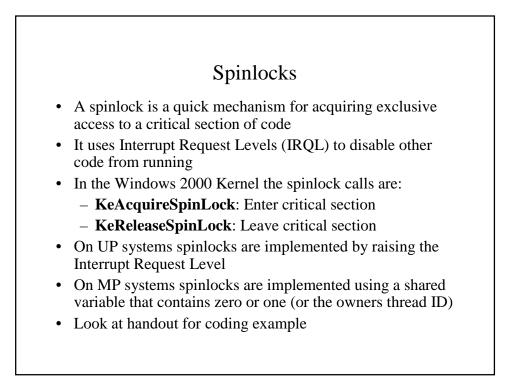


• In particular we're going to look at Windows 2000

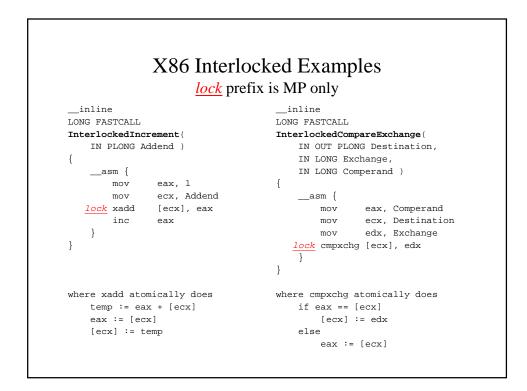
Additional Synchronization Routines

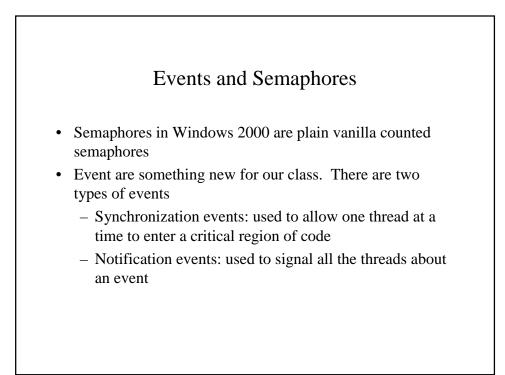
- First two very specialized primitives
 - Spinlocks
 - Interlocked Operations
- Three general kernel synchronization objects
 - Events
 - Semaphores
 - Mutex
- Lastly one general purpose synchronization package
 - Eresource
- Some are kernel mode only
- Some are implemented in kernel mode and exported to the user as a system call
- Some are implemented in both kernel and user mode

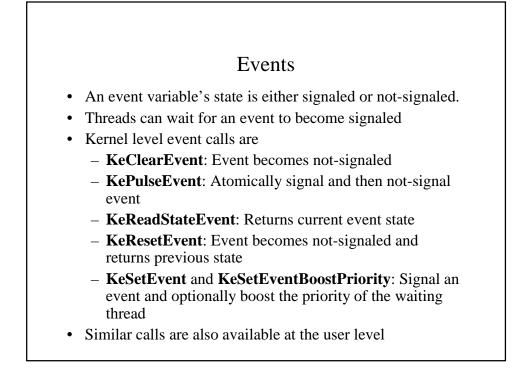




- Interlocked operations are atomic operations that can be used to synchronize access to a single variable or as a building block for more complex synchronization routines
- · Some of the supported interlocked operations are
 - InterlockedIncrement
 - InterlockedDecrement
 - InterlockedCompareExchange
 - and there are more

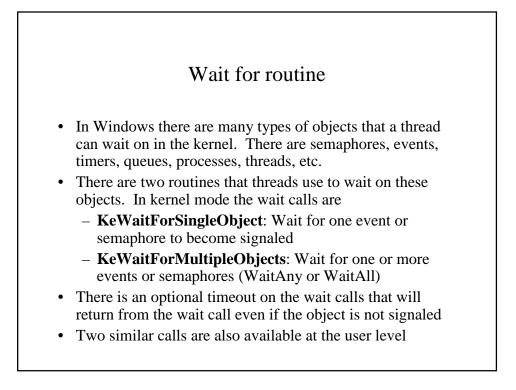


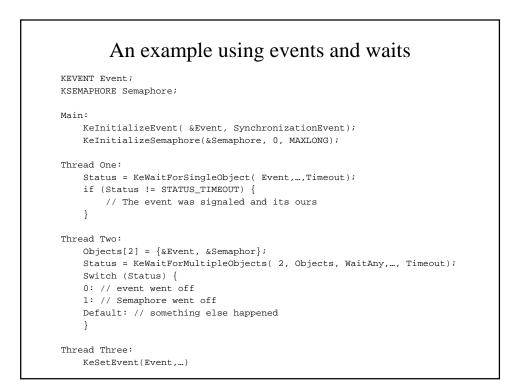


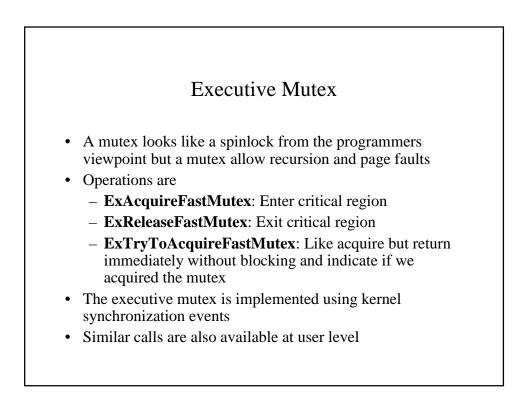


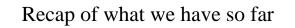
Semaphores

- A semaphore variable is just a plain old vanilla counted semaphore, but its implementation shares much of the event logic
- Kernel semaphore calls are
 - KeReadStateSemaphore: Returns the signal state of the semaphore
 - **KeReleaseSemaphore**: Takes as input a release count and a priority boost of the newly released threads
- This is also a set of calls available at user level

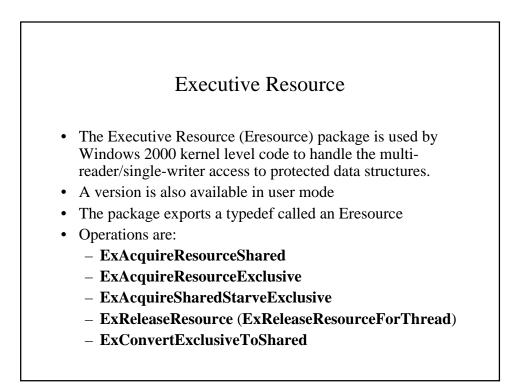








- Spinlocks in theory this can be done at the user level however for practical purposes it rarely is done at that level. Great for protecting short little critical sections. Usually we want everything in memory (I.e., we don't really want to wait while holding a spinlock)
- Interlocked Operations usable in both user and kernel mode. Sometimes difficult to use but a great building block none the less
- Semaphore Both user and kernel mode. Pretty good for general synchronization
- Events Both user and kernel mode. More specialized than semaphores
- Mutex Both user and kernel mode. More usable than spinlocks but also more expensive to use



Eresource features

- A resource can be acquired recursively
- The ownership of a resource is tied to a thread ID
- The users get to decide whether exclusive waiters starve
- The package automatically handles priority inversion
- The routines are implemented using spinlocks, events, and semaphores
- More details at a later lecture if time permits

