Document Generated: 09/20/2024

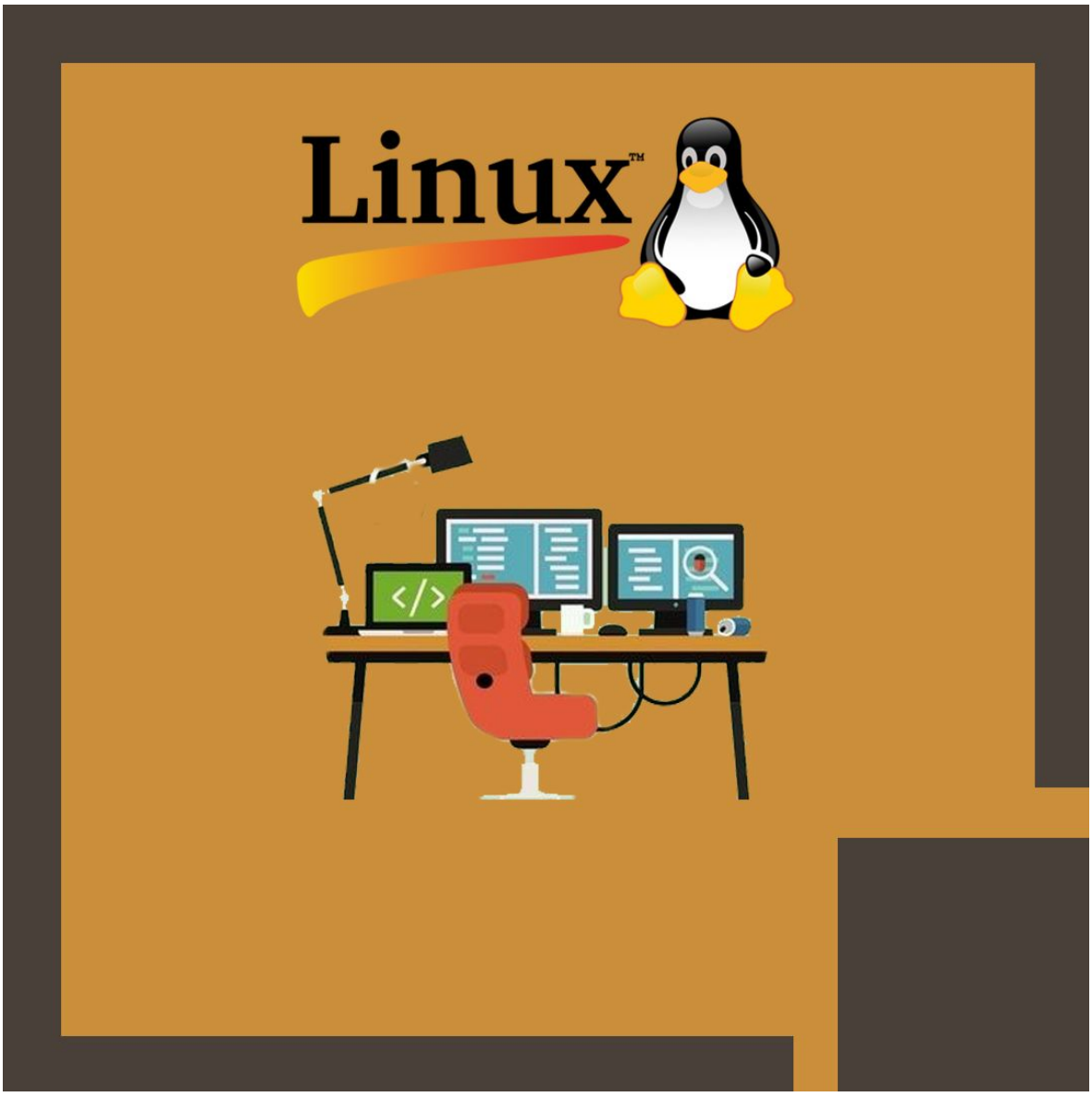
Learning Style: Virtual Classroom

Provider: Linux Foundation

Difficulty: Intermediate

Course Duration: 4 Days

Linux Kernel Internals and Development (LFD420)



About this course:

Learn how to develop for the Linux kernel. In this course you'll learn how Linux is architected, the basic methods for developing on the kernel, and how to efficiently work with the Linux developer community. If you are interested in learning about the Linux kernel, this is absolutely the definitive course on the subject.

This course is designed to provides experienced programmers with a solid understanding of the Linux kernel. In addition to a detailed look at the theory and philosophy behind the Linux kernel, you'll also participate in extensive hands-on exercises and demonstrations designed to give you the necessary tools to develop and debug Linux kernel code.

The average salary of an Embedded Linux Developer is **$107,500** per year.

Course Objective:

In this course you'll learn:

- How Linux is architected
- How kernel algorithms work
- Hardware and memory management
- Modularization techniques and debugging
- How the kernel developer community operates and how to efficiently work with it.
- And much more.

The information in this course will work with any major Linux distribution.

Audience:

- App Developers
- C/C++,C# developers
- Linyx Developers

Prerequisite:

- Students should be proficient in the C programming language, basic Linux (UNIX) utilities such as ls, grep and tar, and be comfortable with any of the available text editors (e.g. emacs, vi, etc.) Experience with any major Linux distribution is helpful but not strictly required.

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**

Credly Badge:

Display your Completion Badge And Get The Recognition You Deserve.

Add a completion and readiness badge to your Linkedin profile, Facebook page, or Twitter account to validate your professional and technical expertise. With badges issued and validated by Credly, you can:

- Let anyone verify your completion and achievement by clicking on the badge
- Display your hard work and validate your expertise
- Display each badge's details about specific skills you developed.

Badges are issued by QuickStart and verified through Credly.

Find Out More or See List Of Badges