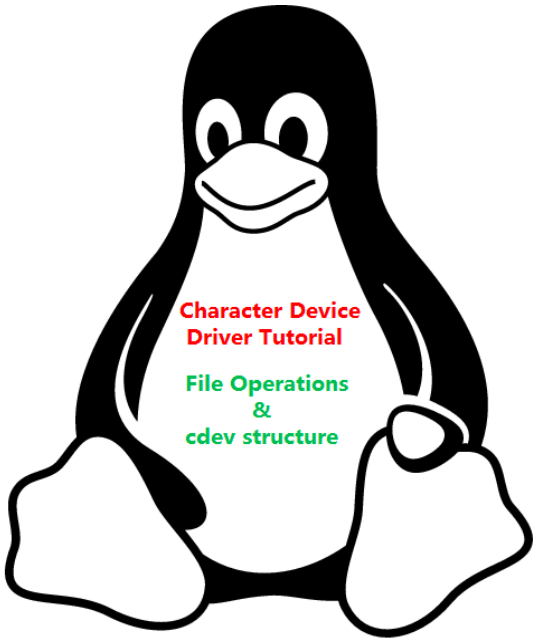
LINUX DEVICE DRIVER AND KERNEL PROGRAMMING

PREREQUISITE : C and Linux Systems Programming

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

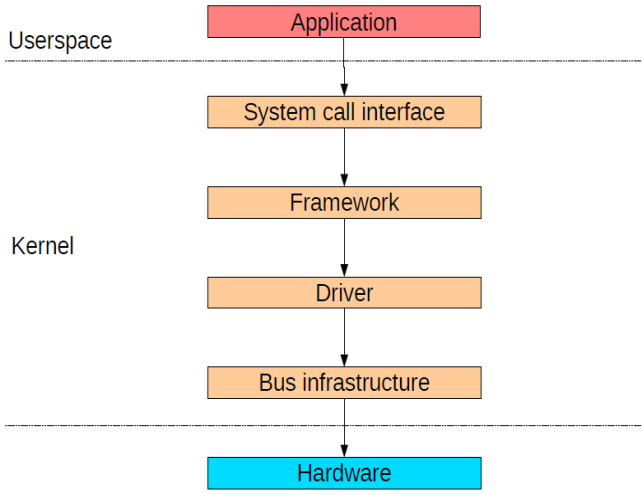
Lab1: Simple Hello Linux Kernel Module.

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: CHARACTER 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.

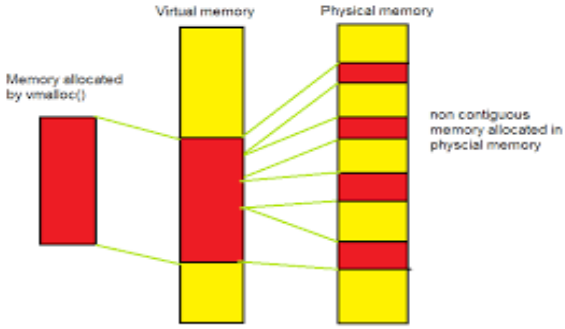
Lab4: Modify the previous driver so that each opening of the device allocates its own data area, which is freed upon release. Thus data will not be persistent across multiple opens.

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

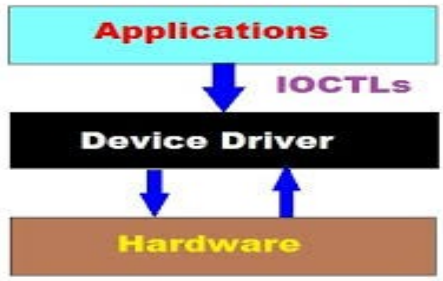


Hands-On Assignments

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

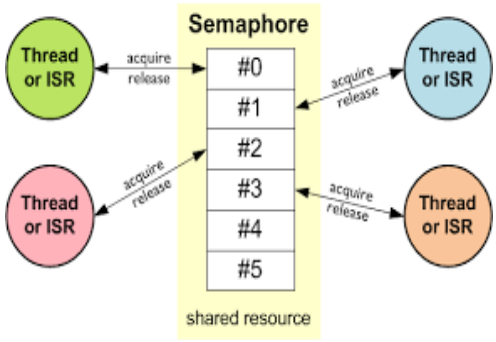


Hands-On Assignments

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



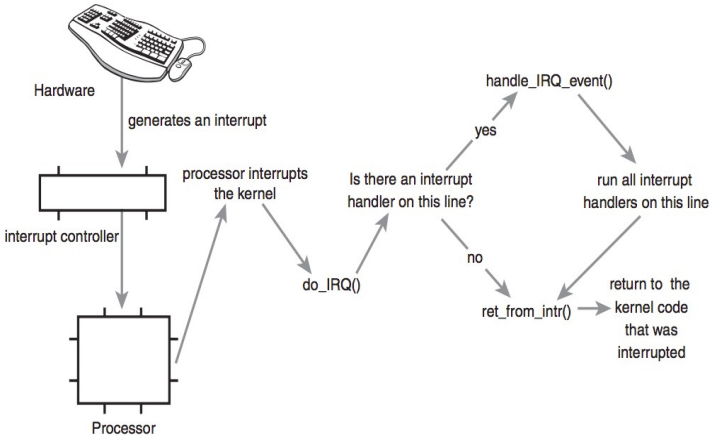
Hands-On Assignments

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

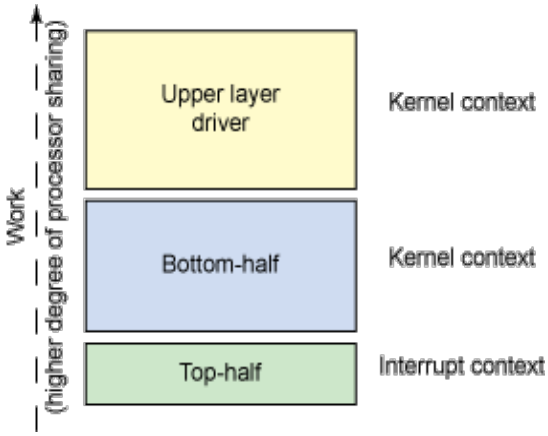


Hands-On Assignments

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.

Why Training in Embisyslabs	Training and Practicals Process
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.	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(40 %) and Practical (60 %)

Total fees = Rs. 6,000 + 18% GST = Rs. 7,080
Total duration = 2 Days 9.30 AM to 5.30 PM(Sat. & Sun.)

Embisyslabs @Bangalore
info@embisyslabs.com
+91-88848 67053