TRAINING GOALS:

This 4 days course will teach you how to develop device drivers for Linux systems, grounded with a basic familiarity and understanding of the underlying Linux kernel. You will also learn about the different types of Linux device drivers as well as the appropriate APIs and methods through which devices interface with the kernel.

During this course you will learn:

- The different kinds of device drivers used in Linux
- The appropriate APIs through which devices (both hardware and software) interface with the kernel.
- Necessary modules and techniques for developing and debugging Linux drivers
- And more.

The information in this course will work with any major Linux distribution.

CONSPECT:

- Introduction
Objectives

Who You Are

The Linux Foundation

Linux Foundation Training

Linux Distributions

Platforms

Preparing Your System

Using and Downloading a Virtual Machine

Things change in Linux

Documentation and Links

Course Registration

Preliminaries

Procedures

Kernel Versions

Kernel Sources and Use of git

Rolling Your Own Kernel

Hardware

Staging Tree

How to Work in OSS Projects **

Overview on How to Contribute Properly

Stay Close to Mainline for Security and Quality

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

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
- \texttt{put\_user()} and \texttt{copy\_to\_user()} and \texttt{copy\_from\_user()}
- Direct Transfer: Kernel I/O and Memory Mapping
- Kernel I/O
- Mapping User Pages
- Memory Mapping
- User-Space Functions for \texttt{mmap()}
- Driver Entry Point for \texttt{mmap()}
- Accessing Files from the Kernel
- 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
- Labs

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
    - 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
- Buses and Ports
- Memory Barriers
- Registering I/O Ports
- Reading and Writing Data from I/O Registers
- Allocating and Mapping I/O Memory
- Accessing I/O Memory
- Access by User - ioperm(), iopl(), /dev/port
Labs

PCI
- What is PCI?
- PCI Device Drivers
- Locating PCI Devices
- Accessing Configuration Space
- Accessing I/O and Memory Spaces
- PCI Express
Labs

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

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
○ Network Drivers I: Basics
  ○ Network Layers and Data Encapsulation
  ○ Datalink Layer
  ○ Network Device Drivers
  ○ Loading/Unloading
  ○ Opening and Closing
  ○ Labs
○ Network Drivers II: Data Structures
  ○ net_device Structure
  ○ net_device_ops Structure
  ○ sk_buff Structure
  ○ Socket Buffer Functions
  ○ netdev_printk() and Associates
  ○ Labs
○ Network Drivers III: Transmission and Reception
  ○ Transmitting Data and Timeouts
  ○ Receiving Data
  ○ Statistics
  ○ Labs
○ Network Drivers IV: Selected Topics
  ○ Multicasting **
  ○ Changes in Link State
  ○ ioccts
  ○ NAPI and Interrupt Mitigation
  ○ NAPI Details
  ○ TSO and TOE
  ○ MII and ethtool **
○ 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

Power Management
Power Management
ACPI and APM
System Power States
Callback Functions
Labs

Block Drivers
What are Block Drivers?
Buffering
Registering a Block Driver
gendisk Structure
Request Handling
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 Linux 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.