Creating a semaphore from WaitOnAddress

devblogs.microsoft.com/oldnewthing/20170612-00

June 12, 2017

Raymond Chen

Some time ago, we explored creating various types of well-known synchronization objects from `WaitOnAddress`. Today we'll create a semaphore with no maximum token count. (Believe it not, I'm building up to something, but it'll take a while.)

```
struct ALT_SEMAPHORE
{
  LONG TokenCount;
};

void InitializeAltSemaphore(ALT_SEMAPHORE* Semaphore,
                            LONG InitialCount)
{
  Semaphore->TokenCount = InitialCount;
}

void ReleaseAltSemaphoreOnce(ALT_SEMAPHORE* Semaphore)
{
  InterlockedIncrement(&Sempahore->TokenCount);
  WakeByAddressSingle(&Sempahore->TokenCount);
}

void ReleaseAltSemaphoreMulti(ALT_SEMAPHORE* Semaphore,
                              LONG ReleaseCount)
{
  InterlockedAdd(&Sempahore->TokenCount, ReleaseCount);
  if (ReleaseCount == 1) {
    WakeByAddressSingle(&Sempahore->TokenCount);
  } else {
    WakeByAddressAll(&Sempahore->TokenCount);
  }
}

void WaitForAltSempahore(ALT_SEMAPHORE* Semaphore)
{
  while (true) {
    LONG OriginalCount = Semaphore->TokenCount;
    while (OriginalCount == 0) {
      WaitOnAddress(&Semaphore->TokenCount,
                    &OriginalCount,
                    sizeof(OriginalCount),
                    INFINITE);
      OriginalCount = Semaphore->TokenCount;
    }
    if (InterlockedCompareExchange(&Semaphore->TokenCount,
                                   OriginalCount - 1,
                                   OriginalCount) == OriginalCount) {
      return;
    }
  }
}
```

The semaphore consists simply of the current token count. To release one token, we increment the token count and wake up any single waiting thread. To release multiple tokens, we increase the token count by the number of tokens being released and the signal all the waiting threads. There is no `WakeByAddressMulti` that lets you wake a specific number

of threads. Your options are to release one or to release all. In the case where we release more than one token, we just have to release all of the waiting threads and let them fight over the tokens.

Claiming a token is the tricky part. There are two parts of claiming a token: Waiting for a token to become available, and claiming it.

To wait for the token to become available, we wait for the token count to be nonzero. If we see that it is zero, we use `WaitOnAddress` to wait until it becomes nonzero. (This relies on `ReleaseAltSemaphoreOnce` remembering to `WakeByAddressSingle` when it changes the token count.) Once the token count changes, we loop back and see if it's nonzero now. Note that this code is resilient to spurious wakes because we always recheck the semaphore count. Actually, the code would have needed to be resilient to spurious wakes anyway, because it's possible that a token got released, but another thread snuck in and stole it from us before we could check it.

Once we confirm that a token is available (or at least was available recently), we try to decrement the token count by one. If that succeeds, then we are done. Otherwise, it means another thread came in and claimed a token before we could claim it, so we loop back to the top and wait for a token to become available.

Note that this synchronization object is not fair. If a token becomes available and a thread is waiting, it's possible for an intruder thread to sneak in and steal the token from the waiting thread. The waiting thread wakes up, sees that there's no token, and goes back to sleep.

Next time, we'll reimplement a semaphore with a maximum count.

Raymond Chen

Follow