All Windows threadpool waits can now be handled by a single thread

devblogs.microsoft.com/oldnewthing/20220406-00

April 6, 2022

Raymond Chen

I noted some time ago that creating a threadpool wait allows the threadpool to combine multiple waits, so that each thread waits for nearly 64 objects. (It's not quite 64 objects because one of the objects is a special sentinel object that means "Stop waiting.")

In the time since I wrote that article, the situation has gotten even better. Starting in Windows 8, the registered waits are associated with a completion port, and a single thread handles all the wait requests by waiting for completions.

We can see the new behavior in action with this simple program:

```
#include <windows.h>
#include <stdio.h>

int main()
{
    static LONG count = 0;
    HANDLE last = CreateEvent(nullptr, true, false, nullptr);

    HANDLE event = last;
    for (int i = 0; i < 10000; i++)
    {
        auto wait = CreateThreadpoolWait(
        [](auto, auto event, auto, auto)
        {
            InterlockedIncrement(&count);
            SetEvent(event);
        }, event, nullptr);
        event = CreateEvent(nullptr, true, false, nullptr);
        SetThreadpoolWait(wait, event, nullptr);
    }

    Sleep(10000);
    SetEvent(event);
    WaitForSingleObject(last, INFINITE);
    printf("%d events signaled\n", count);
    return 0;
}
```

This quick-and-dirty program creates 10,000 threadpool waits, each waiting on a different event, and whose callback signals the next event, creating a chain of waits that eventually lead to setting the event named `last`. Under the old rules, creating 10,000 threadpool waits would result in around 10,000 ÷ 63 ≅ 232 threads to wait on all of those objects. But if you break into the debugger during the `Sleep()`, you'll see that there are just a few. And if you set a breakpoint at the start of the `main` function, you'll see that only one of those threads was created as a result of the threadpool waits; the others were pre-existing.

To prove that all of these waits really are waiting, we signal the most recent one, which sets off a chain of `SetEvent` calls, and wait for the last event to be set. We print the number of events that were signaled (should be 10,000) and call it a day.

This is just a proof of concept to show the thread behavior, so I didn't bother cleaning up the waits or the handles.

Raymond Chen

Follow