# Embedded Linux kernel and driver development training

## 5-day session

<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Embedded Linux kernel and driver development training</th>
</tr>
</thead>
</table>
| **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 labs 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](https://bootlin.com/doc/training/linux-kernel). |
| **Duration** | **Five** days - 40 hours (8 hours per day).  
50% of lectures, 50% of practical labs. |
| **Trainer** | One of the engineers listed on [https://bootlin.com/training/trainers/](https://bootlin.com/training/trainers/) |
| **Language** | Oral lectures: English or French.  
Materials: English. |
| **Audience** | People developing devices using the Linux kernel  
People supporting embedded Linux system developers. |
| **Prerequisites** | **Solid experience in C programming**  
In particular, participants must be familiar with creating and dealing with complex data types and structures, with pointers to such symbols, as well as with function pointers.  

**Knowledge and practice of UNIX or GNU/Linux commands**  
People lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides ([https://bootlin.com/blog/command-line/](https://bootlin.com/blog/command-line/)).  

**Experience in embedded Linux development.**  
Taking our Embedded Linux course ([https://bootlin.com/training/embedded-linux/](https://bootlin.com/training/embedded-linux/)) first is not an absolute prerequisite, but it will definitely help people lacking embedded Linux development experience. They will understand the development environment and board manipulations better, allowing them to concentrate on kernel code programming. |
Required equipment

For on-site sessions only
Everything is supplied by Bootlin in public sessions.

- Video projector
- PC computers with at least 8 GB of RAM, and Ubuntu Linux installed in a free partition of at least 20 GB. Using Linux in a virtual machine is not supported, because of issues connecting to real hardware.
- We need Ubuntu Desktop 20.04 (Xubuntu and other 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

Electronic copies of presentations and labs.
Electronic copy of lab files.

Hardware

The hardware platform used for the practical labs 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.

Labs

The practical labs 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.
Day 1 - Morning

Lecture - Introduction to the Linux kernel

- Kernel features
- 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, Elixir Cross Referencer, VS Code...

Lab - 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.

Day 1 - Afternoon

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.

Lab - 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.
### Day 2 - Morning

<table>
<thead>
<tr>
<th>Lecture - Linux kernel modules</th>
<th>Lab - Writing modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Linux device drivers</td>
<td>Using the BeagleBone Black board</td>
</tr>
<tr>
<td>• A simple module</td>
<td>• Write a kernel module with several capabilities.</td>
</tr>
<tr>
<td>• Programming constraints</td>
<td>• Access kernel internals from your module.</td>
</tr>
<tr>
<td>• Loading, unloading modules</td>
<td>• Set up the environment to compile it</td>
</tr>
<tr>
<td>• Module dependencies</td>
<td></td>
</tr>
<tr>
<td>• Adding sources to the kernel tree</td>
<td></td>
</tr>
</tbody>
</table>

### Day 2 - Afternoon

<table>
<thead>
<tr>
<th>Lecture - Linux device model</th>
<th>Lab - Linux device model for an I2C driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Understand how the kernel is designed to support device drivers</td>
<td>Using the BeagleBone Black board</td>
</tr>
<tr>
<td>• The device model</td>
<td>• Implement a driver that registers as an I2C driver</td>
</tr>
<tr>
<td>• Binding devices and drivers</td>
<td>• Modify the Device Tree to list an I2C device</td>
</tr>
<tr>
<td>• Platform devices, Device Tree</td>
<td>• Get the driver called when the I2C device is enumerated at boot time</td>
</tr>
<tr>
<td>• Interface in user space: /sys</td>
<td></td>
</tr>
</tbody>
</table>

### Day 3 - Morning

<table>
<thead>
<tr>
<th>Lecture - Introduction to the I2C API</th>
<th>Lecture - Pin muxing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The I2C subsystem of the kernel</td>
<td>• Understand the <em>pinctrl</em> framework of the kernel</td>
</tr>
<tr>
<td>• Details about the API provided to kernel drivers to interact with I2C devices</td>
<td>• Understand how to configure the muxing of pins</td>
</tr>
</tbody>
</table>
Lab - 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

Day 3 - Afternoon

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

Lab - 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
### Day 4 - Morning

<table>
<thead>
<tr>
<th>Lecture - Memory management</th>
<th>Lecture - I/O memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Linux: memory management - Physical and virtual (kernel and user) address spaces.</td>
<td>• I/O memory range registration.</td>
</tr>
<tr>
<td>• Linux memory management implementation.</td>
<td>• I/O memory access.</td>
</tr>
<tr>
<td>• Allocating with <code>kmalloc()</code>.</td>
<td>• Read / write memory barriers.</td>
</tr>
<tr>
<td>• Allocating by pages.</td>
<td></td>
</tr>
<tr>
<td>• Allocating with <code>vmalloc()</code>.</td>
<td></td>
</tr>
</tbody>
</table>

### Lab - 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.

### Day 4 - Afternoon

<table>
<thead>
<tr>
<th>Lecture - The misc kernel subsystem</th>
<th>Lab - Output-only serial port driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What the <code>misc</code> kernel subsystem is useful for</td>
<td><strong>Using the BeagleBone Black board</strong></td>
</tr>
<tr>
<td>• API of the <code>misc</code> kernel subsystem, both the kernel side and user space side</td>
<td>• Extend the driver started in the previous lab by registering it into the <code>misc</code> subsystem</td>
</tr>
</tbody>
</table>

• Implement serial port output functionality through the `misc` subsystem
• Test serial output from user space
### 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 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.

### Day 5 - Morning

### 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.

### Lab - Locking
**Using the BeagleBone Black board**
- Add locking to the current driver

### Lecture - Driver debugging techniques
- Debugging with printing functions
- Using Debugfs
- Analyzing a kernel oops
- Using kgdb, a kernel debugger
- Using the Magic SysRq commands

### Lab - 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.
### Day 5 - Afternoon

<table>
<thead>
<tr>
<th>Lecture - ARM board support and SoC support</th>
<th>Lecture - Power management</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Understand the organization of the ARM support code</td>
<td></td>
</tr>
<tr>
<td>• Understand how the kernel can be ported to a new hardware board</td>
<td></td>
</tr>
<tr>
<td>• Overview of the power management features of the kernel</td>
<td></td>
</tr>
<tr>
<td>• Topics covered: clocks, suspend and resume, dynamic frequency scaling, saving power during idle, runtime power management, regulators, etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lecture - The Linux kernel development process</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Organization of the kernel community</td>
</tr>
<tr>
<td>• The release schedule and process: release candidates, stable releases, long-term support, etc.</td>
</tr>
<tr>
<td>• Legal aspects, licensing.</td>
</tr>
<tr>
<td>• How to submit patches to contribute code to the community.</td>
</tr>
<tr>
<td>• Kernel resources: books, websites, conferences</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lecture - If time left</th>
<th>Questions and Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>• DMA</td>
<td></td>
</tr>
<tr>
<td>• mmap</td>
<td></td>
</tr>
<tr>
<td>• Questions and answers with the audience about the course topics</td>
<td></td>
</tr>
<tr>
<td>• Extra presentations if time is left, according what most participants are interested in.</td>
<td></td>
</tr>
</tbody>
</table>