Linux Device Drivers

Covers kernel version 3.x to 6.x

Customized for InfobellIT, Bangalore

Authored and Compiled by: Kaiwan N Billimoria
<kaiwan@kaiwantech.com>

Level:

BASIC INTERMEDIATE **[ADVANCED]**

Duration: 4 days

Course Code: L3
Courseware version 24.07

Contact Information

Email : kaiwan@kaiwantech.com
Reach us online at http://kaiwantech.in/

Brief Description

This training is targeted primarily at software professionals - tech leads, system programmers / developers, maintainers and testers - who would like to delve quite deep into the design and implementation of device drivers on the Linux operating system. This may be in order to work on a project involving this directly or indirectly (for example, for those on an application development project on the Linux platform, this training would give them tremendous insight into optimization and code tweaking based on internal factors). Opensource (and closed-source commercial) contributors / developers (or to-be contributors) would find this training very useful as well.

This training is based on the latest 3.x / 4.x / 5.x Linux kernel. It starts with a quick and comprehensive look at the overall Linux OS architecture, monolithic kernel approach; essential kernel prerequisites to understand drivers and their kernel framework. It then moves into deep technical discussion of various kernel device driver topics: the data structures required to understand, and the actual framework of a Linux character device driver is covered in depth. Sample driver code includes a simple memory driver as well as I/O port drivers.. (Fairly) advanced driver topics include blocking I/O implementation (wait queues), reentrant-safe drivers, Linux's handling of and a driver author's view of hardware interrupt programming and concurrency control (SMP safety).

Throughout, professional / industry best practices are taught and encouraged.

Level:

BASIC [**INTERMEDIATE**] **[ADVANCED]**

Prerequisites

It is very important that the prerequisite(s) marked as Mandatory below be met by all participants intending to attend this training, either by having successfully attended a training program (mentioned below), or having the equivalent knowledge / skill sets.

Mandatory

- Good working knowledge of and experience programming using the 'C' language.
- Successfully attended the "LINUX Fundamentals for Software Developers" training program -or-have the equivalent knowledge / skill set
- Successfully attended the "LINUX System Programming" training program -or-have the equivalent knowledge / skill set. Implies knowledge / skill sets of: POSIX library and system call API set, process management, IPC mechanisms, etc
- Successfully attended the **"LINUX Kernel Internals"** training program -or-have the equivalent knowledge / skill set. Implies knowledge / skill sets of, at a *minimum*: Linux OS architecture, basics of writing a Loadable Kernel Module (LKM), clear VM concepts.

Optional / Advantageous

- Extensive user-space development experience on a POSIX platform
- Experience in working in kernel-space certainly helps.

Daywise Coverage

Kindly note: in case the participant is continuing on this training session after having attended the "Linux Kernel Internals" sessions, any overlap in topics will be eliminated.

Day 1

Module 1 : VFS – An Introduction, focus on key data structures

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.

Module 2 : Device Drivers – an Introduction

- Block and Character Drivers
- Namespaces
  - Major, Minor number
  - Official Device Registry
    - Viewing it within the official Linux kernel doc
  - Inode Changes
- User-Space vs Kernel-Space Drivers.

Module 3 : The Linux misc character Device Driver Framework

Kernel-Space Character Drivers
- The kernel "mem" driver

Writing a Linux char driver by leveraging the 'misc' driver framework
- Framework
- Code implementation

Copying Memory between User and Kernel spaces
- Available kernel APIs / macros.

Writing our own first misc character driver: implementing the zero and null memory devices!

Default Behaviour of f_op methods
Enhanced zero source functionality

Lab Assignment :
Write and test the cz_enh device driver on your Linux (or VM instance).

Day 2

Module 4 : Interfacing with Userspace

Interfacing methods

Interfacing via sysfs

Model + APIs
Demo

Using debugfs programatically
Introduction
Debugfs prerequisites
The debugfs API (ABI)
Full example walk-through.

Using the ioctl()
A mention of
Procfs
Netlink sockets.

Lab Assignment 3 :: *Re-implement the 'misc' char driver, this time passing information via a debugfs pseudo-file (/sys/kernel/debugfs/disp_task/disp_task). When a thread PID is written to it, it "replies" with some task structure details pertaining to that task.*

Module 5 : Working with Hardware IO Memory

Accessing hardware registers and memory
Port IO vs Memory-mapped IO
PIO – APIs, examples
MMIO – APIs, examples
[devm_]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)*.

Day 3

Module 6 : [Non]Blocking I/O and Wait Queues

Motivation – why block
Putting a task to sleep
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.

Module 7 : Hardware Interrupts and Writing IRQ Handlers

Interrupt Handling
do_IRQ()
Low-Level (ARM, x86)
ARM processors – exceptions and modes

- Installing an Interrupt Handler
  - Interrupt Handler Flags
  - Changes in Hardware Interrupt Handling in the Linux Kernel
    - Threaded Interrupts
      - Motivation
      - How it’s done

  - Implementing an Interrupt Handler
  - Handler Arguments and Return Value
- Summary of key points

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

- Tasklets and Bottom-Half Processing
  - Softirq's
  - Using the tasklet
  - A note on Work Queues
  - Which One do I Use – A Quick Comparison.
- Task prioritization on Linux.

Module 8 : Kernel Mechanisms – delays, timers, kthreads, workqueues

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

Day 4

- Timers
  - Using kernel timers
- Kernel Threads
- Work Queues
  - Work Queue Handler
  - Scheduling and Flushing Work.

Module 9 : Modern Linux Driver Model (LDM)

- 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.

Module 10 : Platform Drivers and the Device Tree (DT)

Platform Devices and Drivers

What is a Platform Device
Platform devices under sysfs
Platform Driver
Device Enumeration
Example:
SMSC911x ethernet
Demo – simple platform device + driver.

Device Tree

What is the Device Tree
DT syntax elements
DT – typical properties
Usage on (ARM) Linux, Examples NEW
Demo- a simple platform device defined in the DT with driver binding

Case Study NEW

Understanding I2C basics
Procedures
i2c packages, testing with i2cdetect, editing the DTS, generating the new DTB, test it...
I2C-based temperature+humidity sensor chip kernel driver
The datasheet – reading the key sections
Writing the I2C driver
Code-level walkthrough
Installing and testing it.