Creating a manual-reset event from WaitOnAddress

devblogs.microsoft.com/oldnewthing/20170614-00

June 14, 2017

Raymond Chen

Last time, we created a semaphore with a maximum count from `WaitOnAddress`. Related to semaphores are events, so let's do a manual-reset event.

```
struct ALT_MEVENT
{
  LONG State;
};

void InitializeAltManualEvent(ALT_MEVENT* Event,
                              bool InitialState)
{
  Semaphore->State = InitialState;
}

void SetAltManualEvent(ALT_MEVENT* Event)
{
 if (InterlockedCompareExchange(&Event->State,
                                true, false) == false) {
  WakeByAddressAll(&Event->State);
 }
}

void ResetAltManualEvent(ALT_MEVENT* Event)
{
 InterlockedCompareExchange(&Event->State,
                            false, true);
}

void WaitForAltManualEvent(ALT_MEVENT* Event)
{
 while (!Event->State) {
  LONG Waiting = 0;
  WaitOnAddress(&Event->State,
                &Waiting,
                sizeof(Waiting),
                INFINITE);
 }
}
```

To set the event, we try to transition it from `false` to `true`. If that succeeds, then we wake anybody who was waiting for the event. (If it fails, then it means that the event was already set, so there is nothing to do.) Similarly, to reset the event, we try to transition it from `true` to `false`. In this case, there is no need to signal that the value changed because there will never be anyone waiting for the state to change to `false`.

To wait for the event, we check whether it is currently set. If not, then we use `WaitOnAddress` to wait for its state to change. When we think its state may have changed, we go back and check.

Pretty simple. Next time, the auto-reset event.

Bonus chatter: Although this usage pattern doesn't have anybody waiting for the state to change to `false`, you can imagine a case where you want some type of synchronization object that signals when something becomes unavailable. For example, you might want one side to run as long as the event is signaled, and the other side to run as long as the event is unsignaled. The traditional way is to have a pair of events, and alternately signal them. But with `WaitOnAddress`, we can combine them into a single synchronization object.

Raymond Chen

Follow