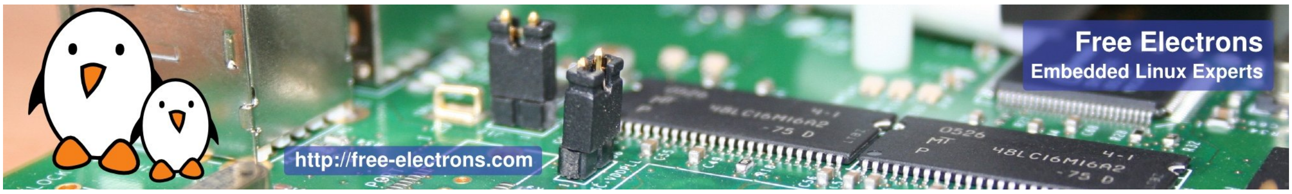
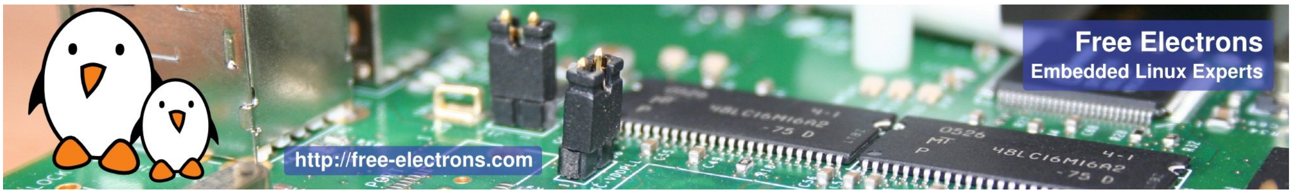


Embedded Linux kernel and driver development training

5 day session

Overview

Title	**Embedded Linux kernel and driver development training**
Overview	Understanding the Linux kernel Developing Linux device drivers Linux kernel debugging Porting the Linux kernel Working with the kernel development community Practical labs with ARM boards as well as emulated PC systems.
Duration	5 days. 50% of presentations and 50% of practical labs.
Trainer	Gregory Clement, Thomas Petazzoni, or Michael Opdenacker. See http://free-electrons.com/company/staff/
Language	Oral lectures: English or French Materials: English
Audience	People developing devices using the Linux kernel People supporting embedded Linux system developers.
Prerequisites	**Knowledge and practice of Unix or GNU/Linux commands** People lacking experience on this topic should get trained by themselves with our freely available on-line slides (http://free-electrons.com/docs/command-line/) **Knowledge and practice of C programming** **Familiarity with configuring, compiling and booting Linux**
Required equipment	Video projector PC computers with at least 1 GB of RAM, and Ubuntu Linux installed in a **free partition of at least 10 GB**. **Using Linux in a virtual machine is not supported**, because of issues connecting to real hardware. We need a 32 bit (`i386`) version of Ubuntu Desktop 11.04 (Xubuntu and Kubuntu variants are fine). We don't support other distributions, because we can't test all possible package versions. **Connection to the Internet** (direct or through the company proxy). **PC computers with valuable data must be backed up** before being used in our sessions. Some people have already made mistakes during our sessions and damaged work data.
Materials	Print and electronic copy of presentations and labs. Electronic copy of lab files.



Embedded Linux kernel and driver development training

See our slides on http://free-electrons.com/doc/training/linux-kernel
This way, you can check by yourself whether our slides correspond to your needs.

Hardware

Using USB-A9263 boards from CALAO Systems in some practical labs.
Using virtual systems emulated by QEMU otherwise.

AT91SAM9263 ARM CPU from ATMEL
64 MB RAM, 256 MB flash
2 USB 2.0 host
1 USB device
100 Mbit Ethernet port
Powered by USB!
Serial and JTAG through this USB port
Multiple expansion boards available

Day 1 - Morning

Lecture – Introduction to the Linux kernel

Kernel features
Understanding the development process.
Legal constraints with device drivers.
Kernel user interface (/proc and /sys)
Userspace device drivers

Lecture - Kernel sources

Specifics of Linux kernel development
Coding standards
Retrieving Linux kernel sources
Tour of the Linux kernel sources
Kernel source code browsers: cscope, Kscope, Linux Cross Reference (LXR)

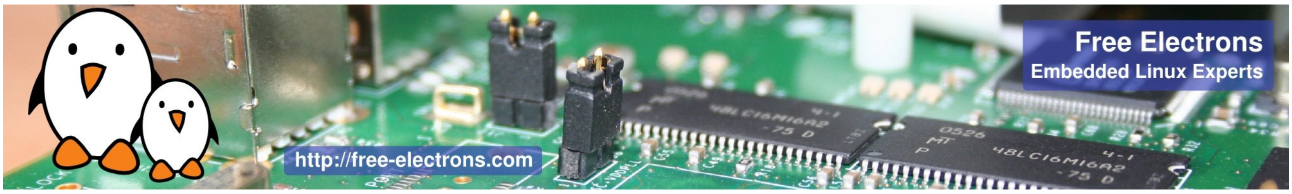
Lab - Kernel source code

Making searches in the Linux kernel sources: looking for C definitions, for definitions of kernel configuration parameters, and for other kinds of information.
Using the Unix command line and then kernel source code browsers.

Day 1 - Afternoon

Lecture – Configuring, compiling and booting the Linux kernel

Kernel configuration.
Native compiling. Generated files.
Booting the kernel. Kernel booting parameters.



Lecture - NFS booting and cross-compiling	**Lab – Kernel configuration, cross-compiling and booting on NFS**
Booting on a directory on your GNU/Linux workstation, through NFS. Kernel cross-compiling	*Using the CALAO board* Configuring, cross-compiling and booting a Linux kernel with NFS boot support.

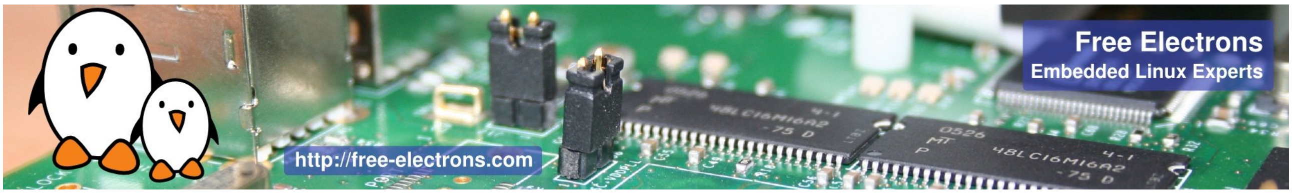
Day 2 - Morning

Lecture – Linux kernel modules	**Lab - Writing modules**
Linux device drivers A simple module Programming constraints Loading, unloading modules Module parameters Module dependencies Adding sources to the kernel tree Generating patches to share them with others	*Continued from the previous lab* Write a kernel module with several capabilities, including module parameters. Access kernel internals from your module. Setup the environment to compile it

Lecture - Memory management
Linux: memory management - Physical and virtual (kernel and user) address spaces. Linux memory management implementation. Allocating with `kmalloc()`. Allocating by pages. Allocating with `vmalloc()`.

Day 2 - Afternoon

Lecture - I/O memory and ports	**Lab - I/O memory and ports**
I/O register and memory range registration. I/O register and memory access. Read / write memory barriers.	Make a remote connection to your board through ssh. Access the system console through the network. Reserve the I/O memory addresses used by the serial port. Read device registers and write data to them, to send characters on the serial port.



Lecture - Character drivers	Lab - Character drivers
Device numbers Getting free device numbers Implementing file operations: read, write, open, close, ioctl... Exchanging data between kernelspace and userspace Character driver registration	*Using the CALAO ARM board* Writing a simple character driver, to write data to the serial port. On your workstation, checking that transmitted data is received correctly. Exchanging data between userspace and kernel space. Practicing with the character device driver API. Using kernel standard error codes.

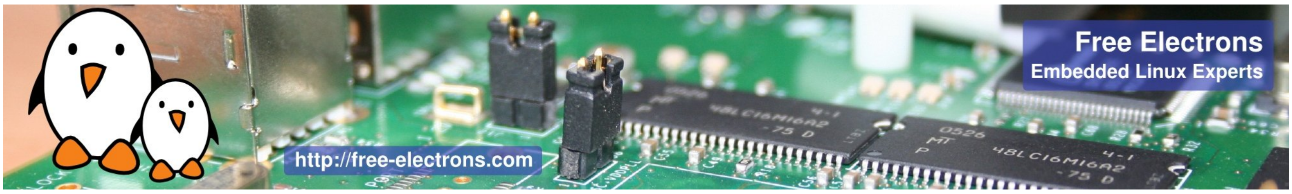
Day 3 - Morning

Lecture – Processes, scheduling, sleeping and interrupts
Process management in the Linux kernel. The Linux kernel scheduler and how processes sleep. Interrupt handling in device drivers: interrupt handler registration and programming, scheduling deferred work.

Lab – Sleeping and handling interrupts in a device driver
Using the CALAO ARM board. Adding read capability to the character driver developed earlier. Register an interrupt handler. Waiting for data to be available in the read file operation. Waking up the code when data is available from the device.

Day 3 - Afternoon

Lecture - Locking	Lab - Locking
Issues with concurrent access to resources Locking primitives: mutexes, semaphores, spinlocks. Atomic operations. Typical locking issues. Using the lock validator to identify the sources of locking problems.	*Continued from the previous lab.* Observe problems due to concurrent accesses to the device. Add locking to the driver to fix these issues.



Lecture - Driver debugging techniques	Lab – Investigating kernel faults
Debugging with printk Proc and debugfs entries Analyzing a kernel oops Using kgdb, a kernel debugger Using the Magic SysRq commands Debugging through a JTAG probe SystemTap and demonstration	*Using the CALAO ARM board* Studying a broken driver. Analyzing a kernel fault and locating the problem in the source code.

Day 4 - Morning

Lecture – mmap	Lecture – The DMA API
Process virtual memory areas Maximizing performance with mmap, allowing applications to access the hardware directly. Implementing mmap in drivers	The Linux kernel DMA API. Using it in device drivers.

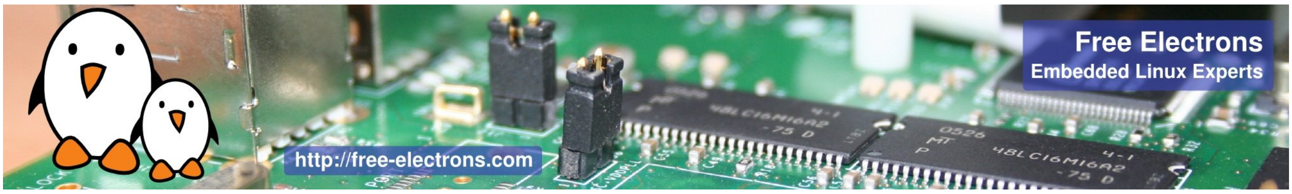
Lecture – Kernel architecture for device drivers
Understand how the kernel is designed to support device drivers The kernel device driver «framework» for common types of devices The device model Binding devices and drivers Platform devices Interface in userspace: /sys

Day 4 - Afternoon

Lecture – Serial drivers	Lab – Implement a serial driver
As an illustration of one particular kernel framework, details on the serial driver framework.	*On the ARM board* Implement parts of a serial driver through the kernel's serial framework.

Day 5 – Morning

Lecture - Kernel boot-up details
Detailed description of the kernel boot-up process, from execution by the bootloader to the execution of the first userspace program. Initcalls: how to register your own initialization routines.



Lecture – Porting the Linux kernel	**Lecture - Introduction to power management**
Porting the Linux kernel. Creating board dependent code. Detail study of code for an ARM board.	Supporting frequency scaling CPU and board specific power management. Power management in device drivers. Control from user space. Saving power in the idle loop. Voltage and current regulator framework Studying power management implementations in the Linux kernel.

Lab – Power management
Using the Linux workstation and if possible, the CALAO ARM board. Practicing with the standard power management interfaces: suspend / resume and cpu frequency control. Identifying top sources of power consumption with PowerTop.

Day 5 - Afternoon

Lecture – Working with the community
How to get help from the community. Report bugs. Generate and send patches. Useful resources about the kernel

Lecture – Managing kernel sources with git	**Lab - Using git**
Very useful to manage your changes to the Linux kernel (drivers, board support code), staying in sync with mainstream updates. Cloning an existing git tree Creating your own branch with your own changes. Generating patches against the reference tree. Review of useful git commands. Understanding the work flow used by kernel developers, through the study of typical scenarios.	Create your own git branch from the mainline tree. Get changes from trees and generate your own patchset. Keep your branch updated with the changes in your reference tree.