DOULOS

Training

Practical Embedded Linux Device Drivers

A hands-on course to enable you to write device drivers for hardware peripherals and devices in an embedded Linux system.

COVID-19 Update: April - August 2020
Training Available Live Online Only - View Now »

COURSE DATES:

June 8th, 2020	ONLINE Americas	Enquire
June 15th, 2020	ONLINE EurAsia	Enquire
August 3rd, 2020	ONLINE Americas	Enquire
August 10th, 2020	ONLINE EurAsia	Enquire

indicates CONFIRMED TO RUN courses.

Looking for team-based training, or other locations?

Complete an on-line form and a Doulos representative will get back to you »

view dates and locations for in-person training only

Standard Level - 4 days

FACE-TO-FACE TRAINING

Practical Embedded Linux Device Drivers is designed to give engineers the knowledge and skills to work confidently with all the components of the kernel to successfully develop device drivers.

Workshops comprise approximately 50% of this 4-day training course, with carefully designed hands-on exercises to reinforce learning. The workshops use devices from the 96Boards project, which provides a flexible learning environment that can replicate the individual projects attendees are facing. The 96Boards family includes devices from many of the major SoC Vendors, applicable to a wide range of different application contexts.

Why choose this particular course?

The ever-growing demand for connectivity and multimedia applications is resulting in embedded Linux systems driving increasingly complex devices. Developing custom device drivers for the Linux kernel can be a complex and difficult task, with an array of implementation choices available.

This course aims to **reduce development time** by demonstrating how the range of helper functions and mechanisms provided by the kernel can make custom device driver creation easier. Attendees
should come away from the course with a clear understanding of how to go about designing their
device driver and what kernel mechanisms they can make use of.

All the main kernel interfaces and structural elements are covered in the course. So as well as building confidence in working with the kernel components needed for a specific project, attendees also gain a sound understanding of the overall framework. This ensures the KnowHow delivered by the training will continue to be useful and valid for future device driver projects.

Who should attend?

This course is suitable for engineers working in SoC, FPGA or custom board environments who want to get started quickly on developing custom kernel drivers. Although the course is primarily targeting

ARM based embedded systems it is still relevant for users of other commonly used embedded CPUs such as MIPS, PowerPC etc.

What you will learn

- An understanding of the capabilities of the embedded Linux kernel and the techniques to work within it effectively
- The facilities and frameworks of the kernel and how they can be used to speed up driver development
- How to utilise the common patterns and data structures for different types of device driver.

Pre-requisites

- Completion of Developing with Embedded Linux training or equivalent working knowledge of using Linux as a host operating system
- Some familiarity with C programming is also necessary.

Hardware

This is a hands-on training course and labs are conducted on a real target board (one of the 96Boards family of development platforms), with an ARM 32bit or 64bit CPU.
The course also takes advantage of the **Grove Sensor Mezzanine** board to enable attendees to develop and test drivers with real hardware.

Please contact the Doulos team to discuss your specific project and hardware requirements.

Course materials

Doulos course materials are renowned for being the most comprehensive and user friendly available. Their unique style, content and coverage has made them sought after resources in their own right. The materials include fully indexed class notes creating a complete reference manual.

Structure and Content

DAY 1

Introduction to Kernel Development

- Course introduction
- The kernel development process
- Module licensing
- Documentation and help
- Appendix: Git
- **Exercises:**
  - Booting the target
  - Working with a kernel source tree browser

Kernel & Module Building

- Configuring & compiling the kernel
- Structure of a kernel module
- Parameters, symbols and dependencies
- Use counts and referencing
- The kernel build system
- Code checking with Sparse
- **Exercises:**
  - Building a driver out of tree
  - Adding the driver to the kernel source tree
  - Testing the driver with real hardware

Kernel Debugging

- Kernel symbols
- Oops and panics
- Printing debug statements
- JTAG
- KGDB/KDB
- Kernel tracing
- **Exercises:**
  - Using warn and panic to debug a driver issue
  - Tracing the kernel to analyze the performance of a driver

DAY 2

Device Driver Models

- Driver model data structures
- Platform devices and drivers
- kobjects & the SysFS

- Managed device resources
- **Exercises:**
  - Adding the driver model structures and functions to a skeleton I2C client driver
  - Testing the driver with real hardware

Working with Devicetrees

- Linux board support
- Devicetree compilation
- Devicetree syntax
- Parsing the devicetree
- Tools and debugging
- **Exercises:**
  - Decompiling a devicetree blob to get the source
  - Extending the devicetree to add information
  - Parsing the devicetree data from a driver

User Space Interfaces

- Device types & files
- File operations
- Char devices
- DebugFS
- **Exercises:**
  - Adding a character device interface to allow the driver to be configured from user space
  - Writing a simple application to use an IOCTL to configure the driver

DAY 3

Hardware Interfaces

- Virtual and physical memory
- Allocating memory
- IO memory
- Direct Memory Access
- GPIO
- **Exercises:**
  - Adding support for device control via a GPIO button
  - Extending the devicetree to support this
- Registering interrupts
- Other HW IO frameworks (Regmap, Pinctrl)
- **Exercises:**
  - Extending the driver to incorporate an interrupt handler
  - Testing with the hardware

Data Structures and Concurrency

- Linked Lists
- Data Trees
- Kernel helper macros and functions
- Concurrency and locks
- Mutexes
- Spinlocks
- **Exercises:**
  - Implementing a simple linked-list
  - Resolving contention within a driver with a mutex

DAY 4

Scheduling and Managing Work

- Tasks and scheduling
- Wait queues and sleeping
- Timers
- Workqueues
- **Exercise:**
  - Using timers and work queues to change how data is displayed by the driver

User Space Drivers

- Accessing Memory from User Space
- User Space IO

- LibGPIOd
- I2C & SPI IO
- FUSE, CUSE & BUSE
- **Exercises:**
  - Implementing a simple devmen driver
  - Using LibGPIOd to manage the same hardware

Frameworks and Subsystems

- ALSA
- DRM
- USB
- Industrial I/O
- Input Devices
- RemoteProc
- Firmware Loading
- Power Management
- Networking
- 'Staging' drivers
- **Exercise:**
  - Reviewing how an existing driver uses kernel frameworks

COURSE DATES:

October 6th, 2020	Boston, MA	Enquire
October 20th, 2020	Munich, DE	Enquire
December 1st, 2020	San Jose, CA	Enquire
December 7th, 2020	Ringwood, UK	Enquire

indicates CONFIRMED TO RUN courses.

Looking for team-based training, or other locations?

Complete an on-line form and a Doulos representative will get back to you »

Price on request

Back to top