

Pthreads Programming in Linux

Course Summary

Description

The POSIX threads library has brought a new paradigm of programming design and development to multitasking applications. Traditional C programmers find new challenges writing applications that properly use the features of multithreading library routines. Threads change the way signals are handled, global data is accessed, memory is dynamically allocated and new processes are created. This course teaches the proper way to design and write multithreaded applications using the Pthread library routines.

Objectives

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

- Design, write, compile and debug multithreaded C programs using the Pthread library routines

Topics

- The World of Thread Programming
- Basic Pthread Programming
- Pthread program Design
- Thread Synchronization
- Thread-Specific Data
- Thread Termination
- Threads and the Linux Operating System

Audience

This course is designed for experienced C programmers who need to learn how to write and debug Linux programs that use the POSIX threads (Pthreads) library of multitasking subroutines, and anyone who wants a better understanding of multithreaded programming techniques.

Prerequisites

Students must be experienced with the C programming language, be comfortable with programming in the Linux environment and should be skilled at using the vi editor and navigating the Linux file system. Students must have experience programming Linux system calls and working with Linux processes. These skills may be obtained by taking the following ProTech training courses: Linux Fundamentals and Linux Application Programming.

Duration

Four days

Pthreads Programming in Linux

Course Outline

I. The World of Thread Programming

- A. Multitasking Without Threads
- B. Components of a Linux Process
- C. Characteristics of a Linux Process
- D. Threads of Execution
- E. Multithreaded Programs
- F. Applications for Multithreading
- G. Considerations and Concerns of Multithreading
- H. The Pthread Library
- I. Thread Tools
- J. A Sample Thread Program
- K. Concurrent Execution Vs Parallel Execution

II. Basic Pthread Programming

- A. The Necessary Components
- B. An Object Oriented Approach
- C. The pthread_t Data Type
- D. The pthread_create() Routine
- E. Thread Attributes
- F. The pthread_exit() Routine
- G. The Detached Thread State
- H. The pthread_join() Routine
- I. Keeping Track of Threads
- J. Handling Thread Routine Failures

III. Pthread Program Design

- A. What Makes a Good Thread Application?
- B. Thread Design Models
- C. The Boss/Worker Model
- D. The Peer Model
- E. The Pipeline Module
- F. Moving Data Between Threads
- G. Producer Thread and Consumer Thread
- H. A POP (Point of Purchase) Example

IV. Thread Synchronization

- A. Thread Synchronization Tools
- B. Mutex Thread Objects
- C. Mutex Thread Attributes
- D. Creating and Initializing a Mutex Lock
- E. Acquiring a Mutex Lock
- F. Recursion Issues and Deadlocks
- G. Releasing a Mutex Lock
- H. Priority Inversion
- I. Acquiring and Using the valgrind Tool
- J. Conditional Variables
- K. The Conditional Variable Data Type
- L. Waiting on a Conditional Variable
- M. Timed Waits
- N. Signalling a Conditional Variable
- O. Broadcasting a Conditional Variable
- P. Using Conditional Variables with Mutex Locks
- Q. A Thread Synchronization Example
- R. Synchronizing the POP Example

V. Thread-Specific Data

- A. Threads and malloc()
- B. Keys
- C. Thread Local Storage (TLS)

VI. Thread Termination

- A. Methods of Ending a Thread
- B. Releasing Thread Resources
- C. Killing the Process
- D. The pthread_exit() Routine
- E. The pthread_kill() Routine
- F. Thread Cancellation
- G. Cancellation States

VII. Threads and The Linux Operating System

- A. PIDs and TIDs S
- B. tack Sizes and Locations
- C. Using errno
- D. Threads and Signals
- E. The pthread_atfork() Routine
- F. Working with Semaphores
- G. Debugging Multithreaded Programs