Embedded Linux driver development

Embedded Linux kernel and driver development

Michael Opdenacker
Free Electrons
http://free-electrons.com/

Thanks

- To Jonathan Corbet, for his very useful news and articles on http://lwn.net/, in particular for porting drivers to 2.6.
- To the OpenOffice.org project, for their presentation and word processor tools which satisfied all my needs.
- To the Handhelds.org community, for giving me so much help and so many opportunities to help.
- To the members of the whole Free Software and Open Source community, for sharing the best of themselves: their work, their knowledge, their friendship.
- To people who sent corrections:
  Matti Aaltonen

Copying this document

Document updates available on http://free-electrons.com/training/drivers
Corrections, suggestions and contributions are welcome!

Document history

Unless specified, contributions are from Michael Opdenacker
See http://free-electrons.com/doc/ChangeLog for detailed changes.

- Oct 1, 2004. Added jffs2 image mounting instructions.
- Sep 28, 2004. First public release
- Sep 20-24, 2004. First session for Atmel, Rousset (France)

About this document

- This document is first of all meant to be used as a visual aid by a speaker or a trainer. Hence, this is just a summary or a complement to what is said. Hence, the explanations are not supposed to be exhaustive.
- However, this document is also meant to become a reference for the audience. It also targets readers interested in self-training. So, a bit more details are given, making the document a bit less visually attractive.

Training contents (1)

Introduction

- System overview and role of the kernel
- History and versioning scheme
- Supported hardware architectures
- Legal issues: licensing constraints, software patents
- Kernel user interface

Training contents (2)

Compiling and booting

- Getting the sources
- Using the patch command
- Structure of source files
- Kernel modules
- Kernel configuration
- Compiling
- Cross-compiling
- The bootloader
- Booting parameters
- Debugging through the serial port
- Creation of an initrd ramdisk

Training contents (3)

Driver development

- Linux device drivers
- A simple module
- Programming constraints
- Loading, unloading modules
- Module parameters
- Module dependencies
- Adding sources to the kernel tree
- Kernel debugging

Contents (4)

Advice and resources

- Using Ethernet over USB
- Root filesystem on the host through NFS
- Review of the various filesystem types. The MTD subsystem. Advice for making a choice
- Getting help and contributions
- Bug report and patch submission to Linux developers.
- References

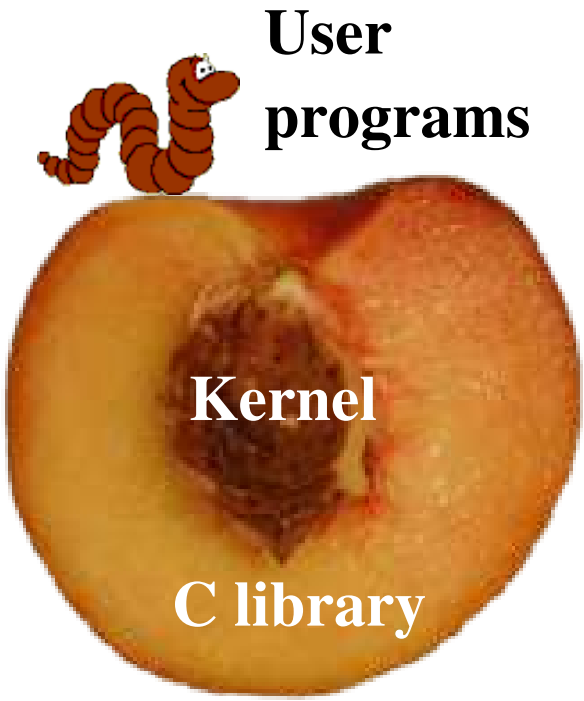
Studied kernel version: 2.6

- Linux 2.4
  - Mature and quite exhaustive
  - But developments stopped; fewer and fewer developers willing to help.
  - Will be definitely obsolete when your new product starts.
  - Still fine if you get your sources, tools and support from commercial Linux vendors
- Linux 2.6
  - Support from Linux hackers and community
  - Getting more and more mature and exhaustive
  - Cutting edge features but some drivers not upgraded yet.

Introduction

Role of the kernel

- Linux is the kernel.
  It provides an interface to the hardware
- GNU / Linux is the whole operating system
- Hurd, Mach, BSD are other kernels
- GNU / Hurd, MacOS, FreeBSD are other operating systems



Linux history

- 1991: Linux kernel written from scratch in 6 months by Linus Torvalds in his Helsinki University room, to overcome limitations of his 80386 PC.
- 1991: Linus shares his kernel on the net. Programmers from the whole world join in and contribute to coding and testing
- 1992: Linux released under the GNU General Public License
- 1994: Linux 1.0 released
- 1994: Red Hat founded by Bob Young and Marc Ewing, creating a new business model.
- 1995-: GNU/Linux and free software developing in Internet servers.
- 2001: IBM invests $1 billion in Linux
- 2002-: GNU/Linux wide adoption starts in many industry sectors.

Linux versioning scheme

Releases are versioned as x.y.z

- Stable versions
  - x.y: main release number
  - y: even number
  - z: identifies the exact release version number (use
  - Examples: 2.0.40, 2.2.26, 2.4.27, 2.6.7 ...
- Development versions
  - y: odd number
  - Examples: 2.3.42, 2.5.74 ...

Supported hardware architectures

- See the `arch/` directory
- Minimum: 32 bit processors, with or without MMU
- 32 bit architectures:
  alpha, arm, cris, h8300, i386, m68k, m68knommu, mips, parisc, ppc, s390, sh, sparc, um, v850
- 64 bit architectures:
  ia64, mips64, ppc64 sh64, sparc64, x86_64
- See `arch/README` or `Documentation/arch/README` for details

Linux key features

- Portability and hardware support
- Scalability
  Can run on super computers as well as on tiny devices
- Compliance to standards and interoperability
- Networking
- Security
- Stability and reliability
- Modularity

Introduction

Legal issues

Introduction

Legal issues

Licensing details and constraints

About Free Software

- Linux is *Free Software*
- *Free Software* grants the below 4 freedoms to the user:
  - The freedom to run the program, for any purpose
  - The freedom to study how the program works, and adapt it to one's needs
  - The freedom to redistribute copies to help others
  - The freedom to improve the program, and release one's improvements to the public
- See http://www.gnu.org/philosophy/free-sw.html

The GNU General Public License (GPL)

- *Copyleft* licenses use copyright laws to make sure that modified versions are free software too
- The GNU GPL requires that modifications and derived works are GPL too:
  - Only applies to **released** software
  - Any program using GPLed code (either by static or even dynamic linking) is considered as an extension of this code
- More details:
  - Copyleft: http://www.gnu.org/copyleft/copyleft.html
  - GPL FAQ: http://www.gnu.org/licenses/gpl-faq.html

Linux kernel licensing constraints

- No constraints before you release.
  You should share your changes early for your own interest
- Constraints at release time:
  - For any device embedding Linux and Free Software, you have to release sources to the end user. You have no obligation to release them to anybody else!
  - Proprietary modules are tolerated (but not recommended) as long as they cannot be considered as derived work of GPLed code.
  - Proprietary drivers can't be statically compiled in the kernel.
  - No issue with drivers available under a GPL compatible license (see `include/linux/modules.h`)

Advantages of free software drivers

From the driver developer / decision maker point of view

- You don't have to write your driver from scratch. You can reuse code from similar free software drivers.
- You get free community contributions, support, code review and testing. Proprietary drivers (even with sources) don't get any.
- Your drivers can be freely shipped by others (mainly by distributions)
- Your drivers can be statically compiled in the kernel
- Users and the community get a positive image of your company. Makes it easier to hire talented developers.
- You don't have to supply binary driver releases for each kernel version and patch version (closed source drivers)
- Modules have all privileges: some users need to review sources.

Introduction

Legal issues

Software patents

Software patents: the big legal threat

- Software implementations very well protected internationally by Copyright Law. This is automatic, no paperwork.
- However, in countries like the USA or Japan, it is now legal to patent what the software does, instead of protecting only the implementation.
- Patents can be used to prevent anyone from re-using or even improving an algorithm or an idea!
- Deadly for software competition and innovation: can't write any program without reusing any technique or idea from anyone.

Software patents hall of shame

- The progression bar
- Amazon 1-click, Amazon gift ordering
- Electronic shopping cart
- Compressing and decompressing text files
- Compression in mobile communication
- Digital signature with extra info
- Hypermedia linking

See http://swpat.ffii.org/patents/samples/index.en.html for more examples

How to avoid patent issues

- Applies too when you develop in software patent free areas. You may not be able to export your products.
- Kernel drivers with patents: always check driver description in kernel configuration. Known patent issues are always documented.
- Always prefer patent free alternatives (PNG instead of JPEG, Linux RTAI instead of RTLinux, etc.)
- Don't file patents on your technologies at your turn. This may expose you more to patent risk. You will lose against software giants.

How to deal with patent issues

When patent lawyers are after you, you may get help from:

- In the USA
  - The Electronic Frontier Foundation
    http://eff.org/
- In the European Union
  - The Foundation for a Free Information Infrastructure
    http://ffii.org/index.en.html
- In other areas
  - *Note to readers: any references are welcome!*

Introduction
Kernel user interface

Kernel userspace interface

A few examples:

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

... and much more! See by yourself!

Compiling and booting Linux

Compiling and booting Linux

Getting the sources

Access to kernel sources

- Download sources from http://kernel.org/:

```
wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.7.tar.bz2
wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.7.tar.bz2.sign
```

- Or get a patch vs the x.y.<z-1> version:

```
wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/patch-2.6.7.bz2
wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/patch-2.6.7.bz2.sign
```

- Check the integrity of sources:

```
gpg --verify linux-2.6.7.tar.bz2.sign linux-2.6.7.tar.bz2
```

  - GnuPG details: http://www.gnupg.org/gph/en/manual.html
  - Kernel source signature details: http://www.kernel.org/signature.html

Using the patch command

- `patch` command: uses the output of the `diff` command to apply a set of changes to a source tree.
- patch basic usage:

`patch -p`*`n`* `<` *`diff_file`*

  - `n`: number of directory levels to skip (example next page)
- Linux patches:
  - Always to apply to the `x.y.<z-1>` version
  - Always produced for `n=1`

```
patch -p1 < linux_patch
```

patch usage example

- Patch file (hardware.diff)

```
--- linux-2.6.8.1/include/asm-arm/hardware.h    2004-08-14 12:54:48.000000000 +0200
+++ linux-2.6.8.1_modified/include/asm-arm/hardware.h   2004-08-17 12:42:06.119650556
+0200
@@ -15,13 +15,4 @@
#include <asm/arch/hardware.h>
-#ifndef __ASSEMBLY__
-
-struct platform_device;
-
-extern int platform_add_devices(struct platform_device **, int);
-extern int platform_add_device(struct platform_device *);
-
-#endif
-
#endif
```

- Commands

```
cd linux-2.6.8.1
patch -p1 < hardware.diff
```

- Applies changes to include/asm-arm/hardware.h

Useful kernel source links

Difficult to find if you don't know them!

- Direct view access to the Linux source repository, useful to create patches against the latest versions:
  http://linux.bkbits.net:8080/linux-2.6/src
- Linux daily source snapshots:
  http://www.kernel.org/pub/linux/kernel/v2.6/snapshots/old/
  http://www.kernel.org/pub/linux/kernel/v2.6/snapshots/

Compiling and booting Linux

Structure of source files

Linux source structure (1)

`arch/`	Architecture dependent code
`COPYING`	Linux copying conditions (GNU GPL)
`CREDITS`	Linux main contributors
`crypto/`	Cryptographic libraries
`Documentation/`	Kernel documentation. Don't miss it!
`drivers/`	All device drivers (`drivers/usb/`, etc.)
`fs/`	Filesystems (`fs/ext3/`, etc.)
`include/`	Kernel headers
`include/asm-<arch>`	Architecture dependent headers
`include/linux`	Linux kernel core headers
`init/`	Linux initialization (including `main.c`)
`ipc/`	Code used for process communication

Linux source structure (2)

`kernel/`	Linux kernel core (very small!)
`lib/`	Misc library routines (zlib, crc32...)
`MAINTAINERS`	Maintainers of each kernel part. Very useful!
`Makefile`	Top Linux makefile (sets arch and version)
`mm/`	Memory management code (small too!)
`net/`	Network support code (not drivers)
`README`	Overview and building instructions
`REPORTING-BUGS`	Bug report instructions
`scripts/`	Scripts for internal or external use
`security/`	Security model implementations (selinux...)
`sound/`	Sound support code and drivers
`usr/`	Utilities: gen_init_cpio and initramfs_data.S

Compiling and booting Linux
Kernel modules

Loadable kernel modules (1)

- Modules: add a given functionality to the kernel (drivers, filesystem support, and many others)
- Can be loaded and unloaded at any time, only when their functionality is need. Once loaded, have full access to the whole kernel. No particular protection.
- Useful to keep the kernel image size to the minimum (essential in GNU/Linux distributions for PCs).

Loadable kernel modules (2)

- Useful to support incompatible drivers (either load one or the other, but not both)
- Useful to deliver binary-only drivers (bad idea) without having to rebuild the kernel.
- Modules make it easy to develop drivers without rebooting: load, test, unload, rebuild, load...
- Modules can also be compiled statically into the kernel.

Compiling and booting Linux

Kernel configuration

Kernel configuration overview

- Makefile edition
  Setting the version and target architecture if needed
- Kernel configuration: defining what features to include in the kernel:
  `make xconfig`
  or `make menuconfig`
  or `make oldconfig`
  or editing by hand
  - Kernel configuration file (Makefile syntax) stored in the `.config` file at the root of the kernel sources
  - Distribution kernel config files usually released in `/boot/`

Makefile changes

- To identify your kernel image with others build from the same sources, use the `EXTRAVERSION` variable:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 7
EXTRAVERSION = -acme1
```

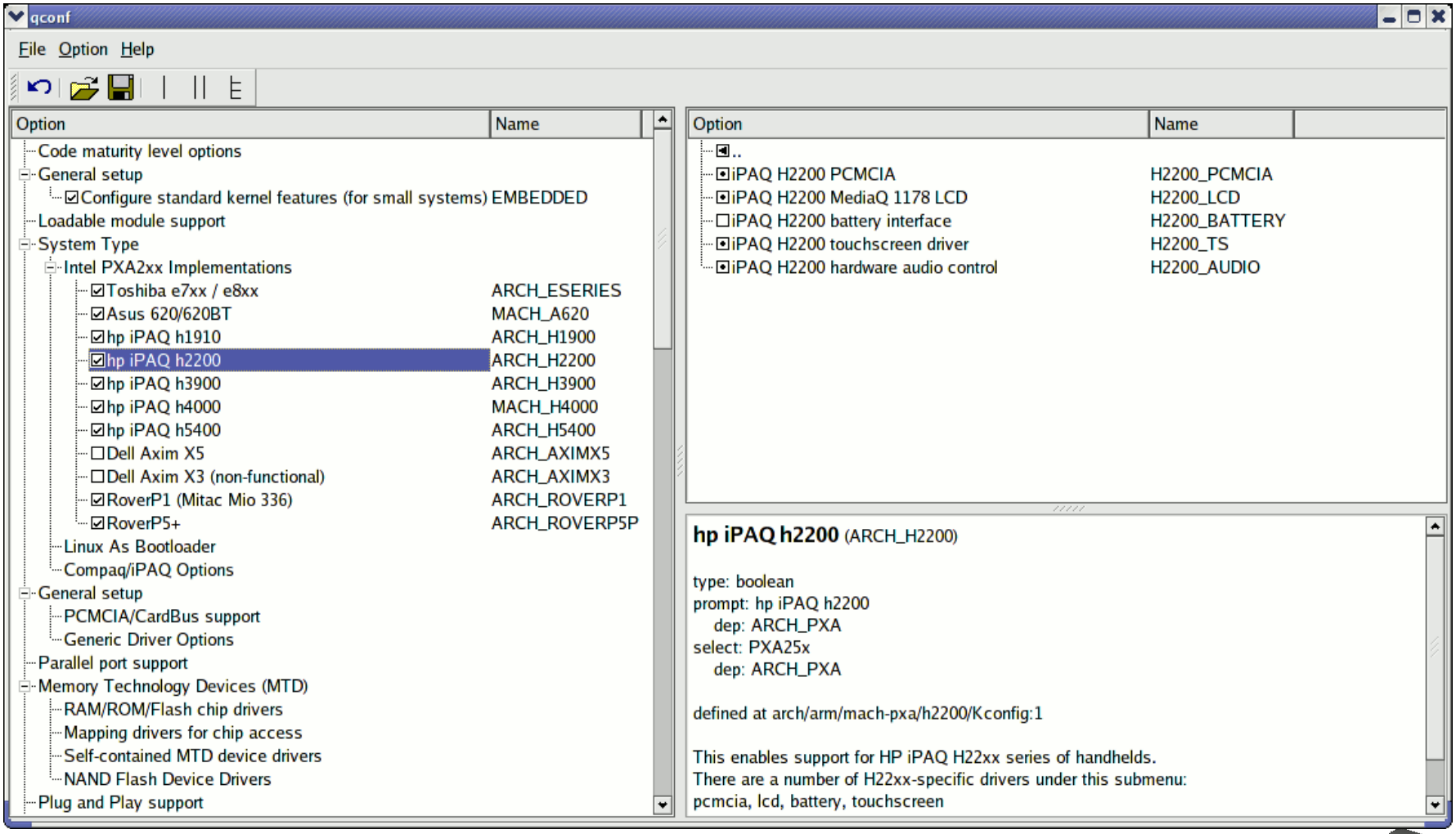
  - `uname -r` will return: `2.6.7-acme1`

make xconfig

`make xconfig`

- qconf: new qt configuration interface for Linux 2.6. Much easier to use!
- Make sure you read help -> introduction: useful options!
- File browser: easier to load configuration files

qconf screenshot



make menuconfig / oldconfig

`make menuconfig`

- Same old text interface. Rarely useful.
  You can just simply edit the .config file by hand! Beware of dependencies though.

`make oldconfig`

- Useful to upgrade a config file from an earlier kernel release
- Issues warnings for obsolete symbols
- Asks for values for new symbols

Compiling and booting Linux

Compiling the kernel

Compiling and installing the kernel

Compiling steps:

- `make`

Install steps (logged as root!)

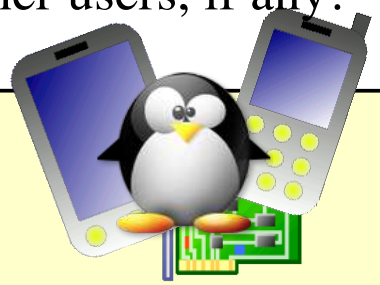
- `make install`
- `make modules_install`

The following commands are no longer needed:

- `make depends`
- `make modules` (done by `make`)

Compiling faster

- Spend a bit more time in kernel configuration and just compile the modules needed on your hardware. This can divide compile time by 30 and save hundreds of MB!
- Compile several files in parallel:
  `make -j <number>`
  `make` runs several targets in parallel, whenever possible
  - `make -j 4`
    Much faster even on uniprocessor workstations! Less time wasted in reading or writing files (the other jobs keep the CPU busy)
  - Not really useful going further than 4. More context switching may even slow down the jobs.
  - `make -j <4*number_of_processors>`
    On a multiprocessor machine. Beware of not disturbing other users, if any!



Kernel compiling tips

- View the full (gcc, ld) command line:

```
make V=1
```

- Remove all generated files (to create patches...):

```
make mrproper
```

Generated files

- `vmlinux`
  Raw Linux kernel image, non compressed
- `arch/<arch>/boot/zImage`
  zlib compressed kernel image
  Default image on arm
- `arch/<arch>/boot/bzImage`
  bzip2 compressed kernel image. Usually small enough to fit on a floppy disk!
  Default image on i386

Installed files (1)

- `/boot/vmlinuz-<version>`
  Kernel image
- `/boot/System.map-<version>`
  Stores kernel symbol addresses
- `/boot/initrd-<version>.img`
  Initial RAM disk, storing the modules you need to mount your root filesystem. `make install` runs `mkinitrd` for you!
- `/etc/grub.conf` or `/etc/lilo.conf`
  make install updates your bootloader configuration files to support your new kernel! It reruns `/sbin/lilo` if LILO is your bootloader.

Installed files (2)

- `/lib/modules/<version>/`
  Kernel modules + extras
  - `build/`
    Everything needed to build more modules for this kernel: . config file (`build/.config`), module symbol information (`build/module.symVers`), kernel headers (`build/include/`)
  - `kernel/`
    Module `.ko` (Kernel Object) files, in the same directory structure as in the sources.

Installed files (3)

- `/lib/modules/<version>/` (continued)
  - `modules.alias`
    Module aliases for insmod and modprobe. Example line:
    `alias sound-service-?-0 snd_mixer_oss`
  - `modules.dep`
    Module dependencies for insmod and modprobe. Also useful to copy only the required modules to a minimum filesystem.
  - `modules.symbols`
    Tells which module a given symbol belongs to.

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

Compiling the kernel in a nutshell

- Edit version information in the `Makefile` file
- `make xconfig`
- `make`
- `make install`
- `make modules_install`

Compiling and booting Linux

Cross-compiling the kernel

Makefile changes

Makefile changes

- Update the version as usual
- You should change the default target platform, e.g.:

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

- or run (ARM example):
  `make ARCH=arm CROSS_COMPILE=arm-linux-`
  (Useful when you compile for several platforms)

See comments in `Makefile` for details

Configuring the kernel

- Same as native compilation
- Don't forget to set the right architecture
- Useful way of sharing your configuration file:
  `cp .config arch/<arch>/config/acme_defconfig`

  To get your standard configuration file, the other people working on the ACME embedded system and using your kernel will just have to run:
  `make acme_defconfig`

Cross-compiling setup

Example

- You have an ARM cross-compiling toolchain in `/usr/local/arm/3.3.2/`
- You just have to add it to your Unix PATH:
  `export PATH=/usr/local/arm/3.3.2/bin:$PATH`

See the `Documentation/Changes` file in the sources for details about minimum tool versions requirements .

Building the kernel

- Run
  `make` (if you have modified your Makefile)
  or otherwise (ARM example)
  `make ARCH=arm CROSS_COMPILE=arm-linux-`
- Copy
  `arch/<platform>/boot/zImage`
  to the target storage
- `make modules_install`
  and copy `/lib/modules/<version>` to the target storage
- You can customize `arch/<arch>/boot/install.sh` so that `make install` does this automatically for you.

Compiling and booting Linux

The bootloader

The bootloader's job

One main mission: load the operating system(s)

Tasks

- Initialize the machine properly (the kernel can do part of this later too).
- Access the kernel and initrd files in their storage medium (need to support the corresponding filesystem too)
- Because of the above 2 tasks, bootloaders are often platform specific!
- Load the kernel and initrd files
- Execute the kernel file with the right command line

2-stage bootloaders

- At startup, the hardware automatically executes the bootloader from a given location, usually with very little space (such as the boot sector on a PC hard disk)
- Because of this lack of space, 2 stages are implemented:
  - 1st stage: minimum functionality. Just accesses the second stage on a bigger location and executes it.
  - 2nd stage: offers the full bootloader functionality. No limit in what can be implemented. Can even be an operating system itself!

A few bootloaders (1)

- LILO: LInux LOad. Original Linux bootloader. Still in use!
  http://freshmeat.net/projects/lilo/
  Supports: x86
- GRUB: GRand Unified Bootloader from GNU. More powerful.
  http://www.gnu.org/software/grub/
  Supports: x86
- LinuxBIOS: Linux based BIOS replacement
  http://www.linuxbios.org/
  Supports: x86
- sh-boot: LinuxSH project bootloader
  http://cvs.sourceforge.net/viewcvs.py/linuxsh/sh-boot/
  Supports: sh

A few bootloaders (2)

- bootldr: Handhelds.org's bootloader for iPAQs
  ftp://ftp.handhelds.org/bootldr/
  Supports: arm
- LAB: Linux As Bootloader, from Handhelds.org
  Part of Handhelds.org's Linux kernel.
  See http://handhelds.org/moin/moin.cgi/Linux26ToolsAndSources
  Supports: arm (experimental)
- U-Boot: Universal Bootloader. The most used on arm.
  http://u-boot.sourceforge.net/
  Supports: arm, ppc, mips, x86
- RedBoot: eCos based bootloader from Red-Hat
  http://sources.redhat.com/redboot/
  Supports: x86, arm, ppc, mips, sh, m68k...

Kernel command line parameters

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

- Useful to configure the kernel at boot time, without having to recompile it.
- Example (used for the HP iPAQ h2200 PDA)

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

Most common command line parameters

- `root`
  Identifies the root filesystem
- `init`
  Script to run at the end of kernel initialization
  Default: `/sbin/init`
- `console`
  Console for booting messages
- `ro` / `rw`
  Mount root device as read-only / read-write

Hundreds of command line parameters described on
`Documentation/kernel-parameters.txt`

Compiling and booting Linux

Debugging through the serial port

Usefulness of a serial port

- Most processors feature a serial port interface (usually very well supported by Linux). Just need this interface to be connected to the outside.
- Easy way of getting the first messages of an early kernel version, even before it boots. A minimum kernel with only serial port support is enough.
- Once the kernel is fixed and has completed booting, possible to access a serial console and issue commands.
- The serial port can also be used to transfer files to the target.

When you don't have a serial port

On the host

- Not an issue. You can get a USB to serial converter. Usually very well supported on Linux and roughly costs $20. The device appears as `/dev/ttyUSB0` on the host.

On the target

- Check whether you have an IrDA port. It's usually a serial port too.
- If you have an Ethernet adapter, try with it
- You may also try to manually hook-up the processor serial interface (check the electrical specifications first!)

Compiling and booting Linux

Creation of an initrd ramdisk

Initrd

Initrd = Initial RAM disk

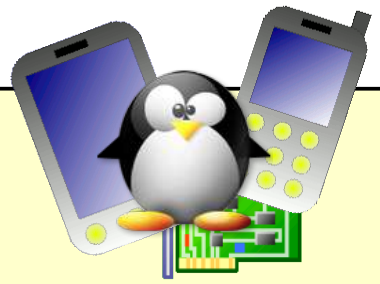
- Very first, minimalistic root (/) filesystem in RAM
- Traditionally used to minimize the number of device drivers built into the kernel.
  Example: contains an ext3 module to mount the final ext3 root filesystem.
- Also useful to run complex initialization scripts
- Useful to load proprietary modules (can't be statically compiled into the kernel)

How to create an initrd

```
mkdir /mnt/initrd
dd if=/dev/zero of=initrd.img bs=1k count=2048
mkfs.ext2 -F initrd.img
mount -o loop initrd.img /mnt/initrd
```

<Populate: busybox, modules, linuxrc script
More details in the Tools for Embedded Linux Systems training!>

```
umount /mnt/initrd
gzip --best -c initrd.img > initrd
```



Driver development

Driver development

Linux device drivers

Character drivers

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

```
crw-rw----  1 root uucp 4, 64 Feb 23  2004 /dev/ttyS0
crw--w----  1 jdoe tty 136, 1 Sep 13 06:51 /dev/pts/1
crw-------  1 root root 13, 32 Feb 23  2004 /dev/input/mouse0
crw-rw-rw-  1 root root 1, 3 Feb 23  2004 /dev/null
```

- Examples: keyboards, mice, parallel port, IrDA, Bluetooth port, consoles, terminals...

Block drivers

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

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

- Examples: hard or floppy disks, ram disks, loop devices...

Device major and minor numbers

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

- First number: major number
  Uniquely associated to each driver
- Second number: minor number
  Uniquely associated to each device

To find out which driver a device corresponds to, or when the device name is too cryptic, see `Documentation/devices.txt`

Device file creation

- Device files are not created when a driver is loaded.
- They have to be created in advance:
  `mknod /dev/<device> [c|b] <major> <minor>`
- Examples:
  `mknod /dev/ttyS0 c 4 64`
  `mknod /dev/hda1 b 3 1`

Other driver types

They don't have any corresponding `/dev` entry you could read or write through a regular Unix command.

- Network drivers
  They are represented by a network device such as `ppp0`, `eth1`, `usbnet`, `irda0` (listed by `ifconfig -a`)
- Other drivers
  Often, intermediate drivers just interfacing with other ones.

Driver development

A simple module

hello module

```
/* hello.c */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL");
```

Thanks to Jonathan Corbet for the example!

Module coding guidelines (1)

- C includes: you can't use standard C library functions (`printf()`, `strcat()`, etc.). The C library is implemented on top of the kernel, not the opposite.
- Linux provides some C functions for your convenience, like `printk()`, which interface is pretty similar to `printf()`.

So, only kernel header includes are allowed.

Module coding guidelines (2)

- Never use floating point numbers in kernel code. Your code may be run on a processor without a floating point unit (like on ARM). Floating point can be emulated by the kernel, but this is very slow.
- Define all symbols as static, except exported ones (avoid namespace pollution)
- See `Documentation/CodingStyle` for more guidelines
- It's also good to follow or at least read GNU coding standards: http://www.gnu.org/prep/standards.html

Compiling a module

- The below Makefile should be reusable for any Linux 2.6 module.
- Just run `make` to build the hello.ko file
- Caution: make sure there is a `[Tab]` character at the beginning of the `$(MAKE)` line (make syntax)

```
# Makefile for the hello module

obj-m := hello.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

Using the module

- Logged as root, run
  `tail -f /var/log/messages`
- Logged as root in another terminal, load the module:
  `insmod ./hello.ko`
- You will see the following in `/var/log/messages`:
  `Sep 13 22:02:30 localhost kernel: Hello, world`
- Now remove the module:
  `rmmod hello`
- You will see:
  `Sep 13 22:02:37 localhost kernel: Goodbye, cruel world`

Module utilities

- `insmod <module_name>`
  `insmod <module_path>.ko`
  Tries to load the given module, if needed by searching for its `.ko` file throughout the default locations (can be redefined by the `MODPATH` environment variable).
- `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.
- `rmmod <module_name>`
  Tries to remove the given module

Driver development

Defining and passing module parameters

hello module with parameters

```
/* hello_param.c */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>

MODULE_LICENSE("GPL");

/* A couple of parameters that can be passed in: how many times we say
   hello, and to whom */

static char *whom = "world";
module_param(whom, charp, 0);
static int howmany = 1;

module_param(howmany, int, 0);

static int hello_init(void)
{
    int i;
    for (i = 0; i < howmany; i++)
      printk(KERN_ALERT "(%d) Hello, %s\n", i, whom);
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel %s\n", whom);
}

module_init(hello_init);
module_exit(hello_exit);
```

Thanks to Jonathan Corbet for the example!

Using the hello_param module

- Load the module. For example:
  `insmod ./hello_param.ko howmany=2 whom=universe`
- You will see the following in `/var/log/messages`:
  ```
  Sep 13 23:04:30 localhost kernel: (0) Hello, universe
  Sep 13 23:04:30 localhost kernel: (1) Hello, universe
  ```
- Now remove the module:
  `rmmod hello_param`
- You will see:
  ```
  Sep 13 23:04:38 localhost kernel: Goodbye, cruel
  universe
  ```


Declaring module parameters

- `module_param(name, type, perm);`
  `name`: regular name symbol
  `type`: either `byte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `charp`, `bool` or `invbool` (checked at compile time!)
  `perm`: permissions for the corresponding entry in `/sys/module/<module_name>/<param>`. Safe to use `0`.
- `module_param_named(name, value, type, perm);`
  To make the `name` variable available outside the module and the `value` variable inside.
- `module_param_string(name, string, len, perm);`
  To defined name as `charp`, string prefilled with string of length `len`, usually `sizeof(string)`
- `module_param_array(name, type, num, perm);`
  To declare an array of parameters

Passing module parameters

- Through insmod or modprobe:

```
insmod ./hello_param.ko howmany=2 whom=universe
```

- Through modprobe
  after changing the `/etc/modprobe.conf` file:

```
options hello_param howmany=2 whom=universe
```

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

```
options hello_param.howmany=2 \
  hello_param.whom=universe
```


Driver development

Module dependencies

Module dependencies

- Module dependencies don't have to be described by the module writer.
- They are automatically computed during kernel building from module exported symbols. `module2` depends on `module1` if `module2` uses a symbol exported by `module1`.
- Module dependencies stored in `/lib/modules/<version>/modules.dep`
- You can update this file by running (as root) `depmod -a [<version>]`

Driver development

Adding sources to the kernel tree

New directory in kernel sources (1)

To add an `acme_drivers/` directory to the kernel sources:

- Move the `acme_drivers/` directory to the appropriate location in kernel sources
- Create an `acme_driver/Kconfig` directory
- Create an `acme_driver/Makefile` file based on the Kconfig variables
- In the parent directory Kconfig file, add `source "acme_driver/Kconfig"`
- Run `make xconfig` and see your new options!

New directory in kernel sources (2)

- In the parent directory Makefile file, add
  `"obj-$(CONFIG_ACME) += acme_driver/"` (just 1 condition)
  or
  `"obj-y += acme_driver/"` (several conditions)
- Run `make xconfig` and see your new options!
- Run `make` and your new files are compiled!
- See `Documentation/kbuild/*.txt` for details

Driver development
Kernel debugging

Debugging with printk

- Universal debugging technique used since the beginning of programming (first found in cavemen drawings)
- Printed or not in console or `/var/log/messages` according to the priority (give details and kernel config switches)
- Available priorities (`include/linux/kernel.h`):

```
#define KERN_EMERG      "<0>"   /* system is unusable */
#define KERN_ALERT      "<1>"   /* action must be taken immediately */
#define KERN_CRIT       "<2>"   /* critical conditions */
#define KERN_ERR        "<3>"   /* error conditions */
#define KERN_WARNING    "<4>"   /* warning conditions */
#define KERN_NOTICE     "<5>"   /* normal but significant condition */
#define KERN_INFO       "<6>"   /* informational */
#define KERN_DEBUG      "<7>"   /* debug-level messages */
```

ksymoops

- Can help decrypting oops messages, by converting addresses and code to useful text
- Easy to use: just copy/paste the oops text to a file
- Command line example:

```
ksymoops --no-ksyms -m System.map -v vmlinux
oops.txt
```

- See `Documentation/oops-tracing.txt` and then `man ksymoops` for details.

Debugging with Kprobes

http://www-124.ibm.com/developerworks/oss/linux/projects/kprobes/

- Fairly simple way of inserting breakpoints in kernel routines
- Unlike printk debugging, you neither have to recompile nor reboot your kernel. You only need to compile and load a dedicated module to declare the address of the routine you want to probe.
- Non disruptive, based on the kernel interrupt handler
- Kprobes can even let you modify register values and global data structure values.

See http://www-106.ibm.com/developerworks/library/l-kprobes.html for a nice overview

Kernel debugging tips

- If your kernel doesn't boot yet, useful to activate Low Level debugging (Kernel Hacking section):

```
CONFIG_DEBUG_LL=y
```

Advice and resources

System security

- In production: disable loadable kernel modules if you can.
- Carefully check data from input devices (if interpreted by the driver) and from user programs (buffer overflows)
- Check kernel sources signature
- Beware of uninitialized memory
- Compile modules by yourself (beware of binary modules)

Advice and resources

Using Ethernet over USB

Ethernet over USB (1)

If your device doesn't have Ethernet connectivity, but has a USB device controller

- You can use Ethernet over USB through the `g_ether` USB device ("gadget") driver (`CONFIG_USB_GADGET`)
- Of course, you need a working USB device driver. Generally available as more and more embedded processors (well supported by Linux) have a built-in USB device controller
- Plug-in both end of the USB cable

Ethernet over USB (2)

- On the PC host, you need to have the `usbnet` module (`CONFIG_USB_USBNET`)
- Plug-in both ends of the USB cable. Configure both ends as regular networking devices. Example:
  - On the target device
    ```
    modprobe g_ether
    ifconfig usb0 192.168.0.202
    route add 192.168.0.200 dev usb0
    ```
  - On the PC
    ```
    modprobe usbnet
    ifconfig usb0 192.168.0.200
    route add 192.168.0.202 dev usb0
    ```
- Works great on iPAQ PDAs!

Advice and resources

Root filesystem on the host through NFS

Usefulness of rootfs on NFS

Once you have setup networking (Ethernet or USB-Ethernet), you can mount a filesystem on the PC through NFS and use it as the new root filesystem. This is very convenient for system development:

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

Example NFS setup

On the PC

- Add the below line to your `/etc/exports` file:
  ```
  /home/rootfs 192.168.0.202/32(rw,insecure,sync,no_wdelay,no_root_squash)
  ```
- Start or restart your NFS server (Fedora Core 2 example)
  ```
  /etc/init.d/nfs restart
  ```

On the target

- ```
  mkdir /mnt/rootfs; mkdir /mnt/initrd
  modprobe nfs
  mount -o nolock,hard,intr -t nfs 192.168.0.200:$rootfs \
  /mnt/rootfs
  ```



  ```

Using pivot_root

Once the NFS share is mounted, you can use it as the new root filesystem:

- Example (continued)

```
umount /proc
cd /mnt/rootfs
pivot_root . mnt/initrd
exec chroot . /linuxrc <dev/console >dev/console 2>&1
```

- Same `pivot_root` usage for a local storage. Used in all GNU/Linux computers with an initrd.

Advice and resources

Choosing filesystem types

Block device or MTD filesystems

- Block devices
  - Floppy or hard disks (SCSI, IDE)
  - Compact Flash (seen as a regular IDE drive)
  - RAM disks
  - Loopback devices
- Memory Technology Devices (MTD)
  - Flash, ROM or RAM chips
  - MTD emulation on block devices
- See `Documentation/filesystems/` for details

Most popular block device filesystems

Traditional filesystems: hard to recover from crashes

- ext2: traditional Linux filesystem
- vfat: traditional Windows filesystem (supporting long file names since Windows 95)

Journaled filesystems:

- ext3: ext2 with journal extension
- reiserFS: most innovative
- Others: JFS (IBM), xfs (SGI)
- NTFS: well supported by Linux in read-mode

Read-only block filesystems

- ISO 9660: used for cdroms
- UDF: used in some cdroms and DVDs
- CramFS: simple, small, compressed filesystems designed for ROM based embedded systems
  (Size < 256 MB, files < 16 MB)

The MTD subsystem

Linux filesystem interface

MTD “User” modules

jffs2

Char device

Block device

Read-only block device

Flash Translation Layers
Caution: patented algorithms!

FTL

NFTL

INFTL

MTD Chip drivers

CFI flash

RAM chips

NAND flash

DiskOnChip flash

ROM chips

Block device

Virtual memory

Virtual devices appearing as MTD devices

Memory devices hardware

MTD filesystems

- jffs2: Journaling Flash File System v2
  - Designed to write flash sectors in an homogeneous way. Flash bits can only be rewritten a relatively small number of times (often < 100 000).
  - Compressed to fit as many data as possible on flash chips. Also compensates for slower access time to those chips.
  - Power down reliable: can restart without any intervention
  - Best choice for your internal flash chips
  - Can of course be mounted as a read-only filesystem

Using block filesystems over MTD

- Can use Flash Translation Layer modules implementing a virtual block device on top of MTD. Can use a regular block filesystem on top of this virtual device then.
- FTL: Flash Translation Layer for NOR flash chips
  Caution: because of patents on algorithms, can only be used on PCMCIA hardware in the US! Better use JFFS2.
- NTFL: NAND Flash Translation Layer.
  Caution: because of M-Systems algorithm patents, can only be implemented on licensed Disc On Chip devices.

Filesystem choices for your flash devices

- MTD devices: use JFFS2 (read-write or read-only)
- Compact Flash or other removable storage
  - Can't use JFFS2 because CF storage is a block device.
    MTD Block device emulation could be used though, but JFFS2 writing scheme could interfered with on-chip flash management (manufacturer independent).
  - **Never use block device journaled filesystems on flash chips!** Keeping the journal would write the same sectors over and over again and quickly damage them.
  - Can use ext2 or vfat (caution: patents), with mount options:
    - noatime: doesn't write access time information in file inodes
    - sync: to avoid perform writes immediately (avoid power down fs failure)

Mounting a jffs2 image

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

- Mounting an MTD device as a loop device is a bit complex task. Here's an example for jffs2:

```
modprobe loop
modprobe mtdblock
losetup /dev/loop0 <file>.jffs2
modprobe blkmtd erasesz=256 device=/dev/loop0
mknod /dev/mtdblock0 b 31 0    (if not done yet)
mkdir /mnt/jffs2               (example mount point, if not done yet)
mount -t jffs2 /dev/mtdblock0 /mnt/initrd/
```

- It's very likely that your standard kernel misses one of these modules. Check the corresponding `.c` file in the kernel sources and look in the corresponding Makefile which option you need to recompile your kernel with.

Advice and resources

Getting help and contributions

Solving issues

- If you face an issue, and it doesn't look specific to your work but rather to the tools you are using, it is very likely that someone else already faced it.
- Search the Internet for similar error reports
  - On web sites or mailing list archives (using a good search engine)
  - On newsgroups: http://groups.google.com/
- You have great chances of finding a solution or workaround, or at least an explanations for your issue.
- Otherwise, reporting the issue is up to you!

Getting help

- If you have a support contract, ask your vendor
- Otherwise, don't hesitate to share your questions and issues on mailing lists
  - Either contact the Linux mailing list for your architecture (like linux-arm-kernel or linuxsh-dev...)
  - Or contact the mailing list for the subsystem you're dealing with (linux-usb-devel, linux-mtd...). Don't ask the maintainer directly!
  - Most mailing lists come with a FAQ page. Make sure you read it before contacting the mailing list
  - Refrain from contacting the Linux Kernel mailing list, unless you're an experienced developer and need advice

Getting contributions

Applies if your project can interest other people:
developing a driver or filesystem, porting Linux on a new device available on the market...

External contributors can help you a lot by

- Testing
- Writing documentation
- Making suggestions
- Even writing code

Encouraging contributions

- Open your development process: mailing list, Wiki, public CVS read access
- Let everyone contribute according to their skills and interests.
- Release early, release often
- Take feedback and suggestions into account
- Recognize contributions
- Make sure status and documentation are up to date
- Publicize your work and progress to broader audiences

Advice and resources

Bug report and patch submission to Linux developers

Reporting Linux bugs

- First make sure you're using the latest version
- Make sure you investigate the issue as much as you can: see `Documentation/BUG-HUNTING`
- Make sure the bug has not been reported yet. Check the Official Linux kernel bug database (http://bugzilla.kernel.org/) in particular.
- If the subsystem you report a bug on has a mailing list, use it. Otherwise, contact the official maintainer (see the `MAINTAINERS` file). Always give as many useful details as possible.
- Or file a new bug in http://bugzilla.kernel.org/

How to create Linux patches

- Download the **latest** kernel sources
- Make a copy of these sources:
```
rsync -a linux-2.6.9-rc2/ linux-2.6.9-rc2-patch/
```
- Apply your changes to the copied sources, and test them.
- Create a patch file:
```
diff -Nru linux-2.6.9-rc2/ \
linux-2.6.9-rc2-patch/ > patchfile
```
  - Always compare the whole source structures (suitable for `patch -p1`)
  - Patch file name: should recall the addressed issue

Thanks to Nicolas Rougier (Copyright 2003, http://webloria.loria.fr/~rougier/) for the Tux image

How to submit patches or drivers

- Don't merge patches addressing different issues
- You should identify and contact the official maintainer for the files to patch.
- See `Documentation/SubmittingPatches` for details. For trivial patches, you can copy the Trivial Patch Monkey.
- Special subsystems:
  - ARM platform: it's best to submit your ARM patches to Russel King's patch system: http://www.arm.linux.org.uk/developer/patches/

Advice and resources

References

Information sites (1)

Linux Weekly News
http://lwn.net/

- The weekly digest off all Linux and free software information sources
- In depth technical discussions about the kernel
- Subscribe to finance the editors ($5 / month)
- Articles available for non subscribers after 1 week.

Information sites (2)

KernelTrap
http://kerneltrap.org/

- Forum website for kernel developers
- News, articles, whitepapers, discussions, polls, interviews
- Perfect if a digest is not enough!

Useful reading

- Linux device drivers, 2nd edition, June 2001
  Alessandro Rubini and Jonathan Corbet, O'Reilly
  Available on-line on a free documentation license:
  http://www.xml.com/ldd/chapter/book/index.html
  Linux 2.6 updates: http://lwn.net/Articles/driver-porting/
- Understanding the Linux Kernel, 2nd Edition, Dec 2002
  Daniel P. Bovet, Marco Cesati, O'Reilly
  http://www.oreilly.com/catalog/linuxkernel2/
  Not updated for Linux 2.6 yet!
- Building Embedded Linux Systems, April 2003
  Karim Yaghmour, O'Reilly
  http://www.oreilly.com/catalog/belinuxsys/

References

- Linux kernel mailing list FAQ
  http://www.tux.org/lkml/
  Complete Linux kernel FAQ
  Read this before asking a question to the mailing list
- Kernel Newbies
  http://kernelnewbies.org/
  Glossaries, articles, presentations, HOWTOs, recommended reading, useful tools for people getting familiar with Linux kernel or driver development.

ARM resources

Processor docs

- ARM manuals: http://www.arm.com/documentation/
- Full ARM technical publications cdrom (free-as-free-beer order) http://www.arm.com/documentation/cd_request.html

ARM Linux Project

- Home page:
  http://www.arm.linux.org.uk/
- Developer documentation:
  http://www.arm.linux.org.uk/developer/
- arm-linux-kernel mailing list:
  http://lists.arm.linux.org.uk/mailman/listinfo/linux-arm-kernel
- FAQ:
  http://www.arm.linux.org.uk/armlinux/mlfaq.php
- How to post kernel fixes:
  http://www.arm.uk.linux.org/developer/patches/

Advice and resources

Last advice

Use the Source, Luke!

Many resources and tricks on the Internet find you will, but solutions to all technical issues only in the Source lie.

Thanks to LucasArts

Related documents

This document belongs to the 500 page materials of an embedded GNU / Linux training from Free Electrons, available under the GNU Free Documentation License.

- Introduction to Unix and GNU / Linux
  http://free-electrons.com/training/intro_unix_linux
- Embedded Linux kernel and driver development
  http://free-electrons.com/training/drivers
- Development tools for embedded Linux systems
  http://free-electrons.com/training/devtools
- Java in embedded Linux systems
  http://free-electrons.com/articles/java
- What's new in Linux 2.6?
  http://free-electrons.com/articles/linux26
- Introduction to uClinux
  http://free-electrons.com/articles/uclinux
- Linux real-time extensions
  http://free-electrons.com/articles/realtime

Training labs

Training labs are also available from the same location:

http://free-electrons.com/training/drivers

They are a useful complement to consolidate what you learnt from this training. They don't tell *how* to do the exercises. However, they only rely on notions and tools introduced by the lectures.

If you happen to be stuck with an exercise, this proves that you missed something in the lectures and have to go back to the slides to find what you're looking for.

Training and consulting services

This training or presentation is funded by Free Electrons customers sending their people to our training or consulting sessions.

If you are interested in attending training sessions performed by the author of these documents, you are invited to ask your organization to order such sessions.

See http:/free-electrons/training for more details.

If you just support this work, do not hesitate to speak about it to your friends, colleagues and local Free Software community.