L-3 Linux Device Drivers

Duration: 5 days	
Pre-requisites	
Mandatory	***Preferable***
Course L-0 “Linux CLI and Shell Scripting” *plus*	Kernel-level experience, exposure to OS concepts
Course C-1 “Programming in C” *plus*	
Course L-1 “Linux Systems Programming” *plus*	
Course L-2 “Linux Kernel Internals”	
Below is the Outline TOC (Table Of Contents) document: it presents the (approximate) Day-wise Coverage.	

Day 1

Module 1: Linux OS : Fundamental Prerequisites

- OS Architecture
  - Processor Protection Levels
  - Monolithic Kernel

Module 2 : Handling Concurrency in the Kernel

- The Need for Atomicity
- Causes of Concurrency in the kernel
- Deadlock Prevention
  - Important Guidelines
- Concurrency in the Kernel
- Spinlocks and Mutexes
- The Semaphore Interface
- Specialized Locking
  - Atomic Operators
  - Reader-Writer Locks
  - Memory Barriers
- Debugging

Module 3 : Kernel VFS and Open Files

- Role of the kernel VFS Layer
- The Open Files Table - the files_struct structure
- The File Table - the file structure
- The File Operations pointer - the file_operations structure

Kernel VFS filesystem I/O call graph

Module 4 : Character Device Drivers – the Framework

Introduction
Block and Character Drivers
Namespaces
- Major, Minor number
- Official Device Registry
- Inode Changes

User-Space vs Kernel-Space Drivers

Day 2

Module 4 : Character Device Drivers – the Framework (contd.)

Kernel-Space Character Drivers
The kernel "mem" driver
- The VFS layer and driver registration
- Copying User <--> Kernel Memory

Writing a character device driver for Linux - the Framework
- Registering with the kernel
- The major-wide open and fop setup
- Minor-specific open

Writing our own character driver: the zero and null memory devices
Device Nodes Management

Default Behaviour of f_op methods
Enhanced zero source functionality

Userspace Entry Points
- A mention of
  - Procfs
  - Sysfs
  - Netlink
- **Debugfs**

Accessing IO Memory
- Accessing hardware registers and memory
- ioremap, ioread, iowrite, etc
- Examples
  - From kernel source tree drivers
  - *Using the "Device IO Memory Read/Write" OSS project (time allowing, can demonstrate the same on an SBC)*

Module 5 : Character Device Drivers II

Blocking I/O and Wait Queues
Kernel implementation of the wait_event_* routines
Awakening from a wait queue
Simple blocking I/O implementation: the sleepy driver

Writing Reentrant-Safe driver code
Handling Blocking and Non-blocking I/O
- Using mutexes for mutual exclusion control
- Blocking I/O in practice – how to correctly handle read() / write() operations in char device drivers

Day 3

Module 6 : Handling Hardware Interrupts

Interrupt Handling
do_IRQ()

Installing an Interrupt Handler
- Interrupt Handler Flags
- (Fairly) Recent Changes in Hardware Interrupt Handling in the Linux Kernel

Implementing an Interrupt Handler
- Handler Arguments and Return Value

Threaded Interrupts

Interrupt Control
- Disabling and Enabling Interrupts
- Disabling a Specific Interrupt Line
- Status of the Interrupt System

Deferred Functionality Mechanisms in the 2.6 Linux Kernel
- Tasklets and Bottom-Half Processing
- Softirq's
- A note on Work Queues
- Which One do I Use – A Quick Comparison

Module 7 : Kernel Mechanisms

Timers
- Using Timers

Delaying Execution
- Busy Looping
- Small Delays
- schedule_timeout()

Kernel Threads

Creating and Managing a Kernel Thread

Work Queues
- Using Work Queues
- Creating Work
- Your Work Queue Handler
- Scheduling Work
- Flushing Work
- Creating New Work Queues

Day 4

Module 8 : Linux Driver Model

The Sysfs filesystem
- Introduction
- Buses, Devices, Drivers
- Exploring

The Linux Driver Model
- Driver frameworks
- Bus Drivers
- Device Drivers – Unified Model
  - Defining
  - Registering
  - Probing and Removal hooks
- Platform Devices and Drivers

Linux and the Device Tree

Module 9 : Block Device Drivers

Introduction

Block Devices Handling
The Generic Block Layer
I/O Schedulers

Block Driver Data Structures and Methods
The Bio Structure

Sample Linux block device driver
Code walk-through and test

Module 10 : USB Device Drivers

USB Architecture
- USB Receptacles and Transceivers
- Bus Speeds
- Host Controllers

Transfer Types
Addressing

Linux-USB Subsystem
Driver Data Structures
The usb_device Structure
USB Request Blocks (URBs)
Pipes
Descriptor Structures

Day 5

Module 11 : USB Device Drivers (contd.)

Enumeration
Device Example: Telemetry Card
Table: Register Space in the Telemetry Card
Initializing and Probing
Accessing Registers
Data Transfer

USB Hardware Device : Developing the USB device driver - demonstration and code-walkthrough.

Module 12 : Linux Network Stack Implementation – Some Details *[Optional – Time Permitting]*

Basic Network Theory
The ISO 7 Layer Model
Realistic TCP Model

Network Encapsulation
Quick Note on TCP/IP Protocols
Socket Buffers (SKBs)

Linux Network Stack - The Flow of a Network Packet
Transmit Path Processing
L5 Session layer – Sockets
L4 Transport layer – TCP
L3 Network layer – IPv4
L2 Link layer – Ethernet

Receive Path Processing
L2 Link layer – Ethernet
L3 Network layer – ARP, IPv4
L4 Transport layer – TCP
L5 Session layer – Sockets

Module 13 : Network Device Drivers

Background Information
Network Encapsulation
Network Interfaces

Network device drivers
- An interface between Linux kernel and device driver
- Operation scenario

Linux Network Subsystem
- Socket Buffers (SKBs)
- SIDEBAR :: Socket Buffer (De)Allocation

Driver Data Structures
- Socket Buffers
- The Net Device Interface

Activation
Data Transfer
Misc: Watchdog / Statistics / Configuration / Bus Specific

Talking with Protocol Layers
- Receive Path
- Transmit Path
- Flow Control

Buffer Management and Concurrency Control
Device Example: Ethernet NIC
An Ethernet NIC Driver

A simple “virtual ethernet” Network Device Driver
- Code walk-through and demo

Packet capture and sniffing with Wireshark / tcpdump
Using Netfilter
Zero-Copy Techniques.

Wrap Up