Embedded Linux kernel and driver development training

On-line seminar

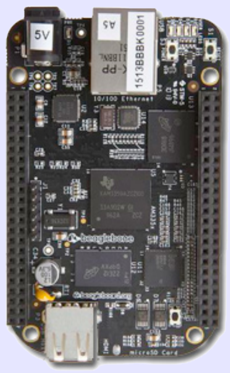
Title	**Embedded Linux kernel and driver development training**
Overview	Understanding the Linux kernel Developing Linux device drivers Linux kernel debugging Porting the Linux kernel to a new board Working with the kernel development community Practical demos with the ARM-based BeagleBone Black board (or with its Wireless variant).
Materials	Check that the course contents correspond to your needs: `https://bootlin.com/doc/training/linux-kernel`.
Duration	**Seven** half days - 28 hours (4 hours per half day). 80% of lectures, 20% of practical demos.
Trainer	One of the engineers listed on `https://bootlin.com/training/trainers/`
Language	Oral lectures: English Materials: English.
Audience	People developing devices using the Linux kernel People supporting embedded Linux system developers.
Prerequisites	**Familiarity with C programming** In particular, participants should understand complex data types and structures, pointers and function pointers. **Familiarity with UNIX or GNU/Linux commands** People lacking experience on this topic could get trained by themselves, for example with our freely available on-line slides (`https://bootlin.com/blog/command-line/`). **Familiarity with embedded Linux development.** Taking our Embedded Linux course (`https://bootlin.com/training/embedded-linux/`) first is not a requirement, but it will definitely allow to understand the development environment and board manipulations, allowing to concentrate on kernel code programming.

Required equipment	• Computer with the operating system of your choice, provided it is supported by Zoom (`https://zoom.us` for videoconferencing • Webcam and microphone (preferably from an audio headset) • High speed access to the Internet
Materials	Electronic copies of presentations, demo instructions and data.

Hardware

The hardware platform used for the practical demos of this training session is the **BeagleBone Black** board, which features:

- An ARM AM335x processor from Texas Instruments (Cortex-A8 based), 3D acceleration, etc.
- 512 MB of RAM
- 2 GB of on-board eMMC storage
(4 GB in Rev C)
- USB host and device
- HDMI output
- 2 x 46 pins headers, to access UARTs, SPI buses, I2C buses and more.



Demos

The practical demos of this training session use the following hardware peripherals to illustrate the development of Linux device drivers:

- A Wii Nunchuk, which is connected over the I2C bus to the BeagleBone Black board. Its driver will use the Linux *input* subsystem.
- An additional UART, which is memory-mapped, and will use the Linux *misc* subsystem.

While our explanations will be focused on specifically the Linux subsystems needed to implement these drivers, they will always be generic enough to convey the general design philosophy of the Linux kernel. The information learnt will therefore apply beyond just I2C, input or memory-mapped devices.

Half day 1

Lecture - Introduction to the Linux kernel

- Kernel features
- Understanding the development process.
- Legal constraints with device drivers.
- Kernel user interface (/proc and /sys)
- User space 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, Elixir

Demo - Kernel sources

- 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.

Lecture - Configuring, compiling and booting the Linux kernel

- Kernel configuration.
- Native and cross compilation. Generated files.
- Booting the kernel. Kernel booting parameters.
- Mounting a root filesystem on NFS.

Half day 2

Demo - Kernel configuration, cross-compiling and booting on NFS

Using the BeagleBone Black board

- Configuring, cross-compiling and booting a Linux kernel with NFS boot support.

Lecture - Linux kernel modules

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

Demo - Writing modules

Using the BeagleBone Black board

- Write a kernel module with several capabilities.
- Access kernel internals from your module.
- Set up the environment to compile it

Lecture - Linux device model

- Understand how the kernel is designed to support device drivers
- The device model
- Binding devices and drivers
- Platform devices, Device Tree
- Interface in user space: `/sys`

Half day 3

Demo - Linux device model for an I2C driver

Using the BeagleBone Black board

- Implement a driver that registers as an I2C driver
- Modify the Device Tree to list an I2C device
- Get the driver called when the I2C device is enumerated at boot time

Lecture - Introduction to the I2C API

- The I2C subsystem of the kernel
- Details about the API provided to kernel drivers to interact with I2C devices

Lecture - Pin muxing

- Understand the *pinctrl* framework of the kernel
- Understand how to configure the muxing of pins

Demo - Communicate with the Nunchuk over I2C

Using the BeagleBone Black board

- Configure the pin muxing for the I2C bus used to communicate with the Nunchuk
- Extend the I2C driver started in the previous lab to communicate with the Nunchuk via I2C

Half day 4

Lecture - Kernel frameworks

- Block vs. character devices
- Interaction of user space applications with the kernel
- Details on character devices, `file_operations`, `ioctl()`, etc.
- Exchanging data to/from user space
- The principle of kernel frameworks

Lecture - The input subsystem

- Principle of the kernel *input* subsystem
- API offered to kernel drivers to expose input devices capabilities to user space applications
- User space API offered by the *input* subsystem

Demo - Expose the Nunchuk functionality to user space

Using the BeagleBone Black board

- Extend the Nunchuk driver to expose the Nunchuk features to user space applications, as a *input* device.
- Test the operation of the Nunchuk using `evtest`

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()`.

Half day 4

Lecture - I/O memory and ports

- I/O register and memory range registration.
- I/O register and memory access.
- Read / write memory barriers.

Demo - Minimal platform driver and access to I/O memory

Using the BeagleBone Black board

- Implement a minimal platform driver
- Modify the Device Tree to instantiate the new serial port device.
- 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 - The misc kernel subsystem

- What the *misc* kernel subsystem is useful for
- API of the *misc* kernel subsystem, both the kernel side and user space side

Demo - Output-only serial port driver

Using the BeagleBone Black board

- Extend the driver started in the previous lab by registering it into the *misc* subsystem
- Implement serial port output functionality through the *misc* subsystem
- Test serial output from user space

Half day 5

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.

Demo - Sleeping and handling interrupts in a device driver

Using the BeagleBone Black 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.

Lecture - Locking

- Issues with concurrent access to shared resources
- Locking primitives: mutexes, semaphores, spinlocks.
- Atomic operations.
- Typical locking issues.
- Using the lock validator to identify the sources of locking problems.

Demo - Locking

Using the BeagleBone Black board

- Add locking to the current driver

Half day 6

Lecture - Driver debugging techniques

- Debugging with printk
- Using Debugfs
- Analyzing a kernel oops
- Using kgdb, a kernel debugger
- Using the Magic SysRq commands
- Debugging through a JTAG probe

Demo - Investigating kernel faults

Using the BeagleBone Black board

- Studying a broken driver.
- Analyzing a kernel fault message and locating the problem in the source code.

Lecture - ARM board support and SoC support

- Understand the organization of the ARM support code
- Understand how the kernel can be ported to a new hardware board

Half day 7

Lecture - Power management

- Overview of the power management features of the kernel
- Topics covered: clocks, suspend and resume, dynamic frequency scaling, saving power during idle, runtime power management, regulators, etc.

Lecture - The Linux kernel development process

- Organization of the kernel community
- The release schedule and process: release candidates, stable releases, long-term support, etc.
- Legal aspects, licensing.
- How to submit patches to contribute code to the community.
- Kernel resources: books, websites, conferences

Lecture - If time left

- DMA
- mmap
- Introduction to Git

Demo - If time left

- Get familiar with git by contributing to a real project: the Linux kernel
- Send your patches to the maintainers and mailing lists.

Questions and Answers

- Questions and answers with the audience about the course topics
- Extra presentations if time is left, according what most participants are interested in.