FORM OF TRAINING | MATERIALS | PRICE | DURATION
---|---|---|---
Traditional | Hardcopy | 1450 EUR | 4 days
Traditional | CTAB Tablet | 1550 EUR | 4 days
Distance learning | Hardcopy | 1450 EUR | 4 days
Distance learning | CTAB Tablet | 1450 EUR | 4 days

LOCATIONS
Krakow - 5 Tatarska Street, II floor, hours: 9:00 am - 4:00 pm
Warsaw - 17 Bielska Street, hours: 9:00 am - 4:00 pm

TRAINING TERMS
2019-11-19 | 4 days | Warszawa
2020-03-09 | 4 days | Warszawa
2020-05-18 | 4 days | Kraków

TRAINING GOALS:

This 4 days course is designed to show experienced programmers how to develop device drivers for embedded Linux systems, and give them a basic understanding and familiarity with the Linux kernel.

Upon mastering this material, you will be familiar with the different kinds of device drivers used under Linux, and have an introduction to many of the appropriate APIs to be used when writing a device driver. The labs for illustrating these concepts will all be performed on ARM hardware in order to get developers familiar with cross-compiling and developing drivers for an embedded target. The included development kit (yours to keep) will be used to illustrate testing kernel drivers using TFTP and NFSroot techniques.

While we will discuss kernel internals and algorithms we will examine deeply only the functions which are normally used in device drivers. More details on things such as scheduling, memory management, etc., belong more properly in a different, kernel-focused course.

CONSPECT:

- Introduction
  - Objectives
  - Who You Are
The Linux Foundation
Linux Foundation Training
Linux Distributions
Preparing Your System
Things change in Linux
Documentation and Links
Course Registration

Preliminaries
Procedures
Kernel Versions
Kernel Sources and Use of git
Hardware
Staging Tree

How to Work in OSS Projects **
Overview on How to Contribute Properly
Study and Understand the Project DNA
Figure Out What Itch You Want to Scratch
Identify Maintainers and Their Work Flows and Methods
Get Early Input and Work in the Open
Contribute Incremental Bits, Not Large Code Dumps
Leave Your Ego at the Door: Don’t Be Thin-Skinned
Be Patient, Develop Long Term Relationships, Be Helpful

Cross-Development Toolchain
The Compiler Triplet
Built-in Linux Distribution Cross Compiler
Linaro
CodeSourcery
crosstool-ng
Buildroot
OpenEmbedded
Yocto Project
Labs

Basic Target Development Board Setup
Objectives of the Lab
Labs

Booting a Target Development Board over Ethernet
Objectives of the Lab

Labs

Kernel Configuration, Compilation, Booting
- Configuring the Kernel for the Development Board
- Labs

Device Drivers
- Types of Devices
- Mechanism vs. Policy
- Avoiding Binary Blobs
- Power Management
- How Applications Use Device Drivers
- Walking Through a System Call Accessing a Device
- Error Numbers
- printk()
- devres: Managed Device Resources
- Labs

Modules and Device Drivers
- The module_driver() Macros
- Modules and Hot Plug
- Labs

Memory Management and Allocation
- Virtual and Physical Memory
- Memory Zones
- Page Tables
- kmalloc()
- __get_free_pages()
- vmalloc()
- Slabs and Cache Allocations
- Labs

Character Devices
- Device Nodes
- Major and Minor Numbers
- Reserving Major/Minor Numbers
- Accessing the Device Node
- Registering the Device
- udev
dev_printk() and Associates
file_operations Structure
Driver Entry Points
The file and inode Structures
Miscellaneous Character Drivers
Labs

Kernel Features
Components of the Kernel
User-Space vs. Kernel-Space
What are System Calls?
Available System Calls
Scheduling Algorithms and Task Structures
Process Context
Labs

Transferring Between User and Kernel Space
Transferring Between Spaces
put(get)_user() and copy_to(from)_user()
Direct Transfer: Kernel I/O and Memory Mapping
Kernel I/O
Mapping User Pages
Memory Mapping
User-Space Functions for mmap()
Driver Entry Point for mmap()
Accessing Files from the Kernel
Labs

Platform Drivers
What are Platform Drivers?
Main Data Structures
Registering Platform Devices
An Example
Hardcoded Platform Data
The New Way: Device Trees
Labs

Device Trees
What are Device Trees?
What Device Trees Do and What They Do Not Do
- Device Tree Syntax
- Device Tree Walk Through
- Device Tree Bindings
- Device Tree support in Boot Loaders
- Using Device Tree Data in Drivers
- Coexistence and Conversion of Old Drivers
- Labs
  - Interrupts and Exceptions
    - What are Interrupts and Exceptions?
    - Exceptions
    - Asynchronous Interrupts
    - MSI
    - Enabling/Disabling Interrupts
    - What You Cannot Do at Interrupt Time
    - IRQ Data Structures
    - Installing an Interrupt Handler
    - Labs
  - Timing Measurements
    - Kinds of Timing Measurements
    - Jiffies
    - Getting the Current Time
    - Clock Sources
    - Real Time Clock
    - Programmable Interval Timer
    - Time Stamp Counter
    - HPET
    - Going Tickless
  - Kernel Timers
    - Inserting Delays
    - What are Kernel Timers?
    - Low Resolution Timer Functions
    - Low Resolution Timer Implementation
    - High Resolution Timers
    - Using High Resolution Timers
    - Labs
  - ioctls
What are ioctls?
- Driver Entry point for ioctls
- Locked and Lockless ioctls
- Defining ioctls
- Labs

Unified Device Model and sysfs
- Unified Device Model
- Basic Structures
- Real Devices
- sysfs
- kset and kobject examples
- Labs

Firmware
- What is Firmware?
- Loading Firmware
- Labs

Sleeping and Wait Queues
- What are Wait Queues?
- Going to Sleep and Waking Up
- Going to Sleep Details
- Exclusive Sleeping
- Waking Up Details
- Polling
- Labs

Interrupt Handling: Deferrable Functions and User Drivers
- Top and Bottom Halves
- Softirqs
- Tasklets
- Work Queues
- New Work Queue API
- Creating Kernel Threads
- Threaded Interrupt Handlers
- Interrupt Handling in User-Space
- Labs

Hardware I/O
- Memory Barriers
Allocating and Mapping I/O Memory
Accessing I/O Memory

Direct Memory Access (DMA)**
- What is DMA?
- DMA Directly to User
- DMA and Interrupts
- DMA Memory Constraints
- DMA Masks
- DMA API
- DMA Pools
- Scatter/Gather Mappings
- Labs

Memory Technology Devices (Flash Memory Filesystems)
- What are MTD Devices?
- NAND vs. NOR vs. eMMC
- Driver and User Modules
- Flash Filesystems
- Labs

USB Drivers
- What is USB?
- USB Topology
- Terminology
- Endpoints
- Descriptors
- USB Device Classes
- USB Support in Linux
- Registering USB Device Drivers
- Moving Data
- Example of a USB Driver
- Labs

Closing and Evaluation Survey
- Evaluation Survey

** These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.
 REQUIREMENTS:

Knowledge of basic kernel interfaces and methods such as how to write, compile, load and unload modules, use synchronization primitives, and the basics of memory allocation and management, such as is provided by LFD420 (Kernel Internals and Development). Pre-class preparation material will be provided before class.

Difficulty level

CERTIFICATE:

The participants will obtain certificates signed by The Linux Foundation.

TRAINER:

Certified The Linux Foundation Trainer.