info@embisyslabs.com +91-8884867053

EMBISYS LABS
key to success
ISO 9001:2015 Certified

100% Knowledge Guarantee
100% Genuine Placement Assistance

LINUX DEVICE DRIVER AND KERNEL PROGRAMMING

Download Linux Device Driver Syllabus

PREREQUISITE : Advanced C And Linux Systems Programming

Embisyslabs is the Linux Kernel Device Drivers Training Institute conducts best Courses on Linux Kernel Programming, Character, Block, USB Gadget, PCI, Network, I2C, SPI Driver, Device tree, Linux internals, Porting linux kernel, Embedded Linux, Embedded Firmware, RTOS, Beaglebone, Raspberry-PI, ARM courses, ARM 7, ARM9 For Freshers and Working Professionals on Weekend as well Weekdays in Bangalore Bengaluru India.

CH1: AN INTRO. TO DEVICE DRIVERS

Role of the Device Drivers
Splitting the kernel
Classes of devices and modules
Kernel Architecture or Model

CH2: BUILDING AND RUNNING MODULES

Types of Modules in the kernel
Writing Your first kernel module
Module Related Commands
Kernel Module vs Applications
Compiling Modules
Loading and Unloading Modules
Module Parameters
Modules and Exporting Symbols
Hands-On Assignments
Lab1: Simple Hello Linux Kernel Module.
Lab2: Write a module that can take an integer parameter when it is loaded, It should have a default value when none is specified.
Lab3: Write a pair of modules where the second one calls a function in the first one
CH3: CHAR DEVICE DRIVERS
Major and Minor Numbers
The Internal Representation of Device Numbers
Allocating and Freeing Device Numbers
File Operations Data structure
Driver methods and Function Pointers
Char Device Registration
The Cdev Structure
The inode Structure
The file Structure
Manual Creation of Device Files
Automatic Creation of Device Files
Hands-On Assignments
Lab1: Print the major and minor numbers when Registering by Static or Dynamic method.
Lab2: Implement a open,write,read and close entry point.
Lab3: Print the major and minor numbers when the device is Opened and keep track of the number of times it has been opened since loading, and print out the counter every time the device is opened.
Lab4: Modify the previous driver so that each opening of the device allocates its own data area, which is freed upon release. Thus data will not be persistent across multiple opens.
Lab5 : Implement a lseek entry point and Keeping track of file position.

Lab6: Dynamical Node Creation,Adapt the previous dynamic registration driver to use udev to create the device node on the fly.

CH4: MEMORY ALLOCATION TECHNIQUE

The Real Story of kmalloc
The Flags Argument
__get_free_pages
Memory zones
vmalloc and Friends
Memory caches

Hands-On Assignments

Lab1:Testing Maximum Memory Allocation ,using kmalloc()
Lab2:Testing Maximum Memory Allocation ,using __get_free_pages().
Lab3: Testing Memory Allocation,using vmalloc().
Lab4 :Memory cachesExtend your chararcter driver to allocate the driver's internal buffer by using your own memory cache.Make sure you free any slabs you create.

CH5: ADVANCED CHAR DRIVER OPERATIONS

Inpout/Output Control (ioctl)
User space, the ioctl system call
The ioctl driver method
Choosing the ioctl Commands
Using the ioctl Argument

Hands-On Assignments

Lab1 : Implement a ioctl entry point along with read and write entry point.
Lab2 : Implement read and write entry point using ioctl command.
Lab3 : Write a character driver that has three ioctl commands: a)Set the process ID to which signals should be sent. b)Set the signal which should be sent. c)Send the signal.

CH6: CONCURRENCY AND RACE CONDITION

Concurrency and its Managements
Semaphores and Mutexes
Linux Semaphore Implementation
Introduction to the Semaphore API
Spinlocks Implementation
Introduction to the Spinlock API

Spinlocks and Atomic Context
Hands-On Assignments
Lab1: Mutex Contention -Write three simple modules where the second and third one use a variable exported from the first one.The second (third) module should attempt to lock the mutex and if it is locked, either fail to load or hang until the mutex is available.
Lab2: Sempahore Contention -Now do the same thing using semaphores .
CH7: INTERRUPT AND INTERRUPT HANDLING
The Definition and Role of Interrupt
Installing an Interrupt Handler
The /proc Interface
Implementing a Handler
Handler Arguments and Return Value
Installing a Shared Handler Hands-On Assignments
Hands-On Assignments
Lab1: Write a module that shares its IRQ with your network card. You can generate some network interrupts either by browsing or pinging.
Lab2: Extend the previous solution to construct a character driver that shares every possible interrupt with already installed handlers.
Lab3: Mutex Unlocking from an Interrupt. Modify the simple interrupt sharing lab to have a mutex taken out and then released in the interrupt handler.
CH8: TIME, DELAY AND DEFERRED WORK
Top and Bottom Halves
Tasklets and Workqueues Mechanisms
Measuring Time Lapses
Using the jiffies Counter
The Timer API
Hands-On Assignments
Lab1: Program based on Kernel Timer API
Lab2: Program based on Jiffies and HZ
Lab3: Program based on Taklet API
Lab4: Prgram based on Workqueue API
Lab5: Write a driver that puts launches a kernel timer whenever a write to the device takes place. Pass some data to the driver and have it print out. Have it print out the current->pid field when the timer function is scheduled, and then again when the function is executed.
Lab6: Write a module that launches a periodic kernel timer function; i.e., it should re-install itself.
CH9: FUNDAMENTALS OF BLOCK DEVICE DRIVER

Block drivers Definitions.
Block drivers Registration.
Block device operations.
Linux Block I/O Layer
I/O Schedulers
Block Driver Data Structures and Methods
How to handle block devices
Hands-On Assignments
Lab1: Registering and unregistering a simple Block Driver to get Major number.
CH10: IMPLEMENTATION OF RAMDISK DEVIVE DRIVER
RAMDISK-based block device driver
Using the RAMDISK block device
Driver registration
Obtaining a gendisk object
Implement the driver's methods
Request Queue & Handle the request queue
Hands-On Assignments
Lab1: Write a simple Block driver program to read (and/or write) from the node, using the standard I/O functions (open(), read(), write(), close() .After loading the module with insmod use this program to access the node.
Lab2: Mountable Read/Write Block Driver, Extend the previous exercise in order to put or create an ext3 or ex4 file system on your Block device.
Lab3: Write a program to implement a ram disk device and make it into many partition like systems Hard disk and perform read() , write() operation through block driver vertical.
CH11: UNDERSTANDING PARTITIONING OF BLOCK DEVICES
Partitioning a Block Device
Sector,Cylinder and Head
Structure of a generic MBR
Partition Table
The Bootstrap Code Area/Bootloader
MBR – Partition Table Entries
Boot Record Signature/Magic Number
Creating a RAM Block Device

Hands-On Assignments

Lab1: Write a program to implement a ram disk device and make it into many partition like systems Hard disk and perform read() , write() operation through block driver vertical.

CH12: UNDERSTANDING USB DEVICE DRIVER

USB Device Basics
USB Driver types
Defferents types of data transfers
USB and Sysfs
USB Request Block
Writing a USB Driver
Registering a USB Driver

Hands-On Assignments

Lab1:Installing a and writing a simple USB device driver.The driver should register itself with the USB subsystem upon loading and unregister upon unloading.

Lab2: Write a USB device driver to print out information abot configuration , interfaces and endpoint for a registered usb device.

CH13: USB GADGET DEVICE DRIVER ON BEAGLEBONE

Linux USB Gadget & Host Drivers
USB Gadget Driver Mechanism
USB Host Driver Mechanism
USB Core & Hot Plug n Play
USB Gadget Transfer Functions
Integration with a Vertical
Types of USB Device Drivers
Register a composite driver
Structure of usb_composite_driver
Structure of struct usb_function

Hands-On Assignment

Lab1:Beaglebone Black as standard USB Gadget Devices
Lab2:First take at a USB Gadget Driver
Lab3:Getting down to the hardware of BBB
Lab4:Creating Interface for USB Gadget Driver
Lab5:Creating Endpoint for USB Gadget Driver

Lab6:LoopBack USB Gadget Device Driver
Lab7:Creating Multiple Interface for USB Gadget Driver
Lab8:Controlling using custom USB Host Driver & App
Lab9:Controlling BBB GPIO LED through USB Drivers

CH14: PCI DEVICE DRIVER AND ITS ROLE

Understanding the x86 processor bus: PCI
PCI Core & Programming the PCI
Finding & Interacting with a PCI Device
PCI Bus, Device and Function numbers
Registering & Finding a PCI device
Mapping & Accessing the PCI device regions
Accessing the Configuration Space
Accessing the I/O and Memory Regions
Enabling the PCI Device

Hands-On Assignments

Lab1:Registering the driver with the PCI subsystem.
Lab2: Write a module that scans your PCI devices, and gathers information about them.For each found device, read some information from its configuration register.Fields you may wish to obtain could include: PCI_VENDOR_ID, PCI_DEVICE_ID, PCI_REVISION_ID,PCI_INTERRUPT_LINE,PCI_LATENCY_TIMER,PCI_COMMAND.
Lab3: Write a Character based PCI driver to find Information about IRQ Line,Memory region,I/O region,configuration region,prefetchable and non-prefetchable region in BAR.

CH15: MEMORY MAPPING AND DMA

What memory is DMA'able
DMA addressing limitations
Types of DMA mappings
DMA Direction

Hands-On Assignments

Lab1: Write a module that allocates and maps a suitable DMA buffer, and obtains the bus address handle. Do this in three ways: (a)Using dma_alloc_coherent(), (b)Using dma_map_single(), (c)Using a DMA Pool.

CH16: BASIC NETWORK DEVICE DRIVER

Registering the Network Driver
Buffer Management with skbuffs

Packet Transmission & Reception
Reception using interrupt and poll
Start the network interface's transmit Queue
Other network operations including statistics

Hands-On Assignments

Lab1: Building a Transmitting Network Driver, Module to include a transmission function.
Lab2: Adding Reception, Extend your transmitting device driver to include a reception function.

CH17: ADVANVED NETWORK DEVICE DRIVER

Registering with the Linux low level bus interface subsystem
Allocating interface descriptor block (net_device)
Device specific structure and initializing media specific fields
Getting device specific structure object pointer
Enabling Network interface card
Getting the Device resources (MMIO and PMIO)
Getting device MAC address
Initialization of device methods in the net_device
Registering net_device object with the kernel
Registering the interrupt handler (ISR)
Allocating Rxring and Txring
Initializing the hardware (network interface card)

Hands-On Assignments

Lab3: Writing the PCI based Network Driver for NIC(Network Interface Card). Programming the Network Device Registers, Implementing the Transmission & Reception with the actual device(NIC) and Setting up the network across computers.

Why Training in Embisyslabs

Flexible and Convenient time Slots for Classes.
Experience and co-operative Trainers
Maximum 6 to 8 Participants in one Batch.
Indivisual Attention to each Participant.
High Quality practical/application Oriented Training Repeatation classes will be conducted as required.

Training and Practicals Process

Classes 5-Days a week for Weekdays Batch

Theory(1 1/2 -2 hrs.) and practical (2hrs.)
OR
Classes 2-Days for a Weekend Batch(Sat & Sun)
Theory(2 1/2 -3 hrs) and practical (3hrs.)

info@embisyslabs.com
+91-88848 67053

Embisys Labs Development,Training,Consultancy & Support www.embisyslabs.com

CONTACT ADDRESS

VK heights, 3rd floor,
26th main, 9th block
Jayanagar, Opp to
Raggiguda Temple
Arch Road.
Land Mark: Near
Central Mall .
Contact No. +91-
88848 67053
Bangalore - 560069