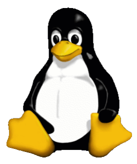
Writing Linux Device Drivers

Standard Level - 5 days

Currently available in the US ONLY
If you are looking for device driver training in Europe, please click here »

view dates and locations in the US



Linux is being used more and more on embedded systems driven by increasingly complex devices and a greater requirement for connectivity & multimedia applications. Developing custom device drivers for the Linux kernel can be a complex and difficult task, with an array of choices available on how best to implement what is required for your system.
Writing Linux Device Drivers is a 5 day course providing the practical skills and knowledge required to work with the Linux kernel in this environment.
The course provides a step by step approach on how to interact with complex hardware in the Linux kernel, building up from a simple "hello world" module through to managing much more advanced devices. Good practice when writing Linux drivers is explained in-depth, looking at issues to do with synchronisation, concurrency, power management and so on. Working with USB and network devices is also considered in detail.
Delegates will achieve the learning outcomes and gain the skills they need to develop and maintain custom Kernel device drivers for embedded Linux based products.
Workshops comprise approximately 50% of class time, and are based around carefully designed Labs to reinforce and challenge the extent of learning. The course uses a recent Linux kernel, as delivered on www.kernel.org.
This course is provided in partnership with embedded experts AC6.

What you will learn

- Mastering kernel development and debug tools
- Discovering multi-core programming in the Linux kernel
- Programming IOs, interrupts, timers and DMA
- Installing and integrating drivers inside the Linux kernel
- Managing synchronous and asynchronous IOs and ioctl
- Writing a complete character driver
- Mastering kernel debugging techniques with Lauterbach JTAG probes.

Who should attend?

- This course is designed for anyone wishing to get started quickly on developing custom kernel drivers for an embedded Linux system.

Pre-requisites

Attendance of Developing with Embedded Linux is strongly recommended and/or a good understanding of:
- Knowledge and practice of C programming
- Preferably knowledge of Linux user programming.

Hardware

This is a hands-on training course and labs are conducted on a real target board:
- Quad Arm® Cortex®-A9-based "SabreLite" boards from NXP.
- Quad Arm Cortex-A53-based "imx8q-evk" boards from NXP.

We use a recent (4.x) linux kernel, as supported by the chip supplier.

Structure and Content

Day 1

Linux kernel programming

- Development in the Linux kernel
- Memory allocation
- Linked lists
- Debug tools

Lab: Writing the "hello world" kernel module
Lab: Adding a driver to kernel sources and configuration menu
Lab: Using module parameters
Lab: Writing interdependent modules using memory allocations, reference counting and linked lists

Kernel multi-tasking

- Task handling
- Concurrent programming
- Timers
- Kernel threads

Lab: Fixing race conditions in the previous lab with mutexes

Day 2

Introduction to Linux drivers

- Accessing the device driver from user space
- Driver registration

Lab: Step by step implementation of a character driver:

- driver registration (major/minor reservation) and device special file creation (/dev)

Driver I/O functions

- Kernel structures used by drivers
- Opening and closing devices
- Data transfers
- Controlling the device
- Mapping device memory

Lab: Step by step implementation of a character driver:

- Implementing open and release
- Implementing read and write
- Implementing ioctl
- Implementing mmap

Day 3

Synchronous and asynchronous requests

- Task synchronization
- Synchronous request
- Asynchronous requests

Lab: Implementation of a pipe-like driver:

- implementing waiting and waking
- adding non-blocking, asynchronous and multiplexed operations
  (O_NONBLOCK, SIGIO, poll/select)

Input/Output and interrupts

- Memory-mapped registers
- Interrupts
- Gpios

Lab: Polling gpio driver with raw register access
Lab: Interrupt-based gpio driver with raw register access
Lab: gpio driver using the gpiolib

Busses

- Plug-and-Play management
- Static devices declaration
  - in the BSP code

    - in the device tree
- Platform bus
- PCI
- SPI
- Power management
    - System sleep
    - Implementing power management in drivers
    - Remote wakeup

Lab: Implementing a platform driver and customizing the device tree to associate it to its device (a serial port)
Lab: Implementing power management in the previous driver
Lab: Implementing remote wakeup in the previous driver
Â

Day 4
Linux Driver Model

- Linux Driver Model Architecture
    - Overview
    - Classes
    - Busses
- Hot plug management
    - Plugging devices
    - Removing devices
- Writing udev rules

Lab: Writing a custom class driver
Lab: Writing a misc driver

Memory management

- Virtual Memory
- Memory Allocation
    - Free page management
    - Normal memory allocation
    - Virtual memory allocation
    - Huge allocations

DMA

- Direct Memory Access
    - DMA scenarios
    - Buffer access
- DMA programming
    - Bus master DMA
    - Slave DMA
- Memory barriers

Lab: Implementing slave DMA in a serial port driver

Day 5
USB Drivers

- The USB bus
- USB devices
- User-space USB interface
- USB descriptors
- USB requests
- USB device drivers

Lab: Writing a USB host driver

Network drivers

- Structures
    - network interface representation (struct net_device)
    - network packet (struct sk_buff)
- Scatter/gather
- Interface
    - receiving packets
    - sending packets
    - lost packets management
    - network interface statistics
- New network API (NAPI)
    - "interrupt mitigation" (suppression of unneeded IRQs)

  - "packet throttling" (suppression of packets in the driver itself when system is overwhelmed)

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