



## D3 - Linux Drivers

### *Writing Linux Drivers*

#### Objectives

- 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 Linux kernel
- Managing synchronous and asynchronous IOs and ioctl
- Writing a complete character driver
- Understanding specificities of 2.6 and 3.x versions
- Mastering kernel debugging technics with Lauterbach JTAG probes.

Labs are conducted on target boards, that can be:

Dual Cortex/A7-based "STM32MP15-DISCO" boards from STMicroelectronics.

Quad Cortex/A9-based "SabreLite" boards from NXP.

Quad Cortex/A53-based "imx8q-evk" boards from NXP.

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

#### Target audience

- This course is for engineers that install Linux on a custom platform and have to create specific device drivers.

#### Course environment

- Printed course material (in English)
- One Linux PC for two trainees.
- One target platform for two trainees

#### Prerequisite

- Good C programming skills
- Preferably knowledge of Linux user programming (see our [D0 - Linux user mode programming](#) course or [oD0 - Linux User Mode Programming](#) course)

## Course Outline

### FIRST DAY

#### Linux kernel programming

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

**Exercise:** Writing the "hello world" kernel module

**Exercise:** Adding a driver to kernel sources and configuration menu

**Exercise:** Using module parameters

**Exercise:** Writing interdependent modules using memory allocations, reference counting and linked lists

## Linux kernel debugging

- The /proc and debugfs filesystems
- Traces
- The kernel Dynamic Debugging interface
- The Kernel Address Sanitizer
- Debugging memory problems with kmemleak
- Using the Undefined Behavior Sanitizer
- Code coverage using gcov
- Debugging with kgdb
- Debugging with a JTAG probe

**Exercise:** Display dynamic traces on the running kernel

**Exercise:** Debug a module initialization using kgdb

## Kernel multi-tasking

- Task handling
- Concurrent programming
- Timers
- Kernel threads

**Exercise:** Fixing race conditions in the previous lab with mutexes

# SECOND DAY

## Introduction to Linux drivers

- Accessing the device driver from user space
- Driver registration

**Exercise:** 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

**Exercise:** Step by step implementation of a character driver:

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

# THIRD DAY

## Synchronous and asynchronous requests

- Task synchronization
- Synchronous request
- Asynchronous requests

**Exercise:** 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
- User-level access through /sys or the GPIO character driver

**Exercise:** Polling gpio driver with raw register access

**Exercise:** Interrupt-based gpio driver with raw register access

**Exercise:** 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

**Exercise:** Implementing a platform driver and customizing the device tree to associate it to its device (a serial port)

**Exercise:** Implementing power management in the previous driver

**Exercise:** Implementing remote wakeup in the previous driver

## FOURTH DAY

### Linux Driver Model

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

**Exercise:** Writing a custom class driver

**Exercise:** Writing a misc driver

### DMA

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

**Exercise:** Implementing slave DMA in a serial port driver

# ANNEXES

## USB Drivers

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

**Exercise:** 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)

## Memory management

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