

Linux Application Programming Course Summary

Description

UNIX-like systems, such as Linux, are well known for their wealth of application development tools. This course provides the student with information on how the Linux GNU C compiler and link editor (binder) work, as well as how applications interface with the operating system through The Linux kernel API (system calls.) It also provides practical training in the use of the make utility, as well the GNU debugger GDB. This course also teaches the student about features common to all UNIX operating systems, such as process creation and management, the Native POSIX Threads Library (NPTL), interprocess communications, network programming and file I/O. It stresses the importance of standards such as POSIX and X/Open and their impact on the Linux programming environment.

Objectives

At the end of this course, students will be able to:

- Compile and debug C applications in the Linux environment
- Manage software development projects by using the make facility, the GNU debugger and various integrated software development tools
- Understand the Linux application address space and memory model, real-time processing, memory mapped files and file systems, threads and sockets
- Create shared and non-shared libraries

Topics

- The Linux Programming Environment
- The Linux Kernel API
- Linux Process
- Linux File I/O
- Linux Pipes
- Memory Mapped Files
- Linux Signals
- Linux and POSIX Threads
- Shared and Non-Shared Libraries
- System V IPSs
- An Introduction to Sockets
- Using the Make Facility Using gdb

Audience

This course is designed for C programmers who already have the skills needed to use Linux and now wants to know how to use the vast array of programming tools found in the Linux environment.

Prerequisites

Students must possess basic C skills prior to taking this course. Students must also have user-level experience with Linux or another UNIX-based operating system and be able to create files with the vi editor.

Duration

Five days

Linux Application Programming

Course Outline

I. The Linux Programming Environment

- A. Components of the Linux Programming Environment
- B. The Compilation Process
- C. Preprocessor Directives
- D. Linux Header Files
- E. The Link Editor
- F. Standard Linux Libraries
- G. Object Files
- H. The Program Image (ELF)
- I. The Process Image
- J. User Limitations
- K. Building Applications from Multiple Source Files
- L. Multi-Threaded Processes
- M. Compiler and Link Editor Options
- N. Optimization

II. The Linux Kernel API

- A. The Linux Kernel
- B. System Calls and Mode Switches
- C. The errno Variable
- D. System Calls Vs Library Routines
- E. Using perror

III. Linux Processes

- A. Process Management System Calls
- B. Processes and Threads
- C. Attributes of Processes
- D. Credentials and Security
- E. Capabilities
- F. The Life Cycle of a Process
- G. The fork() System Call Inheritance
- H. The exec Family of System Calls
- I. The exit() System Call
- J. The wait Family of System Calls
- K. The Exit Code
- L. Dealing with Zombies
- M. PIDs, UIDs and GIDs

IV. Linux File I/O

- A. Linux File Types
- B. Inodes and Directories
- C. File Systems
- D. Superblocks and the Inode Table
- E. Data Block Access

- F. Hard Links Vs Symbolic Links
- G. Basic File I/O System Calls
- H. Understanding File Descriptors
- I. The open() System Call
- J. The creat() System Call
- K. The dup Family of System Calls
- L. The read() and write() System Calls
- M. The close() System Call
- N. open() Vs fopen()
- O. The lseek() System Calls
- P. Sparse Files
- Q. The link() and unlink() System Calls
- R. The stat() System Call and Friends
- S. Working with Device Files
- T. Basic File System Subroutines
- U. The access() System Call
- V. The umask() System Call
- W. The chmod() and chroot() System Calls
- X. The fcntl() System Call
- Y. File and Record Locking

V. Linux Pipes

- A. Named and Unnamed Pipes
- B. The pipe() System Call
- C. The popen() Subroutine
- D. The mkfifo() Subroutine

VI. Memory Mapped Files

- A. Traditional Linux File I/O
- B. Memory Mapped Files
- C. The mmap() System Call
- D. The munmap() System Call
- E. The mmap2() System Call
- F. Anonymous Mapping
- G. mprotect(), msync() and mlock()

VII. Linux Signals

- A. Linux Signal Names and Numbers
- B. The kill() and killpg() System Calls
- C. Ignoring, Blocking and Handling Signals
- D. The signal() System Call
- E. The sigaction() System Call
- F. The sigprocmask() System Call

Due to the nature of this material, this document refers to numerous hardware and software products by their trade names. References to other companies and their products are for informational purposes only, and all trademarks are the properties of their respective companies. It is not the intent of ProTech Professional Technical Services, Inc. to use any of these names generically

Linux Application Programming

Course Outline (cont'd)

VIII. Linux and POSIX Threads

- A. Multi-Tasking
- B. Multi-Threaded Applications
- C. Amdahl's Law
- D. The pthreads Library
- E. Thread Objects and Attributes
- F. The pthread_create() Routine
- G. The pthread_exit() Routine
- H. Destroying Threads and Thread Attributes
- I. Passing Parameters into
- J. Threads Thread Return Values
- K. The pthread_join() Routine
- L. Setting the Thread Detach State
- M. Threads and Global Data
- N. The Mutex Object
- O. Mutex Attributes
- P. Recursion Issues
- Q. Initializing Mutex Locks
- R. Testing, Setting and Releasing Mutex Locks
- S. Destroying Mutex Locks
- T. Using Conditional Variables
- U. Thread Local Storage
- V. Thread Cancellation
- W. Using valgrind to Profile Applications

IX. Shared and Non-Shared Libraries

- A. Library Conventions
- B. Creating a Non-Shared Library
- C. The ar Command
- D. Creating a Shared Library
- E. ld Options

X. System V IPCs

- A. The ipc_perm Structure
- B. The ipcs and ipcrm Commands
- C. Keys
- D. Shared Memory
- E. Semaphores
- F. Message Queues

XI. An Introduction to Sockets

- A. The TCP/IP Protocol Stack
- B. Socket Domains
- C. Connection and Connectionless Sockets
- D. Creating Socket Clients and Servers

XII. Using the make Facility Using gdb