 2.4 version of the Linux kernel is: [2.4.35.1](#) 2007-08-15 08:34 UTC [F](#) [V](#) [C](#) [Changelog](#)

The latest 2.2 version of the Linux kernel is: [2.2.26](#) 2004-02-25 00:28 UTC [F](#) [V](#) [C](#) [Changelog](#)

The latest [prepatch](#) for the 2.2 Linux kernel tree is: [2.2.27-rc2](#) 2005-01-12 23:55 UTC [B](#) [V](#) [VI](#) [C](#) [Changelog](#)

The latest [-mm patch](#) to the stable Linux kernels is: [2.6.23-rc4-mm1](#) 2007-09-01 04:31 UTC [B](#) [V](#) [C](#) [Changelog](#)

Find: [Next](#) [Previous](#) [Highlight all](#)

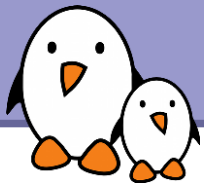
Done

Download kernel sources from <http://kernel.org>



Linux sources structure (1)

<code>arch/<arch></code>	Architecture specific code
<code>arch/<arch>/<mach></code>	Machine / board specific code
<code>block/</code>	Block layer core
<code>COPYING</code>	Linux copying conditions (GNU GPL)
<code>CREDITS</code>	Linux main contributors
<code>crypto/</code>	Cryptographic libraries
<code>Documentation/</code>	Kernel documentation. Don't miss it!
<code>drivers/</code>	All device drivers except sound ones (usb, pci...)
<code>fs/</code>	Filesystems (<code>fs/ext3/</code> , etc.)
<code>include/</code>	Kernel headers
<code>include/asm-<arch></code>	Architecture and machine dependent headers
<code>include/linux</code>	Linux kernel core headers
<code>init/</code>	Linux initialization (including <code>main.c</code>)



Linux sources structure (2)

<code>ipc/</code>	Code used for process communication
<code>Kbuild</code>	Part of the kernel build system
<code>kernel/</code>	Linux kernel core (very small!)
<code>lib/</code>	Misc library routines (zlib , crc32...)
<code>MAINTAINERS</code>	Maintainers of each kernel part. Very useful!
<code>Makefile</code>	Top Linux makefile (sets arch and version)
<code>mm/</code>	Memory management code (small too!)
<code>net/</code>	Network support code (not drivers)
<code>README</code>	Overview and building instructions
<code>REPORTING-BUGS</code>	Bug report instructions
<code>samples/</code>	Sample code (markers, kprobes, kobjects)
<code>scripts/</code>	Scripts for internal or external use
<code>security/</code>	Security model implementations (SELinux...)
<code>sound/</code>	Sound support code and drivers
<code>usr/</code>	Code to generate an initramfs cpio archive.



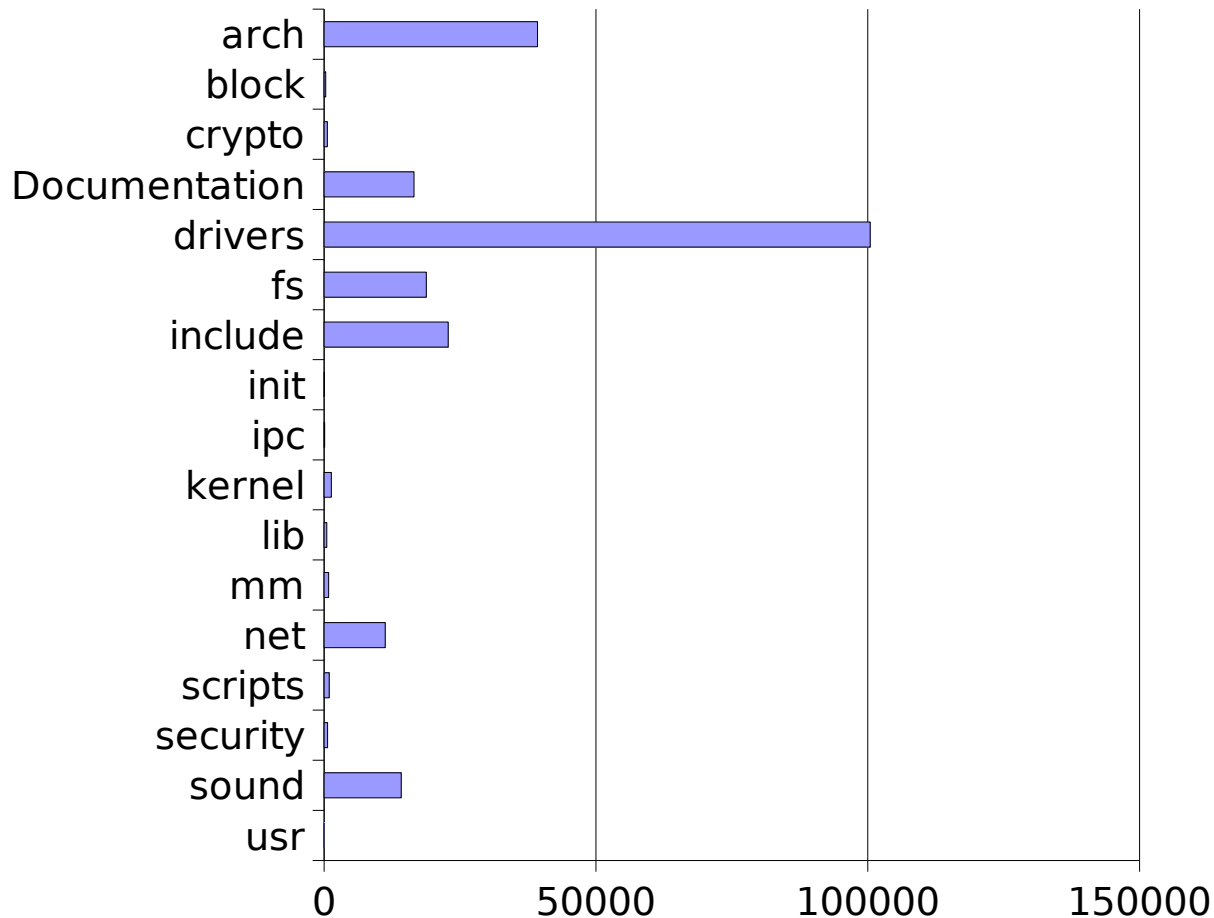
Linux kernel size (1)

- ▶ Linux 2.6.26 sources:
Raw size: 264 MB (25700 files, approx 8.5 million lines of code)
`gzip` compressed tar archive: 60 MB
`bzip2` compressed tar archive: 48 MB (better)
`lzma` compressed tar archive: 39 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 (2)

Size of Linux source directories (KB)



Linux 2.6.17

Measured with:

`du -s --apparent-size`



Getting Linux sources: 2 possibilities

Full sources

- ▶ The easiest way, but longer to download.
- ▶ Example:
<http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.7.tar.bz2>

Or patch against the previous version

- ▶ Assuming you already have the full sources of the previous version
- ▶ Example:
<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)



Downloading full kernel sources

Downloading from the command line

- ▶ With a web browser, identify the version you need on <http://kernel.org>
- ▶ In the right directory, download the source archive and its signature (copying the download address from the browser):

```
wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.7.tar.bz2
```

- ▶ Extract the contents of the source archive:

```
tar jxf linux-2.6.14.7.tar.bz2
```

~/.wgetrc config file for proxies:

```
http_proxy = <proxy>:<port>  
ftp_proxy = <proxy>:<port>  
proxy_user = <user> (if any)  
proxy_password = <passwd> (if any)
```




Downloading kernel source patches (1)

Assuming you already have the `linux-x.y.<n-1>` version

- ▶ Identify the patches you need on <http://kernel.org> with a web browser
- ▶ Download the patch files:

Patch from `2.6.13` to `2.6.14`

```
wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.bz2
```

Patch from `2.6.14` to `2.6.14.7` (latest stable fixes)

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.7.bz2
```



Downloading kernel source patches (2)

Apply the patches in the right order:

```
cd linux-2.6.13/  
bzipcat ../patch-2.6.14.bz2 | patch -p1  
bzipcat ../patch-2.6.14.7.bz2 | patch -p1  
cd ..  
mv linux-2.6.13 linux-2.6.14.7
```



Anatomy of a patch file

A patch file is the output of the `diff` command

```
diff -Nru a/Makefile b/Makefile
--- a/Makefile 2005-03-04 09:27:15 -08:00
+++ b/Makefile 2005-03-04 09:27:15 -08:00
@@ -1,7 +1,7 @@
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 11
-EXTRAVERSION =
+EXTRAVERSION = .1
NAME=Woozy Numbat

# *DOCUMENTATION*
```

← diff command line

← File date info

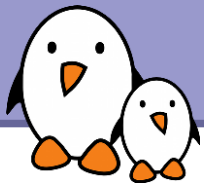
← Line numbers in files

← Context info: 3 lines before the change
Useful to apply a patch when line numbers changed

← Removed line(s) if any

← Added line(s) if any

← Context info: 3 lines after the change



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





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  
cd ../; mv linux-2.6.13 linux-2.6.14
```
- ▶ 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`
(**s**ilent)



Tested on `patch-2.6.23.bz2`



Ketchup - Easy access to kernel sources

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

- ▶ Makes it easy to get the latest version of a given kernel source tree ([2.4](#), [2.6](#), [2.6-rc](#), [2.6-git](#), [2.6-mm](#), [2.6-rt...](#))
- ▶ Only downloads the needed patches.
Reverts patches when needed to apply a more recent patch.
- ▶ Also checks the signature of sources and patches.



Ketchup examples

- ▶ Get the version in the current directory:

```
> ketchup -m  
2.6.10
```

- ▶ Upgrade to the latest stable version:

```
> ketchup -G 2.6-tip  
2.6.10 -> 2.6.12.5  
Applying patch-2.6.11.bz2  
Applying patch-2.6.12.bz2  
Applying patch-2.6.12.5.bz2
```

- ▶ You can get back to 2.6.8:

```
> ketchup -G 2.6.8
```

The `-G` option of ketchup disables source signature checking.

See our [Kernel sources annex](#) for details about enabling kernel source integrity checking.



On-line kernel documentation

<http://free-electrons.com/kerneldoc/>

- ▶ Provided for the latest kernel release
- ▶ Easier than downloading kernel sources to access documentation
- ▶ Indexed by Internet search engines
Makes kernel pieces of documentation easier to find!
- ▶ Unlike most other sites offering this service too, also includes an HTML translation of kernel documents in the DocBook format.

Never forget documentation in the kernel sources! It's a very valuable way of getting information about the kernel.



Practical lab – Kernel sources

- ▶ Get the sources
- ▶ Apply patches





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.
- ▶ Most useful commands to create this config file:
`make [xconfig|gconfig|menuconfig|oldconfig]`
- ▶ To modify a kernel in a GNU/Linux distribution:
config files usually released in `/boot/`, together with kernel images: `/boot/config-2.6.17-11-generic`
- ▶ The configuration file can also be found in the kernel itself:
> `zcat /proc/config.gz`
(if enabled in `General Setup -> Kernel .config support`)



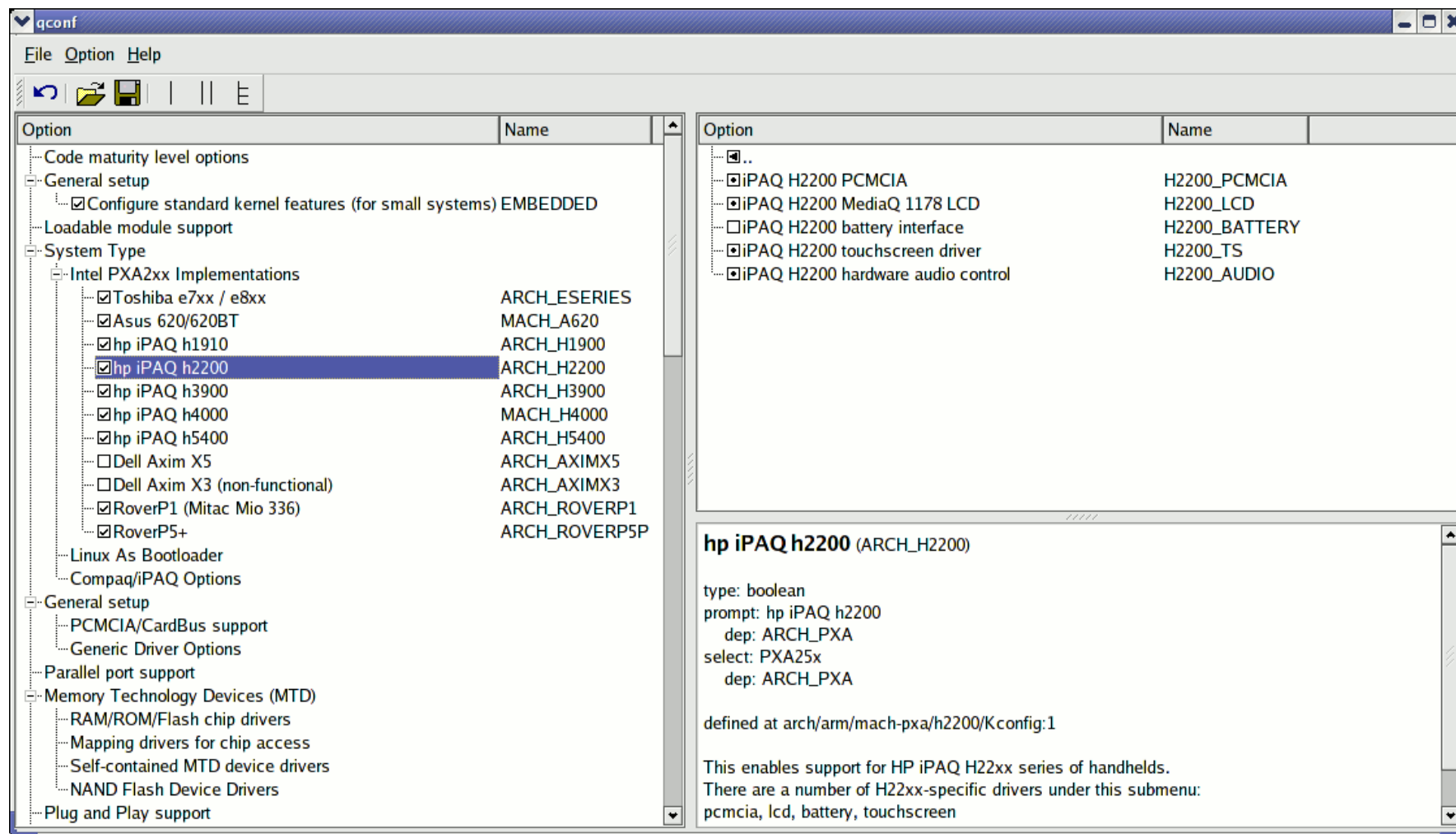
make xconfig

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:
`libqt3-mt-dev, g++`

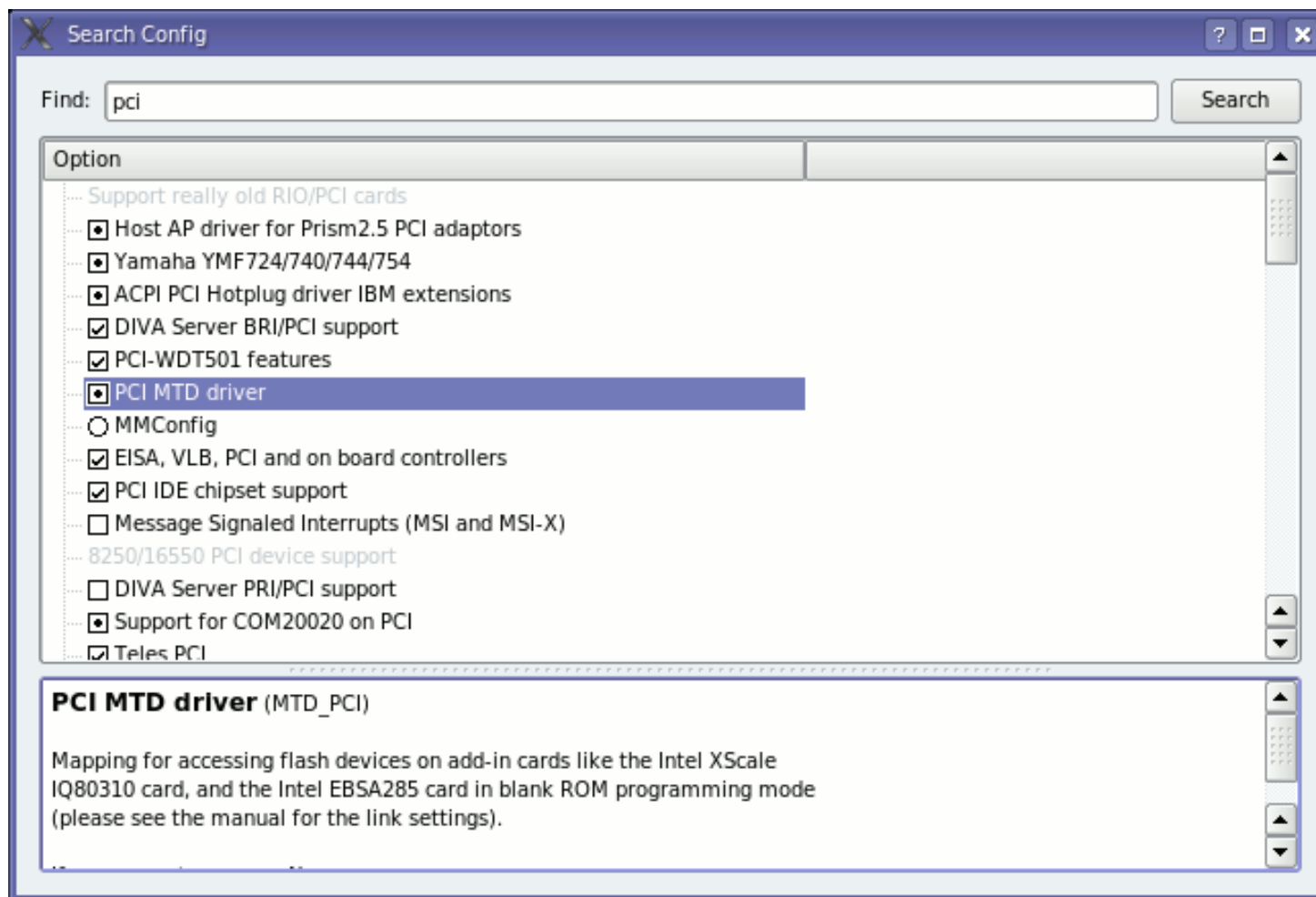


make xconfig screenshot





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`

- ☒ ISO 9660 CDROM file system support
- ☒ Microsoft Joliet CDROM extensions
- ☒ Transparent decompression extension
- ☒ UDF file system support

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
```

Section name
(helps to locate settings in the interface)

All parameters are prefixed
with CONFIG_

```
#  
# 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
```



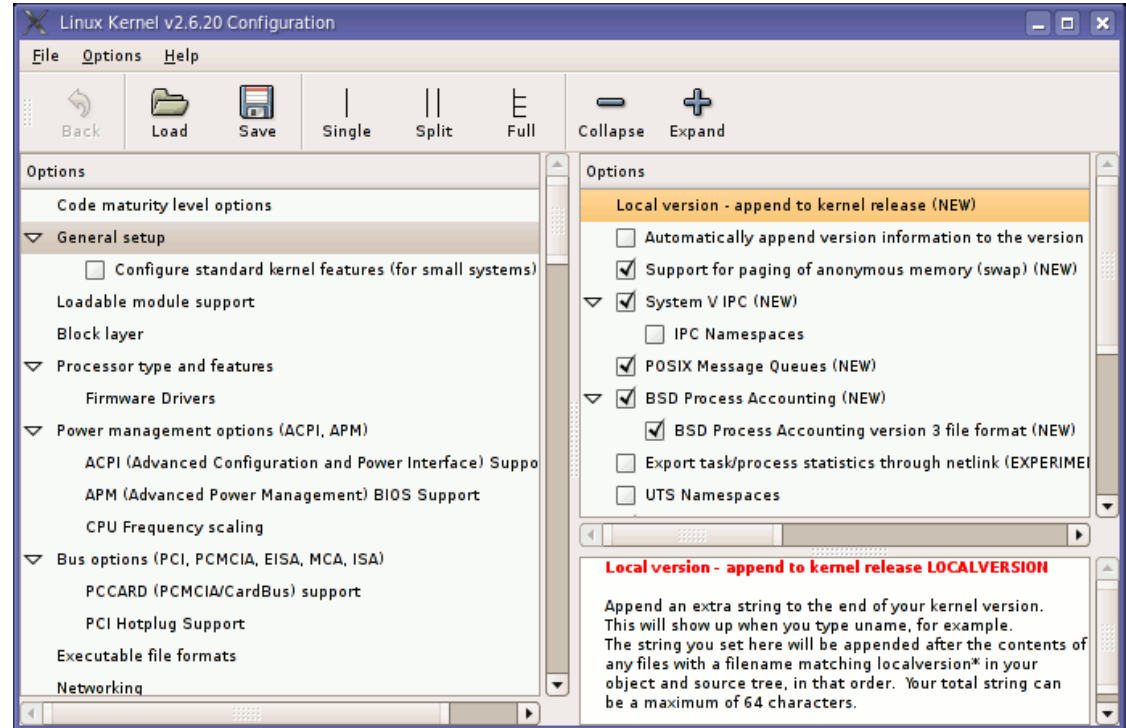

make gconfig

make gconfig

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

Just lacking a search functionality.

Required Debian packages:
libglade2-dev





make menuconfig

Linux Kernel v2.6.19 Configuration

Processor type and features

Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [] excluded <M> module < > module capable

```
[ ] Symmetric multi-processing support
    Subarchitecture Type (PC-compatible) --->
    Processor family (Pentium-Pro) --->
[*] Generic x86 support
[ ] HPET Timer Support
    Preemption Model (No Forced Preemption (Server)) --->
[ ] Local APIC support on uniprocessors
[ ] Machine Check Exception
< > Toshiba Laptop support
< > Dell laptop support
[ ] Enable X86 board specific fixups for reboot
<M> /dev/cpu/microcode - Intel IA32 CPU microcode support
< > /dev/cpu/*/msr - Model-specific register support
[*] /dev/cpu/*/cpuid - CPU information support
    Firmware Drivers --->
```

v(+)

<Select>

< Exit >

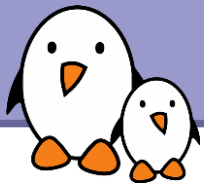
< Help >

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 obsolete symbols
- ▶ Asks for values for new symbols

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!

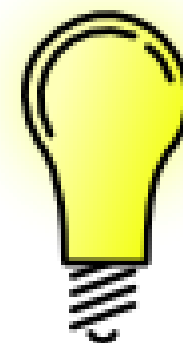


Undoing configuration changes

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.





make help

make help

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



Customizing the version string

- ▶ To identify your kernel image with others built from the same sources (but a different configuration), use the `LOCALVERSION` setting (in `General Setup`)

- ▶ Example:

```
#  
# General setup  
#  
CONFIG_LOCALVERSION="-acme1"
```

- ▶ The `uname -r` command
(in the running system) will return:
`2.6.20-acme1`



Compiling and booting Linux

Compiling the kernel



Compiling and installing the kernel

Compiling step

▶ `make`

Install steps

▶ `sudo make install`

▶ `sudo make modules_install`



Kernel compiling tips



- ▶ View the full (`gcc`, `ld...`) command line:
`make V=1`
- ▶ Clean-up generated files
(to force re-compiling drivers):
`make clean`
- ▶ Remove **all** generated files
Caution: also removes your `.config` file!
`make mrproper`
- ▶ Also remove editor backup and patch reject files:
(mainly to generate patches):
`make distclean`



Compiling faster on multiprocessor hosts

- ▶ If you are using a workstation with `n` processors, you may roughly divide your compile time by `n` by compiling several files in parallel
- ▶ `make -j <n>`
Runs several targets in parallel, whenever possible
- ▶ You can also use `make -j 2` or `make -j 3` on single processor workstations. Several parallel compile jobs can keep the processor busy while other processes are waiting for files to be read or written.
- ▶ Our recommendation:
`make -j <number of processors * 2>`



Generated files

Created when you run the `make` command

- ▶ `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”!



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



Files created by make modules_install (1)

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

- ▶ `build/`

Everything needed to build more modules for this kernel:

`Makefile`, `.config` file, module symbol information (`module.symVers`), kernel headers (`include/` and `include/asm/`)

- ▶ `kernel/`

Module `.ko` (Kernel Object) files, in the same directory structure as in the sources.



Files created by make modules_install (2)

`/lib/modules/<version>/` (continued)

► `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.

All the files in this directory are text files.

Don't hesitate to have a look by yourself!



Compiling the kernel in a nutshell

```
▶ make xconfig  
make  
sudo make install  
sudo make modules_install
```




Compiling and booting Linux Linux device files



Character device files

- ▶ Accessed through a sequential flow of individual characters
- ▶ Character devices can be identified by their **c** type
(**ls -l**):

```
crw-rw---- 1 root uucp    4,  64 Feb 23 2004 /dev/ttyS0
crw--w---- 1 jdoe tty    136,   1 Feb 23 2004 /dev/pts/1
crw----- 1 root root   13,  32 Feb 23 2004 /dev/input/mouse0
crw-rw-rw- 1 root root    1,   3 Feb 23 2004 /dev/null
```

- ▶ Example devices: keyboards, mice, parallel port, IrDA, Bluetooth port, consoles, terminals, sound, video...



Block device files

- ▶ Accessed through data blocks of a given size. Blocks can be accessed in any order.
- ▶ Block devices can be identified by their **b** type (**ls -l**):

```
brw-rw----    1 root disk      3,    1 Feb 23  2004 hda1
brw-rw----    1 jdoe floppy    2,    0 Feb 23  2004 fd0
brw-rw----    1 root disk      7,    0 Feb 23  2004 loop0
brw-rw----    1 root disk      1,    1 Feb 23  2004 ram1
brw-----    1 root root      8,    1 Feb 23  2004 sda1
```

- ▶ Example devices: hard or floppy disks, ram disks, loop devices...



Device major and minor numbers

As you could see in the previous examples, device files have 2 numbers associated to them:

- ▶ First number: *major* number
- ▶ Second number: *minor* number
- ▶ Major and minor numbers are used by the kernel to bind a driver to the device file. Device file names don't matter to the kernel!
- ▶ To find out which driver a device file corresponds to, or when the device name is too cryptic, see [Documentation/devices.txt](#).



Device file creation

- ▶ Device files are not created when a driver is loaded.

- ▶ They have to be created in advance:

```
sudo mknod /dev/<device> [c|b] <major> <minor>
```

- ▶ Examples:

```
sudo mknod /dev/ttyS0 c 4 64
```

```
sudo mknod /dev/hda1 b 3 1
```



Practical lab – Configuring and compiling

- ▶ Configure your kernel
- ▶ Compile it
- ▶ Boot it on a virtual PC
- ▶ Modify a root filesystem image by adding entries to the `/dev/` directory





Embedded Linux usage

Compiling and booting Linux
Overall system startup



Traditional booting sequence

Bootloader

- Executed by the hardware at a fixed location in ROM / Flash
- Initializes support for the device where the kernel image is found (local storage, network, removable media)
- Loads the kernel image in RAM
- Executes the kernel image (with a specified command line)

Kernel

- Uncompresses itself
- Initializes the kernel core and statically compiled drivers (needed to access the root filesystem)
- Mounts the root filesystem (specified by the `root` kernel parameter)
- Executes the first userspace program (specified by the `init` kernel parameter)

First userspace program

- Configures userspace and starts up system services





Drawbacks

- ▶ Assumption that all device drivers needed to mount the root filesystem (storage and filesystem drivers) are statically compiled inside the kernel.
- ▶ Assumption can be correct for most embedded systems, where the hardware is known and the kernel can be fine-tuned for the system.
- ▶ Assumption is mostly wrong for desktop and servers, since a single kernel image should support a wide range of devices and filesystems
 - ▶ More flexibility was needed
 - ▶ Modules have this flexibility, but they are not available before mounting the root filesystem
 - ▶ Need to handle complex setups (RAID, NFS, etc.)



Solution

- ▶ A solution is to include a small temporary root filesystem with modules, in the kernel itself. This small filesystem is called the *initramfs*.
- ▶ This initramfs is a gzipped cpio archive of this basic root filesystem
 - ▶ A gzipped cpio archive is a kind of zip file, with a much simpler format
- ▶ The initramfs scripts will detect the hardware, load the corresponding kernel modules, and mount the real root filesystem.
- ▶ Finally the initramfs scripts will run the init application in the real root filesystem and the system can boot as usual.
- ▶ The initramfs technique completely replaces init ramdisks (initrds). Initrds were used in Linux 2.4, but are no longer needed.



Booting sequence with initramfs

Bootloader

- Executed by the hardware at a fixed location in ROM / Flash
- Initializes support for the device where the images are found (local storage, network, removable media)
- Loads the kernel image in RAM
- Executes the kernel image (with a specified command line)

Kernel

- Uncompresses itself
- Initializes the kernel core and statically compiled drivers
- Uncompresses the initramfs cpio archive included in the kernel file cache (no mounting, no filesystem).
- If found in the initramfs, executes the first userspace program: `/init`

Userspace: `/init` script (what follows is just a typical scenario)

- Runs userspace commands to configure the device (such as network setup, mounting `/proc` and `/sys...`)
- Mounts a new root filesystem. Switch to it (`switch_root`)
- Runs `/sbin/init` (or sometimes a new `/linuxrc` script)

Userspace: `/sbin/init`

- Runs commands to configure the device (if not done yet in the initramfs)
- Starts up system services (daemons, servers) and user programs

unchanged



Initramfs features and advantages

- ▶ Root filesystem directly embedded in the kernel image, simple solution
- ▶ Very easy to create (at kernel build time).
No need for root permissions (for `mount` and `mknod`).
- ▶ Just a plain compressed cpio archive extracted in the file cache.
Neither needs a block nor a filesystem driver.
- ▶ Simpler to mount complex filesystems from flexible userspace scripts rather than from rigid kernel code. More complexity moved out to user-space!
- ▶ Possible to add non GPL files (firmware, proprietary drivers) in the filesystem. This is not linking, just file aggregation (not considered as a derived work by the GPL).



How to populate an initramfs

Using `CONFIG_INITRAMFS_SOURCE`
in kernel configuration (`General Setup` section)

- ▶ Either give an existing `cpio` archive
(file name ending with `.cpio`)
- ▶ Or give a directory to be archived.
- ▶ Any other regular file will be taken as a text specification file
(see next page).

see `Documentation/filesystems/ramfs-rootfs-initramfs.txt`
and `Documentation/early-userspace/README` in kernel sources.

See also <http://www.linuxdevices.com/articles/AT4017834659.html> for a nice
overview of initramfs (by Rob Landley).



Initramfs specification file example

```
dir /dev 755 0 0
nod /dev/console 644 0 0 c 5 1
nod /dev/loop0 644 0 0 b 7 0
dir /bin 755 1000 1000
file /bin/busybox /stuff/initramfs/busybox 755 0 0
slink /bin/sh busybox 777 0 0
dir /proc 755 0 0
dir /sys 755 0 0
dir /mnt 755 0 0
file /init /stuff/initramfs/init.sh 755 0 0
```

Annotations:

- major** (points to '5' in 'c 5 1')
- minor** (points to '1' in 'c 5 1')
- permissions** (points to '755' in '755 0 0' of the 'file /bin/busybox' line)
- user id** (points to '0' in '755 0 0' of the 'file /init' line)
- group id** (points to '0' in '755 0 0' of the 'file /init' line)

No need for `root` user access!



How to handle cpio archives

Useful when you want to build the kernel with a ready-made **cpio** archive, instead of letting the kernel do it for you.

▶ Extracting:

```
cpio -id < dir.cpio
```

▶ Creating:

```
cd dir
```

```
find . | cpio -H newc -o > ../dir.cpio
```

Note that the **-H newc** option is required to generate a **cpio** archive that can be used by the Linux kernel.



Summary

- ▶ For embedded systems, two interesting solutions
 - ▶ No initramfs: all needed drivers are included inside the kernel, and the final root filesystem is mounted directly
 - ▶ Everything inside the initramfs
- ▶ Another interesting solution is XIP
 - ▶ eXecute In Place
 - ▶ The kernel image is directly executed from the storage
 - ▶ Can be faster and save RAM.
However, the kernel image can't be compressed
 - ▶ Requires NOR flash



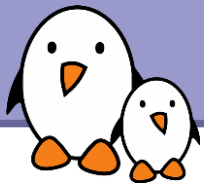
Compiling and booting Linux Kernel booting



Kernel command line parameters

Like most C programs, the Linux kernel accepts command line arguments

- ▶ Kernel command line arguments are part of the bootloader configuration settings.
- ▶ Useful to modify the behavior of the kernel at boot time, without having to recompile it.
- ▶ Useful to perform advanced kernel and driver initialization, without having to use complex user-space scripts.



Kernel command line example

HP iPAQ h2200 PDA booting example:

```
root=/dev/ram0 \
rw \
init=/linuxrc \
console=ttyS0,115200n8 \
console=tty0 \
ramdisk_size=8192 \
cachepolicy=writethrough
```

Root filesystem (first ramdisk)
Root filesystem mounting mode
First userspace program
Console (serial)
Other console (framebuffer)
Misc parameters...

Hundreds of command line parameters described on
[Documentation/kernel-parameters.txt](#)



Usefulness of rootfs on NFS

Once networking works, your root filesystem could be a directory on your GNU/Linux development host, exported by NFS (Network File System). This is very convenient for system development:

- ▶ Makes it very easy to update files (driver modules in particular) on the root filesystem, without rebooting. Much faster than through the serial port.
- ▶ Can have a big root filesystem even if you don't have support for internal or external storage yet.
- ▶ The root filesystem can be huge. You can even build native compiler tools and build all the tools you need on the target itself (better to cross-compile though).



NFS boot setup (1)

On the host (NFS server)

- ▶ Add the below line to your `/etc/exports` file:
`/home/rootfs 192.168.0.202(rw,no_root_squash,no_subtree_check)`
 client address NFS server options
- ▶ Start or restart your NFS server (Example: Debian, Ubuntu)
`/etc/init.d/nfs-kernel-server restart`



NFS boot setup (2)

On the target (NFS client)

- ▶ Compile your kernel with `CONFIG_NFS_FS=y`,
`CONFIG_IP_PNP=y` (configure IP at boot time)
and `CONFIG_ROOT_NFS=y`
- ▶ Boot the kernel with the below command line options:
`root=/dev/nfs`
 virtual device
`ip=192.168.1.111`
 local IP address
`nfsroot=192.168.1.110:/home/nfsroot`
 NFS server IP address Directory on the NFS server



First user-space program

- ▶ Specified by the `init` kernel command line parameter
Examples: `init=/bin/sh`
- ▶ Default value: `/sbin/init`
- ▶ Executed at the end of booting by the kernel
- ▶ Takes care of starting all other user-space programs (system services and user programs).
- ▶ Gets the `1` process number (pid)
Parent or ancestor of all user-space programs
The system won't let you kill it.
- ▶ Lightweight implementation of `/sbin/init` available through `BusyBox`. It doesn't have runlevels, but they are not often needed in embedded systems.



Embedded Linux usage

Compiling and booting Linux
Cross-compiling the kernel



Cross-compiling the kernel

When you compile a Linux kernel for another CPU architecture

- ▶ Much faster than compiling natively, when the target system is much slower than your GNU/Linux workstation.
- ▶ Much easier as development tools for your GNU/Linux workstation are much easier to find.
- ▶ To make the difference with a native compiler, cross-compiler executables are prefixed by the name of the target system, architecture and sometimes library. Examples:

`mips-linux-gcc`

`m68k-linux-uclibc-gcc`

`arm-linux-gnueabi-gcc`



Specifying a cross-compiler (1)

The CPU architecture and cross-compiler prefix are defined through the `ARCH` and `CROSS_COMPILE` variables in the toplevel `Makefile`.

- ▶ The `Makefile` defines `CC = $(CROSS_COMPILE)gcc`
See comments in `Makefile` for details
- ▶ The easiest solution is to modify the `Makefile`.
Example, ARM platform, cross-compiler: `arm-linux-gcc`
`ARCH` `?= arm`
`CROSS_COMPILE` `?= arm-linux-`



Specifying a cross-compiler (2)

Another solution is to set `ARCH` and `CROSS_COMPILE` through the `make` command line

► Explanation: any variable set through the `make` command line overrides any setting in the `Makefile`.

► Examples:

```
make ARCH=sh CROSS_COMPILE=sh-linux- xconfig
```

```
make ARCH=sh CROSS_COMPILE=sh-linux-
```

```
make ARCH=sh CROSS_COMPILE=sh-linux- modules_install
```

► Big drawback:

You should never forget these settings when you run `make`!
That's error prone and not convenient at all.



Specifying a cross compiler (3)

Another solution: set `ARCH` and `CROSS_COMPILE` as environment variables in your terminal:

```
export ARCH=arm
```

```
export CROSS_COMPILE=arm-linux-
```

- ▶ Can be set in project specific environments.
- ▶ Not hard-coded in the Makefile.
Do not interfere with patches.
- ▶ You don't forget to set them when you run any `make` command.
- ▶ Caution: only apply to shells in which these variables have been set.



Configuring the kernel

`make xconfig`

- ▶ Same as in native compiling.
- ▶ Don't forget to set the right board / machine type!



Ready-made config files

```
assabet_defconfig      integrator_defconfig  mainstone_defconfig
badge4_defconfig       iq31244_defconfig    mxlads_defconfig
bast_defconfig          iq80321_defconfig    neponset_defconfig
cerfcube_defconfig     iq80331_defconfig    netwinder_defconfig
clps7500_defconfig     iq80332_defconfig    omap_h2_1610_defconfig
ebsa110_defconfig      ixdp2400_defconfig   omnimeter_defconfig
edb7211_defconfig      ixdp2401_defconfig   pleb_defconfig
enp2611_defconfig      ixdp2800_defconfig   pxa255-idp_defconfig
ep80219_defconfig      ixdp2801_defconfig   rpc_defconfig
epxa10db_defconfig     ixp4xx_defconfig     s3c2410_defconfig
footbridge_defconfig  jornada720_defconfig shannon_defconfig
fortunet_defconfig     lart_defconfig        shark_defconfig
h3600_defconfig        lpd7a400_defconfig    simpad_defconfig
h7201_defconfig        lpd7a404_defconfig    smdk2410_defconfig
h7202_defconfig        lubbock_defconfig     versatile_defconfig
hackkit_defconfig      lus17200_defconfig
```

arch/arm/configs example



Using ready-made config files

- ▶ Default configuration files available for many boards / machines! Check if one exists in `arch/<arch>/configs/` for your target.
- ▶ Example: if you found an `acme_defconfig` file, you can run:
`make acme_defconfig`
- ▶ Using `arch/<arch>/configs/` is a very good good way of releasing a default configuration file for a group of users or developers.



Like all `make` commands, you must run `make <machine>_defconfig` in the toplevel source directory.



Cross-compiling setup

Example

- ▶ If you have an ARM cross-compiling toolchain in `/usr/local/arm/3.3.2/`
- ▶ You just have to add it to your Unix search path:
`export PATH=/usr/local/arm/3.3.2/bin:$PATH`
(Caution: the scope of this definition is limited to the current shell).

Choosing a toolchain

- ▶ See the [Documentation/Changes](#) file in the sources for details about minimum tool versions requirements.
- ▶ More about toolchains: Free Software tools for embedded systems training: <http://free-electrons.com/training/devtools/>



Building the kernel

- ▶ Run
`make`
- ▶ Copy
`arch/<arch>/boot/zImage`
to the target storage
- ▶ You can customize `arch/<arch>/boot/install.sh` so
that `make install` does this automatically for you.
- ▶ `make INSTALL_MOD_PATH=<dir>/ modules_install`
and copy `<dir>/` to `/lib/modules/` on the target storage



Cross-compiling summary

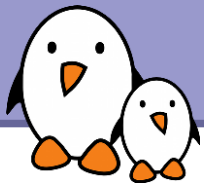
- ▶ Edit `Makefile`: set `ARCH` and `CROSS_COMPILE`
- ▶ Get the default configuration for your machine:
`make <machine>_defconfig` (if existing in `arch/<arch>/configs`)
- ▶ Refine the configuration settings according to your requirements:
`make xconfig`
- ▶ Add the cross-compiler path to your `PATH` environment variable
- ▶ Compile the kernel: `make`
- ▶ Copy the kernel image from `arch/<arch>/boot/` to the target
- ▶ Copy modules to a directory which you replicate on the target:
`make INSTALL_MOD_PATH=<dir> modules_install`



Practical lab – Cross-compiling



- ▶ Set up a cross-compiling environment
- ▶ Configure the kernel `Makefile` accordingly
- ▶ Cross-compile the kernel for an `arm` target platform
- ▶ On this platform, interact with the bootloader and boot your kernel.



Kernel modules



Loadable kernel modules

- ▶ Modules: add a given functionality to the kernel (drivers, filesystem support, and many others).
- ▶ Can be loaded and unloaded at any time, only when their functionality is needed.
- ▶ Useful to keep the kernel image size to the minimum (essential in GNU/Linux distributions for PCs).
- ▶ Also useful to reduce boot time: you don't spend time initializing devices and kernel features that you only need later.
- ▶ Caution: once loaded, have full access to the whole kernel address space. No particular protection.



Module dependencies

- ▶ Some kernel modules can depend on other modules, which need to be loaded first.
- ▶ Example: the `usb-storage` module depends on the `scsi_mod`, `libusual` and `usbcore` modules.
- ▶ Dependencies are described in `/lib/modules/<kernel-version>/modules.dep`



Kernel log

When a new module is loaded,
related information is available in the kernel log.

- ▶ The kernel keeps its messages in a circular buffer
(so that it doesn't consume more memory with many messages)
- ▶ Kernel log messages can be accessed from user space through
system calls, or through `/proc/kmsg`
- ▶ Kernel log messages are also displayed in the system console.



Accessing the kernel log

Many ways are available!

- ▶ Watch the system console

- ▶ `syslogd` / `klogd`

Daemon gathering kernel messages

in `/var/log/messages`

Follow changes by running:

`tail -f /var/log/messages`

Caution: this file grows!

Use `logrotate` to control this

- ▶ `dmesg` (“**d**iagnostics **m**essage”)

The most usual way in embedded systems.

Command found in all systems

Displays the kernel log buffer.

- ▶ `cat /proc/kmsg`

Waits for kernel messages and displays them.

Useful when none of the above user space programs are available (tiny system)



Module utilities (1)

- ▶ `modinfo <module_name>`
`modinfo <module_path>.ko`

Gets information about a module: parameters, license, description and dependencies.

Very useful before deciding to load a module or not.

- ▶ `sudo insmod <module_path>.ko`
Tries to load the given module.



Understanding module loading issues

- ▶ When loading a module fails,
`insmod` often doesn't give you enough details!
- ▶ Details are often available in the kernel log.
- ▶ Example:

```
> sudo insmod ./intr_monitor.ko
insmod: error inserting './intr_monitor.ko': -1
Device or resource busy
> dmesg
[17549774.552000] Failed to register handler for
irq channel 2
```



Module utilities (2)

- ▶ `sudo modprobe <module_name>`

Most common usage of `modprobe`: tries to load all the modules the given module depends on, and then this module. Lots of other options are available.

- ▶ `lsmod`

Displays the list of loaded modules

Compare its output with the contents of
`/proc/modules!`



Module utilities (3)

▶ `sudo rmmod <module_name>`

Tries to remove the given module.

Will only be allowed if the module is no longer in use (for example, no more processes opening a device file)

▶ `sudo modprobe -r <module_name>`

Tries to remove the given module and all dependent modules (which are no longer needed after the module removal)



Passing parameters to modules

- ▶ Through `insmod`:

```
sudo insmod ./snd-intel8x0m.ko index=-2
```



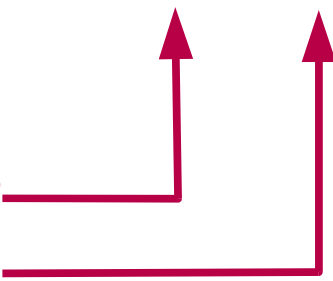
- ▶ Through `modprobe`:

Set parameters in `/etc/modprobe.conf` or in any file in `/etc/modprobe.d/`:

```
options snd-intel8x0m index=-2
```

- ▶ Through the kernel command line,
when the module is built statically into the kernel:

```
options snd-intel8x0m.index=-2
```

module name 
module parameter name 
module parameter value 

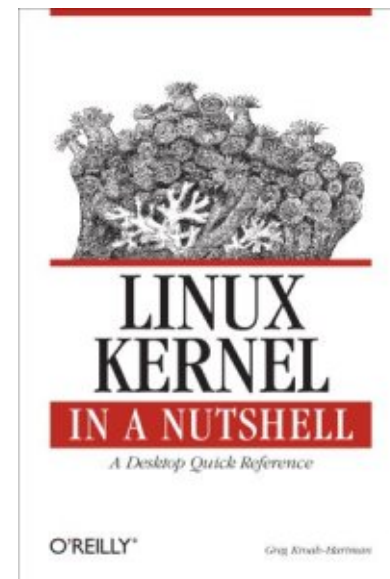


Useful reading

Linux Kernel in a Nutshell, Dec 2006



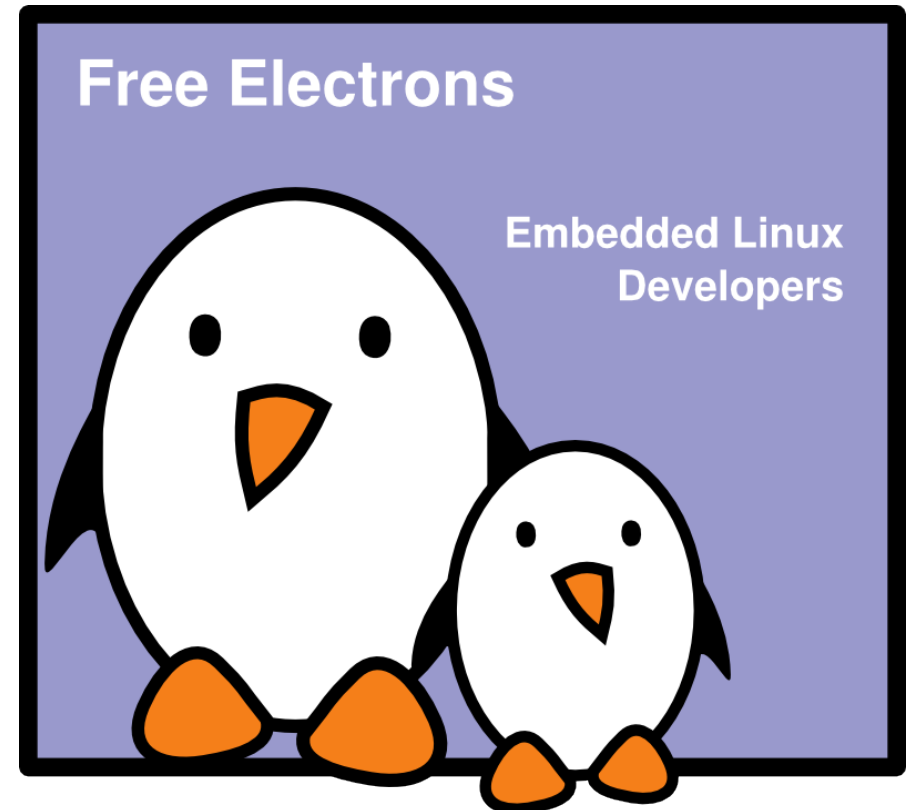
- ▶ By Greg Kroah-Hartman, O'Reilly
<http://www.kroah.com/lkn/>
- ▶ A good reference book and guide on configuring, compiling and managing the Linux kernel sources.
- ▶ **Freely available on-line!**
Great companion to the printed book
for easy electronic searches!
Available as single PDF file on
<http://free-electrons.com/community/kernel/lkn/>





BusyBox

Thomas Petazzoni
Michael Opdenacker
Free Electrons

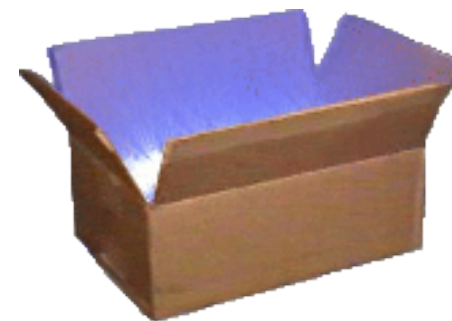




General purpose toolbox: BusyBox

<http://www.busybox.net/>

- ▶ Most Unix command line utilities within a single executable!
It even includes a web server!
- ▶ Sizes less than < 500 KB (statically compiled with [uClibc](#)) or less than 1 MB (statically compiled with [glibc](#)).
- ▶ Easy to configure which features to include.
- ▶ The best choice for
 - ▶ Initramfs / initrd with complex scripts
 - ▶ Small and medium size embedded systems



See <http://www-128.ibm.com/developerworks/linux/library/l-busybox/> for a nice introduction.



BusyBox commands!

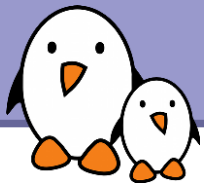
```
[, [[, addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, basename, bbconfig, bbsh,
brctl, bunzip2, busybox, bzip2, cal, cat, catv, chat, chatter, chcon, chgrp, chmod,
chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab,
cryptpw, cttyhack, cut, date, dc, dd, dealloct, delgroup, deluser, depmod, devfsd, df,
dhcprelay, diff, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg_deb, du, dumpkmap, dumpleases,
e2fsck, echo, ed, egrep, eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd,
false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep, find, findfs, fold, free,
freeramdisk, fsck, fsck_minix, ftpget, ftpput, fuser, getenforce, getopt, getsebool, getty,
grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock,
id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd, insmod, install, ip, ipaddr,
ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5,
klogd, lash, last, length, less, linux32, linux64, linuxrc, ln, load_policy, loadfont,
loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lzmacat,
makedevs, man, matchpathcon, md5sum, mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix,
mknod, mkswap, mktemp, modprobe, more, mount, mountpoint, msh, mt, mv, nameif, nc, netstat,
nice, nmeter, nohup, nslookup, od, openvt, parse, passwd, patch, pgrep, pidof, ping, ping6,
pipe_progress, pivot_root, pkill, poweroff, printenv, printf, ps, pscan, pwd, raidautorun,
rdate, rdev, readahead, readlink, readprofile, realpath, reboot, renice, reset, resize,
restorecon, rm, rmdir, rmmmod, route, rpm, rpm2cpio, rtcwake, run_parts, runcon, runlevel,
runsv, runsvdir, rx, script, sed, selinuxenabled, sendmail, seq, sestatus, setarch,
setconsole, setenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, setsid,
setuidgid, sh, shasum, showkey, slattach, sleep, softlimit, sort, split, start_stop_daemon,
stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl,
syslogd, tac, tail, tar, taskset, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, top,
touch, tr, traceroute, true, tty, ttysize, tune2fs, udhcpc, udhcpd, udpsvd, umount, uname,
uncompress, unexpand, uniq, unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode,
vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zcip
```

Commands available in BusyBox 1.13



Configuring BusyBox

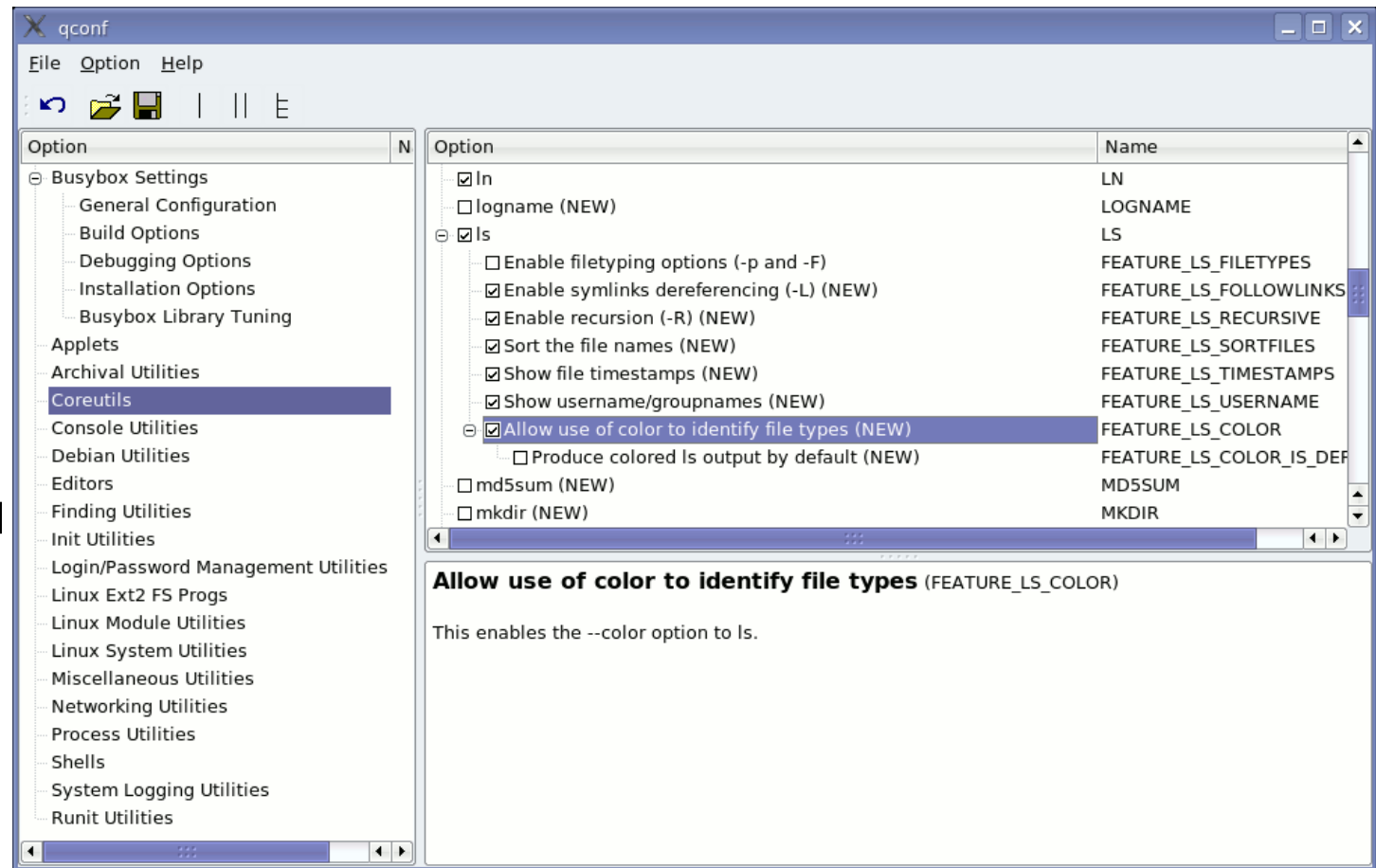
- ▶ Get the latest stable sources from <http://busybox.net>
- ▶ Configure **BusyBox** (creates a `.config` file):
 - ▶ `make defconfig`
Good to begin with **BusyBox**.
Configures **BusyBox** with all options for regular users.
 - ▶ `make allnoconfig`
Unselects all options. Good to configure only what you need.
- ▶ `make xconfig` (graphical) or `make menuconfig` (text)
Same configuration interfaces as the ones used by the Linux kernel.



BusyBox make xconfig

You can choose:

- ▶ the commands to compile,
- ▶ and even the command options and features that you need!





Compiling BusyBox

- ▶ Set the cross-compiler prefix in the configuration interface:
BusyBox Settings -> Build Options -> Cross Compiler prefix
Example: `arm-linux-`
- ▶ Set the installation directory in the configuration interface:
BusyBox Settings -> Build Options -> BusyBox installation prefix
- ▶ Add the cross-compiler path to the PATH environment variable:
`export PATH=/usr/local/arm/3.3.2/bin:$PATH`
- ▶ Compile BusyBox:
`make`
- ▶ Install it (this creates a Unix directory structure symbolic links to the `busybox` executable):
`make install`



Creating a new BusyBox applet

Useful to create your own, lightweight executables

- ▶ Take advantage of all BusyBox facilities (mainly implemented in the `libbb/` subdirectory).
- ▶ Achieves smaller results than if you made a standalone executable.
- ▶ You also take advantage of the configuring and (cross)compiling facilities brought by BusyBox.
- ▶ Caution: your application license will have to be GPL!



New BusyBox applet - readahead example (1)

Added lines... Tested with BusyBox 1.8.2

include/applets.h

```
USE_READAHEAD(APPLET(readahead, _BB_DIR_USR_BIN, _BB_SUID_NEVER))
```

include/usage.h

```
#define readahead_trivial_usage \  
    "[FILE]..."  
#define readahead_full_usage \  
    "Preloads FILE(s) in RAM cache so that subsequent reads" \  
    "for those files do not block on disk I/O."
```



New BusyBox applet - readahead example (2)

miscutils/Config.in

```
config CONFIG_READAHEAD
    bool "readahead"
    default n
    depends on LFS
    help
        Preload the given list of files in RAM cache so that
        subsequent reads on these files will not block on disk I/O.

        This applet just calls the readahead(2) system call
        on each file.
```

miscutils/Kbuild

```
lib-$(CONFIG_READAHEAD) += readahead.o
```



New BusyBox applet - readahead example (3)

miscutils/readahead.c

```
/*
 * readahead implementation for BusyBox
 * Copyright (C) 2006 Michael Opdenacker <michael@free-electrons.com>
 * Licensed under GPLv2 or later, see file License in this tarball for details.
 */

#include "busybox.h"

int readahead_main(int argc, char **argv)
{
    FILE *f;
    int retval = EXIT_SUCCESS;

    if (argc == 1) bb_show_usage();

    while (*++argv) {
        if ((f = fopen_or_warn(*argv, "r")) != NULL) {
            int r, fd=fileno(f);
            r = readahead(fd, 0, fdlength(fd));
            fclose(f);
            if (r >= 0) continue;
        }
        retval = EXIT_FAILURE;
    }
    return retval;
}
```

libbb functions



Alternative to BusyBox: embutils

<http://www.fefe.de/embutils/>

From the creator of [diet libc](#)

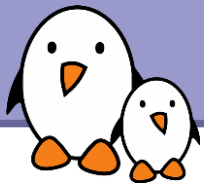
- ▶ A similar set of tiny utilities for embedded systems. Version 0.18 (Oct. 2006): 75 common commands are implemented.
- ▶ Can only be built statically with diet libc!
- ▶ Compared to BusyBox: Much less momentum, user and developer base. Still misses key commands and features.
- ▶ But can achieve smaller size than BusyBox on standalone executables.



Practical lab – A tiny embedded system

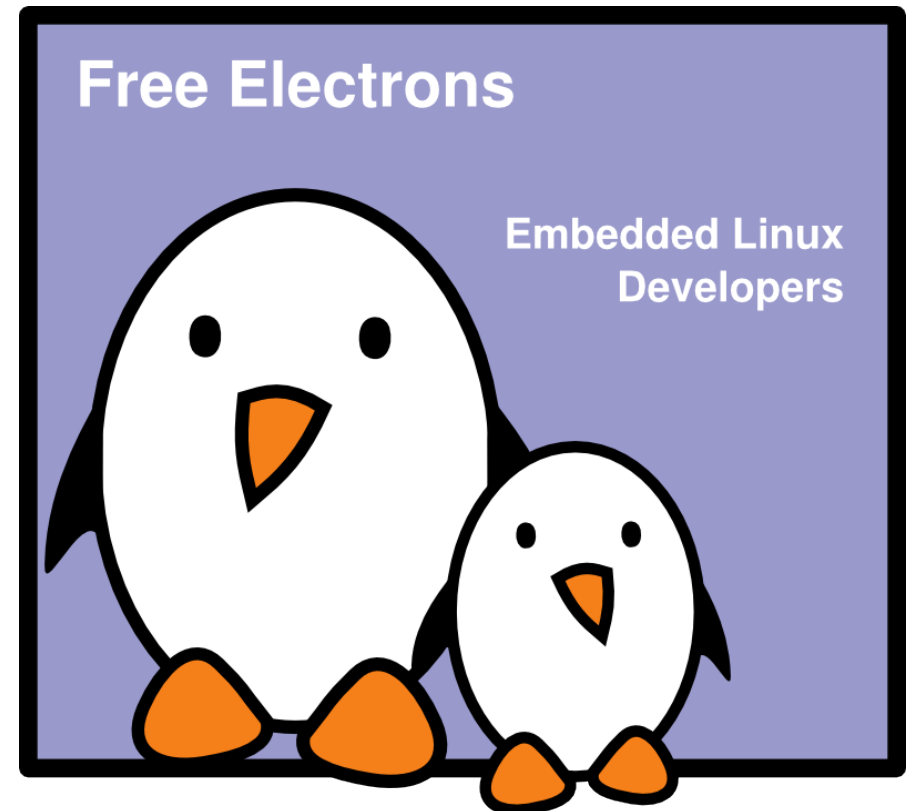
- ▶ Make Linux boot on a directory on your workstation, shared by NFS
- ▶ Create and configure a minimalistic Linux embedded system
- ▶ Install and use BusyBox
- ▶ System startup with `/sbin/init`
- ▶ Setup a simple web interface
- ▶ Use shared libraries





Block filesystems

Michael Opdenacker
Thomas Petazzoni
Free Electrons





Block devices

Block devices

- ▶ Floppy or hard disks
(SCSI, IDE)
- ▶ Compact Flash (seen as a regular IDE drive)
- ▶ RAM disks
- ▶ Loopback devices
(devices used to mount filesystem images with `mount -o loop`)



Traditional block filesystems

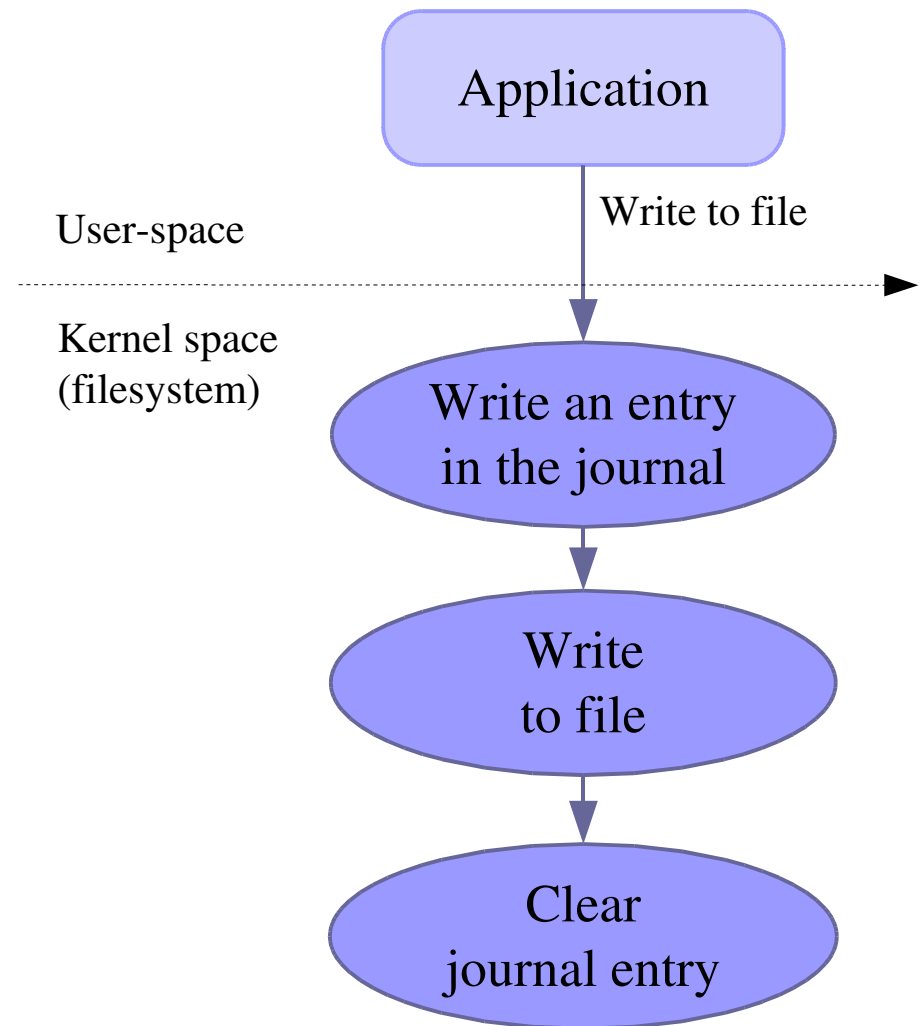
Traditional filesystems

- ▶ Hard to recover from crashes. Can be left in a corrupted (“half finished”) state after a system crash or sudden power-off.
- ▶ `ext2`: traditional Linux filesystem
(repair it with `fsck.ext2`)
- ▶ `vfat`: traditional Windows filesystem
(repair it with `fsck.vfat` on GNU/Linux or `Scandisk` on Windows)



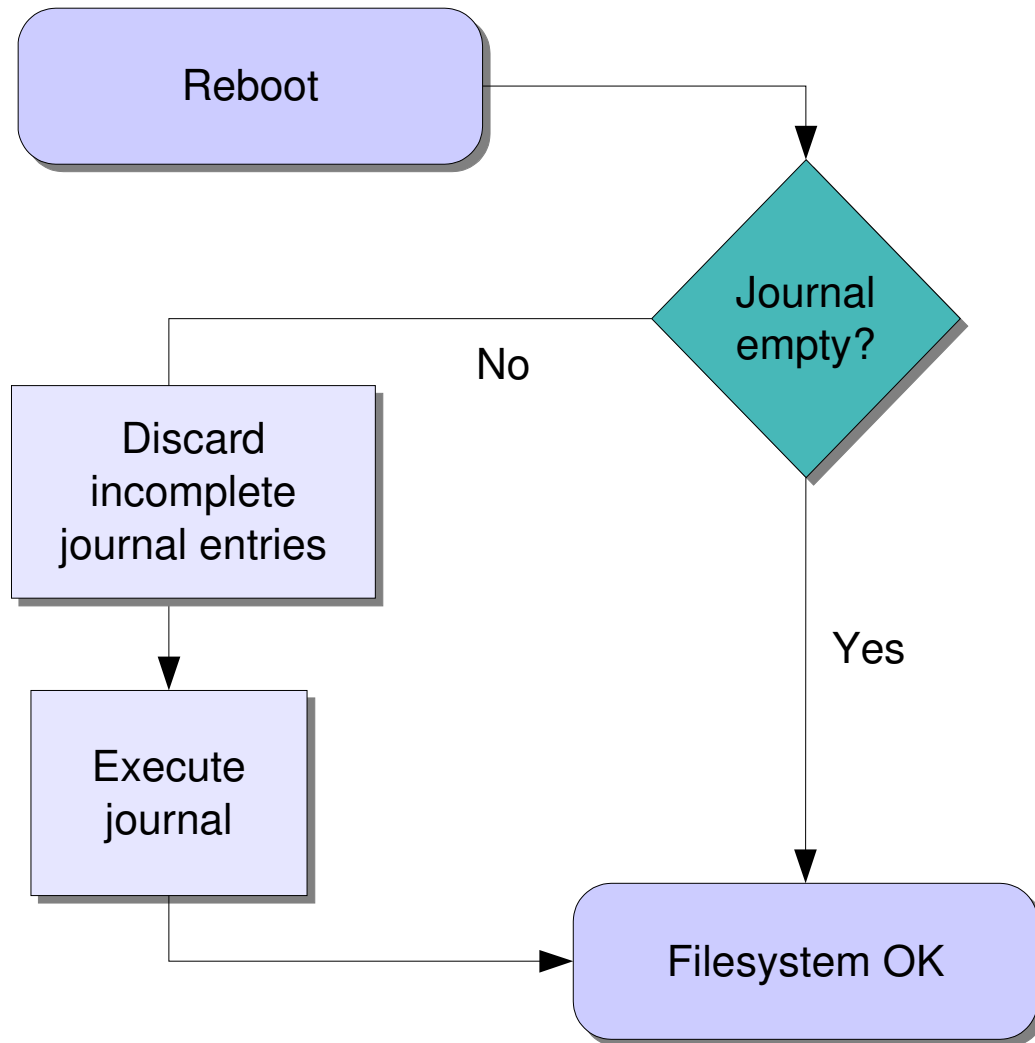
Journalled filesystems

- ▶ Designed to stay in a correct state even after system crashes or a sudden power-off
- ▶ All writes are first described in the journal before being committed to files





Filesystem recovery after crashes



- ▶ Thanks to the journal, the filesystem is never left in a corrupted state
- ▶ Recently saved data could still be lost



Journalled block filesystems

Journalled filesystems

- ▶ **ext3**: **ext2** with journal extension
ext4: the next generation with many improvements.
Ready for production. See <http://kernelnewbies.org/Ext4>
- ▶ **reiserFS**: most innovative (fast and extensible)
Caution: needs at least 32 MB!
reiser4: the latest version.
Available through patches (not in mainstream yet).
- ▶ Others: **JFS** (IBM), **XFS** (SGI)



Cramfs

- ▶ Simple, small, read-only compressed filesystem designed for embedded systems .
- ▶ Maximum filesystem size: 256 MB
- ▶ Maximum file size: 16 MB

See [Documentation/filesystems/cramfs.txt](#) in kernel sources.

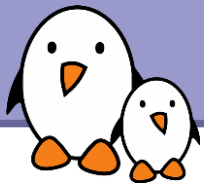


Squashfs

Squashfs: <http://squashfs.sourceforge.net>

- ▶ A must-use replacement for **Cramfs**! Also read-only.
- ▶ Maximum filesystem and file size: 2^{64} bytes!
- ▶ Achieves better compression and much better performance. It supports block sizes up to 64 K (instead of 4K) for greater compression, and even detects duplicate files!
- ▶ Available in mainstream Linux since version 2.6.29. Patches available for all earlier versions.

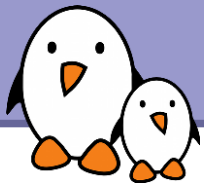
Benchmarks: (roughly 3 times smaller than **ext3**, and 2-4 times faster)
http://elinux.org/Squash_Fs_Comparisons



Squashfs - How to use

Very simple!

- ▶ On your workstation, create your filesystem image:
`mksquashfs rootdir rootdir.sqfs`
- ▶ Caution: if the image already exists remove it first,
or use the `-noappend` option.
- ▶ Let's assume your partition on the target is in `/dev/sda1`
- ▶ On your target, copy the filesystem image on the device
(CAUTION: don't run this on your workstation!
You could destroy critical system partitions.)
`dd if=rootdir.sqfs of=/dev/sda1`
- ▶ Mount your filesystem:
`mount -t squashfs /dev/sda1 /mnt/root`



tmpfs

Useful to store temporary data in RAM:
system log files, connection data, temporary files...

- ▶ Don't use ramdisks! They have many drawbacks: fixed in size, Remaining space not usable as RAM, files duplicated in RAM (in the block device and file cache)!
- ▶ tmpfs configuration: [File systems -> Pseudo filesystems](#)
Lives in the Linux file cache. Doesn't waste RAM: grows and shrinks to accommodate stored files. Saves RAM: no duplication; can swap out pages to disk when needed.
- ▶ How to use: choose a name to distinguish the various tmpfs instances you could have. Examples:

```
mount -t tmpfs varrun /var/run  
mount -t tmpfs udev /dev
```

See [Documentation/filesystems/tmpfs.txt](#) in kernel sources.



Mixing read-only and read-write filesystems

Good idea to split your block storage into

- ▶ A compressed read-only partition (**Squashfs**)
Typically used for the root filesystem (binaries, kernel...).
Compression saves space. Read-only access protects your system from mistakes and data corruption.
- ▶ A read-write partition with a journaled filesystem (like **ext3**)
Used to store user or configuration data.
Guarantees filesystem integrity after power off or crashes.
- ▶ Ram storage for temporary files (**tmpfs**)

Squashfs
read-only
compressed
root
filesystem

ext3
read-write
user and
configuration
data

tmpfs
read-write
volatile data

Block Storage

RAM



Practical lab – Block filesystems

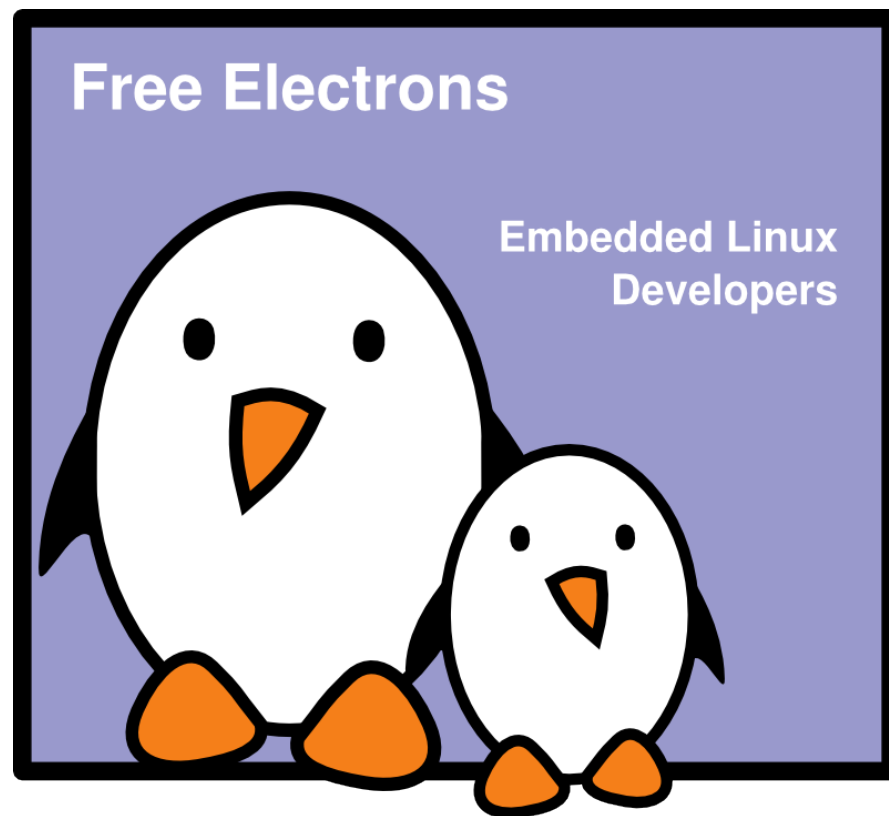
- ▶ Creating partitions on your block storage
- ▶ Booting your system with a mix of filesystems: SquashFS for applications, ext3 for configuration and user data, and tmpfs for temporary system files.





Flash filesystems

Michael Opdenacker
Thomas Petazzoni
Free Electrons

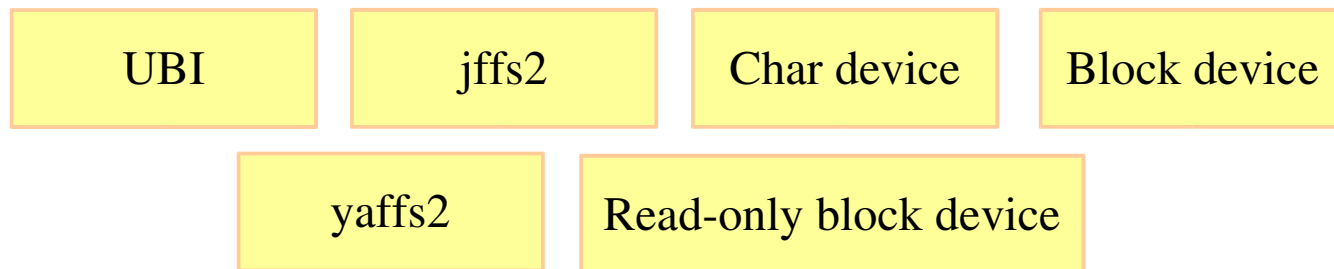




The MTD subsystem

Linux filesystem interface

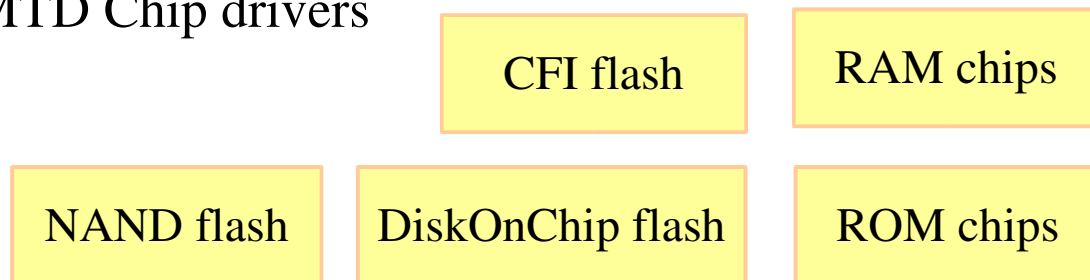
MTD “User” modules



Flash Translation Layers
for block device emulation
Caution: patented algorithms!

FTL NFTL INFTL

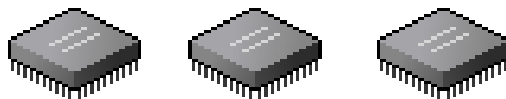
MTD Chip drivers



Block device Virtual memory

Virtual devices appearing as
MTD devices

Memory devices hardware





MTD - How to use

Creating the device nodes

Char device files

- ▶ `mknod /dev/mtd0 c 90 0`
- ▶ `mknod /dev/mtd1 c 90 2` (Caution: 2 and not 1)
- ▶ `mknod /dev/mtd2 c 90 4` (Caution: 4 and not 2)

Block device files

- ▶ `mknod /dev/mtdblock0 b 31 0`
- ▶ `mknod /dev/mtdblock1 b 31 1`
- ▶ `mknod /dev/mtdblock2 b 31 2`



Definition of MTD partitions

MTD partitions are defined in the kernel, in the board definitions:
`arch/arm/mach-at91/board-usb-a9263.c` example:

```
static struct mtd_partition __initdata ek_nand_partition[] = {
    {
        .name      = "Linux Kernel",
        .offset     = 0,
        .size       = SZ_16M,
    },
    {
        .name      = "Root FS",
        .offset     = MTDPART_OFS_NXTBLK,
        .size       = 120 * SZ_1M,
    },
    {
        .name      = "FS",
        .offset     = MTDPART_OFS_NXTBLK,
        .size       = 120 * SZ_1M,
    }
};
```



Modifying MTD partitions (1)

- ▶ MTD partitions can fortunately be defined through the kernel command line.
- ▶ First need to find the name of the MTD device.

Look at the kernel log at boot time:

```
NAND device: Manufacturer ID: 0xec, Chip ID:  
0xda (Samsung NAND 256MiB 3,3V 8-bit)
```

```
Scanning device for bad blocks
```

```
Bad eraseblock 2000 at 0x0fa00000
```

```
Creating 3 MTD partitions on "atmel_nand":
```

```
0x00000000-0x01000000 : "Linux Kernel"
```

```
0x01000000-0x08800000 : "Root FS"
```

```
0x08800000-0x10000000 : "FS"
```



Modifying MTD partitions (2)

- ▶ You can now use the `mtdparts` kernel boot parameter
- ▶ Example:
`mtdparts=atmel_nand:2m(kernel)ro,1m(rootfs)ro,-(data)`
- ▶ We've just defined 3 partitions in the `atmel_nand` device:
 - ▶ `kernel` (2M)
 - ▶ `rootfs` (1M)
 - ▶ `data`
- ▶ `ro` lists the partition as read only
- ▶ `-` is used to use all the remaining space.



jffs2

- ▶ Today's standard filesystem for MTD flash
- ▶ Nice features:
 - ▶ On the fly compression. Saves storage space and reduces I/O.
 - ▶ Power-down reliable.
 - ▶ Implements wear-leveling
- ▶ Drawbacks: doesn't scale well
 - ▶ Mount time depending on filesystem size: the kernel has to scan the whole filesystem at mount time, to read which block belongs to each file.
 - ▶ Keeping this information in RAM is memory hungry too.

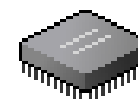
Standard file
API



JFFS2
filesystem



MTD driver



Flash chip



New jffs2 features

▶ `CONFIG_JFFS2_SUMMARY`

Dramatically reduces mount time. No longer needed to scan the whole filesystem at mount time, because collected information is now stored in flash.

▶ New jffs2 compression options:

- ▶ Now supports lzo compression, and not only zlib (and also the rtime and rubin compressors)
- ▶ Can try all compressors and keep the one giving the best results
- ▶ Can also give preference to lzo, to the expense of size, because lzo has the fastest decompression times.



jffs2 - How to use

- ▶ Compile mtd-tools if needed:
`git-clone git://git.infradead.org/mtd-utils.git`
- ▶ Erase and format a partition with jffs2:
`flash_eraseall -j /dev/mtd2`
- ▶ Mount the partition:
`mount -t jffs2 /dev/mtdblock2 /mnt/flash`
- ▶ Fill the contents by writing
- ▶ Or, use an image (see next page to produce it):
`nandwrite -p /dev/mtd2 rootfs.jffs2`



How to create a jffs2 image

- ▶ `mkfs.jffs2` available in `mtd-utils`
- ▶ First, find the erase block size from U-boot `nand info`:
`Device 0: NAND 256MiB 3,3V 8-bit, sector size 128 KiB`
- ▶ Then, find or choose the size of your MTD partition
Example: 16M (0x1000000)
- ▶ Then create the image on your workstation:
`mkfs.jffs2 --no-cleanmarkers --pad=0x1000000
--eraseblock=128 -d rootfs/ -o rootfs.jffs2`
- ▶ The `--no-cleanmarkers` option is for NAND flash only.
Without it, we got lots of warnings at jffs2 partition mount time.



Initializing jffs2 partitions from U-boot

Doesn't need `mtd-tools` on your target!

- ▶ Create a JFFS2 image on your workstation
- ▶ In the U-Boot prompt:
 - ▶ Download `fs.img` to RAM with `tftp`
 - ▶ Flash it inside an MTD partition
(exact instructions depending on flash type, NOR or NAND, reuse the instructions used to flash your kernel)
 - ▶ If you boot on a jffs2 root filesystem, add `root=/dev/mtdblock<x>` and `rootfstype=jffs2` to the Linux command line arguments.



Mounting a jffs2 image

Useful to create or edit `jffs2` images on your GNU / Linux PC!

► Mounting an MTD device as a loop device is a bit complex task.

Here's an example for `jffs2`, for your reference:

```
losetup /dev/loop0 root.jffs2
mknod /dev/mtdblock0 b 31 0
mkdir /mnt/jffs2
modprobe block2mtd
echo "/dev/loop0" > /sys/module/block2mtd/parameters/block2mtd
mount -t jffs2 /dev/mtdblock0 /mnt/jffs2
```

(if not done yet)

(example mount point, if not done yet)



yaffs2

<http://www.yaffs.net/>

- ▶ Supports both NAND and NOR flash
- ▶ No compression
- ▶ Wear leveling, ECC, power failure resistant
- ▶ Fast boot time
- ▶ Code available separately through CVS
(Dual GPL / Proprietary license
for non Linux operating systems)

Standard file
API

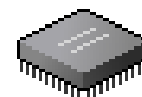
— — — —

YAFFS2
filesystem

— — — —

MTD driver

— — — —



Flash chip



yaffs2 - How to use

- ▶ Erase a partition:

```
flash_eraseall /dev/mtd2
```

- ▶ Format the partition:

```
sleep (any command can do!)
```

- ▶ Mount the partition:

```
mount -t yaffs2 /dev/mtdblock2 /mnt/flash
```



UBI (1)

Unsorted Block Images

- ▶ <http://www.linux-mtd.infradead.org/doc/ubi.html>
- ▶ Volume management system on top of MTD devices.
- ▶ Allows to create multiple logical volumes and spread writes across all physical blocks.
- ▶ Takes care of managing the erase blocks and wear leveling. Makes filesystem easier to implement.



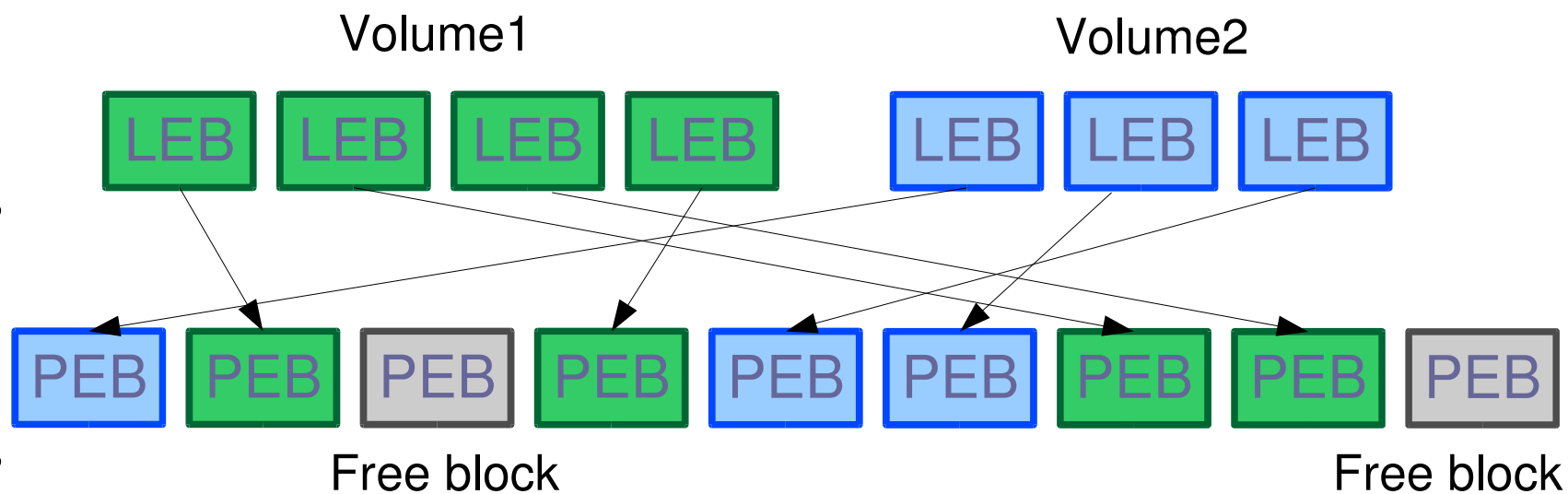
UBI (2)

UBI

Logical
Erase Blocks

MTD

Physical
Erase Blocks





UBI - How to use (1)

- ▶ Erase your flash partition while preserving your erase counters
`ubiformat /dev/mtd1`
See <http://www.linux-mtd.infradead.org/faq/ubi.html> if you face problems
- ▶ Need to create a `/dev/ubi_ctrl` char device (if you don't have `udev`)
Major and minor number allocated in the kernel. Find these numbers in `/sys/class/misc/ubi_ctrl/dev` (e.g.: `10:63`)
Or run `ubinfore`:

UBI version:	1
Count of UBI devices:	1
UBI control device major/minor:	10:63
Present UBI devices:	ubi0



UBI - How to use (2)

- ▶ Attach UBI to one (of several) of the MTD partitions:

```
ubiattach /dev/ubi_ctrl -m 1
```

- ▶ Find the major and minor numbers used by UBI:

```
cat /sys/class/ubi/ubi0/dev (e.g. 253:0)
```

- ▶ Create the UBI device file:

```
mknod /dev/ubi0 c 253 0
```




UBIFS

<http://www.linux-mtd.infradead.org/doc/ubifs.html>

- ▶ The next generation of the jffs2 filesystem, from the same linux-mtd developers.
- ▶ Available in Linux 2.6.27
- ▶ Works on top of UBI volumes

Standard file
API

— — — —

UBIFS

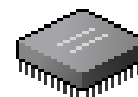
— — — —

UBI

— — — —

MTD driver

— — — —



Flash chip



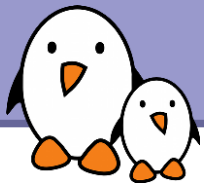
UBIFS - How to use

Creating

- ▶ `ubimkvol /dev/ubi0 -N test -s 116MiB`
or `ubimkvol /dev/ubi0 -N test -m (max available size)`
- ▶ `mount -t ubifs ubi0:test /mnt/flash`

Deleting

- ▶ `umount /mnt/flash`
- ▶ `ubirmvol /dev/ubi0 -N test`
- ▶ Detach the MTD partition:
`ubidetach /dev/ubi_ctrl -m 1`



Suitability for very small partitions

8M MTD partition

- ▶ jffs2 fits 13 MB of files
But probably doesn't leave enough free blocks
- ▶ UBI consumes 0.9 MB
ubifs fits 6.6 MB of files

4M MTD partition

- ▶ jffs2 fits 5.1 MB of files
- ▶ UBI consumes 0.8 MB
ubifs fits only 1.6 MB of files!

Bigger sizes: UBI overhead can be neglected:

32 MB: consumes 1.2 MB

128 MB: consumes 3.6 MB



LogFS

<http://logfs.org/logfs/>

- ▶ Also developed as a replacement for jffs2
- ▶ Expect very fast mount times!
- ▶ Didn't seem to be ready when we ran our benchmarks.
- ▶ Now ready for experimental tests with recent kernels:
<http://logfs.org/git?p=logfs;a=summary>



- ▶ Advanced XIP FileSystem for Linux
<http://axfs.sourceforge.net/>
- ▶ Allows to execute code directly from flash, instead of copying it to memory.
- ▶ As XIP is not possible with NAND flash, works best when there is a mix of NOR flash (for code) and NAND (for non XIP sections).
- ▶ Currently posted for review / inclusion in the mainstream Linux kernel.



SquashFS

<http://squashfs.sourceforge.net/>

- ▶ Filesystem for block storage!
Doesn't support the MTD API.
- ▶ But read-only! No problem with managing erase blocks and wear-leveling. So, it's fine to use it with the `mtdblock` driver.
- ▶ You can use it for the read-only sections in your filesystem.



SquashFS - How to use

Very simple!

- ▶ On your workstation, create your filesystem image:
`mksquashfs rootdir rootdir.sqfs`
- ▶ **Caution:** if the image already exists remove it first, or use the `-noappend` option.
- ▶ Erase your flash partition:
`flash_eraseall /dev/mtd2`
- ▶ Make your filesystem image available to your device (NFS, copy, etc.) and flash your partition:
`dd if=rootdir.sqfs of=/dev/mtdblock2`
- ▶ Mount your filesystem:
`mount -t squashfs /dev/mtdblock2 /mnt/flash`



Our benchmarks

jffs2

- ▶ Dramatically outperformed by ubifs in most aspects.
- ▶ Huge mount / boot time unless CONFIG_SUMMARY is used.

yaffs2

- ▶ Also outperformed by ubifs.
- ▶ May not fit all your data
- ▶ Ugly file removal time (poor directory update performance?)
- ▶ Memory usage not scaling
- ▶ ubifs leaves no reason to stick to yaffs2.

ubifs

- ▶ Great performance in all corner cases.

SquashFS

- ▶ Best or near best performance in all read-only scenarios.

Full benchmark details on

<http://free-electrons.com/pub/conferences/2008/elce/flash-filesystems.pdf>



Conclusions

- ▶ Convert your jffs2 partitions to ubifs!
- ▶ It may only make sense to keep jffs2 for MTD partitions smaller than 10 MB, in case size is critical.
- ▶ No reason left to use yaffs2 instead of jffs2?
- ▶ You may also use SquashFS to squeeze more stuff on your flash storage. Advisable to use it on top of UBI, to let all flash sectors participate to wear leveling.

SquashFS

— — — —

MTD block

— — — —

MTD API

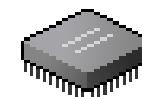
— — — —

UBI

— — — —

MTD driver

— — — —

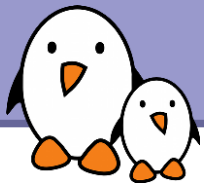


Flash chip



Issues with flash-based block storage

- ▶ Flash storage made available only through a block interface.
- ▶ Hence, no way to access a low level flash interface and use the Linux filesystems doing wear leveling.
- ▶ No details about the layer (Flash Translation Layer) they use. Details are kept as trade secrets, and may hide poor implementations.
- ▶ Hence, it is highly recommended to limit the number of writes to these devices.



Reducing the number of writes

- ▶ Mount your filesystems as read-only, or use read-only filesystems (SquashFS), whenever possible.
- ▶ Keep volatile files in RAM (tmpfs)
- ▶ Use the `noatime` mount option, to avoid updating the filesystem every time you access a file. Or at least, if you need to know whether files were read after their last change, use the `relatime` option.
- ▶ Don't use the `sync` mount option (commits writes immediately). No optimizations possible.
- ▶ You may decide to do without journaled filesystems. They cause more writes, but are also much more power down resistant.

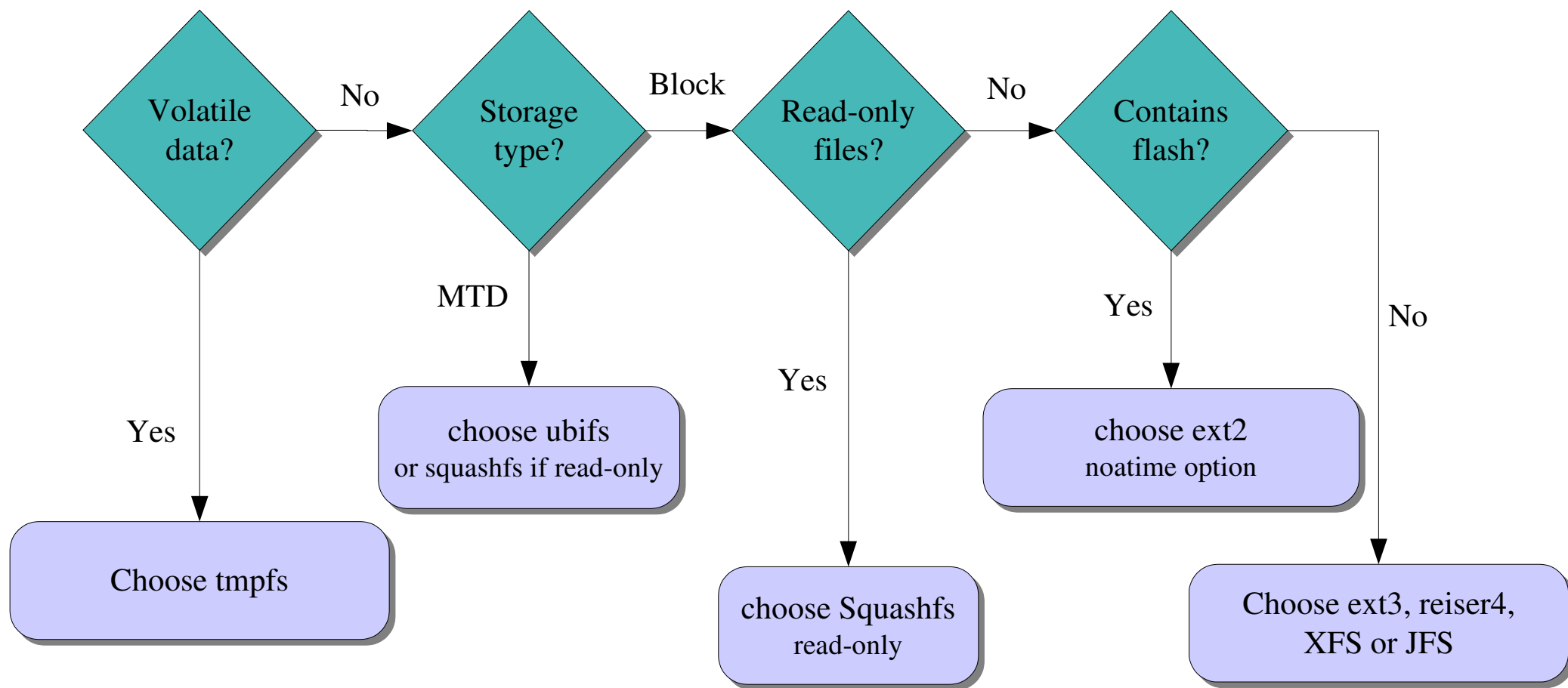


Useful reading

- ▶ Introduction to JFFS2 and LogFS:
<http://lwn.net/Articles/234441/>
- ▶ Nice UBI presentation from Toshiba:
<http://free-electrons.com/redirect/celf-ubi.html>
- ▶ Documentation on the linux-mtd website:
<http://www.linux-mtd.infradead.org/>



Filesystem choice summary



See [Documentation/filesystems/](#) in kernel sources for details about all available filesystems.



Practical lab – Flash filesystems

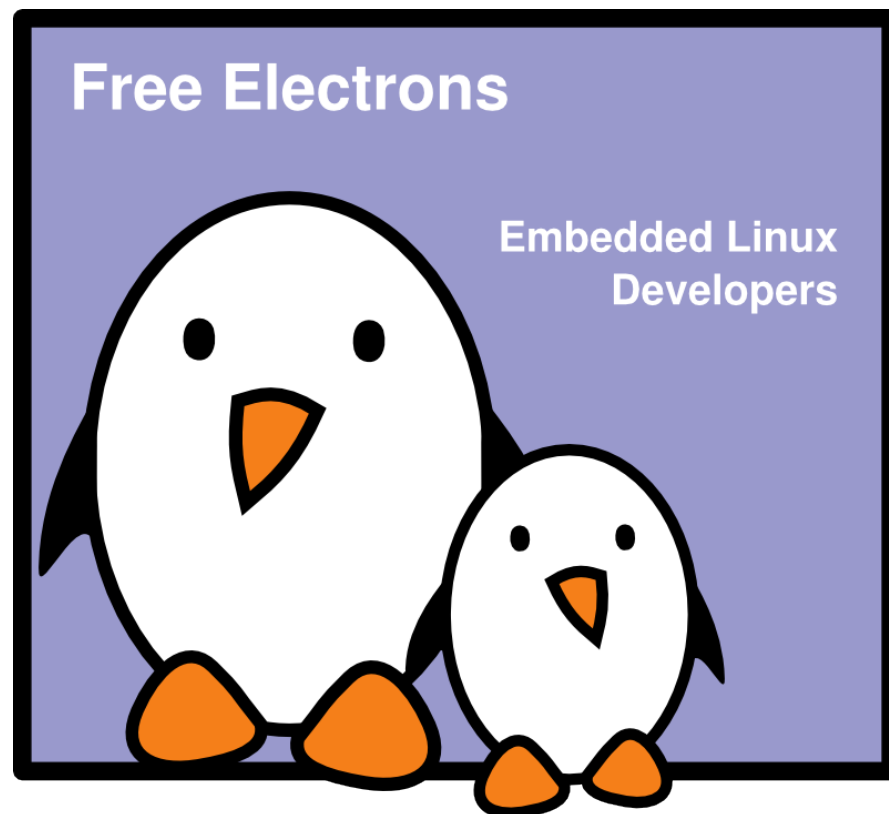
- ▶ Creating partitions in your internal flash storage.
- ▶ Formatting the main partition with SquashFS on mtdblock.
- ▶ Using jffs2 for system data.





Embedded Linux system development

Thomas Petazzoni
Michael Opdenacker
Free Electrons





Training contents (1)

GNU / Linux workstation

- ▶ System building
- ▶ Emulators
- ▶ Various tools
- ▶ Commercial toolsets and distributions



Training contents (2)

Target device

- ▶ http and ssh servers
- ▶ Graphical toolkits
- ▶ Web browsers
- ▶ Text editors



GNU / Linux workstation System building



Third party libraries and applications

- ▶ One of the advantages of embedded Linux is the wide range of third-party libraries and applications that one can leverage in its product
- ▶ However, efficiently re-using these components is not always easy. One must :
 - ▶ Find these components
 - ▶ Choose the most appropriate ones
 - ▶ Cross-compile them
 - ▶ Integrate them in the embedded system and with the other applications



Finding free software components

- ▶ Available everywhere on the Internet. These websites offer a large choice, probably too large.
 - ▶ SourceForge
<http://sourceforge.net>
 - ▶ Freshmeat
<http://freshmeat.net>
 - ▶ Free Software Directory
<http://directory.fsf.org>
 - ▶ Google
<http://www.google.com>
- ▶ More specific to embedded systems : the list of packages available in embedded build systems.
 - ▶ Buildroot
<http://sources.busybox.net/index.py/trunk/buildroot/package/>
 - ▶ OpenEmbedded
<http://cgit.openembedded.net/cgit.cgi?url=openembedded/tree/recipes>



Choosing components

- ▶ Not all free software components are necessarily good to re-use. One must pay attention to :
 - ▶ **Vitality** of the developer and user communities. This vitality ensures long-term maintenance of the component, and relatively good support. It can be measured by looking at the mailing-list traffic and the version control system activity.
 - ▶ **Quality** of the component. Typically, if a component is already available through embedded build systems, and has a dynamic user community, it probably means that the quality is relatively good.
 - ▶ **License**. The license of the component must match your licensing constraints. For example, GPL libraries cannot be used in proprietary applications.
 - ▶ **Technical requirements**. Of course, the component must match your technical requirements. But don't forget that you can improve the existing components if a feature is missing !



System building

Several solutions

- ▶ Manually
- ▶ System building tools
- ▶ Distributions or ready-made filesystems



System building: manually

- ▶ Manually building a target system involves downloading, configuring, compiling and installing all the components of the system.
- ▶ All the libraries and dependencies must be configured, compiled and installed in the right order.
- ▶ Sometimes, the build system used by libraries or applications is not very cross-compile friendly, so some adaptations are necessary.
- ▶ There is no infrastructure to reproduce the build from scratch, which might cause problems if one component needs to be changed, if somebody else takes over the project, etc.



Cross-compiling

- ▶ Once the component is chosen, it must be compiled for the target embedded system. This cross-compilation process can sometimes be tedious.
 - ▶ The compiling tools are different (arm-linux-gcc instead of gcc)
 - ▶ The files are not installed in /usr, /usr/lib, but inside a different directory. This is true for binaries, libraries, pkgconfig configuration files, includes, etc.
- ▶ Some knowledge on how the major build systems handle cross-compilation is useful to fix build issues when they occur.
 - ▶ The autoconf / automake / pkgconfig / libtool build system, which is the most important one today.
 - ▶ CMake, a new generation build system
 - ▶ Waf and Scons, Python-based build systems

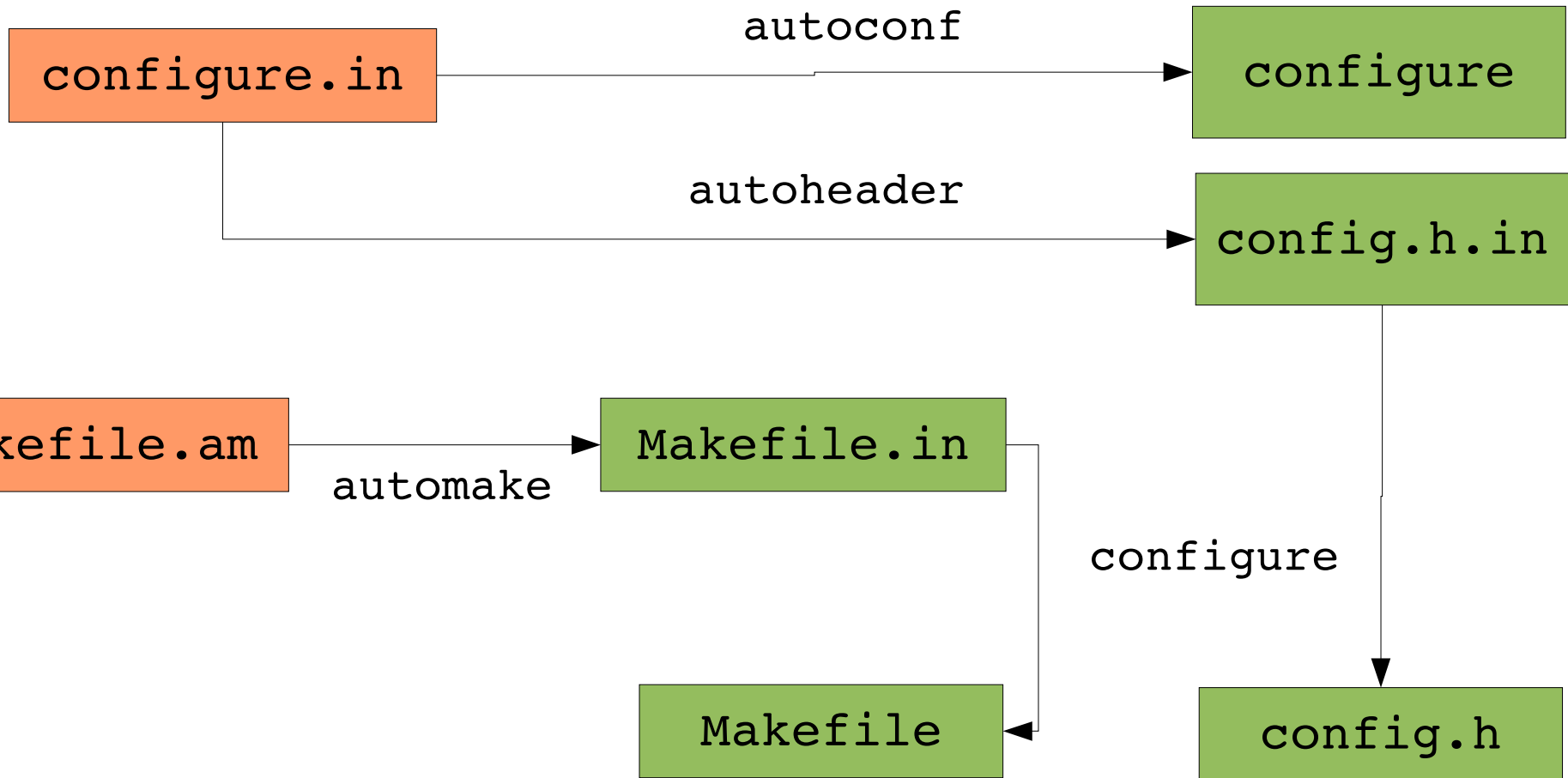



Autotools and friends


- ▶ Their usage
 - ▶ **autoconf** is used to handle the configuration of the software package
 - ▶ **automake** is used to generate the Makefiles needed to build the software package
 - ▶ **pkgconfig** is used to ease compilation against already installed shared libraries
 - ▶ **libtool** is used to handle the generation of shared libraries in a system-independent way
- ▶ Most of these tools are old and relatively complicated to use, but they are used by a majority of free software packages today. One must have a basic understanding of what they do and how they work.



automake / autoconf / autoheader



 Written by the developer

 Automatically generated



- ▶ Files written by the developer
 - ▶ `configure.in` describes the configuration options and the checks done at configure time
 - ▶ `Makefile.am` describes how the software should be built
- ▶ The `configure` script and the `Makefile.in` files are generated by `autoconf` and `automake` respectively.
 - ▶ They should never be modified directly
 - ▶ They are usually shipped pre-generated in the software package, because there are several versions of `autoconf` and `automake`, and they are not completely compatible
- ▶ The `Makefile` files are generated at configure time, before compiling
 - ▶ They are never shipped in the software package.



Configuring and compiling (1)

- ▶ The traditional steps to configure and compile an autotools based package are :
`./configure`
`make`
`make install`
- ▶ These steps work well for native compilation. For cross-compiling, things are a little bit more complex.
 - ▶ At least some of the environment variables `AR`, `AS`, `LD`, `NM`, `CC`, `GCC`, `CPP`, `CXX`, `STRIP`, `OBJCOPY` must be defined to point to the proper cross-compiling tools
 - ▶ The `--build`, `--host` and `--target` arguments must be passed to the `configure` script.
 - ▶ It is recommended to pass the `--prefix` argument. It defines from which location the software will run in the target environment. Usually, `/usr` is fine.



Configuring and compiling (2)

- ▶ If one simply runs `make install`, the software will be installed in the directory passed as `--prefix`. For cross-compiling, one must pass the `DESTDIR` argument to specify where the software must be installed.
- ▶ Making the distinction between the prefix (as passed with `--prefix` at configure time) and the destination directory (as passed with `DESTDIR` at installation time) is very important.

▶ Example

```
export PATH=/usr/local/arm-linux/bin:$PATH
export CC=arm-linux-gcc
export STRIP=arm-linux-strip
./configure --host=arm-linux --target=arm-linux
              --build=i386-pc-linux-gnu
make
make DESTDIR=/home/<user>/work/rootfs install
```



Installation

- ▶ The autotools based software packages provide both a `install` and `install-strip` make targets, used to install the software, either stripped or unstripped.
- ▶ For applications, the software is usually installed in `<prefix>/bin`, with configuration files in `<prefix>/etc` and data in `<prefix>/share/<application>/`.
- ▶ The case of libraries is a little more complicated :
 - ▶ In `<prefix>/lib`, the library itself (a `.so.<version>`), a few symbolic links, and the libtool description file (a `.la` file)
 - ▶ The `pkgconfig` description file in `<prefix>/lib/pkgconfig`
 - ▶ Include files in `<prefix>/include/`
 - ▶ Sometimes a `<libname>-config` program in `<prefix>/bin`
 - ▶ Documentation in `<prefix>/share/man` or `<prefix>/share/doc/`



Installation (2)

Contents of /usr after installation of zlib and libpng

./lib		Libtool description file
./lib/libpng12.la	←	
./lib/libpng.la	-> libpng12.la	Static version of the library
./lib/libpng12.a	←	
./lib/libpng.a	-> libpng12.a	
./lib/libpng.so.3.32.0	←	Dynamic version of the library
./lib/libpng12.so.0.32.0		
./lib/libpng12.so.0	-> libpng12.so.0.32.0	
./lib/libpng12.so	-> libpng12.so.0.32.0	
./lib/libpng.so	-> libpng12.so	
./lib/libpng.so.3	-> libpng.so.3.32.0	
./lib/pkgconfig		
./lib/pkgconfig/libpng.pc	-> libpng12.pc	Pkgconfig description file
./lib/pkgconfig/libpng12.pc	←	
./lib/libz.so.1.2.3	←	
./lib/libz.so	-> libz.so.1.2.3	Zlib dynamic library
./lib/libz.so.1	-> libz.so.1.2.3	



Installation (3)

Contents of `/usr` after installing `zlib` and `libpng`

- `./share`
- `./share/man`
- `./share/man/man5`
- `./share/man/man5/png.5`
- `./share/man/man3`
- `./share/man/man3/libpngpf.3`
- `./share/man/man3/libpng.3`
- `./share/man/man3/zlib.3`

Documentation

- `./bin`
- `./bin/libpng-config`
- `./bin/libpng12-config`

Script to get `libpng`
installation configuration

- `./include`
- `./include/zconf.h`
- `./include/pngconf.h`
- `./include/png.h`
- `./include/libpng12`
- `./include/libpng12/pngconf.h`
- `./include/libpng12/png.h`
- `./include/zlib.h`

Header files to compile applications
relying on `zlib` and `libpng`



Build and target spaces

- ▶ From all these files, everything except documentation is necessary to build an application that relies on libpng.
- ▶ However, only the library binary in `<prefix>/lib` and some symbolic links are needed to execute the application on the target.
- ▶ Because of this, it is useful to distinguish two spaces
 - ▶ The build space, where all the libraries and applications will be installed using the normal `make install` command.
 - ▶ The target space, where only the strictly required files will be copied, in order to reduce the size and complexity of the target filesystem.
- ▶ The build space must be kept in order to build other applications or recompile existing applications.



Let's find the libraries

- ▶ When compiling an application or a library that relies on other libraries, the build process by default looks in `/usr/lib` for libraries and `/usr/include` for headers.
- ▶ The first thing to do is to set the `CFLAGS` and `LDFLAGS` environment variables :

```
export CFLAGS=-I/my/build/space/usr/include/  
export LDFLAGS=-L/my/build/space/usr/lib
```
- ▶ The `libtool` files (`.la` files) must be modified because they include the absolute paths of the libraries :
 - `libdir='/usr/lib'`
 - + `libdir='/my/build/space/usr/lib'`
- ▶ The `PKG_CONFIG_PATH` environment variable must be set to the location of the `.pc` files and the `PKG_CONFIG_SYSROOT_DIR` variable must be set to the build space directory.



pkg-config

- ▶ `pkg-config` is a tool that allows to query a small database to get information on how to compile programs that depend on libraries
- ▶ The database is made of `.pc` files, installed by default in `<prefix>/lib/pkgconfig/`.
- ▶ `pkg-config` is used by the configure scripts to get the library configurations
- ▶ It can also be used manually to compile an application :

```
arm-linux-gcc -o test test.c $(pkg-config --libs --cflags)
```
- ▶ By default, `pkg-config` looks in `/usr/lib/pkgconfig` for the `*.pc` files, and assumes that the paths in these files are correct.
- ▶ `PKG_CONFIG_PATH` allows to set another location for the `*.pc` files and `PKG_CONFIG_SYSROOT_DIR` to prepend a prefix to the paths mentioned in the `.pc` files.



Autotools references

- ▶ Autoconf
<http://www.gnu.org/software/autoconf/manual/>
- ▶ Automake
<http://www.gnu.org/software/automake/manual/>
- ▶ Simple and good tutorial
<http://www.seul.org/docs/autotut/>



Other build systems

- ▶ Cmake, Cross Platform Make
 - ▶ <http://www.cmake.org/>
 - ▶ Used by large projects such as KDE or Second Life
 - ▶ Much newer and simpler than the Autotools
 - ▶ Supports cross-compilation,
http://www.cmake.org/Wiki/CMake_Cross_Compiling
- ▶ Waf: <http://code.google.com/p/waf/>
- ▶ Scons: <http://www.scons.org/>
- ▶ Hand-made
 - ▶ Only your skills and experience will tell you how to work with these!
- ▶ See our [annex about Scons and Cmake](#)



Practical lab – Manual cross-compiling

- ▶ Manually cross-compiling applications and libraries
- ▶ Learning about common techniques and issues.





System building: manually with Scratchbox

- ▶ To solve the cross-compiling issues while manually building a target system, a possible solution is to use **Scratchbox**.
 - ▶ <http://www.scratchbox.org>
- ▶ Used by Nokia to develop its Internet tablets (770, N800, N810)
- ▶ Works by allowing tools to be cross-compiled in a transparent way, making building tools believe they are doing a native compile job.
- ▶ Supported platforms: [arm](#), [x86](#)
Uses the [qemu](#) emulator to transparently run built [arm](#) binaries.
Experimental support for [ppc](#), [mips](#) and [cris](#).



System building: manually with Scratchbox

- ▶ Scratchbox benefits
 - ▶ Chrooted environment: you are still on the host, but you only see the target files.
 - ▶ Transparent cross-compiling: the cross-compiler looks like a native one.
 - ▶ Transparent execution: either through remote execution on the target. Or through CPU code emulation ([qemu](#))
- ▶ Drawbacks
 - ▶ No infrastructure for build reproduction
 - ▶ Limited number of host tools available
 - ▶ Requires modified toolchains. Only old ones released, causing problems to compile recent stuff, like mtd-utils with UBI support.



System building : tools

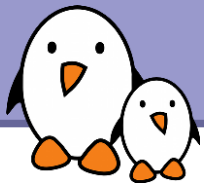
- ▶ Different tools are available to automate the process of building a target system, including the kernel, and sometimes the toolchain.
- ▶ They automatically download, configure, compile and install all the components in the right order, sometimes after applying patches to fix cross-compiling issues.
- ▶ They already contain a large number of packages, that should fit your main requirements, and are easily extensible.
- ▶ The build becomes reproducible, which allows to easily change the configuration of some components, upgrade them, fix bugs, etc.



System building: tools

Large choice of tools

- ▶ **Buildroot**, developed by the community
<http://www.buildroot.net>
- ▶ **PTXdist**, developed by Pengutronix
http://www.pengutronix.de/software/ptxdist/index_en.html
- ▶ **LTIB**, developed mainly by Freescale. Good support for Freescale boards
<http://www.bitshrine.org/>
- ▶ **OpenEmbedded**, more flexible but also far more complicated
<http://www.openembedded.org>
- ▶ **Gentoo Embedded**
<http://www.gentoo.org/proj/en/base/embedded/handbook/>



Buildroot

- ▶ Simple `menuconfig` interface to create the configuration
 - ▶ Target architecture
 - ▶ Toolchain
 - ▶ Packages
- ▶ `make menuconfig`
- ▶ `make`

```
.config - buildroot v0.10.0-svn Configuration

Buildroot Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selects a feature, while
<N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] feature is selected [ ] feature is excluded

Target Architecture (arm) --->
Target Architecture Variant (generic_arm) --->
Target ABI (EABI) --->
Target options --->
Build options --->
Toolchain --->
[*] Package Selection for the target --->
Target filesystem options --->
Kernel --->
---
Load an Alternate Configuration File
Save an Alternate Configuration File

<Select> < Exit > < Help >
```

See our full Buildroot presentation for details:
<http://free-electrons.com/docs/buildroot/>



Buildroot: adding new packages

- ▶ Create a new directory, for example `package/gqview/`
- ▶ Add a `Config.in` file in this directory to describe the configuration options

```
config BR2_PACKAGE_GQVIEW
    bool "gqview"
    select BR2_PACKAGE_PKGCONFIG
    help
        GQview is an image viewer for Unix
        operating systems

        http://prdownloads.sourceforge.net/gqview
```



Buildroot : adding new packages (2)

- ▶ Insert the new `Config.in` file in the configuration system by adding to `package/Config.in`

```
source "package/gqview/Config.in"
```

- ▶ Create the `gqview.mk` file to describe the build steps

```
GQVIEW_VERSION = 2.1.5
GQVIEW_SOURCE = gqview-$(GQVIEW_VERSION).tar.gz
GQVIEW_SITE = http://prdownloads.sourceforge.net/gqview
GQVIEW_AUTORECONF = NO
GQVIEW_INSTALL_STAGING = NO
GQVIEW_INSTALL_TARGET = YES
GQVIEW_DEPENDENCIES = uclibc pkgconfig libgtk2
$(eval $(call AUTOTARGETS,package,gqview))
```



Embedded distributions (1)

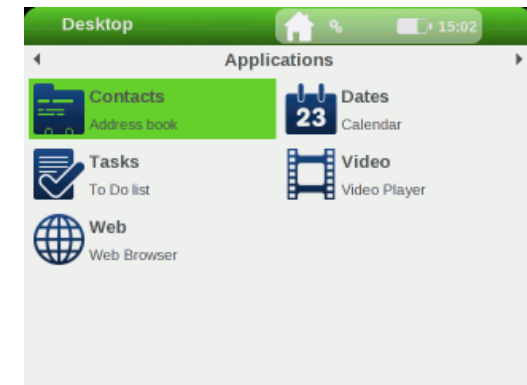
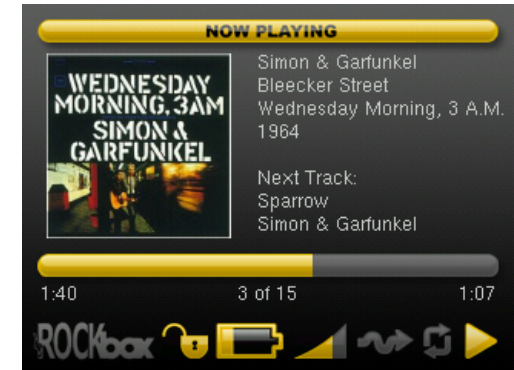
- ▶ Debian GNU/Linux is available for ARM, MIPS and PowerPC architectures
 - ▶ Offers a huge flexibility thanks to the package management system, but only works only on suitably-sized systems
 - ▶ <http://www.debian.org>
- ▶ Emdebian is a project to improve Debian for embedded systems
 - ▶ Supports all embedded platforms
 - ▶ Leverages standard **Debian** package descriptions and sources, but adapts them for embedded systems: **uClibc** usage, reduced package size, removed dependencies, smaller configuration..
 - ▶ <http://www.emdebian.org>





Embedded distributions (2)

- ▶ Distributions designed for specific types of devices are also available
 - ▶ Ångström: <http://www.angstrom-distribution.org/>
Targets PDAs and webpads (Siemens Simpad...) Binaries available for `arm` little endian.
 - ▶ Rockbox: <http://www.rockbox.org/>
A distribution for portable media players. Supports a wide range of players and open to new ones!
 - ▶ Poky: <http://www.pokylinux.org>
from OpenedHand. GNOME-based Linux distribution for mobile devices. Includes a Sato graphical application framework (GTK based).





Ready-made root filesystems

- ▶ Very useful to start porting work.
Contain C library, and minimum apps.
- ▶ However, there are not very flexible, and it is usually difficult to add new components or to upgrade the existing components.
 - ▶ uClibc : <http://www.uclibc.org/toolchains.html>
i386, arm(eb), mips(el), i386, ppc, sh4
Images where generated by Buildroot and include a native gcc toolchain
 - ▶ ARM: http://www.arm.com/linux/linux_download.html
Include kernel image, bootloader, and rootfs (cramfs or packages).
 - ▶ Other architectures :
If some readers know more useful links, they are welcome!



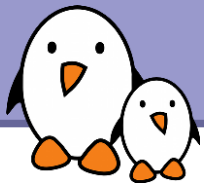
Practical lab – Buildroot

- ▶ Rebuild the same system, this time with Buildroot.
- ▶ See how easier it gets!
- ▶ Adding your own DirectFB based application.





GNU / Linux workstation Emulators



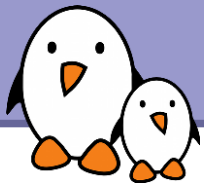
<http://fabrice.bellard.free.fr/qemu/>

Fast processor emulator
using a portable dynamic translator.

QEMU

Full system emulation

- ▶ Emulates the processor and various peripherals
Supported: `x86`, `x86_64`, `ppc`, `arm`, `sparc`, `mips`, `m68k`
- ▶ To know which machine types are supported:
`qemu-system-arm -M ?`
- ▶ `i386`, `x86_64` system emulation: now close to native speeds
thanks to the `kqemu` kernel module (now GPL v2!).



qemu user emulation

User emulation

(Linux host only)

- ▶ User mode emulation : can run applications compiled for another CPU.
Supported: `x86`, `ppc`, `arm`, `sparc`, `mips`, `m68k`
- ▶ Easy to run `BusyBox` for `arm` on `i386` GNU / Linux:

```
qemu-arm -L /usr/local/arm/3.3.2 \
/home/bart/arm/busybox-1.4.1/busybox ls
```
- ▶ `-L`: target C library binaries path (here cross-compiler toolchain path)
- ▶ On Ubuntu (since 8.04), for security reasons, you will need to put the following lines in `/etc/sysctl.conf` to make this work:

```
vm.vdso_enabled = 0
vm.mmap_min_addr = 4096
```

Reboot or just reload the new values:

```
sudo /sbin/sysctl -p /etc/sysctl.conf
```



Other emulators

▶ ARM platform

▶ **SkyEye**: <http://skyeye.sourceforge.net>
Emulates several **ARM** platforms (**AT91**, **Xscale**...) and can boot several operating systems (**Linux**, **uClinux**, and others)

▶ **Softgun**: <http://softgun.sourceforge.net>
Virtual **ARM** system with many virtual on-board peripherals.
Boots **Linux**.

▶ **SWARM** - Software **ARM** - **arm7** emulator
<http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html>
Can run **uClinux**

▶ **ColdFire** emulator
<http://www.slicer.ca/coldfire/>
Can boot **uClinux**



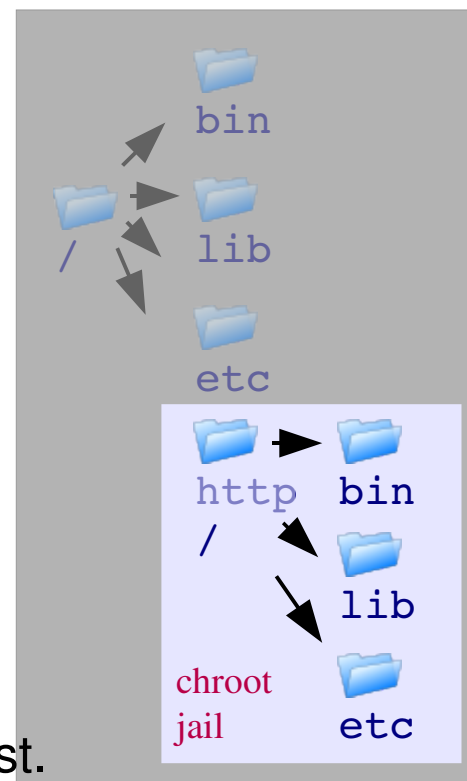
GNU / Linux workstation Various tools



chroot

Available in all GNU/Linux distributions

- ▶ Usage:
`chroot <dir> [command]`
- ▶ Runs a command or an interactive shell with a special root directory.
- ▶ Standard usage: Internet servers
Services executed in *chroot jails*: even when compromised, the service cannot gain access to the rest of the system.
- ▶ Usage for embedded system development:
Develop and test a new root filesystem on the development host.
Very easy to use when the host and target have the same CPU instruction set.
Used by LFS (Linux From Scratch).
Also used by [Scratchbox](#): [Qemu](#) makes the usage of target binaries transparent.





Minicom (1)

- ▶ Definition: serial communication program
- ▶ Available in all **GNU / Linux** distributions
- ▶ Capabilities (all through a serial link):
 - ▶ Serial console to a remote Unix system
 - ▶ File transfer
 - ▶ Modem control and dial-up
 - ▶ Serial port configuration



Minicom (2)

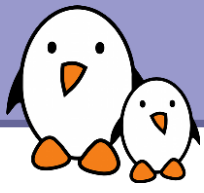
```
root@localhost:~  
File Edit View Terminal Tabs Help  
  
A - Serial Device      : /dev/ttyUSB0  
B - Lockfile Location  : /var/lock  
C - Callin Program     :  
D - Callout Program    :  
E - Bps/Par/Bits       : 115200 8N1  
F - Hardware Flow Control : No  
G - Software Flow Control : No  
  
Change which setting?  
  
Screen and keyboard  
Save setup as dfl  
Save setup as..  
Exit  
Exit from Minicom
```

- ▶ Start by running `minicom -s` to setup Minicom
- ▶ A bit austere at first glance, but quickly gets friendly (see the labs for details)



Other terminal emulators

- ▶ **GTKTerm:** <http://www.jls-info.com/julien/linux/>
Graphical. Less powerful than **Minicom**, but with a simpler and more attractive interface. Available in recent distros.
- ▶ **CuteCom:** <http://cutecom.sourceforge.net/>
Another graphical and user-friendly terminal emulator. Available in recent distros.
- ▶ **GNU Screen:** can also be used on a serial console:
`screen <device> <baudrate>`
Example:
`screen /dev/ttyS0 115200`



GNU / Linux workstation
Commercial toolsets



Commercial toolsets

Caution: *commercial* doesn't mean *proprietary*!

- ▶ Vendors play fair with the GPL and do make their source code available to their users, and most of the time, to the community.
 - ▶ As long as they distribute the sources to their users, the GPL doesn't require vendors to share their sources with any third party.
- ▶ No issue with all the GPL sources developed by or with the community.
- ▶ Graphical toolkits developed by the vendors look proprietary. Their licenses are not advertised on their websites! You have to be a customer to know or get a free preview kit to know.



Commercial toolset strengths

- ▶ Technical advantages
 - ▶ Well tested and supported kernel and tool versions
 - ▶ Including early patches not supported by the mainstream kernel yet
- ▶ Complete development tool sets: kernels, toolchains, utilities, binaries for impressive lists of target platforms
- ▶ Integrated utilities for automatic kernel image, initrd and filesystem generation.
- ▶ Graphical developments tools
- ▶ Development tools available on multiple platforms: [GNU / Linux](#), [Solaris](#), [Windows](#)...
- ▶ Support services
 - ▶ Useful if you don't have your own support resources
 - ▶ Long term support commitment, even for versions considered as obsolete by the community, but not by your users!



Montavista

<http://www.mvista.com/>



The market leader

- Employs some of the most active kernel hackers, in particular on the [arm](#) platform.
- Kernel development eventually shared with the community. kernel. Many drivers merged in mainstream Linux.
- Graphical development tools are proprietary.



<http://timesys.com>



- Similar toolset offering as other vendors. Great flexibility available to their LinuxLink™ subscribers
- Community friendly: they share very interesting and generic technical whitepapers and articles. They also employ key community hackers (Thomas Gleixner, Rob Landley...).
- Free Software BSPs (Board Support Packages) available.
- **Linux** soft and hard real-time OS product.
- Development tools seem to be proprietary.



Wind River

Wind River Linux:

<http://www.windriver.com/products/linux/>

WIND RIVER

- ▶ A lot of embedded and real-time experience from **VxWorks**.
- ▶ Now say they integrate, test and support Linux as rigorously as they do with **VxWorks**.
Linux development supported with their Workbench integrated development environment, already used for **VxWorks**.
- ▶ Support standard and recent Linux kernel sources, including real-time preempt patches (Linux 2.6.27). Also offer hard real-time Linux (**Real Time Core**: formerly **RTLinux**).



Sysgo - Koan Software

<http://sysgo.com>



- ▶ ELinOS development toolset, in particular based on Eclipse and the Linux Trace Toolkit.
- ▶ Includes FreeToolBox, a freely downloadable compiling and rootfs creating toolchain.
- ▶ Supports i386, arm and ppc.
- ▶ Hard real-time support with their own microkernel (PikeOS), an approach similar to RTAI.

<http://koansoftware.com>



- Makers of KaeilOS (http://koansoftware.com/kaeilos/index_en.htm), a GPL embedded Linux distribution for industrial applications.
- KaeilOS supports i386 and popular arm platforms. Other platforms supported upon request.
- Includes several graphical toolkits and supports hard real-time (RTAI, Xenomai, preemption patches).
- Unfortunately, KaeilOS is GPL but not available for public download.



Denx Software Engineering



<http://denx.de>

- ▶ Created by Wolfgang Denk, the author of the U-Boot bootloader.
- ▶ Create and support the Embedded Linux Development Kit (ELDK), a complete and well documented development environment.
- ▶ This kit is not only Free Software, it can be downloaded freely by anyone.
- ▶ A great community member and contributor!



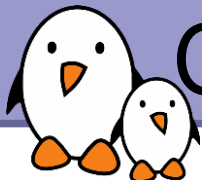
BlueCat Linux

LynuxWorks BlueCat

<http://lynuxworks.com/products/bluecat/>



- ▶ From a traditional RTOS vendor trying to survive the Linux tidal wave.
- ▶ Only advertise features of very early Linux 2.6 kernels!
- ▶ No visible contribution to the development community.
- ▶ Mainly seems to be a way of surfing the Linux wave and attracting customers to their **LynxOS** RTOS (API compatible with Linux).



Contributions from embedded Linux vendors

Counted the number of lines in [Linux 2.6.23](#)
with both [Copyright](#) and the vendor name (case insensitive search)

<i>Embedded Vendor</i>	<i>Number of lines</i>
MontaVista	444
Denx	22
TimeSys	21
Sysgo	8
Windriver	61
LynuxWorks	2
Koan Software	0
Free Electrons 🙄	0

Disclaimer: this is just a very inaccurate indicator!



Commercial toolsets - Summary

- ▶ Major vendors: [MontaVista](#), [Wind River](#), [TimeSys](#)
Involved in Linux development.
- ▶ Smaller vendors: [Koan](#), [Sysgo](#), [Denx...](#)
Trying to differentiate their products.
- ▶ Community based companies: [Denx](#), [CodeSourcery](#)
Contribute to community tools.
Mainly offer support and development.
- ▶ Traditional RTOS vendors: [LynuxWorks](#)
Forced to support Linux to retain market share.
Strong real-time expertise but lack of community involvement.



Commercial or community solutions?

Commercial distributions and toolsets

- ▶ Best if you don't have your own support resources and have a sufficient budget
- ▶ Really help focusing on your real job: making an embedded device.
- ▶ You can even subcontract driver development to the vendor

Community distributions and tools

- ▶ Best if you are on a tight budget
- ▶ Best if you are willing to build your own embedded Linux expertise and train your own support resources.

In any case, your products are based on Free Software!



Tools for the target device
http and ssh servers



ssh server and client: dropbear

<http://matt.ucc.asn.au/dropbear/dropbear.html>

- ▶ Very small memory footprint ssh server for embedded systems
- ▶ Satisfies most needs. Both client and server!
- ▶ Size: **110 KB**, statically compiled with **uClibc** on **i386**.
(**OpenSSH** client and server: approx **1200 KB**,
dynamically compiled with **glibc** on **i386**)
- ▶ Useful to:
 - ▶ Get a remote console on the target device
 - ▶ Copy files to and from the target device (**scp** or **rsync -e ssh**).



Benefits of a web server interface

Many network enabled devices can just have a network interface

- ▶ Examples: modems / routers, IP cameras, printers...
- ▶ No need to develop drivers and applications for computers connected to the device. No need to support multiple operating systems!
- ▶ Just need to develop static or dynamic HTML pages (possibly with powerful client-side JavaScript).
Easy way of providing access to device information and parameters.
- ▶ Reduced hardware costs (no LCD, very little storage space needed)



Tiny/Turbo/Throttling HTTP
server

<http://acme.com/software/thttpd/>

- ▶ Simple
Implements the HTTP/1.1 minimum (or just a little more)
Simple to configure and run.
- ▶ Small
Executable size: 88K (version 2.25b), Apache 2.0.52: 264K
Very low memory consumption:
does not fork and very careful about memory consumption.

- ▶ Portable
Compiles cleanly on most Unix-like operating systems
- ▶ Fast
About as fast as full-featured servers. Much faster on very high loads (because reduces the server load for the same amount of work)
- ▶ Secure
Designed to protect the webserver machine from attacks.



Other web servers (1)



- ▶ **BusyBox** http server: <http://busybox.net>

Tiny: only adds 9 K to **BusyBox** 1.5 (dynamically linked with **glibc** on **i386**, with all features enabled.)! Sufficient features for many devices with a web interface, including CGI, http authentication and script support (like PHP).

License: GPL

- ▶ **KLone**: <http://koanlogic.com/kl/cont/gb/html/klone.html>

Lightweight but full featured web server for embedded systems.

Can enclose dynamic (written in C/C++ `<% code %>`) and compressed content all in an executable of an approximate size of 150 KB.

License: Dual GPL / Commercial.

See also <http://linuxdevices.com/news/NS8234701895.html>



Other web servers (2)

- ▶ **Boa:** <http://www.boa.org/>

Designed to be simple, fast and secure.

Unlike [thttpd](#), no particular care for memory or disk footprint though.

Embedded systems: pretty popular, though not targeted by developers.



- ▶ **lighttpd:** <http://lighttpd.net>

Low footprint server good at managing high loads.

May be useful in embedded systems too.





Tools for the target device Graphical toolkits



Graphical toolkits

« Low-level » solutions



DirectFB (1)



- ▶ Low-level graphical library
 - ▶ Lines, rectangles, triangles drawing and filling
 - ▶ Blitting, flipping
 - ▶ Text drawing
 - ▶ Windows and transparency
 - ▶ Image loading and video display
- ▶ But also handles input event handling: mouse, keyboard, joystick, touchscreen, etc.
- ▶ Provides accelerated graphic operations on various hardware, more can be added in an easy way
- ▶ Integrated windowing infrastructure

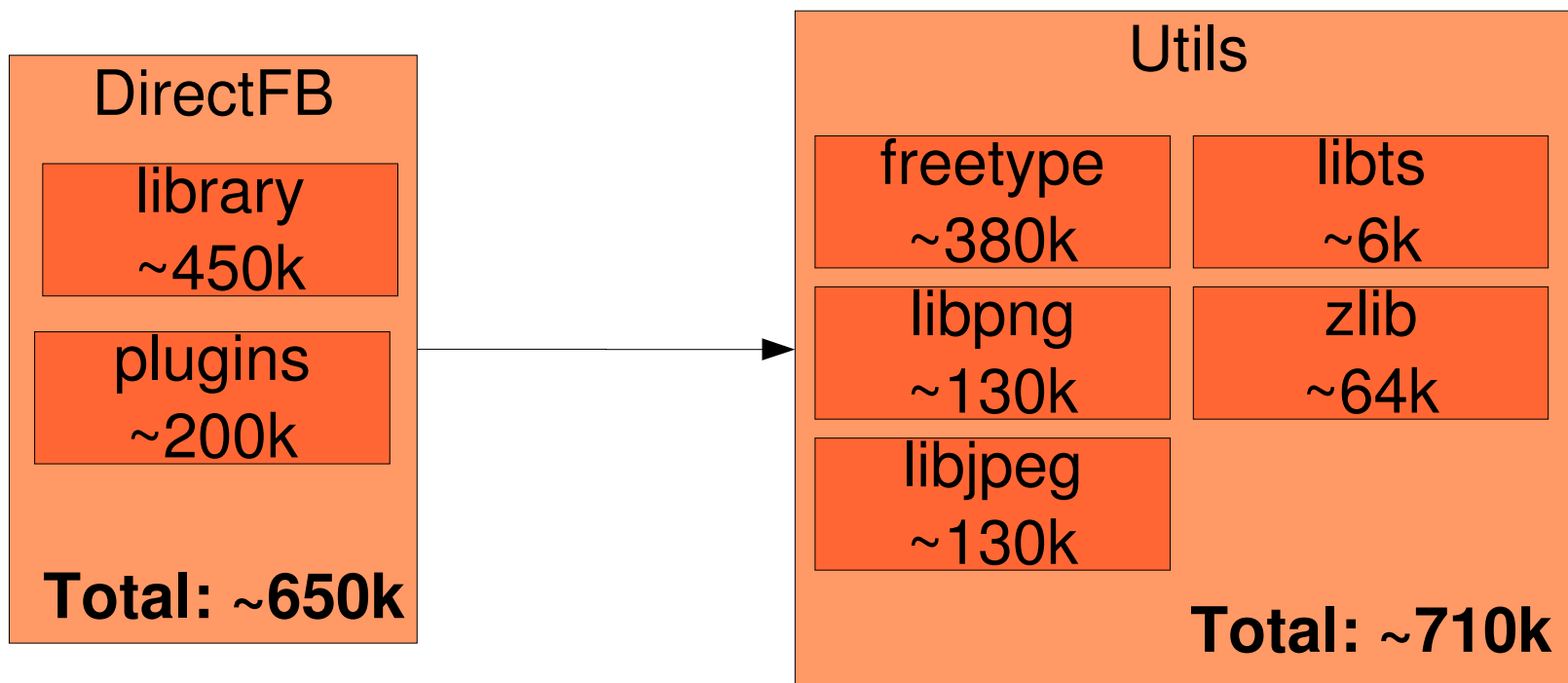


DirectFB (2)

- ▶ Single-application by default, but multiple applications can share the framebuffer thanks to « fusion »
- ▶ Development and community: very active
- ▶ License: LGPL 2.1
- ▶ <http://www.directfb.org>



DirectFB: size and dependencies

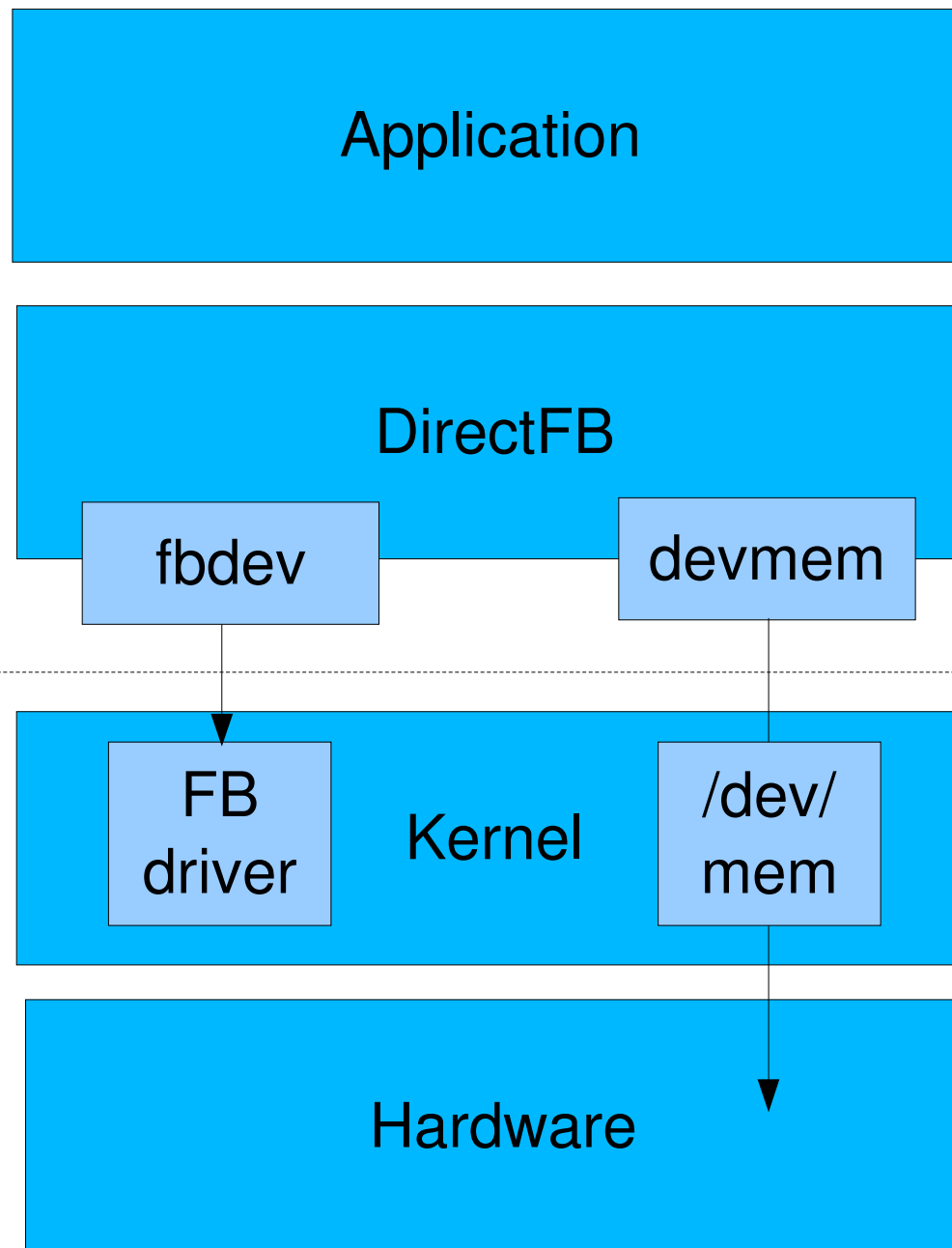


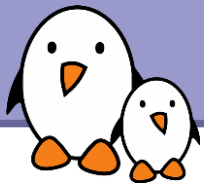
Total: ~1.4m

Some of these dependencies are optional.
This is a typical setup.



DirectFB: architecture





DirectFB: usage (1)

- ▶ Multimedia applications
 - ▶ For example the Disko framework, for set-top box related applications
- ▶ « Simple » graphical applications
 - ▶ Industrial control
 - ▶ Device control with limited number of widgets
- ▶ Visualization applications
- ▶ As a lower layer for higher-level graphical libraries



DirectFB: usage (2)





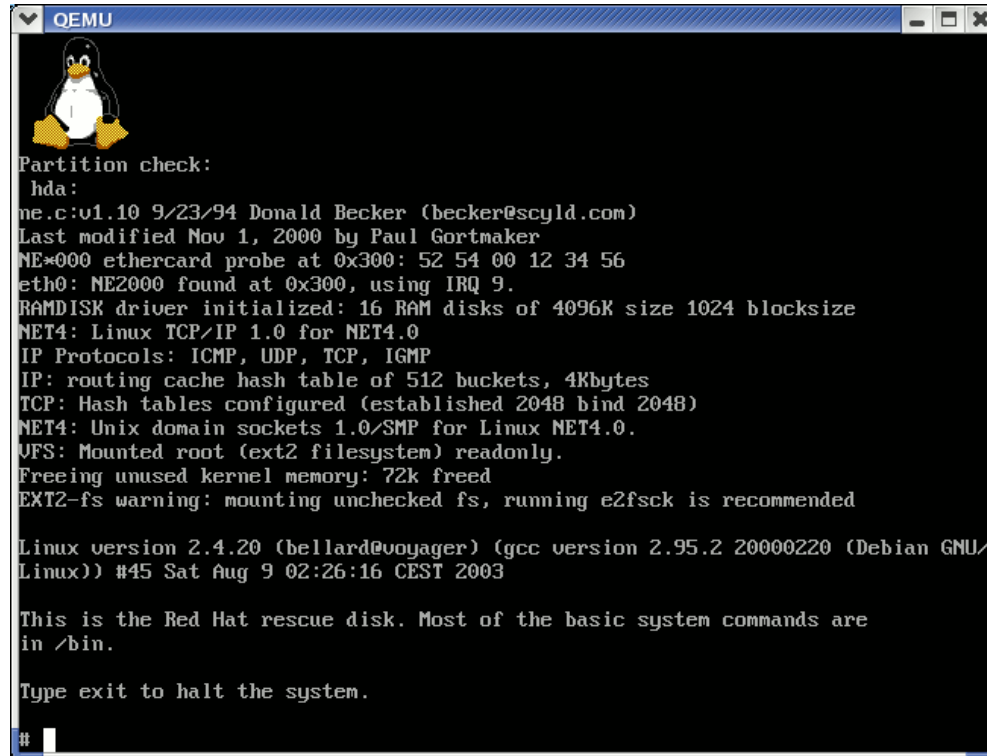
SDL

- ▶ A library originally designed for game development
- ▶ In addition to graphic display, also provides input event management, sound, CD-ROM audio, threads, timers, etc.
- ▶ Can work on top of X11, the framebuffer or DirectFB
 - ▶ And DirectFB can work on top of SDL as well :-)
- ▶ The API is roughly the same level as the one of DirectFB
- ▶ Developed in C, C API, many bindings available
- ▶ Actively maintained
- ▶ DirectFB is probably more common in embedded systems, while SDL is more common for small desktop games
- ▶ License: LGPL
- ▶ <http://www.libsdl.org/>





SDL screenshots



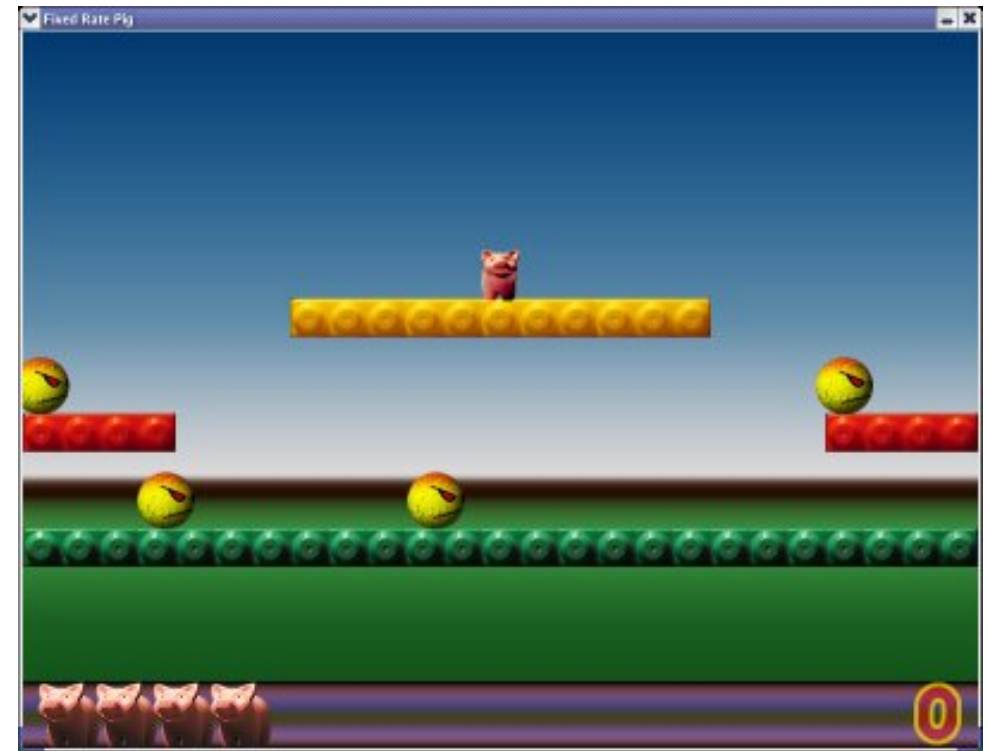
```
QEMU
Partition check:
hda:
ne.c:v1.10 9/23/94 Donald Becker (becker@scyld.com)
Last modified Nov 1, 2000 by Paul Gortmaker
NE*000 ethcard probe at 0x300: 52 54 00 12 34 56
eth0: NE2000 found at 0x300, using IRQ 9.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 2048)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 72k freed
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended

Linux version 2.4.20 (bellard@voyager) (gcc version 2.95.2 20000220 (Debian GNU/
Linux)) #45 Sat Aug 9 02:26:16 CEST 2003

This is the Red Hat rescue disk. Most of the basic system commands are
in /bin.

Type exit to halt the system.
#
```

QEMU: CPU and system emulator



Pig: a demo arcade game. 7000 lines.



X.org - KDrive

- ▶ Stand-alone simplified version of the X server, for embedded systems
 - ▶ Formerly known as Tiny-X
 - ▶ Kdrive is integrated in the official X.org server
- ▶ Works on top of the Linux frame buffer, thanks to the Xfbdev variant of the server
- ▶ Real X server
 - ▶ Fully supports the X11 protocol : drawing, input event handling, etc.
 - ▶ Allows to use any existing X11 application or library
- ▶ Actively developed and maintained
- ▶ X11 license
- ▶ <http://www.x.org>





Kdrive: size and dependencies

X server

Xfbdev
~1.2m

Fonts

from a few kb
to several mb

X libraries

libxcb
~300k

libXfont
~380k

liblbxutil
~156k

Misc libs
~770k

libX11
~920k

Total: 2.5m

X toolkit (optional)

libXaw6,7,8
~900k

libXt
~330k

Utils

dbus
lib: ~200k
bin: ~350k

libsfsfs
~27k

libpng
~130k

expat
~120k

zlib
~64k

freetype
~380k

pixman
~130k

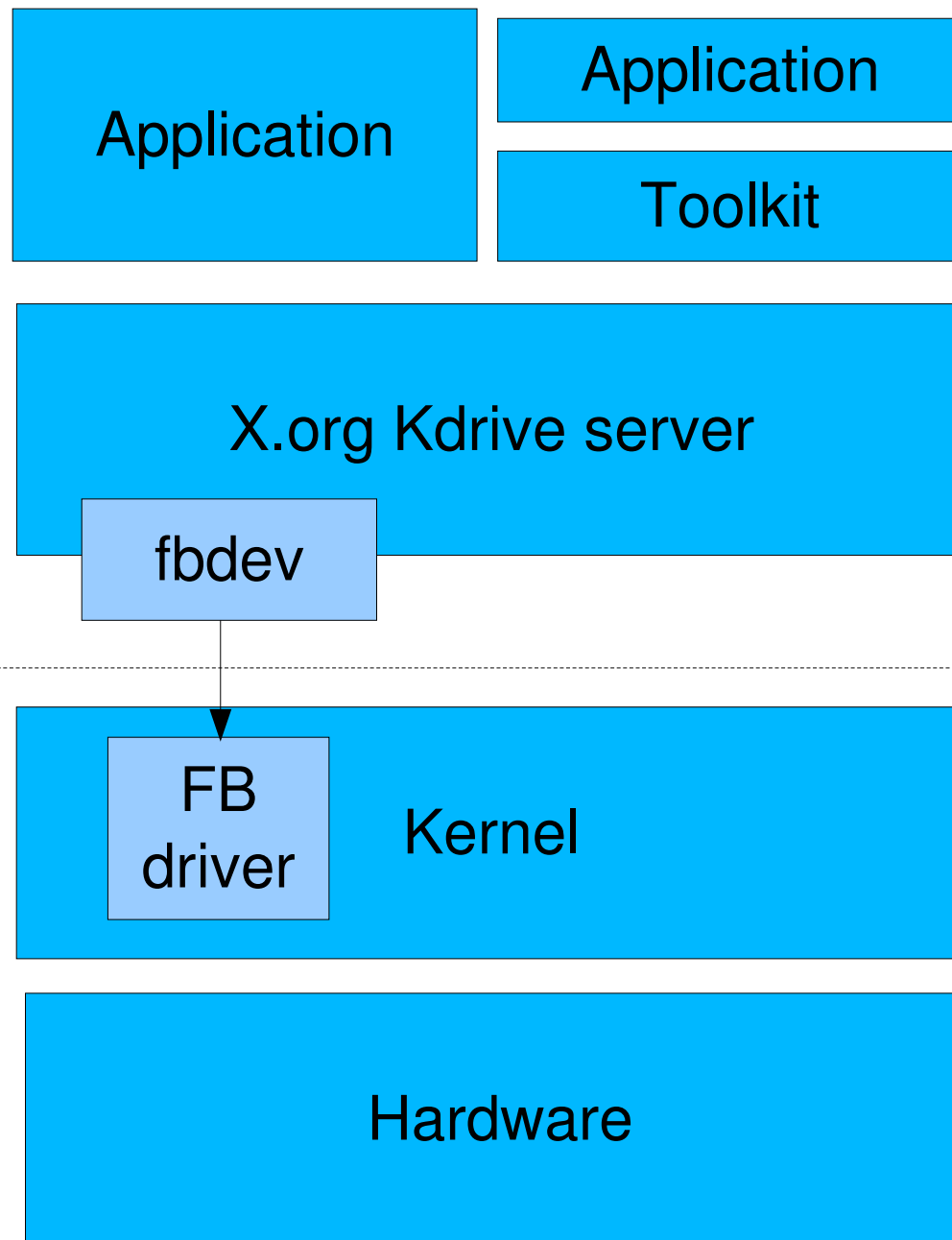
fontconfig
~165k

Total: 1.5m

**Total, without X
toolkit: 5.4m**



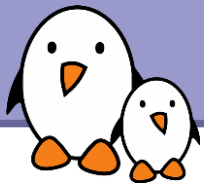
Kdrive: architecture





Kdrive: usage

- ▶ Can be directly programmed using Xlib / XCB
 - ▶ Low-level graphic library
 - ▶ Probably doesn't make sense since DirectFB is a more lightweight solution for an API of roughly the same level (no widgets)
- ▶ Or, usually used with a toolkit on top of it
 - ▶ Gtk
 - ▶ Qt
 - ▶ Fltk
 - ▶ WxEmbedded



Nano-X (1)

- ▶ An alternative graphic system
- ▶ Various graphic back-ends, including Linux framebuffer
- ▶ Provides two programming APIs
 - ▶ A Win32-like API
 - ▶ A Xlib-like API, but not compatible with Xlib
 - ▶ Requires specially designed widget toolkits
- ▶ Client/server model, with clients communicating with the server through an Unix socket. Optionally, a client can be directly linked with the server
- ▶ Lightweight



Nano-X (2)

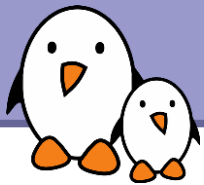
- ▶ Doesn't come with a widget library, but offers a window manager
- ▶ Relatively small user and developer base. No release since 2005, but some activity on the mailing-list, including the official maintainer.
 - ▶ Probably not lively enough to provide a long-term maintenance guarantee
- ▶ Licensed under the Mozilla Public License
- ▶ Webpage: <http://www.microwindows.org/>

349



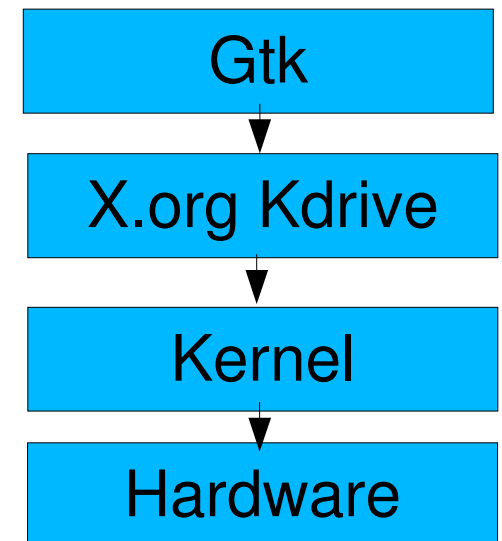
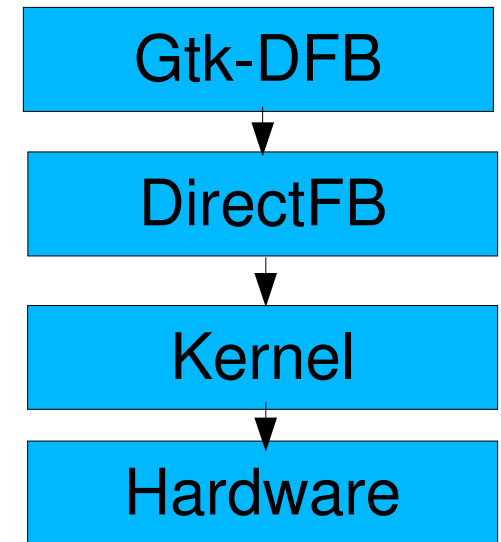
Graphical toolkits

« High-level » solutions



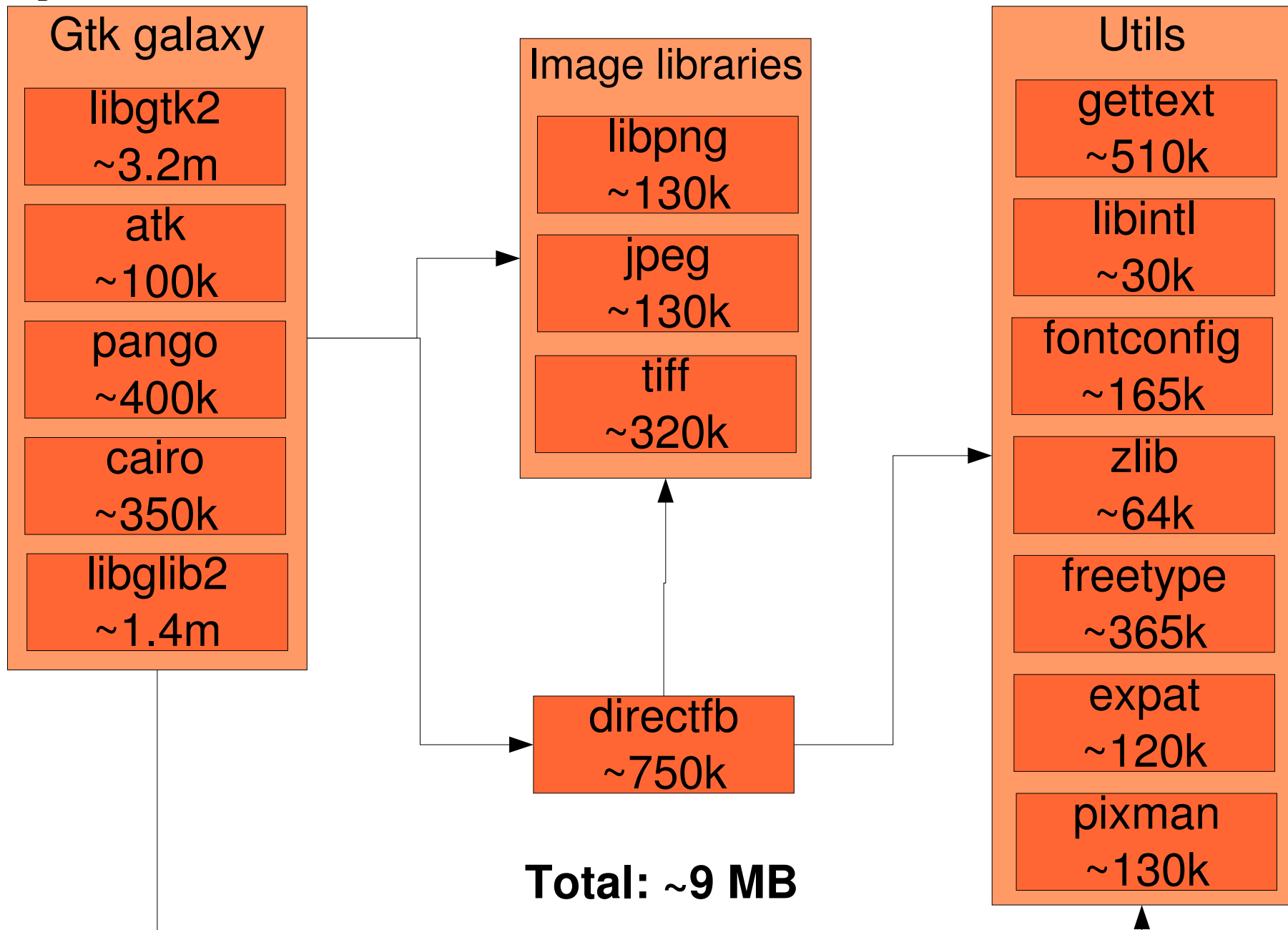
Gtk

- ▶ The famous toolkit, providing widget-based high-level APIs to develop graphical applications
- ▶ Standard API in C, but bindings exist for various languages: C++, Python, etc.
- ▶ Two GDK back-ends
 - ▶ The classical Xorg back-end
 - ▶ The DirectFB back-end, which removes the need for an Xorg server
- ▶ No windowing system, a lightweight window manager needed to run several applications. Possible solution: Matchbox.
- ▶ License: LGPL
- ▶ <http://www.gtk.org>





Gtk-DFB: dependencies and size





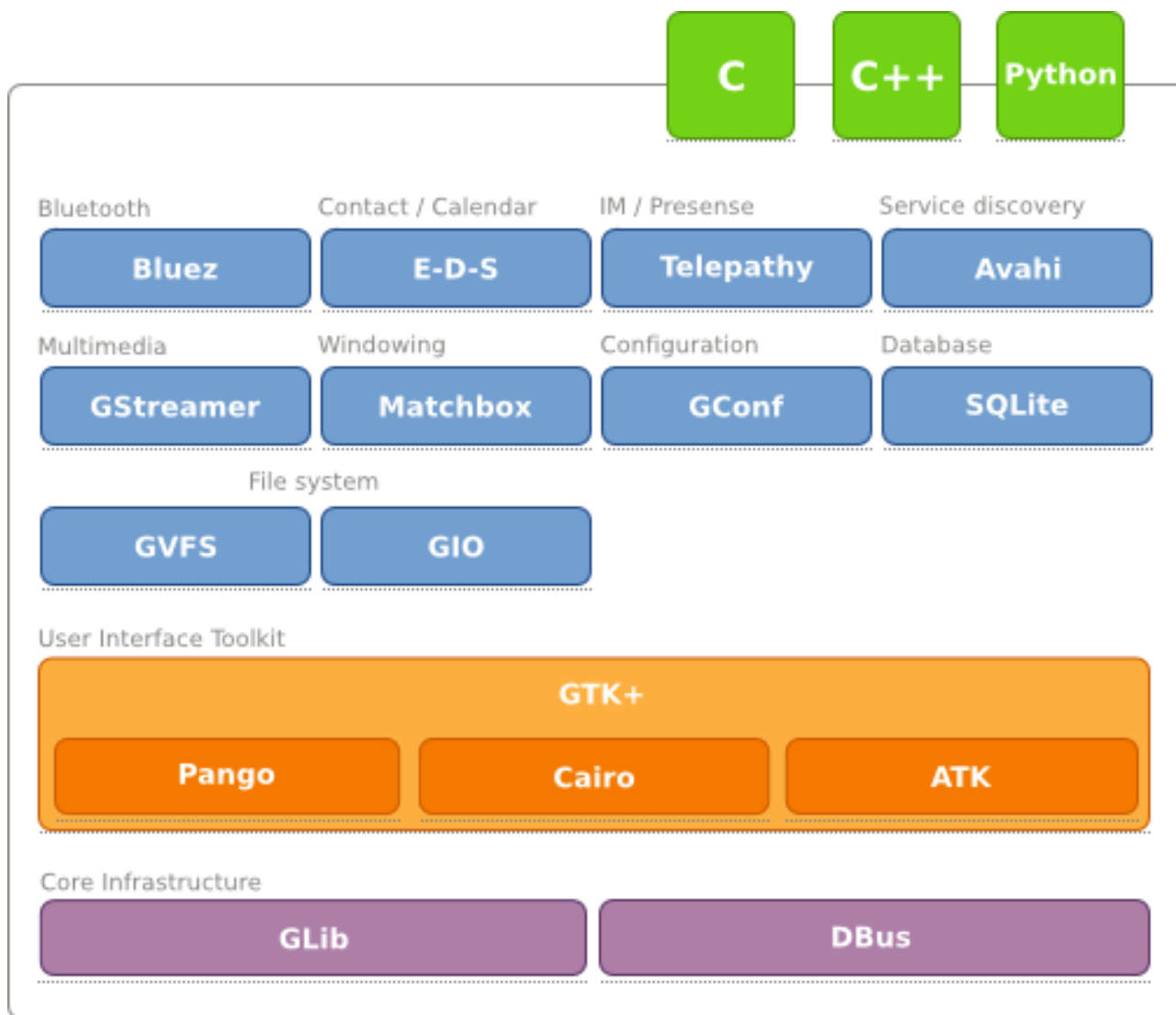
Gtk stack components

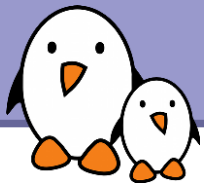
- ▶ **Glib**, core infrastructure
 - ▶ Object-oriented infrastructure GObject
 - ▶ Event loop, threads, asynchronous queues, plug-ins, memory allocation, I/O channels, string utilities, timers, date and time, internationalization, simple XML parser, regular expressions
 - ▶ Data types : memory slices and chunks, linked lists, arrays, trees, hash tables, etc.
- ▶ **Pango**, internationalization of text handling
- ▶ **ATK**, accessibility toolkit
- ▶ **Cairo**, vector graphics library
- ▶ **Gtk+**, the widget library itself
- ▶ *The Gtk stack is a complete framework to develop applications*



GNOME Mobile

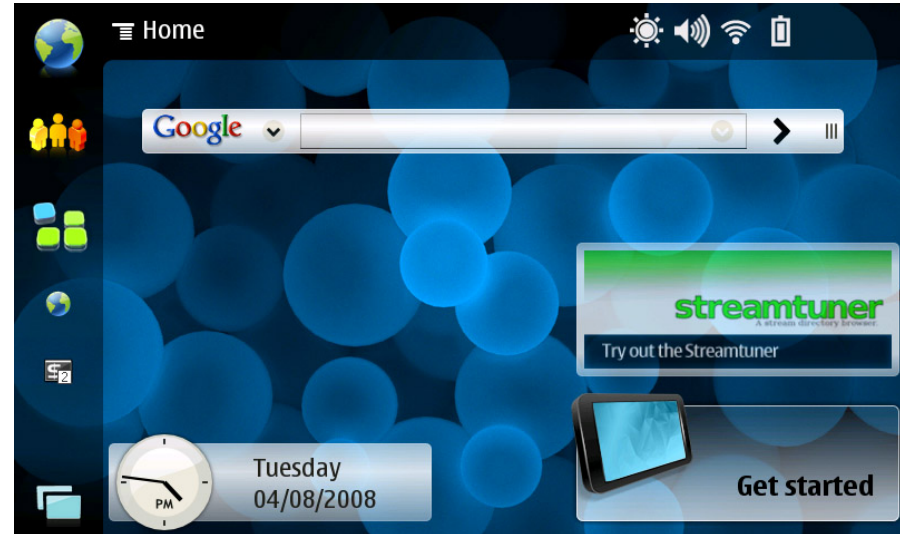
- ▶ The Gtk stack is part of the larger GNOME Mobile platform





Gtk examples

OM 2007.2 platform on
OpenMoko phone



Maemo platform on
Nokia Internet tablets



Interface of Vernier data
acquisition and
visualization systems



Qt (1)

- ▶ The other famous toolkit, providing widget-based high-level APIs to develop graphical applications
 - ▶ « Qt for Embedded Linux », formerly known as Qtopia Core, is the version of Qt that runs on top of a frame buffer, on embedded devices. It includes a windowing system
 - ▶ « Qt Extended », formerly known as Qtopia, extends « Qt for Embedded Linux » with useful components on embedded devices : communication, contents, application-specific and user experience components.
- ▶ Implemented in C++
 - ▶ the C++ library is required on the target system
 - ▶ standard API in C++, but bindings are also available for other languages

<http://www.qtsoftware.com/products/platform/qt-for-embedded-linux>

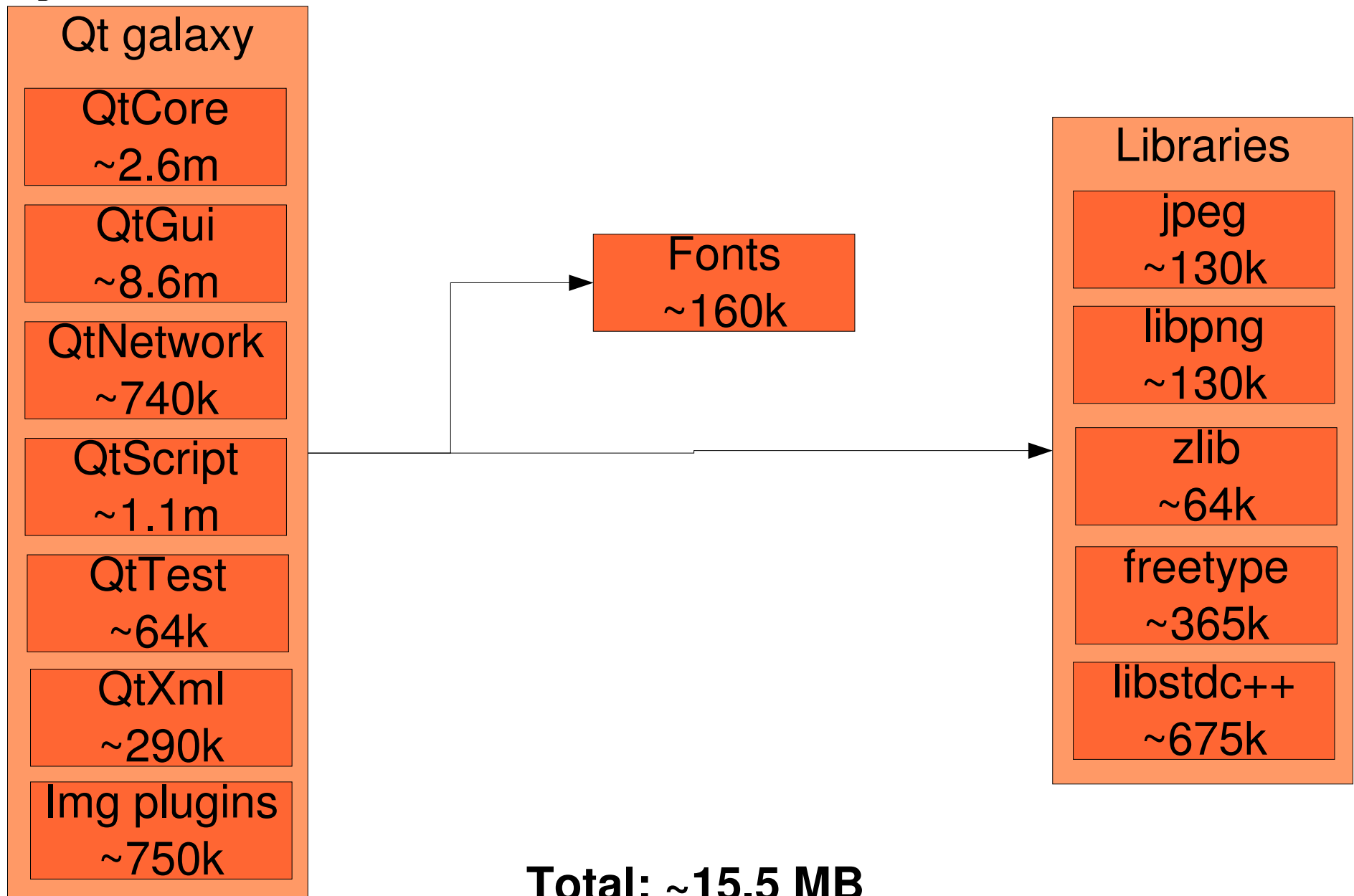


Qt (2)

- ▶ Works either on top of
 - ▶ Framebuffer
 - ▶ X11
 - ▶ DirectFB backend integrated in version 4.4, which allows to take advantage of the acceleration provided by DirectFB drivers
- ▶ Qt is more than just a graphical toolkit, it also offers a complete development framework: data structures, threads, network, databases, XML, etc.
- ▶ Very well documented
- ▶ Used to be available under the GPL license for open source applications or commercial licenses for proprietary applications
- ▶ Since version 4.5, also available under the LGPL, allowing proprietary applications



Qt: size and dependencies





Qt's usage



Qt on the OpenMoko phone



Qt on the Dash Express navigation system



Qt on the Netflix Player by Roku



FLTK



- ▶ Another high-level toolkit, providing widget-level graphic programming, currently in version 2.x
- ▶ Written in C++, main programming API in C++
- ▶ Designed with smallness in mind
 - ▶ Designed to reduce size when statically linked, small memory consumption for each widget
 - ▶ Core of the library, as included in an statically compiled hello world program: 82K. An example application which uses all widgets: 352K
- ▶ Works on top of Xlib
- ▶ Actively maintained, licensed under the LGPL
- ▶ A port of FLTK 1.x to DirectFB was made, available from DirectFB's git repository, but only maintained by a single person
- ▶ <http://www.fltk.org>

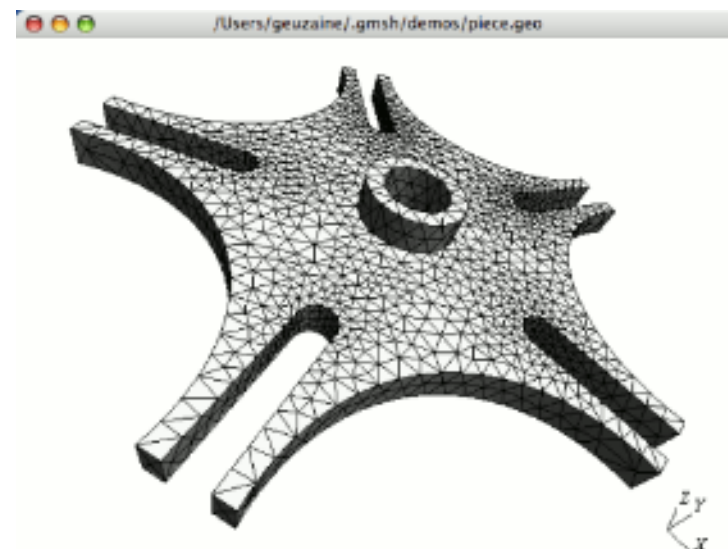


FLTK screenshots

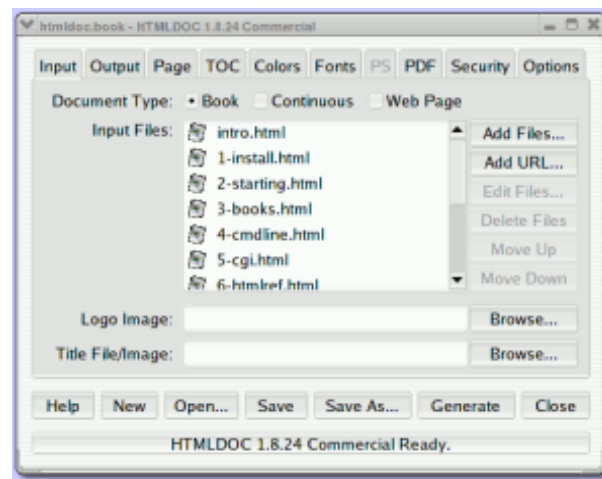
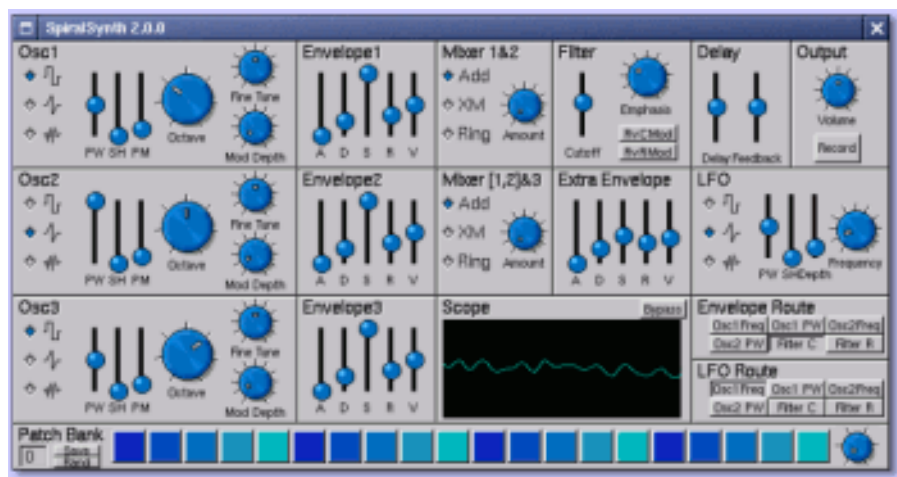


flPhoto

Gmsh



Spiral Synth

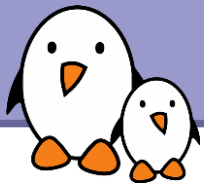


HTMLDOC



WxEmbedded (1)

- ▶ Originally, WxWidgets is a library abstracting existing toolkits (Win32 native toolkit, Gtk2, etc.), allowing the creation of portable applications
 - ▶ Depends on an underlying toolkit
- ▶ An internal toolkit, WxUniversal has been implemented. It allows to write WxWidgets applications without the need for an underlying toolkit.
- ▶ Works on top of various graphic back-ends
 - ▶ X11
 - ▶ DirectFB
 - ▶ Nano-X Xlib-like and Win32-like APIs
- ▶ Written in C++, C++ API, C++ library required



WxEmbedded (2)

► Size

- 2.5 Mb of shared library size
- 1 Mb for a simple statically-linked example, but the size doesn't increase much for more complicated examples
- While WxWidgets is active and well-maintained, the amount of attention given to WxUniversal and WxEmbedded is much smaller
 - Some widgets are not implemented, some graphic backends don't receive a lot of attention
- Licensed under the LGPL, with an exception giving more freedom on distribution of derived works in binary form
- WxEmbedded: <http://www.wxwidgets.org/docs/embedded.htm>
- WxUniversal: <http://www.wxwidgets.org/about/wxuniv.htm>



- ▶ LiTE is a Toolbox Engine
- ▶ Designed exclusively for DirectFB
- ▶ Two layers
 - ▶ Lite: low-level plumbing to ease the implementation of widgets
 - ▶ Leck: implementation of the widgets
- ▶ Only very simple widgets available
- ▶ Limited user base
- ▶ Very small: only 43k + 36k in addition to a typical DirectFB installation.
- ▶ LGPL 2.1
- ▶ <http://www.directfb.org/wiki/index.php/LiTE:About>



Other less used solutions

▶ MiniGUI

- ▶ Another toolkit, with several graphical backends including a frame buffer backend
- ▶ The GPL version is limited in functionalities, commercial versions available
- ▶ <http://www.minigui.org/>

▶ Enlightenment foundation libraries

- ▶ Very powerful, but complicated to use due to the lack of documentation
- ▶ <http://www.enlightenment.org/p.php?p=about/efl>



Tools for the target device
Web browsers



Fast and tiny web browser: Dillo

<http://www.dillo.org/>

- ▶ Very fast, lightweight web browser written in C/C++, with a FLTK2 interface (since version 2)
The Dillo binary fits in 860 KB (statically compiled on arm)
- ▶ License: GPL
- ▶ Supports many standard features: cookies, images, tables... Extensible through plugins (e.g. ftp)
- ▶ Fits well on small screens
- ▶ Still missing: frames, CSS...

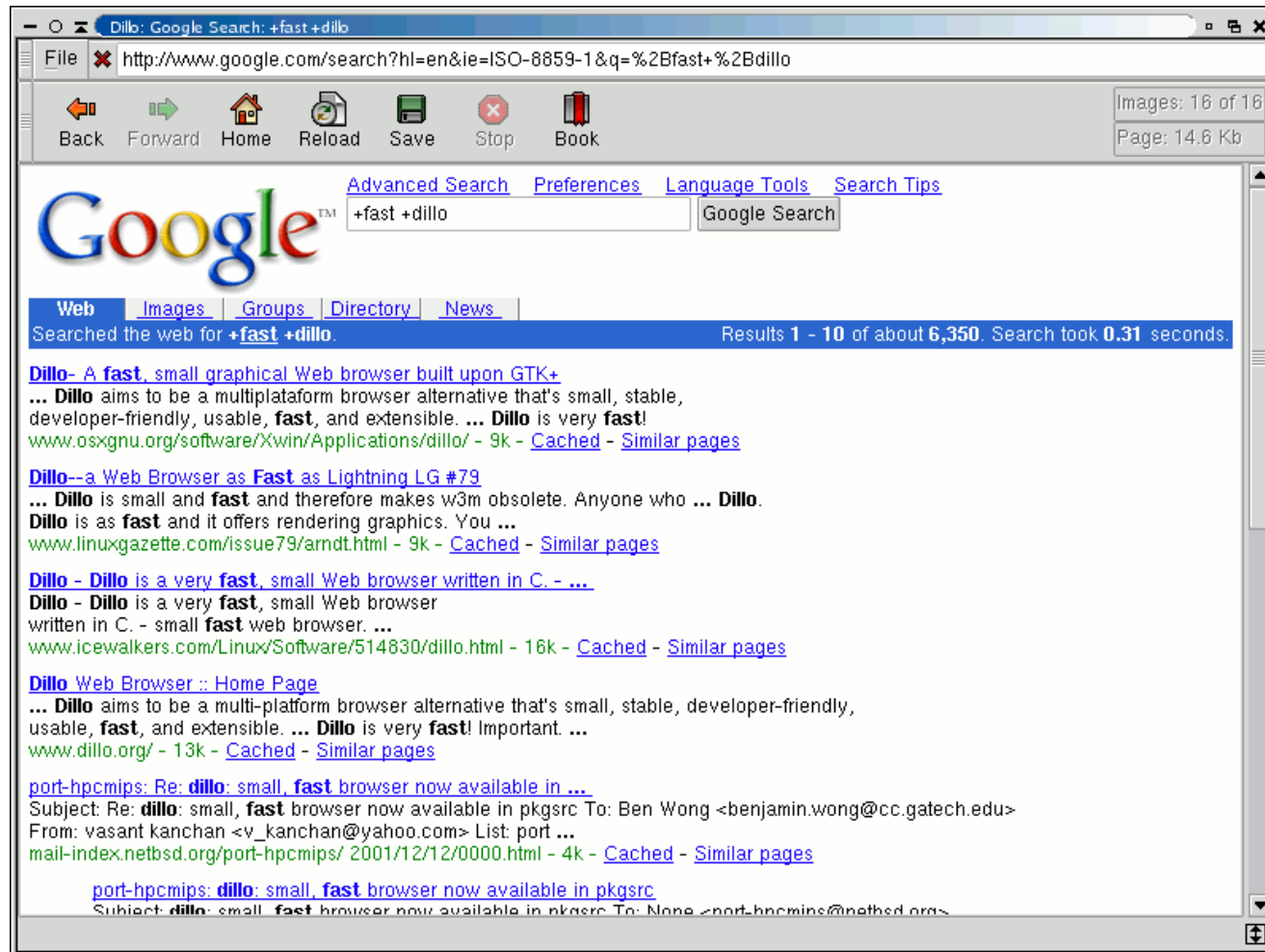


MiniGUI based implementation:

<http://www.minigui.org/app/app-edillo1.shtml.shtml>



Dillo screenshots



Old screenshots from
the time when Dillo
was based on GTK



Links

<http://links.twibright.com/>

- ▶ Portable browser with many features: frames, tables, SSL, Javascript...
- ▶ Runs either in text mode or with several graphical back-ends: [X](#), [DirectFB](#), [SDL](#), [SVGAlib](#).
- ▶ Size: 4 MB ([i386](#), [glibc](#)) + shared libraries ([libpng](#), [libssl](#)...)





Full featured browser: Mozilla Firefox

<http://www.mozilla.org/products/firefox/>

- Lightweight and fast browser based on [Mozilla](#)
- Full featured: [CSS](#), [SSL](#), [Javascript](#), tabbed browsing, pop-up blocking..., but very easy to configure.
- Takes around 40 MB of RAM with 8 tabs open.
Need 25 MB of storage space ([Sharp Zaurus](#))
- Designed to be cross-platform. Already used in embedded systems with sufficient screen resolution (web pads, high-end PDAs)
- Great for consumers appliances. Looks familiar to consumers: the default theme recalls [IE](#).



WebKit

<http://webkit.org/>

- ▶ Web browser engine.
Application framework that can be used to develop web browsers.
- ▶ License: portions in LGPL and others in BSD.
Proprietary applications allowed.
- ▶ Used in Apple's Safari browser.
Many applications (browsers, e-mail clients...) are already using WebKit:
<http://trac.webkit.org/projects/webkit/wiki/Applications%20using%20We>
- ▶ Multiple graphical back-ends: Qt4, GTK...
- ▶ You could use it to create your custom browser.





Tools for the target device

Text editors



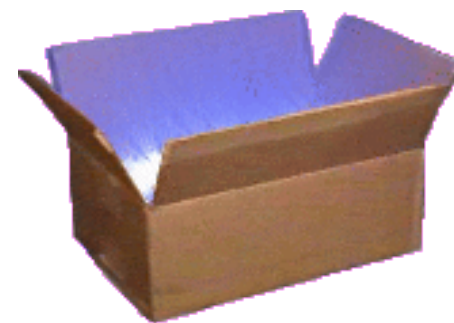
Text editors

- ▶ Graphical text editors are fine, but often can't be used on embedded systems when you just have a text console.
- ▶ For small changes, using a text-only editor through a remote connection to the target is much more efficient than editing files on the host and transferring them over and over again.
- ▶ You could use standard text-only editors from the desktop: See http://free-electrons.com/training/intro_unix_linux
- ▶ However, none of them are designed to be small in size.



BusyBox vi

- ▶ If you are using **BusyBox**, adding **vi** supports only adds **20K**.
(built with shared libraries, using **uClibc**).
- ▶ You can select which exact features to compile in.
- ▶ Users hardly realize that they are using a lightweight **vi** version!
- ▶ Tip: you can learn **vi** on the desktop,
by running the **vimtutor** command.





e3 editor

<http://freshmeat.net/projects/e3/>

- ▶ Extremely small text-only text editor:

13K for **i386** Linux

35K for **arm** Linux

Written in assembly for **i386** and **arm**

- ▶ Supports key commands
for **WordStar**, **Emacs**, **Pico**, **Nedit**, or **vi**

Key help available in the interface.

- ▶ C implementation available for other platforms
(only supports **WordStar**).

Key bindings in vi mode:

<ESC>	enter cmd mode
h,j,k,l,+,-,<Ret>,<SPC>	move by chars&lines
^B,^F,^D,^U	move by [half]page
\$,0,^,w,b,e,H,L,M,z.	move in line/screen
/,?,G	srch fwd/bwd, go EOF
ma,'a	set/go marker a
x,X,,dw,D	dele chr,word,EOL
S,C,dd,d'a,yy,y'a	subst,change,dele,yank
p,P	paste
A,a,I,i,<Ins>,O,o	enter ins.mode
R,r	enter ovw.mode,ovw.chr
J	join lines
'ZZ,^Z, u	save&quit,suspend, undo!
;,#	E3 SPECIAL: set edit mode,calculate
:w,:wq,:x,:q,:q!, :e	ex mode:save,quit,save_as,edit other
:0,:\$, :<line#>	ex mode:go BOF,EOF,line
:h	ex mode:help
:<other cmd>	pipe buffer thru 'sed'

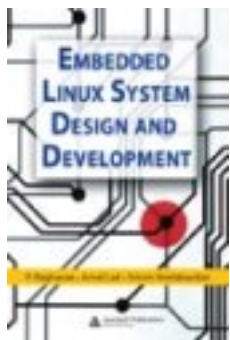
mode EX: h



References



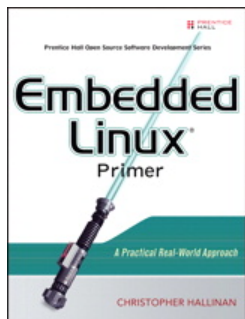
Useful reading (1)



- ▶ Embedded Linux System Design and Development ★ ★
P. Raghavan, A. Lad, S. Neelakandan, Auerbach, Dec. 2005.
<http://free-electrons.com/redirect/elsdd-book.html>
Useful book covering most aspects of embedded Linux system development (kernel and tools).



- ▶ Building Embedded Linux Systems, O'Reilly ★ ★ ★
By Karim Yaghmour, Jon Masters, Gilad Ben-Yossef and Philippe Gerum, and others (including Michael Opdenacker), August 2008
<http://oreilly.com/catalog/9780596529680/>



- ▶ Embedded Linux Primer, Prentice Hall ★ ★ ★
By Christopher Hallinan, September 2006
Covers a very wide range of interesting topics.



Useful reading (2)

► <http://www.denx.de/wiki/DULG/Manual>



Lots of useful command examples, generic help and advice for embedded Linux systems.

See <http://www.linuxdevices.com/articles/AT2969812114.html> for more books on Linux for embedded systems.



Useful web sites

LinuxDevices.com: <http://linuxdevices.com>

- ▶ Weekly newsletter with news and announcements about embedded devices running Linux.
- ▶ Articles, whitepapers, and Linux embedded devices catalog.
- ▶ An excellent site to follow industry news!





International conferences

Useful conferences featuring embedded Linux and kernel topics

- ▶ Ottawa Linux Symposium (July): <http://linuxsymposium.org/>
Kernel and system development presentations.
Freely available proceedings.



- ▶ Fosdem: <http://fosdem.org> (Brussels, February)
For developers. Presentations about system development.



- ▶ Embedded Linux Conference:
<http://embeddedlinuxconference.com/>
Organized by the CE Linux Forum: California
(San Francisco, April), in Europe (October-November).
Very interesting kernel and userspace topics for embedded
systems developers. Presentation slides freely available



CE Linux Forum

- ▶ Don't miss our free conference videos on
<http://free-electrons.com/community/videos/conferences/>!



Introduction to software development tools

Michael Opdenacker
Thomas Petazzoni
Free Electrons

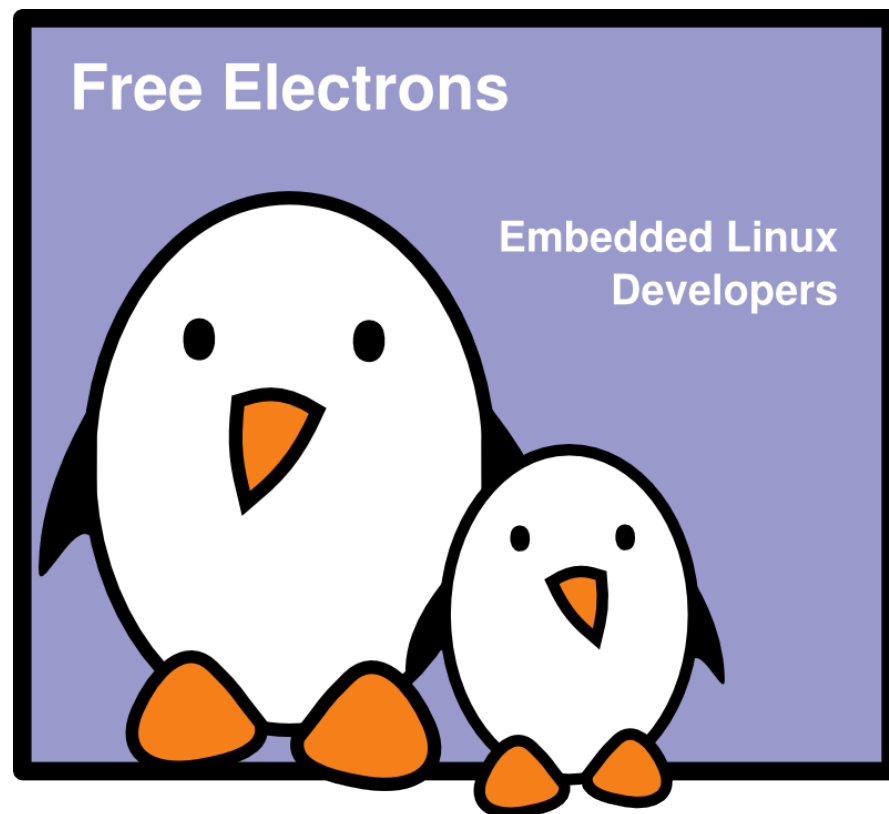
.S

.h

make

.C

.c++





Contents

Source management

- ▶ Integrated development environments (IDEs)

Debugging and analysis tools

- ▶ Debuggers
- ▶ Static code checkers
- ▶ Memory checkers
- ▶ System analysis

Development environments

- ▶ Developing on Windows



Source management Integrated Development Environments (IDE)



Source browsers

- ▶ **LXR: Linux Cross Reference**

Allows to browse code through a web browser.

- ▶ **cscope**

Console mode source browsing tool.

Integrates with editors like **vi** and **emacs**.

- ▶ **KScope**

A graphical interface to **cscope**.

See our Linux kernel and driver development training materials (<http://free-electrons.com/training/drivers>) for more details.



KDevelop

<http://kdevelop.org>

- ▶ A full featured IDE!
- ▶ License: GPL
- ▶ Supports many languages:
Ada, C, C++, Database, Java, Perl, PHP, Python, Ruby, Shell
- ▶ Supports many kinds of projects:
KDE, but also GTK, Gnome, kernel drivers, embedded (Opie)...
- ▶ Many features: editor, syntax highlighting, code completion, compiler interface, debugger interface, file manager, class browser...



Nice overview: <http://en.wikipedia.org/wiki/Kdevelop>



KDevelop screenshot

Ruby
debugger

trymain - file:/home/duke/play/testit/trymain/src/main.rb - KDevelop

File Edit View Project Build Debug Bookmarks Window Tools Settings Help

(no function)

arks Classes File Groups Variables / Watch

Variable	Value
app	#<KDE::Application:0x3...
children	Array (5 element(s))
[0]	#<Qt::Object:0x3004903...
[1]	#<Qt::SessionManager...
[2]	#<Qt::EventLoop:0x300...
[3]	#<Qt::Object:0x30048df0>
[4]	#<Qt::Translator:0x3004...
metaObject	#<Qt::MetaObject:0x0>
name	"trymain"
receivers	Hash (1 element(s))
["aboutToQuit()"]	Array (1 element(s))
[0]	#<Qt::Connection:0x300...
memberName	"shutDown()"
memberType	SIGNAL
object	#<KDE::Application:0x3...
args	nil
description	"A KDE Application"

Expression to watch:

Add

trymain.rb trymainiface.rb pref.rb main.rb

```
description = I18N_NOOP("A KDE Application")
version = "0.1"
options = [ [ "+[URL]", I18N_NOOP( "Document to open" ), ""

about = KDE::AboutData.new("trymain", I18N_NOOP("TryMain"),
                           KDE::AboutData::License_GPL, "(C) 2005 F
about.addAuthor( "Richard Dale", nil, "Richard_Dale@tipitina
KDE::CmdLineArgs.init(ARGV, about)
KDE::CmdLineArgs.addCmdLineOptions(options)
app = KDE::Application.new

# register ourselves as a dcop client
app.dcopClient().registerAs(app.name, false)

# see if we are starting with session management
if app.restored?
  RESTORE(TryMain)
else
  # no session.. just start up normally
  args = KDE::CmdLineArgs.parsedArgs
  if args.count == 0
    widget = TryMain.new
    widget.show
  else
    for i in 0...args.count do
```

Application Diff Messages Find in Files Replace Konsole Breakpoints CTAGS Frame Stack

1 #<Thread:0x3005b798 run> /home/duke/play/testit/trymain/src/main.rb:22
#1 /home/duke/play/testit/trymain/src/main.rb:22



<http://www.anjuta.org/>



Anjuta DevStudio

- ▶ **Gnome's IDE**
Less features than **KDevelop** so far.
- ▶ License: GPL
- ▶ Supported languages: C and C++
- ▶ Supported project types: command line, **GTK**, **Gnome**, **wxWindows**, **Xlib**.
- ▶ Features: integrated editor, syntax highlighting, code completion, compiler and debugger interface, tag browser.



Anjuta screenshot

The screenshot displays the Anjuta IDE interface. The main window is titled 'Anjuta' and contains several panes:

- Symbols:** A tree view on the left showing a hierarchy of classes. The 'CallTip' class is expanded, showing methods like 'CallTip(const Ca', 'CallTip()', 'CallTipCancel()', 'CallTipStart(int p', 'DrawChunk(Surf', 'MouseClicked(Poin', 'PaintCT(Surface', 'PaintContents(Si', and 'RefreshColourPa'.
- Inheritance Graph:** A central pane showing a graph of class inheritance. The graph includes nodes for 'DocWatcher', 'LexerModule', 'Font', 'Window', 'wxApp', 'Editor', 'ExternalLexerModule', 'FontCached', 'ListBox', 'MyApp', 'ScintillaBase', and 'ScintillaGTK'. Arrows indicate the inheritance relationships between these classes.
- Project:** A pane on the right showing the project structure. It includes a list of files and directories, such as 'libanjuta-interfaces.la', 'libanjuta.la', 'libanjuta-egg.la', 'test-multi-drag', 'test-tree-utils', 'test-actions', 'test-actions.c', and 'test-union'.
- Terminal:** A pane at the bottom showing a terminal window with the command prompt '[naba@localhost naba]\$' and the output of the 'ls' command, listing files and directories in the current directory.

The terminal output shows the following files and directories:

```
[naba@localhost naba]$ ls
accounts  belzabar  Documents  evolution  hexagon.png  logo  Movies  rpms  vnc-scripts
anjuta    Desktop  Downloads  GNUstep    Images       mail  Projects  test.txt
```

Anjuta's class inheritance graph and terminal



Eclipse (1)

<http://www.eclipse.org/>



- ▶ An extensible, plugin based software development kit, typically used for creating IDEs.
- ▶ Supported by the Eclipse foundation, a non-profit consortium of major software industry vendors (IBM, Intel, Borland, Nokia, WindRiver, Zend, Computer Associates...).
- ▶ Free Software license (Eclipse Public License). Incompatible with the GPL.
- ▶ Supported platforms: GNU/Linux, Unix, Windows

Extremely popular: created a lot of attraction.



Eclipse (2)

- ▶ Eclipse is actually a platform composed of many projects:
<http://www.eclipse.org/projects/>
- ▶ The platform is used by major embedded Linux software vendors for their (proprietary) system development kits:
MontaVista DevRocket, TimeSys TimeStorm, Windriver Workbench, Sysgo ELinOS.

Eclipse is a huge project.

It would require an entire training session!



Debugging and analysis tools

Debuggers

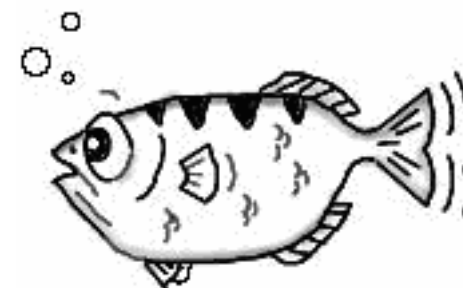


GDB

The GNU Project Debugger

<http://www.gnu.org/software/gdb/>

- ▶ The debugger on GNU/Linux, available for most embedded architectures. Supported languages: C, C++, Pascal, Objective-C, Fortran, Ada...
- ▶ Console interface (useful for remote debugging). Graphical front-ends available.
- ▶ Can be used to control the execution of a program, set breakpoints or change internal variables. You can also use it to see what a program was doing when it crashed (by loading its memory image, dumped into a **core** file).



See also <http://en.wikipedia.org/wiki/Gdb>



GDB graphical front-ends

- ▶ **DDD - Data Display Debugger**
<http://www.gnu.org/software/ddd/>
The most popular graphical front-end,
with advanced data plotting capabilities.
- ▶ **GDB/Insight**
<http://sources.redhat.com/insight/>
From the **GDB** maintainers (Red Hat).
- ▶ **KDbg**
<http://www.kdbg.org/>
Another front-end, for the **K Display Environment**.



Introduction to software development tools

Debugging and analysis tools
Remote debugging



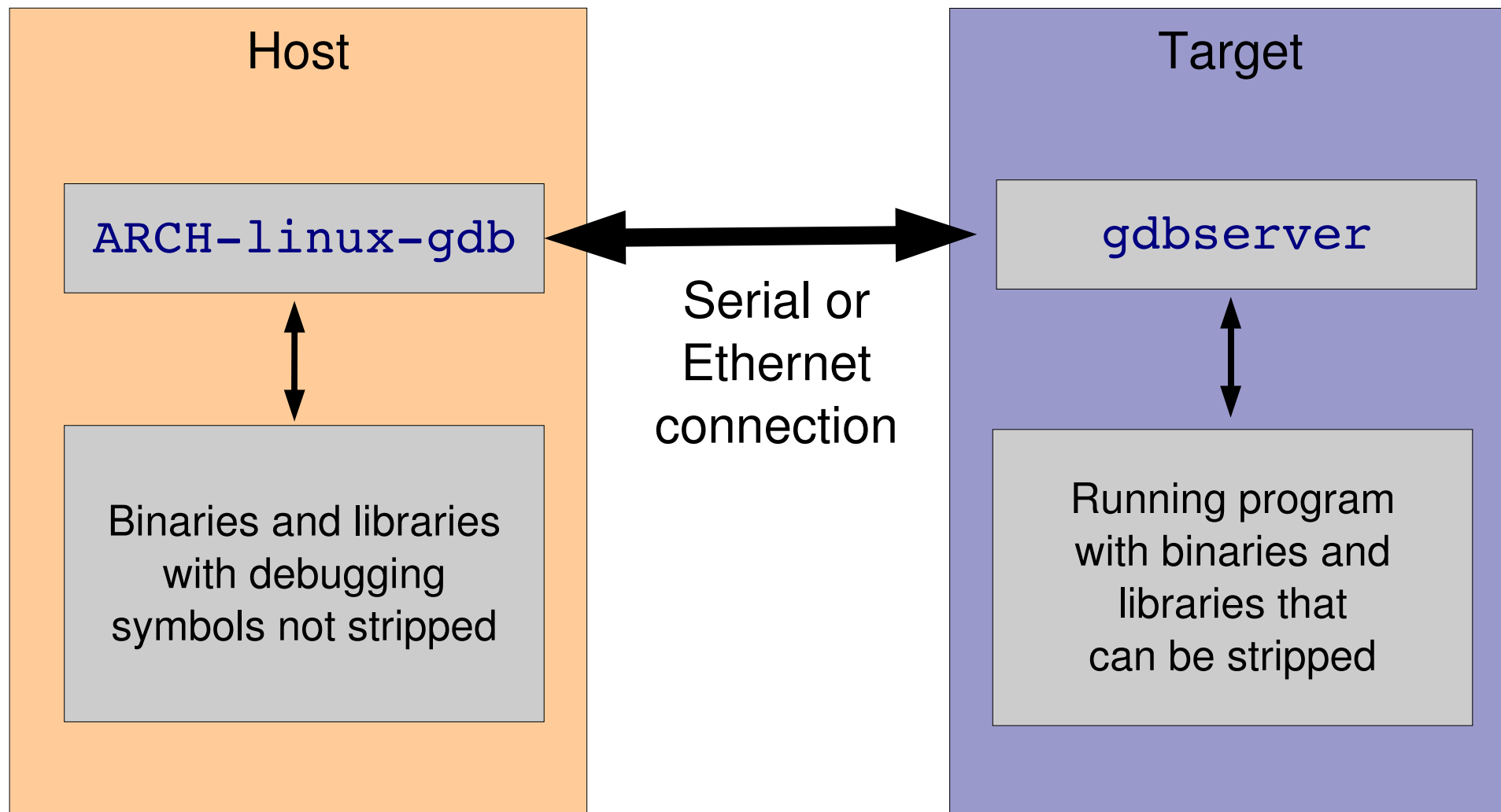
Remote debugging

- ▶ In a non-embedded environment, debugging takes place using `gdb` or one of its front-end.
- ▶ `gdb` has direct access to the binary and libraries compiled with debugging symbols.
- ▶ However, in an embedded context, the target platform environment is often too limited to allow direct debugging with `gdb` (2.4 MB on `x86`).
- ▶ Remote debugging is preferred
 - ▶ `gdb` is used on the development workstation, offering all its features.
 - ▶ `gdbserver` is used on the target system (only 40 KB on `arm`).





Remote debugging: architecture





Remote debugging: requirements

► Requirements

- On the host, a toolchain compiled with gdb support : [ARCH-linux-gdb](#) must be available. It is a version of [gdb](#) that runs on your host but that understand the specificities of the target architecture
- On the target, a [gdbserver](#) program, compiled for your target architecture. It is usually part of the cross-compiling toolchain.
- A serial or Ethernet connection



Remote debugging: usage

- ▶ On the target, run a program through `gdbserver`.
Program execution will not start immediately.
`gdbserver localhost:<port> <executable> <args>`
- ▶ Otherwise, attach `gdbserver` to an already running program:
`gdbserver --attach localhost:<port> <pid>`
- ▶ Then, on the host, run `ARCH-linux-gdb` program,
and use the following `gdb` commands:
 - ▶ To connect to the target:

<code>gdb > target remote <target>:<port></code>	(networking)
<code>gdb > target remote /dev/ttyS0</code>	(serial link)
 - ▶ To tell `gdb` where shared libraries are:

<code>gdb > set sysroot <library-path></code>	(without <code>lib/</code>)
--	------------------------------



Software development tools

Debugging and analysis tools
Static code checkers



Splint

<http://splint.org/>, from the University of Virginia

- ▶ GPL tool for statically checking C programs for security vulnerabilities and coding mistakes
- ▶ Today's `lint` program for GNU/Linux.
The successor of LClint.
- ▶ Very complete manual and documentation
- ▶ Doesn't support C++



Debugging and analysis tools

Memory checkers



memcheck

<http://hald.dnsalias.net/projects/memcheck/>

- ▶ GNU GPL tool for dynamic memory checking
- ▶ Works by replacing glibc's memory management functions by its own.
- ▶ Supports most useful CPU architectures.



DUMA

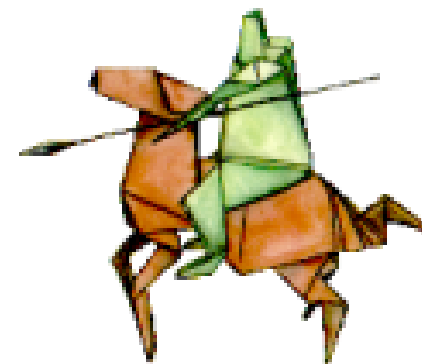
Detect Unintended Memory Access

<http://duma.sourceforge.net/>

- ▶ Fork and replacement for Electric Fence
- ▶ Stops your program on the exact instruction that overruns or underruns a `malloc()` memory buffer.
- ▶ GDB will then display the source-code line that causes the bug.
- ▶ Works by using the virtual-memory hardware to create a red-zone at the border of each buffer - touch that, and your program stops.
- ▶ Works on any platform supported by Linux, whatever the CPU (provided virtual memory support is available).



Valgrind (1)



<http://valgrind.org/>

- ▶ GNU GPL Software suite for debugging and profiling programs.
- ▶ Supported platforms: Linux on [x86](#), [x86_64](#), [ppc32](#), [ppc64](#)
Others: compile your program to these platforms to use [Valgrind](#).
- ▶ Can detect many memory management and threading bugs.
- ▶ Profiler: provides information helpful to speed up your program and reduce its memory usage.
- ▶ The most popular tool for this usage.
Even used by projects with hundreds of programmers.



Valgrind (2)

- ▶ Can be used to run any program, without the need to recompile it.
- ▶ Example usage
`valgrind --leak-check=yes ls -la`
- ▶ Works by adding its own instrumentation to your code and then running in on its own virtual x86 (or ppc) core. Significantly slows down execution, but still fine for testing!
- ▶ More details on <http://valgrind.org/info/> and http://valgrind.org/docs/manual/coregrind_core.html#howwor





Valgrind on other platforms

To check programs on non x86 or non ppc CPU architectures

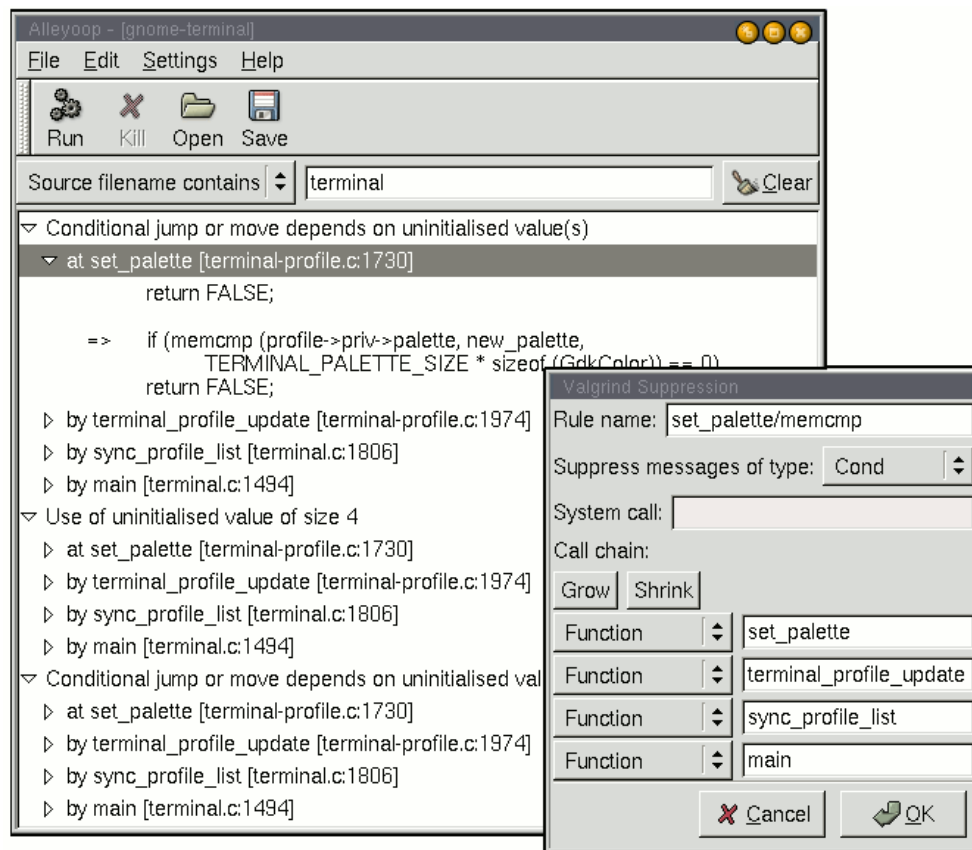
- ▶ Write portable code (always useful),
and compile it for x86 or ppc
- ▶ Then, debug your application with Valgrind.
- ▶ Once bugs are found and fixed,
you can get back to your original platform.
- ▶ This is particularly useful with uClinux,
in which memory errors are very difficult to find.



Alleyoop

<http://alleyoop.sourceforge.net/>

Graphical front-end to Valgrind





Software development tools

Debugging and analysis tools
System analysis



strace

System call tracer

<http://sourceforge.net/projects/strace/>

- ▶ Available on all GNU/Linux systems
- ▶ Allows to see what any of your processes is doing:
accessing files, allocating memory...
Easy substitute to debuggers in simple cases.
- ▶ Usage:
`strace <command>` (starting a new process)
`strace -p<pid>` (tracing an existing process)

See `man strace` for details.



ltrace

A tool to trace library calls used by a program and all the signals it receives

- ▶ Very useful complement to strace, which shows only system calls.
- ▶ Of course, works even if you don't have the sources
- ▶ Allows to filter library calls with regular expressions, or just by a list of function names.
- ▶ Manual page: <http://linux.die.net/man/1/ltrace>

See <http://en.wikipedia.org/wiki/Ltrace> for details



ltrace example output

```
strace nedit index.html
sscanf(0x8274af1, 0x8132618, 0x8248640, 0xbfaadfe8, 0) = 1
sprintf("const 0", "const %d", 0) = 7
strcmp("startScan", "const 0") = 1
strcmp("ScanDistance", "const 0") = -1
strcmp("const 200", "const 0") = 1
strcmp("$list_dialog_button", "const 0") = -1
strcmp("$shell_cmd_status", "const 0") = -1
strcmp("$read_status", "const 0") = -1
strcmp("$search_end", "const 0") = -1
strcmp("$string_dialog_button", "const 0") = -1
strcmp("$rangeset_list", "const 0") = -1
strcmp("$calltip_ID", "const 0") = -1
...
```



ltrace summary

Example summary at the end of the `ltrace` output (`-c` option)

Process 17019 detached

% time	seconds	usecs/call	calls	errors	syscall
100.00	0.000050	50	1		set_thread_area
0.00	0.000000	0	48		read
0.00	0.000000	0	44		write
0.00	0.000000	0	80	63	open
0.00	0.000000	0	19		close
0.00	0.000000	0	1		execve
0.00	0.000000	0	2	2	access
0.00	0.000000	0	3		brk
0.00	0.000000	0	1		munmap
0.00	0.000000	0	1		uname
0.00	0.000000	0	1		mprotect
0.00	0.000000	0	19		mmap2
0.00	0.000000	0	50	46	stat64
0.00	0.000000	0	18		fstat64
100.00	0.000050		288	111	total



Oprofile

<http://oprofile.sourceforge.net>

- ▶ A system-wide profiling tool
- ▶ Can collect statistics like the top users of the CPU.
- ▶ Works without having the sources.
- ▶ Requires a kernel patch to access all features, but is already available in a standard kernel.
- ▶ Requires more investigation to see how it works.
- ▶ Ubuntu/Debian packages:
[oprofile](#), [oprofile-gui](#)

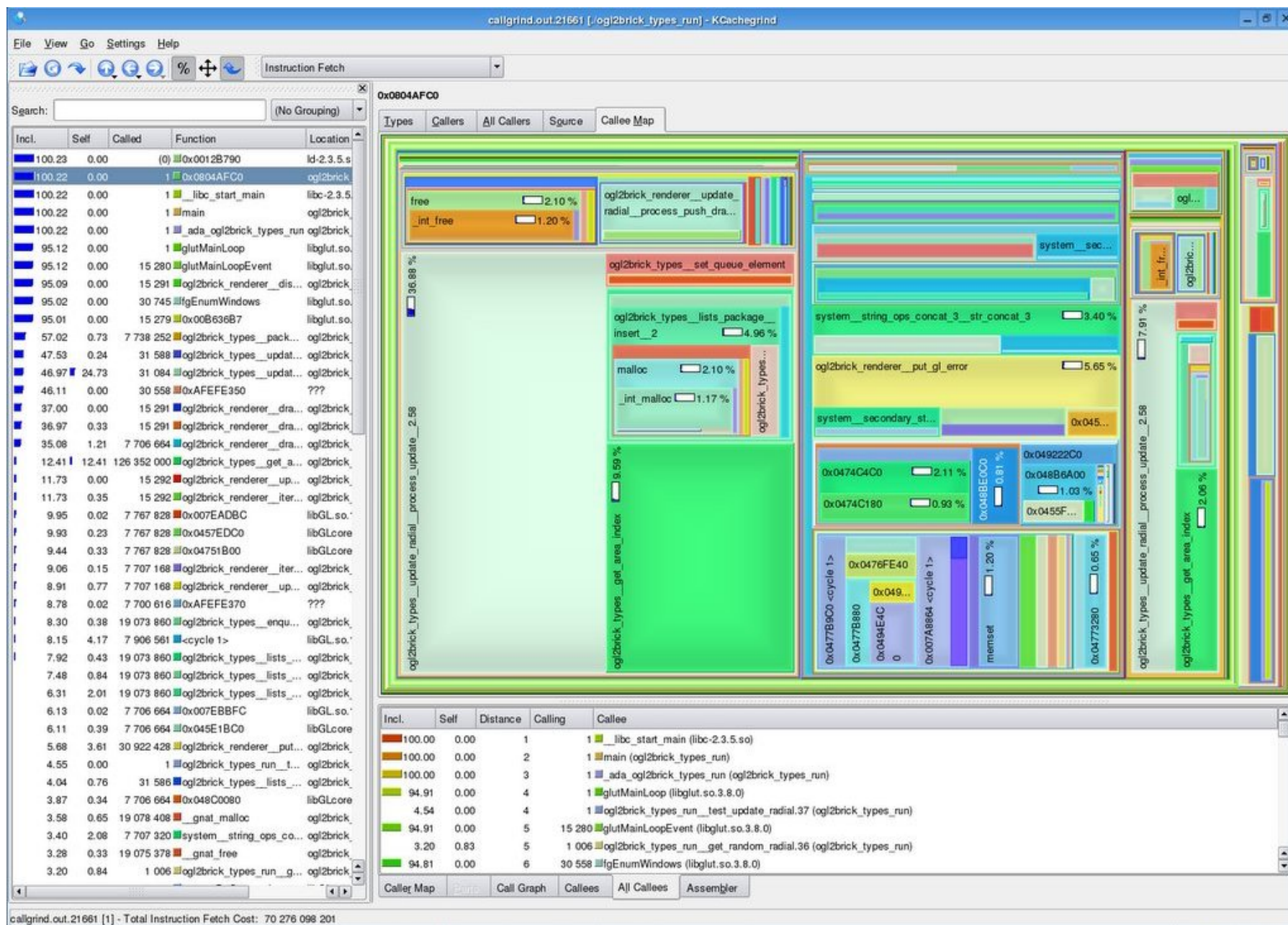


Callgrind / KCachegrind

- ▶ **Cachegrind / Callgrind**: part of the **Valgrind** tool suite
Collects function call statistics and call graphs.
Useful to know in which functions most time is spent.
- ▶ **KCachegrind**: <http://kcachegrind.sourceforge.net/>
An amazing visualizer for **Cachegrind / Callgrind** data.
- ▶ **KCachegrind** can also import data from other profilers
(such as **OProfile**), and from profiling output from Python, Perl
and PHP.



KCachegrind screenshot





Practical lab – Remote debugging

Time to start **Lab**!

- ▶ Set up remote debugging tools on the target: **strace**, **ltrace** and **gdbserver**.
- ▶ Debug a simple application running on the target using remote debugging





Developing on Windows



Developing on Windows!?

Using a GNU/Linux workstation is the easiest way to create software for GNU/Linux or embedded Linux

- ▶ You use the same tools and environment as all community developers do. Much fewer issues you are the only one to face.
- ▶ You get familiar with the system.
Essential for understanding issues.

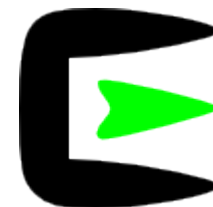
However, some developers have no choice:
Windows is the only desktop OS allowed in their company.



Cygwin

<http://cygwin.com/>

Linux (POSIX)-like environment for Windows



- ▶ 2 components:
 - Linux API emulation layer: `cygwin1.dll`
 - A collection of tools originally found in GNU/Linux
- ▶ Allows to compile and run many GNU/Linux programs on Windows: shells, compiler, http servers, [X Window](#), [GTK](#)...
- ▶ Very easy to install. Can choose which tools to download and install.
- ▶ For embedded Linux system developers:
 - makes it possible to use GNU toolchains (compiled for Windows)
 - required to build Linux binaries (kernel, libraries or applications).



Cygwin screenshot

X -

```
harold@aerosmith ~  
$ xeyes  
[3] 4040  
  
harold@aerosmith ~  
$
```

DDD: Debugger Console

File Edit View Program Commands Status Source Data Help

0: main

Xfig - ex06_fig01.fig

File Edit View Help /home/harold/cse824/nidtern/ex06_fig01.fig

Current figure "/home/harold/cse824/nidtern/ex06_fig01.fig" (40)

Mouse Buttons

1in = 1.00in

Diagram showing a sequence of events represented by circles labeled with dollar signs and letters, connected by lines indicating time intervals:

- \$A\$ (10ms) -> \$X\$ (1ms) -> \$Y\$ (100ms) -> \$H\$ (1ms) -> \$G\$ (1ms) -> \$F\$ (1ms) -> \$E\$ (1ms) -> \$D\$ (1ms) -> \$C\$ (1ms) -> \$B\$ (1ms) -> \$A\$ (10ms)

Toolbox (left):

- Shapes: Circle, Square, Rectangle, Oval, Triangle, Star, Hexagon, Polygon, Image, Text.
- Editing: Scale, Move, Copy, Paste, Rotate, Delete, Update, Edit, Zoom, Grid Mode.

Right Panel:

- Depths: All On, All Off, Toggle, Gray, Blank.
- Front: 50.
- Back: 0.

Taskbar:

- start
- Cygwin/X - Screens...
- X ~
- DDD: Debugger Co...
- Xfig - ex06_fig01.fig
- xeyes
- 12:09 AM



Cygwin limitations

Cygwin is not a complete substitute for a real GNU/Linux system.

- ▶ Almost all developers work on GNU/Linux or on another Unix platform (typically BSD). Don't expect them to test that their tools build on Windows with Cygwin.
- ▶ The number of Cygwin users is quite small.
You may be the first to face or report building issues on this platform for a given compiler or tool version.

So, the best solution is to run Linux inside Windows!



QEMU on Windows

QEMU

<http://www.h7.dion.ne.jp/~qemu-win/>

Fast processor emulator using a portable dynamic translator.

- ▶ License: GNU GPL
- ▶ Allows to run a full virtual PC, emulating processor instructions. Originally runs on [Linux](#).
- ▶ Now available on [Windows](#) too and pretty stable.
- ▶ Best to use with the [kqemu](#) accelerator (now free), otherwise compiling big projects (Linux kernel) can be very long.



VMware

<http://en.wikipedia.org/wiki/VMware>



- ▶ License: proprietary
- ▶ Can run a **GNU/Linux** PC from **Windows**, almost at the host speed.
- ▶ **VMware Player** is now available free of charge.
Many Free Software system images available for download.

The most popular solution in the corporate world.



VirtualBox

<http://virtualbox.org> from Sun Microsystems

- ▶ PC emulation solution available on both Windows and GNU/Linux
- ▶ 2 licenses:
 - ▶ Proprietary: free of cost for personal use and evaluation. Binaries available for Windows. Full features.
 - ▶ Open Source Edition (OSE): GPL license. Most features (except in particular USB support). No binaries released for Windows so far (but possible).
- ▶ Based on QEMU's core engine. Performance similar to that of WMware.

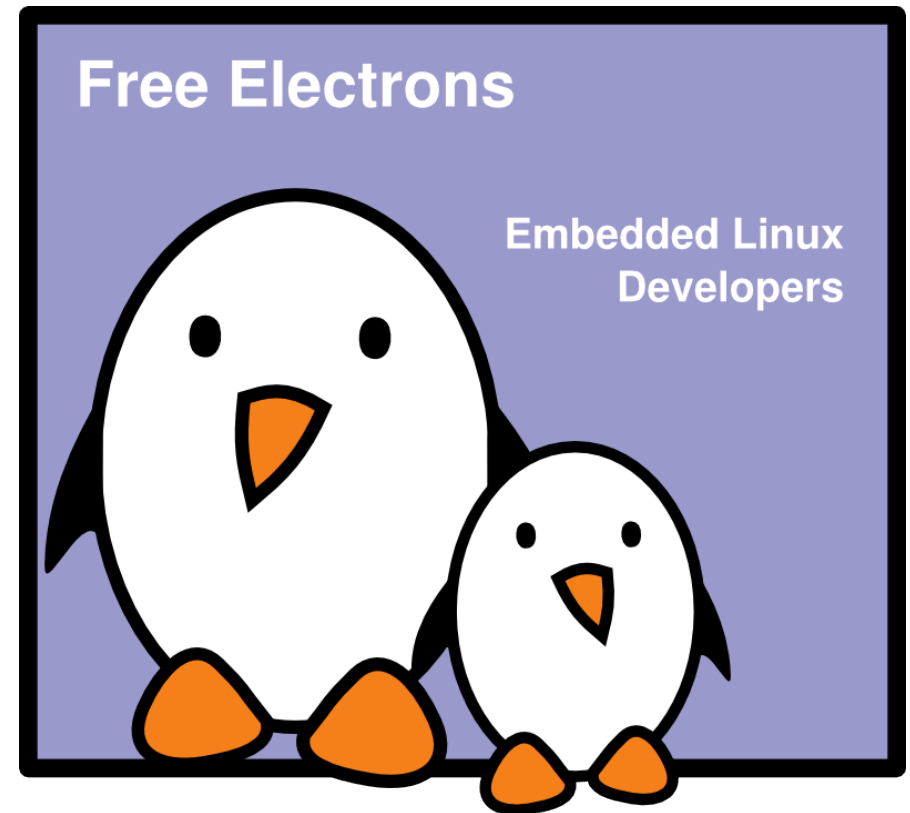
See <http://en.wikipedia.org/wiki/VirtualBox>





Real-time in embedded Linux systems

Michael Opdenacker
Thomas Petazzoni
Free Electrons





Contents

- ▶ Introduction

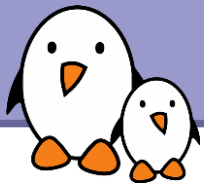
- Using Standard Linux

- ▶ Latency in Linux

- ▶ Linux 2.6 features for real-time

- ▶ Linux rt-preempt patches

- ▶ Debugging the real-time kernel



Real Time in Embedded Linux Systems

Introduction



Hard Real Time

A system is considered as a *hard real time* if it can answer to an internal or external stimulus **within a given maximum amount of time**. “Guaranteed worst case”

Hard real time systems are used wherever failing to react in time can cause a system failure or damage, or put its users in danger.

Typical examples

- ▶ Industrial process control
- ▶ Transportation
- ▶ Medicine (pacemakers, etc.)





Soft Real Time

A system is considered as *soft real time* if it is built to react to stimuli as quickly as it can: “best effort”.

However, if the system loses events or fails to process them in time, there is no catastrophic consequence on its operation. There is just a degradation in quality.

Typical examples

- ▶ Audio
- ▶ Video
- ▶ Voice over IP
- ▶ Coffee machines (in Italy, need to stop the steam pump early enough!), and many other devices.





Standard Linux and Real Time

The Vanilla Linux kernel was not designed as a hard real time system:

- ▶ Like Unix, it is a time sharing operating system designed to maximize throughput and give a fair share of the CPU in a multi-user environment.
- ▶ Non deterministic timing behavior of some kernel services: memory allocation, system calls...
- ▶ By default, processes are not preemptible when they execute system calls.

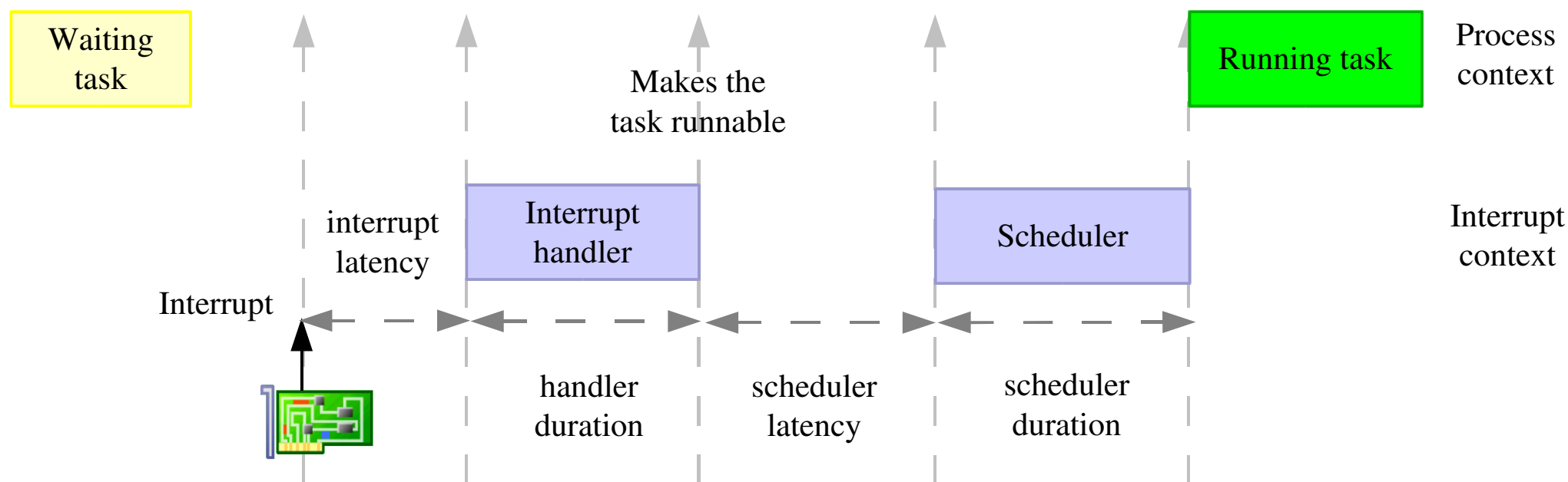


Latency in Linux



Linux kernel latency components

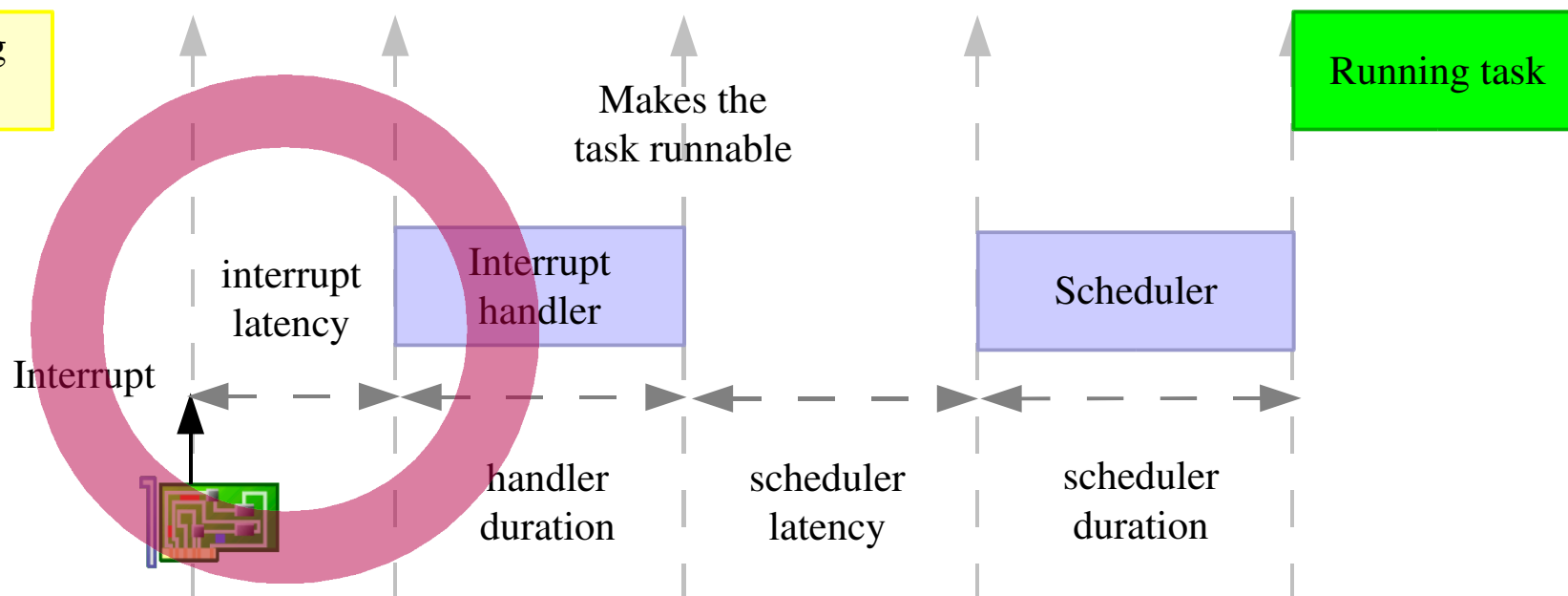
Typical scenario: process waiting for the completion of device I/O (signaled by an interrupt) to resume execution.



kernel latency = interrupt latency + handler duration
+ scheduler latency + scheduler duration



Interrupt latency



Time elapsed before executing the interrupt handler

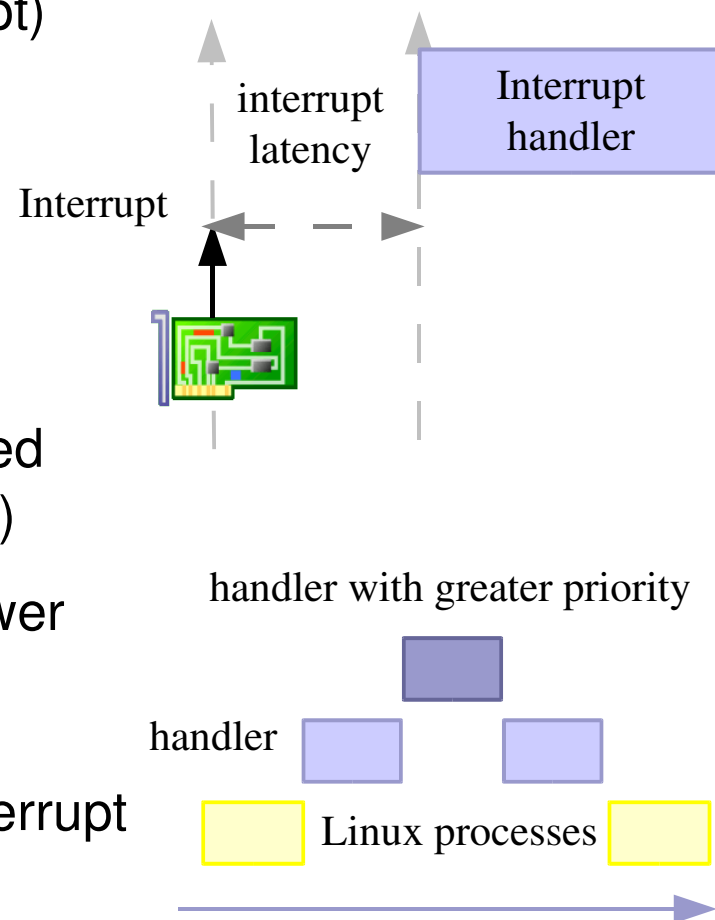


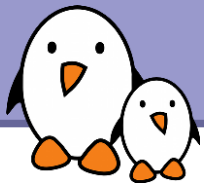
Sources of interrupt latency

Time between an event happens (raising an interrupt) and the handler is called. Sources of latency:

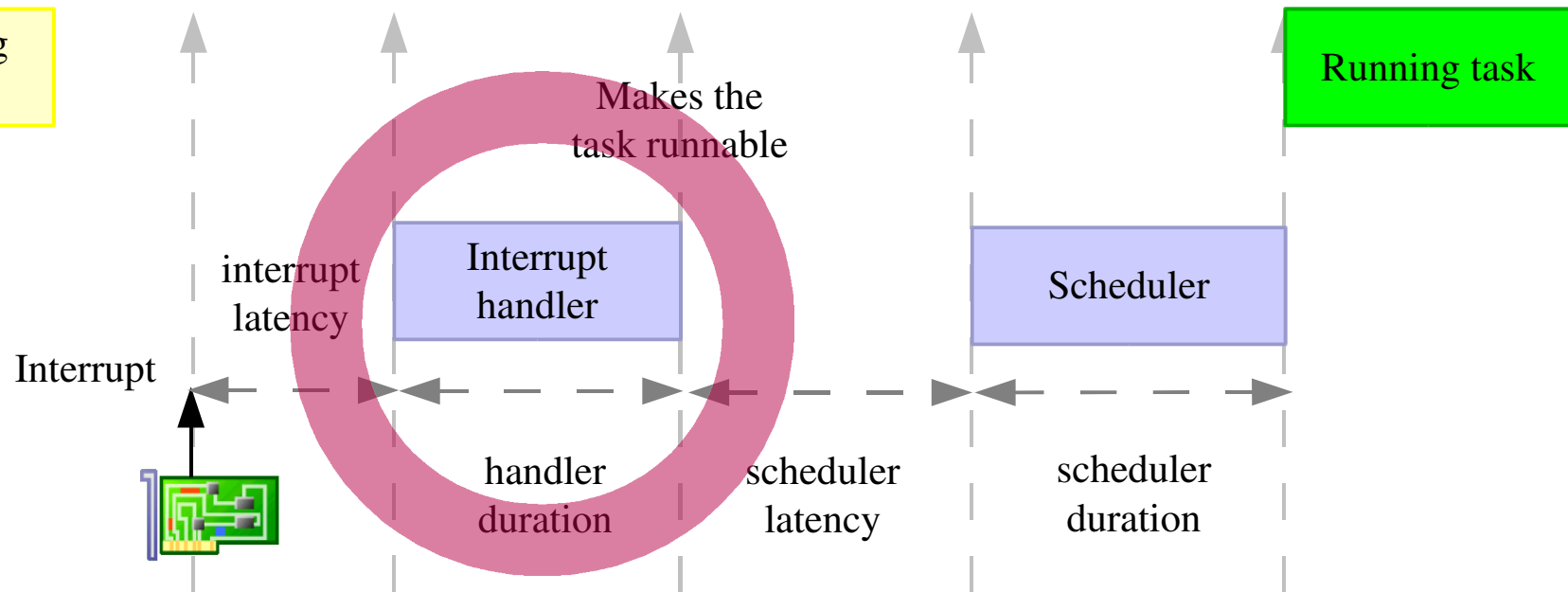
- ▶ Interrupts disabled by kernel code: spinlocks, evil drivers masking out interrupts.
- ▶ Other interrupts processed before:
 - ▶ Shared interrupt line: all handlers are executed (don't use shared IRQ lines for critical events)
 - ▶ Interrupts with higher priority can preempt lower priority ones (managed by the CPU, not by the scheduler). Not a problem in a correctly designed product: you use the top priority interrupt source.

Summary: the only real problem is disabled interrupts.





Interrupt handler duration

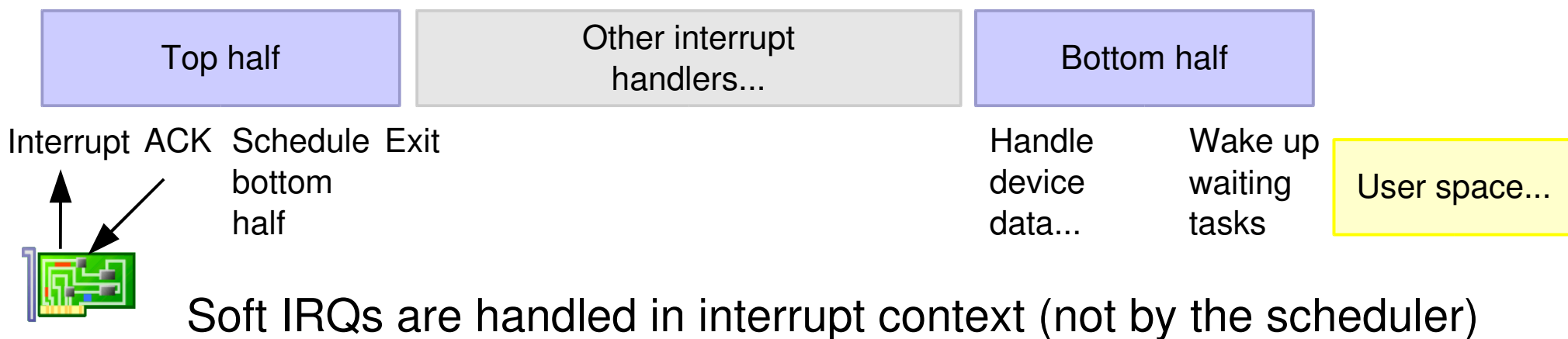


Time taken to execute the interrupt handler



Interrupt handler implementation

- ▶ Run with interrupts disabled (at least on the current line).
- ▶ Hence, they need to complete and restore interrupts as soon as possible.
- ▶ To satisfy this requirement, interrupt handlers are often implemented in 2 parts: top half and bottom half (soft IRQ)

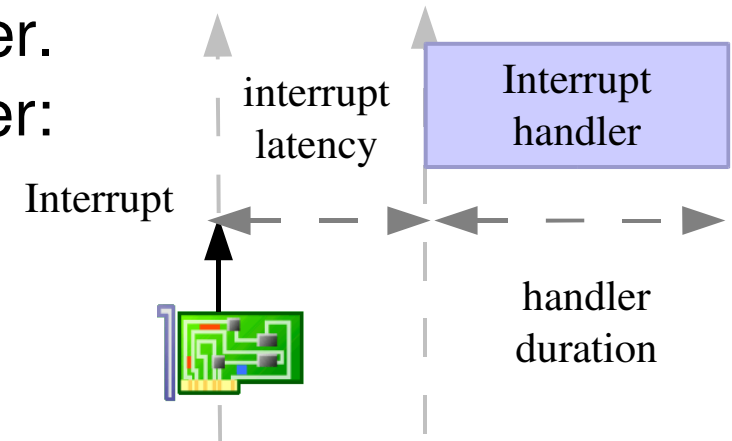




Sources of interrupt handler execution time

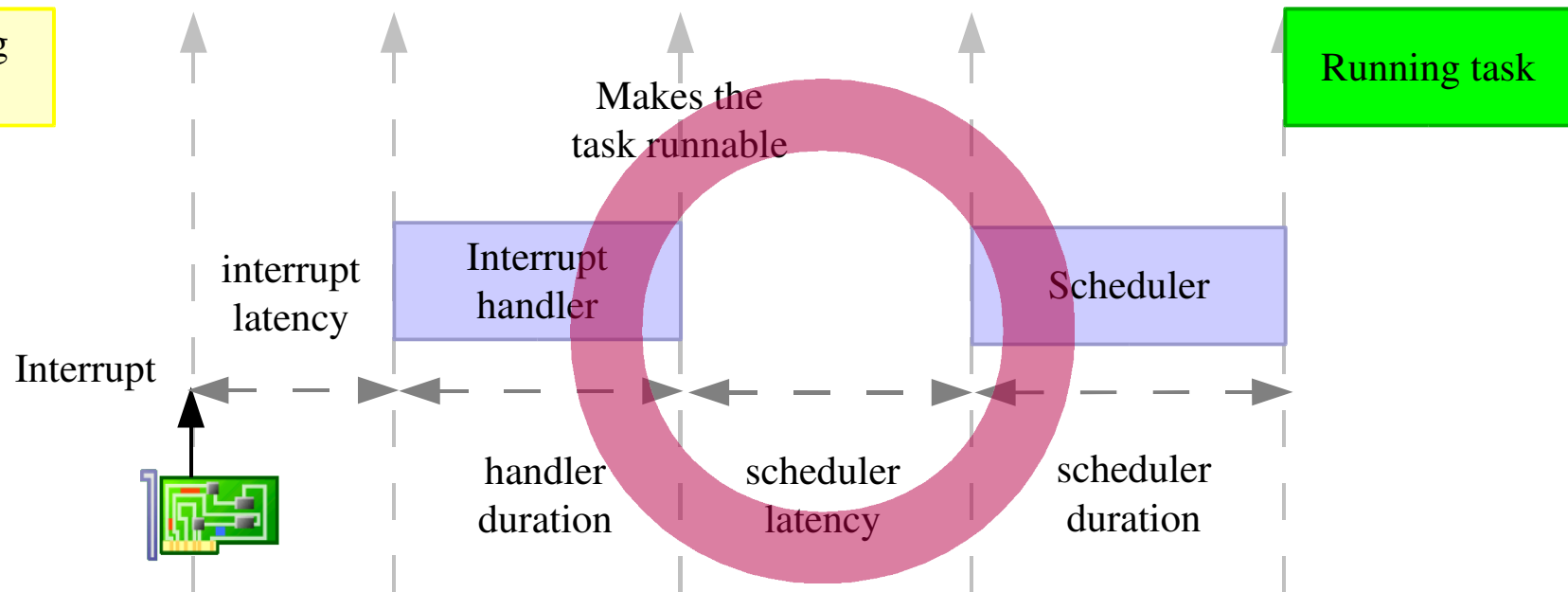
Time taken to execute your interrupt handler.
Causes which can make this duration longer:

- ▶ Preemption by other interrupts with a greater priority. Again, not a problem if priorities are configured properly.
- ▶ Interrupt handler with a softirq (“bottom half”) component:
run after all interrupts are serviced.
=> For critical processing, better not have a softirq
(may require rewriting the driver if reusing an existing one).





Scheduler latency



Time elapsed before executing the scheduler



When is the scheduler run next?

In the general case (still after a task is woken up)

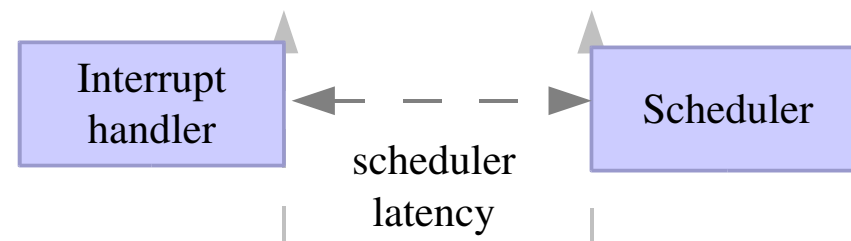
- ▶ Upon return from interrupt context (our particular case), unless the current process is running a system call (system calls are not preemptible by default).
- ▶ After returning from system calls.
- ▶ When the current process runs an action which sleeps / waits (voluntarily or not, in kernel or user mode), or explicitly calls the scheduler.



Sources of scheduler latency

Possible sources of latency:

- ▶ Other interrupts coming in (even with a lower priority).

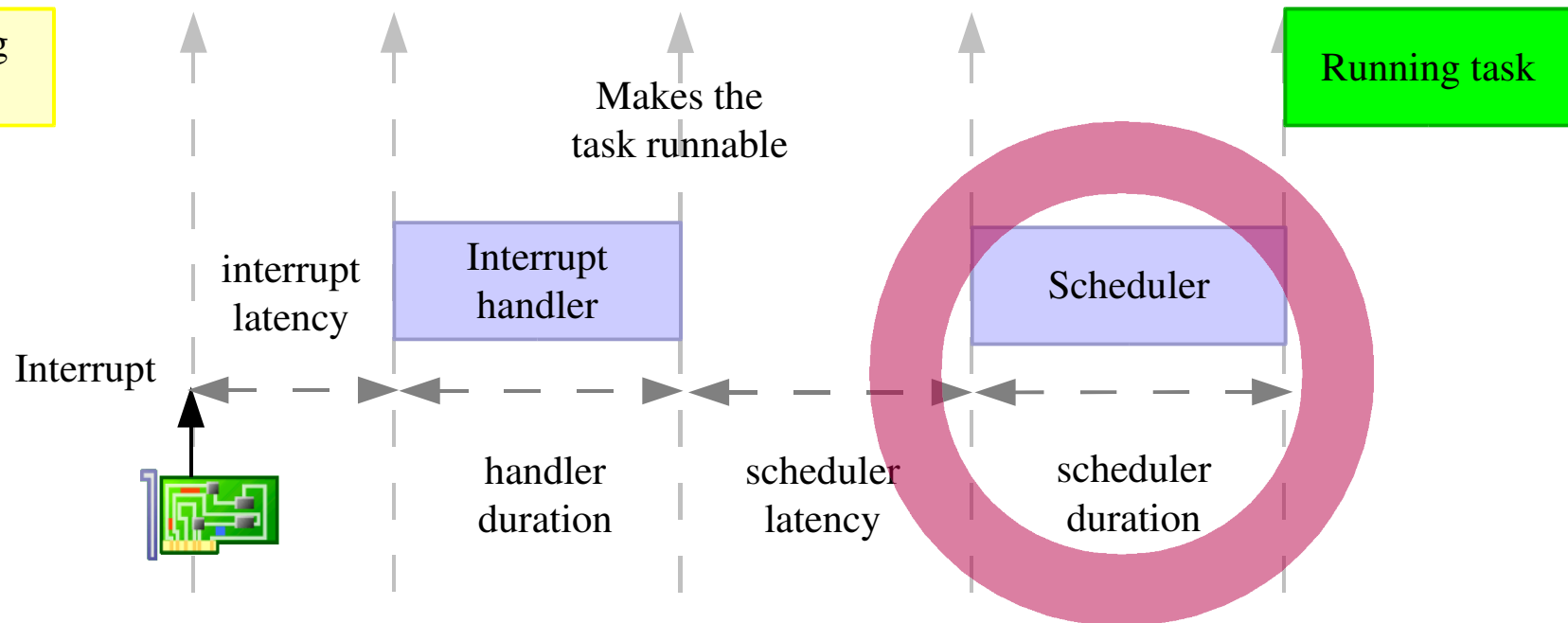


Back to our case

- ▶ We are returning from interrupt context. The scheduler is run immediately if the current task was running in userspace. Otherwise, we have to wait for the end of the current system call.



Scheduler duration



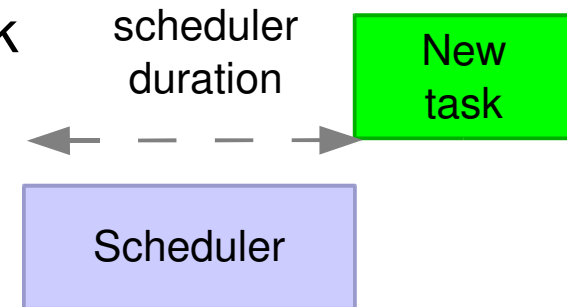
Time taken to execute the scheduler and switch to the new task.

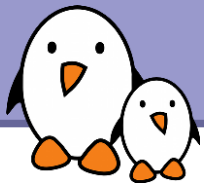


Sources of scheduler execution time

Time to execute the scheduler and switch to a new task

- ▶ Time to execute the scheduler: in Linux 2.4, depended on the number of running processes.
- ▶ Context switching time:
time to save the state of the current process (CPU registers)
and restore the new process to run.
This time is constant, so if not an issue for real-time.



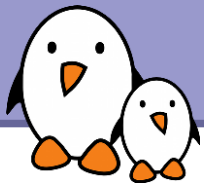


Other sources of latency (1)

Demand paging

- ▶ When it starts a process, the Linux kernel first sets up the virtual address space of the program (code, stack and data).
- ▶ However, to improve performance and reduce memory consumption, Linux loads the corresponding pages in RAM only when they are needed. When a part of the address space is accessed (like a particular part of the code) and is not in RAM yet, a page fault is raised by the MMU. This triggers the loading of the page in RAM.

This usually involves reading from disk: unexpected latencies!



Other sources of latency (2)

- ▶ Virtual memory in general
Swapping (always disabled in embedded systems), context switching... handling any MMU event.
- ▶ Memory allocation
The Linux allocator is fast,
but does not guarantee a maximum allocation time.



Other sources of latency (3)

- ▶ System calls

Similarly, no guaranteed response time.

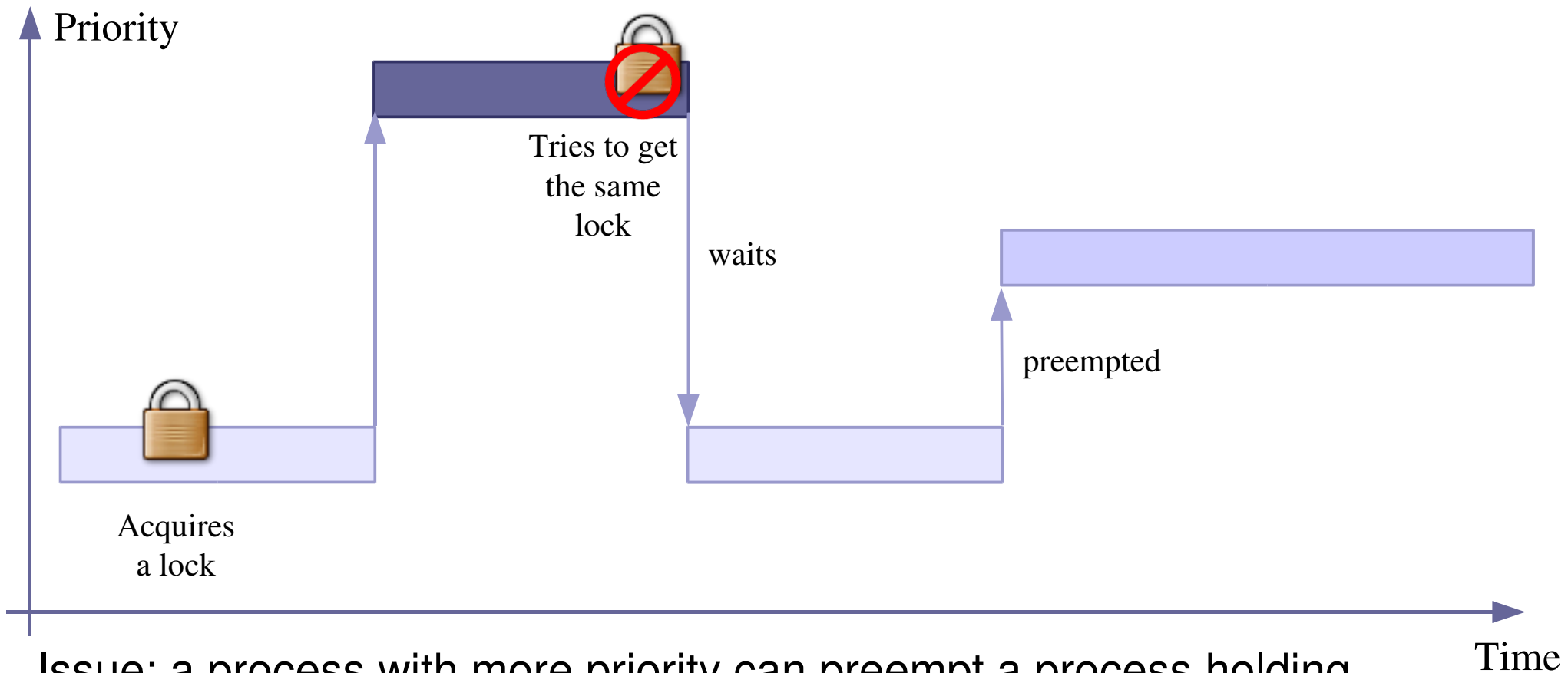
- ▶ Drivers

System calls handlers or interrupt handlers usually not implemented with care for real-time requirements.

Expect uncertainties unless you implemented all kernel support code by yourself.



Issue: priority inversion



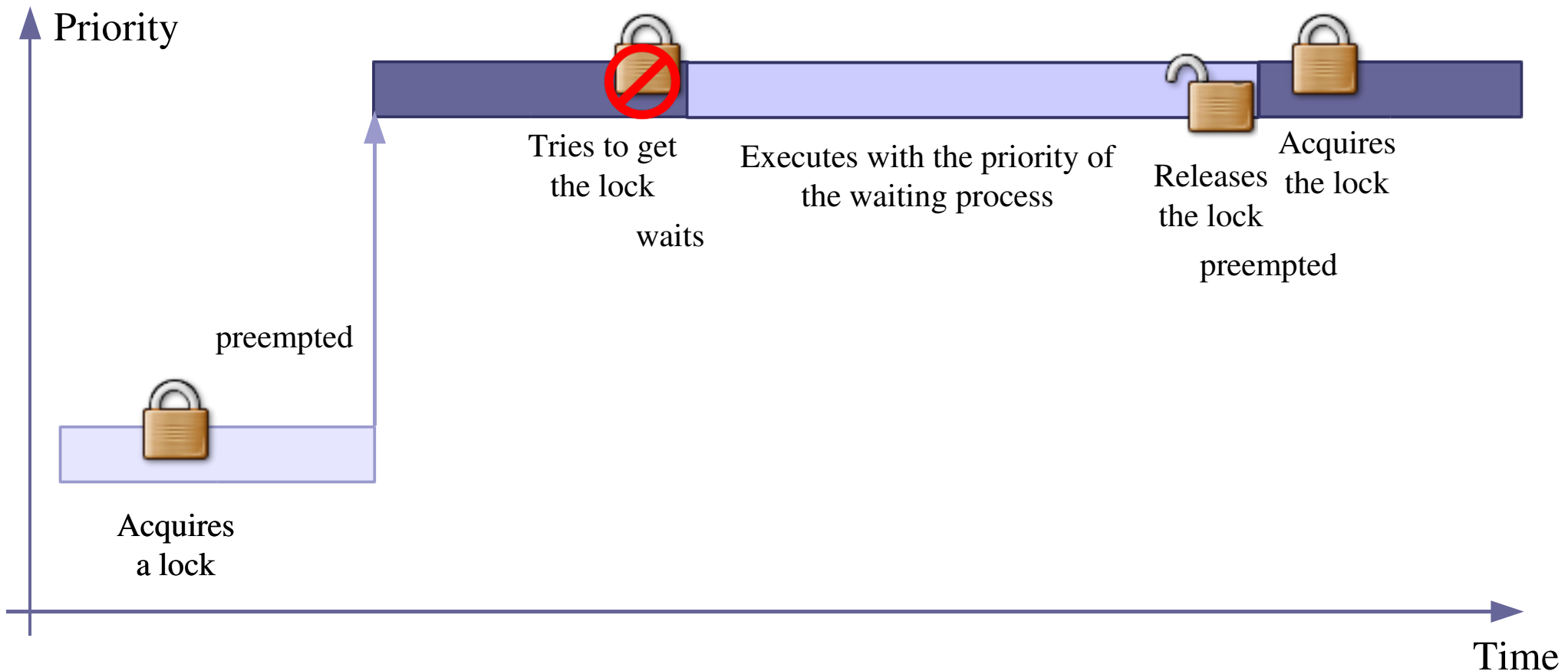
Issue: a process with more priority can preempt a process holding the lock.

The top priority process could wait for a very long time.

This happened with standard Linux before version 2.6.18.



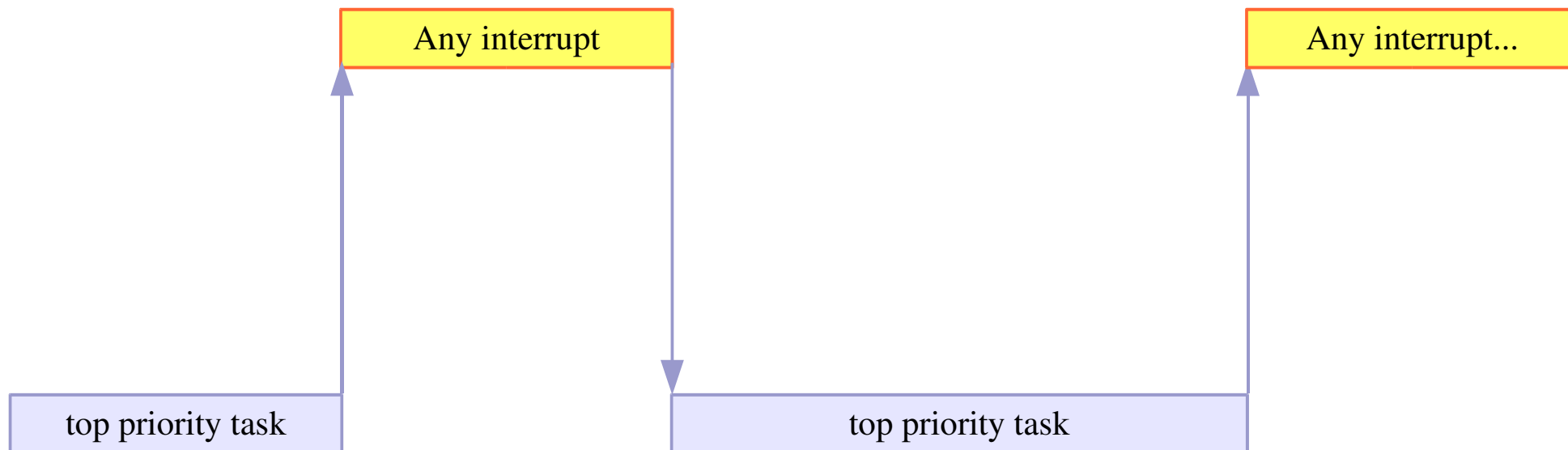
Solution: priority inheritance



Solution: *priority inheritance*. The process holding the lock inherits the priority of the process waiting for the lock with the greatest priority.



Issue: interrupt inversion



Issue: even your top priority task can be “preempted” by any interrupt handler, even for interrupts feeding tasks with lower priority.
Solution: threaded interrupts (scheduled for later execution).



Benchmarking tools

- ▶ Interbench: <http://users.on.net/~ckolivas/interbench/>
Interactivity benchmark. Measures the system latency under various simulated load conditions. Supports real-time tasks.
- ▶ Netperf: <http://www.netperf.org/netperf/NetperfPage.html>
Measures networking performance.
- ▶ Hackbench: <http://developer.osdl.org/craiger/hackbench/>
Tests the performance, overhead and scalability of the Linux scheduler.
- ▶ dbench: <http://samba.org/ftp/tridge/dbench/README>
Produces filesystem load.
- ▶ stress: <http://weather.ou.edu/~apw/projects/stress/>
System load generator (CPU, memory, I/O, disk)
- ▶ GNU/Emacs: <http://www.gnu.org/software/emacs/> 🙄
Also produces system load.



Introduction to rt-preempt patches

Linux 2.6 features for real-time



Linux 2.6 improvements

Helping to reduce latency and non determinism

- ▶ Scheduler: now selects the next task to run in constant time, whatever the number of processes (“O(1) scheduler”).
- ▶ May be built with no virtual memory support (on supported platforms). Benefits: reduced context switching time (address space shared by all processes), better performance (CPU cache still valid after switching).
- ▶ Full POSIX real-time API support.
This is the standard Unix API to implement real-time applications.



New preemption options in Linux 2.6

2 new preemption models offered by standard Linux 2.6:

Preemption Model

- | | |
|--|-------------------|
| ○ No Forced Preemption (Server) | PREEMPT_NONE |
| ● Voluntary Kernel Preemption (Desktop) | PREEMPT_VOLUNTARY |
| ○ Preemptible Kernel (Low-Latency Desktop) | PREEMPT |



1st option: no forced preemption

`CONFIG_PREEMPT_NONE`

Kernel code (system calls) never preempted.

Default behavior in standard kernels.

- ▶ Best for systems making intense computations, on which overall throughput is key.
- ▶ Best to reduce task switching to maximize CPU and cache usage (by reducing context switching).
- ▶ Still benefits from some Linux 2.6 improvements:
O(1) scheduler, increased multiprocessor safety (work on RT preemption was useful to identify hard to find SMP bugs).
- ▶ Can also benefit from a lower timer frequency (100 Hz instead of 250 or 1000).



2nd option: voluntary kernel preemption

`CONFIG_PREEMPT_VOLUNTARY`

Kernel code can preempt itself

- ▶ Typically for desktop systems, for quicker application reaction to user input.
- ▶ Adds explicit rescheduling points throughout kernel code.
- ▶ Minor impact on throughput.



3rd option: preemptible kernel

CONFIG_PREEMPT

Most kernel code can be involuntarily preempted at any time. When a process becomes runnable, no more need to wait for kernel code (typically a system call) to return before running the scheduler.

- ▶ Exception: kernel critical sections (holding spinlocks).
- ▶ Typically for desktop or embedded systems with latency requirements in the milliseconds range.
- ▶ Still a relatively minor impact on throughput.



Priority inheritance support

Available since Linux 2.6.18

- ▶ Implemented in kernel space locking

- ▶ Available in user space locking too, through fast user-space mutexes (“futex”).

Priority inheritance needs to be explicated for each mutex though.

Supported in glibc since version 2.5 (September 2006).

Apparently not supported yet in the last uClibc version (as of May 2009).

- ▶ See the kernel documentation for details:

[Documentation/pi-futex.txt](#)

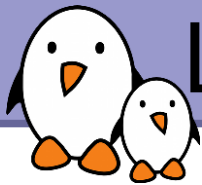
[Documentation/rt-mutex-design.txt](#)



High-resolution timers

- ▶ Main contributors: Thomas Gleixner, Ingo Molnar
- ▶ Make it possible for POSIX timers and `nanosleep()` to be as accurate as allowed by the hardware (typically 1 us).
- ▶ Together with tickless support, allow for “on-demand” timer interrupts.

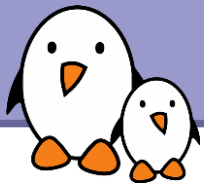
Available in the Vanilla kernel since 2.6.21.



Linux 2.6 improvements - What to remember

- ▶ Bounded scheduler execution time
- ▶ Increased determinism with improved POSIX RT API support.
- ▶ By default, system calls still not preemptible.
- ▶ New `CONFIG_PREEMPT` option making most of the kernel code preemptible.

However, even `CONFIG_PREEMPT` was not enough for audio users (need guaranteed latency in the milliseconds range), and of course for real-time users with stronger requirements.



Introduction to rt-preempt patches

Linux rt-preempt patches



Linux real-time preemption patches

<http://www.kernel.org/pub/linux/kernel/projects/rt/>

- ▶ Main contributors: Ingo Molnar and Steven Rostedt ([Red Hat](#)), Thomas Gleixner ([linutronix.de](#))
- ▶ Available for the most recent Linux kernel releases. Supports all the major embedded architectures.
- ▶ Configurable in the [Processor type and features \(x86\)](#), [Kernel Features \(arm\)](#) or [Platform options \(ppc\)](#)...

Preemption Mode	
<input type="radio"/> No Forced Preemption (Server)	PREEMPT_NONE
<input type="radio"/> Voluntary Kernel Preemption (Desktop)	PREEMPT_VOLUNTARY
<input type="radio"/> Preemptible Kernel (Low-Latency Desktop)	PREEMPT_DESKTOP
<input checked="" type="radio"/> Complete Preemption (Real-Time)	PREEMPT_RT
Thread Softirqs	PREEMPT_SOFTIRQS
Thread Hardirqs	PREEMPT_HARDIRQS



New option: complete real-time preemption

Makes all but the most critical kernel code involuntarily preemptible.

- ▶ Typically for desktop or embedded systems with maximum latency requirements within the 100 μ s range.
- ▶ Replaces almost every kernel spinlock by preemptible mutexes with priority inheritance.
- ▶ Since spinlocks can sleep, need to allow soft and hard IRQs to sleep (now executed in thread context).
- ▶ Of course, impact on global performance (more time managing tasks and switching between them).



Soft and hard IRQs run in thread context

- ▶ Can be configured separately, though always enabled with complete real-time preemption.
- ▶ Reduces latency by running soft IRQs and selected or all hard interrupts in a dedicated kernel thread (managed in process context by the regular Linux scheduler).
- ▶ Thanks to this, real-time tasks can have priority over some or all IRQs. Previously, even soft IRQs were run before any task managed by the scheduler.



Current issues and limitations

- ▶ Linux RT patches not mainstream yet.
Facing resistance to retain the general purpose nature of Linux.
Ingo, Thomas and the community are patiently closing the gap.
- ▶ Though they are preemptible, kernel services (such as memory allocation) do not have a guaranteed latency yet! However, not really a problem: real-time applications should allocate all their resources ahead of time.
- ▶ Kernel drivers are not developed for real-time constraints yet (think about the USB or PCI bus stacks, for example).
- ▶ Binary-only drivers have to be recompiled for RT preempt (how?)



Recent developments

- ▶ No RT-preempt patch releases for 2.6.27!
- ▶ Development is going on in a git tree:
[git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/linux-2.6-rt.git](http://git.kernel.org/pub/scm/linux/kernel/git/rostedt/linux-2.6-rt.git)
- ▶ They are working on new locks, which could be either non-preemptible or preemptible according to a boot option. Spinlocks would remain non-preemptible.
- ▶ This solution should at last be clean enough to be accepted in the mainline kernel! You would no longer need a separate kernel image for real-time needs (feature enabled at boot time).

See details on <http://lwn.net/Articles/310391/>



Wrong ideas about real-time preemption

- ▶ *It will improve throughput and overall performance*

Wrong: it will degrade overall performance.

- ▶ *It will reduce latency*

Often wrong. The maximum latency will be reduced.

The primary goal is to make the system predictable and deterministic.



Distributions

▶ Red Hat MRG Real Time

RPM packages for Red Hat Enterprise Linux, bringing consistent, predictable response times for high priority apps. Based on the real-time preempt kernel.

<http://www.redhat.com/mrg/realtime/>

▶ SUSE Linux Enterprise Real Time 10

A distribution using the realtime kernel, targeting running applications needing maximum predictability

<http://www.novell.com/products/realtime/>

▶ Ubuntu

Just install the `linux-rt` package!



Useful reading

About real-time support in the standard Linux kernel

- ▶ Internals of the RT Patch, Steven Rostedt, Red Hat, June 2007
<http://www.kernel.org/doc/ols/2007/ols2007v2-pages-161-172.pdf>
Definitely worth reading.
- ▶ The Real-Time Linux Wiki: <http://rt.wiki.kernel.org>
“The Wiki Web for the `CONFIG_PREEMPT_RT` community,
and real-time Linux in general.”
Contains nice and useful documents!
- ▶ See also our books page.



Introduction to rt-preempt patches

Debugging the real-time kernel



ftrace - Kernel function tracer

New infrastructure that can be used for debugging or analyzing latencies and performance issues in the kernel.

- ▶ Developed by Steven Rostedt. Merged in 2.6.27.
For earlier kernels, can be found from the rt-preempt patches.
- ▶ Very well documented in `Documentation/ftrace.txt`
- ▶ For the first time in kernel history,
it was documented (in 2.6.26) even before it was included (2.6.27)!
- ▶ Negligible overhead when tracing is not enabled at run-time.
- ▶ Can be used to trace any kernel function!
- ▶ See our video of Steven's tutorial at OLS 2008:
<http://free-electrons.com/community/videos/conferences/>



Using ftrace

- ▶ Tracing information available through the debugfs virtual fs (`CONFIG_DEBUG_FS` in the `Kernel Hacking` section)
- ▶ Mount this filesystem as follows:
`mount -t debugfs nodev /debug`
- ▶ When tracing is enabled (see the next slides), tracing information is available in `/debug/tracing`.
- ▶ Check available tracers
in `/debug/tracing/available_tracers`



Scheduling latency tracer (1)

`CONFIG_SCHED_TRACER` (Kernel Hacking section)

- ▶ Maximum recorded time between waking up a top priority task and its scheduling on a CPU, expressed in μs .
- ▶ Check that `wakeup` is listed in `/debug/tracing/available_tracers`
- ▶ To select, reset and enable this tracer:

```
echo wakeup > /debug/tracing/current_tracer  
echo 0 > /debug/tracing/tracing_max_latency  
echo 1 > /debug/tracing/tracing_enabled
```
- ▶ Let your system run, in particular real-time tasks.
Example: `chrt -f 5 sleep 1`
- ▶ Disable tracing:

```
echo 0 > /debug/tracing/tracing_enabled
```
- ▶ Read the maximum recorded latency:

```
cat /debug/tracing/tracing_max_latency
```



Scheduling latency tracer (2)

```
> cat /debug/tracing/latency_trace
```

```
wakeup latency trace v1.1.5 on 2.6.26-rc8
```

```
-----  
latency: 4 us, #2/2, CPU#1 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:2)
```

```
-----  
| task: sleep-4901 (uid:0 nice:0 policy:1 rt_prio:5)  
-----
```

```
#          -----=> CPU#  
#          /-----=> irqs-off  
#          | /-----=> need-resched  
#          || /-----=> hardirq/softirq  
#          ||| /-----=> preempt-depth  
#          |||| /  
#          |||||  
#          ||||| delay  
#  cmd      pid  time  caller  
#  \      /  
#  \      /  
<idle>-0  1d.h4    0us+: try_to_wake_up (wake_up_process)  
<idle>-0  1d..4    4us : schedule (cpu_idle)
```




Other tracers

- ▶ Also available through the same tracing interface. Enable them in the `Kernel Hacking` section.
- ▶ `irqsoff`: measures time spent in sections with interrupts off, and records the maximum value.
- ▶ `preemptoff`: time spent in critical sections with preemption disabled.
- ▶ `preemptirqsoff`: time when either interrupts or preemption are disabled.
- ▶ `sched_switch`: traces the context switches between tasks.
- ▶ `none`: just empties the list of active tracers.

Several tracers have been added: kernel stack size tracer, boot tracer, and this will continue.



Your real-time Linux checklist

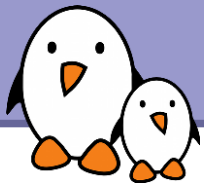
Trying to meet your real-time requirements...

- ▶ First, make sure your critical tasks are run as real-time tasks. This will guarantee that regular processes will never preempt them.
- ▶ Second, try to use a standard kernel with `CONFIG_PREEMPT`.
- ▶ If this is not enough, patch your kernel with the real-time preempt patches and configure it with `CONFIG_PREEMPT_RT`.
- ▶ Also try to analyze and fix latency issues using the latency tracing options.



Real Time in Embedded Linux Systems

Linux hard real-time extensions



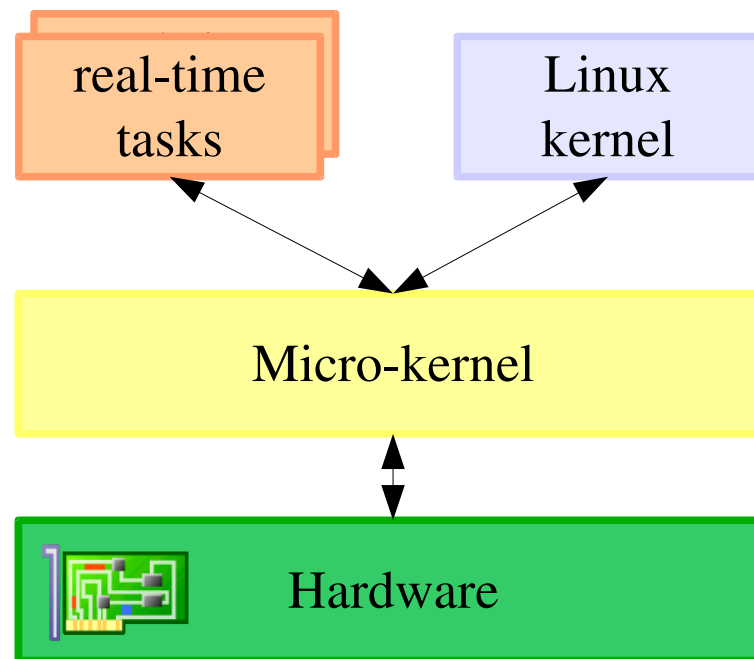
Linux hard real-time extensions

Three generations

- ▶ RTLinux
- ▶ RTAI
- ▶ Xenomai

A common principle

- ▶ Add a extra layer between the hardware and the Linux kernel, to manage real-time tasks separately.





RTLinux

First real-time extension for Linux, created by Victor Yodaiken.

- ▶ Nice, but the author filed a software patent covering the addition of real-time support to general operating systems as implemented in RTLinux!
- ▶ Its Open Patent License drew many developers away and frightened users. Community projects like RTAI and Xenomai now attract most developers and users.
- ▶ February, 2007: RTLinux rights sold to Wind River.
Now supported by Wind River as “Real-Time Core for Wind River Linux.”
See our [Wind River](#) page.
- ▶ Free version still advertised by Wind River on <http://www.rtlinuxfree.com>, but no longer a community project.



<http://www.rtai.org/> - Real-Time Application Interface for Linux

- ▶ Created in 1999, by Prof. Paolo Montegazza (long time contributor to RTLinux), Dipartimento di Ingegneria Aerospaziale Politecnico di Milano (DIAPM).
- ▶ Community project. Significant user base. Attracted contributors frustrated by the RTLinux legal issues.
- ▶ However, only actively maintained on x86.
- ▶ Lack of stability and industrial maturity. Development driven by the immediate needs of its maintainer. Not enough care for changes that can break platforms used by other people.



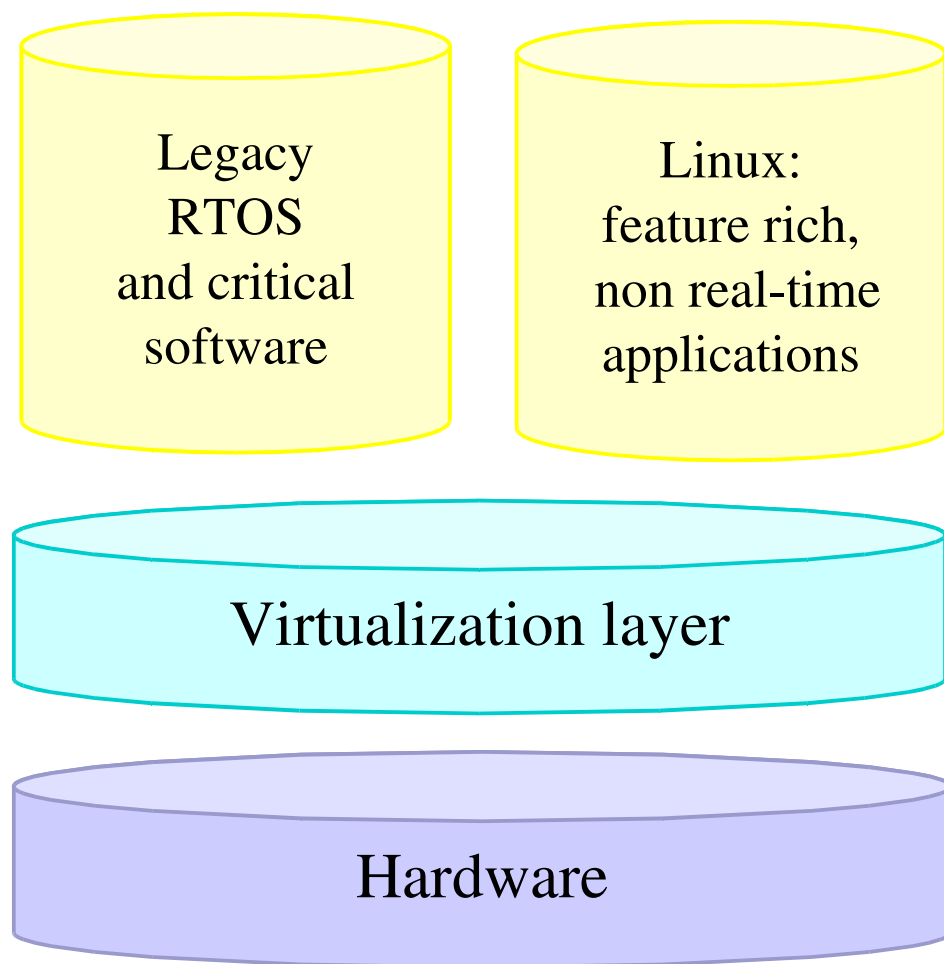
Adeos nanokernel

<http://home.gna.org/adeos/>

- ▶ Flexible environment to share hardware resources between several operating systems (or several instances of the same operating system).
- ▶ May 2007: supports `i386`, `x86_64`, `ia64`, `arm`, `ppc`, `ppc64` and `blackfin` (`mips` not yet).
- ▶ Originally created for RTAI as a replacement Hardware Abstraction Layer beneath the Linux kernel. Not impacted by the RTLinux patent!
- ▶ Also used for SMP clustering, patchless kernel debugging...



Virtualization



Cheap and safe transition to Linux!

- ▶ Makes it possible to switch to Linux for feature-rich applications.
- ▶ Allows to keep legacy RTOS and software for realtime tasks.
- ▶ Free solution: Adeos
<http://home.gna.org/adeos/>
- ▶ Proprietary solution: VirtualLogix (formerly Jaluna)
<http://www.virtuallogix.com>



Linux hard real-time extensions Xenomai



Xenomai project

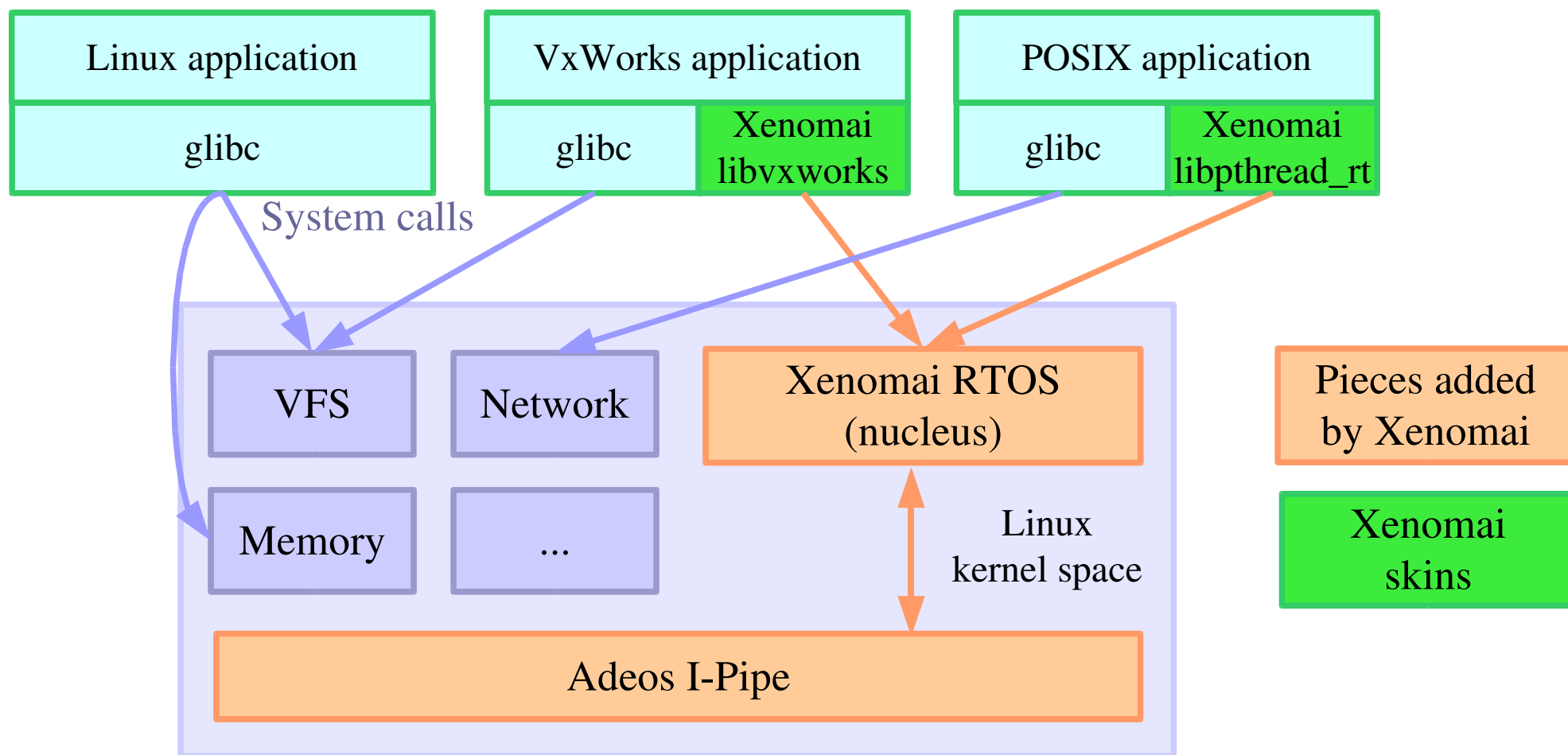


<http://www.xenomai.org/>

- ▶ Started in the RTAI project (called RTAI / fusion).
- ▶ Skins mimicking the APIs of traditional RTOS such as VxWorks, pSOS+, and VRTXsa.
- ▶ Initial goals: facilitate the porting of programs from traditional RTOS to RTAI on GNU / Linux.
- ▶ Now an independent project and an alternative to RTAI. Many contributors left RTAI for Xenomai, frustrated by its goals and development style.



Xenomai architecture





Real-Time Driver Model

- ▶ An approach to unify the interfaces for developing device drivers and associated applications under real-time Linux.
- ▶ Currently available for Xenomai and RTAI
RTDM-native: a port over native Linux with the real-time preemption patch.
- ▶ See the whitepaper on
<http://www.xenomai.org/documentation/branches/v2.3.x/pdf/RTDM-and-Applications.pdf>



Most recent kernel versions supported

From [Xenomai 2.4.7](#) sources (Feb. 2009 status)
(according to the patches in [ksrc/arch/<arch>/patches/](#))

- ▶ [x86](#): 2.6.28 (latest release at that time)
- ▶ [arm](#) (several platforms supported): 2.6.28 (latest release)
- ▶ [ppc](#): 2.6.28 (latest release)
- ▶ [blackfin](#) (BF533 and BF537 boards): 2.6.28 (latest release)
- ▶ [ia64](#): 2.6.16 (Mar. 2006!)



Kernel versions supported by Xenomai

Remarks

- ▶ Xenomai patches are just Adeos patches for the Linux kernel.
- ▶ Everything else is standard Linux.
- ▶ Xenomai supports recent Linux versions, showing that it is actively maintained.
- ▶ Xenomai also supports older Linux versions: 2.4.



How to use Xenomai

- ▶ Download the latest Xenomai release
- ▶ Download one of the Linux versions supported by this release (see [ksrc/arch/<arch>/patches/](#))
- ▶ From the Xenomai source directory, run the following command to patch the Linux sources:
[scripts/prepare-kernel.sh](#)
(Tell it your kernel source directory and your architecture)
- ▶ Compile Linux, and boot this modified kernel
- ▶ Compile and install the Xenomai libraries
- ▶ You are now ready to compile your own programs.
You can reuse the Makefiles in Xenomai's [examples](#) directory.

Details on http://xenomai.org/index.php/Xenomai_quick_build_guide



Real-Time in Embedded Linux Systems

Commercial real-time Linux solutions



http://www.mvista.com/real_time_linux.php

- ▶ Keeps the standard Linux API
- ▶ Support the latest RT patches from the community for the mainstream Linux kernel, in their Linux Professional Edition 5.0. Technical details on <http://mvista.com/news/2007/profed5.html>
- ▶ (Proprietary) tools to analyze performance and track down latency sources.



Wind River

<http://windriver.com/products/linux/>

WIND RIVER

- ▶ A lot of real-time experience from the **VxWorks** system.
- ▶ Wind River Linux 3.0 supports Linux 2.6.27 with the real-time preempt patches.
- ▶ Main offering: Real Time Core for Wind River Linux
 - ▶ Originating from the RTLinux core.
 - ▶ Design similar to that of RTAI: micro-kernel, RT tasks implemented as kernel modules.
 - ▶ Can achieve single digit microsecond latency.
 - ▶ See <http://free-electrons.com/redirect/windriver-rtcore.html>



<http://timesys.com>



- ▶ Release (and support!) the latest Linux RT patches through their [LinuxLink](#) platform.
(Example: http://www.timesys.com/news/press_releases/atmel_realtime)
- ▶ Also contribute to their development (contracting Thomas Gleixner).
- ▶ They share a few nice documents on:
https://linuxlink.timesys.com/docs/real_time
- ▶ But very few details on their website about their product features (except perhaps on their webinars).



BlueCat Linux

LynuxWorks BlueCat:

<http://www.lynuxworks.com/products/bluecat/>



- ▶ Traditional RTOS vendor trying to outlive the Linux tidal wave.
- ▶ Based on early Linux 2.6 (years old) with standard soft real-time improvements! Not updated at all with real-time preempt patches.
- ▶ Apparently, Linux is just a way to attract customers to LynxOS: easily switching when their real-time requirements are no longer met (compatibility with Linux applications and same development tools).
- ▶ No visible contribution to community projects.



Other offerings

► Koan Software

<http://koansoftware.com/kaeilos/en/realtime.htm>

Hard-real time support with **Xenomai**, **RTAI**,
or real-time preemption patches in their **KaeilOS** product.

Based on Linux 2.6.20

(see <http://koansoftware.com/kaeilos/en/features.htm>)



► SysGo: <http://sysgo.com>

Hard-real time support with their own microkernel (**PikeOS**)
managing the regular Linux kernel and real-time POSIX threads.

See <http://www.sysgo.com/products/elinos-family/elinos-real-time/>
for details (approach similar to **RTAI**).

Based on a pretty old kernel version (2.6.15)



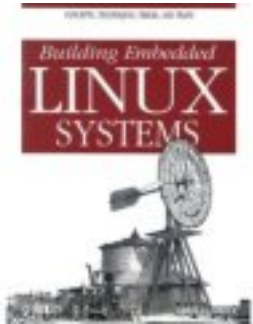
Conclusion

Linux was not designed as a real-time OS

- ▶ However, you can make your system **hard real-time** by using one of the hard real-time extensions (RTLinux, RTAI, Xenomai), and writing your critical applications and drivers with the corresponding APIs.
- ▶ You can also get **hard real-time** with a regular Linux kernel with the rt-preempt patches, once all the Linux kernel timing bugs are fixed on your architecture / hardware and **all** the latencies are within the required specifications.
- ▶ You can get **soft real-time** with the standard kernel preemption mode. **Most** of the latencies will be reduced, offering better quality, but probably not all of them.
- ▶ Anyway, using **hard real-time extensions will not guarantee that your system will be hard real-time**. Your system and applications will also have to be designed properly (correct priorities, use of deterministic APIs, allocation of critical resources ahead of time...).



Books



- ▶ Building Embedded Linux Systems, O'Reilly
By Karim Yaghmour, Jon Masters,
Gilad Ben-Yossef, Philippe Gerum and others
(including Michael Opdenacker), August 2008



A nice coverage of Xenomai (Philippe Gerum)
and the RT patch (Steven Rostedt)

<http://oreilly.com/catalog/9780596529680/>



Using the POSIX API

Using the POSIX API
Threads, real-time and IPC



Real-time processes

Real-time processes can be started by `root` using the POSIX API

- ▶ 100 real-time priorities available
- ▶ `SCHED_FIFO` scheduling class:
The process runs until completion unless it is blocked by an I/O, voluntarily relinquishes the CPU, or is preempted by a higher priority process.
- ▶ `SCHED_RR` scheduling class:
Difference: the processes are scheduled in a Round Robin way. Each process is run until it exhausts a max time quantum. Then other processes with the same priority are run, and so and so...
- ▶ `SCHED_OTHER` scheduling class:
Used for regular processes with non-real-time priority.



Differences with standard processes

- ▶ Processes with real-time priority are always run before processes with lower priority (real-time or not).
- ▶ Real-time processes do not expire (except `SCHED_RR` ones for processes with the same priority).
- ▶ Hence, real-time processes which never yield the processor can completely starve processes with lower priority (doesn't happen with standard processes).



Managing real-time tasks

Available through `<sched.h>` (see `man sched.h` for details)

- ▶ `sched_getscheduler, sched_setscheduler`
Get / set the scheduling class of a process
- ▶ `sched_getparam, sched_setparam`
Get / set the priority of a process
- ▶ `sched_get_priority_max, sched_get_priority_min`
Get the maximum / minimum priorities allowed for a scheduling class.
- ▶ `sched_rr_get_interval`
Get the current timeslice of the `SCHED_RR` process
- ▶ `sched_yield`
Yield execution to another process.



chrt

Command line utility to start real-time processes and manipulate the real-time attributes of a process.

- ▶ **Ubuntu, Debian**: available in the `util-linux` package.
- ▶ Embedded systems: implemented in **BusyBox**
- ▶ Original version: <ftp://ftp.kernel.org/pub/linux/utils/util-linux/>
- ▶ Example: start a RT_FIFO task with top priority:
`chrt -f 99 ./rttest`



Locking pages in RAM

A solution to latency issues with demand paging!

▶ Available through `<sys/mman.h>` (see `man mman.h`)

▶ `mlock`

Lock a given region of process address space in RAM.
Makes sure that this region is always loaded in RAM.

▶ `mlockall`

Lock the whole process address space.

▶ `munlock`, `munlockall`

Unlock a part or all of the process address space.



POSIX clocks and timers

Compared to standard (BSD) timers in Linux

- ▶ Possibility to have more than 1 timer per process.
- ▶ Increased precision, up to nanosecond accuracy
- ▶ Timer expiration can be notified either with a signal or with a thread.
- ▶ Several clocks available.



Available POSIX clocks (1)

Defined in `/usr/include/linux/time.h`

▶ `CLOCK_REALTIME`

System-wide clock measuring the time in seconds and nanoseconds since Jan 1, 1970, 00:00. Can be modified.
Accuracy: 1/HZ (1 to 10 ms)

▶ `CLOCK_MONOTONIC`

System-wide clock measuring the time in seconds and nanoseconds since system boot. Cannot be modified, so can be used for accurate time measurement.
Accuracy: 1/HZ



Available POSIX clocks (2)

- ▶ `CLOCK_PROCESS_CPUTIME_ID`

Measures process uptime. 1/HZ accuracy. Can be changed.

- ▶ `CLOCK_THREAD_CPUTIME_ID`

Same, but only for the current thread.



Time management

Functions defined in `time.h`

- ▶ `clock_gettime`
Set the specified clock to a value
- ▶ `clock_gettime`
Read the value of a given clock
- ▶ `clock_getres`
Get the resolution of a given clock.

See `man time.h` and the manual of each of these functions.



Using timers (1)

Functions also defined in `time.h`

- ▶ `clock_nanosleep`

Suspend the current thread for the specified time, using a specified clock.

- ▶ `nanosleep`

Same as `clock_nanosleep`, using the `CLOCK_REALTIME` clock.



Using timers (2)

- ▶ `timer_create`

Create a timer based on a given clock.

- ▶ `timer_delete`

Delete a timer

- ▶ `timer_settime`

Arm a timer.

- ▶ `timer_gettime`

Access the current value of a timer.



Using high resolution timers

- ▶ Available in Linux since 2.6.21 (on `x86`).
Now available on most supported platforms.
- ▶ Depending on the hardware capabilities, this feature gives microsecond or nanosecond accuracy to the regular clocks (`CLOCK_REALTIME`, `CLOCK_MONOTONIC`).
- ▶ No need to recompile your applications!



Compiling instructions

- ▶ Includes: nothing special to do.
Available in the standard path.
- ▶ Libraries: link with `librt`
- ▶ Example:
`gcc -lrt -o rttest rttest.c`



POSIX manual pages

POSIX manual pages may not be installed on your system

- ▶ On Debian Linux, based systems,
to find the names of the corresponding packages:

```
apt-cache search posix
```

Then, install these packages as follows:

```
apt-get install manpages-posix manpages-posix-dev
```

- ▶ Other distributions should have similar package names.
- ▶ These manual pages are also available on-line:
<http://www.opengroup.org/onlinepubs/009695399/idx/realtime.html>

You can almost consider these manual pages as specifications.

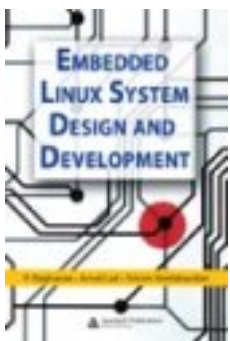
The standard can also be accessed on

<http://www.unix.org/online.html> (registration required).



More information on the POSIX interface

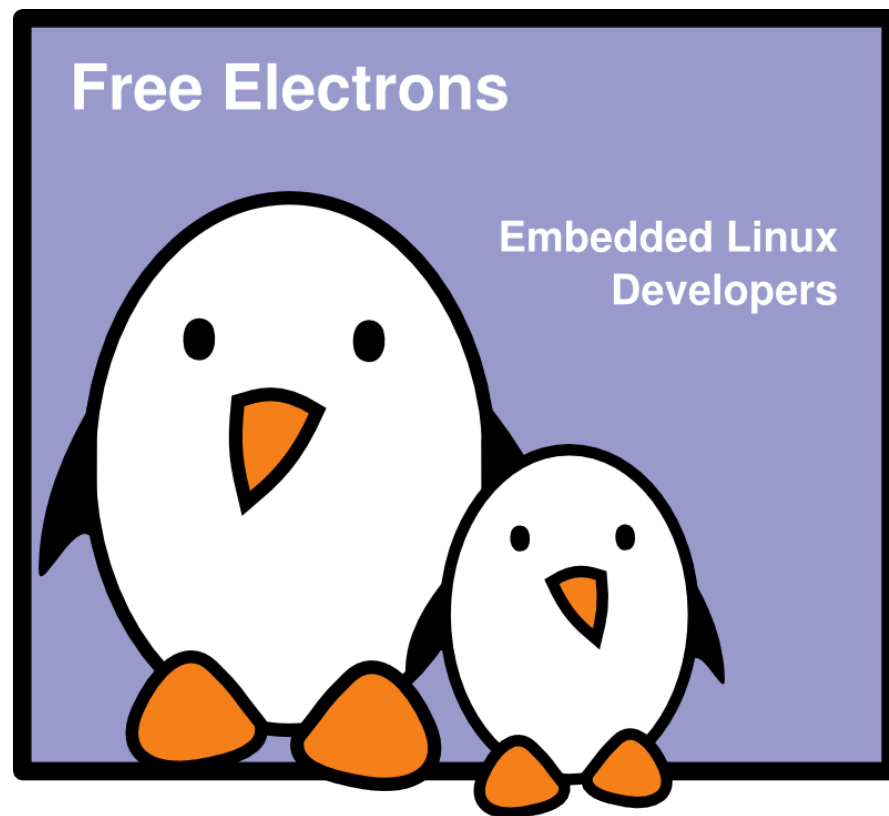
- ▶ The POSIX manual pages
- ▶ Embedded Linux System Design and Development
P. Raghavan, A. Lad, S. Neelakandan, Auerbach, Dec. 2005.
<http://free-electrons.com/redirect/elsdd-book.html>
Very nice and clear coverage on real-time programming with the POSIX interface. Nice and useful examples.
- ▶ Guide to real-time programming
<http://www.phys.uu.nl/DU/unix/HTML/APS33DTE/TITLE.HTM>
A 11-year old document, with some Digital Unix specifics, but still up to date (thanks to standards).





Hotplugging with udev

Michael Opdenacker
Free Electrons





/dev issues and limitations

- ▶ On Red Hat 9, 18000 entries in `/dev`!
All entries for all possible devices
had to be created at system installation.
- ▶ Needed an authority to assign major numbers
<http://lanana.org/>: Linux Assigned Names and Numbers
Authority
- ▶ Not enough numbers in 2.4, limits extended in 2.6.
- ▶ Userspace neither knew what devices were present in the
system, nor which real device corresponded to each `/dev`
entry.



devfs solution and limitations

devfs: a first solution implemented in Linux 2.3.

- ▶ Only showed present devices
- ▶ But used different names as in `/dev`, causing issues in scripts.
- ▶ But no flexibility in device names, unlike with `/dev/`, e.g. the 1st IDE disk device had to be called either `/dev/hda` or `/dev/ide/hd/c0b0t0u0`.
- ▶ But didn't allow dynamic major and minor number allocation.
- ▶ But required to store the device naming policy in kernel memory. Kept forever in kernel RAM even when no longer needed.

devfs was completely removed in Linux 2.6.18.



The udev solution

Takes advantage of `sysfs` introduced by Linux 2.6.

- ▶ Created by Greg Kroah Hartman, a huge contributor. Other key contributors: Kay Sievers, Dan Stekloff.
 - ▶ **Entirely** in user space.
 - ▶ Automatically creates / removes device entries in `/dev/` according to inserted / removed devices.
 - ▶ Major and minor device transmitted by the kernel.
 - ▶ Requires no change to driver code.
 - ▶ Fast: written in C
- Small size: `udev` version 108: 61 KB in Ubuntu 7.04



hotplug history

`udev` was first implemented through the hotplug infrastructure:

- ▶ Introduced in Linux 2.4. Pioneered by USB.
- ▶ Whenever a device was inserted or removed, the kernel executed `/sbin/hotplug` to notify user space programs.
- ▶ For each subsystem (USB, PCI...), `/sbin/hotplug` then ran scripts (*agents*) taking care of identifying the hardware and inserting/removing the right driver modules.
- ▶ Linux 2.6: much easier device identification thanks to `sysfs`.
- ▶ `udev` was one of the agents run by `/sbin/hotplug`.



udev issues with hotplug

- ▶ **sysfs** timing issues.
- ▶ Out of order execution of hotplug processes.
- ▶ Out of memory issues when too many processes are run in a very short time.

Eventually, **udev** took over several parts of the hotplug infrastructure and completely replaced it.



Starting udev (1)

- ▶ At the very beginning of user-space startup, mount the `/dev/` directory as a `tmpfs` filesystem: `sudo mount -t tmpfs udev /dev`
- ▶ `/dev/` is populated with static devices available in `/lib/udev/devices/` :

Ubuntu 6.10 example:

```
crw----- 1 root root    5, 1 2007-01-31 04:18 console
lrwxrwxrwx 1 root root    11 2007-01-31 04:18 core -> /proc/kcore
lrwxrwxrwx 1 root root    13 2007-01-31 04:18 fd -> /proc/self/fd
crw-r----- 1 root kmem    1, 2 2007-01-31 04:18 kmem
brw----- 1 root root    7, 0 2007-01-31 04:18 loop0
lrwxrwxrwx 1 root root    13 2007-01-31 04:18 MAKEDEV -> /sbin/MAKEDEV
drwxr-xr-x 2 root root  4096 2007-01-31 04:18 net
crw----- 1 root root    1, 3 2007-01-31 04:18 null
crw----- 1 root root 108, 0 2007-01-31 04:18 ppp
drwxr-xr-x 2 root root  4096 2006-10-16 14:39 pts
drwxr-xr-x 2 root root  4096 2006-10-16 14:39 shm
lrwxrwxrwx 1 root root    24 2007-01-31 04:18 sndstat -> /proc/asound/oss/sndstat
lrwxrwxrwx 1 root root    15 2007-01-31 04:18 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root    15 2007-01-31 04:18 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root    15 2007-01-31 04:18 stdout -> /proc/self/fd/1
```



Starting udev (2)

- ▶ The **udev** daemon is started.
It listens to *uevents* from the driver core, which are sent whenever devices are inserted or removed.
- ▶ The **udev** daemon reads and parses all the rules found in `/etc/udev/rules.d/` and keeps them in memory.
- ▶ Whenever rules are added, removed or modified, **udev** receives an *inotify* event and updates its ruleset in memory.
- ▶ When an event is received, **udev** starts a process to:
 - ▶ try to match the event against udev rules,
 - ▶ create / remove device files,
 - ▶ and run programs (to load / remove a driver, to notify user space...)

The ***inotify*** mechanism lets userspace programs subscribe to notifications of filesystem changes. Possibility to watch individual files or directories.



Event queue management

- ▶ **udev** takes care of processing events in the right order. This is useful to process events after the ones then depend on (example: partition events need the parent block device event processing to be complete, to access its information in the udev database).
- ▶ **udev** also limits the number of processes it starts. When the limit is exceeded, only events carrying the **TIMEOUT** key are immediately processed.
- ▶ The `/etc/.udev/queue/` directory represents currently running or queued events. It contains symbolic links to the corresponding sysfs devices. The directory is removed after removing the last link.
- ▶ Event processes which failed are represented by `/etc/.udev/failed/`. Symbolic links in this directory are removed when an event for the same device is successfully processed.



netlink sockets

Kernel netlink sockets are used to carry uevents. Advantages:

- ▶ They are asynchronous. Messages are queued. The receiver can choose to process messages at its best convenience.
- ▶ Other userspace - kernelspace communication means are synchronous: system calls, `ioctl`s, `/proc/` and `/sys`.
- ▶ System calls have to be compiled statically into the kernel. They cannot be added by module-based device drivers.
- ▶ Multicasting is available. Several applications can be notified.

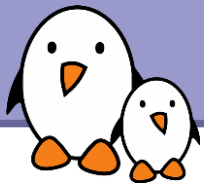
See <http://www.linuxjournal.com/article/7356>
for a very nice description of netlink sockets.



uevent message example

Example inserting a USB mouse

```
recv(4,                                     // socket id
     "add@/class/input/input9/mouse2\0    // message
     ACTION=add\0                          // action type
     DEVPATH=/class/input/input9/mouse2\0  // path in /sys
     SUBSYSTEM=input\0                    // subsystem (class)
     SEQNUM=1064\0                         // sequence number
     PHYSDEVPATH=/devices/pci0000:00/0000:00:1d.1/usb2/2-2/2-2:1.0\0
                                           // device path in /sys
     PHYSDEVBUS=usb\0                     // bus
     PHYSDEVDRIVER=usbhid\0               // driver
     MAJOR=13\0                           // major number
     MINOR=34\0",                          // minor number
     2048,                                // message buffer size
     0)                                    // flags
= 221                                     // actual message size
```



udev rules

When a udev rule matching event information is found, it can be used:

- ▶ To define the name and path of a device file.
- ▶ To define the owner, group and permissions of a device file.
- ▶ To execute a specified program.

Rule files are processed in lexical order.



udev naming capabilities

Device names can be defined

- ▶ from a label or serial number,
- ▶ from a bus device number,
- ▶ from a location on the bus topology,
- ▶ from a kernel name,
- ▶ from the output of a program.

See http://www.reactivated.net/writing_udev_rules.html for a very complete description. See also `man udev`.



udev naming rule examples

```
# Naming testing the output of a program
BUS=="scsi", PROGRAM="/sbin/scsi_id", RESULT=="OEM 0815", NAME="disk1"

# USB printer to be called lp_color
BUS=="usb", SYSFS{serial}=="W09090207101241330", NAME="lp_color"

# SCSI disk with a specific vendor and model number will be called boot
BUS=="scsi", SYSFS{vendor}=="IBM", SYSFS{model}=="ST336", NAME="boot%n"

# sound card with PCI bus id 00:0b.0 to be called dsp
BUS=="pci", ID=="00:0b.0", NAME="dsp"

# USB mouse at third port of the second hub to be called mouse1
BUS=="usb", PLACE=="2.3", NAME="mouse1"

# ttyUSB1 should always be called pda with two additional symlinks
KERNEL=="ttyUSB1", NAME="pda", SYMLINK="palmtop handheld"

# multiple USB webcams with symlinks to be called webcam0, webcam1, ...
BUS=="usb", SYSFS{model}=="XV3", NAME="video%n", SYMLINK="webcam%n"
```



udev permission rule examples

Excerpts from `/etc/udev/rules.d/40-permissions.rules`

```
# Block devices
SUBSYSTEM!="block", GOTO="block_end"
SYSFS{removable}!="1",                                GROUP="disk"
SYSFS{removable}=="1",                                  GROUP="floppy"
BUS=="usb",                                              GROUP="plugdev"
BUS=="ieee1394",                                         GROUP="plugdev"
LABEL="block_end"

# Other devices, by name

KERNEL=="null",                                         MODE="0666"
KERNEL=="zero",                                         MODE="0666"
KERNEL=="full",                                         MODE="0666"
```



Identifying device driver modules

Kernel / module compiling

Each driver announces which device and vendor ids it supports. Information stored in module files.



The `depmod -a` command processes module files and generates `/lib/modules/<version>/modules.alias`

System everyday life

The driver core (usb, pci...) reads the device id, vendor id and other device attributes.



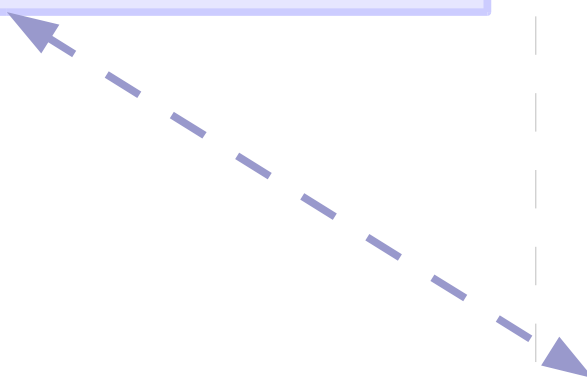
The kernel sends an event to `udev`, setting the `MODALIAS` environment variable, encoding these data.



A udev event process runs `modprobe $MODALIAS`



`modprobe` finds the module to load in the `modules.alias` file.





Module aliases

- ▶ **MODALIAS** environment variable example (USB mouse):

```
MODALIAS=usb:v046DpC03Ed2000dc00dsc00dp00ic03isc01ip02
```

- ▶ Matching line in `/lib/modules/<version>/modules.alias`:

```
alias usb:v*p*d*dc*dsc*dp*ic03isc01ip02* usbmouse
```




udev modprobe rule examples

Even module loading is done with **udev**!

Excerpts from `/etc/udev/rules.d/90-modprobe.rules`

```
ACTION!="add", GOTO="modprobe_end"
```

```
SUBSYSTEM!="ide", GOTO="ide_end"
```

```
IMPORT{program}="ide_media --export $devpath"
```

```
ENV{IDE_MEDIA}=="cdrom", RUN+="/sbin/modprobe -Qba ide-cd"
```

```
ENV{IDE_MEDIA}=="disk", RUN+="/sbin/modprobe -Qba ide-disk"
```

```
ENV{IDE_MEDIA}=="floppy", RUN+="/sbin/modprobe -Qba ide-floppy"
```

```
ENV{IDE_MEDIA}=="tape", RUN+="/sbin/modprobe -Qba ide-tape"
```

```
LABEL="ide_end"
```

```
SUBSYSTEM=="input", PROGRAM="/sbin/grepmap --udev", \  
    RUN+="/sbin/modprobe -Qba $result"
```

```
# Load drivers that match kernel-supplied alias
```

```
ENV{MODALIAS}=="?*", RUN+="/sbin/modprobe -Q $env{MODALIAS}"
```



Coldplugging

- ▶ Issue: losing all device events happening during kernel initialization, because udev is not ready yet.
- ▶ Solution: after starting `udev`, have the kernel emit uevents for all devices present in `/sys`.
- ▶ This can be done by the `udevtrigger` utility.
- ▶ Strong benefit: completely transparent for userspace. Legacy and removable devices handled and named in exactly the same way.



Debugging events - udevmonitor (1)

udevmonitor visualizes the driver core events and the **udev** event processes.
Example event sequence connecting a USB mouse:

```
UEVENT[1170452995.094476] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2
UEVENT[1170452995.094569] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UEVENT[1170452995.098337] add@/class/input/input28
UEVENT[1170452995.098618] add@/class/input/input28/mouse2
UEVENT[1170452995.098868] add@/class/input/input28/event4
UEVENT[1170452995.099110] add@/class/input/input28/ts2
UEVENT[1170452995.099353] add@/class/usb_device/usbdev4.30
UDEV [1170452995.165185] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2
UDEV [1170452995.274128] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UDEV [1170452995.375726] add@/class/usb_device/usbdev4.30
UDEV [1170452995.415638] add@/class/input/input28
UDEV [1170452995.504164] add@/class/input/input28/mouse2
UDEV [1170452995.525087] add@/class/input/input28/event4
UDEV [1170452995.568758] add@/class/input/input28/ts2
```

It gives time information measured in microseconds.

You can measure time elapsed between the uevent (**UEVENT** line), and the completion of the corresponding **udev** process (matching **UDEV** line).



Debugging events - udevmonitor (2)

udevmonitor --env shows the complete event environment for each line.

```
UDEV [1170453642.595297] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UDEV_LOG=3
ACTION=add
DEVPATH=/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
SUBSYSTEM=usb
SEQNUM=3417
PHYSDEVBUS=usb
DEVICE=/proc/bus/usb/004/031
PRODUCT=46d/c03d/2000
TYPE=0/0/0
INTERFACE=3/1/2
MODALIAS=usb:v046DpC03Dd2000dc00dsc00dp00ic03isc01ip02
UDEVD_EVENT=1
```



Misc udev utilities

- ▶ `udevinfo`

Lets users query the `udev` database.

- ▶ `udevtest <sysfs_device_path>`

Simulates a `udev` run to test the configured rules.



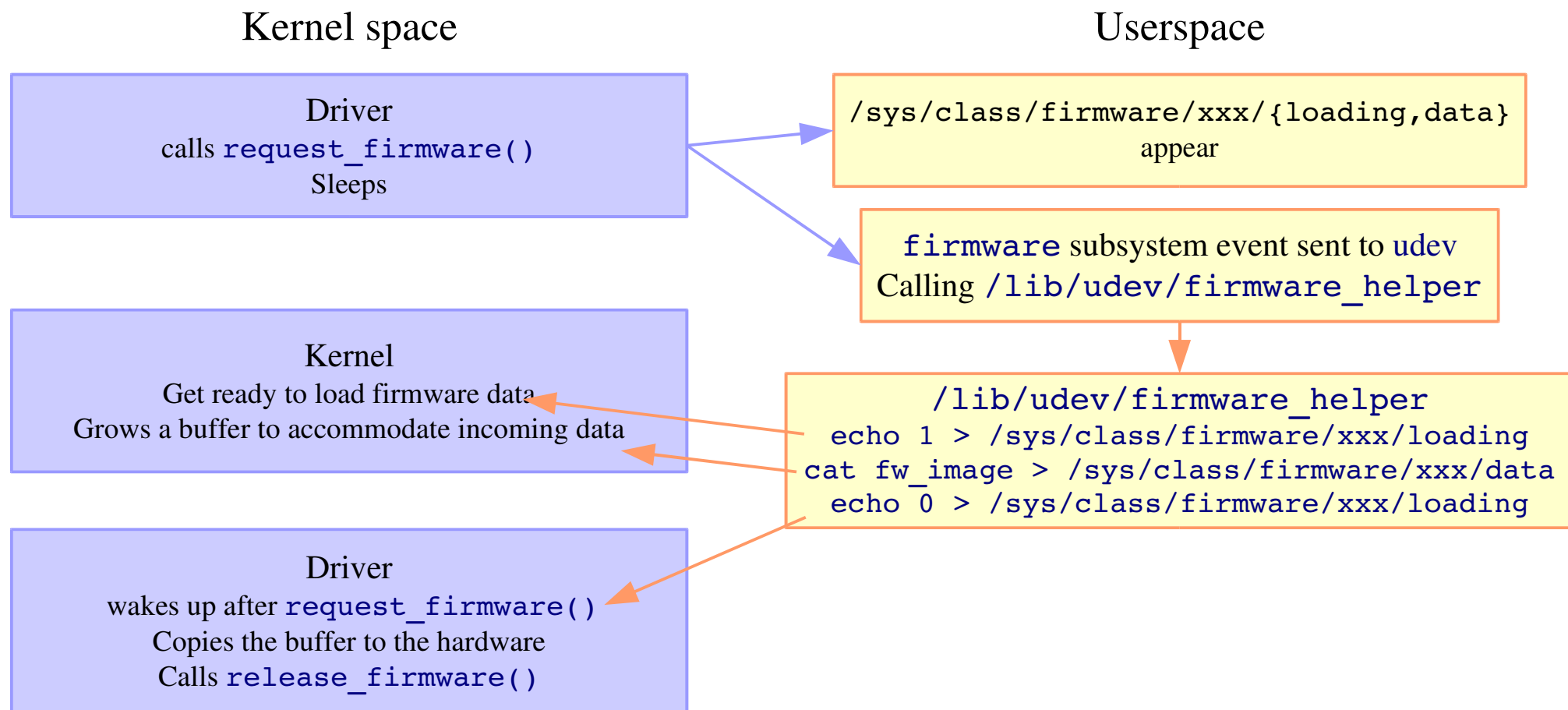
Firmware hotplugging

Also implemented with `udev`!

- ▶ Firmware data are kept outside device drivers
 - ▶ May not be legal or free enough to distribute
 - ▶ Firmware in kernel code would occupy memory permanently, even if just used once.
- ▶ Kernel configuration: needs to be set in `CONFIG_FW_LOADER`
(Device Drivers -> Generic Driver Options -> hotplug firmware loading support)



Firmware hotplugging implementation



See [Documentation/firmware_class/](#) for a nice overview



udev files

- ▶ `/etc/udev/udev.conf`
udev configuration file.
Mainly used to configure syslog reporting priorities.
Example setting: `udev_log="err"`
- ▶ `/etc/udev/rules.d/*.rules`
udev event matching rules.
- ▶ `/lib/udev/devices/*`
static `/dev` content (such as `/dev/console`, `/dev/null...`).
- ▶ `/lib/udev/*`
helper programs called from udev rules.
- ▶ `/dev/*`
Created device files.



Kernel configuration for udev

Created for 2.6.19

Caution: no documentation found, and not tested yet on a minimalistic system.
Some settings may still be missing.

Subsystems and device drivers (USB, PCI, PCMCIA...) should be added too!

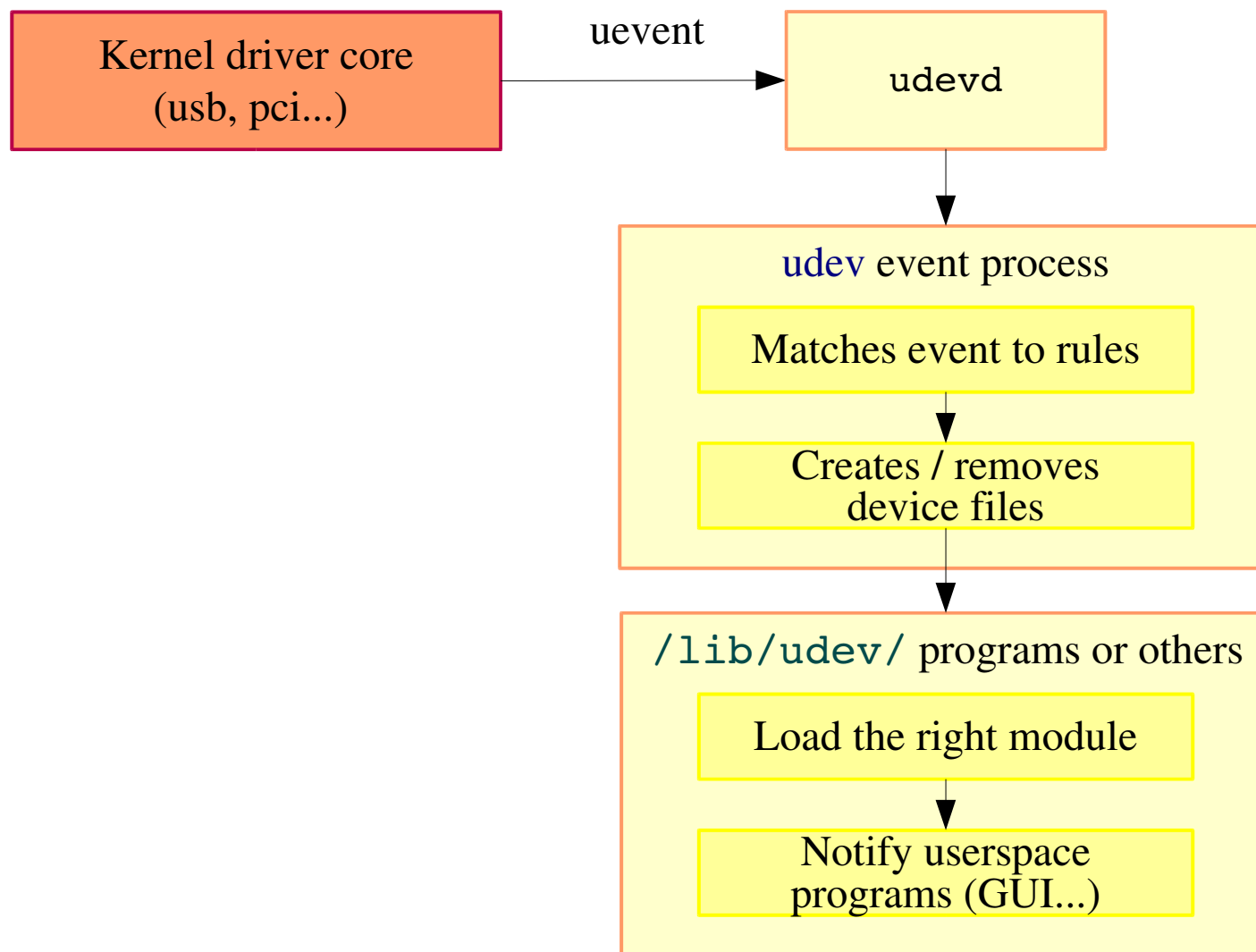
```
# General setup
CONFIG_HOTPLUG=y
# Networking, networking options
CONFIG_NET=y
CONFIG_UNIX=y
CONFIG_NETFILTER_NETLINK=y
CONFIG_NETFILTER_NETLINK_QUEUE=y
# Pseudo filesystems
CONFIG_PROC_FS=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
CONFIG_RAMFS=y
```

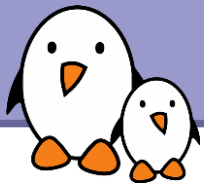
Unix domain sockets

Needed to manage /dev



udev summary - typical operation





udev resources

- ▶ Home page
<http://kernel.org/pub/linux/utils/kernel/hotplug/udev.html>
- ▶ Sources
<http://kernel.org/pub/linux/utils/kernel/hotplug/>
- ▶ Recent state of [udev](#), by Kay Sievers (very good article):
<http://vrfy.org/log/recent-state-of-udev.html>
- ▶ The udev manual page:
`man udev`



mdev, the udev for embedded systems

- ▶ **udev** might be too heavy-weight for some embedded systems, the **udev**d daemon staying in the background waiting for events.
- ▶ **BusyBox** provides a simpler alternative called **mdev**, available by enabling the **MDEV** configuration option.
- ▶ **mdev**'s usage is documented in `doc/mdev.txt` in the **BusyBox** source code.
- ▶ **mdev** is also able to load firmware to the kernel like **udev**



mdev usage

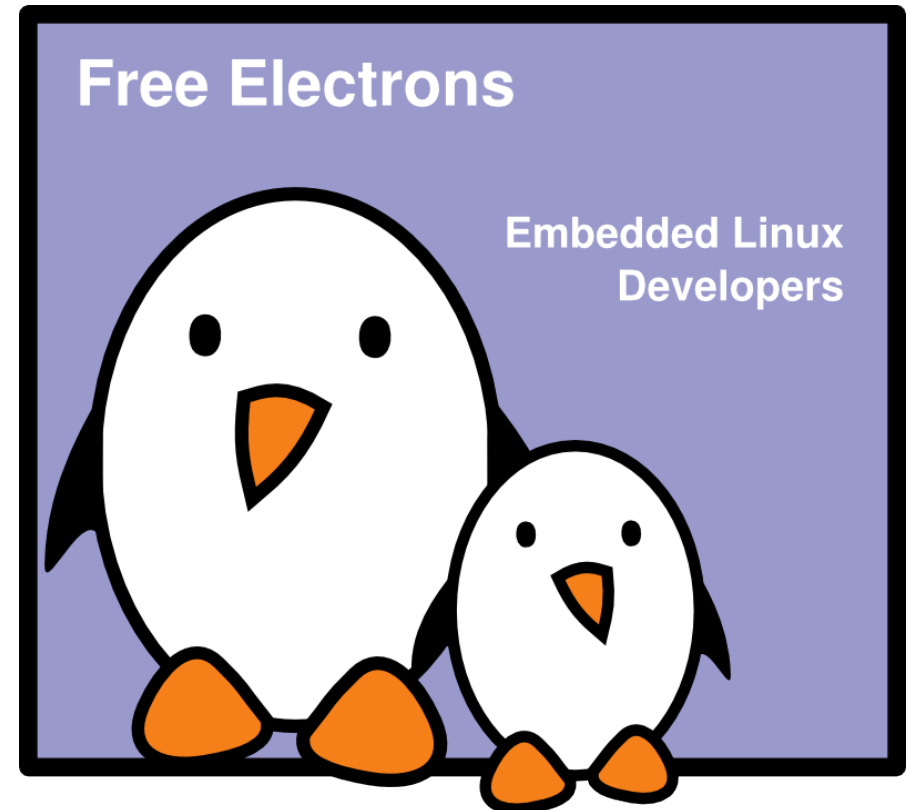
- ▶ To use **mdev**, the **proc** and **sysfs** filesystems must be mounted
- ▶ **mdev** must be enabled as the hotplug event manager
`echo /sbin/mdev > /proc/sys/kernel/hotplug`
- ▶ Tell **mdev** to create the **/dev** entries corresponding to the devices detected during boot when **mdev** was not running:
`mdev -s`
- ▶ The behaviour is specified by the `/etc/mdev.conf` configuration file, with the following format
`<device regex> <uid>:<gid> <octal permissions>
[=path] [@|$|*<command>]`
- ▶ Example
`hd[a-z][0-9]* 0:3 660`



Embedded Linux optimizations

Size, RAM, speed,
power, cost

Michael Opdenacker
Thomas Petazzoni
Free Electrons





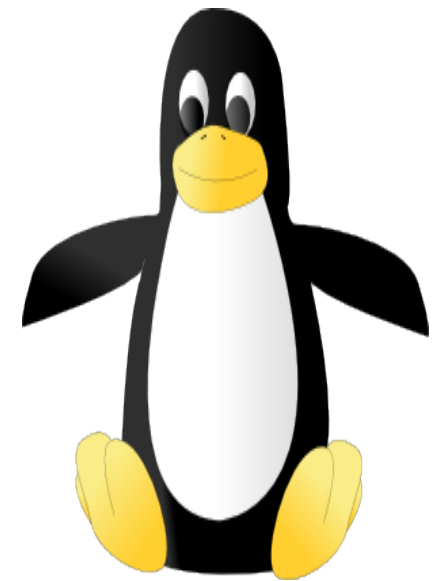
Penguin weight watchers

Make your penguin slimmer, faster, and reduce its consumption of fish!

Before



2 weeks after





CE Linux Forum

<http://celinuxforum.org/>



CE Linux Forum

- ▶ Non profit organization, whose members are embedded Linux companies and Consumer Electronics (CE) devices makers.
- ▶ Mission: develop the use of Linux in CE devices
- ▶ Hosts many projects to improve the suitability of Linux for CE devices and embedded systems. All patches are meant to be included in the mainline Linux kernel.
- ▶ Most of the ideas introduced in this presentation have been gathered or even implemented by CE Linux Forum projects!



Contents

Ideas for optimizing the Linux kernel and executables

- ▶ Increasing speed
- ▶ Reducing size: disk footprint and RAM
- ▶ Reducing power consumption
- ▶ Global perspective: cost and combined optimization effects
- ▶ The ultimate optimization tool!



Embedded Linux Optimizations

Increasing speed
Reducing kernel boot time



Measuring kernel boot time

CONFIG_PRINTK_TIME

- ▶ Configure it in the [Kernel Hacking](#) section.
- ▶ Adds timing information to kernel messages. Simple and robust.
- Not accurate enough on some platforms (1 jiffy = 10 ms on arm!)

See http://elinux.org/Printk_Times

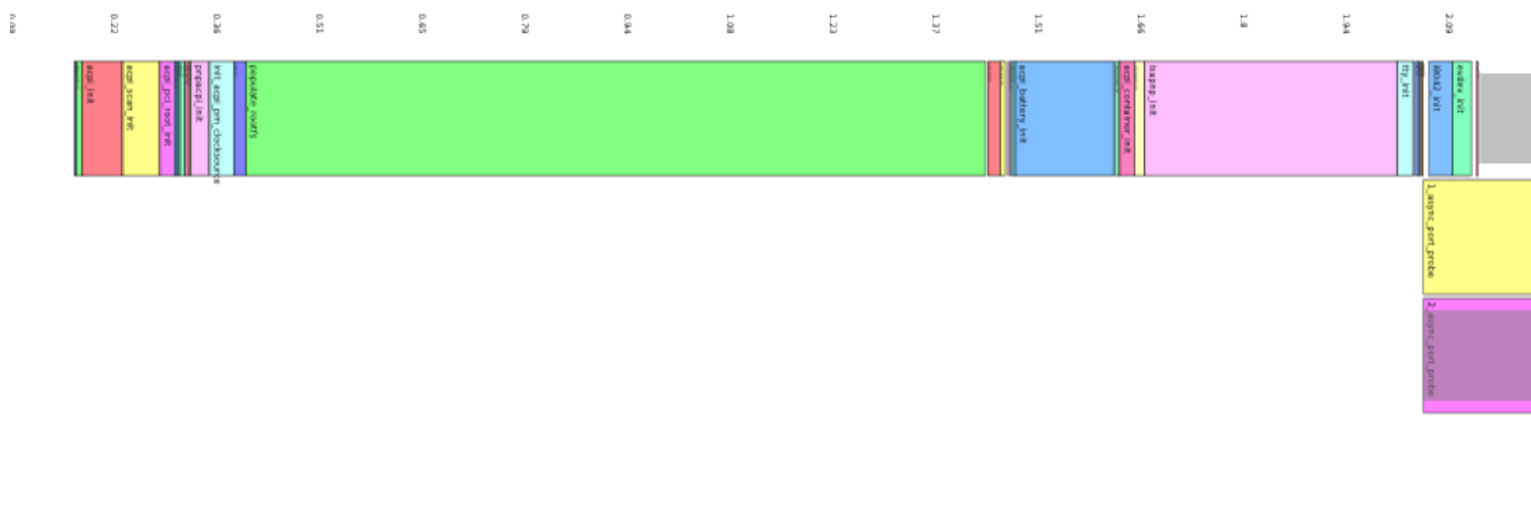
```
...  
[42949372.970000] Memory: 64MB = 64MB total  
[42949372.970000] Memory: 54784KB available (1404K code, 296K data, 72K init)  
[42949373.180000] Mount-cache hash table entries: 512  
[42949373.180000] CPU: Testing write buffer coherency: ok  
[42949373.180000] checking if image is initramfs...it isn't (bad gzip magic numbers); looks like an initrd  
[42949373.200000] Freeing initrd memory: 8192K  
[42949373.210000] NET: Registered protocol family 16  
...
```



Boot tracer

CONFIG_BOOT_TRACER in kernel configuration

- ▶ Introduced in Linux 2.6.28
Based on the `ftrace` tracing infrastructure
- ▶ Allows to record the timings of initcalls
- ▶ Boot with the `initcall_debug` and `printk.time=1` parameters,
and run `dmesg | perl scripts/bootgraph.pl > output.svg`
to generate a graphical representation





Disable console output

- ▶ The output of kernel bootup messages to the console takes time! Even worse: scrolling up in framebuffer consoles! Console output not needed in production systems.
- ▶ Console output can be disabled with the `quiet` argument in the Linux kernel command line (bootloader settings)
- ▶ Example:
`root=/dev/ram0 rw init=/startup.sh quiet`
- ▶ Benchmarks: can reduce boot time by 30 or even 50%!

See http://elinux.org/Disable_Console





Preset loops_per_jiffy

- ▶ At each boot, the Linux kernel calibrates a delay loop (for the `udelay` function). This measures a `loops_per_jiffy` (`lpj`) value. This takes about 25 jiffies (1 jiffy = time between 2 timer interrupts).
In embedded systems, it can be about 250 ms!
- ▶ You just need to measure this once! Find the `lpj` value in kernel boot messages (if you don't get it in the console, boot Linux with the `loglevel=8` parameter). Example:

```
Calibrating using timer specific routine... 187.59  
BogoMIPS (lpj=937984)
```

- ▶ At the next boots, start Linux with the below option:
`lpj=<value>`



Faster rebooting (1)

kexec system call: executes a new kernel from a running one.

- ▶ Must faster rebooting: doesn't go through bootstrap / bootloader code.
- ▶ Great solution for rebooting after system (“firmware”) upgrades.
- ▶ Useful for automatic rebooting after kernel panics.

See <http://developer.osdl.org/andyp/kexec/whitepaper/kexec.pdf>
and [Documentation/kdump/kdump.txt](#) in kernel sources.



Faster rebooting (2)

Another option: use `reboot=soft` in the kernel command line

- ▶ When you reboot, the firmware will be skipped.
- ▶ Drawback: unlike `kexec`, cannot be chosen from userspace.
- ▶ Supported platforms: `i386`, `x86_64`, `arm`, `arm26` (Aug. 2006)
- ▶ See `Documentation/kernel-parameters.txt` in the kernel sources for details. Not supported on all platforms.



Skip memory allocation

Idea: spare memory at boot time and manage it by yourself!

- ▶ Assume you have 32 MB of RAM

- ▶ Boot your kernel with `mem=30`

The kernel will just manage the first 30 MB of RAM.

- ▶ Driver code can now reclaim the 2 MB left:

```
buf = ioremap (  
    0x1e00000,          /* Start: 30 MB */  
    0x200000           /* Size: 2 MB */  
);
```

- ▶ This saves time allocating memory.

Critical drivers are also sure to always have the RAM they need.

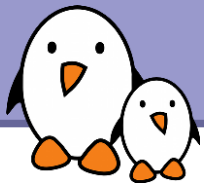


Other ideas

- ▶ Copy kernel from flash to RAM using DMA.
- ▶ Reducing probe time on some IDE operations.
- ▶ Compile drivers as modules for devices not used at boot time. This reduces time spent initializing drivers.
A smaller kernel is also faster to copy to RAM.
- ▶ Multithreaded device probing.

See http://elinux.org/Boot_Time for more resources

See also Tim Bird's presentation on boot time reduction:
<http://free-electrons.com/redir/tbird-elce2008.html>



Embedded Linux Optimizations



Increasing speed
System startup time and application speed



Starting system services

- ▶ **SysV init:**

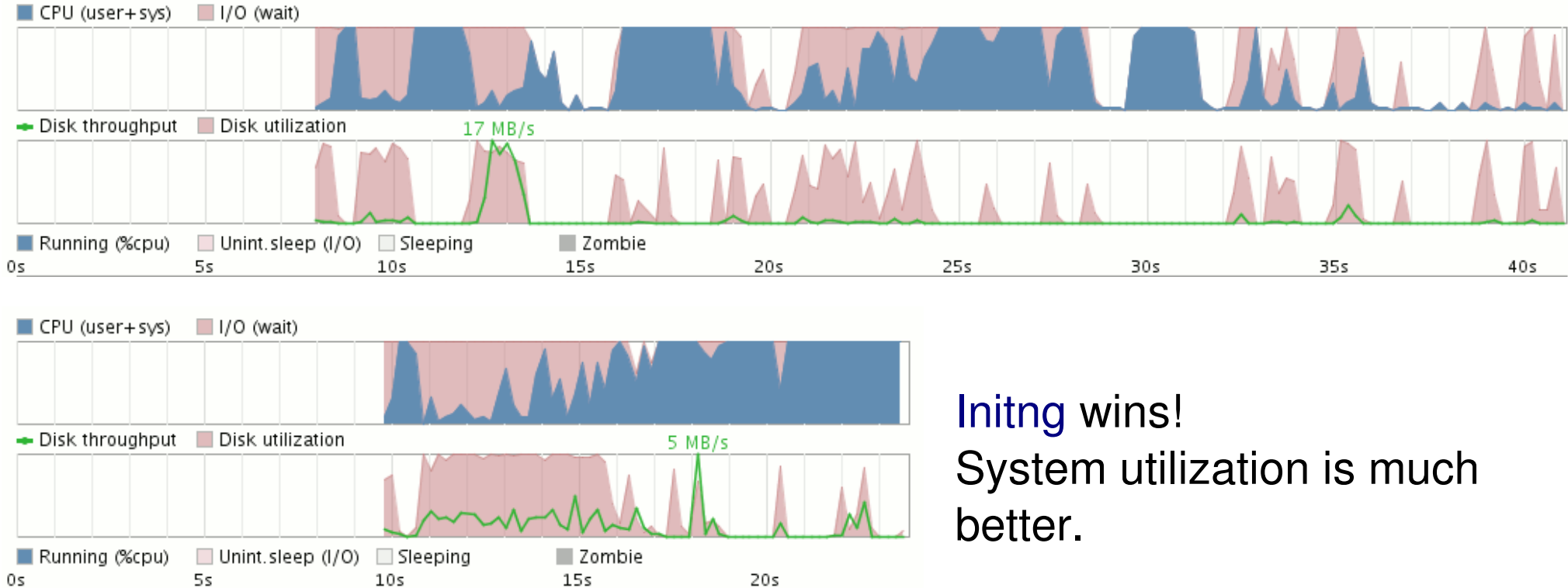
Starts services sequentially. Waits for the current startup script to be complete to start the next one! While dependencies exist, some tasks can be run in parallel!

- ▶ **Initng: <http://initng.org>**

New alternative to SysV init, which can start services in parallel, as soon as their preconditions are met.



Initng vs. SysV init: bootcharts



Initng wins!
System utilization is much better.

Note: you can hunt system startup trouble by using the [Bootchart](http://www.bootchart.org/) program (<http://www.bootchart.org/>). Used to investigate Fedora booting time.



Reading ahead

- ▶ Linux keeps the contents of all the files it reads in RAM (in the *page cache*), as long as it doesn't need the RAM pages for something else.
- ▶ Idea: load files (programs and libraries in particular) in RAM cache before using them. Best done when the system is not doing any I/O.
- ▶ Thanks to this, programs are not stuck waiting for I/O. Used the Knoppix distribution to achieve very nice boot speed-ups.
- ▶ Also planned to be used by [Initng](#).
- ▶ Not very useful for systems with very little RAM: cached pages are recycled before the files are accessed.



Implementing readahead

- ▶ You can use the `sys_readahead()` system call in your C programs. See `man readahead` for details.
- ▶ You can also use the `readahead-list` utility, which reads a file containing the list of files to load in cache.
Available on: <http://freshmeat.net/projects/readahead-list/>.
- ▶ In embedded systems using `Busybox`, you can use the `readahead` command (implemented by Free Electrons).



Compiler speed optimizations

- ▶ By default, most tools are compiled with compiler optimizations. Make sure you use them for your own programs!
- ▶ `-O2` is the most common optimization switch of `gcc`. Lots of optimization techniques are available. See http://en.wikipedia.org/wiki/Compiler_optimization
- ▶ `-O3` can be also be used for speed critical executables. However, there is done at the expense of code size (for example “inlining”: replacing function calls by the function code itself).



Using processor acceleration instructions

- ▶ **liboil** - <http://liboil.freedesktop.org/>
Library of functions optimized for special instructions from several processors (**AltiVec**, **MMX**, **SSE**, etc.)
- ▶ Mainly functions implementing loops on data arrays: type conversion, copying, simple arithmetics, direct cosine transform, random number generation...
- ▶ Transparent: keeps your application portable!
- ▶ So far mainly supports desktop processors
- ▶ License: BSD type



Prelinking (1)

Applies to executables using shared libraries

- ▶ To load and start an executable, the dynamic linker has a significant amount of work to do (mainly address relocation)
- ▶ It can take a lot of time for executables using many shared libraries!
- ▶ In many systems in which executables and shared libraries never change, the same job is done every time the executable is started.



Prelinking (2)

`prelink`

<http://people.redhat.com/jakub/prelink/>

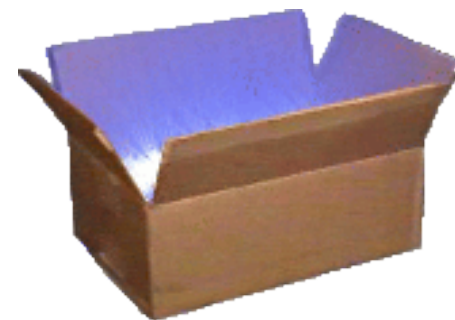
- ▶ `prelink` modifies executables and shared libraries to simplify the dynamic linker relocation work.
- ▶ This can greatly reduce startup time for big applications (50% less for KDE!). This also saves memory consumed by relocations.
- ▶ Can be used to reduce the startup time of a Linux system.
- ▶ Just needs to be run again when libraries or executables are updated.

Details on http://elinux.org/Pre_Linking



Use simpler Unix executables

- ▶ Big, feature rich executables take time to load.
Particularly true for shell scripts calling the `bash` shell!
- ▶ Idea: replace standard Unix / GNU executables by lightweight, simplified implementations by `busybox` (<http://busybox.net>).
- ▶ Implemented by Ubuntu 6.10 to reduce boot time, replacing `bash` (649 K) by `dash` (79 K, see http://en.wikipedia.org/wiki/Debian_Almquist_shell).
This broke various shell scripts which used bash specific features (“bashisms”).
- ▶ In non-embedded Linux systems
where feature-rich executables are still needed,
should at least use `busybox ash` for system scripts.





Shells: reducing forking

- ▶ `fork` / `exec` system calls are very heavy.
Because of this, calls to executables from shells are slow.
- ▶ Even executing `echo` in `busybox` shells results in a `fork` syscall!
- ▶ Select `Shells -> Standalone shell` in `busybox` configuration to make the `busybox` shell call applets whenever possible.
- ▶ Pipes and back-quotes are also implemented by `fork` / `exec`.
You can reduce their usage in scripts. Example:
`cat /proc/cpuinfo | grep model`
Replace it with: `grep model /proc/cpuinfo`

See http://elinux.org/Optimize_RC_Scripts



Use faster filesystems

Run faster by using the most appropriate filesystems!

- ▶ Compressed read-only filesystem (block device):
use **SquashFS** (<http://squashfs.sourceforge.net>)
instead of **CramFS** (much slower, getting obsolete).
- ▶ NAND flash storage: you should try **UBIFS**
(<http://www.linux-mtd.infradead.org/doc/ubifs.html>), the
successor of **JFFS2**. It is much faster. You could also use
SquashFS. See our Choosing filesystems presentation
(<http://free-electrons.com/docs/filesystems>).



Use faster filesystems (2)

- ▶ Use RAM filesystems for temporary, speed critical files with no need for permanent storage. Details in the kernel sources: [Documentation/filesystems/tmpfs.txt](#)
- ▶ Benchmark your system and application on competing filesystems! [Reiser4](#) is more innovative and benchmarks found it faster than [ext3](#).
- ▶ Good to benchmark your system with JFS or XFS too. XFS is reported to be the fastest to mount (good for startup time), and JFS to have the lowest CPU utilization. See <http://www.debian-administration.org/articles/388>
- ▶ [ext4](#) is also ready to be used now.



Speed up applications with tmpfs

- ▶ When enough RAM is available, the OS keeps recently accessed files and applications in RAM (*page cache*). This significantly speeds up any new usage. However, depending on system activity, this may not last long.
- ▶ For programs that need fast startup even if they haven't been run for a long time: copy them to a tmpfs filesystem at system startup! This makes sure they are always accessed from the file cache in RAM (provided you do not have a swap partition).
- ▶ See [Documentation/filesystems/tmpfs.txt](#) in kernel sources for details about tmpfs.
- ▶ Caution: don't use ramdisks instead!
Ramdisks duplicate files in RAM and unused space cannot be reclaimed.
- ▶ Caution: use with care. May impact overall performance.
Not needed if there's enough RAM to cache all files and programs.



Boot from a hibernate image

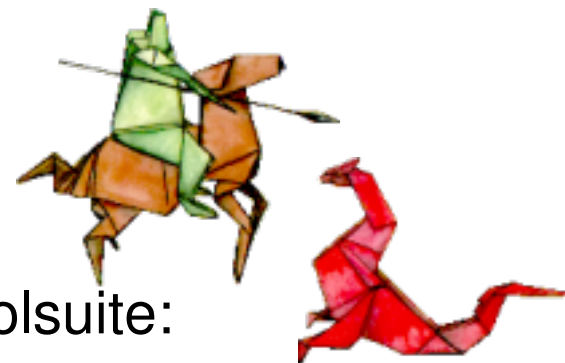
The ultimate technique for instant boot!

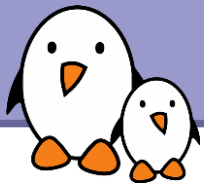
- ▶ In development: start the system, required applications and the user interface. Hibernate the system to disk / flash in this state.
- ▶ In production: boot the kernel and restore the system state from with this predefined hibernation image.
- ▶ This way, you don't have to initialize the programs one by one. You just get the back to a valid state.
- ▶ Used in Sony cameras to achieve instant power on time.
- ▶ Unlike Suspend to RAM, still allows to remove batteries!



Use a profiler

- ▶ Using a profiler can help to identify unexpected behavior degrading application performance.
- ▶ For example, a profiler can tell you in which functions most of the time is spent.
- ▶ Easy to do with **Valgrind**: <http://valgrind.org/>
 - ▶ Compile your application for **x86** architecture
 - ▶ You can then profile it with the whole **Valgrind** toolsuite:
Cachegrind: sources of cache misses and function statistics.
Massif: sources of memory allocation.





Reducing size
Kernel size and RAM usage



Linux-Tiny

Goal: reduce the disk footprint and RAM size of the Linux kernel

http://elinux.org/Linux_Tiny

- ▶ Set of patches against the mainstream Linux kernel.
Mergeability in mainstream is a priority.
Many changes have already been merged in recent kernels.
- ▶ All features can be selected in kernel configuration
(`CONFIG_EMBEDDED`).
- ▶ Also ships utilities or patches for tracking sources
of memory usage or code size.



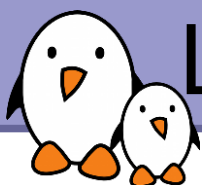
Linux-Tiny ideas (1)

- ▶ Remove kernel messages (`printk`, `BUG`, `panic`...)
- ▶ Hunt excess inlining (speed vs. size tradeoff)
2.6.26: can allow gcc to uninline functions marked as inline: (`CONFIG_OPTIMIZE_INLINING=y`). Only used by `x86` so far.
- ▶ Hunt excess memory allocations
- ▶ Memory (`slob` instead of `slab`) allocator more space efficient for small systems.
- ▶ Reduce the size of kernel data structures (may impact performance)
- ▶ Simpler alternative implementations of kernel functionalities with less features, or not supporting special cases.

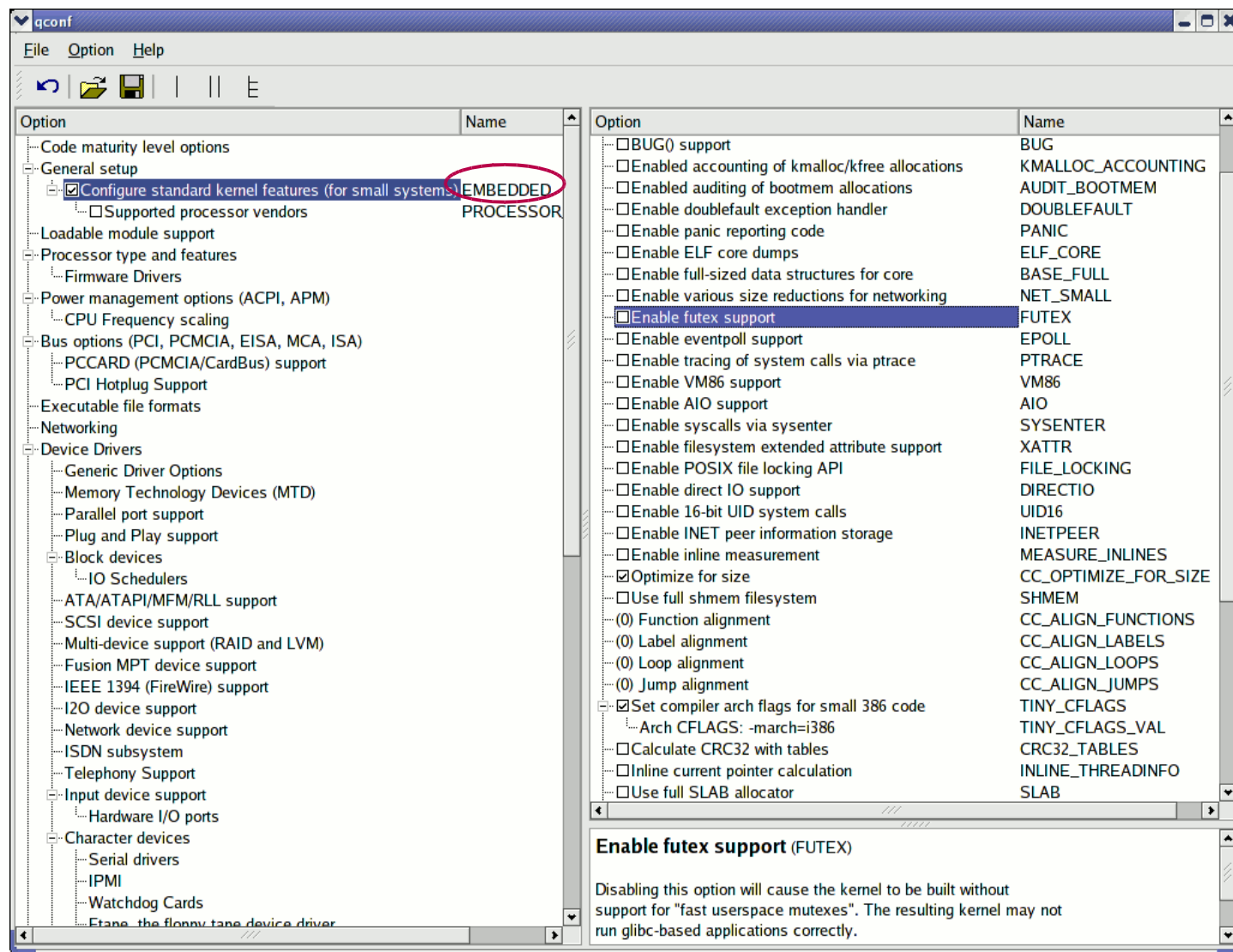


Linux-Tiny ideas (2)

- ▶ Remove some features which may not be needed in some systems.
- ▶ Compiling optimizations for size.
- ▶ A smaller kernel executable also saves RAM (unless executed in place from storage).



Linux-Tiny: kernel configuration screenshot



Many features
configured out



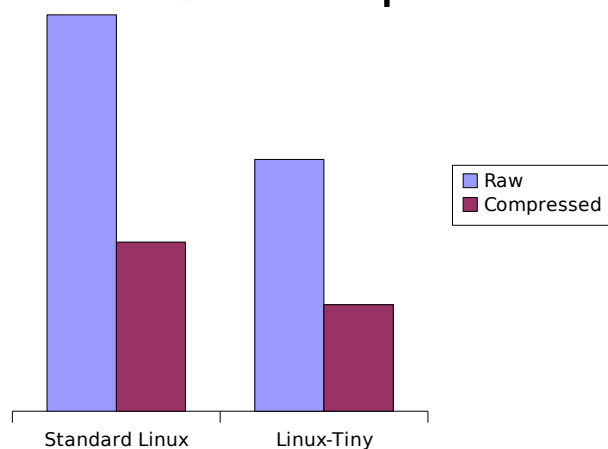
Kernel size reduction example

Standard 2.6.13.4 kernel with minimum features (without `CONFIG_EMBEDDED`), for a minimalistic PC.

- ▶ Raw: 1205956 bytes
- ▶ Compressed: 515045 bytes

Linux-Tiny patched 2.6.13.4 kernel with all `CONFIG_EMBEDDED` space saving settings, for a minimalistic PC.

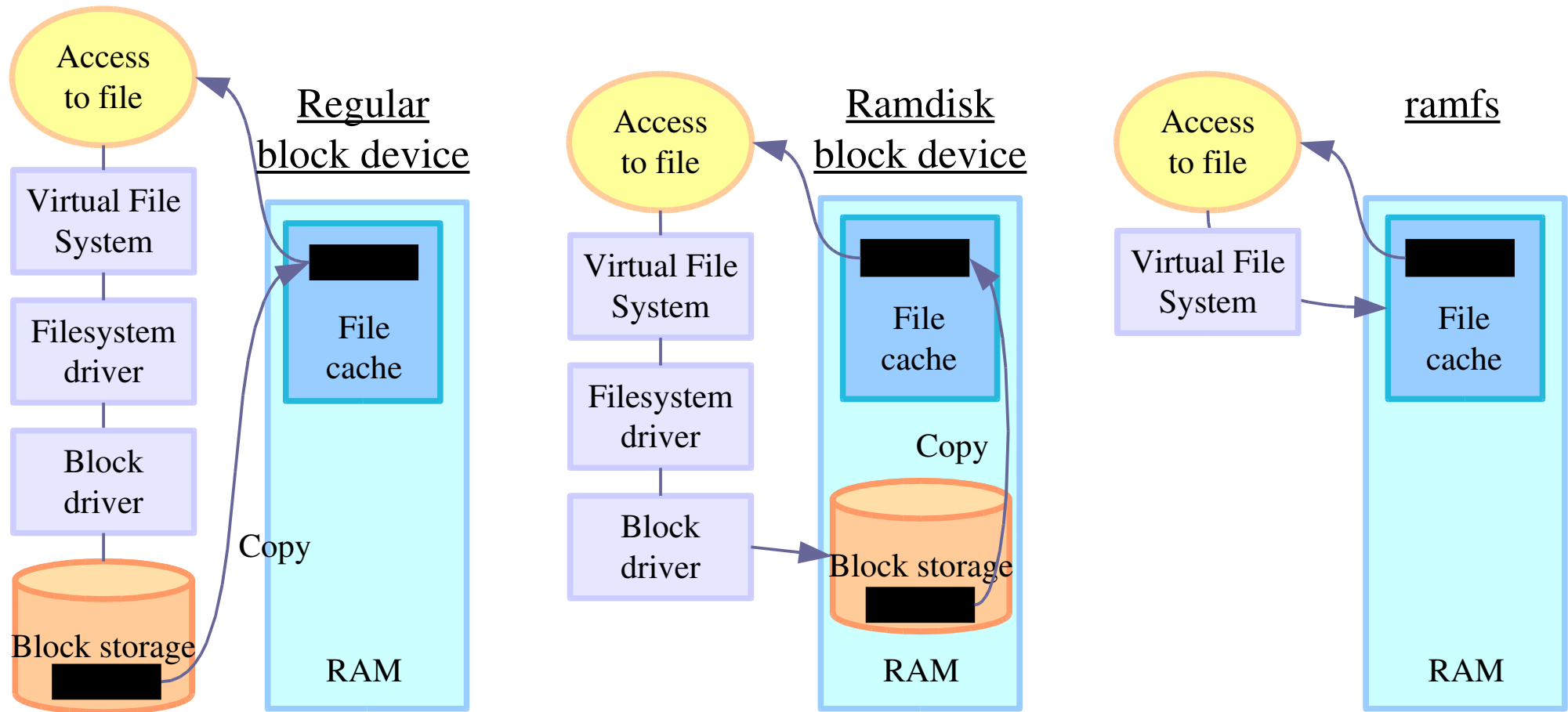
- ▶ Raw: 766153 bytes
- ▶ Compressed: 323788 bytes





Replace initrd by initramfs

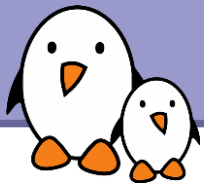
Replace init ramdisks (initrd) with initramfs:
much less overhead and ram waste!





ramfs advantages over ramdisks

- ▶ No block and filesystem overhead.
- ▶ No duplication in RAM.
- ▶ Files can be removed (reclaiming RAM) after use.
- ▶ Initramfs: ramfs archive embedded in the Linux kernel file.



Reducing size
Application size and RAM usage



Static or dynamic linking? (1)

Static linking

- ▶ All shared library code duplicated in the executables
- Allows not to copy the C library in the filesystem.
Simpler and smaller when very few executables (busybox)
- Library code duplication: bad for systems with more executables
(code size and RAM)

Best for small systems (< 1-2 MB) with few executables!



Static or dynamic linking? (2)

Dynamic linking

- ▶ Shared library code not duplicated in the executables
- Makes much smaller executables
- Saves space in RAM (bigger executables take more RAM)
- Requires the library to be copied to the filesystem

Best for medium to big systems (> 500 KB - 1 MB)



Using a lighter C library

- ▶ **glibc** (GNU C library): <http://www.gnu.org/software/libc/>
Found on most computer type GNU/Linux machines
Size on **arm**: approx 1.7 MB
- ▶ **uClibc**: <http://www.uclibc.org/>
Found in more and more embedded Linux systems!
Size on **arm**: approx 400 KB (you save 1.2 MB!)
- ▶ Executables are slightly smaller too:

<i>C program</i>	<i>Compiled with shared libraries</i>		<i>Compiled statically</i>	
	<i>glibc</i>	<i>uClibc</i>	<i>glibc</i>	<i>uClibc</i>
Plain “hello world”	4.6 K	4.4 K	475 K	25 K
Busybox	245 K	231 K	843 K	311 K



How to use uClibc?

- ▶ Need to compile all your executables with a **uClibc** toolchain.
- ▶ Ready-to-use toolchains can be found on <http://free-electrons.com/community/tools/uclibc>
- ▶ You can very easily build your own with **buildroot**:
<http://buildroot.uclibc.org/>
- ▶ You also have to copy the **uClibc** files from the toolchain to the `/lib` directory in the target root filesystem.
- ▶ Ready-to-use filesystems can also be generated by **buildroot**.
You just need to add your specific stuff then.



Need for stripping

- ▶ Compiled executables and libraries contain extra information which can be used to investigate problems in a debugger.
- ▶ This was useful for the tool developer, but not for the final user.
- ▶ To remove debugging information, use the `strip` command. This can save a very significant amount of space!
`gcc -o hello hello.c` (output size: 4635 bytes)
`strip hello` (output size: 2852 bytes, -38.5%)
- ▶ Don't forget to strip libraries too!



Are my executables stripped?

You can use the `file` command to get the answer

```
gcc -o hello hello.c
```

```
file hello
```

```
hello: ELF 32-bit LSB executable, Intel 80386, version 1  
(SYSV), for GNU/Linux 2.2.5, dynamically linked (uses  
shared libs), not stripped
```

```
strip hello
```

```
hello: ELF 32-bit LSB executable, Intel 80386, version 1  
(SYSV), for GNU/Linux 2.2.5, dynamically linked (uses  
shared libs), stripped
```

You can use `findstrip` (<http://packages.debian.org/stable/source/perforate>) to find all executables and libraries that need stripping in your system.



How to strip

- ▶ Some lightweight tools, like busybox, are automatically stripped when you build them.
- ▶ Makefiles for many standard tools offer a special command:
`make install-strip`
- ▶ Caution: stripping is architecture dependent.
Use the strip command from your cross-compiling toolchain:
`arm-linux-strip potato`



sstrip: “super strip”

<http://muppetlabs.com/~breadbox/software/elfkickers.html>

- ▶ Goes beyond **strip** and can strip out a few more bits that are not used by Linux to start an executable.
- ▶ Can be used on libraries too. Minor limitation: processed libraries can no longer be used to compile new executables.
- ▶ Can also be found in toolchains made by **Buildroot** (optional)

	<i>Hello World</i>	<i>Busybox</i>	<i>Inkscape</i>
Regular	4691 B	287783 B	11397 KB
stripped	2904 B (-38 %)	230408 B (-19.9 %)	9467 KB (-16.9 %)
sstripped	1392 B (-70 %)	229701 B (-20.2 %)	9436 KB (-17.2 %)

Best for tiny
executables!



Library Optimizer

<http://libraryopt.sourceforge.net/>

- ▶ Contributed by MontaVista
- ▶ Examines the complete target file system, resolves all shared library symbol references, and rebuilds the shared libraries with only the object files required to satisfy the symbol references.
- ▶ Can also take care of stripping executables and libraries.
- ▶ However, requires to rebuild all the components from source. Would be nicer to achieve this only with ELF manipulations.



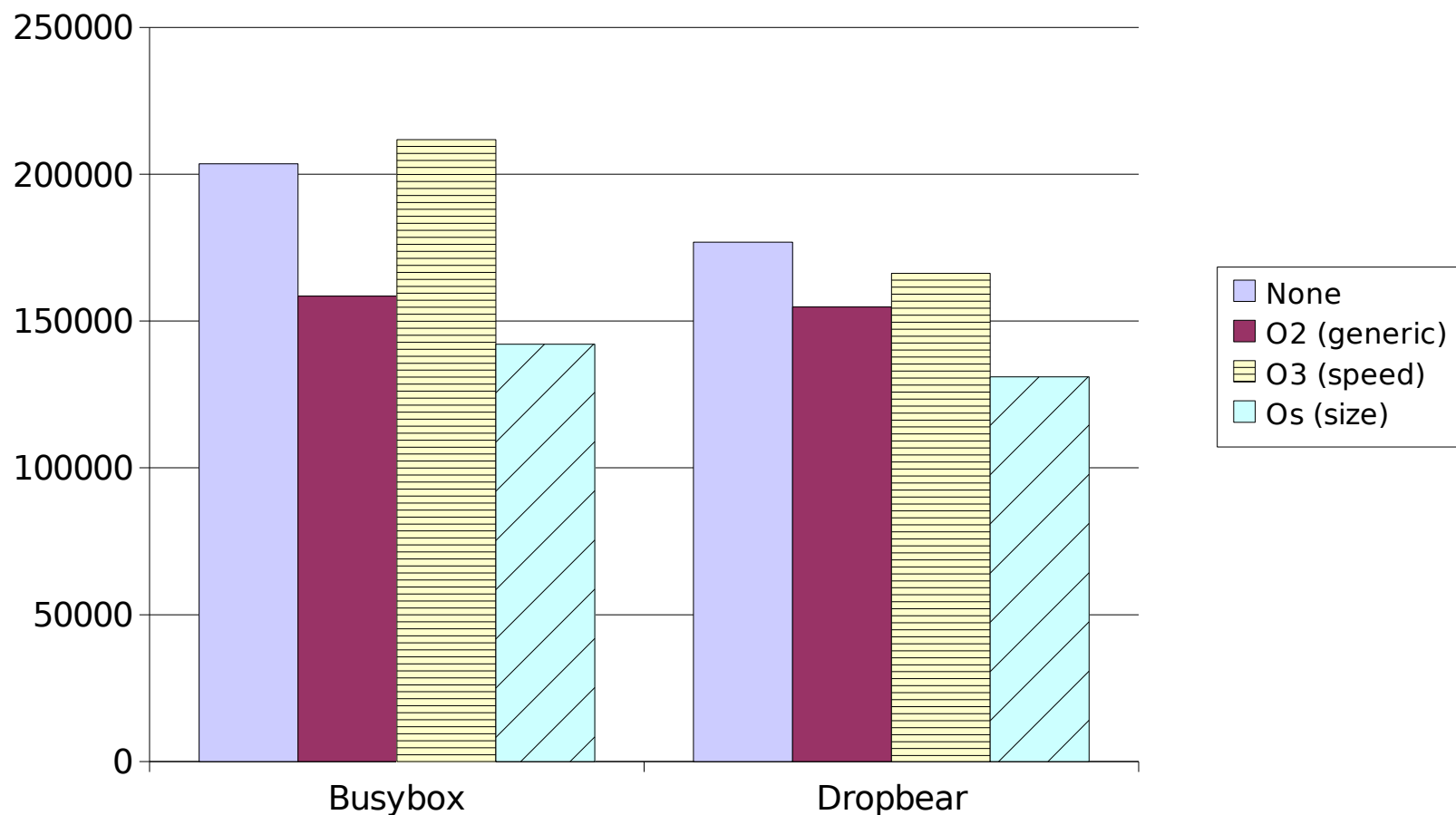
Compiler space optimizations

- ▶ Regular compiler optimizations simplifying code also reduce size
- ▶ You can also reduce the size of executables by asking `gcc` to optimize generated code size:
`gcc -Os -o husband husband.c`
- ▶ `-Os` corresponds to `-O2` optimizations except the ones increasing size, plus extra size-specific ones.
- ▶ `-Os` is already used by default to build `busybox`.
- ▶ Possible to further reduce the size by compiling and optimizing all sources at once, with the `-fwhole-program --combine` gcc options.
- ▶ See <http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html> for all gcc optimization options.



Simple gcc optimization benchmark

Executable size





Restartable applications

- ▶ When RAM is scarce, can be useful to abort applications that are not in use (for example hidden graphical interfaces).
- ▶ Better to do it before the Linux Kernel OOM (Out Of Memory) killer comes and makes bad decisions.
- ▶ To support this, design your programs with the capability to save their state when they are aborted and return to it when they are restarted.



Compressing filesystems

Can significantly increase your storage capacity

- ▶ MTD (flash or ROM) storage: use **UBIFS** or **JFFS2** for small partitions.
- ▶ Block storage: use **SquashFS** (<http://squashfs.sourceforge.net>) instead of **CramFS** for read-only partitions. It compresses much better and is much faster too.



Merging duplicate files

Software compiling and installing often create duplicate files...
Check that your root filesystem doesn't contain any!

- ▶ **dupmerge2**: <http://sourceforge.net/projects/dupmerge>
Replaces duplicate files by hard links.
- ▶ **clink**: <http://free-electrons.com/community/tools/utils/clink>
Replaces duplicate files by symbolic links.
Example: saves 4% of total space in **Fedora Core 5**.
- ▶ **finddup**: <http://www.shelldorado.com/scripts/cmds/finddup>
Finds duplicate files.



Reducing power consumption



Tickless kernel

Kernel configuration: `NO_HZ` setting in **Processor type and features**

- ▶ To implement multitasking, the processor receives a timer interrupt at a given frequency (every 4 ms by default on Linux 2.6). On idle systems, this wakes up the processor all the time, just to realize there is nothing to do!
- ▶ Idea: when all processors are idle, disable the timer interrupt, and re-enable it when something happens (a real interrupt). This saves power in laptops, in embedded systems and with virtual servers!
- ▶ 2.6.24: supports `x86`, `arm`, `mips` and `powerpc`

Option	Name
<input checked="" type="checkbox"/> Tickless System (Dynamic Ticks)	<code>NO_HZ</code>
<input type="checkbox"/> High Resolution Timer Support (NEW)	<code>HIGH_RES_TIMERS</code>



PowerTOP

<http://www.lesswatts.org/projects/powertop/>

- ▶ A very nice utility showing the top 10 sources of power consumption.
- ▶ Requirements: a tickless kernel (> 2.6.21), and a “mobile” (x86) CPU. Best run on laptops running on battery (to estimate battery savings).
- ▶ PowerTOP measures the number of wake-ups, and counts the time spent in low-power modes.
- ▶ It detects issues in both kernel space (drivers) and user space. It already uncovered several bugs in today's distributions.
- ▶ Embedded systems: non Intel processors probably not supported yet. You can still run your apps on a PC to check their behavior.

See also <http://www.lesswatts.org> for other useful resources.



PowerTOP in action

```
PowerTOP version 1.8      (C) 2007 Intel Corporation

Cn      Avg residency      P-states (frequencies)
C0 (cpu running)      (12.0%)      1.60 Ghz      0.0%
C1      0.0ms ( 0.0%)      1400 Mhz      0.0%
C2      5.0ms (88.0%)      800 Mhz      2.8%
C3      0.0ms ( 0.0%)      600 Mhz      97.2%
C4      0.0ms ( 0.0%)

Wakeups-from-idle per second : 177.5      interval: 15.0s
Power usage (ACPI estimate): 18.4W (1.9 hours) (long term: 250.0W,/0.1h)

Top causes for wakeups:
48.2% ( 93.9)      <interrupt> : uhci_hcd:usb1, uhci_hcd:usb2, uhci_hcd:usb3, ehci_hcd:usb4, yenta,
16.1% ( 31.4)      <interrupt> : libata
10.6% ( 20.7)      firefox-bin : futex_wait (hrtimer_wakeup)
5.1% ( 10.0)      hald-addon-cpuf : cpufreq_governor_dbs (delayed_work_timer_fn)
5.1% ( 9.9)      <interrupt> : Intel 82801DB-ICH4, ipw2200
2.9% ( 5.7)      artsd : schedule_timeout (process_timeout)
2.0% ( 3.9)      <kernel module> : usb_hcd_poll_rh_status (rh_timer_func)
1.5% ( 2.9)      gnome-screensav : schedule_timeout (process_timeout)
1.5% ( 2.9)      <kernel core> : cfq_completed_request (cfq_idle_slice_timer)
0.7% ( 1.3)      kicker : schedule_timeout (process_timeout)
0.5% ( 1.1)      klipper : schedule_timeout (process_timeout)
0.5% ( 1.0)      dhcdbd : schedule_timeout (process_timeout)
0.5% ( 1.0)      artsd : do_setitimer (it_real_fn)

Suggestion: Enable laptop-mode by executing the following command:
echo 5 > /proc/sys/vm/laptop_mode

Q - Quit      R - Refresh      L - enable Laptop mode
```



cpufreq

Configuration: `CPU_FREQ` in Power management options

- ▶ Allows to change the CPU frequency on the fly
- ▶ Supported architectures (2.6.20):
`i386`, `sh`, `ia64`, `sparc64`, `x86_64`, `powerpc`, `arm` (`i.MX` only).
- ▶ Usually controlled from userspace through `/sys` by a user configurable *governor* process, according to CPU load, heat, battery status... The most common is `cpuspeed`:
<http://carlthompson.net/software/cpuspeed/>
- ▶ Saves a significant amount of battery life in notebooks.



Suspend hidden GUIs

Idea: suspend hidden user interfaces to save CPU and power.

- ▶ Send a suspend (stop) signal:

```
kill -SIGTSTP <pid>
```

- ▶ Send a continue signal:

```
kill -SIGCONT <pid>
```



Software suspend

<http://www.suspend2.net/>

- ▶ Lots of great features for notebook users, such as RAM suspend or hibernate to disk.
- Unfortunately, restricted on some Intel compatible processors and targeting only machines with APM or ACPI (rarely found in non PC embedded systems!).
- Not addressing the requirements of embedded systems (support for other CPUs, voltage reduction...).



Global perspective
Cost and combined optimization effects



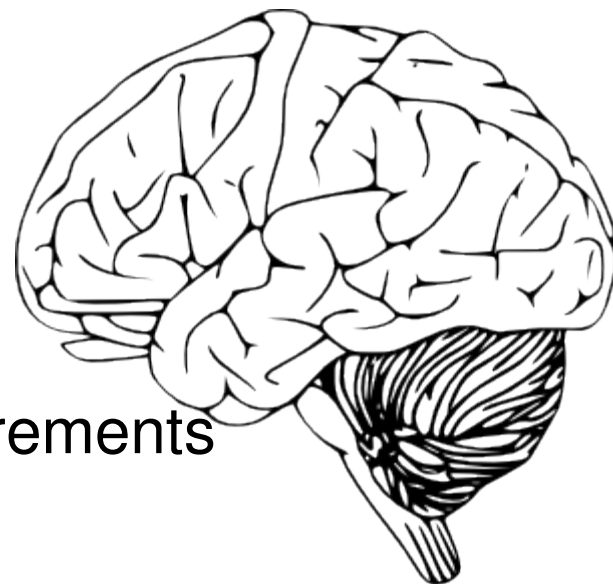
Combined benefits

→	Speed increase	RAM reduction	Power reduction	Cost reduction
More speed			- CPU can run slower or stay longer in power saving mode	- Slower, cheaper CPU
Less RAM	<ul style="list-style-type: none">- Faster allocations- Less swapping- Sometimes less cache flushing		<ul style="list-style-type: none">- Fewer / smaller RAM chips: less dynamic and standby power.- CPU with less cache: less power	<ul style="list-style-type: none">- Fewer / cheaper RAM chips- CPU with less cache: cheaper
Less space	<ul style="list-style-type: none">- Faster application loading from storage and in RAM.- Sometimes, simpler, faster code.	- Less RAM usage	- Fewer / smaller storage chips: less power	- Fewer / cheaper storage
Less power				<ul style="list-style-type: none">- Cheaper batteries- or cheaper AC/DC converter




The ultimate optimization tool!

- ▶ We have seen many ways to optimize an existing system.
- ▶ However, nothing replaces a good design!
- ▶ So, first carefully design and implement your system and applications with their requirements in mind.
- ▶ Then, use the optimization techniques to further improve your system and the parts that you reused (kernel and applications).





Related documents



Free Electrons

Embedded Freedom

HOME DEVELOPMENT SERVICES TRAINING DOCS COMMUNITY COMPANY BLOG

Recent blog posts

ELC Europe in Grenoble

Free Electrons at ELC

Linux kernel 2.6.29 - New features for embedded users

The Buildroot project begins a new life

FOSDEM 2009 videos

USB-Ethernet device for Linux

Program for Embedded Linux Conference 2009 announced

Public session changes


Real hardware in our training sessions

Call for presentations for the LSM embedded track

Docs

Most of the below documents are presentations used in our [training sessions](#), or in technical conferences.

License

 All our documents are available under the terms of the [Creative Commons Attribution-ShareAlike 3.0 license](#). This essentially means that you are free to download, distribute and even modify them, provided you mention us as the original authors and that you share these documents under the same conditions.

Linux kernel

- [Embedded Linux kernel and driver development](#)
- [New features in Linux 2.6](#) (since 2.6.10)
- [Kernel initialization](#)
- [Porting Linux to new hardware](#)
- [Power management in Linux](#)
- [Linux PCI drivers](#)
- [Block device drivers](#)
- [Linux USB drivers](#)
- [DMA](#)

Architecture specific documents

- [ARM Linux specifics](#)
- [Linux on TI OMAP processors](#)

Embedded Linux system development

- [Embedded Linux system development](#)
- [Real time in embedded Linux systems](#)
- [Block filesystems](#)
- [Flash filesystems](#)
- [Free software development tools](#)
- [The U-boot bootloader](#)
- [The GRUB bootloader](#)
- [The blob bootloader](#)
- [Hotplugging with udev](#)
- [Introduction to uClinux](#)
- [Java in embedded Linux](#)
- [Embedded Linux optimizations](#)
- [Audio in embedded Linux systems](#)
- [Multimedia in embedded Linux systems](#)
- [Embedded Linux From Scratch... in 40 minutes!](#)
- [Building embedded Linux systems with Buildroot](#)
- [Developing embedded distributions with OpenEmbedded](#)
- [The Scratchbox development environment](#)

Miscellaneous

- [Introduction to the Unix command line](#)
- [SSH](#)
- [Linux virtualization solutions](#) (with an embedded perspective)
- [Advantages of Free Software and Open Source in embedded systems](#)
- [Introduction to GNU/Linux and Free Software](#)

All our technical presentations
on <http://free-electrons.com/docs>

- ▶ Linux kernel
- ▶ Device drivers
- ▶ Architecture specifics
- ▶ Embedded Linux system development



Thank you!

And may the Source be with you