On the finer points of cancelling timers and wait objects in Windows thread pool

devblogs.microsoft.com/oldnewthing/20230428-00

April 28, 2023

Raymond Chen

The Windows thread pool lets you create, among other things, timers and waits, and you can cancel them, too.

There are some finer points here.

The way you cancel a timer is to call `SetThreadpoolTimer` or `SetThreadpoolTimerEx` with a null pointer as the due time. And the way you cancel a wait is to call `SetThreadpool-Wait` or `SetThreadpoolWaitEx` with a null handle.

When you cancel a timer or wait, no future callbacks will be created, but existing ones are not recalled. Therefore, there's a race condition if you issue you your cancellation just after the timer or wait has triggered: You cancelled it too late to prevent the callback from being scheduled, and you can't detect this race from your callback because the callback may not have started running yet. This could be because the callback has been scheduled but hasn't yet been given a thread to run on yet. Or it could be that your callback was unluckily pre-empted at its very first instruction.

Thread pool	You
Object is signaled / timer is ready Callback scheduled	
	Cancel timer / wait
Thread assigned to callback Callback runs	

A timer or wait is created in the "unset" state. You put into the "set" state by calling one of the `Set...` functions with a non-null due time or handle. You return it to the "unset" state by calling the the `Set...` functions with a null due time or handle.

When you call one of the `Set...Ex` functions, it returns a value that tells you whether a callback was cancelled. But that's only part of what you need to know in order to determine whether a callback is on its way. You also need to know whether the timer or wait was previously set.

Here's a table of the possibilities:

Previous state	**Was callback pending**	**Could callback be recalled**	**Set returns…**
Unset	N/A	N/A	`FALSE`
Set	No	N/A	`TRUE`
Set	Yes	Yes	`TRUE`
Set	Yes	No	`FALSE`

The last row is the interesting one: When you cancel the timer or wait object, the thread pool tries to recall any pending callbacks, but sometimes a callback has already gone too far and could not be recalled. For example, the callback could be already in progress. In that case, the `Set...Ex` function returns `FALSE` to tell you that you're not finished yet. You have to wait for the callback to complete before everything is finally done.

The way you wait for the callback to complete is to call `WaitForThreadpoolTimer-Callbacks` or `WaitForThreadpoolWaitCallbacks`. It will wait for the completion of all outstanding callbacks for the specified timer or wait. A callback is deemed to have completed when it returns, or when it calls `DissociateCurrentThreadFromCallback`. We can add this as another column to our table:

Previous state	**Was callback pending**	**Could callback be recalled**	**Set returns…**	**Wait returns…**
Unset	N/A	N/A	`FALSE`	Immediately
Set	No	N/A	`TRUE`	Immediately
Set	Yes	Yes	`TRUE`	Immediately
Set	Yes	No	`FALSE`	After callback completes

Fortunately, the `WaitFor...Callbacks` functions wait in exactly the case we need them to wait.

We can now put this information to use: Suppose you have a thread pool timer or wait that has been set, and you later realize that you don't want to wait that long after all. What is the pattern for safely accelerating the callback?

If we assume that the timer or wait is definitely set, then the *Unset* row is removed from consideration, and that means that a `FALSE` return value from `Set...Ex` tells us that a callback is in progress, or at least has proceeded past the point of no return. In that case, we don't need to accelerate the callback; it's already on its way.

Otherwise, there was no callback in progress, so we need to make one. We can do that by resetting the timer or wait callback with a timeout of zero (which means *now*).

```
if (SetThreadpoolTimerEx(timer, nullptr, 0, 0)) {
    FILETIME now = { 0, 0 };
    SetThreadpoolTimer(timer, &now, 0, 0);
}

if (SetThreadpoolWaitEx(wait, nullptr, nullptr, nullptr)) {
    FILETIME now = { 0, 0 };
    SetThreadpoolWait(wait, GetCurrentProcess(), &now);
}
```

In the case of making a wait callback run immediately, we have to give it a non-null handle, although we don't care what it is. We use the pseudo-handle to the process itself, which the process will never observe as signaled. That way, the callback is made with the report that the wait timed out.