FIDILITY SOLUTIONS

Inspired by Technology, Driven by Trust

Linux Device Drivers – Foundational

Course Syllabus

Course Title

Linux Device Drivers – Foundational

Course Overview

Linux is one of the most widely used Operating Systems in Embedded Systems Design for a diverse range of applications like IoT, Automotive, Healthcare, Consumer Electronics and Smart Gadgets. Linux OS by itself has a layered architecture that spans across Application, System and the Kernel space with each layer having a specific role and access permissions to the various components and interfaces of the Linux system.

Device drivers is one such vital component of the kernel that enables access and operation on the hardware devices that are connected to the system. This course provides you an in-depth understanding of Linux device drivers and its interface to the kernel space and user-space for interacting with the hardware devices. At the end of this course, you would have gained mastery over the kernel subsystems and acquired necessary skills to do efficient kernel programming primarily focusing on customization and integration of device drivers to the Linux kernel space

Course Duration

40+ hrs – 17hr Lecture, 23hr Lab Sessions

Course Eligibility

- Aspiring graduates seeking job opportunities in embedded Linux development
- Professionals interested in up-skilling themselves in Linux Device Drivers
- Embedded Linux developers who wish to become proficient in Linux device driver development

Course Highlights

- Course delivered by seasoned industry experts with more than 17 years of experience in embedded software design and development
- Course contents designed exclusively to give deep insights into kernel level programming and Linux device drivers
- Well-structured and modular program framework to ensure participants gain mastery in embedded system design and development concepts
- Online and interactive sessions with a balanced mix of theory, hands-on assignments, and mini projects

Course Pre-requisites

- Ubuntu or Linux host PC for development
- Basic Knowledge of Linux commands – command memento and vi memento
- Basic Knowledge of Embedded Linux Systems
- Basic programming skills in C and Assembly Language
- Good Know-How of working with embedded COTS boards

Course Requirements

- BeagleBone Black
- Ubuntu 20.04 LTS Host Machine
- Candidates must have a laptop/desktop with minimum configuration: - 64-bit processor - 8GB RAM - 200GB HDD space - Minimum 2Mbps internet connection

Course Contents

1. Introduction to Linux Device Drivers
2. Character Driver Model
3. Interrupts, Concurrency and Memory Management
4. Block Driver Model
5. Network Driver Model
6. Driver Debugging Techniques
7. Embedded Device Driver Frameworks

Module Contents

Module - 1

- **Module Title**
  Introduction to Linux Device Drivers

- **Module Overview**
  This module begins with a brief introduction to the Linux architecture and the various subsystems it is comprised of. We explore the Linux User space Vs Kernel space components and how a Linux device driver fits into the kernel space. We will learn LDD as a kernel module which can be developed, compiled off-the shelf from the kernel source. We will gain hands-on experience by writing our first device driver and loading it onto a running kernel. We shall also learn the different types of device drivers and their characteristic features

- **Module Topics**
  - Overview of Linux Architecture
    - A brief History of Linux OS
    - Components of Linux Systems
    - User Space Vs Kernel Space
    - Role of a Device Driver
    - Overview of Driver Stacks
  - Linux Kernel Modules
    - Categories of LKM – LDD, Filesystem Drivers, System Calls
    - Advantages of LKM
    - Kernel Modules Vs User Programs
    - Kernel Modules Vs Device Drivers
  - Types of Linux Device Drivers
    - Character Drivers
    - Block Drivers
    - Network Drivers
  - Setting up of Ubuntu / Hardware Board
  - Writing the First Driver
    - Module Information, Init, Exit, Probe, Params, printk, dmesg
    - Building, Running, Loading and Unloading of Device Driver

Module - 2

- **Module Title**
  Character Driver Model

- **Module Overview**
  In this module we shall learn the characteristic features of character drivers which is the most suitable class of driver for majority of simple hardware devices. We shall learn about major numbers, minor numbers, and device numbers and how to register and unregister drivers using these unique numbers for devices. We shall also learn the typical driver entry points which include the open, close, read and write interfaces through which user space programs get access to the device for various operations. Apart from the device entry points we shall also explore the other file operation functions like ioctl, mmap, llseek etc.

- **Module Topics**
  - Overview of Character Drivers
    - Features and examples of character drivers
    - Major and Minor Numbers
  - Driver registration

    - Device Numbers
    - Register and Unregister
  - Driver file entry points
    - Open, Close, Read, Write, Ioctl
  - Driver file structure
  - Device file operations
    - open, close, read, write, Ioctl, mmap, llseek, poll, select
  - Writing a Simple Character Device Driver

Module - 3

- **Module Title**
  Interrupts, Concurrency and Memory Management

- **Module Overview**
  In this module we continue to understand the concepts of device drivers for interrupt handling, concurrency conditions and memory management. We will begin the discussion on interrupts and understand the ways to implement and install an ISR with the kernel. We shall also discuss how the ISRs are split into Top Halves and Bottom Halves for processing of interrupts in kernel context. We dwell deep into understanding various concurrency and race conditions that can occur in a multiprocessing environment due to simultaneous access of resources. Finally, we shall learn about memory management techniques in Linux systems. We shall understand the intricacies of virtual and physical memory mapping and allocation concepts of the Linux OS

- **Module Topics**
  - Understanding Linux Interrupt Handlers
    - Installing and Implementing an ISRTop Halves and Bottom Halves
    - Tasklets , Workqueue, Softirq and Threaded IRQ
    - Hotplug and Udev events
  - Concurrency and Race Conditions
    - Semaphores and Mutex
    - Completions
    - Spinlocks
    - Kernel Timers
  - Introduction to Linux Memory Management
    - Linux Memory Model – Virtual Vs Physical Address
    - Address Translation & Page Tables
    - Page Allocator
    - Mpool & Mmap
    - Port Mapped and Memory Mapped IO

- DMA

Module - 4

- **Module Title**
  Block Driver model

- **Module Overview**
  This module provides an in-depth understanding of the Linux block drivers which is the standard framework employed by most of the memory and storage device drivers. We begin with a discussion on the Block driver architecture and various device operations permitted to use with block devices. We shall gain some hands-on experience by writing a simple block driver and analyse its working by loading and registering it with a running kernel. Finally, we shall explore the MMC/SD framework available in kernel sources and understand its working along with typical storage devices like SD cards

- **Module Topics**
  - Overview of Architecture of a Block Driver
  - Linux Block I/O Layer
  - Linux Block Device Operations
  - Registering and Unregistering the Block driver
  - Request Processing
  - Writing a Simple Block Driver
  - Overview of MMC/SD framework

Module – 5

- **Module Title**
  Network Driver Model

- **Module Overview**
  This module focuses on Linux networking system and understanding the network driver stack and the kernel interfaces while working with ethernet controllers for transmission and reception of network packets. We shall discuss some of the important data structures of network devices and its configuration parameters. We shall also discuss in detail the concepts for interfacing with the physical layer and initialization of the MDIO and MAC. We shall then discuss the driver support for handling link state changes in the ethernet link. Finally, we will explore some of the standard tools available in the Linux systems for configuration, statistics and throughput measurement of ethernet interfaces

- **Module Topics**

- Overview of Network Driver Architecture
- Understanding net_device and sk_buf
- Packet Transmission and Reception
- Communicating with PHY
- MDIO Initialization
- MAC address Resolution
- Handling Link State Changes
- Ethtool and iperf for statistics information

Module - 6

- **Module Title**
  Driver Debugging Techniques

- **Module Overview**
  In this module we introduce the various tools, techniques, and tips for debugging code at kernel level. Kernel code brings about its own challenges and limitations for debug support and we shall learn the various facilities kernel programmers have at their disposal for being able to debug and fix buggy codes in kernel space. We begin with understanding and observing the debug mechanisms using printing and logging features available in the kernel system. We shall next proceed with understanding the various system interfaces available in the kernel space for query and response which provides a snapshot of the running system. We shall learn to debug system faults by analysing the kernel panic and OOPS messages. Finally, we will have some hands-on experience in using the GDB and STRACE tools for debugging and fixing drivers and kernel issues

- **Module Topics**
  - Overview of Debug support in Kernel
  - Debug using Printing
    - printk and its priorities
  - Debug using logging
    - Klogd, syslogd, system map
  - Debug using Querying
    - procfs, debugfs, sysfs
  - Debug using Watching
    - Using Strace
  - Debugging System Faults
    - Kernel Panic, OOPS Messages
  - Debug Tools
    - Using gdb, kgdb

Module - 7

- **Module Title**
  Embedded Device Driver Frameworks

- **Module Overview**

  This module introduces some of the standard embedded device driver frameworks present in Linux systems for interfacing the hardware devices with the Linux system. We shall begin with the GPIO APIs available in Linux kernel for export, access, read and write of GPIO values. Next, we shall understand the I2C bus driver for communicating with low-speed i2c devices like memory and RTC. Next, we explore the SPI driver subsystem comprising of core, controller, and protocol driver subsystems in Linux. We shall also dwell deep into understanding the TTY framework for communication with typical serial port devices like UART, Modem etc. Finally, we will end this module by going deep into the USB driver framework. We will understand the typical USB device classes and gadget drivers available in Linux kernel for enumeration and communication with typical USB device types

- **Module Topics**
  - Overview of driver frameworks for interfacing with Hardware
  - Overview of GPIO framework
  - Overview of I2C framework
  - Overview of SPI framework
  - Overview of TTY framework
  - Overview of USB framework