Document Generated: 02/17/2026

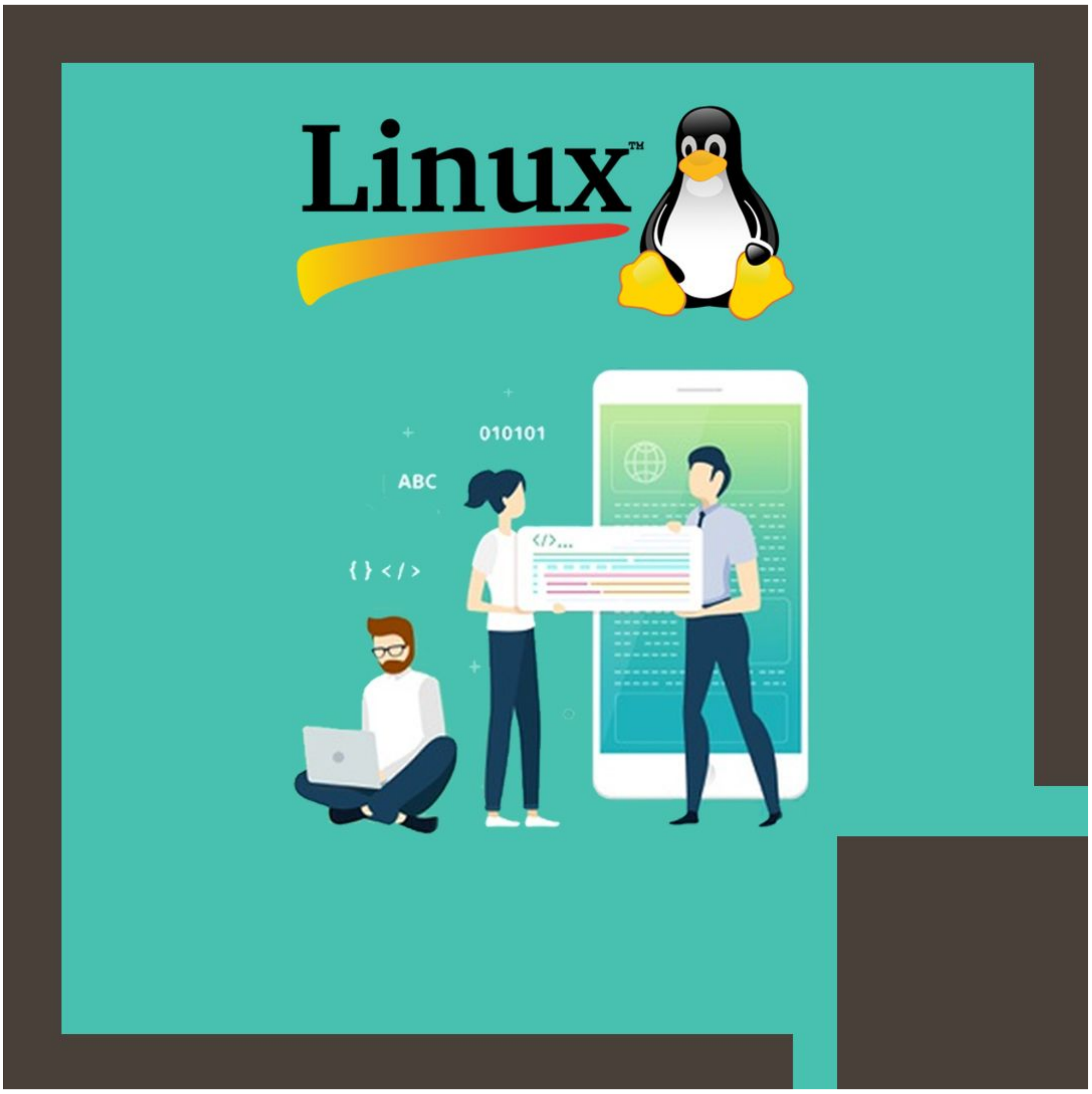
Learning Style: Virtual Classroom

Technology: Linux Foundation

Difficulty: Intermediate

Course Duration: 4 Days

Developing Applications For Linux (LFD401)



About this Course:

Professionals sharing proficiency in developing applications and software for Linux Environment have great demand in a Linux-based business enterprise and organization. On average, an Embedded Linux Developer earns $107,500 annually and can easily secure a reputable job in today's competitive world of business.

The Developing Applications For Linux (LFD401) course is designed for experienced App Developers and Language Programmers who want to learn the art of developing apps in Linux environment. Through this course, professionals get practical knowledge and experience of working with the major tools and development methods used in Linux App Development. This teachings of this course also covers unique techniques used in Linux App Development for designing and optimizing applications.

Experienced App developers can greatly benefit from the teachings of this course and can expand their skill set in accordance with the evolving needs of the app development world. This allows them to efficiently manage processes, troubleshoot, and debug errors in a Linux-based business enterprise.

Course Objectives:

The core objective of this course is to help professionals gain a better understanding and sound knowledge of the following key principles:

- C Programs and App Development Tools and Methods
- Linux System Programming
- Process Management, Troubleshooting, and Debugging Techniques
- Streamlining Linux App Development
- Linux System and Paid Calls
- Working with Major Linux Distributions

Audience:

- C# & C/C++ App Developers
- Linux Developers
- IT Professionals and Experts

Prerequisites:

Professionals planning to enroll in the Developing Applications For Linux (LFD401) Course must have the practical experience and technical knowledge of software and app development.

Candidates must have C Programming Language proficiency and should be familiar with the key concepts of Text Editors and Linux Utilities.

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
- Course Registration

Preliminaries

- Procedures
- Standards and the LSB

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

Compilers

- GCC
- Other Compilers
- Major gcc Options
- Preprocessor
- Integrated Development Environments (IDE)
- Labs

Libraries

- Static Libraries
- Shared Libraries
- Linking To Libraries
- Dynamic Linking Loader
- Labs

Make

- Using make and Makefiles

- Building large projects
- More complicated rules
- Built-in rules
- Labs

Source Control

- Source Control
- RCS and CVS
- Subversion
- git
- Labs

Debugging and Core Dumps

- gdb
- What are Core Dump Files?
- Producing Core Dumps
- Examining Core Dumps
- Labs

Debugging Tools

- Electric Fence
- Getting the Time
- Profiling and Performance
- valgrind
- Labs

System Calls

- System Calls vs. Library Functions
- How System Calls are Made
- Return Values and Error Numbers
- Labs

Memory Management and Allocation

- Memory Management
- Dynamical Allocation
- Tuning malloc()
- Locking Pages
- Labs

Files and Filesystems in Linux **

- Files, Directories and Devices
- The Virtual File System
- The ext2/ext3 Filesystem
- Journaling Filesystems

- The ext4/ Filesystem
- Labs

File I/O

- UNIX File I/O
- Opening and Closing
- Reading, Writing and Seeking
- Positional and Vector I/O
- Standard I/O Library
- Large File Support (LFS)
- Labs

Advanced File Operations

- Stat Functions
- Directory Functions
- inotify
- Memory Mapping
- flock() and fcntl()
- Making Temporary Files
- Other System Calls
- Labs

Processes – I

- What is a Process?
- Process Limits
- Process Groups
- The proc Filesystem
- Inter-Process Communication Methods
- Labs

Processes – II

- Using system() to Create a Process
- Using fork() to Create a Process
- Using exec() to Create a Process
- Using clone()
- Exiting
- Constructors and Destructors
- Waiting
- Daemon Processes
- Labs

Pipes and Fifos

- Pipes and Inter-Process Communication
- popen() and pclose()
- pipe()

- Named Pipes (FIFOs)
- splice(), vmsplice() and tee()
- Labs

Asynchronous I/O**

- What is Asynchronous I/O?
- The POSIX Asynchronous I/O API
- Linux Implementation
- Labs

Signals – I

- What are Signals?
- Signals Available
- Dispatching Signals
- Alarms, Pausing and Sleeping
- Setting up a Signal Handler
- Signal Sets
- sigaction()
- Labs

Signals – II

- Reentrancy and Signal Handlers
- Jumping and Non-Local Returns
- siginfo and sigqueue()
- Real Time Signals
- Labs

POSIX Threads – I

- Multi-threading under Linux
- Basic Program Structure
- Creating and Destroying Threads
- Signals and Threads
- Forking vs. Threading
- Labs

POSIX Threads – II

- Deadlocks and Race Conditions
- Mutex Operations
- Semaphores
- Futexes
- Conditional Operations
- Labs

Networking and Sockets

- Networking Layers
- What are Sockets?
- Stream Sockets
- Datagram Sockets
- Raw Sockets
- Byte Ordering
- Labs

Sockets – Addresses and Hosts

- Socket Address Structures
- Converting IP Addresses
- Host Information
- Labs

Sockets – Ports and Protocols

- Service Port Information
- Protocol Information
- Labs

Sockets – Clients

- Basic Client Sequence
- socket()
- connect()
- close() and shutdown()
- UNIX Client
- Internet Client
- Labs

Sockets – Servers

- Basic Server Sequence
- bind()
- listen()
- accept()
- UNIX Server
- Internet Server
- Labs

Sockets – Input/Output Operations

- write(), read()
- send(), recv()
- sendto(), recvfrom()
- sendmsg(), recvmsg()
- sendfile()
- socketpair()
- Labs

Sockets – Options

- Getting and Setting Socket Options
- fcntl()
- ioctl()
- getsockopt() and setsockopt()
- Labs

Netlink Sockets**

- What are netlink Sockets?
- Opening a netlink Socket
- netlink Messages
- Labs

Sockets – Multiplexing and Concurrent Servers

- Multiplexed and Asynchronous Socket I/O
- select()
- poll()
- pselect() and ppoll()
- epoll
- Signal Driven and Asynchronous I/O
- Concurrent Servers
- Labs

Inter Process Communication

- Methods of IPC
- POSIX IPC
- System V IPC**
- Labs

Shared Memory

- What is Shared Memory?
- POSIX Shared Memory
- System V Shared Memory**
- Labs

Semaphores

- What is a Semaphore?
- POSIX Semaphores
- System V Semaphores**
- Labs

Message Queues

- What are Message Queues?

- POSIX Message Queues
- System V Message Queues**
- 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