Embedded Linux training

Gregory Clement
Thomas Petazzoni
Michael Opdenacker
Maxime Ripard

Rights to copy

© Copyright 2004-2011, Free Electrons
feedback@free-electrons.com

Electronic version of this document available on
http://free-electrons.com/doc/training/embedded-linux

Updates will be available on
http://free-electrons.com/doc/training/embedded-linux/

Corrections, suggestions,
contributions and translations are welcome!

Latest update: Nov 16, 2011

creative commons
COMMONS DEED

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

Linux kernel

Linux device drivers
Board support code
Mainlining 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

Hardware used in this training session

- TI DM3730 (OMAP3 CPU @ 1 Ghz, DSP, OpenGL)
- 512 MB RAM, 512 MB flash
- Wifi + Bluetooth (according to availability), Ethernet
- RS-232 serial
- USB Host, USB OTG
- DVI-D, Audio In and Out
- microSD
- Expansion connectors : UART, SPI, I2C, GPIO, keyboard, LCD
- Approximately 130 EUR
- http://www.igep.es

ISEE IGEP v2

Participate!

During the lectures...

- Don't hesitate to ask questions. Other people in the audience may have similar questions too.
- This helps the trainer to detect any explanation that wasn't clear or detailed enough.
- Don't hesitate to share your experience, for example to compare Linux with other operating systems used in your company.
- Your point of view is most valuable, because it can be similar to your colleagues' and different from the trainer's.
- Your participation can make our session more interactive and make the topics easier to learn.

Practical lab guidelines

During practical labs...

- We cannot support more than 8 workstations at once (each with its board and equipment). Having more would make the whole class progress slower, compromising the coverage of the whole training agenda (exception for public sessions: up to 10 people).
- So, if you are more than 8 participants, please form up to 8 working groups.
- Open the electronic copy of your lecture materials, and use it throughout the practical labs to find the slides you need again.
- Don't copy and paste from the PDF slides.
  The slides contain UTF-8 characters that look the same as ASCII ones, but won't be understood by shells or compilers.

Cooperate!

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

- 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.
- Explain what you understood to other participants when needed. It also helps to consolidate your knowledge.
- Don't hesitate to report potential bugs to your instructor.
- Don't hesitate to look for solutions on the Internet as well.

Command memento sheet

This memento sheet gives command examples for the most typical needs (looking for files, extracting a tar archive...)

It saves us 1 day of UNIX / Linux command line training.

Our best tip: in the command line shell, always hit the `[Tab]` key to complete command names and file paths. This avoids 95% of typing mistakes.

Get an electronic copy on
http://free-electrons.com/docs/command-line

vi basic commands

vi basic commands

Summary of most useful commands

©Copyright 2009, Free Electrons. http://free-electrons.com Latest update Mar 26, 2009
Free to copy under the terms of the Creative Commons Attribution-ShareAlike 3.0 license (http://creativecommons.org/licenses/by-sa/3.0/)
Sources and updates in the free training material: http://free-electrons.com/docs/command-line, HTML, PDF

Entering command mode

[esc] Exit editing mode. Keyboard keys now interpreted as commands.

Moving the cursor

- h (or left arrow key) move the cursor left.
- l (or right arrow key) move the cursor right.
- j (or up arrow key) move the cursor up.
- k (or down arrow key) move the cursor down.
- [ctrl] f move the cursor one page forward.
- [ctrl] b move the cursor one page backward.
- ^ move the cursor to the beginning of the current line.
- $ move the cursor to the end of the current line.
- G go to the last line in the file.
- nG go to line number n.
- [ctrl] g display the name of the current file and the cursor position in it.

Entering editing mode

- i insert new text before the cursor.
- a append new text after the cursor.
- o start to edit a new line after the current one.
- O start to edit a new line before the current one.

Replacing characters, lines and words

- r replace the current character (does not enter edit mode).
- R enter edit mode and substitute the current character by several ones.
- cw enter edit mode and change the word after the cursor.
- C enter edit mode and change the rest of the line after the cursor.

Copying and pasting

- yy copy (yank) the current line to the copy/paste buffer.
- p paste the copy/paste buffer after the current line.
- P Paste the copy/paste buffer before the current line.

Deleting characters, words and lines

All deleted characters, words and lines are copied to the copy/paste buffer.

- x delete the character at the cursor location.
- dw delete the current word.
- D delete the remainder of the line after the cursor.
- dd delete the current line.

Repeating commands

- . repeat the last insertion, replacement or delete command.

Looking for strings

- /string find the first occurrence of string after the cursor.
- ?string find the first occurrence of string before the cursor.
- n find the next occurrence in the last search.

Replacing strings

Can also be done manually, search and replacing once, and then using n (next occurrence) and . (repeat last edit).

- n,pm/str1/str2/g between line numbers n and p, substitute all (g: global) occurrences of str1 by str2.
- 1,$s/str1/str2/g in the whole file (1: first line, $: last line), substitute all occurrences of str1 by str2.

Applying a command several times - Examples

- 5j move the cursor 5 lines down.
- 30dd delete 30 lines.
- 4cw change 4 words from the cursor.
- 1G go to the first line in the file.

Misc

- [ctrl] l redraw the screen.

Exiting and saving

- zz save current file and exit vi.
- :w write (save) buffer to the current file.
- :w file write (save) buffer to the file file.
- :q! quit vi without saving changes.

Going further

vi has much more flexibility and many more commands for power users!
It can make you extremely productive in editing and creating text.
Learn more by taking the quick tutorial: just type vimtutor.
Many extra resources are also available on the net.

Free Electrons

The `vi` editor is very useful to make quick changes to files in a embedded target.

Though not very user friendly at first, `vi` is very powerful and its main 15 commands are easy to learn and are sufficient for 99% of everyone's needs!

You can also take the quick tutorial by running `vimtutor`.
This is a worthy investment!

Get an electronic copy on
http://free-electrons.com/docs/command-line

Introduction to embedded Linux

Michael Opdenacker
Thomas Petazzoni
Free Electrons

© Copyright 2009-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/embedded-linux-intro
Corrections, suggestions, contributions and translations are welcome!

Birth of free software

- *1983*, Richard Stallman, **GNU project** and the "*free software*" concept. Beginning of the development of gcc, gdb, glibc and other important tools
- *1991*, Linus Torvalds, **Linux kernel project**, a Unix-like operating system kernel. Together with GNU software and many other open-source components: a completely free operating system, GNU/Linux
- *~1995*, Linux is more and more popular on server systems
- *~2000*, Linux is more and more popular on **embedded systems**
- *~2005*, Linux is more and more popular on desktop systems

Free software ?

- A program is considered **free** when its license offers to all its users the following four freedoms
  - Freedom to run the software for any purpose
  - Freedom to study the software and to change it
  - Freedom to redistribute copies
  - Freedom to distribute copies of modified versions
- Those freedoms are granted for both commercial and non-commercial use
- They imply the availability of source code, software can be modified and distributed to customers
- **Good match for embedded systems !**

What is embedded Linux ?

Embedded Linux is the usage of the **Linux kernel** and various **open-source** components in embedded systems

Advantages of Linux and open-source for embedded systems

Re-using components

- The key advantage of Linux and open-source in embedded systems is the **ability to re-use components**
- The open-source ecosystem already provides many components for standard features, from hardware support to network protocols, going through multimedia, graphic, cryptographic libraries, etc.
- As soon as a hardware device, or a protocol, or a feature is wide-spread enough, high chance of having open-source components that support it.
- Allows to quickly design and develop complicated products, based on existing components.
- No-one should re-develop yet another operating system kernel, TCP/IP stack, USB stack or another graphical toolkit library.
- **Allows to focus on the added value of your product.**

Low cost

- Free software can be duplicated on as many devices as you want, free of charge.
- If your embedded system uses only free software, you can reduce the cost of software to zero. Even the development tools are free, unless you choose a commercial embedded Linux edition.
- **Allows to have an higher budget for the hardware or to increase the company's skills and knowledge**

Full control

- With open-source, you have the source code for all components in your system
- Allows unlimited modifications, changes, tuning, debugging, optimization, for an unlimited period of time
- Without locking or dependency from a third-party vendor
  - To be true, non open-source components must be avoided when the system is designed and developed
- **Allows to have full control over the software part of your system**

Quality

- Many open-source components are widely used, on millions of systems
- Usually higher quality than what an in-house development can produce, or even proprietary vendors
- Of course, not all open-source components are of good quality, but most of the widely-used ones are.
- **Allows to design your system with high-quality components at the foundations**

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Eases testing of new features

- Open-source being freely available, it is easy to get one and evaluate it
- Allows to easily study several options while making a choice
- Much easier than purchasing and demonstration procedures needed with most proprietary products
- **Allows to easily explore new possibilities and solutions**

Community support

- Open-source software components are developed by communities of developers and users
- This community can provide a high-quality support: you can directly contact the main developers of the component you are using
- Often better than traditional support, but one needs to understand how the community works to properly use the community support possibilities
- **Allows to speed up the resolution of problems when developing your system**

Taking part into the community

- Possibility of taking part into the development community of some of the components used in the embedded systems: bug reporting, test of new versions or features, patches that fix bugs or add new features, etc.
- Most of the time the open-source components are not the core value of the product: it's the interest of everybody to contribute back.
- For the *engineers*: a very **motivating** way of being recognized outside the company, communication with others in the same field, **opening of new possibilities**, etc.
- For the *managers*: **motivation factor** for engineers, allows the company to be **recognized** in the open-source community and therefore get support more easily and be **more attractive** to open-source developers

A few examples of embedded systems running Linux

Personal routers

Television

Point of sale terminal

Laser cutting machine

Viticulture machine

Embedded hardware for Linux systems

Processor and architecture (1)

- The Linux kernel and most other architecture-dependent component support a wide range of 32 and 64 bits architectures
  - x86 and x86-64, as found on PC platforms, but also embedded systems (multimedia, industrial)
  - ARM, with hundreds of different SoC (multimedia, industrial)
  - PowerPC (mainly real-time, industrial applications)
  - MIPS (mainly networking applications)
  - SuperH (mainly set top box and multimedia applications)
  - Blackfin (DSP architecture)
  - Microblaze (soft-core for Xilinx FPGA)
  - Coldfire, SCore, Tile, Xtensa, Cris, FRV, AVR32, M32R

Processor and architecture (2)

- Both MMU and no-MMU architectures are supported, even though no-MMU architectures have a few limitations.
- Linux is not designed for small microcontrollers.
- Besides the toolchain, the bootloader and the kernel, all other components are generally **architecture-independent**

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

RAM and storage

- **RAM**: a very basic Linux system can work within 8 MB of RAM, but a more realistic system will usually require at least 32 MB of RAM. Depends on the type and size of applications.
- **Storage**: a very basic Linux system can work within 4 MB of storage, but usually more is needed.
  - Flash storage is supported, both NAND and NOR flash, with specific filesystems
  - Block storage including SD/MMC cards and eMMC is supported
- Not necessarily interesting to be too restrictive on the amount of RAM/storage: having flexibility at this level allows to re-use as many existing components as possible.

Communication

- The Linux kernel has support for many common communication busses
  - I2C
  - SPI
  - CAN
  - 1-wire
  - SDIO
  - USB
- And also extensive networking support
  - Ethernet, Wifi, Bluetooth, CAN, etc.
  - IPv4, IPv6, TCP, UDP, SCTP, DCCP, etc.
  - Firewalling, advanced routing, multicast

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Types of hardware platforms

- **Evaluation platforms** from the SoC vendor. Usually expensive, but many peripherals are built-in. Generally unsuitable for real products.
- **Component on Module**, a small board with only CPU/RAM/flash and a few other core components, with connectors to access all other peripherals. Can be used to build end products for small to medium quantities.
- **Community development platforms**, a new trend to make a particular SoC popular and easily available. Those are ready-to-use and low cost, but usually have less peripherals than evaluation platforms. To some extent, can also be used for real products.
- **Custom platform**. Schematics for evaluation boards or development platforms are more and more commonly freely available, making it easier to develop custom platforms.

Criteria for choosing the hardware

- Make sure the hardware you plan to use is already supported by the Linux kernel, and has an open-source bootloader, especially the SoC you're targeting.
- Having support in the official versions of the projects (kernel, bootloader) is a lot better: quality is better, and new versions are available.
- Some SoC vendors and/or board vendors do not contribute their changes back to the mainline Linux kernel. Ask them to do so, or use another product if you can. A good measurement is to see the delta between their kernel and the official one.
- **Between properly supported hardware in the official Linux kernel and poorly-supported hardware, there will be huge differences in development time and cost.**

Embedded Linux system architecture

Global architecture

Software components

- Cross-compilation toolchain
  - Compiler that runs on the development machine, but generates code for the target
- Bootloader
  - Started by the hardware, responsible for basic initialization, loading and executing the kernel
- Linux Kernel
  - Contains the process and memory management, network stack, device drivers and provides services to userspace applications
- C library
  - The interface between the kernel and the userspace applications
- Libraries and applications
  - Third-party or in-house

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Embedded Linux work

Several distinct tasks are needed when deploying embedded Linux in a product

- **Board Support Package development**
  - A BSP contains a bootloader and kernel with the suitable device drivers for the targeted hardware
  - Purpose of our « Kernel Development » training
- **System integration**
  - Integrate all the components, bootloader, kernel, third-party libraries and applications and in-house applications into a working system
  - Purpose of this training
- **Development of applications**
  - Normal Linux applications, but using specifically chosen libraries

Embedded Linux development environment

Michael Opdenacker
Thomas Petazzoni
Free Electrons

© Copyright 2009-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/embedded-linux-intro
Corrections, suggestions, contributions and translations are welcome!

Embedded Linux solutions

- Two ways to switch to embedded Linux
  - Use **solutions provided and supported by vendors** like MontaVista, Wind River or TimeSys. These solutions come with their own development tools and environment. They use a mix of open-source components and proprietary tools.
  - Use **community solutions**. They are completely open, supported by the community.
- In Free Electrons training sessions, we do not promote a particular vendor, and therefore use community solutions
  - However, knowing the concepts, switching to vendor solutions will be easy

OS for Linux development

- We strongly recommend to use Linux as the desktop operating system to embedded Linux developers, for multiple reasons.
- All community tools are developed and designed to run on Linux. Trying to use them on other operating systems (Windows, Mac OS X) will lead to trouble, and their usage on those systems is generally not supported by community developers.
- As Linux also runs on the embedded device, all the knowledge gained from using Linux on the desktop will apply similarly to the embedded device.

Desktop Linux distribution

- **Any good and sufficiently recent Linux desktop distribution** can be used for the development workstation
  - Ubuntu, Debian, Fedora, openSUSE, Red Hat, etc.
- We have chosen Ubuntu, as it is a **widely used and easy to use** desktop Linux distribution
- The Ubuntu setup on the training laptops has intentionally been left untouched after the normal installation process. Learning embedded Linux is also about learning the tools needed on the development workstation!

Linux root and non-root users

- Linux is a multi-user operating system
  - The **root user is the administrator**, and it can do privileged operations such as: mounting filesystems, configuring the network, creating device files, changing the system configuration, installing or removing software
  - All **other users are unprivileged**, and cannot perform those administrator-level operations
- On an Ubuntu system, it is not possible to log in as `root`, only as a normal user.
- The system has been configured so that the user account created first is allowed to run privileged operations through a program called `sudo`.
  - Example: `sudo mount /dev/sda2 /mnt/disk`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Software packages

- The distribution mechanism for software in GNU/Linux is different from the one in Windows
- Linux distributions provides a central and coherent way of installing, updating and removing applications and libraries: **packages**
- Packages contains the application or library files, and associated meta-information, such as the version and the dependencies
  - .deb on Debian and Ubuntu, .rpm on Mandriva, Fedora, openSUSE
- Packages are stored in **repositories**, usually on HTTP or FTP servers
- One should only use packages from official repositories of its distribution, unless strictly required.

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>`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`
- Get information about a package:
  `apt-cache show <package>`
- Graphical interfaces
  - Synaptic for GNOME
  - KPackageKit for KDE

Further details on package management:
http://www.debian.org/doc/manuals/apt-howto/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Host vs. target

- When doing embedded development, there is always a split between
  - The “host”, the development workstation, which is typically a powerful PC
  - The “target”, which is the embedded system under development
- They are connected by various means: almost always a serial line for debugging purposes, frequently an Ethernet connection, sometimes a JTAG for low-level debugging

Serial line communication program

- An essential tool for embedded development is a serial line communication program, like HyperTerminal in Windows.
- There are multiple options available in Linux: Minicom, Picocom, Gtkterm, Putty, etc.
- In this training session, we recommend using the simplest of them: `picocom`
  - Installation with `sudo apt-get install picocom`
  - Run with `picocom -b BAUD_RATE /dev/SERIAL_DEVICE`
  - Exit with `Control-A Control-X`
- `SERIAL_DEVICE` is typically
  - `ttyUSBx` for USB to serial converters
  - `ttySx` for real serial ports

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Command line tips

- Using the command line is mandatory for many operations needed for embedded Linux development
- It is a very powerful way of interacting with the system, with which you can save a lot of time.
- Some useful tips
  - You can use several tabs in the Gnome Terminal
  - Remember that you can use relative paths (for example: `../../linux`) in addition to absolute paths (for example: `/home/user`)
  - In a shell, hit `[Control] [r]`, then a keyword, will search through the command history. Hit `[Control] [r]` again to search backwards in the history
  - You can copy/paste paths directly from the file manager to the terminal by drag-and-drop.

Training setup

Download files and directories used in practical labs

Update your system

Time might have elapsed since your system was last updated.

Keep your system up to date:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Install lab data

For the different labs in the training, your instructor has prepared a set of data (kernel images, kernel configurations, root filesystems and more). Download the tarball at http://free-electrons.com/labs/embedded_linux.tar.bz2.

Then, from a terminal, extract the tarball using the following command:

```
cd      (going to your home directory)
sudo tar jxf embedded_linux.tar.bz2
sudo chown -R <user>.<user> felabs
```

`root` permissions are required to extract the character and block device files contained in the lab structure.

Lab data are now available in a `felabs` directory in your home directory. For each lab there is a directory containing various data. This directory can also be used as a working space for each lab so that you properly keep the work on each lab well-separated.

Exit Synaptic if it is still open. If you don't, you won't be able to run `apt-get install` commands, because only one package management tool is allowed at a time.

You are now ready to start the real practical labs!

Install extra packages

Ubuntu comes with a very limited version of the vi editor. Install vim, a improved version of this editor.

```
sudo apt-get install vim
```

More guidelines

Can be useful throughout any of the labs

- Read instructions and tips carefully. Lots of people make mistakes or waste time because they missed an explanation or a guideline.
- Always read error messages carefully, in particular the first one which is issued. Some people stumble on very simple errors just because they specified a wrong file path and didn't pay enough attention to the corresponding error message.
- Never stay stuck with a strange problem more than 5 minutes. Show your problem to your colleagues or to the instructor.
- You should only use the `root` user for operations that require super-user privileges, such as: mounting a file system, loading a kernel module, changing file ownership, configuring the network. Most regular tasks (such as downloading, extracting

© 2004-2011 Free Electrons, http://free-electrons.com Creative Commons License

sources, compiling...) can be done as a regular user.

- If you ran commands from a root shell by mistake, your regular user may no longer be able to handle the corresponding generated files. In this case, use the `chown -R` command to give back the new files to your regular user.
  Example: `chown -R myuser.myuser linux-2.6.25`
- In Debian, Ubuntu and other derivatives, don't be surprised if you cannot run graphical applications as `root`. You could set the `DISPLAY` variable to the same setting as for your regular user, but again, it's unnecessary and unsafe to run graphical applications as `root`.

Cross-compiling toolchains

Thomas Petazzoni
Michael Opdenacker
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/toolchains
Corrections, suggestions, contributions and translations are welcome!

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Definition (2)

Compilation machine

Execution machine

Components

- Binutils
- Kernel headers
- C/C++ libraries
- GCC compiler
- GDB debugger (optional)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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;
        [...]
};
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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: approx 2.5 MB on `arm` (version 2.9 - `libc`: 1.5 MB, `libm`: 750 KB)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

uClibc

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

uClibc (2)

- Most of the applications compile with uClibc. This applies to all applications used in embedded systems.
- Size (`arm`): 4 times smaller than glibc!
  - uClibc 0.9.30.1: approx. 600 KB (`libuClibc`: 460 KB, `libm`: 96KB)
  - glibc 2.9: approx 2.5 MB
- Some features not available or limited: priority-inheritance mutexes, NPTL support is very new, fixed Name Service Switch functionality, etc.
- Used on a large number of production embedded products, including consumer electronic devices
- Actively maintained, large developer and user base
- Supported and used by MontaVista, TimeSys and Wind River.

Honey, I shrunk the programs!

C program	*Compiled with shared libraries*		*Compiled statically*	
	glibc	*uClibc*	*glibc*	*uClibc*
Plain “hello world” (stripped)	5.6 K (glibc 2.9)	5.4 K (uClibc 0.9.30.1)	472 K (glibc 2.9)	18 K (uClibc 0.9.30.1)
Busybox (stripped)	245 K (older glibc)	231 K (older uClibc)	843 K (older glibc)	311 K (older uClibc)

Executable size comparison on ARM

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.
- The Debian distribution is switching to eglibc too: http://blog.aurel32.net/?p=47
- http://www.eglibc.org

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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://sourceware.org/newlib/
  - Klibc, http://www.kernel.org/pub/linux/libs/klibc/, designed for use in an initramfs or initrd at boot time.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Machines in build procedures

- 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 are executed.
- Four common build types are possible for toolchains

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Different toolchain build procedures

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

The most common solution in embedded

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

ABI

- When building a toolchain, the ABI used to generate binaries needs to be defined
- ABI, for Application Binary Interface, defines the calling conventions (how function arguments are passed, how the return value is passed, how system calls are made) and the organization of structures (alignement, etc.)
- All binaries in a system must be compiled with the same ABI, and the kernel must understand this ABI.
- On ARM, two main ABIs: OABI and EABI
  - Nowadays everybody uses EABI
- On MIPS, several ABI: o32, o64, n32, n64
- http://en.wikipedia.org/wiki/Application_Binary_Interface

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Floating point support

- Some processors have a floating point unit, some others do not.
  - For example, many ARMv4 and ARMv5 CPUs do not have a floating point unit. Since ARMv7, a VFP unit is mandatory.
- For processors having a floating point unit, the toolchain should generate « hard float » code, in order to use the floating point instructions directly
- For processors without a floating point unit, two solutions
  - Generate « hard float code » and rely on the kernel to emulate the floating point instructions. This is very slow.
  - Generate « soft float code », so that instead of generating floating point instructions, calls to an userspace library are generated
- Decision taken at toolchain configuration time
- Also possible to configure which floating point unit should be used

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

CPU optimization flags

- A set of cross-compiling tools is specific to a CPU architecture (ARM, x86, MIPS, PowerPC)
- However, with the `-march=`, `-mcpu=`, `-mtune=` options, one can select more precisely the targetted CPU type
  - For example, `-march=armv7 -mcpu=cortex-a8`
- At the toolchain compilation time, values can be chosen. They are used:
  - As the default values for the cross-compiling tools, when no other `-march, -mcpu, -mtune` options are passed
  - To compile the C library
- Even if the C library has been compiled for armv5t, it doesn't prevent from compiling other programs for armv7

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Building a toolchain manually

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

- Lots of details to learn: many components to build, complicated configuration
- Lots of decisions to make (such as C library version, ABI, floating point mechanisms, component versions)
- 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
- See the Crosstool-NG `docs/` directory for details on how toolchains are built.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Get a precompiled toolchain

- Solution that many people choose
  - Advantage: it is the simplest and most convenient solution
  - Drawback: you can't fine tune the toolchain to your needs
- Determine what toolchain you need: CPU, endianism, C library, component versions, ABI, soft float or hard float, etc.
- Check whether the available toolchains match your requirements.
- Possible choices
  - CodeSourcery toolchains
  - Linaro toolchains
  - More references at http://elinux.org/Toolchains

CodeSourcery

- Company that has an extended expertise on free software toolchains: `gcc`, `gdb`, `binutils` and `glibc`
- They sell toolchains with support, but they also provide a “Lite” version, which is free and usable for commercial products
- They have toolchains available for
  - ARM
  - MIPS
  - PowerPC
  - SuperH
- Be sure to use the GNU/Linux versions. The EABI versions are for bare-metal development (no operating system)

Linaro toolchains

https://wiki.linaro.org/WorkingGroups/ToolChain

- Linaro contributes to improving mainline gcc on ARM, in particular by hiring CodeSourcery developers.
- For people who can't wait for the next releases of gcc, Linaro releases modified sources of stable releases of gcc, with these optimizations for ARM (mainly for recent Cortex A CPUs).
- As any gcc release, these sources can be used by build tools to build their own binary toolchains (Buildroot, OpenEmbedded...) This allows to support glibc, uClibc and eglibc.
- Binary packages are available for Ubuntu users: See https://launchpad.net/~linaro-maintainers/+archive/toolchain

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Installing and using a precompiled toolchain

- Follow the installation procedure proposed by the vendor
- Usually, it is simply a matter of extracting a tarball
  - Old toolchains (gcc 3.x based) were not relocatable, they had to be installed. You must install them in the location they were built for.
  - Now you can install the toolchains wherever you want.
- Then, add the path to toolchain binaries in your `PATH`:
  `export PATH=/path/to/toolchain/bin/:$PATH`
- Finally, compile your applications
  `PREFIX-gcc -o foobar foobar.c`
- The PREFIX depends on the toolchain configuration, and allows to distinguish cross-compilation tools from native compilation utilities

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 maintained anymore, don't use it!
  - 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://crosstool-ng.org/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Toolchain building utilities (3)

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

- Buildroot
  - Makefile-based, has a Crosstool-NG backend, maintained by the community
  - http://www.buildroot.net
- 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/

Crosstool-NG: installation and usage

- Installation of Crosstool-NG can be done system-wide, or just locally in the source directory. For local installation:
```
./configure --local
make
make install
```
- Some sample configurations for various architectures are available in samples, they can be listed using
```
./ct-ng list-samples
```
- To load a sample configuration
```
./ct-ng <sample-name>
```
- To adjust the configuration
```
./ct-ng menuconfig
```
- To build the toolchain
```
./ct-ng build
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Toolchain contents

- The cross compilation tool binaries, in `bin/`
  - This directory can be added to your PATH to ease usage of the toolchain
- One or several *sysroot*, each containing
  - The C library and related libraries, compiled for the target
  - The C library headers and kernel headers
- There is one *sysroot* for each variant: toolchains can be *multilib* if they have several copies of the C library for different configurations (for example: ARMv4T, ARMv5T, etc.)
  - CodeSourcery ARM toolchain are multilib, the sysroots are in `arm-none-linux-gnueabi/libc/`, `arm-none-linux-gnueabi/libc/armv4t/`, `arm-none-linux-gnueabi/libc/thumb2`
  - Crosstool-NG toolchains are never multilib, the sysroot is in `arm-unknown-linux-uclibcgnueabi/sysroot`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Practical lab – Using Crosstool-NG

Time to build your toolchain

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Bootloaders

Thomas Petazzoni
Michael Opdenacker
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/bootloaders
Corrections, suggestions, contributions and translations are welcome!

Bootloaders

- The bootloader is a piece of code responsible for
  - Basic hardware initialization
  - Loading of an application binary, usually an operating system kernel, from flash storage, 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 non-volatile storage.
  - This piece of code is usually the first 512 bytes of a storage device
- 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.
  - See our dedicated presentation for details:
    http://free-electrons.com/docs/grub/
- Syslinux, for network and removable media booting
  http://www.kernel.org/pub/linux/utils/boot/syslinux/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Booting on embedded CPUs: case 1

- When powered, the CPU starts executing code at a fixed address
- There is no other booting mechanism provided by the CPU
- The hardware design must ensure that a NOR flash chip is wired so that it is accessible at the address at which the CPU starts executing instructions
- The first stage bootloader must be programmed at this address in the NOR
- NOR is mandatory, because it allows random access, which NAND doesn't allow
- **Not very common anymore** (unpractical, and requires NOR flash)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Booting on embedded CPUs: case 2

- The CPU has an integrated boot code in ROM
  - BootROM on AT91 CPUs, “ROM code” on OMAP, etc.
  - Exact details are CPU-dependent
- This boot code is able to load a first stage bootloader from a storage device into an internal SRAM (DRAM not initialized yet)
  - Storage device can typically be: MMC, NAND, SPI Flash, UART, etc.
- The first stage bootloader is
  - Limited in size due to hardware constraints (SRAM size)
  - Provided either by the CPU vendor of through community projects
- This first stage bootloader must initialize DRAM and other hardware devices and load a second stage bootloader into RAM

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Booting on ARM Atmel AT91

RomBoot
Stored in ROM

Tries to find a valid bootstrap image from various storage sources, and load it into SRAM (DRAM not initialized yet). Size limited to 4 KB. No user interaction possible in standard boot mode.

AT91Bootstrap
Stored in NAND or SPI Flash
Runs in SRAM

Runs from SRAM. Initializes the DRAM, the NAND or SPI controller, and loads the secondary bootloader into RAM and starts it. No user interaction possible.

U-Boot
Stored in NAND or SPI Flash
Runs in RAM

Runs from RAM. Initializes some other hardware devices (network, USB, etc.). Loads the kernel image from storage or network to RAM and starts it. Shell with commands provided.

Linux Kernel
Stored in NAND
Runs in RAM

Runs from RAM. Takes over the system completely (bootloaders no longer exists).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Booting on ARM OMAP3

ROM Code
Stored in ROM

Tries to find a valid bootstrap image from various storage sources, and load it into SRAM or RAM (RAM can be initialized by ROM code through a configuration header). Size limited to < 64 KB. No user interaction possible.

X-Loader
Stored in MMC or NAND
Runs in SRAM

Runs from SRAM. Initializes the DRAM, the NAND or MMC controller, and loads the secondary bootloader into RAM and starts it. No user interaction possible. File called "`MLO`".

U-Boot
Stored in MMC or NAND
Runs in RAM

Runs from RAM. Initializes some other hardware devices (network, USB, etc.). Loads the kernel image from storage or network to RAM and starts it. Shell with commands provided. File called "`u-boot.bin`".

Linux Kernel
Stored in MMC or NAND
Runs in RAM

Runs from RAM. Takes over the system completely (bootloaders no longer exists).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Generic bootloaders for embedded CPUs

- We will focus on the generic part, the main bootloader, offering the most important features.
- There are several open-source generic bootloaders.
  Here are the most popular ones:
  - U-Boot, the universal bootloader by Denx
    The most used on ARM, also used on PPC, MIPS, x86, m68k, NIOS, etc.
    The de-facto standard nowadays. We will study it in detail.
    http://www.denx.de/wiki/U-Boot
  - Barebox, a new architecture-neutral bootloader, written as a successor of U-Boot. Better design, better code, active development, but doesn't yet have as much hardware support as U-Boot.
    http://www.barebox.org
- There are also a lot of other open-source or proprietary bootloaders, often architecture-specific
  - RedBoot, Yaboot, PMON, etc.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

The U-boot bootloader

Michael Opdenacker
Thomas Petazzoni
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/u-boot
Corrections, suggestions, contributions and translations are welcome!

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

U-Boot configuration

- 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/igep0020.h`
- This file can also be adjusted for add or remove features from U-Boot

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

U-Boot configuration file excerpt

```
#define CONFIG_ARMV7            1
#define CONFIG_OMAP             1
#define CONFIG_OMAP34XX         1
#define CONFIG_OMAP3430         1
#define CONFIG_OMAP3_IGEP0020   1
[...]
#define CONFIG_NR_DRAM_BANKS    2
#define PHYS_SDRAM_1            OMAP34XX_SDRC_CS0
#define PHYS_SDRAM_1_SIZE       (32 << 20)
#define PHYS_SDRAM_2            OMAP34XX_SDRC_CS1
[...]
/* USB */
#define CONFIG_MUSB_UDC         1
#define CONFIG_USB_OMAP3        1
#define CONFIG_TWL4030_USB      1
[...]
#define CONFIG_CMD_CACHE
#define CONFIG_CMD_EXT2
#define CONFIG_CMD_FAT
#define CONFIG_CMD_I2C
#define CONFIG_CMD_MMC
#define CONFIG_CMD_ONENAND
#define CONFIG_CMD_NET
#define CONFIG_CMD_DHCP
#define CONFIG_CMD_PING
#define CONFIG_CMD_NFS
#define CONFIG_CMD_MTDPARTS
```

CPU configuration

Memory configuration

USB configuration

Available commands and features

Configuring and compiling U-Boot

- U-Boot must be configured before being compiled
  - `make BOARDNAME_config`
  - Where `BOARDNAME` is usually the name of the configuration file in `include/configs/`, without the `.h`
- Make sure that the cross-compiler is available in `PATH`
- Compile U-Boot, by specifying the cross-compiler prefix. Example, if your cross-compiler executable is `arm-linux-gcc`: `make CROSS_COMPILE=arm-linux-`
- The result is a `u-boot.bin` file, which is the U-Boot image

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Installing U-Boot

- U-Boot must usually be installed in flash memory to be executed by the hardware. Depending on the hardware, the installation of U-Boot is done in a different way:
  - The CPU provides some kind of specific boot monitor with which you can communicate through serial port or USB using a specific protocol
  - The CPU boots first on removable media (MMC) before booting from fixed media (NAND). In this case, boot from MMC to reflash a new version
  - 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 2010.06-2 (May 13 2011 – 12:13:22)
OMAP3630/3730-GP ES2.0, CPU-OPP2, L3-165MHz
IGEP v2 board + LPDDR/ONENAND
I2C:   ready
DRAM:  512 MiB
Muxed OneNAND(DDP) 512MB 1.8V 16-bit (0x58)
OneNAND version = 0x0031
Chip support all block unlock
OneNAND: 512 MiB
OneNAND: Read environment from 0x00200000
In:    serial
Out:   serial
Err:   serial
Die ID #4d5400011ff0000001592f350202c01d
Net:   smc911x-0
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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Information commands

Flash information (NOR)

```
U-Boot> flinfo
DataFlash:AT45DB021
Nb pages:   1024
Page Size:    264
Size=  270336 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0001FFF (RO) Bootstrap
Area 1: C0002000 to C0003FFF      Environment
Area 2: C0004000 to C0041FFF (RO) U-Boot
```

NAND flash information

```
U-Boot> nand info
Device 0: NAND 256MiB 3,3V 8-bit, sector size 128 KiB
```

U-Boot information

```
U-Boot> version
U-Boot 2009.08 (Nov 15 2009 - 14:48:35)
```

Can vary from one board to the other
(according to the U-Boot compile configuration)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Important commands (1)

- The exact set of commands depends on the U-Boot configuration
- `help` and `help command`
- `boot`, runs the default boot command, stored in `bootcmd`
- `bootm <address>`, starts a kernel image loaded at the given address in RAM
- `ext2load`, load a file from an ext2 filesystem to RAM
  - And also `ext2ls` to list files, `ext2info` for information
- `fatload`, load a file from a FAT filesystem to RAM
  - And also `fatls` and `fatinfo`
- `tftp`, load a file from the network to RAM
- `ping`, to test the network

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Important commands (2)

- `loadb, loads, loady`, load a file from the serial line to RAM
- `usb`, to initialize and control the USB subsystem, mainly used for USB storage devices such as USB keys
- `mmc`, to initialize and control the MMC subsystem, used for SD and microSD cards
- `nand`, to erase, read and write contents to the NAND
- `erase, protect, cp`, to erase, modify protection and write to a NOR flash
- `md`, display memory contents. Can be useful to check the contents loaded in memory, or to look at hardware registers.
- `mm`, modify memory contents. Can be useful to modify directly hardware registers, for testing purposes.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Environment variables commands

- U-Boot can be configured through environment variables, which affect the behavior of the different commands.
- Environment variables are loaded from Flash to RAM at U-Boot startup, can be modified and saved back to Flash for persistence
- There is a dedicated location in Flash to store U-Boot environment, defined in the board configuration file
- Commands to manipulate environment variables:
  - `printenv`, show all variables
  - `printenv <variable-name>`, show value of one variable
  - `setenv <variable-name> <variable-value>`, change the value of a variable, only in RAM
  - `saveenv`, save to Flash the current state of the environment

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Environment variables commands (2)

```
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

u-boot # printenv serverip
serverip=10.0.0.2

u-boot # setenv serverip 10.0.0.100
u-boot # saveenv
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Important U-Boot env variables

- **`bootcmd`**, contains the command that U-Boot will automatically execute at boot time after a configurable delay, if the process is not interrupted
- **`bootargs`**, contains the arguments passed to the Linux kernel, covered later
- **`serverip`**, the IP address of the server that U-Boot will contact for network related commands
- **`ipaddr`**, the IP address that U-Boot will use
- **`netmask`**, the network mask to contact the server
- **`ethaddr`**, the MAC address, can only be set once
- **`bootdelay`**, the delay in seconds before which U-Boot runs bootcmd
- **`autostart`**, if yes, U-Boot starts automatically an image that has been loaded into memory

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Scripts in environment variables

- Environment variables can contain small scripts, to execute several commands and test the results of commands.
  - Useful to automate booting or upgrade processes
  - Several command can be chained using the `;` operator
  - Tests can be done using `if command ; then … ; else … ; fi`
  - Scripts are executed using `run <variable-name>`
  - You can reference other variables using `${variable-name}`
- Example
  - `setenv mmc-boot 'mmc init 0; if fatload mmc 0 80000000 boot.ini; then source; else if fatload mmc 0 80000000 uImage; then run mmc-bootargs; bootm; fi; fi'`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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. This is the fastest and most efficient solution.
  - Through an USB key, if U-Boot support the USB controller of your platform
  - Through a SD or microSD card, if U-Boot supports the MMC controller of your platform
  - Through the serial port

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

TFTP

- Network transfer from the development workstation and U-Boot on the target takes place through TFTP
  - *Trivial File Transfer Protocol*
  - Somewhat similar to FTP, but without authentication and over UDP
- A TFTP server is needed on the development workstation
  - `sudo apt-get install tftpd-hpa`
  - All files in `/var/lib/tftpboot` are then visible through TFTP
  - A TFTP client is available in the `tftp-hpa` package, for testing
- A TFTP client is integrated into U-Boot
  - Configure the `ipaddr` and `serverip` environment variables
  - Use `tftp <address> <filename>` to load a file

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 header gives details such as the image size, the expected load address, the compression type, etc.
- 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Flashing a kernel image

- Compile your kernel and generate the U-Boot header running `make uImage`
- 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Practical lab – U-Boot

Time to start the practical lab !

- Communicate with the board using a serial console
- Configure, build and install X-Loader and U-Boot
- Learn U-Boot commands
- Set up TFTP communication with the board

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux kernel introduction

Michael Opdenacker
Thomas Petazzoni
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/kernel-intro
Corrections, suggestions, contributions and translations are welcome!

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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..

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

3.0 status

- See the `arch/` directory in the kernel sources
- Minimum: 32 bit processors, with or without MMU, and gcc support
- 32 bit architectures (`arch/` subdirectories)
  `arm`, `avr32`, `blackfin`, `cris`, `frv`, `h8300`, `m32r`, `m68k`, `microblaze`, `mips`, `mn10300`, `parisc`, `s390`, `score`, `sparc`, `um`, `unicore32`, `xtensa`
- 64 bit architectures:
  `alpha`, `ia64`, `sparc64`, `tile`
- `32/64 bit architectures`
  `powerpc`, `x86`, `sh`
- Find details in kernel sources: `arch/<arch>/Kconfig`, `arch/<arch>/README`, or `Documentation/<arch>/`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

System calls

- The main interface between the kernel and userspace is the set of system calls
- About ~300 system calls that provides the main kernel services
  - File and device operations, networking operations, inter-process communication, process management, memory mapping, timers, threads, synchronization primitives, etc.
- This interface is stable over time: only new system calls can be added by the kernel developers
- This system call interface is wrapped by the C library, and userspace applications usually never make a system call directly but rather use the corresponding C library function

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Virtual filesystems

- Linux makes system and kernel information available in user-space through virtual filesystems.
- Virtual filesystems allow applications to see directories and files that do not exist on any real storage: they are created on the fly by the kernel
- The two most important virtual filesystems are
  - `proc`, for process-related information
  - `sysfs`, for device-related information

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Until 2.6 (2)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.
- So far, there was no need to create a new development branch (such as `2.9`), which would massively break compatibility with the stable branch.
- Thanks to this, more features are released to users at a faster pace.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Merge and bug fixing windows

More stability for the 2.6 kernel tree

- Issue: bug and security fixes only released for most recent stable kernel versions.
- Some people need to have a recent kernel, but with long term support for security updates.
- You could get long term support from a commercial embedded Linux provider.
- You could reuse sources for the kernel used in Ubuntu Long Term Support releases (5 years of free security updates).
- You could choose one of the versions advertised as “long term” in the http://kernel.org front page. They will be maintained longer (2 or 3 years), unlike other versions.

linux-next:	**next-20110118**
snapshot:	**2.6.37-git18**
mainline:	**2.6.37**
stable:	**2.6.37**
stable:	**2.6.36.3**
longterm:	**2.6.35.10**
stable:	**2.6.35.9**
longterm:	**2.6.34.8**
stable:	**2.6.34.7**
stable:	**2.6.33.7**
longterm:	**2.6.32.28**
stable:	**2.6.32.28**
longterm:	**2.6.27.57**
stable:	**2.6.27.57**
stable:	**2.4.37.11**

New 3.x branch

- From 2003 to 2011, the official kernel versions were named 2.6.x.
- Linux 3.0 was released in July 2011
- There is no change to the development model, only a change to the numbering scheme
  - Official kernel versions will be named 3.x (3.0, 3.1, 3.2, etc.)
  - Stabilized versions will be named 3.x.y (3.0.2, 3.4.3, etc.)
  - It effectively only removes a digit compared to the previous numbering scheme

What's new in each Linux release?

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

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

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

    Bug was introduced with the TLB flush filter fix for i386

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

- The official list of changes for each Linux release is just a huge list of individual patches!
  - Very difficult to find out the key changes and to get the global picture out of individual changes.
- Fortunately, there are some useful resources available
  - http://wiki.kernelnewbies.org/LinuxChanges
  - http://lwn.net
  - http://linuxfr.org, for French readers

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Embedded Linux kernel usage

Michael Opdenacker
Thomas Petazzoni
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/kernel-usage
Corrections, suggestions, contributions and translations are welcome!

Compiling and booting Linux

Linux kernel sources

Location of kernel sources

- The official version of the Linux kernel, as released by Linus Torvalds is available at http://www.kernel.org
  - This version follows the well-defined development model of the kernel
  - However, it may not contain the latest development from a specific area, due to the organization of the development model and because features in development might not be ready for mainline inclusion
- Many kernel sub-communities maintain their own kernel, with usually newer but less stable features
  - Architecture communities (ARM, MIPS, PowerPC, etc.), device drivers communities (I2C, SPI, USB, PCI, network, etc.), other communities (real-time, etc.)
  - They generally don't release official versions, only development trees are available

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux kernel size (1)

- Linux 2.6.37 sources:
  Raw size: 412 MB (37,300 files, approx 14,000,000 lines)
  `gzip` compressed tar archive: 89 MB
  `bzip2` compressed tar archive: 71 MB (better)
  `lzma` compressed tar archive: 61 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!

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux kernel size (2)

Getting Linux sources

- Full tarballs
  - Contain the complete kernel sources
  - Long to download and uncompress, but must be done at least once
  - Example:
    http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.38.7.tar.bz2
- Incremental patches between versions
  - It assumes you already have a base version and you apply the correct patches in the right order
  - Quick to download and apply
  - Examples
    http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.38.bz2 (2.6.37 to 2.6.38)
    http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.38.7.bz2 (2.6.38 to 2.6.38.7)
- All previous kernel versions are available in
  http://kernel.org/pub/linux/kernel/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Patch

- A patch is the difference between two source trees
  - Computed with the `diff` tool, or with more elaborate version control systems
- They are very common in the open-source community
- Excerpt from a patch :

```
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*
```

File being modified

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Using the patch command

The `patch` command :

- Takes the patch contents on its standard input
- Applies the modifications described by the patch into the current directory

`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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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!)
- Need to run the `patch` command inside the kernel source directory
- Linux patch command line example:

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


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Practical lab – Kernel sources

- Get the Linux kernel sources
- Apply patches

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Compiling and booting Linux

Kernel configuration

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel configuration and build system

- The kernel configuration and build system is based on multiple Makefiles
- One only interacts with the main `Makefile`, present at the **top directory** of the kernel source tree
- Interaction takes place
  - using the `make` tool, which parses the Makefile
  - through various targets, defining which action should be done (configuration, compilation, installation, etc.). Run `make help` to see all available targets.
- Example
  - `cd linux-2.6.x/`
  - `make <target>`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel configuration (1)

- The kernel contains thousands of device drivers, filesystem drivers, network protocols and other configurable items
- Thousands of options are available, that are used to selectively compile parts of the kernel source code
- The kernel configuration is the process of defining the set of options with which you want your kernel to be compiled
- The set of options depends
  - On your hardware (for device drivers, etc.)
  - On the capabilities you would like to give to your kernel (network capabilities, filesystems, real-time, etc.)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel configuration (2)

- The configuration is stored in the `.config` file at the root of kernel sources
  - Simple text file, `key=value` style
- As options have dependencies, typically never edited by hand, but through graphical or text interfaces :
  - `make xconfig`, `make gconfig` (graphical)
  - `make menuconfig`, `make nconfig` (text)
  - You can switch from one to another, they all load/save the same `.config` file, and show the same set of options
- To modify a kernel in a GNU/Linux distribution:
  the configuration files are usually released in `/boot/`, together with kernel images: `/boot/config-2.6.17-11-generic`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel or module ?

- The ***kernel image*** is a **single file**, resulting from the linking of all object files that correspond to features enabled in the configuration
  - This is the file that gets loaded in memory by the bootloader
  - All included features are therefore available as soon as the kernel starts, at a time where no filesystem exists
- Some features (device drivers, filesystems, etc.) can however be compiled as ***modules***
  - Those are *plugins* that can be loaded/unloaded dynamically to add/remove features to the kernel
  - Each **module is stored as a separate file in the filesystem**, and therefore access to a filesystem is mandatory to use modules
  - This is not possible in the early boot procedure of the kernel, because no filesystem is available

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel option types

- There are different types of options
  - `bool` options, they are either
    - *true* (to include the feature in the kernel) or
    - *false* (to exclude the feature from the kernel)
  - `tristate` options, they are either
    - *true* (to include the feature in the kernel image) or
    - *module* (to include the feature as a kernel module) or
    - *false* (to exclude the feature)
  - `int` options, to specify integer values
  - `string` options, to specify string values

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel option dependencies

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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: `libqt4-dev`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

make xconfig screenshot

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

make xconfig search interface

Looks for a keyword in the description string

Allows to select or unselect found parameters.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Corresponding .config file excerpt

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

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

Section name
(helps to locate settings in the interface)

All parameters are prefixed with `CONFIG_`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

make nconfig

`make nconfig`
A newer, similar text interface

More user friendly (for example, easier to access help information).

Required Debian packages:
`libncurses-dev`

make oldconfig

`make oldconfig`

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

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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!

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Configuration per architecture

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

Overview of kernel options (1)

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

Overview of kernel options (2)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (3)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (4)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (5)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (6)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (7)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (8)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (9)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overview of kernel options (10)

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Compiling and installing the kernel for the host system

Kernel compilation

- `make`
  - in the main kernel source directory
  - Remember to run `make -j 4` if you have multiple CPU cores to speed up the compilation process
  - No need to run as root !
- Generates
  - `vmlinux`, the raw uncompressed kernel image, at the ELF format, useful for debugging purposes, but cannot be booted
  - `arch/<arch>/boot/*Image`, the final, usually compressed, kernel image that can be booted
    - `bzImage` for x86, `zImage` for ARM, `vmImage.gz` for Blackfin, etc.
  - All kernel modules, spread over the kernel source tree, as `.ko` files.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel installation

- `make install`
  - Does the installation for the host system by default, so needs to be run as root. Generally not used when compiling for an embedded system, and it installs files on the development workstation.
- Installs
  - `/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
- Typically re-runs the bootloader configuration utility to take into account the new kernel.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Module installation

- `make modules_install`
  - Does the installation for the host system by default, so needs to be run as root
- Installs all modules in `/lib/modules/<version>/`
  - `kernel/`
    Module `.ko` (Kernel Object) files, in the same directory structure as in the sources.
  - `modules.alias`
    Module aliases for module loading utilities. Example line:
    `alias sound-service-?-0 snd_mixer_oss`
  - `modules.dep`
    Module dependencies
  - `modules.symbols`
    Tells which module a given symbol belongs to.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel cleanup targets

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

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Specifying cross-compilation

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

- `ARCH` is the name of the architecture. It is defined by the name of the subdirectory in arch/ in the kernel sources
- `CROSS_COMPILE` is the prefix of the cross compilation tools
  - Example: `arm-linux-` if your compiler is `arm-linux-gcc`
- Three solutions
  - Force these two variables in the main kernel Makefile

```
ARCH      ?= arm
CROSS_COMPILE ?= arm-linux-
```

  - Pass `ARCH` and `CROSS_COMPILE` on the make command line
  - Define `ARCH` and `CROSS_COMPILE` as environment variables
  - Don't forget to have the values properly set at all steps, otherwise the kernel configuration and build system gets confused

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Predefined configuration files

- Default configuration files available, per board or per-CPU family
  - They are stored in `arch/<arch>/configs/`, and are just minimal `.config` files
  - This is the most common way of configuring a kernel for embedded platforms
- Run `make help` to find if one is available for your platform
- To load a default configuration file, just run `make acme_defconfig`
  - This will overwrite your existing `.config` !
- To create your own default configuration file
  - `make savedefconfig`, to create a minimal configuration file
  - `mv defconfig arch/<arch>/configs/myown_defconfig`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Configuring the kernel

- After loading a default configuration file, you can adjust the configuration to your needs with the normal `xconfig`, `gconfig` or `menuconfig` interfaces
- You can also start the configuration from scratch without loading a default configuration file
- As the architecture is different than your host architecture
  - Some options will be different from the native configuration (processor and architecture specific options, specific drivers, etc.)
  - Many options will be identical (filesystems, network protocol, architecture-independent drivers, etc.)
- Make sure you have the support for the right *CPU*, the right *board* and the right *device drivers*.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Building and installing the kernel

- Run `make`
- Copy the final kernel image to the target storage
  - can be `uImage, zImage, vmlinux, bzImage` in `arch/<arch>/boot`
- `make install` is rarely used in embedded development, as the kernel image is a single file, easy to handle
  - It is however possible to customize the make install behaviour in `arch/<arch>/boot/install.sh`
- `make modules_install` is used even in embedded development, as it installs many modules and description files
  - `make INSTALL_MOD_PATH=<dir>/ modules_install`
  - The `INSTALL_MOD_PATH` is needed to install the modules in the target root filesystem instead of your host root filesystem.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Kernel command line

- In addition to the compile time configuration, the kernel behaviour can be adjusted with no recompilation using the ***kernel command line***
- The kernel command line is a string that defines various arguments to the kernel
  - It is very important for system configuration
  - `root=` for the root filesystem (covered later)
  - `console=` for the destination of kernel messages
  - and many more, documented in `Documentation/kernel-parameters.txt` in the kernel sources
- This kernel command line is either
  - Passed by the bootloader. In U-Boot, the contents of the `bootargs` environment variable is automatically passed to the kernel
  - Built into the kernel, using the `CONFIG_CMDLINE` option.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Practical lab – Kernel 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.

Using kernel modules

Advantages of modules

- Modules make it easy to develop drivers without rebooting: load, test, unload, rebuild, load...
- 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 control and privileges in the system. No particular protection. That's why only the root user can load and unload modules.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`
  This file is generated when you run `make modules_install`.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 are available through the `dmesg` command.
  ("**d**iagnostic **mes**sa**g**e")
- Kernel log messages are also displayed in the system console
  (console messages can be filtered by level using the `loglevel` kernel, or completely disabled with the `quiet` parameter).
- Note that you can write to the kernel log from userspace too:
  `echo "Debug info" > /dev/kmsg`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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. The full path to the module object file must be given.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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. Modprobe automatically looks in `/lib/modules/<version>/` for the object file corresponding to the given module name.
- `lsmod`
  Displays the list of loaded modules
  Compare its output with the contents of `/proc/modules`!

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Passing parameters to modules

- Find available parameters:
  `modinfo snd-intel8x0m`
- 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 driver is built statically into the kernel:
  `snd-intel8x0m.index=-2`

driver name
driver parameter name
driver parameter value

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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/

Linux root filesystem
Principle and solutions

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Filesystems

- Filesystems are used to organize data in directories and files on storage devices or on the network. The directories and files are organized as a hierarchy
- In Unix systems, applications and users see a ***single global hierarchy*** of files and directories, which can be composed of several filesystems.
- Filesystems are ***mounted*** in a specific location in this hierarchy of directories
  - When a filesystem is mounted in a directory (called *mount point*), the contents of this directory reflects the contents of the storage device
  - When the filesystem is unmounted, the *mount point* is empty again.
- This allows applications to access files and directories easily, regardless of their exact storage location

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Filesystems (2)

- Create a mount point, which is just a directory
```
$ mkdir /mnt/usbkey
```
- It is empty
```
$ ls /mnt/usbkey
$
```
- Mount a storage device in this mount point
```
$ mount -t vfat /dev/sda1 /mnt/usbkey
$
```
- You can access the contents of the USB key
```
$ ls /mnt/usbkey
docs prog.c picture.png movie.avi
$
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

mount / umount

- `mount` allows to mount filesystems
  - `mount -t type device mountpoint`
  - `type` is the type of filesystem
  - `device` is the storage device, or network location to mount
  - `mountpoint` is the directory where files of the storage device or network location will be accessible
  - `mount` with no arguments shows the currently mounted filesystems
- `umount` allows to unmount filesystems
  - This is needed before rebooting, or before unplugging an USB key, because the Linux kernel caches writes in memory to increase performances. umount makes sure that those writes are committed to the storage.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Root filesystem

- A particular filesystem is mounted at the root of the hierarchy, identified by `/`
- This filesystem is called the ***root filesystem***
- As `mount` and `umount` are programs, they are files inside a filesystem.
  - They are not accessible before mounting at least one filesystem.
- As the root filesystem is the first mounted filesystem, it cannot be mounted with the normal mount command
- It is mounted directly by the kernel, according to the `root=` kernel option
- When no root filesystem is available, the kernel panics

```
Please append a correct "root=" boot option
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown
block(0,0)
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Location of the root filesystem

- It can be mounted from different locations
  - From the partition of an hard disk
  - From the partition of an USB key
  - From the partition of a SD card
  - From the partition of a NAND Flash or similar type of storage device
  - From the network, using the NFS protocol
  - From memory, using a pre-loaded filesystem (by the bootloader)
  - etc.
- It is up to the system designer to choose the configuration for the system, and configure the kernel behaviour with `root=`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mounting rootfs from storage devices

- Partitions of a hard disk or USB key
  - `root=/dev/sdXY`, where X is a letter indicating the device, and Y a number indicating the partition
  - `/dev/sdb2` is the second partition of the second disk drive (either USB key or ATA hard drive)
- Partitions of a SD card
  - `root=/dev/mmcblkXpY`, where X is a number indicating the device and Y a number indicating the partition
  - `/dev/mmcblk0p2` is the second partition of the first device
- Partitions of a NAND Flash
  - `root=/dev/mtdblockX`, where X is the partition number
  - `/dev/mtdblock3` is the fourth partition of the NAND flash (if only one NAND flash is present)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mounting rootfs over the network (1)

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 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).

Mounting rootfs over the network (2)

On the development workstation side, a NFS server is needed

- Install an NFS server (example: Debian, Ubuntu)
  `sudo apt-get install nfs-kernel-server`
- Add the exported directory to your `/etc/exports` file:
  `/home/tux/rootfs 192.168.1.111(rw,no_root_squash,no_subtree_check)`

  client address NFS server options
- Start or restart your NFS server (example: Debian, Ubuntu)
  `sudo /etc/init.d/nfs-kernel-server restart`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mounting rootfs over the network (3)

- On the target system

- The kernel must be compiled with

  `CONFIG_NFS_FS=y` (NFS support)
  `CONFIG_IP_PNP=y` (configure IP at boot time)
  `CONFIG_ROOT_NFS=y` (support for NFS as rootfs)

- The kernel must be booted with the following parameters

  `root=/dev/nfs` (we want rootfs over NFS)
  `ip=192.168.1.111` (target IP address)
  `nfsroot=192.168.1.110:/home/tux/rootfs/`
  (NFS server details)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mounting rootfs over the network (4)

Host

NFS server

```
/home/tux/rootfs/
/home/tux/rootfs/root/
/home/tux/rootfs/root/README
/home/tux/rootfs/usr/
/home/tux/rootfs/bin/
/home/tux/rootfs/bin/ls
```

Ethernet

Target

NFS client built into the kernel

```
/
/root
/root/README
/usr
/bin
/bin/ls
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

rootfs in memory: initramfs (1)

- It is also possible to have the root filesystem integrated into the kernel image
- It is therefore loaded into memory together with the kernel
- This mechanism is called ***initramfs***
  - It integrates a compressed archive of the filesystem into the kernel image
- It is useful for two cases
  - Fast booting of very small root filesystems. As the filesystem is completely loaded at boot time, application startup is very fast.
  - As an intermediate step before switching to a real root filesystem, located on devices for which drivers not part of the kernel image are needed (storage drivers, filesystem drivers, network drivers). This is always used on the kernel of desktop/server distributions to keep the kernel image size reasonable.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

rootfs in memory: initramfs (2)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

rootfs in memory: initramfs (3)

- The contents of an initramfs are defined at the kernel configuration level, with the `CONFIG_INITRAMFS_SOURCE` option
  - Can be the path to a directory containing the root filesystem contents
  - Can be the path to a cpio archive
  - Can be a text file describing the contents of the initramfs (see documentation for details)
- The kernel build process will automatically take the contents of the `CONFIG_INITRAMFS_SOURCE` option and integrate the root filesystem into the kernel image
- `Documentation/filesystems/ramfs-rootfs-initramfs.txt`
  `Documentation/early-userspace/README`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux root filesystem
Contents

Root filesystem organization

- The organization of a Linux root filesystem in terms of directories is well-defined by the ***Filesystem Hierarchy Standard***
- http://www.pathname.com/fhs/
- Most Linux systems conform to this specification
  - Applications expect this organization
  - It makes it easier for developers and users as the filesystem organization is similar in all systems

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Important directories (1)

- `/bin` Basic programs
- `/boot` Kernel image (only when the kernel is loaded from a filesystem, not common on non-x86 architectures)
- `/dev` Device files (covered later)
- `/etc` System-wide configuration
- `/home` Directory for the users home directories
- `/lib` Basic libraries
- `/media` Mount points for removable media
- `/mnt` Mount points for static media
- `/proc` Mount point for the proc virtual filesystem

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Important directories (2)

- `/root` Home directory of the root user
- `/sbin` Basic system programs
- `/sys` Mount point of the sysfs virtual filesystem
- `/tmp` Temporary files
- `/usr`
  - `/usr/bin` Non-basic programs
  - `/usr/lib` Non-basic libraries
  - `/usr/sbin` Non-basic system programs
- `/var` Variable data files. This includes spool directories and files, administrative and logging data, and transient and temporary files

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Separation of programs and libraries

- Basic programs are installed in `/bin` and `/sbin` and basic libraries in `/lib`
- All other programs are installed in `/usr/bin` and `/usr/sbin` and all other libraries in `/usr/lib`
- In the past, on Unix systems, `/usr` was very often mounted over the network, through NFS
- In order to allow the system to boot when the network was down, some binaries and libraries are stored in `/bin`, `/sbin` and `/lib`
- `/bin` and `/sbin` contain programs like ls, ifconfig, cp, bash, etc.
- `/lib` contains the C library and sometimes a few other basic libraries
- All other programs and libraries are in `/usr`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Devices

- One of the kernel important role is to allow applications to access hardware devices
- In the Linux kernel, most devices are presented to userspace applications through two different abstractions
  - Character device
  - Block device
- Internally, the kernel identifies each device by a triplet of information
  - Type (character or block)
  - Major (typically the category of device)
  - Minor (typically the identifier of the device)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Types of devices

- Block devices
  - A device composed of fixed-sized blocks, that can be read and written to store data
  - Used for hard disks, USB keys, SD cards, etc.
- Character devices
  - Originally, an infinite stream of bytes, with no beginning, no end, no size. The pure example: a serial port.
  - Used for serial ports, terminals, but also sound cards, video acquisition devices, frame buffers
  - Most of the devices that are not block devices are represented as character devices by the Linux kernel

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Devices: everything is a file

- A very important Unix design decision was to represent most of the “system objects” as files
- It allows applications to manipulate all “system objects” with the normal file API (open, read, write, close, etc.)
- So, devices had to be represented as files to the applications
- This is done through a special artefact called a ***device file***
- It a special type of file, that associates a file name visible to userspace applications to the triplet (type, major, minor) that the kernel understands
- All *device files* are by convention stored in the `/dev` directory

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Device files examples

Example of device files in a Linux system

```
$ ls -l /dev/ttyS0 /dev/tty1 /dev/sda1 /dev/sda2 /dev/zero
brw-rw---- 1 root disk    8,  1 2011-05-27 08:56 /dev/sda1
brw-rw---- 1 root disk    8,  2 2011-05-27 08:56 /dev/sda2
crw------- 1 root root    4,  1 2011-05-27 08:57 /dev/tty1
crw-rw---- 1 root dialout 4, 64 2011-05-27 08:56 /dev/ttyS0
crw-rw-rw- 1 root root    1,  5 2011-05-27 08:56 /dev/zero
```

Example C code that uses the usual file API to write data to a serial port

```
int fd;
fd = open("/dev/ttyS0", O_RDWR);
write(fd, "Hello", 5);
close(fd);
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Creating device files

- On a basic Linux system, the device files have to be created manually using the mknod command
  - `mknod /dev/<device> [c|b] major minor`
  - Needs root privileges
  - Coherency between device files and devices handled by the kernel is left to the system developer
- On more elaborate Linux systems, mechanisms can be added to create/remove them automatically when devices appear and disappear
  - `devtmpfs` virtual filesystem, since kernel 2.6.32
  - `udev` daemon, solution used by desktop and server Linux systems
  - `mdev` program, a lighter solution than `udev`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

proc virtual filesystem

- The `proc` virtual filesystem exists since the beginning of Linux
- It allows
  - The kernel to expose statistics about running processes in the system
  - The user to adjust at runtime various system parameters about process management, memory management, etc.
- The `proc` filesystem is used by many standard userspace applications, and they expect it to be mounted in `/proc`
- Applications such as `ps` or `top` would not work without the `proc` filesystem
- Command to mount `/proc`:
  `mount -t proc nodev /proc`
- `Documentation/filesystems/proc.txt` in the kernel sources
- `man proc`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

proc contents

- One directory for each running process in the system
  - `/proc/<pid>`
  - `cat /proc/3840/cmdline`
  - It contains details about the files opened by the process, the CPU and memory usage, etc.
- `/proc/interrupts, /proc/devices, /proc/iomem, /proc/ioports` contain general device-related information
- `/proc/cmdline` contains the kernel command line
- `/proc/sys` contains many files that can be written to to adjust kernel parameters
  - They are called *sysctl*. See `Documentation/sysctl/` in kernel sources.
  - Example
    `echo 3 > /proc/sys/vm/drop_caches`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

sysfs filesystem

- The sysfs filesystem is a feature integrated in the 2.6 Linux kernel
- It allows to represent in userspace the vision that the kernel has of the buses, devices and drivers in the system
- It is useful for various userspace applications that need to list and query the available hardware, for example udev or mdev
- All applications using sysfs expect it to be mounted in the `/sys` directory
- Command to mount `/sys`:
  ```
  mount -t sysfs nodev /sys
  ```
- ```
  $ ls /sys/
  block   bus     class   dev     devices  firmware
  fs      kernel  module  power
  ```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**
  ```

Basic applications

- In order to work, a Linux system needs at least a few applications
- An `init` application, which is the first userspace application started by the kernel after mounting the root filesystem
  - The kernel tries to run `/sbin/init, /bin/init, /etc/init` and `/bin/sh`.
  - If none of them are found, the kernel panics and the boot process is stopped.
  - The init application is responsible for starting all other userspace applications and services
- Usually a shell, to allow an user to interact with the system
- Basic Unix applications, to copy files, move files, list files (commands like `mv, cp, mkdir, cat`, etc.)
- Those basic components have to be integrated into the root filesystem to make it usable

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Overall booting process

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

BusyBox

Thomas Petazzoni
Michael Opdenacker
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/busybox
Corrections, suggestions, contributions and translations are welcome!

Why Busybox ?

- A Linux system needs a basic set of programs to work
  - An init program
  - A shell
  - Various basic utilities for file manipulation and system configuration
- In normal Linux systems, those programs are provided by different projects
  - `coreutils`, `bash`, `grep`, `sed`, `tar`, `wget`, `modutils`, etc. are all different projects
  - A lot of different components to integrate
  - Components not designed with embedded systems constraints in mind: they are not very configurable and have a wide range of features
- Busybox is an alternative solution, extremely common on embedded systems

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

General purpose toolbox: BusyBox

http://www.busybox.net/

- Rewrite of many useful Unix command line utilities
  - Integrated into a single project, which makes it easy to work with
  - Designed with embedded systems in mind: highly configurable, no unnecessary features
- All the utilities are compiled into a single executable, `/bin/busybox`
  - Symbolic links to `/bin/busybox` are created for each application integrated into Busybox
- For a fairly featureful configuration, less than < 500 KB (statically compiled with uClibc) or less than 1 MB (statically compiled with glibc).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

BusyBox commands!

[, [[, addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, basename, bbconfig, bbsh, brctl, bunzip2, busybox, bzcat, bzip2, cal, cat, catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, 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, rmmod, 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, sha1sum, 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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Applet highlight : Busybox init

- Busybox provides an implementation of an init program
- Simpler than the init implementation found on desktop/server systems : no runlevels are implemented
- A single configuration file : `/etc/inittab`
  - Each line has the form `<id>::<action>:<process>`
- Allows to run services at startup, and to make sure that certain services are always running on the system
- See `examples/inittab` in Busybox for details on the configuration

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Applet highlight - 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 -> Installation Options -> BusyBox installation prefix`
- Add the cross-compiler path to the PATH environment variable:
  `export PATH=/usr/xtools/arm-unknown-linux-uclibcgnueabi/bin:$PATH`
- Compile BusyBox:
  `make`
- Install it (this creates a Unix directory structure symbolic links to the `busybox` executable):
  `make install`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/block-filesystems
Corrections, suggestions, contributions and translations are welcome!

Block vs. flash

- Storage devices are classified in two main types: **block devices** and **flash devices**
  - They are handled by different subsystems and different filesystems
- **Block devices** can be read and written to on a per-block basis, without erasing, and do not wear out when being used for a long time
  - Hard disks, floppy disks, RAM disks
  - USB keys, Compact Flash, SD card, these are based on flashes, but have an integrated controller that emulates a block device
- **Flash devices** can be read, but writing requires erasing, and often occurs on a larger size than the “block” size
  - NOR flash, NAND flash

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Block device list

- The list of all block devices available in the system can be found in `/proc/partitions`

```
$ cat /proc/partitions
major minor  #blocks  name

   8        0  312571224 sda
   8        1  303903744 sda1
   8        2          1 sda2
   8        5    8665088 sda5
```

- And also in `/sys/block/`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Traditional block filesystems

Traditional filesystems

- Can be left in a non-coherent state after a system crash or sudden power-off, which requires a full filesystem check after reboot.
- `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)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Journaled 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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Journaled block filesystems

Journaled filesystems

- `ext3`: `ext2` with journal extension
  `ext4`: the new generation with many improvements.
  Ready for production. They are the default filesystems for all Linux systems in the world.
- The Linux kernel supports many other filesystems: `reiserFS`, `JFS`, `XFS`, etc. Each of them have their own characteristics, but are more oriented towards server or scientific workloads
- `btrfs` (“Butter F S”)
  The next generation. In mainline but still experimental.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Creating ext2/ext3 volumes

- To create an empty ext2/ext3 filesystem on a block device or inside an already-existing image file
  - `mkfs.ext2 /dev/hda3`
  - `mkfs.ext3 /dev/sda2`
  - `mkfs.ext2 disk.img`
- To create a filesystem image from a directory containing all your files and directions
  - Use the genext2fs tool, from the package of the same name
  - `genext2fs -d rootfs/ rootfs.img`
  - Your image is then ready to be transferred to your block device

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mounting filesystem images

- Once a filesystem image has been created, one can access and modifies its contents from the development workstation, using the ***loop*** mechanism
- Example:
```
genext2fs -d rootfs/ rootfs.img
mkdir /tmp/tst
mount -t ext2 -o loop rootfs.img /tmp/tst
```
- In the `/tmp/tst`, one can access and modify the contents of the rootfs.img file.
- This is possible thanks to loop, which is a kernel driver that emulates a block device with the contents of a file.
- Do not forget to umount before using the filesystem image !

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Squashfs

Squashfs: http://squashfs.sourceforge.net

- Read-only, compressed filesystem for block devices
  Fine for parts of a filesystem which can be read-only (kernel, binaries...)
- Great compression rate and read access performance
- Used in most live CDs and live USB distributions
- Supports LZO compression for better performance
  on embedded systems with slow CPUs
  (at the expense of a slightly degraded compression rate)
- Available in mainline 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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Squashfs - How to use

- Need to install the `squashfs-tools` package
- Creation of the image
  - On your workstation, create your filesystem image:
    `mksquashfs rootfs/ rootfs.sqfs`
  - Caution: if the image already exists remove it first,
    or use the `-noappend` option.
- Installation of the image
  - Let's assume your partition on the target is in `/dev/sdc1`
  - Copy the filesystem image on the device
    `dd if=rootfs.sqfs of=/dev/sdc1`
    Be careful when using dd to not overwrite the incorrect partition!
- Mount your filesystem:
  `mount -t squashfs /dev/sdc1 /mnt/root`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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)

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Flash filesystems

Michael Opdenacker
Thomas Petazzoni
Gregory Clement
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/flash-filesystems
Corrections, suggestions, contributions and translations are welcome!

The MTD subsystem

MTD: Memory Technology Devices (flash, ROM, RAM)

Linux filesystem interface

MTD “User” modules

UBI | jffs2 | Char device | Block device

yaffs2 | Read-only block device

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

FTL | NFTL | INFTL

MTD Chip drivers

NOR flash | RAM chips

NAND flash | DiskOnChip flash | ROM chips

Block device | Virtual memory

Virtual devices appearing
as MTD devices

Memory devices
hardware

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

MTD devices

- MTD devices are visible in `/proc/mtd`
- The **mtdchar** driver creates a character device for each MTD device of the system
  - Usually named `/dev/mtdX`, major 90. Even minors for read-write access, odd minors for read-only access
  - Provide `ioctl()` to erase and manage the flash
  - Used by the *mtd-utils*
- The **mtdblock** driver creates a block device for each MTD device of the system
  - Usually named `/dev/mtdblockX`, major 31. Minor is the number of the MTD device
  - Allows read/write block-level access. But bad blocks are not handled, and no wear leveling is done for writes.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

MTD partitioning

- MTD devices are usually partitioned
  - It allows to use different areas of the flash for different purposes : read-only filesystem, read-write filesystem, backup areas, bootloader area, kernel area, etc.
- Unlike block devices, which contains their own partition table, the partitioning of MTD devices is described externally
  - Hard-coded into the kernel code
  - Specified through the kernel command line
- Each partition becomes a separate MTD device
  - Different from block device labeling (hda3, sda2)
  - `/dev/mtd1` is either the second partition of the first flash device, or the first partition of the second flash device

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Definition of MTD partitions

MTD partitions are defined in the kernel, in the board definitions: `arch/arm/mach-omap2/board-igep0020.c` example:

```
static struct mtd_partition igep2_onenand_partitions[] = {
    {
        .name           = "X-Loader",
        .offset         = 0,
        .size           = 2 * (64*(2*2048))
    },
    {
        .name           = "U-Boot",
        .offset         = MTDPART_OFS_APPEND,
        .size           = 6 * (64*(2*2048)),
    },
    [...]
    {
        .name           = "File System",
        .offset         = MTDPART_OFS_APPEND,
        .size           = MTDPART_SIZ_FULL,
    },
};
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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:

```
Muxed OneNAND(DDP) 512MB 1.8V 16-bit (0x58)
OneNAND version = 0x0031
Scanning device for bad blocks
OneNAND eraseblock 1812 is an initial bad block
[…]
Creating 5 MTD partitions on "omap2-onenand":
0x000000000000-0x000000080000 : "X-Loader"
0x000000080000-0x000000200000 : "U-Boot"
0x000000200000-0x000000280000 : "Environment"
0x000000280000-0x000000580000 : "Kernel"
0x000000580000-0x000020000000 : "File System"
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Modifying MTD partitions (2)

- You can now use the `mtdparts` kernel boot parameter
- Example:
  `mtdparts=omap2-onenand:512k(X-Loader),1536k(U-Boot)ro,512k(Environment),3m(Kernel),-(RootFS)`
- We've just defined 5 partitions in the `omap2-onenand` device:
  - `2nd stage bootloader` (512KiB)
  - `U-Boot` (1536KiB)
  - `U-Boot Environment` (512KiB)
  - `Kernel` (3MiB)
  - `data`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Modifying MTD partitions (3)

- Partition sizes must be multiple of the erase block size.
  You can use sizes in hexadecimal too. Remember the below sizes:
  `0x20000` = 128k, `0x100000` = 1m, `0x1000000` = 16m
- `ro` lists the partition as read only
- `-` is used to use all the remaining space.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

mtd-utils

- A set of utilities to manipulate MTD devices
  - `mtdinfo` to get detailed information about a MTD device
  - `flash_eraseall` to completely erase a given MTD device
  - `flashcp` to write to NOR flash
  - `nandwrite` to write to NAND flash
  - UBI utilities
  - Flash filesystem image creation tools: `mkfs.jffs2`, `mkfs.ubifs`
- Usually available as the mtd-utils package in your distribution
- See http://www.linux-mtd.infradead.org/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

jffs2

http://www.linux-mtd.infradead.org/doc/jffs2.html

- Today's standard filesystem for MTD flash
- Nice features: on the fly compression (saves storage space and reduces I/O), power down reliable, wear-leveling and ECC.
- 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.
  - Need to use the `CONFIG_JFFS2_SUMMARY` kernel option to store such information in flash. This dramatically reduces mount time (from 16 s to 0.8s for a 128 MB partition).

Standard file API

JFFS2 filesystem

MTD driver

Flash chip

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

jffs2 - How to use

On the Linux **target**

- Need either the `mtd-utils` from the MTD project, or their embedded variants from Busybox
- 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
  (copying from NFS or from external storage)
- Other possibility: use a jffs2 image (see next page to produce it):
  `flash_eraseall /dev/mtd2`
  `nandwrite -p /dev/mtd2 rootfs.jffs2`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

How to create a jffs2 image (1)

- `mkfs.jffs2` command available in the `mtd-utils` package.
  Caution: unlike some `mkfs` commands, it doesn't create a filesystem, but a filesystem image.
- First, find the erase block size from `mtdinfo -m 4`:
  ```
  [...]
  Eraseblock size: 262144 bytes, 256.0 KiB
  ```
- Then create the image on your workstation:
  ```
  mkfs.jffs2 --pad --no-cleanmarkers --eraseblock=256
  -d rootfs/ -o rootfs.jffs2
  ```
- The `--pad` option pads the jffs2 image contents until the end of the final erase block.
- It is fine if the jffs2 image is smaller than the MTD partition. The jffs2 file system will use the entire partition anyway.
- The `--no-cleanmarkers` option is for NAND flash only.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mounting a jffs2 image on your host

Useful to edit `jffs2` images on your development system
Mounting an MTD device as a loop device is a bit complex task.
Here's an example for `jffs2`, for your reference:

- First find the erase block size used to create the jffs2 image.
  Let's assume it is 256KiB (262144 bytes).
- Create a block device from the image
  `losetup /dev/loop0 root.jffs2`
- Emulate an MTD device from a block device,
  using the block2mtd kernel module
  `modprobe block2mtd block2mtd=/dev/loop0,262144`
- Make sure mtdblock support is available
  `modprobe mtdblock`
- Finally, mount the filesystem (create `/mnt/jffs2` if needed)
  `mount -t jffs2 /dev/mtdblock0 /mnt/jffs2`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Initializing jffs2 partitions from U-boot

You may not want to have `mtd-utils` on your target!

- Create a JFFS2 image on your workstation
- In the U-Boot prompt:
  - Download the jffs2 image to RAM with `tftp`
    Or copy this image to RAM from external storage
    (U-boot understands FAT filesystems and supports USB storage)
  - Flash it inside an MTD partition
    (exact instructions depending on flash type, NOR or NAND,
    reuse the instructions used to flash your kernel). Make sure to write only the size of the image, not more!
  - If you boot on a jffs2 root filesystem, add `root=/dev/mtdblock<x>` and `rootfstype=jffs2` to the Linux command line arguments.
- Limitation: need to split the jffs2 image in several chunks
  if bigger than the RAM size.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

yaffs2

http://www.yaffs.net/

- Mainly supports NAND flash
- No compression
- Wear leveling, ECC, power failure resistant
- Fast boot time
- Code available separately through `git` (Dual GPL / Proprietary license for non Linux operating systems)

yaffs2 - How to use

- Erase a partition:
  `flash_eraseall /dev/mtd2`
- The filesystem is automatically formatted at the first mount:
  `mount -t yaffs2 /dev/mtdblock2 /mnt/flash`
- Images can be created with the `mkyaffs` tool, from yaffs-utils
  http://code.google.com/p/yaffs2utils/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.
- Volumes can be dynamically resized or, on the opposite, can be read-only (static).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

UBI (2)

UBI
Logical
Erase Blocks

MTD
Physical
Erase Blocks

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
- Has a noticeable metadata overhead on very small partitions (4M, 8M)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

UBI layout

UBIFS mounted on /
(from ubi0:rootfs)
UBI Volume: rootfs
UBI Volume: raw
UBIFS mounted on /myapp
(from ubi0:data)
UBI Volume: data
UBI device ubi0
MTD partition 0
JFFS2 mounted on /appsettings
(from /dev/mtd1)
MTD partition 1
UBIFS mounted on /var/log
(from ubi1:log)
UBI Volume: log
UBI device ubi1
MTD partition 2
Flash device

UBI - Preparation

- 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)
  - This special character device is used by other UBI utilities
  - Major and minor number allocated in the kernel. Find these numbers in `/sys/class/misc/ubi_ctrl/dev` (e.g.: `10:63`)
  - Or run `ubinfo`:

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

- These steps are done once for all

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

UBI - Attaching

- Attach UBI to one (of several) of the MTD partitions:
  `ubiattach /dev/ubi_ctrl -m 1`
- This command creates the ubi0 device, which represent the full UBI space stored on MTD device 1
  - Find the major and minor numbers used by UBI:
    `cat /sys/class/ubi/ubi0/dev` (e.g. `250:0`)
  - Create the UBI device file:
    `mknod /dev/ubi0 c 250 0`
  - This UBI space can contain several volumes

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

UBI - Volume management

- Volume creation with `ubimkvol`
  - `ubimkvol /dev/ubi0 -N test -s 116MiB`
  - `ubimkvol /dev/ubi0 -N test -m` (max available size)
  - The volume is then identified as `ubi0:test` for the mount/umount commands
- Volume removal with `ubirmvol`
  - `ubirmvol /dev/ubi0 -N test`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

UBIFS - How to use

- When a UBI volume is created, creating an empty UBIFS filesystem is just a matter of mounting it
  - `mount -t ubifs ubi0:test /mnt/flash`
- Images of UBIFS filesystems can be created using the `mkfs.ubifs` utility
  - `mkfs.ubifs -m 4096 -e 252KiB -c 1000 -r /opt/img ubifs.img`
  - Can be written to a UBI volume using `ubiupdatevol` and the `/dev/ubiX_Y` devices

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Ubinize

- Images of a full UBI space, containing several volumes can be created using the `ubinize` utility
  - Can be written to a raw MTD using `onenand write`
  - But `onenand erase` will also erase the Erase Counters
- `ubinize -o ubi.img -p 256KiB -m 4096 rootubifs.ini`
  - Creates ubi.img, with 256KiB physical eraseblocks, 4096 minimum I/O size (-m).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

UBIFS - How to prepare a root fs

- Create the UBIFS image from the target directory
- Write the configuration file for the UBI device

```
[RFS-volume]
mode=ubi
image=rootfs.ubifs
vol_id=1
vol_size=30MiB
vol_type=dynamic
vol_name=rootfs
vol_flags=autoresize
vol_alignment=1
```

- Create the UBI device image
- Flash it using a bad block aware command from the bootloader
- Pass UBI layout information to the kernel:
  - `rootfstype=ubifs ubi.mtd=1 root=ubi0:rootfs`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Full benchmark details on
http://free-electrons.com/pub/conferences/2010/elce/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Reducing the number of writes

- Of course, do not use your flash storage as swap area (rare in embedded systems anyway)
- 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 (default setting since Linux 2.6.30).
- Don't use the `sync` mount option (commits writes immediately). Use the `fsync()` system call for per-file synchronization.
- You may decide to do without journaled filesystems. They cause more writes, but are also much more power down resistant (trade-off).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Useful reading

- Arnd Bergmann: Optimizing Linux with cheap flash drives
  In depth coverage of flash storage with a block interface.
  http://lwn.net/Articles/428584/
- 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/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Practical lab – Flash filesystems

- Creating partitions in your internal flash storage.
- Use a read-only JFFS2 partition for the system
- Use a read-write JFFS2 partition for data

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Embedded Linux system development

Thomas Petazzoni
Michael Opdenacker
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/sysdev
Corrections, suggestions, contributions and translations are welcome!

Contents

- Using open-source components
- Tools for the target device
  - Networking
  - System utilities
  - Language interpreters
  - Audio, video and multimedia
  - Graphical toolkits
  - Databases
  - Web browsers
- System building
- Emulators
- Commercial toolsets and distributions

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Leveraging open-source components in an Embedded Linux system

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
  - They are freely available, freely distributable, and thanks to their open-source nature, they can be analyzed and modified according to the needs of the project
- 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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Find existing components

- Freshmeat, a website referencing most open-source projects
  http://www.freshmeat.net
- Free Software Directory
  http://directory.fsf.org
- Look at other embedded Linux products, and see what their components are
- Look at the list of software packaged by embedded Linux build systems
  - These are typically chosen for their suitability to embedded systems
- Ask the community or Google
- This presentation will also feature a list of components for common needs

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 !

Licenses (1)

- All software that are under a free software license give four freedoms to all users
  - Freedom to use
  - Freedom to study
  - Freedom to copy
  - Freedom to modify and distribute modified copies
- See http://www.gnu.org/philosophy/free-sw.html for a definition of Free Software
- Open Source software, as per the definition of the Open Source Initiative, are technically similar to Free Software in terms of freedoms
- See http://www.opensource.org/docs/osd for the definition of Open Source Software

Licenses (2)

- Free Software licenses fall in two main categories
  - The copyleft licenses
  - The non-copyleft licenses
- The concept of « copyleft » is to ask for reciprocity in the freedoms given to an user.
- The result is that when you receive a software under a copyleft free software license and distribute modified versions of it, you must do so under the same license
  - Same freedoms to the new users
  - It's an incentive to contribute back your changes instead of keeping them secret
- Non-copyleft licenses have no such requirements, and modified versions can be kept proprietary, but they still require attribution

GPL

- GNU General Public License
- Covers ~55% of the free software projects
  - Including the Linux kernel, Busybox and many applications
- Is a copyleft license
  - Requires derivative works to be released under the same license
  - Programs linked with a library released under the GPL must also be released under the GPL
- Some programs covered by version 2 (Linux kernel, Busybox and others)
- More and more programs covered by version 3, released in 2007
  - Major change for the embedded market: the requirement that the user must be able to **run** the modified versions on the device, if the device is a « consumer » device

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

GPL: redistribution

- No obligation when the software is not distributed
  - You can keep your modifications secret until the product delivery
- It is then authorized to distribute binary versions, if one of the following conditions is met:
  - Convey the binary with a copy of the source on a physical medium
  - Convey the binary with a written offer valid for 3 years that indicates how to fetch the source code
  - Convey the binary with the network address of a location where the source code can be found
  - See section 6. of the GPL license
- In all cases, the attribution and the license must be preserved
  - See section 4. and 5.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

LGPL

- GNU Lesser General Public License
- Covers ~10% of the free software projects
- A copyleft license
  - Modified versions must be released under the same license
  - But, programs linked against a library under the LGPL do not need to be released under the LGPL and can be kept proprietary
  - However, the user must keep the ability to update the library independently from the program, so dynamic linking must be used
- Used instead of the GPL for most of the libraries, including the C libraries
  - Some exceptions: MySQL, or Qt <= 4.4
- Also available in two versions, v2 and v3

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Licensing: examples

- You make modifications to the Linux kernel (to add drivers or adapt to your board), to Busybox, U-Boot or other GPL software
  - You must release the modified versions under the same license, and be ready to distribute the source code to your customers
- You make modifications to the C library or any other LGPL library
  - You must release the modified versions under the same license
- You create an application that relies on LGPL libraries
  - You can keep your application proprietary, but you must link dynamically with the LGPL libraries
- You make modifications to a non-copyleft licensed software
  - You can keep your modifications proprietary, but you must still credit the authors

Non-copyleft licenses

- A large family of non-copyleft licenses that are relatively similar in their requirements
- A few examples
  - Apache license (~ 4%)
  - BSD license (~ 6%)
  - MIT license (~ 4%)
  - X11 license
  - Artistic license (~9 %)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

BSD license

```
Copyright (c) <year>, <copyright holder>
All rights reserved.

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions are met:
    * Redistributions of source code must retain the above
copyright notice, this list of conditions and the following
disclaimer.
    * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided
with the distribution.
    * Neither the name of the <organization> nor the names of its
contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

[...]
```

Is this free software?

- Most of the free software projects are covered by ~10 well-known licenses, so it is fairly easy for the majority of project to get a good understanding of the license
- Otherwise, read the license text
- Check Free Software Foundation's opinion
  http://www.fsf.org/licensing/licenses/
- Check Open Source Initiative's opinion
  http://www.opensource.org/licenses

Respect free software licenses

- Free Software is not public domain software, the distributors have obligations due to the licenses
  - **Before** using a free software component, make sure the license matches your project constraints
  - Make sure to keep a complete list of the free software packages you use, the original version you used and to keep your modifications and adaptations well-separated from the original version
  - Conform to the license requirements **before** shipping the product to the customers
- Free Software licenses have been enforced successfully in courts
  - GPL-violations.org, http://www.gpl-violations.org
  - Software Freedom Law Center, http://www.softwarefreedom.org/
- Ask your legal department !

Keeping changes separate (1)

- When integrating existing open-source components in your project, it is sometimes needed to make modifications to them
  - Better integration, reduced footprint, bug fixes, new features, etc.
- Instead of mixing these changes, it is much better to keep them separate from the original component version
  - If the component needs to be upgraded, easier to know what modifications were made to the component
  - If support from the community is requested, important to know how different the component we're using is from the upstream version
  - Makes contributing the changes back to the community possible
- It is even better to keep the various changes made on a given component separate
  - Easier to review and to update to newer versions

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Keeping changes separate (2)

- The simplest solution is to use Quilt
  - Quilt is a tool that allows to maintain a stack of patches over source code
  - Makes it easy to add, remove modifications from a patch, to add and remove patches from stack and to update them
  - The stack of patches can be integrated into your version control system
  - https://savannah.nongnu.org/projects/quilt/
- Another solution is to use a version control system
  - Import the original component version into your version control system
  - Maintain your changes in a separate branch

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Tools for the target device

Networking

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`).
- An alternative to OpenSSH, used on desktop and server systems.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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)

Web servers

- BusyBox http server: http://busybox.net
  - Tiny: only adds 9 K to BusyBox (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, with a sepate interpreter).
  - License: GPL
- lighthttpd: http://lighttpd.net
  Low footprint server good at managing high loads.
  May be useful in embedded systems too
- Other possibilities: Boa, thttpd, etc.

Network utilities (1)

- **avahi** is an implementation of Multicast DNS Service Discovery, that allows programs to publish and discover services on a local network
- **bind**, a DNS server
- **iptables**, the userspace tools associated to the Linux firewall, Netfilter
- **iw and wireless tools**, the userspace tools associated to Wireless devices
- **netsnmp**, implementation of the SNMP protocol
- **openntpd**, implementation of the Network Time Protocol, for clock synchronization
- **openssl**, a toolkit for SSL and TLS connections

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Network utilities (2)

- **pppd**, implementation of the Point to Point Protocol, used for dialup connections
- **samba**, implements the SMB and CIFS protocols, used by Windows to share files and printers
- **coherence**, a UPnP/DLNA implementation
- **vsftpd, proftpd**, FTP servers

Tools for the target device

System utilities

System utilities

- **dbus,** an inter-application object-oriented communication bus
- **gpsd,** a daemon to interpret and share GPS data
- **hal**, the Hardware Abstraction Layer suite. A daemon that receives hardware notifications, maintains a database of available hardware devices and offers a D-Bus interface
- **libraw1394**, raw access to Firewire devices
- **libusb**, a userspace library for accessing USB devices without writing an in-kernel driver
- Utilities for kernel subsystems: **i2c-tools** for I2C, **input-tools** for input, **mtd-utils** for MTD devices, **usbutils** for USB devices

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Tools for the target device

Language interpreters

Language interpreters

- Interpreters for the most common scripting languages are available. Useful for
  - Application development
  - Web services development
  - Scripting
- Languages supported
  - Lua
  - Python
  - Perl
  - Ruby
  - TCL
  - PHP

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Tools for the target device

Audio, video and multimedia

Audio, video and multimedia

- **gstreamer**, a multimedia framework
  - Allows to decode/encode a wide variety of formats
  - Supports hardware encoders and decoders through plugins
- **alsa-lib**, the userspace tools associated to the ALSA sound kernel subsystem
- Encoding and decoding libraries such as flac, libogg, libtheora, libvorbis, libmad, libsndfile, speex, etc.

Tools for the target device

Graphical toolkits

Graphical toolkits
« Low-level » solutions and layers

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

DirectFB: usage (2)

X.org - KDrive

- Stand-alone simplified version of the X server, for embedded systems
  - Formerly know 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

libsysfs
~27k

libpng
~130k

expat
~120k

zlib
~64k

freetype
~380k

pixman
~130k

fontconfig
~165k

Total: 1.5m

Total, without X toolkit: 5.4m

Kdrive: architecture

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Gtk-DFB: dependencies and size

Total: ~9 MB

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://doc.qt.nokia.com/stable/qt-embedded-linux.html

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.
- Qt Embedded has an integrated windowing system, allowing several applications to share the same screen
- Very well documented
- Since version 4.5, available under the LGPL, allowing proprietary applications

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Qt: size and dependencies

Total: ~15.5 MB

Qt's usage

Qt on the Dash Express navigation system

Qt on the OpenMoko phone

Qt on the Netflix Player by Roku

Other less used solutions

- Enlightenment foundation libraries
  - Very powerful, but complicated to use due to the lack of documentation
  - http://www.enlightenment.org/p.php?p=about/efl
- FLTK
  - Lightweight C++ toolkit. Version 2.x works only on top of X.org. Used by relatively few applications.
  - http://www.fltk.org
- WxEmbedded
  - The embedded version of WxWindows. Works on top of X.org and DirectFB
  - http://www.wxwidgets.org/docs/embedded.htm

Tools for the target device

Databases

Lightweight database - SQLite

http://www.sqlite.org

- SQLite is a small C library that implements a self-contained, embeddable, lightweight, zero-configuration SQL database engine
- The database engine of choice for embedded Linux systems
  - Can be used as a normal library
  - Can be directly embedded into a application, even a proprietary one since SQLite is released in the public domain

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
  The Dillo binary fits in 940 KB on x86
  (doesn't include the FLTK2 shared libraries)
- License: GPL
- Supports many standard features: cookies, images, tables, CSS...
  Extensible through plugins (e.g. ftp)
- Fits well on small screens
- Still missing: frames, javascript
- A good solution if your product just needs to display its own HTML pages. Not sufficient as a general purpose web browser.

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 everywhere (MacOS X, iPhone, Google Android and Chrome...)
  Many applications (browsers, e-mail clients...) are already using WebKit:
  http://trac.webkit.org/projects/webkit/wiki/Applications%20using%20WebKit
- Multiple graphical back-ends: Qt4, GTK...
- Lightweight web-browsers : Midori (GTK), Arora (Qt)
- You could use it to create your custom browser.

Example of components used in real devices

Industrial applications

- In many industrial applications, the system is only responsible for monitoring and control a device
- Such a system is usually relatively simple in terms of components
  - Kernel
  - BusyBox
  - C library
  - Applications relying directly on the C library, sometimes using the real-time capabilities of the Linux kernel
  - Sometimes a Web server for remote control, or another server implementing a custom protocol

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Digital Photo Frame: requirements

- Example taken from a conference of Matt Porter, Embedded Alley at ELC 2008
- Hardware: ARM SoC with DSP, audio, 800x600 LCD, MMC/SD, NAND, buttons, speakers
- The photo frame must be able to
  - Display to the LCD
  - Detect SD card insertion, notify applications of the insertion so that applications can build a catalog of the pictures on the SD card
  - Modern 3D GUI with nice transitions
  - Navigation through buttons
  - Support audio playback (MP3, playlists, ID3 tag)
  - JPEG resize and rotation

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Digital Photo Frame: components (1)

- Base system
  - Components present in virtually all embedded Linux systems
  - The U-Boot bootloader
  - Linux Kernel
    - Drivers for SD/MMC, framebuffer, sound, input devices
  - Busybox
  - Build system, in this case was OpenEmbedded
  - Components: **u-boot, linux, busybox**

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Digital Photo Frame: components (2)

- Event handling to detect SD card insertion
  - udev, that receives events from the kernel, creates device nodes, and sends events to HAL
  - HAL, which maintains a database of available devices and provides a D-Bus API
  - D-Bus to connect HAL with the application. The application subscribes to HAL event through D-Bus and gets notified when they are triggered
  - Components: **udev, hal, dbus**

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Digital Photo Frame: components (3)

- JPEG display
  - **libjpeg** to decode the pictures
  - **jpegtran** to resize and rotate them
  - FIM (Fbi Improved) for dithering
- MP3 support
  - **libmad** for playing
  - **libid3** for ID3 tags reading
  - **libm3u** to support playlists
  - Used vendor-provided components to leverage the DSP to play MP3

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Digital Photo Frame: components (4)

- 3D interface
  - Vincent, an open-source implementation of OpenGL ES
  - Clutter, higher-level API to develop 3D applications
- Application itself
  - Manages media events
  - Uses the JPEG libraries to decode and render pictures
  - Receives Linux input events from buttons and draws OpenGL-based UI developed with Clutter
  - Manage a user-defined configuration
  - Play the music with the MP3-related libraries
  - Display photo slideshow

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

System building

System building: goal and solutions

- Goal
  - Integrate all the software components, both third-party and in-house, into a working root filesystem
  - It involves the download, extraction, configuration, compilation and installation of all components, and possibly fixing issues and adapting configuration files
- Several solutions
  - Manually
  - System building tools
  - Distributions or ready-made filesystems

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

System building: manually (2)

- Manual system building is not recommended for production projects
- However, using automated tools often requires the developer to dig into specific issues
- Having a basic understanding of how a system can be built manually is therefore very useful to fix issues encountered with automated tools
  - We will first study manual system building, and during a practical lab, create a system using this method
  - Then, we will study the automated tools available, and use one of them during a lab

System foundations

- A basic root file system needs at least
  - A traditional directory hierarchy, with `/bin, /etc, /lib, /root, /usr/bin, /usr/lib, /usr/share, /usr/sbin, /var, /sbin`
  - A set of basic utilities, providing at least the init program, a shell and other traditional Unix command line tools. This is usually provided by Busybox
  - The C library and the related libraries (thread, math, etc.) installed in `/lib`
  - A few configuration files, such as `/etc/inittab`, and initialization scripts in `/etc/init.d`
- On top of this foundation common to most embedded Linux system, we can add third-party or in-house components

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Target and build spaces

- The system foundation, Busybox and C library, are the core of the target root filesystem
- However, when building other components, one must distinguish two directories
  - The « target » space, which contains the target root filesystem, everything that is needed for **execution** of the application
  - The « build » space, which will contain a lot more files than the «target» space, since it is used to keep everything needed to **compile** libraries and applications. So we must keep the headers, documentation, and other configuration files

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Build systems

- Each open-source component comes with a mechanism to configure, compile and install it
  - A basic simple Makefile
    - Need to read the Makefile to understand how it works and how to tweak it for cross-compilation
  - A build system based on the Autotools
    - As this is the most common build system, we will study it in details
  - CMake, http://www.cmake.org/
    - Newer and simpler than the autotools. Used by large projects such as KDE or Second Life
  - Scons, http://www.scons.org/
  - Waf, http://code.google.com/p/waf/
  - Other manual build systems

Autotools and friends

- A family of tools, which associated together form a complete and extensible build system
  - **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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

automake / autoconf / autoheader

Written by the developer

Generated by the developer using the autotools

Generated by the user by running the configure script

automake / autoconf

- 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 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Configuring and compiling : native case

- The traditional steps to configure and compile an autotools based package are
  - Configuration of the package
    `./configure`
  - Compilation of the package
    `make`
  - Installation of the package
    `make install`
- Additional arguments can be passed to the ./configure script to adjust the component configuration.
- Only the make install needs to be done as root if the installation should take place system-wide

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Configuring and compiling : cross case (1)

- For cross-compilation, things are a little bit more complicated.
- 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-compilation tools. The host tuple is also by default used as prefix.
- The `--host` argument must be passed to the `configure` script.
  - `--build` argument is automatically detected
  - `--target` is only for tools generating code.
- 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Configuring and compiling : cross case (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
make
make DESTDIR=/home/<user>/work/rootfs install
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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/`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Installation (2)

Contents of /usr after installation of zlib and libpng

```
./lib
./lib/libpng12.la
./lib/libpng.la                -> libpng12.la
./lib/libpng12.a
./lib/libpng.a                 -> libpng12.a
./lib/libpng.so.3.32.0
./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
./lib/pkgconfig/libpng12.pc
./lib/libz.so.1.2.3
./lib/libz.so                  -> libz.so.1.2.3
./lib/libz.so.1                -> libz.so.1.2.3
```

Libtool description file

Static version of the library

Dynamic version of the library

Pkgconfig description file

Zlib dynamic library

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Installation in the build and target spaces

- From all these files, everything except documentation is necessary to build an application that relies on libpng.
  - These files will go into the «build space»
- However, only the library binary in `<prefix>/lib` and some symbolic links are needed to execute the application on the target.
  - Only these files will go in the «target space»
- The build space must be kept in order to build other applications or recompile existing applications.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 thelib)
  ```
- 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Practical lab – Manual cross-compiling

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

System building tools: principle

- 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Available 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
- **OpenWRT**, originally a fork of Buildroot for wireless routers, not a more generic project
  http://www.openwrt.org
- **LTIB**, developed mainly by Freescale. Good support for Freescale boards, but small community
  http://ltib.org/
- **OpenEmbedded**, more flexible but also far more complicated
  http://www.openembedded.org
- Vendor specific tools (silicon vendor or embedded Linux vendor)

Buildroot (1)

- Allows to build a toolchain, a root filesystem image with many applications and libraries, a bootloader and a kernel image
  - Or any combination of the previous items
- Supports building uClibc toolchains only, but can use external uClibc or glibc toolchains
- Over 500+ applications or libraries integrated, from basic utilities to more elaborate software stacks: X.org, Gstreamer, Qt, Gtk, Webkit, etc.
- Good for small to medium embedded systems, with a fixed set of features
  - No support for generating packages (.deb or .ipk)
  - Needs complete rebuild for most configuration changes.
- Active community, releases published every 3 months.

Buildroot (2)

- Configuration takes place through a `*config` interface similar to the kernel
  `make menuconfig`
- Allows to define
  - Architecture and specific CPU
  - Toolchain configuration
  - Set of applications and libraries to integrate
  - Filesystem images to generate
  - Kernel and bootloader configuration
- Build by just running
  `make`

Buildroot: adding a new package (1)

- A package allows to integrate a user application or library to Buildroot
- Each package has its own directory (such as `package/gqview`). This directory contains:
  - A `Config.in` file (mandatory), describing the configuration options for the package. At least one is needed to enable the package. This file must be sourced from `package/Config.in`
  - A `gqview.mk` file (mandatory), describing how the package is built.
  - Patches (optional). Each file of the form `gqview-*.patch` will be applied as a patch.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Buildroot: adding a new package (2)

- For a simple package with a single configuration option to enable/disable it, the `Config.in` file looks like:

```
config BR2_PACKAGE_GQVIEW
        bool "gqview"
        depends on BR2_PACKAGE_LIBGTK2
        help
          GQview is an image viewer for Unix operating systems

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

- It must be sourced from `package/Config.in`:

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


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Buildroot: adding new package (3)

- 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 = host-pkg-config libgtk2

$(eval $(call AUTOTARGETS,package,gqview))
```

- The last argument of the `AUTOTARGETS` macro, the prefix of all variables must be identical to the suffix of the main configuration option `BR2_PACKAGE_GQVIEW`
- The `AUTOTARGETS` infrastructure knows how to build autotools packages. A more generic `GENTARGETS` infrastructure is available for packages not using the autotools as their build system.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

OpenEmbedded

- The most versatile and powerful embedded Linux build system
  - A collection of recipes (`.bb` files)
  - A tool that processes the recipes: `bitbake`
- Integrates 2000+ application and libraries, is highly configurable, can generate packages (`.ipk`) to make the system customizable, supports multiple versions/variants of the same package, no need for full rebuild when the configuration is changed.
- Configuration takes place by editing various configuration files
- Good for larger embedded Linux systems, or people looking for more configurability and extensibility
- Drawbacks: no stable releases, very steep learning curve, very long first build.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Distributions (1)

Debian GNU/Linux, http://www.debian.org

- Available on ARM, MIPS and PowerPC architectures
- Provides a ready-to-use filesystem with all the software you need.
- Huge flexibility thanks to the package management system, but only works only systems with enough storage size (> 300 MB) and RAM (> 64 MB).
- Software is compiled natively by default.
- You can build your own root filesystem images on x86 by using the `debootstrap` command.
- Emdebian is a project to make Debian better for embedded systems: leverage Debian package descriptions, but reduces dependencies, smaller configuration, removes documentation, supports uClibc... See http://emdebian.org.

Distributions (2)

Ubuntu GNU/Linux

- Based on Debian, same benefits
- New release every 6 months,
  supported for 18 months or even 3 years.
- Supported on ARM, but only on Cortex A8 and beyond.
  Supplies Thumb2 binaries. Neon not supported.
- Good solution for mobile multimedia devices.

Others

- Fedora also has support for ARM, but not actively maintained.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Embedded distributions

Distributions designed for specific types of devices

- Meego: http://meego.com/
  Distribution targeting phones, media players, netbooks, TVs and In Vehicle Infotainment.
  Backed by Intel and Nokia.
- Android: http://www.android.com/
  Google's distribution for phones and tablet PCs.
  Except the Linux kernel, very different userspace than other Linux distributions. Very successful, lots of applications available (many proprietary).
- Ångström: http://www.angstrom-distribution.org/
  Targets PDAs and webpads (Siemens Simpad...)
  Binaries available for `arm` little endian.

Practical lab – Buildroot

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

GNU / Linux workstation

Emulators

qemu

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!).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 mainline 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!

Wind River

Wind River Linux:
http://www.windriver.com/products/linux/

WIND RIVER

- New market leader, recently acquired by Intel
- 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).

Montavista

http://www.mvista.com/

The second 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 mainline Linux.
- Graphical development tools are proprietary.

TimeSys

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.

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!

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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!

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

- 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
- Linux Plumbers
  http://linuxplumbersconf.org
  Conference on the low-level plumbing of Linux: kernel, audio, power management, device management, multimedia, etc.
- Fosdem: http://fosdem.org (Brussels, February)
  For developers. Presentations about system development.
- Don't miss our free conference videos on
  http://free-electrons.com/community/videos/conferences/!

Embedded Linux application development

Michael Opdenacker
Thomas Petazzoni
Free Electrons

.S .h make .c .c++

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/appdev
Corrections, suggestions, contributions and translations are welcome!

Contents

Application development

- Developing applications on embedded Linux
- Building your applications

Source management

- Integrated development environments (IDEs)
- Version control systems

Debugging and analysis tools

- Debuggers
- Memory checkers
- System analysis

Development environments

- Developing on Windows

Developing applications on embedded Linux

Application development

- An embedded Linux system is just a normal Linux system, with usually a smaller selection of components
- In terms of application development, developing on embedded Linux is exactly the same as developing on a desktop Linux system
- All existing skills can be re-used, without any particular adaptation
- All existing libraries, either third-party or in-house, can be integrated into the embedded Linux system
  - Taking into account, of course, the limitation of the embedded systems in terms of performance, storage and memory

Programming language

- The default programming language for system-level application in Linux is usually C
  - The C library is already present on your system, nothing to add
- C++ can be used for larger applications
  - The C++ library must be added to the system
  - Some libraries, including Qt, are developed in C++ so they need the C++ library on the system anyway
- Scripting languages can also be useful for quick application development, web applications or scripts
  - But they require an interpreter on the embedded system and have usually higher memory consumption and slightly lower performances
- Languages: Python, Perl, Lua, Ada, Fortran, etc.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

C library or higher-level libraries ?

- For many applications, the C library already provides a relatively large set of features
  - file and device I/O, networking, threads and synchronization, inter-process communication
  - Thoroughly described in the glibc manual, or in any « Linux system programming » book
  - However, the API carries a lot of history and is not necessarily easy to grasp for new comers
- Therefore, using a higher level framework, such as Qt or the Gtk stack, might be a good idea
  - These frameworks are not only graphical libraries, their core is separate from the graphical part
  - But of course, these libraries have some memory and storage footprint, in the order of a few megabytes

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Building your applications

- For simple applications that do not need to be really portable or provide compile-time configuration options, a simple Makefile will be sufficient
- For more complicated applications, or if you want to be able to run your application on a desktop Linux PC and on the target device, using a build system is recommended
  - Look at the *autotools* (ancient, complicated but very widely used) or *CMake* (modern, simpler, smaller but growing user base)
- The QT library is a special case, since it comes with its own build system for applications, called *qmake*

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Simple Makefile (1)

- Case of an application that only uses the C library, contains two source files and generates a single binary

```
CROSS_COMPILE?=arm-linux-
CC=$(CROSS_COMPILE)gcc
OBJS=foo.o bar.o

all: foobar

foobar: $(OBJS)
   $(CC) -o $@ $^

clean:
   $(RM) -f foobar $(OBJS)
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Simple Makefile (2)

- Case of an application that uses the Glib and the GPS libraries

```
CROSS_COMPILE?=arm-linux-
LIBS=libgps glib-2.0
OBJS=foo.o bar.o

CC=$(CROSS_COMPILE)gcc
CFLAGS=$(shell pkg-config --cflags $(LIBS))
LDFLAGS=$(shell pkg-config --libs $(LIBS))

all: foobar

foobar: $(OBJS)
   $(CC) -o $@ $^ $(LDFLAGS)

clean:
   $(RM) -f foobar $(OBJS)
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

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

ıriable	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 R
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

Ruby debugger

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Eclipse (2)

- Eclipse is actually a platform composed of many projects: http://www.eclipse.org/projects/
  - Some projects are dedicated at integrating into Eclipse features useful for embedded developers (cross-compilation, remote development, remote debugging, etc.)
- 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!

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Source management

Version control systems

Version control systems

Real projects can't do without them

- Allow multiple developers to contribute on the same project. Each developer can see the latest changes from the others, or choose to stick with older versions of some components.
- Allow to keep track of changes, and revert them if needed.
- Allow developers to have their own development branch (branching)
- Supposed to help developers resolving conflicts with different branches (merging)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Traditional version control systems

Rely on a central repository. The most popular open-source ones:

- CVS - Concurrent Versions System
  - Still quite popular
  - Should no longer be used for new projects
  - http://en.wikipedia.org/wiki/Concurrent_Versions_System
- Subversion
  - Created as a replacement of CVS, removing many of its limitations.
  - Commits on several files, proper renaming support, better performances, etc.
  - The user interface is very similar to CVS
  - http://en.wikipedia.org/wiki/Subversion_(software)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Distributed source control systems (1)

No longer have a central repository

- More adapted to the way the Free Software community develops software and organizes
- Allow each developer to have a local full history of the project, to create local branches. Makes each developer's work easier.
- People get working copies from other people's working copies, and exchange changes between themselves. Branching and merging is made easier.
- Make it easier for new developers to join, making their own experiments without having to apply for repository access.

Distributed source control systems (2)

- Git
  - Initially designed and developed by Linus Torvalds for Linux kernel development
  - Extremely popular in the community, and used by more and more projects (kernel, U-Boot, Barebox, uClibc, GNOME, X.org, etc.)
  - Outstanding performance, in particular in big projects
  - http://en.wikipedia.org/wiki/Git_(software)
- Mercurial
  - Another system, created with the same goals as Git.
  - Used by some big projects too
  - http://en.wikipedia.org/wiki/Mercurial

http://en.wikipedia.org/wiki/Version_control_systems#Distributed_revision_control

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

GDB crash course

- A few useful GDB commands
  - **break foobar**
    puts a breakpoint at the entry of function foobar()
  - **break foobar.c:42**
    puts a breakpoint in foobar.c, line 42
  - **print var** or **print task->files[0].fd**
    prints the variable var, or a more complicated reference. GDB can also nicely display structures with all their members
  - **continue**
    continue the execution
  - **next**
    continue to the next line, stepping over function calls
  - **step**
    continue to the next line, entering into subfunctions

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

GDB graphical front-ends

- DDD - Data Display Debugger
  http://www.gnu.org/software/ddd/
  A popular graphical front-end, with advanced data plotting capabilities.
- GDB/Insight
  http://sourceware.org/insight/
  From the GDB maintainers.
- KDbg
  http://www.kdbg.org/
  Another front-end, for the K Display Environment.
- Integration with other IDEs: Eclipse, Emacs, KDevelop, etc.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 100 KB on `arm`).

Remote debugging: architecture

Remote debugging: usage

- On the target, run a program through `gdbserver`.
  Program execution will not start immediately.
  `gdbserver localhost:<port> <executable> <args>`
  `gdbserver /dev/ttyS0 <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:
    `gdb > target remote <target>:<port>` (networking)
    `gdb > target remote /dev/ttyS0` (serial link)
  - To tell `gdb` where shared libraries are:
    `gdb > set sysroot <library-path>` (without `lib/`)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Post mortem analysis

- When an application crashes due to a segmentation fault and the application was not under control of a debugger, we get no information about the crash
- Fortunately, Linux can generate a *core* file that contains the image of the application memory at the moment of the crash, and gdb can use this *core* file to let us analyze the state of the crashed application
- On the target
  - Use `ulimit -c unlimited` to enable the generation of a *core* file when a crash occurs
- On the host
  - After the crash, transfer the *core* file from the target to the host, and run `ARCH-linux-gdb -c core-file application-binary`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Valgrind (1)

http://valgrind.org/

- GNU GPL Software suite for debugging and profiling programs.
- Supported platforms: Linux on `x86`, `x86_64`, `ppc32`, `ppc64` and `arm` (armv7 only: Cortex A8, A9 and A5)
- 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 cpu 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#howworks

Debugging and analysis tools

System analysis

strace

System call tracer
http://sourceforge.net/projects/strace/

- Available on all GNU/Linux systems
  Can be built by your cross-compiling toolchain generator.
- Allows to see what any of your processes is doing:
  accessing files, allocating memory...
  Often sufficient to find simple bugs.
- Usage:
  `strace <command>` (starting a new process)
  `strace -p<pid>` (tracing an existing process)

See `man strace` for details.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

strace example output

```
> strace cat Makefile
execve("/bin/cat", ["cat", "Makefile"], [/* 38 vars */]) = 0
brk(0)                                  = 0x98b4000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7f85000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=111585, ...}) = 0
mmap2(NULL, 111585, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7f69000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/tls/i686/cmov/libc.so.6", O_RDONLY) = 3
read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320h\1\0004\0\0\0\344"..., 512)
= 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442180, ...}) = 0
mmap2(NULL, 1451632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb7e06000
mprotect(0xb7f62000, 4096, PROT_NONE)   = 0
mmap2(0xb7f63000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x15c) = 0xb7f63000
mmap2(0xb7f66000, 9840, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0xb7f66000
close(3)                                = 0
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

ltrace example output

```
ltrace 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
...
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

KCachegrind screenshot

Labs – App. development and remote debugging

Application development

- Compile your own application with the DirectFB libraries

Remote debugging

- 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).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.
- Cygwin is very slow.

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 VMware.

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Real-time in embedded Linux systems

Michael Opdenacker
Thomas Petazzoni
Gilles Chanteperdrix
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/realtime
Corrections, suggestions, contributions and translations are welcome!

Introduction

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Embedded Linux and real time

- Due to its advantages, Linux and the open-source softwares are more and more commonly used in embedded applications
- However, some applications also have real-time constraints
- They, at the same time, want to
  - Get all the nice advantages of Linux: hardware support, components re-use, low cost, etc.
  - Get their real-time constraints met

Embedded Linux and real time

- Linux is an operating system part of the large Unix family
- It was originally designed as a time-sharing system
  - The main goal is to get the best throughput from the available hardware, by making the best possible usage of resources (CPU, memory, I/O)
  - Time determinism is not taken into account
- On the opposite, real-time constraints imply time determinism, even at the expense of lower global throughput
- Best throughput and time determinism are contradictory requirements

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux and real-time approaches

- Over time, two major approaches have been taken to bring real-time requirements into Linux
- **Approach 1**
  - Improve the Linux kernel itself so that it matches real-time requirements, by providing bounded latencies, real-time APIs, etc.
  - Approach taken by the mainline Linux kernel and the PREEMPT_RT project.
- **Approach 2**
  - Add a layer below the Linux kernel that will handle all the real-time requirements, so that the behaviour of Linux doesn't affect real-time tasks.
  - Approach taken by RTLinux, RTAI and Xenomai

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Approach 1

Improving the main Linux kernel with PREEMPT_RT

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Understanding latency

- When developing real-time applications with a system such as Linux, the typical scenario is the following
  - An event from the physical world happens and gets notified to the CPU by means of an interrupt
  - The interrupt handler recognizes and handles the event, and then wake-up the user-space task that will react to this event
  - Some time later, the user-space task will run and be able to react to the physical world event
- Real-time is about providing guaranteed worst case latencies for this reaction time, called *latency*

Linux kernel latency components

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

Interrupt latency

Time elapsed before executing the interrupt handler

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Source of interrupt latency

- One of the concurrency prevention mechanism used in the kernel is the **spinlock**
- It has several variants, but one of the variant commonly used to prevent concurrent accesses between a process context and an interrupt context works by disabling interrupts
- Critical sections protected by spinlocks, or other section in which interrupts are explictly disabled will delay the beginning of the execution of the interrupt handler
  - The duration of these critical sections is unbounded
- Other possible source: shared interrupts

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Interrupt handler duration

Time taken to execute the interrupt handler

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Interrupt handler implementation

- In Linux, many interrupt handlers are split in two parts
  - A top-half, started by the CPU as soon as interrupt are enabled. It runs with the interrupt line disabled and is supposed to complete as quickly as possible.
  - A bottom-half, scheduled by the top-half, which starts after all pending top-half have completed their execution.
- Therefore, for real-time critical interrupts, bottom-half shouldn't be used: their execution is delayed by all other interrupts in the system.

Scheduler latency

Time elapsed before executing the scheduler

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Understanding preemption (1)

- The Linux kernel is a preemptive operating system
- When a task runs in user-space mode and gets interrupted by an interruption, if the interrupt handler wakes up another task, this task can be scheduled as soon as we return from the interrupt handler.

Understanding preemption (2)

- However, when the interrupt comes while the task is executing a system call, this system call has to finish before another task can be scheduled.
- By *default,* the Linux kernel does not do *kernel preemption.*
- This means that the time before which the scheduler will be called to schedule another task is unbounded.

Scheduler duration

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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Other non-deterministic mechanisms

- Outside of the critical path detailed previously, other non-deterministic mechanisms of Linux can affect the execution time of real-time tasks
- Linux is highly based on virtual memory, as provided by an MMU, so that memory is allocated on demand. Whenever an application accesses code or data for the first time, it is loaded on demand, which can creates huge delays.
- Many C library services or kernel services are not designed with real-time constraints in mind.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Priority inversion

A process with a low priority might hold a lock needed by a higher priority process, effectively reducing the priority of this process. Things can be even worse if a middle priority process uses the CPU.

Interrupt handler priority

In Linux, interrupt handlers are executed directly by the CPU interrupt mechanisms, and not under control of the Linux scheduler. Therefore, all interrupt handlers have an higher priority than all tasks running on the system.

The PREEMPT_RT project

- Long-term project lead by Linux kernel developers Ingo Molnar, Thomas Gleixner and Steven Rostedt
  - https://rt.wiki.kernel.org
- The goal is to gradually improve the Linux kernel regarding real-time requirements and to get these improvements merged into the mainline kernel
  - PREEMPT_RT development works very closely with the mainline development
- Many of the improvements designed, developed and debugged inside PREEMPT_RT over the years are now part of the mainline Linux kernel
  - The project is a long-term branch of the Linux kernel that ultimately should disappear as everything will have been merged

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Improvements in the mainline kernel

- Coming from the PREEMPT_RT project
- Since the beginning of 2.6
  - O(1) scheduler
  - Kernel preemption
  - Better POSIX real-time API support
- Since 2.6.18
  - Priority inheritance support for mutexes
- Since 2.6.21
  - High-resolution timers
- Since 2.6.30
  - Threaded interrupts
- Since 2.6.33
  - Spinlock annotations

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 (interrupts, exceptions, 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).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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), but a rescheduling point occurs when exiting the outer critical section, in case a preemption opportunity would have been signaled while in the critical section.
- Typically for desktop or embedded systems with latency requirements in the milliseconds range.
- Still a relatively minor impact on throughput.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Priority inheritance

- One classical solution to the priority inversion problem is called priority inheritance
  - The idea is that when a task of a low priority holds a lock requested by an higher priority task, the priority of the first task gets temporarly raised to the priority of the second task : it has *inherited* its priority.
- In Linux, since 2.6.18, mutexes support priority inheritance
- In userspace, priority inheritance must be explictly enabled on a per-mutex basis.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

High resolution timers

- The resolution of the timers used to be bound to the resolution of the regular system tick
  - Usually 100 Hz or 250 Hz, depending on the architecture and the configuration
  - A resolution of only 10 ms or 4 ms.
  - Increasing the regular system tick frequency is not an option as it would consume too much resources
- The high-resolution timers infrastructure, merged in 2.6.21, allows to use the available hardware timers to program interrupts at the right moment.
  - Hardware timers are multiplexed, so that a single hardware timer is sufficient to handle a large number of software-programmed timers.
  - Usable directly from user-space using the usual timer APIs

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Threaded interrupts

- To solve the interrupt inversion problem, PREEMPT_RT has introduced the concept of threaded interrupts
- The interrupt handlers run in normal kernel threads, so that the priorities of the different interrupt handlers can be configured
- The real interrupt handler, as executed by the CPU, is only in charge of masking the interrupt and waking-up the corresponding thread
- The idea of threaded interrupts also allows to use sleeping spinlocks (see later)
- Merged since 2.6.30, the conversion of interrupt handlers to threaded interrupts is not automatic : drivers must be modified
- In PREEMPT_RT, all interrupt handlers are switched to threaded interrupts

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

PREEMPT_RT specifics

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

CONFIG_PREEMPT_RT (1)

- The PREEMPT_RT patch adds a new « level » of preemption, called CONFIG_PREEMPT_RT
- This level of preemption replaces all kernel spinlocks by mutexes (or so-called sleeping spinlocks)
  - Instead of providing mutual exclusion by disabling interrupts and preemption, they are just normal locks : when contention happens, the process is blocked and another one is selected by the scheduler
  - Works well with threaded interrupts, since threads can block, while usual interrupt handlers could not
  - Some core, carefully controlled, kernel spinlocks remain as normal spinlocks

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

CONFIG_PREEMPT_RT (2)

- With CONFIG_PREEMPT_RT, virtually all kernel code becomes preemptible
  - An interrupt can occur at any time, when returning from the interrupt handler, the woken up process can start immediately
- This is the last big part of PREEMPT_RT that isn't fully in the mainline kernel yet
  - Part of it has been merged in 2.6.33 : the spinlock annotations. The spinlocks that must remain as spinning spinlocks are now differentiated from spinlocks that can be converted to sleeping spinlocks. This has reduced a lot the PREEMPT_RT patch size !

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Threaded interrupts

- The mechanism of threaded interrupts in PREEMPT_RT is still different from the one merged in mainline
- In PREEMPT_RT, all interrupt handlers are unconditionally converted to threaded interrupts.
- This is a temporary solution, until interesting drivers in mainline get gradually converted to the new threaded interrupt API that has been merged in 2.6.30.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Setting up PREEMPT_RT

PREEMPT_RT setup (1)

- PREEMPT_RT is delivered as a patch against the mainline kernel
  - Best to have a board supported by the mainline kernel, otherwise the PREEMPT_RT patch may not apply and may require some adaptations
- Many official kernel releases are supported, but not all. For example, 2.6.31 and 2.6.33 are supported, but not 2.6.32.
- Quick set up
  - Download and extract mainline kernel
  - Download the corresponding PREEMPT_RT patch
  - Apply it to the mainline kernel tree

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

PREEMPT_RT setup (2)

- In the kernel configuration, be sure to enable
  - CONFIG_PREEMPT_RT
  - High-resolution timers
- Compile your kernel, and boot
- You are now running the real-time Linux kernel
- Of course, some system configuration remains to be done, in particular setting appropriate priorities to the interrupt threads, which depend on your application.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Real-time application development

Development and compilation

- No special library is needed, the POSIX realtime API is part of the standard C library
- The glibc or eglibc C libraries are recommended, as the support of some real-time features is not available yet in uClibc
  - Priority inheritance mutexes or NPTL on some architectures, for example
- Compile a program
  - `ARCH-linux-gcc -o myprog myprog.c -lrt`
- To get the documentation of the POSIX API
  - Install the `manpages-posix-dev` package
  - Run `man functioname`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Process, thread ?

- Confusion about the terms «process», «thread» and «task»
- In Unix, a process is created using `fork()` and is composed of
  - An address space, which contains the program code, data, stack, shared libraries, etc.
  - One thread, that starts executing the main() function.
  - Upon creation, a process contains one thread
- Additional threads can be created inside an existing process, using `pthread_create()`
  - They run in the same address space as the initial thread of the process
  - They start executing a function passed as argument to `pthread_create()`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Process, thread: kernel point of view

- The kernel represents each thread running in the system by a structure of type task_struct
- From a scheduling point of view, it makes no difference between the initial thread of a process and all additional threads created dynamically using pthread_create()

Process after `fork()`

Same process after `pthread_create()`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Creating threads

- Linux support the POSIX thread API
- To create a new thread
  -
```
pthread_create(pthread_t *thread,
               pthread_attr_t *attr,
               void *(*routine)(*void*),
               void *arg);
```
  - The new thread will run in the same address space, but will be scheduled independently
- Exiting from a thread
  - `pthread_exit(void *value_ptr);`
- Waiting for a thread termination
  - `pthread_join(pthread_t *thread, void **value_ptr);`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Scheduling classes (1)

- The Linux kernel scheduler support different scheduling classes
- The default class, in which processes are started by default is a time-sharing class
  - All processes, regardless of their priority, get some CPU time
  - The proportion of CPU time they get is dynamic and affected by the nice value, which ranges from -20 (highest) to 19 (lowest). Can be set using the nice or renice commands
- The real-time classes **SCHED_FIFO** and **SCHED_RR**
  - The highest priority process gets all the CPU time, until it blocks.
  - In SCHED_RR, round-robin scheduling between the processes of the same priority. All must block before lower priority processes get CPU time.
  - Priorities ranging from 0 (lowest) to 99 (highest)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Scheduling classes (2)

- An existing program can be started in a specific scheduling class with a specific priority using the chrt command line tool
  - Example: `chrt -f 99 ./myprog`
- The `sched_setscheduler()` API can be used to change the scheduling class and priority of a process
  - `int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param);`
  - `policy` can be `SCHED_OTHER, SCHED_FIFO, SCHED_RR`, etc.
  - `param` is a structure containing the priority

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Scheduling classes (3)

- The priority can be set on a per-thread basis when a thread is created :

```
struct sched_param parm;
pthread_attr_t attr;

pthread_attr_init(&attr);
pthread_attr_setinheritsched(&attr,
                             PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
parm.sched_priority = 42;
pthread_attr_setschedparam(&attr, &parm);
```

- Then the thread can be created using `pthread_create()`, passing the `attr` structure.
- Several other attributes can be defined this way: stack size, etc.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Memory locking

- In order to solve the non-determinism introduced by virtual memory, memory can be locked
  - Guarantee that the system will keep it allocated
  - Guarantee that the system has pre-loaded everything into memory
- `mlockall(MCL_CURRENT | MCL_FUTURE);`
  - Locks all the memory of the current address space, for currently mapped pages and pages mapped in the future
- Other, less useful parts of the API: `munlockall, mock, munlock.`
- Watch out for non-currently mapped pages
  - Stack pages
  - Dynamically-allocated memory

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Mutexes

- Allows mutual exclusion between two threads in the same address space
  - Initialization/destruction

```
pthread_mutex_init(pthread_mutex_t *mutex, const
pthread_mutexattr_t *mutexattr);
pthread_mutex_destroy(pthread_mutex_t *mutex);
```

  - Lock/unlock

```
pthread_mutex_lock(pthread_mutex_t *mutex);
pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- Priority inheritance must explictly be activated

```
pthread_mutexattr_t attr;
pthread_mutexattr_init (&attr);
pthread_mutexattr_getprotocol
      (&attr, PTHREAD_PRIO_INHERIT);
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Timers

- **`timer_create`**`(clockid_t clockid,`
  `struct sigevent *evp,`
  `timer_t *timerid)`
  - Create a timer. **`clockid`** is usually CLOCK_MONOTONIC. **`sigevent`** defines what happens upon timer expiration : send a signal or start a function in a new thread. **`timerid`** is the returned timer identifier.
- **`timer_settime`**`(timer_t timerid, int flags,`
  `struct itimerspec *newvalue,`
  `struct itimerspec *oldvalue)`
  - Configures the timer for expiration at a given time.
- **`timer_delete`**`(timer_t timerid)`, delete a timer
- **`clock_getres`**`()`, get the resolution of a clock
- Other functions: `timer_getoverrun()`, `timer_gettime()`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Signals

- Signals are an asynchronous notification mechanism
- Notification occurs either
  - By the call of a signal handler. Be careful with the limitations of signal handlers!
  - By being unblocked from the **`sigwait()`**, **`sigtimedwait()`** or **`sigwaitinfo()`** functions. Usually better.
- Signal behaviour can be configured using **`sigaction()`**
- Mask of blocked signals can be changed with **`pthread_sigmask()`**
- Delivery of a signal using **`pthread_kill()`** or **`tgkill()`**
- All signals between **`SIGRTMIN`** and **`SIGRTMAX`**, 32 signals under Linux.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Inter-process communication

- Semaphores
  - Usable between different processes using named semaphores
  - **`sem_open()`, `sem_close()`, `sem_unlink()`, `sem_init()`, `sem_destroy()`, `sem_wait()`, `sem_post()`**, etc.
- Message queues
  - Allows processes to exchange data in the form of messages.
  - **`mq_open()`, `mq_close()`, `mq_unlink()`, `mq_send()`, `mq_receive()`**, etc.
- Shared memory
  - Allows processes to communicate by sharing a segment of memory
  - **`shm_open()`, `ftruncate()`, `mmap()`, `munmap()`, `close()`, `shm_unlink()`**

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Debugging real-time latencies

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`
- 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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Scheduling latency tracer

`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 and the corresponding trace:
```
cat /debug/tracing/tracing_max_latency
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Approach 2

Real-time extensions to the Linux kernel

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux 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.”
- Free version still advertised by Wind River on http://www.rtlinuxfree.com, but no longer a community project.

RTAI

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.
- Only really actively maintained on x86
- May offer slightly better latencies than Xenomai, at the expense of a less maintainable and less portable code base
- Since RTAI is not really maintained on ARM and other embedded architectures, our presentation is focused on Xenomai.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Xenomai project

http://www.xenomai.org/

- Started in 2001 as a project aiming at emulating traditional RTOS.
- Initial goals: facilitate the porting of programs to GNU / Linux.
- Initially related to the RTAI project (as the RTAI / fusion branch), now independent.
- Skins mimicking the APIs of traditional RTOS such as VxWorks, pSOS+, and VRTXsa as well as the POSIX API, and a “native” API.
- Aims at working both as a co-kernel and on top of PREEMPT_RT in the upcoming 3.0 branch.
- Will never be merged in the mainline kernel.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Xenomai architecture

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

The Adeos interrupt pipeline abstraction

- From Adeos point of view, guest OSes are prioritized domains.
- For each event (interrupts, exceptions, syscalls, etc...), the various domains may handle the event or pass it down the pipeline.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Adeos virtualized interrupts disabling

- Each domain may be “stalled”, meaning that it does not accept interrupts.
- Hardware interrupts are not disabled however (except for the domain leading the pipeline), instead the interrupts received during that time are logged and replayed when the domain is unstalled.

Adeos additional features

- The Adeos I-pipe patch implement additional features, essential for the implementation of the Xenomai real-time extension:
  - Disables on-demand mapping of kernel-space vmalloc/ioremap areas.
  - Disables copy-on-write when real-time processes are forking.
  - Allow subscribing to event allowing to follow progress of the Linux kernel, such as Linux system calls, context switches, process destructions, POSIX signals, FPU faults.
  - On the ARM architectures, integrates the FCSE patch, which allows to reduce the latency induced by cache flushes during context switches.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Xenomai features

- Factored real-time core with skins implementing various real-time APIs
- Seamless support for hard real-time in user-space
- No second-class citizen, all ports are equivalent feature-wise
- Xenomai support is as much as possible independent from the Linux kernel version (backward and forward compatible when reasonable)
- Each Xenomai branch has a stable user/kernel ABI
- Timer system based on hardware high-resolution timers
- Per-skin time base which may be periodic
- RTDM skin allowing to write real-time drivers

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Xenomai user-space real-time support.

- Xenomai supports real-time in user-space on 5 architectures, including 32 and 64 bits variants.
- Two modes are defined for a thread
  - the primary mode, where the thread is handled by Xenomai scheduler
  - the secondary mode, when it is handled by Linux scheduler.
- Thanks to the services of the Adeos I-pipe service, Xenomai system calls are defined.
  - A thread migrates from secondary mode to primary mode when such a system call is issued
  - It migrates from primary mode to secondary mode when a Linux system call is issued, or to handle gracefully exceptional events such as exceptions or Linux signals.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Life of a Xenomai application

- Xenomai applications are started like normal Linux processes, they are initially handled by the Linux scheduler and have access to all Linux services
- After their initialization, they declare themselves as *real-time* application, which migrates them to primary mode. In this mode:
  - They are scheduled directly by the Xenomai scheduler, so they have the real-time properties offered by Xenomai
  - They don't have access to any Linux service, otherwise they get migrated back to secondary mode and looses all real-time properties
  - They can only use device drivers that are implemented in Xenomai, not the ones of the Linux kernel
- Need to implement device drivers in Xenomai, and to split real-time and non real-time parts of your applications.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Real Time Driver Model (RTDM)

- An approach to unify the interfaces for developing device drivers and associated applications under real-time Linux
  - An API very similar to the native Linux kernel driver API
- Allows the development, in kernel space, of
  - Character-style device drivers
  - Network-style device drivers
- See the whitepaper on http://www.xenomai.org/documentation/xenomai-2.4/pdf/RTDM-and-Applications.pdf
- Current notable RTDM based drivers:
  - Serial port controllers;
  - RTnet UDP/IP stack;
  - RT socket CAN, drivers for CAN controllers;
  - Analogy, fork of the Comedy project, drivers for acquisition cards.

Setting up Xenomai

How to build Xenomai

- Download Xenomai sources at http://download.gna.org/xenomai/stable/
- Download one of the Linux versions supported by this release (see `ksrc/arch/<arch>/patches/`)
- Since version 2.0, split kernel/user building model.
- Kernel uses a script called script/prepare-kernel.sh which integrates Xenomai kernel-space support in the Linux sources.
- Run the kernel configuration menu.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linux options for Xenomai configuration

File Edit Option Help

Option

- General setup
  - RCU Subsystem
  - ☐ Control Group support
  - ☐ Configure standard kernel fea
- ☑ Enable loadable module support
- Enable the block layer (NEW)
  - IO Schedulers
- Real-time sub-system
- System Type
  - Atmel AT91 System-on-Chip
- Bus support
  - ☐ PCCard (PCMCIA/CardBus) s
- Kernel Features
- Boot options
- CPU Power Management
- Floating point emulation
- Userspace binary formats
- Power management options

Option

- ☑ Xenomai
- ☑ Nucleus
  - ☑ Pervasive real-time support in user-space
  - ☑ Optimize as pipeline head
  - ☐ Extra scheduling classes
  - (32) Number of pipe devices
  - (512) Number of registry slots
  - (256) Size of the system heap (Kb)
  - (128) Size of the private stack pool (Kb)
  - (12) Size of private semaphores heap (Kb)
  - (12) Size of global semaphores heap (Kb)
  - ☑ Statistics collection

Xenomai (XENOMAI)

Xenomai is a real-time extension to the Linux kernel. Note that Xenomai relies on Adeos interrupt pipeline (CONFIG_IPIPE option) to be enabled, so enabling this option selects the CONFIG_IPIPE option.

Xenomai user-space support

- User-space libraries are compiled using the traditional autotools
  - `./configure --target=arm-linux && make && make DESTDIR=/your/rootfs/ install`
- The xeno-config script, installed when installing Xenomai user-space support helps you compiling your own programs.
- See Xenomai's `examples` directory.
- Installation details may be found in the README.INSTALL guide.
- For an introduction on programming with the native API, see: http://www.xenomai.org/documentation/branches/v2.3.x/pdf/Native-API-Tour-rev-C.pdf
- For an introduction on programming with the POSIX API, see: http://www.xenomai.org/index.php/Porting_POSIX_applications_to_Xenomai

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Developing applications on Xenomai

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

The POSIX skin

- The POSIX skin allows to recompile without changes a traditional POSIX application so that instead of using Linux real-time services, it uses Xenomai services
  - Clocks and timers, condition variables, message queues, mutexes, semaphores, shared memory, signals, thread management
  - Good for existing code or programmers familiar with the POSIX API
- Of course, if the application uses any Linux service that isn't available in Xenomai, it will switch back to secondary mode
- To link an application against the POSIX skin

```
DESTDIR=/path/to/xenomai/
export DESTDIR
CFL=`$DESTDIR/bin/xeno-config --posix-cflags`
LDF=`$DESTDIR/bin/xeno-config --posix-ldflags`
ARCH-gcc $CFL -o rttest rttest.c $LDF
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Communication with a normal task

- If a Xenomai real-time application using the POSIX skin wishes to communicate with a separate non-real-time application, it must use the *rtipc* mechanism
- In the Xenomai application, create an `IPCPROTO_XDDP` socket

```
socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP);
setsockopt(s, SOL_RTIPC, XDDP_SETLOCALPOOL,&poolsz,
sizeof(poolsz));
memset(&saddr, 0, sizeof(saddr));
saddr.sipc_family = AF_RTIPC;
saddr.sipc_port = MYAPPIDENTIFIER;
ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
```

  - And then the normal socket API `sendto()` / `recvfrom()`
- In the Linux application
  - Open `/dev/rtpX`, where X is the XDDP port
  - Use `read()` and `write()`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

The native API (1)

- A Xenomai-specific API for developing real-time tasks
  - Usable both in user-space and kernel space. Development of tasks in user-space is the preferred way.
  - More coherent and more flexible API than the POSIX API. Easier to learn and understand. Certainly the way to go for new applications.
- Applications should include `<native/service.h>`, where service can be `alarm, buffer, cond, event, heap, intr, misc, mutex, pipe, queue, sem, task, timer`
- To compile applications :

```
DESTDIR=/path/to/xenomai/
export DESTDIR
CFL=`$DESTDIR/bin/xeno-config --xeno-cflags`
LDF=`$DESTDIR/bin/xeno-config --xeno-ldflags`
ARCH-gcc $CFL -o rttest rttest.c $LDF -lnative
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

The native API (2)

- Task management services
  - `rt_task_create(), rt_task_start(), rt_task_suspend(), rt_task_resume(), rt_task_delete(), rt_task_join(), etc.`
- Counting semaphore services
  - `rt_sem_create(), rt_sem_delete(), rt_sem_p(), rt_sem_v(), etc.`
- Message queue services
  - `rt_queue_create(), rt_queue_delete(), rt_queue_alloc(), rt_queue_free(), rt_queue_send(), rt_queue_receive(), etc.`
- Mutex services
  - `rt_mutex_create(), rt_mutex_delete(), rt_mutex_acquire(), rt_mutex_release(), etc.`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

The native API (3)

- Alarm services
  - `rt_alarm_create(), rt_alarm_delete(), rt_alarm_start(), rt_alarm_stop(), rt_alarm_wait(), etc.`
- Memory heap services
  - Allows to share memory between processes and/or to pre-allocate a pool of memory
  - `rt_heap_create(), rt_heap_delete(), rt_heap_alloc(), rt_heap_bind()`
- Condition variable services
  - `rt_cond_create(), rt_cond_delete(), rt_cond_signal(), rt_cond_broadcast(), rt_cond_wait(), etc.`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Xenomai and normal task communication

- Using *rt_pipes*
- In the native Xenomai application, use the Pipe API
  - `rt_pipe_create()`, `rt_pipe_delete()`, `rt_pipe_receive()`, `rt_pipe_send()`, `rt_pipe_alloc()`, `rt_pipe_free()`
- In the normal Linux application
  - Open the corresponding `/dev/rtpX` file, the minor is specified at `rt_pipe_create()` time
  - Then, just `read()` and `write()` to the opened file

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Real-time approaches

The following table is Paul Mac Kenney's summary of his own article describing the various approaches for real-time on Linux:

Approach	Quality	Inspection	API	Complexity	Fault isolation	HW/SW Configs
Vanilla Linux	10s of ms all services	All	POSIX + RT	N/A	None	All
PREEMPT	100s of us Schd, Int	preempt or irq disable	POSIX + RT	N/A	None	All
Nested OS (co-kernel)	~10us RTOS svcs	RTOS, hw irq disable	RTOS (can be POSIX RT)	Dual env.	Good	All
Dual-OS/Dual-Core (ASMP)	<1us RTOS svcs	RTOS	RTOS (can be POSIX RT)	Dual env.	Excellent	Specialized
PREEMPT_RT	10s of us Schd, Int	preempt and irq disable (most ints in process ctx), (mostly drivers)	POSIX + RT	"Modest" patch (careful tuning)	None	All (except some drivers)
Migration between OSes	? us RTOS svcs	RTOS, hw irq disable	RTOS (can be POSIX RT)	Dual env. (easy mix)	OK	All
Migration within OS	? us RTOS svcs	Sched, RTOS svcs	POSIX + RT	Small patch	None	All?

(additions in blue)

Full story at http://lwn.net/Articles/143323

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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/

Organizations

- http://www.realtimelinuxfoundation.org/
  Community portal for real-time Linux.
  Organizes a yearly workshop.
- http://www.osadl.org
  Open Source Automation Development Lab (OSADL)
  Created as an equivalent of OSDL for machine and plant control systems. Member companies are German so far (Thomas Gleixner is on board). One of their goals is to supports the development of RT preempt patches in the mainline Linux kernel (HOWTOs, live CD, patches).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Hotplugging with udev

Michael Opdenacker
Free Electrons

© Copyright 2004-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/udev
Corrections, suggestions, contributions and translations are welcome!

/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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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: `udevd` version 108: 61 KB in Ubuntu 7.04

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Starting udev (2)

- The `udevd` daemon is started.
  It listens to *uevents* from the driver core,
  which are sent whenever devices are inserted or removed.
- The `udevd` 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,
  `udevd` receives an *inotify* event and updates its
  ruleset in memory.
- When an event is received, `udevd` 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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"
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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"
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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 `udevd`, 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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}"
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Coldplugging

- Issue: loosing all device events happening during kernel initialization, because udev is not ready yet.
- Solution: after starting `udevd`, 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.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Debugging events - udevmonitor (1)

`udevadm monitor` 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).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Debugging events - udevmonitor (2)

`udevadm monitor --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
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Misc udev utilities

- `udevinfo`
  Lets users query the udev database.
- `udevtest <sysfs_device_path>`
  Simulates a `udev` run to test the configured rules.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Firmware hotplugging implementation

See `Documentation/firmware_class/` for a nice overview

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

udev files

- `/etc/udev/udev.conf`
  udev configuration file.
  Mainly used to configure syslog reporting priorities.
  Example setting: `udev_log="err"`
- `/lib/udev/rules.d/`
  Standard udev event matching rules, installed by the distribution.
- `/etc/udev/rules.d/*.rules`
  Local (custom) udev event matching rules. Best to modify these.
- `/lib/udev/devices/*`
  static `/dev` content (such as `/dev/console`, `/dev/null`...).
- `/lib/udev/*`
  helper programs called from udev rules.
- `/dev/*`
  Created device files.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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/
- The udev manual page:
  `man udev`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

mdev, the udev for embedded systems

- udev might be too heavy-weight for some embedded systems, the `udevd` 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`
- Need to mount `/dev` as a tmpfs:
  `mount -t tmpfs mdev /dev`
- Tell mdev to create the `/dev` entries corresponding to the devices detected during boot when mdev was not running:
  `mdev -s`
- The behavior 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`

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Reducing embedded Linux boot time

Michael Opdenacker
Free Electrons

© Copyright 2007-2011, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Nov 16, 2011,
Document sources:
http://free-electrons.com/blog/boot-time-genivi-2011/
Corrections, suggestions, contributions and translations are welcome!

Why trying to reduce boot time?

To achieve better user perception

Traditional solutions

Expose the user to relativistic acceleration

$$\tau = t\sqrt{1-(v^2/c^2)}$$

- Major drawback: the user gets to far from the device to see it boot faster.

Time travel

- Drawback: the user gets 2 devices in his hands for a certain amount of time.

Distract the user

Make the boot process faster

Alternative solutions

- Suspend to RAM and resume
  Used in Android phones. Not acceptable for devices which can stay idle for a long time (e.g. digital cameras)
- Hibernate to flash and resume
  Used in SONY digital cameras (booting in about 1s). Used in credit card payment terminals (our customer).
  Requires sufficient amount of RAM.

We will focus on reducing “cold” boot time, from power on to the execution of the system application.

The techniques shown should be applicable to all kinds of systems and distributions.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Our test system

Hardware

- TNY-A9260 board from CALAO Systems
- AT91SAM9260 CPU at 180 MHz
- RAM: 64 MB, NAND flash: 256 MB
- Serial port
- USB device port (used for networking)

Software

- Simple system built with BusyBox
- Mounting a JFFS2 partition with JPG photos on it (204 MB)
- Starting a BusyBox web server to view and upload photos
- Initial boot time: 37.75 s

Methodology

Boot time components

Bootstrap and bootloader time

Measure time - Grabserial

- From Tim Bird
  http://elinux.org/Grabserial
- A simple script to add timestamps to messages coming from a serial console.
- Key advantage: starts counting very early (bootstrap and bootloader)
- Another advantage: no overhead on the target, because run on the host machine.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Bootloader - Unnecessary functionality

U-Boot is full of features for development and debugging

- U-Boot boot delay
- Recompile U-boot without Ethernet, USB, filesystems...
- You can even disable the console.

You could even switch to a simpler boot loader (more expensive)

- Qi bootloader from OpenMoko
  http://gitorious.org/+0xlab/0xlab-bootloader/qi-bootloader
- BareBox supports an increasing number of boards
  http://barebox.org

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Optimize functionality

Accelerate kernel copy to RAM

- Rebuild the kernel with `CONFIG_CC_OPTIMIZE_FOR_SIZE`
- Rebuild the kernel without unneeded drivers and features:
  A smaller kernel is faster to copy. See our later slide about kernel size reduction.
- Depending on flash read throughput and CPU performance, choose between various kernel compression schemes:
  - Gzip
  - Bzip2
  - LZMA
  - XZ (improved LZMA compression for executable code)
  - No compression

LZO kernel decompression

- LZO is a compression algorithm that is much faster than Gzip and other compressors, at the cost of a slightly degraded compression ratio (+10%)
- Albin Tonnerre from Free Electrons added support for LZO compressed kernels. See http://lwn.net/Articles/350985/

Kernel decompression benchmarks

	Gzip	LZO	Uncompressed
Kernel size	1.33Mb	1.45Mb	2.45Mb
Bootloader + kernel load time	0.30s	0.33s	0.60s
Early kernel init time	0.52s	0.33s	0.02s
Total time	0.82s	**0.66s**	**0.62s**

- Bzip2, LZMA and XZ are not tested here but are slower than Gzip and LZO
- See http://free-electrons.com/blog/lzo-kernel-compression/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Skip the bootloader (1)

In our AT91 case

- Idea: make a slight change to the bootstrap code to directly load and execute the Linux kernel image instead of the U-boot one.
- Rather straightforward when boot U-boot and the kernel are loaded from NAND flash.
- Requires to hardcode the kernel command line in the kernel image (`CONFIG_CMDLINE`)
- Requires more development work when U-boot is loaded from a different type of storage
- Time savings: about 2 s

http://free-electrons.com/blog/at91bootstrap-linux/

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Skip the bootloader (2)

TI OMAP case

Saves 3-5 seconds of boot time
See http://j.mp/hgFeVz (http://0xlab.org)

Kernel boot time

Measure time: 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, run `dmesg > boot.log` and on your workstation, run `cat boot.log | perl scripts/bootgraph.pl > boot.svg` to generate a graphical representation

Kernel - Remove unnecessary functionality

- Make sure you have no unused kernel drivers
  If devices are not in the critical boot path, load their drivers later.
- Also remove unused networking protocols, filesystems, debugging features...
- If you have a dedicated system, you can even disable standard kernel features with `CONFIG_EXPERT`
  (previously `CONFIG_EMBEDDED`)
- If possible, disable support for loadable kernel modules and make all your drivers static (smaller kernel once more)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Removing functionality: 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`
- You can still see the messages through the `dmesg` command.

See http://elinux.org/Disable_Console

Optimizing - Disable IP auto config

- Stopped initializing the IP address on the kernel command line (old remains from NFS booting, was convenient not to hardcode the IP address in the root filesystem.)
- Instead, did it in the `/etc/init.d/rcS` script.
- This saved 1.56 s!
- You will save even more if you had other related options in your kernel (DHCP, BOOP, RARP)

IP: kernel level autoconfiguration	IP_PNP
IP: DHCP support	IP_PNP_DHCP
IP: BOOTP support	IP_PNP_BOOTP
IP: RARP support	IP_PNP_RARP

Optimizing - Reducing the number of PTYs

- PTYs are needed for remote terminals (through SSH)
  They are not needed in our dedicated system!
- The number of PTYs can be reduced through the `CONFIG_LEGACY_PTY_COUNT` kernel parameter.
  If this number is set to 4, we save 0.63 s.
- As we're not using PTYs at all in our production system, we disabled them with completely with `CONFIG_LEGACY_PTYS`.
  We saved 0.64 s.
- Note that this can also be achieved without recompiling the kernel, using the `pty.legacy_count` kernel parameter.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Optimizing - Skip probes

- The kernel and drivers probe the hardware at each boot. But the hardware is always the same!
- Example: delay loop calibration:
  - At each boot, the Linux kernel calibrates a delay loop (for the `udelay` function). This measures a `loops_per_jiffy` (`lpj`) value.
  - You just need to measure this once! Find the `lpj` value in kernel boot messages. Example:

```
Calibrating delay loop... 99.73 BogoMIPS (lpj=498688)
```

  - At the next boots, start Linux with the below option: `lpj=<value>`
  - It saved us 0.18 s
- Check kernel drivers for probing parameters

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Optimizing filesystem performance

- The time to mount the root filesystem is a major component of kernel boot time.
- JFFS2 example:
  `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.

  Switching this on saved 27.86 s!
- Very cheap to switch to other filesystems.
  See our benchmarks and presentation on
  http://elinux.org/Flash_Filesystem_Benchmarks

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Initramfs: the silver bullet

- Linux can boot on an intermediate root filesystem in RAM, called the initramfs. Its contents can be included in the kernel image.
- Using the file cache, this filesystem doesn't need any drivers to work (no filesystem driver, no disk driver). Hence, it can be accessed very early in the kernel boot process.
- This allows to be in userspace within a few hundred milliseconds after power on.
- System builders use it to show very early signs of life, such as a splashscreen. Even if the boot process isn't complete yet, this definitely helps with user perception.
- Use LZO Initramfs compression to save time: (`INITRAMFS_COMPRESSION_LZO`)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Other kernel optimizations

- NAND: just check for bad blocks once
  Atmel: see http://patchwork.ozlabs.org/patch/27652/
- Fast boot, asynchronous initcalls in drivers
  http://lwn.net/Articles/314808/
  Mainlined, but API still used by very few drivers.
  Mostly useful when your CPU has idle time in the boot process.
- Use deferred initcalls
  See http://elinux.org/Deferred_Initcalls

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Userspace boot time

User space: measure time

- If you are using Grabserial, you can still send message to the kernel console from your applications, by writing to `/dev/kmsg`.
- Use utilities to track processes run in the boot sequence: http://elinux.org/Bootchart

Measure time: strace

System call tracer
http://sourceforge.net/projects/strace/

- Mainly useful for your main application
- Can be built by your cross-compiling toolchain generator (crosstool-ng for example)
- Allows to see what any of your processes is doing: accessing files, allocating memory...
  Very useful to detect waste of time.
- Usage:
  `strace <command>` (starting a new process)
  `strace -p<pid>` (tracing an existing process)

See `man strace` for details.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

strace example output

```
> strace cat Makefile
execve("/bin/cat", ["cat", "Makefile"], [/* 38 vars */]) = 0
brk(0)                                  = 0x98b4000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7f85000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=111585, ...}) = 0
mmap2(NULL, 111585, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7f69000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/tls/i686/cmov/libc.so.6", O_RDONLY) = 3
read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320h\1\0004\0\0\0\344"..., 512)
= 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442180, ...}) = 0
mmap2(NULL, 1451632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb7e06000
mprotect(0xb7f62000, 4096, PROT_NONE)   = 0
mmap2(0xb7f63000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x15c) = 0xb7f63000
mmap2(0xb7f66000, 9840, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0xb7f66000
close(3)                                = 0
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

measure time: 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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

ltrace example output

```
ltrace 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
...
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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
```


Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Measure time: use a profiler

Profile your application to detect unnecessary activity or performance issues.

- For example, a profiler can tell you in which functions most of the time is spent.
- Valgrind (http://valgrind.org/) is the most popular profiler
  - Now available for the `arm` architecture, thanks to Linaro (armv7 only: Cortex A8, A9 and A5)
  - Complete suite of profiling tools, in particular:
    Cachegrind: sources of cache misses and function statistics.
    Massif: sources of memory allocation.

Optimize and remove services

If you are using a distribution or an automatically generated root filesystem

- Remove services you don't need (ssh...), or start them later.
- Start your services directly from a single startup script.
  This eliminates multiple calls to `/bin/sh`.
- Remove udev (or mdev) if you just need them for device files.
  Use devtmpfs (`CONFIG_DEVTMPFS`) instead,
  automatically managed by the kernel, and cheaper.
- This saves tens of seconds with root filesystems generated with OpenEmbedded (for example).

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Shells: reducing forking

- `fork` / `exec` system calls are very expensive.
  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

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Optimizing: executables and libraries (1)

- Use statically linked applications
  (less CPU overhead, less libraries to load). At least true for small root filesystems.
- Use a lighter C library reduced to the minimum
  (uClibc or eglibc). Can save up to 1 MB.

C program	*Compiled statically*	
	glibc	*uClibc*
Plain “hello world”	475 K	25 K
Busybox	843 K	311 K

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Optimizing: executables and libraries (2)

- Strip your executables and libraries, removing ELF sections only needed for development and debugging.
  `strip` command provided by your toolchain:
  `arm-linux-strip potato`
- `superstrip`:
  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.

	Hello World	*Busybox*	*Inkscape*
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 %)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

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!
- Linaro has optimized C library functions (memset, memcpy...) for recent arm cores, using NEON instructions. See https://launchpad.net/cortex-strings.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Linker optimizations

Group application code used at startup

- Find the functions called during startup
- Create a custom linker script to put them all together in the same section, using the `-ffunction-sections` gcc option.
- Particularly useful for flash storage with rather big MTD read blocks. As the whole read blocks are read, you end up reading unnecessary data.

See http://j.mp/m4d1Q6

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Results

- This test case:
  Initial boot time: 38 s
  Final boot time: approximately 4 s
- Customer audit 1: AMD Geode board with X graphics
  From 32 to 10 seconds in only 3 days
- Customer audit 2: AT91SAM9263 based system
  From 32 to 7 seconds in only 2 days
- Lots of techniques not applied yet.
  I just wanted to show how much you can achieve with limited time and effort.
  With more effort, it is possible to boot any system within 1-5 seconds after power-on.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Conclusion

These optimizations are cheap!

- Find the low hanging fruit
  and divide your boot-time by 2 or 3 in a few days.
- None of them require any re-redesign
  Another team can take care of them, and this can be done very late in product development.
- With the exception of bootstrap tricks, and application fixes, you won't have any extra development to do. You may just have to recompile your bootloader, kernel and root filesystem.
- Quick learning curve: very easy to reuse the same techniques in future products.

Don't tell your boss that just a few days were enough! And with the extra spare time, contribute to community projects ;-)

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Evaluation form

Please take a few minutes to rate this training session,
by answering our on-line survey:

http://j.mp/xxxxxx

Archive your lab directory

- Clean up files that are easy to retrieve, remove downloads.
- Generate an archive of your lab directory.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Related documents

All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development

Life after training

Here are things we could do to support you in your embedded Linux and kernel projects:

- BSP development for your hardware (drivers, bootloader, toolchain)
- Make the official Linux sources support your hardware
- System development and integration
- System optimization
- Hunting and fixing nasty bugs
- More training: see http://free-electrons.com/training/. Your colleagues who missed this class could go to our public sessions.

See http://free-electrons.com/development
and http://free-electrons.com/services

Thank you!

And may the Source be with you

Archive your lab directory

- Clean up files that are easy to retrieve, remove downloads.
- Generate an archive of your lab directory.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. **http://free-electrons.com**

Related documents

All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development

Life after training

Here are things we could do to support you in your embedded Linux and kernel projects:

- BSP development for your hardware (drivers, bootloader, toolchain)
- Make the official Linux sources support your hardware
- System development and integration
- System optimization
- Hunting and fixing nasty bugs
- More training: see http://free-electrons.com/training/. Your colleagues who missed this class could go to our public sessions.

See http://free-electrons.com/development
and http://free-electrons.com/services

Thank you!

And may the Source be with you