Windows Multithreading Using C/C++

Threading Concepts, Kernel Obj, Processes/Threads, Synchronization, Async I/O, Debugging, DLLs

[Sample: lab exercises] This course explores how to use the Windows C API to build sophisticated multithreaded architectures. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. The Windows kernel object is the opaque foundation for multithreading – based on this are the process, thread, job & various synchronization objects - mutex, event, semaphore and waitable timer – each of which targets different needs. Thread activity, lifetimes and inter-thread commun-ication must be co-ordinated. Threads impact how to develop DLLs, memory and debug.

Various higher-level design patterns may be used to route workitems in multithreaded servers. The optimal server architecture is one active application thread per processor logical core. Tools may be developed to determine which thread is blocked waiting on which resource, and the state/owner of each resource. A server must efficiently multiplex many I/O requests over a few threads - which is the goal of I/O completion ports.

This course supplies attendees with an understanding of the concepts underlying threading, together with experience of multithreaded development.

Target Audience

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

Prerequisites

Attendees must have good experience of system-level programming, on either Windows or 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

Kernel Objects

Windows kernel objects
Usage counting
Kernel object handles
Sharing handles among processes

Processes

CreateProcess API & child processes
Retrieving the exit code of a process
Job objects

Threads

CreateThread API & Threadproc
The C runtime library
Thread priority & processor affinity
Thread management & lifetime

Synchronization

Critical Sections, Mutexes, Events, Semaphores, Waitable Timers, WaitForSingle/MultipleObject
The “Protect data, not code” principle

Memory and Threads

Dynamic & static Thread Local Storage
Heap storage vs. stack storage

Asynchronous I/O

Overlapped, APC & Scatter/Gather
I/O Completion Ports, Asynchronous I/O, Overlapped, APC & Scatter/Gather
I/O Completion Ports

Thread Pools

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

DLLs and Threads

How threads interact with DLLs
Serialized DllMain, shared sections
Robust DLL design for threads

Debugging with Threads

Querying information about running processes/threads and their attributes
The serialized OutputDebugString API

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

Multithreaded Project

Development of a complete multithreaded embedded HTTP web server that uses I/O completion ports to efficiently manage large numbers of requests