K Computing

Essential Linux Device Driver Development Skills

Description

This fast paced 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.

Students should be prepared for a significant amount of hands-on programming.

Overview

This two 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

After this course attendees will be able too ...

- Create a Linux device driver
- Configure and build a Linux kernel
- utilize a variety of kernel API functions for kernel level code including device drivers
- explain a number of gotchas and concerns to device driver writers in Linux

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 and be able to perform basic Linux commands.

Others, such as engineering managers wishing to gain fundamental understanding of the Linux kernel and device drivers will also benefit.

Duration

Two days

Course Materials

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

Students should bring a personal USB thumbdrive to use to bring home class files.

Course Workshop and Set-up:

The workshop makes use of laptops with a desktop Linux distribution for development. The course will make use of laptops and laptop devices as examples.

Students will share a computer with two students per computer. Students work as a team on the laboratory exercises.

Course Outline

1. How To Configure And Install The Kernel

- The kernel source code
- Configure and build a new kernel
- Install the new kernel

Lab Exercises

- configure and build a Linux kernel
- peruse Linux kernel source code

2. How Loadable Modules Work

- Benefits of loadable modules
- Use of insmod, modprobe, rmmod, and lsmod
- Passing parameters to a module
- The GPL and Linux

Lab Exercises

- Work with loading and unloading modules
- Work with module dependencies
- Work with modprobe.conf file

3. Compiling

- Identifying important header files

- Writing a simple module
- Compiling modules
- Loading/unloading modules
- Exporting symbols from a loadable module
- Creating stacked loadable modules

Lab Exercises

- Write a loadable kernel module
- Use module macros related to the modinfo command
- Write a module that uses module parameters
- Write a module to avoid word size and endian issues.

4. Tracing and Debugging

- printk for debugging
- Device information in /proc and /sys
- strace to track system calls
- ksyms
- Debuggers, e.g., gdb, and kgdb

Lab Exercises

- Work with controlling printk messages on the console
- Create a proc file
- Work with an Oops message
- Use strace to see driver function results

5. Character Devices

- Classes of device files
- Major and minor numbers
- Creating device files with mknod
- Registering character device file
- Listing character device driver methods
- Dynamic major/minor numbers

Lab Exercises

- Examine /proc/devices
- Work with device files
- Create a skeleton character device driver
- Adapt the driver to act differently depending upon the minor number of the device file.
- Integrate your driver into a kernel source tree

6. Data: User To/From Kernel

- Important functions for accessing user space
- Shared Memory
- Issues with accessing user space from kernel space

Lab Exercises

- Implement a write() function
- Implement a read() function

8. Blocking and Wait Queues

- Schedule()
- Wait Queues
- Safe sleeping

- Poll()

Lab Exercises

- Mimic named pipe behavior via a driver
- Create and use a wait queue
- Implement non-blocking
- Implement the fsync() function
- Implement the poll() function

7. I/O ports and interrupts

- Uses of I/O ports and IRQs
- Platform dependency issues
- Functions used for reading and writing I/O ports
- Interrupt Handler functions
- Restrictions on kernel code running in interrupt context

Lab Exercises:

- Do IO Port operations inb and outb
- Attach an interrupt handler to the mouse and keyboard interrupts
- Create a proc file to track interrupts