Developing Linux Device Drivers (LFD430)

This instructor-led Linux device driver course will teach you about the different types of Linux device drivers as well as the appropriate APIs and methods through which devices interface with the kernel.

Duration: 5 Days

Prerequisites for this course

To make the most of this course you must have:

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.

Outline for this course

Introduction- Objectives
- Who You Are
- The Linux Foundation{
- Copyright and No Confidential Information
- The Linux Foundation{ Training
- Certification Programs and Digital Badging
- Linux Distributions
- Platforms
- Preparing Your System
- Using and Downloading a Virtual Machine
- Things Change in Linux and Open Source Projects
- Documentation and Links

Preliminaries- Procedures
- Kernel Versions
- Kernel Sources and Use of git
- Rolling Your Own Kernel
- Hardware
- Staging Tree
- Labs

How to Work in OSS Projects **- Overview on How to Contribute Properly
- Know Where the Code is Coming From: DCO and CLA
- 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 texttt {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
- texttt {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
- 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

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
- 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
- texttt {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
- ioctls
- 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

Kernel Architecture I- UNIX and Linux **
- Monolithic and Micro Kernels
- Object-Oriented Methods
- Main Kernel Components
- User-Space and Kernel-Space

Kernel Programming Preview- Task Structure
- Memory Allocation
- Transferring Data between User and Kernel Spaces
- Object-Oriented Inheritance - Sort Of
- Linked Lists
- Jiffies
- Labs

Modules- What are Modules?
- A Trivial Example
- Compiling Modules
- Modules vs Built-in

- Module Utilities
- Automatic Module Loading
- Module Usage Count
- Module Licensing
- Exporting Symbols
- Resolving Symbols **
- Labs

Kernel Architecture II- Processes, Threads, and Tasks
- Kernel Preemption
- Real Time Preemption Patch
- Labs

Kernel Configuration and Compilation- Installation and Layout of the Kernel Source
- Kernel Browsers
- Kernel Configuration Files
- Kernel Building and Makefiles
- initrd and initramfs
- Labs

Kernel Style and General Considerations- Coding Style
- Using Generic Kernel Routines and Methods
- Making a Kernel Patch
- sparse
- Using likely() and unlikely()
- Writing Portable Code, CPU, 32/64-bit, Endianness
- Writing for SMP
- Writing for High Memory Systems
- Power Management
- Keeping Security in Mind
- Labs

Race Conditions and Synchronization Methods- Concurrency and Synchronization Methods
- Atomic Operations
- Bit Operations
- Spinlocks
- Seqlocks
- Disabling Preemption
- Mutexes
- Semaphores
- Completion Functions
- Read-Copy-Update (RCU)
- Reference Counts
- Labs

Memory Addressing- Virtual Memory Management
- Systems With and Without MMU and the TLB
- Memory Addresses
- High and Low Memory
- Memory Zones
- Special Device Nodes
- NUMA
- Paging
- Page Tables
- page structure
- Labs

Memory Allocation- Requesting and Releasing Pages
- Buddy System
- Slabs and Cache Allocations
- Memory Pools
- kmalloc()
- vmalloc()
- Early Allocations and bootmem()
- Memory Defragmentation
- Labs