Linux Kernel Development

This Linux Kernel Development course aims to make you a Linux kernel developer. You'll start with understanding operating systems and the Linux kernel's role. Then, you'll gain hands-on experience building and modifying the kernel source code. The core of the course dives into essential concepts like process management and device drivers.

Duration: 5 Days

Lab: Koenig DC

Module 1 – Introduction

History of Unix
Linus: Introduction to Linux
Overview of Operating Systems and Kernels
Linux Versus Classic Unix Kernels
Linux Kernel Versions
The Linux Kernel Development Community
What is Memory
The Boot Process
Installing What We Need For Real Mode Development

Module 2 – Assembly Language

What is assembly language?
Installing the emulator
Hello World In Assembly
Transistors And Logic Gates Understanding The Processor
Registers in the 8086
Segmentation Memory Model Explained
The Stack, Subroutines And Endiness Explained

Module 3 – Real Mode Development

Hello World Bootloader
Understanding Real Mode
Segmentation Memory Model
Improving Our Bootloader
Preparing our bootloader to be booted on real hardware
Writing our bootloader to a USB stick
The Interrupt Vector Table Explained
Implementing our own interrupts in real mode
Disk Access And How It Works
Reading from the hard disk

Module 4 – Protected Mode Development

What is Protected Mode?
Switching To Protected Mode
Restructuring Our Project
Enabling the A20 line
Creating a Cross Compiler So We Can Code In C

Loading our 32 bit kernel into memory and working with debugging symbols
Cleaning our object files
Dealing With Alignment Issues
C Code In Protected Mode
Text Mode Explained
Writing To The Screen, Hello World Tutorial
Interrupt Descriptor Table Explained
Implementing The Interrupt Descriptor Table
Implementing In and Out Functions
Programmable Interrupt Controller Explained
Programmable Interrupt Controller Implementation
Understanding The Heap And Memory Allocation
Implementing Our Heap
Creating the enable interrupts function

Module 5 – Paging

Implementing Paging
Modifying the page table
Preparing To Read From The Hard Disk
Reading from the disk in C with the ATA controller
Improving Our Disk Driver

Module 6 – File System

What is a file system?
Creating a path parser
Creating a disk stream
File Allocation Table Explained
Starting To Create our FAT File system
Understanding the VFS(Virtual File System) Layer
Implementing our virtual filesystem core functionality
implementing FAT16 filesystem driver core functionality
Implementing FAT16 Structures
Implementing The FAT16 Resolver Function
Implementing the VFS fopen function
Implementing FAT16 fopen function
Implementing the VFS fread function
Implementing FAT16 fread functionality
Implementing the VFS fseek functionality
Implementing the FAT16 fseek functionality
Implementing the fstat VFS functionality
Implementing the FAT16 fstat function
Implementing the VFS fclose functionality
Implementing the FAT16 fclose functionality

Module 7 – Kernel,User Land and Process

Implementing a kernel panic
Understanding User Land
Changing our kernel segment and data descriptors to be written in C
Implementing The TSS(Task Switch Segment)
Implementing Task Foundations
Implementing Process Foundations Part 1
Implementing Process Foundations Part 2
Packing the GDT
Implementing User Land Functionality

Creating our first user process application
Executing the process and dropping into user land privileges
Changing the paging functionality
Talking with the kernel from userland
Creating the interrupt 0x80 for user process to kernel communication
Creating the ability to create and execute kernel commands
Creating our first kernel command
Calling our kernel command
Copying strings from the tasks process
Reading the task's stack
Creating the print command in the kernel

Module 8 – Keyboard Access & Driver

Understanding keyboard access in protected mode
Creating the virtual keyboard layer
Creating the PS2 port keyboard driver part 1
Improving our interrupt descriptor table design
Creating a cleaner way to create interrupt handlers in the interrupt descriptor
Changing The Current Process
Creating the PS2 port keyboard driver part 2
Getting a key from the keyboard buffer in user land
Creating a putchar command that writes one character to the terminal
Implementing backspace in the terminal
Revising our stream reader

Module 9 – ELF Loader & User Programs

ELF(Executable Linkable Format) files
Implementing The Elf Loader
Writing User Programs In C
Implementing system print in stdlib
Implementing system get key in stdlib
Implementing Malloc In Our stdlib
Implementing Free In Our stdlib
Changing the way we map virtual pages for the process
Implementing itoa function
Implementing the putchar function
Implementing the printf function
Implementing the ability to read lines

Module 10 – Shell

Creating a shell
Loading other programs from our shell
Creating some important stdlib functions
Memory Mapping malloc in stdlib
Memory Unmapping free In stdlib
Process arguments
Implementing A 'System' Command
Implementing program termination
Handling program crashes
Creating an exit command
Handling caps lock, upper case and lower case letters
Running multiple tasks at the same time multi-tasking

KOENIG
step forward

Module 11 – fat16 Functionaility

Changing our fat16_new_fat_item_for_directory_item function
Changing our fat16_open function
Changing our fat16_get_root_directory function
Changing our process_load_binary function
Improvements to our fat16_to_proper_string function
Changing our restore_general_purpose_registers function