INTUITIVE EMBEDDED TECH

Interactive Tech Education

www.intuitiveembedded.com

Call: 9900721303 / 9892616426 | Email: info@intuitiveembedded.com

Complete Course in Linux Device Driver Development

Total Duration for Complete Course:-
60 Hours (Only On SATURDAY and SUNDAY)

NOTE: Practice will be done on Linux (Ubuntu)

Why 'Linux Device Driver':-
Many Linux professionals would like to write device drivers in Linux, but don't know how to learn and understand the essentials of writing a driver. There are thousands of device drivers in Linux kernel and are normally characterized as Character drivers, Block drivers, Network drivers and Bus drivers. Furthermore, these bus device drivers can be of various types depending on the buses like USB drivers, PCI drivers, HDMI drivers, SPI drivers, I2C drivers, UART drivers and a lot more.

So, how does one master so many device drivers on Linux? The ideal approach is to learn one device driver at a time. Take it as a project on Linux and complete that driver project before moving on to another driver project. Our Linux device driver training course helps people learn design and develop Character device driver, on a standard desktop PC architecture (on an x86/x8-64 hardware platform). Every participant will be writing substantial code from scratch and complete that as a project in the training session.

This intensive training course transforms an IT-Professional or a Student into a Linux Device Driver & Kernel Developer. The participant will develop a deep understanding of Linux device driver subsystem, how it interfaces with Linux kernel as well as various devices; Participant will also learn other kernel subsystems and skills necessary to do efficient programming in kernel mode in Linux.

Course Outline

Duration: (60 Hours)

1. Introduction to Linux Kernel

1. What is Kernel?
2. Linux Kernel Architecture
3. Linux System Architecture
4. Basic Kernel Services
5. Source Code Browsing Tools
    1. cscope
    2. ctag
6. Linux Kernel/source Code browsing using CScope

2. Introduction to device drivers

1. Class of devices and modules
2. Character devices
3. Block devices
4. Network device drivers
5. Pseudo device drivers
6. Role of Device driver

3. Loadable Kernel Modules

1. Simple loadable kernel module
2. Compiling and validating the kernel module
3. Kernel modules versus application programs
4. User space versus kernel space
5. Concurrency in kernel space
6. Current process and current pointer
7. Loading and unloading of modules
8. Initializing and shutting down the modules
9. Error handling during module loading
11. Module parameters
12. Developing modules with more than one source file
13. Exporting symbols
14. Kernel symbol table
15. Introduction to the /sys file system
16. Introduction to user space device drivers
17. Namespace

Lab Exercises:-

Ex1. Simple loadable module
Ex2. Example to demonstrate __init, __initdata, __exit
Ex3. Simple loadable module with more than one source file
Ex4. 'Current' pointer
Ex5. Exporting symbols and using exported symbols
Ex6. Module parameters using int and char *
Ex7. Module parameters using array of int.
Ex8. Example to demonstrate the usage of /sys file system
Ex9. String parameters in /sys file system

4. Character Drivers

1. Character driver introduction
2. Major number & minor number
    1. Why Major No. and Minor No.?
    2. Reserved Major Numbers
3. Internal representation of device numbers
4. Device file creation and operations
5. File pointer
6. inode pointer
7. Character driver registration, old style
8. Allocating and freeing device numbers
9. Dynamically allocating major numbers
10. Character driver registration, new style
11. File operations in detail
12. File Structure
13. Driver-User Data Transfer

14. Driver-Kernel Communication
15. Driver-Device Communication

Lab Exercises:-

Ex10. Simple character driver using old style
Ex11. Allocating major numbers using new style
Ex12. Simple character driver using new style

5. Advanced character driver operations

1. Support for more than one driver instance
2. private_data inside the driver

Lab Exercises:-

Ex13. Character driver for more than one instance of the device
Ex14. Character driver using 'private data'

6. Blocking IO using wait queue heads Belongs

1. wait_queue_head

Lab Exercises:-

Ex15. Character driver using wait_queue_head

7. Synchronization using Semaphore and Mutex

1. Synchronization inside the kernel using semaphores
2. Client server application using kernel mode driver

Lab Exercises:-

Ex16. Character driver using semaphores
Ex17. Client server application demonstration using character driver

8. Introduction to /proc file system

1. Introduction to the /proc file system
2. Creating and deleting /proc entries
3. Simple driver to demonstrate the /proc file system usage
4. Atomic variables inside the kernel

Lab Exercises:-

Ex18. Simple module for creating /proc entries
Ex19. Character driver using /proc entries
Ex20. Character driver using atomic variables

9. Time and delay in kernel

1. Jiffies
2. Different techniques to access time inside the kernel
3. timespec and timeval
4. cpu_relax (), schedule(), schedule_timeout() for delay operations
5. Short delays inside the kernel

Lab Exercises:-

Ex21. Kernel module using the following time and delay mechanisms:-

1. Jiffies, jiffies_64
2. time_before, time_after
3. jiffies_to_timespec, jiffies to timeval
4. cpu_relax, schedule,
5. wait_queue_head,
6. set_current_state, schedule timeout

10. Kernel timers, Spin Locks and Tasklets

1. Introduction to kernel timers
2. Demonstration of kernel timers using character driver
3. Spin locks for synchronization
4. Introduction to tasklets
5. Demonstration of using tasklets character driver

Lab Exercises:-

Ex22. Character driver using kernel timers
Ex23. Character driver using kernel timers and tasklets
Ex24. Character driver using kernel timers, tasklets and spin lock for synchronization

11. Kernel work queue and deferred work queue

1. Introduction to work queues
2. Kernel work queue
3. Driver specific work queue
4. Delayed work queue
5. Demonstration of workqueues using /proc file system
6. Demonstration of workqueues using character driver

Lab Exercises:-

Ex25. Demonstration of workqueues using /proc file system Ex26. Character driver using workqueues
Ex27. Character driver using kernel timers, workqueues and spin lock for synchronization

12. Kernel linked list

1. Introduction to kernel linked list
2. Character driver example using kernel linked list

Lab Exercises:-

Ex28. Character driver using kernel linked list

13. Advanced character driver operations

1. Exclusive open and blocking open
2. lseek () restrictions in character drivers

Lab Exercises:-

Ex29. Character driver using exclusive open
Ex30. Character driver with blocking open
Ex31. Restricting lseek () in character drivers

14. Interrupt Handling

1. Introduction to hardware interrupts
2. Linux interrupt handling support
3. Auto detecting IRQ numbers
4. Implementing interrupt handlers
5. Demonstration of interrupt handlers using character driver
6. Disabling interrupts
7. Demonstration of interrupts using character drivers
8. Top half and bottom half

Lab Exercises:-

Ex32. Simple module using interrupt handlers
Ex33. Character driver using interrupt handler, tasklets and spin locks
NOTE: - if possible try to show: - Implementation of Switch Interrupt (on raspberry Pi)

15. Hardware communication using port IO and memory mapped IO

1. Introduction to hardware communication
2. IO port access
3. IO memory access
3. Requesting for IO port and communicating with IO port devices
4. Requesting for IO memory and communicating with IO memory devices
5. Demonstration of parallel port driver

Lab Exercises:-

Ex34. Parallel port driver for 'running leds'

16. Advanced character driver operations

1. Introduction to IOCTL's
2. Demonstration of parallel port example with IOCTL's

Lab Exercises:-

Ex35. IOCTL example for the following functionality:
    1. Get integer by value
    2. Get integer by pointer
    3. Set integer by pointer
    4. Get structure by pointer
    5. Set structure by pointer
Ex36. IOCTL based parallel port driver
Ex37. Running LED application using IOCTL parallel port driver
Ex38. 16x2 character display using IOCTL parallel port driver

17. Adding a Driver to the Kernel Tree

1. Kernel layout for drivers
2. Modifying the Makefile
3. Adding it to configuration options - the Kconfig file

THANK YOU