pthreads: POSIX And Linux Multithreading using C/C++

Threading Concepts, Creating/Managing Threads, Synchronization, pthreads & .so, NPTL Internals

This course explores how to use the pthreads C API to build sophisticated multithreaded architectures for POSIX-compatible OSes such as Linux. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. POSIX defines a multithreading specification commonly known as pthreads. This is a C API that strictly specifies the expected behavior of threading and synchronization primitives. Code written to work against pthreads can run on any OS that implements this spec. Linux is one such OS, and the focus for this course, but it is noted that pthreads is also implemented on a wide variety of other popular and specialist OSes.

This course supplies attendees with an understanding of the concepts underlying threading, together with hands-on experience of multithreaded development on Linux. Topics covered include a comprehensive tour of thread creation and lifetime management, the various synchronization approaches, how threads interact with share libraries, memory access and debugging, intra-thread comms and various higher-level design patterns to work with large multithreaded servers.

Target Audience

System architects and experienced developers who need to gain an in-depth understanding of POSIX and Linux multithreading.

Prerequisites

Attendees must have good experience of system-level programming on Linux.

Contents of One-Day Training Course

Thread definition

Scheduling vs. synchronization
Parallelism and concurrency
Compute-bound and I/O bound apps
Race conditions, deadlock, starvation, priority inversion

Threads

Tour of pthreads.h
Overview of main APIs and C structures
Creating a thread with pthread_create()
The threadproc
Thread priority
Thread management & lifetime
pthread_exit() and joining a thread to catch exit and to access exit code
pthread_[detach|attr_setdetachstate]()
Thread cancelation
C11/18 threads vs. pthreads (quite similar)

Synchronization

Conditionals, mutexes, rwlock, spin, barrier – compare & contrast
The “protect data, not code” principle
pthread_cond_[init|destroy|attr_init]()
pthread_mutex_[lock|trylock|unlock]
Blocking vs. non-blocking
pthread_[rwlock|spin|barrier]_init()

Thread Pools

Managing pools of threads for processing timers, work-items and I/O

Memory and Threads

Thread local storage : pthread_key_create
Heap storage vs. stack storage
pthread_attr_[set|get]stack[size|addr]()

Shared Libraries and Threads

How threads interact with shared libraries
Serialized methods
Robust .so design for threads

Debugging with Threads

Querying information about running processes/threads and their attributes
Serialized calls

Resource Management

Creating a custom resource browser, to display which thread is waiting on which synchronization resource
Threads with C++
Threads & exceptions; threads & classes
Accessing resources using smart pointers

Design Issues

Single Writer/Multiple Readers, Monitor, Once-Off Initialization,Dining Philosopher
Calling legacy code from multiple threads
Converting legacy code to multithreading

Multithreaded Architectures

Pipeline, Producer-Consumer, Work-Crew and Master-Slave Models
Create threads on demand vs. elastic pool

NPTL Internals

Native POSIX Thread Library (NPTL) implements pthreads on Linux
Threads and Linux scheduling

Multithreaded Project

Development of a complete multithreaded embedded HTTP web server that uses efficiently manage large numbers of requests