

## Embedded Linux and Device Drivers Programming Training Workshop

**Course Duration :** Three Full Days

### **Course Outline**

#### **Embedded Linux System Development**

- **Introduction to Embedded Linux ( Obisense )**
  - What is Embedded System?
  - Anatomy of an Embedded System
  - Why Linux?
  - Is Linux Real-Time Enough?
  - The Status of Embedded Linux Today
  - Which Embedded Linux Distribution to Choose?
  - Embedded Linux Architecture
  
- **Basic requirements for Embedded Linux Product Development**
  - The four basic elements: toolchain, bootloader, kernel, root file system
  - Configuration, compilation and cross-compiling
  
- **Building Development Environment ( Obisense )**
  - Toolchain installation
  
  - Cross compilation using Toolchain
  
- **Target & Host Setup**
- 
- **Setting Up Networking Services**
  - Network Settings
    - Static and Dynamic IP addresses
    - Subnet mask
  - Setting up services
    - TFTP

- DHCP
  - HTTP
  - FTP etc
  - SNMP
  - Telnet
  - SSHD
  - loading files using TFTP, HTTP, FTP etc.
- 
- **Bootloader commands and usage**
    - Getting familiar with bootloader commands
    - Downloading kernel images and RootFS on Target board.
- 
- **Loading RootFS in Platform by various techniques -**
    - Configuring TFTP and downloading kernel image over TFTP.
    - Configuring NFS and using rootfs over NFS.
    - Using SD-Card for rootfs.
    - Using USB for rootfs.
- 
- **Building an Embedded File System from Scratch -**
    - Creating a minimal root file-system using Busybox
    - Creating a RAM disk image
- 
- **Building Your Own Embedded Linux Distribution -**
    - Buildroot
    - Scratchbox
    - OpenEmbedded
    - Crosstool
    - Angstrom – narcissus
- 
- **Kernel Configuration and Compilation**
    - Kernel Building System
    - Patching the Kernel

- Kernel Configuration for Embedded Systems Settings (Porting)
  - Cross-compiling the Linux Kernel
  - **Booting Linux**
    - The Linux boot sequence
    - boot-loaders : U-boot
    - System Initialization Scripts
- 
1. **Bring up X11 (X-Windows) on Embedded Platform (GUI)**
  2. **Loading various RootFS (Distributions) in platform**
    - Angstrom
    - Ubuntu
    - Fedora etc. RootFS

### Linux Device Driver Programming

- **An introduction to device drivers**
  - User space vs Kernel space
  - Kernel Architecture or Model
  
  - Splitting the kernel
  
  - Kernel modules
  
  - Kernel Module vs Applications
  
  - Role of the Device Drivers
  
  - Classes of devices and modules
  
- **Kernel Module Programming Basics**
  - Modules Defined
  
  - Data Type in the Kernel
  
  - Version dependency
  
  - Building and Running Modules
  
  - Types of Modules in the kernel

- Writing Your first kernel module
  - Module Related Commands
  - Statically linked vs Dynamically linked drivers/modules
  - The kernel symbol table
  - Exporting symbols from modules
  - Module Parameters
  - Lab exercises for above mentioned topics
- 
- **Kernel Debugging Techniques**
    - Kernel Debugging: dmesg, printk
    - Lab exercises for above mentioned topics
- 
- **Accessing Hardware Mechanisms**
    - System Memory
    - Device Memory
    - I/O Ports
    - I/O ports vs. memory mapping
    - Allocating and mapping I/O space
    - Functions for reading and writing I/O ports
    - Side effects and compiler optimization
    - I/O APIs
    - Driver example
    - Barriers
    - Accessing hardware from User Space
- 
- **The proc file system programming**
    - Using /proc
    - Creating proc file system entries
    - Registration
    - Reading from /proc
    - Writing to /proc
  - Lab exercises

- **Communicating with Hardware**
  - Using I/O Ports
  - Example: the Parallel Port
  - Side effects and compiler optimization
  - I/O APIs
  - The hardware
  - Driver example
  - Barriers
  - Accessing hardware from User Space
  - User-Mode Access to Devices
    - open, close, read, write
    - ioctl
    - ioperm, iopl, inb, outb
    - mmap, munmap
  
- **Hardware and Interrupt Handling**
  - Installing and implementing an interrupt handler
  - Restrictions of kernel code running in interrupt context
  - IRQs & their Registration
  - IRQ Handling & Control
  - Top & Bottom Halves
  - Autodetecting IRQ's
  - Kernel Help in detecting IRQ's
  - Probing for the interrupt source
  - Enabling and Disabling Interrupts
  - Lab exercises
  
- **Tasklets and Bottom halves**
  - Task queues
  - Lab exercises

- **Kernel Threads**
  - Lab exercises
  
- **Sleep and wakeup (wait queues)**
  
- **Buffer allocation**
  
- **Memory Mapping and DMA**
  - Direct Memory Access/Bus Mastering
    - Consistent Mapping
    - Streaming Mapping
    - Scatter/Gather Mapping
  
- **Memory Management**
  - Allocating Memory
  - Accessing Memory
  - Get Some Space (kmalloc()), kfree(), various flags
  - Get Some Pages (get\_free\_page())
  - Get Some Virtual Memory - vmalloc()
  - Get Some Boot-time Space
  
- **Concurrency and Race Conditions**
  - UP vs SMP Issues
  - Combating Race Conditions
  - Atomic Operations
  - Semaphores
  - Spin Locks
  
- **Time, Delays and Deferred Work**
  - Kernel Timers
  - Timer handling
  - HZ and Jiffies
  - Time of Day
  - Delayed Execution
  - Kernel Timers
  - Current time
  
- **The Linux Device Model**

- **Character Device Drivers**
  - Registering a character device driver
  - The file structure
  
  - Major and minor numbers
  
  - Character Device Methods `open()`, `release()`, `read()`, `write()`
  
  - Data Transfers between User Process and Driver
    - `copy_from_user()`, `copy_to_user()`
  - Making a Device File
  
  - Memory Access in Kernel Space
  
  - Programming with `ioctl()`, `mmap()`
  
  - `devfs` / `lseek` / `ioctl`
  
  - Lab exercises
  
- **Writing various Character Drivers**
  - Memory Based Driver
  - IO PORT (Hardware) Based Driver
  
- **Enhanced Character Device Drivers**
  - Driver and Device Control with '`ioctl()`'
  - Blocking VS Non-Blocking I/O
  - Sleep vs Wakeup
  - `poll()`
  - `llseek()`
  - Access Control through Drivers
  
- **Programming with `ioctl()`**
  - writing device driver with `ioctl()`
  
  - Adding `ioctl`'s in an existing device driver
  
  - Lab exercises

- **Netlink socket interface**
  - point to point, multicast and broadcast
  - UDP, TCP and Raw sockets
  - Writed kernel module and userspace applilcations using Netlink sockets
  - Lab exercises
  
- **Network Drivers**
  - The net\_device structure in detail
  - Packet transmission
  - Packet reception
  - Simulating a network device
  - Lab exercises
  
- **Block Device Drivers**
  - Handling requests
  - Ram Disk Driver
  - Block drivers structures
  - Block Device Driver Registration
  - Block Device Operations & its related Kernel DS
  - Request Queues & their Processing
  - Lab exercises
  
- **Adding a Driver to the Kernel Tree**
- **A sample device driver project**