Linux Kernel Internals and Development (LFD420)

Modality: Virtual Classroom

Duration: 4 Days

SATV Value:

CLC:

NATU:

SUBSCRIPTION: No

Intro to Course:

Know how to design Linux Kernel. This course is designed for you to learn how the Linux is structured. The primary tools for designing the Kernel and how productively work with the Linux Developer community. This course is absolutely designed for you if you are interested to know about Linux Kernel.

The main design of the course is to offer experience programmers with the strong knowledge of the Linux Kernel. Additionally, it provides in-depth knowledge of the theoretical and ideological aspects of Linux Kernel. Participations will be offered in practical trainings and demonstrative presentations, developed to give you important utilities to design and debug Linux Kernel code.

An Embedded Linux Developer usually gets a pay of US $ 107,500 per annum on an average.

Course Outline:

After attending this course, you will have a knowledge of:

- How Linux is structured.
- How Kernel algorithms work
- Hardware and Memory Management
- Modularization tools and debugging.
- The operation of kernel developer communities and how to work with it more productively.
- And so on.

This course will also teach you on how to go on with any major Linux distribution.

Who can enroll?

- App Developers
- C/C ++, C# Developers
- Linux Developers

Main Requirement!

Before you enrolled in this course, you must:

- Expert in C programming language.
- Have knowledge of basic Linux (UNIX) tools such as Is, Grep and Tar.
- Hand on any available text editors such as Emacs, Vi etc.
- An experience of important Linux distribution will be an added advantage, but not a requirement.

Course Outline:

Introduction

- Objectives
- Who You Are
- The Linux Foundation
- Linux Foundation Training
- Linux Distributions
- Platforms
- Preparing Your System
- Using and Downloading a Virtual Machine
- Things change in Linux
- Documentation and Links
- Course Registration

Preliminaries

- Procedures
- Kernel Versions
- Kernel Sources and Use of git

How to Work in OSS Projects **

- Overview on How to Contribute Properly
- Stay Close to Mainline for Security and Quality
- Study and Understand the Project DNA
- Figure Out What Itch You Want to Scratch
- Identify Maintainers and Their Work Flows and Methods
- Get Early Input and Work in the Open
- Contribute Incremental Bits, Not Large Code Dumps
- Leave Your Ego at the Door: Don’t Be Thin-Skinned
- Be Patient, Develop Long Term Relationships, Be Helpful

Kernel Architecture I

- UNIX and Linux **
- Monolithic and Micro Kernels
- Object-Oriented Methods
- Main Kernel Tasks

- User-Space and Kernel-Space
- Kernel Mode Linux **

Kernel Programming Preview

- Error Numbers and Getting Kernel Output
- Task Structure
- Memory Allocation
- Transferring Data between User and Kernel Spaces
- Linked Lists
- String to Number Conversions
- Jiffies
- Labs

Modules

- What are Modules?
- A Trivial Example
- Compiling Modules
- Modules vs Built-in
- Module Utilities
- Automatic Loading/Unloading of Modules
- Module Usage Count
- The module struct
- Module Licensing
- Exporting Symbols
- Resolving Symbols **
- Labs

Kernel Architecture II

- Processes, Threads, and Tasks
- Process Context
- Kernel Preemption
- Real Time Preemption Patch
- Dynamic Kernel Patching
- Run-time Alternatives **
- Porting to a New Platform **
- Labs

Kernel Initialization

- Overview of System Initialization
- System Boot
- Das U-Boot for Embedded Systems**

Kernel Configuration and Compilation

- Installation and Layout of the Kernel Source
- Kernel Browsers
- Kernel Configuration Files
- Kernel Building and Makefiles
- initrd and initramfs
- Labs

System Calls

- What are System Calls?
- Available System Calls
- How System Calls are Implemented
- Adding a New System Call
- Labs

Kernel Style and General Considerations

- Coding Style
- kernel-doc **
- Using Generic Kernel Routines and Methods
- Making a Kernel Patch
- sparse
- Using likely() and unlikely()
- Writing Portable Code, CPU, 32/64-bit, Endianness
- Writing for SMP
- Writing for High Memory Systems
- Power Management
- Keeping Security in Mind
- Mixing User- and Kernel-Space Headers **
- Labs

Race Conditions and Synchronization Methods

- Concurrency and Synchronization Methods
- Atomic Operations
- Bit Operations
- Spinlocks
- Seqlocks
- Disabling Preemption
- Mutexes
- Semaphores
- Completion Functions
- Read-Copy-Update (RCU)
- Reference Counts
- Labs

SMP and Threads

- SMP Kernels and Modules
- Processor Affinity
- CPUSETS
- SMP Algorithms – Scheduling, Locking, etc.
- Per-CPU Variables **
- Labs

Processes

- What are Processes?
- The task_struct
- Creating User Processes and Threads
- Creating Kernel Threads
- Destroying Processes and Threads
- Executing User-Space Processes From Within the Kernel
- Labs

Process Limits and Capabilities **

- Process Limits
- Capabilities
- Labs

Monitoring and Debugging

- Debuginfo Packages
- Tracing and Profiling
- sysctl
- SysRq Key
- oops Messages
- Kernel Debuggers
- debugfs
- Labs

Scheduling

- Main Scheduling Tasks
- SMP
- Scheduling Priorities
- Scheduling System Calls
- The 2.4 schedule() Function
- O(1) Scheduler
- Time Slices and Priorities
- Load Balancing
- Priority Inversion and Priority Inheritance **
- The CFS Scheduler
- Calculating Priorities and Fair Times
- Scheduling Classes

- CFS Scheduler Details
- Labs

Memory Addressing

- Virtual Memory Management
- Systems With and Without MMU and the TLB
- Memory Addresses
- High and Low Memory
- Memory Zones
- Special Device Nodes
- NUMA
- Paging
- Page Tables
- page structure
- Kernel Samepage Merging (KSM) **
- Labs

Huge Pages

- Huge Page Support
- libhugetlbfs
- Transparent Huge Pages
- Labs

Memory Allocation

- Requesting and Releasing Pages
- Buddy System
- Slabs and Cache Allocations
- Memory Pools
- kmalloc()
- vmalloc()
- Early Allocations and bootmem()
- Memory Defragmentation
- Labs

Process Address Space

- Allocating User Memory and Address Spaces
- Locking Pages
- Memory Descriptors and Regions
- Access Rights
- Allocating and Freeing Memory Regions
- Page Faults
- Labs

Disk Caches and Swapping

- Caches
- Page Cache Basics
- What is Swapping?
- Swap Areas
- Swapping Pages In and Out
- Controlling Swappiness
- The Swap Cache
- Reverse Mapping **
- OOM Killer
- Labs

Device Drivers**

- Types of Devices
- Device Nodes
- Character Drivers
- An Example
- Labs

Signals

- What are Signals?
- Available Signals
- System Calls for Signals
- Sigaction
- Signals and Threads
- How the Kernel Installs Signal Handlers
- How the Kernel Sends Signals
- How the Kernel Invokes Signal Handlers
- Real Time Signals
- Labs

Closing and Evaluation Survey

**** These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints**