

# 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 communication 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.

<b>Contents of One-Day Training Course</b>	
<p><b>Target Audience</b> System architects and experienced developers who need to gain an in-depth understanding of Windows multithreading.</p> <p><b>Prerequisites</b> Attendees must have good experience of system-level programming, on either Windows or Linux.</p>	<p style="text-align: center;"><b>Thread definition</b></p> <p>Scheduling vs. synchronization Parallelism and concurrency Compute-bound and I/O bound apps Race conditions, deadlock, starvation, priority inversion</p> <p style="text-align: center;"><b>Kernel Objects</b></p> <p>Windows kernel objects Usage counting Kernel object handles Sharing handles among processes</p> <p style="text-align: center;"><b>Processes</b></p> <p>CreateProcess API &amp; child processes Retrieving the exit code of a process Job objects</p> <p style="text-align: center;"><b>Threads</b></p> <p>CreateThread API &amp; Threadproc The C runtime library Thread priority &amp; processor affinity Thread management &amp; lifetime</p> <p style="text-align: center;"><b>Synchronization</b></p> <p>Critical Sections, Mutexes, Events, Semaphores, Waitable Timers, WaitForSingle/MultipleObject The “Protect data, not code” principle</p> <p style="text-align: center;"><b>Memory and Threads</b></p> <p>Dynamic &amp; static Thread Local Storage Heap storage vs. stack storage</p> <p style="text-align: center;"><b>Asynchronous I/O</b></p> <p>Overlapped, APC &amp; Scatter/Gather I/O Completion Ports, Asynchronous I/O, Overlapped, APC &amp; Scatter/Gather I/O Completion Ports</p>
	<p style="text-align: center;"><b>Thread Pools</b></p> <p>OS-managed pools of threads for processing timers, work-items and I/O</p> <p style="text-align: center;"><b>DLLs and Threads</b></p> <p>How threads interact with DLLs Serialized DllMain, shared sections Robust DLL design for threads</p> <p style="text-align: center;"><b>Debugging with Threads</b></p> <p>Querying information about running processes/threads and their attributes The serialized OutputDebugString API</p> <p style="text-align: center;"><b>Resource Management</b></p> <p>Creating a custom resource browser, to display which thread is waiting on which synchronization resource Threads with C++ Threads &amp; exceptions; threads &amp; classes Accessing resources using smart pointers</p> <p style="text-align: center;"><b>Design Issues</b></p> <p>Single Writer/Multiple Readers, Monitor, Once-Off Initialization, Dining Philosopher Calling legacy code from multiple threads Converting legacy code to multithreading</p> <p style="text-align: center;"><b>Multithreaded Architectures</b></p> <p>Pipeline, Producer-Consumer, Work-Crew Master-Slave Models Create threads on demand vs. elastic pool</p> <p style="text-align: center;"><b>Multithreaded Project</b></p> <p>Development of a complete multithreaded embedded HTTP web server that uses I/O completion ports to efficiently manage large numbers of requests</p>