3-Day Linux Device Driver Development

Course Description

This course teaches attendees to develop device drivers 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 three day course provides 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, and how to debug drivers.

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

Course Objectives

• To provide an understanding of the essentials of Linux device drivers.
• To give you practical experience in developing Linux device drivers.

Attendees will learn:

• The steps necessary to add devices 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
• 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 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

Three days

Course Materials

The workshop materials include a comprehensive student workbook and CD. The workbook contains all of the slides used in the course as well as hands-on lab exercises.

The CD contains the lab exercise code as well as a large amount of Linux software.

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.

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 module
- 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. Device 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. Important functions for accessing user space
- 6.2. Shared Memory
- 6.3. Issues with accessing user space from kernel space

7. Blocking and Wait Queues
- 7.1. Schedule()
- 7.2. Wait Queues
- 7.3. Safe sleeping
- 7.4. Poll()

8. I/O ports and interrupts
- 8.1. Uses of I/O ports and IRQs
- 8.2. Platform dependency issues
- 8.3. Functions used for reading and writing I/O ports
- 8.4. Interrupt Handler functions
- 8.5. Restrictions on kernel code running in interrupt context

9. Accessing PCI hardware
- 9.1. Code to detect PCI devices
- 9.2. Resource conflicts
- 9.3. Vendor/device IDs
- 9.4. I/O mapping
- 9.5. DMA

10. Network Drivers
- 10.1. The net_device structure
- 10.2. Naming scheme
- 10.3. Network driver methods

11. Block Device Drivers
- 11.1. Block device drivers
- 11.2. Header files
- 11.3. Registering block drivers
- 11.4. The block_device_operations structure
- 11.5. Special methods