CSE 451: Operating Systems
Hard Lessons Learned

Windows
Reader/Writer Locks

Gary Kimura

But first some Truth in advertising Wait()

- Wait() in Windows comes in many flavors and is not as simple was we've made it out to seem.
- You can wait() for a single, any, or multiple events/objects and not just locks
- You can optionally specify a timeout period
- When returning from a wait you therefore need to check why wait() returned.

Without going into great details
A brief look at deadlocks and starvation

- In lay terms a Deadlock is when a thread holds a lock (lock1) and is waiting for another lock (lock2) that it will never get because a second thread holds lock2 and is waiting to get lock1.
  - Circular wait. Aka deadly embrace.
  - Deadlocked threads are typically in the blocked state.
  - Root cause is often how one uses (misuses) locks

Without going into great details
A brief look at deadlocks and starvation

- In lay terms Starvation is when a thread is ready to run but because of scheduling peculiarities it never gets a chance to run, most likely because there is a higher priority thread always running.
  - Starved threads are typically stuck in the ready queue.
  - A problem mostly blamed on the scheduler.

Priority Inversion and starvation

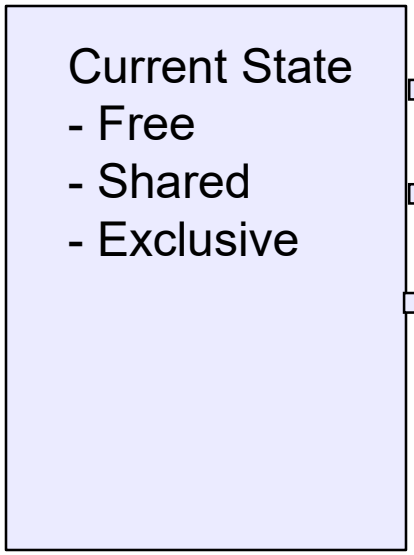
- In lay terms Priority Inversion is when a high priority thread is waiting for a lock owned by a lower priority thread that cannot make progress because it is being starved.
- Example using Undergraduate, Graduate, and Professor waiting to get coffee.

- One solution is to do a priority boost.
- Note: This is not practical using monitors.

Windows Readers/Writers nuances

- Call EResource in Windows.
- Used the terms exclusive and shared access.
- Avoided starving exclusive by making shared requests wait
- Allowed recursive acquisition of a lock. Meant keeping ownership information
- Addressed an issue called priority inversion
- Then one hack added after another.
  - Added call to “Try” to acquire access without blocking
  - Added call to starve an exclusive waiter
  - Added call to release lock for a different thread
  - Augh…

Picture of the resource



List of current owners with their count

List of waiting readers/shared

List of waiting writers/exclusive

State Transitions

- Free -> Shared
- Free -> Exclusive
- Shared -> Free
- Exclusive -> Free
- Shared -> Exclusive
- Exclusive -> Shared

Where we started

- ExInitializeResource
- ExAcquireResourceShared
  - If currently free, then it's yours
  - If currently exclusive, then you wait
  - If currently shared and you already have it, then up your count
  - If currently shared and there are no exclusive waiters then you it's yours
  - If currently shared and there is an exclusive waiter, then you wait
- ExAcquireResourceExclusive
  - Similar logic as above
- ExReleaseResource
  - The usual logic but now Ping-Pong back-and-forth between shared and exclusive if necessary

Added “features?”

- ExAcquireResourceShared(Wait);
- ExAcquireResourceExclusive(Wait);

- ExAcquireSharedStarveExclusive
- ExReleaseResourceForThread

- ExConvertExclusiveToShared
- ExDisableResourceBoost

- ExReinitializeResource
- ExSetResourceOwnerPointer
- ExDeleteResource

More added “features?”

- ExGetExclusiveWaiterCount
- ExGetSharedWaiterCount

- ExIsResourceAcquiredExclusive
- ExIsResourceAcquiredShared

- Bottom line: Learning to say “NO” to requests for adding new features.