



# Linux Device Driver Development

## Course Objectives

### Course Description

This course teaches attendees to develop device drivers for a wide range of device types for Linux.

This course acquaints developers with the issues essential for Linux device driver development. The course progresses through a number of topics. Each topic is presented along with a supporting laboratory exercise before moving on to the next topic.

### Overview

This four and one-half day (four day when on-site) course provides substantial practice with the key steps in developing Linux device drivers. The course shows attendees how device drivers work with the Linux kernel, how to compile and load drivers, how to debug drivers, how to access PCI hardware, how to interact with USB devices, as well as other essential topics.

Attendees will develop a complete, simple, driver that demonstrates the process of creating a Linux device driver. The course covers the key issues in Linux device drivers. Such questions as: how do I develop a character device, how do I debug a driver, how do I use tasklets are examined.

Attendees will spend approximately 50 percent of the class time actually gaining hands-on experience with these topics.

This course covers a recent 2.6 kernel such as 2.6.34.

- To provide an understanding of the essentials of Linux device drivers.
- To give you practical experience in developing Linux device drivers.
- To explain the characteristics of the Linux kernel important to device driver writers.

### Attendees will learn:

- The steps necessary to add device drivers to a Linux system
- How to determine what hardware is present on a Linux system
- The purpose and functionality of device drivers
- Compiling and linking device drivers
- Basics of network device drivers.
- Trade-offs between loadable modules and drivers compiled into the kernel.

### Who Should Attend:

The course is designed for software engineers who are new to Linux and/or device drivers. Attendees should have experience with C, be able to perform basic UNIX commands, and have some experience with the basic GNU tools of gcc, gdb, and make. The Linux Development Fundamentals, Tools & Techniques course is a good prerequisite to this course.

### Duration

Four and one-half days (four days when on-site)

### Course Materials

The workshop materials include a comprehensive student work-



book. The workbook contains all of the slides used in the course as well as hands-on lab exercises. Students are able to bring home the lab and example code used in the class.

### **Course Workshop and Set-up:**

The workshop makes use of standard PC's with a desktop Linux distribution for development. The course will make use of PC's and PC devices as examples.

Classroom computers need to have the required Linux distribution.



# Device Driver Development Outline

## 1. How To Configure And Install The Kernel

- 1.1. The kernel source code
- 1.2. Configure and build a new kernel
- 1.3. Install the new kernel

## 2. How Loadable Modules Work

- 2.1. Benefits of loadable modules
- 2.2. Correct use of insmod, modprobe, rmmod, and lsmod
- 2.3. Passing parameters to a module
- 2.4. The GPL and Linux

## 3. Compiling

- 3.1. Identifying important header files
- 3.2. Writing a simple module
- 3.3. Compiling modules
- 3.4. Loading/unloading modules
- 3.5. Exporting symbols from a loadable module
- 3.6. Creating stacked loadable modules

## 4. Tracing and Debugging

- 4.1. **printk** for debugging
- 4.2. Information in **/proc** and **/sys**
- 4.3. **strace** to track system calls
- 4.4. **ksyms**
- 4.5. Debuggers, e.g., **gdb**, and **kgdb**

## 5. Character Devices

- 5.1. Classes of device files
- 5.2. Major and minor numbers
- 5.3. Creating device files with **mknod**
- 5.4. Registering character device file
- 5.5. Listing character device driver methods
- 5.6. Dynamic major/minor numbers

## 6. Data: User To/From Kernel

- 6.1. Functions for accessing user space
- 6.2. Shared Memory
- 6.3. Issues with accessing user space from kernel space

## 7. IOCTL, mmap() and fasync()

- 7.1. What is **ioctl**
- 7.2. Using **ioctl** commands Implementing IOCTL in drivers
- 7.3. Implementing **mmap()** in a driver
- 7.4. Implementing **fasync()** in a driver

## 8. Blocking and Wait Queues

- 8.1. **Schedule()**
- 8.2. Wait Queues



8.3. Safe sleeping

8.4. Poll()

## 9. Memory management

9.1. Memory allocation with **kmalloc** and **kfree**

9.2. Page-oriented memory allocation

## 10. I/O ports and interrupts

10.1. Uses of I/O ports and IRQs

10.2. Platform dependency issues

10.3. Reading and writing I/O ports

10.4. Interrupt Handler functions

10.5. Restrictions on kernel code running in an interrupt context

## 11. Time Mgmt. And Tasklets

11.1. Timer interrupts

11.2. Delay execution techniques

11.3. Tasklets and workqueues

11.4. Obtaining the current time

## 12. Synchronization

12.1. Race conditions

12.2. Atomic access

12.3. Spinlocks

12.4. The Kernel Lock

12.5. Disabling interrupts

## 13. Accessing PCI hardware

13.1. Code to detect PCI devices

13.2. Resource conflicts

13.3. Vendor/device IDs

13.4. I/O mapping

13.5. DMA

## 14. Network Drivers

14.1. The net\_device structure

14.2. Naming scheme

14.3. Network driver methods

## 15. Block Device Drivers

15.1. Block device drivers

15.2. Header files

15.3. Registering block drivers

## 16. USB and Udev

16.1. Dynamic creation of device files

16.2. Dynamic loading of USB drivers