The case of the APC that never arrives

devblogs.microsoft.com/oldnewthing/20220909-00

September 9, 2022

Raymond Chen

A customer encountered found that sometimes, their application hung in its clean up code. Here's a simplified version.

```
bool ShuttingDown = false;

void MainThread()
{
    DWORD id;
    auto hThread = CreateThread(nullptr, 0, WorkerThread,
                                nullptr, 0, &id); // succeeds

    BlahBlahBlah(); // do useful work

    // Time to clean up. Post an APC to the worker thread
    // to tell it that it's time to go home.
    QueueUserAPC(WakeWorker, hThread, 0); // succeeds

    WaitForSingleObject(hThread, INFINITE); // hangs

    CloseHandle(hThread);
}

void CALLBACK WakeWorker(ULONG_PTR)
{
    ShuttingDown = true;
}

DWORD CALLBACK WorkerThread(void*)
{
    // Do work until shut down.
    do
    {
        // All work is posted via APCs.
        SleepEx(INFINITE, TRUE);
    } while (!ShuttingDown);

    return 0;
}
```

The idea is that the program has a worker thread to, y'know, do some work. All of the work items are posted via `QueueUserAPC`, and the worker thread simply calls `SleepEx` alertably, over and over again. Each call to `SleepEx` sleeps the thread until an APC is queued, at which point it returns (because the `bAlertable` parameter is `TRUE`).

One way of looking at this design is that it's a sneaky way of making the operating system manage your work queue for you. Another way of looking at it is as a single-threaded I/O completion port.

Call it what you will.

Anyway, the problem is that the worker thread is stuck in `SleepEx`, as if it never got the `WakeWorker` APC that tells it to exit.

But is that really the problem?

The customer was able to get some full memory dumps of systems that got into this state, and the telling detail is that the `ShuttingDown` variable was set to `true`. So it wasn't that the `WakeWorker` APC never arrived. It totally did arrive and set `ShuttingDown` to `true`. Yet somehow that didn't wake up the `SleepEx`.

One of my colleagues offered the possibility that when the code entered the `SleepEx` loop, the `WakeWorker` APC had *already run*. If that's the case, then the `SleepEx` is going to wait for an APC that has already arrived.

I was able to confirm with a test program that this was indeed a possibility.

```
DWORD apc = 0;

void CALLBACK WakeWorker(ULONG_PTR)
{
    apc = GetCurrentThreadId();
}

DWORD CALLBACK WorkerThread(void*)
{
    if (apc == GetCurrentThreadId()) DebugBreak();
    return 0;
}

int __cdecl main()
{
    DWORD id;
    while (true)
    {
        apc = 0;
        auto h = CreateThread(nullptr, 0, WorkerThread, nullptr, 0, &id);
        QueueUserAPC(WakeWorker, h, 0);
        WaitForSingleObject(h, INFINITE);
        CloseHandle(h);
        Sleep(10);
    }
    // notreached
}
```

This program creates a thread and immediately queues an APC to it. The thread procedure checks if the APC already ran on the same thread, and if so, it breaks into the debugger.

If you run this program, it breaks immediately. If you set a breakpoint on the `WakeWorker` function, you'll see that it does indeed run on the thread before the `WorkerThread` function starts.

From the stack trace on that breakpoint, it appears that the kernel processes APCs during the early phases of thread creation, so if you had any queued APCs, they will get processed before the thread procedure even starts.

So that's the bug: The `ShuttingDown` flag was already set by the time the thread procedure started, but the code assumes that it can only be set by an APC that is processed by the `SleepEx`.

The fix is simple: Change the loop from a `do...while` to a `while`, so that the test is at the top of the loop instead of at the bottom.

```
DWORD CALLBACK WorkerThread(void*)
{
    // Do work until shut down.
    while (!ShuttingDown)
    {
        // All work is posted via APCs.
        SleepEx(INFINITE, TRUE);
    }

    return 0;
}
```

The case where this happens is if the `BlahBlahBlah()` function returns very quickly, allowing the `QueueUserAPC` to win the race against the `WorkerThread`.

Raymond Chen

Follow