Why not just share a single event across all critical section?

devblogs.microsoft.com/oldnewthing/20200323-00

March 23, 2020

Raymond Chen

Neil Rashbrook wonder why there isn't a single event that all critical sections share. When a critical section is exited, and there is a waiting thread, then "signal all threads waiting on critical sections". One thread will get the critical section, and threads that are waiting for unrelated critical sections will just think that they lost a nonexistent race and go back to sleep.

There are a few problems with this idea.

The first problem is specific to the way Windows kernel events work. How would you "signal all threads waiting on critical sections" with a kernel event? If you use an automatic-reset event, then only one thread will wake. In order to wake all threads, you need a manual-reset event. But how do you know when to reset the event? You want to reset the event once all threads have woken up, but you have no way of knowing when that has happened.

You might think of using `PulseEvent` and then realize that you have no easy way of closing the race condition between a thread realizing that it needs to go to sleep and the actual sleep. You would have to use something like `SignalObjectAndWait`, but that means that every critical section acquisition would need to take a kernel mutex so it could prevent the owner from pulsing the event before the waiter could reach the `WaitForSingleObject` on the kernel event. But if you're going to take a kernel mutex on every acquisition, then you destroyed the purpose of the critical section, which is to be a lightweight alternative to a kernel mutex! You may as well just use a kernel mutex as your critical section.

And all that is on top of the fact that `PulseEvent` is fundamentally flawed.

Now, maybe you could come up with an alternative to a kernel mutex and a kernel event. Say, a global slim reader-writer lock and a paired condition variable. The condition variable lets you `notify_all` in a reliable way, and then every thread checks whether their critical section is available. Could we use that for our critical section design?

I guess you could do that, but you wouldn't want to. Waking every thread that's waiting for a critical section, even though you know that only one will succeed, is the definition of a thundering herd problem. You take a lot of context switches, lose a lot of CPU cache, page in a lot of memory, and waste a lot of CPU time with all the pointless work. You really want to wake just one candidate thread and leave the others sleeping. That's why our critical section built out of `WaitOnAddress` uses `WakeByAddressSingle`.

Note that the candidate thread you wake up may not actually succeded at claiming the critical section, because another thread may sneak in and claim the critical section before your candidate can wake up. Although this sounds unfair, unfairness is actually a feature, because it avoids lock convoys.

Raymond Chen

Follow