When I create a waitable timer with a callback, do I have to wait alertably on that specific timer before the callback will run?

devblogs.microsoft.com/oldnewthing/20221230-00

December 30, 2022

Raymond Chen

A customer had a question about waitable timers. If you create a waitable timer with a callback, the callback runs only on the thread that created the waitable timer, and only when the thread "enters an alertable await state." Does this mean specifically an alertable await state on that specific timer? Or will any alertable wait do?

Any alertable wait will do. You don't even have to wait on any objects. Calling `SleepEx(n, TRUE)` is an alertable wait on zero objects, and that will process Asynchronous Procedure Calls (APCs).

Associated with each thread is a queue of asynchronous callbacks, and many operations append work to that list. There are actually three categories of APC, user APCs, normal kernel APCs, and special kernel APCs.

The two kernel APCs are available only to kernel mode. Kernel APCs are pretty much invisible to user mode, but their side effects may be visible. For example, they contribute to the uselessness of the `PulseEvent` function and threads waiting on a synchronization object waking in non-FIFO order (not that FIFO was ever guaranteed). They are also the mechanism by which the `SuspendThread` function asks threads to suspend, and the delay in transmitting the kernel APC is detectable from user mode.

The third type of APC is the user APC. This one can be created both from kernel mode and user mode. The most common source is I/O completion callbacks, but waitable timers also use this mechanism, and you can create your own by calling the `QueueUserAPC` function.

They are called asynchronous procedure calls because the request is queued up and processed when the thread reaches a state where the corresponding code is ready to be interrupted.

- User APCs run when the thread enters an alertable wait state.

- Normal kernel APCs run when the kernel is about to return to user mode, which means that no kernel code is active on the thread.
- Special kernel APCs[1] run when the kernel is below APC level. This allows them to interrupt normal kernel processing, but not other APCs, hardware interrupts, or other higher-priority events.

The idea behind APCs is that a particular category of APC can interrupt code that is lower in the hierarchy, but it will wait until all other code at the same or higher level in the hierarchy has indicated that they are ready to run the APC.

higher ↑	
Non-passive kernel mode	Special kernel APC
Passive kernel mode	Normal kernel APC
User mode	User APC

Okay, so let's look specifically at user APCs, since those are the ones queued by waitable timers. A thread indicates that it is ready to accept user APCs by calling an alertable wait function. If there is a user APC waiting to run, it runs, and then the alertable wait returns with the code `WAIT_IO_COMPLETION`[2] to indicate "The wait completed due to one or more user APCs." The caller can then choose to reissue the wait, or it might decide to do something else, say, because it really was just waiting for the user APC.

The user APC dispatching code doesn't care who queued the APC or what the alertable wait is waiting for. Once a thread goes into any alertable wait state, the kernel will dispatch any pending user APCs for that thread. It doesn't try to filter only to APCs related to the things that caused you to enter the alertable wait.

Bonus analogy: You can think of a user APC as `PostMessage` and an alertable wait as an unfiltered `PeekMessage` that dispatches all pending queued messages.

[1] Kernel APCs are collectively known as KAPCs, which means that the special kernel APC is the "Special K" APC.

[2] The name of the return code strongly hints that I/O completion was the original use case for user APCs.

Raymond Chen

Follow