How can I force a WriteFile or ReadFile to complete synchronously or hang, in order to test something?

devblogs.microsoft.com/oldnewthing/20220425-00

April 25, 2022

Raymond Chen

Some time ago, I showed how to force a `CreateFile` call to hang, for testing purposes. Today we'll show how to force a `WriteFile` call on an overlapped handle to complete synchronously. Forcing various types of behaviors are handy when you want to make sure your code is covering all the possible cases.

If I had written this a few years ago, I would have taken advantage of the behavior that writes which extend a file are synchronous. I would have done so with the understanding that the behavior is not contractual, and maybe someday extending writes would become asynchronous, so my unit test would also have asserted that the call did in fact complete synchronously, so if the behavior changed in the future, the unit test alert alert me that it needs to be updated.

But that was a few years ago. In the meantime, I learned that the days of asynchronous file-extending writes are here. So a file-extending write is no longer sufficient. I'm going to have to find another way.

That other way is a named pipe.

The idea here is that writes to a named pipe complete synchronously if there is sufficient space in the write buffer to hold the data. The data goes into the write buffer, and it sits there until the reader pulls it out.

Here's a scratch program to demonstrate. All error checking has been elided.

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

int __cdecl wmain(int argc, wchar_t** argv)
{
    auto write = CreateNamedPipeW(LR"(\\.\PIPE\fred)",
        PIPE_ACCESS_OUTBOUND | FILE_FLAG_OVERLAPPED,
        PIPE_TYPE_MESSAGE | PIPE_REJECT_REMOTE_CLIENTS, 1, 1024, 1024,
        0, nullptr);

    // Set up a pending read.
    auto read = CreateFileW(LR"(\\.\PIPE\fred)", GENERIC_READ,
        FILE_SHARE_READ | FILE_SHARE_WRITE, nullptr, OPEN_EXISTING,
        FILE_FLAG_OVERLAPPED, nullptr);
    OVERLAPPED oRead{};
    oRead.hEvent = CreateEvent(nullptr, TRUE, FALSE, nullptr);
    BYTE readBuffer[1024];
    DWORD actual;
    auto readRc = ReadFile(read, readBuffer, 1024, &actual, &oRead);
    auto readPending = !readRc && GetLastError() == ERROR_IO_PENDING;

    // Can now connect the two sides.
    ConnectNamedPipe(write, nullptr);

    // Now perform the write.
    BYTE writeBuffer[1] = { 0 };
    OVERLAPPED oWrite{};
    oWrite.hEvent = CreateEvent(nullptr, TRUE, FALSE, nullptr);
    auto writeRc = WriteFile(write, writeBuffer, 1, &actual, &oWrite);
    if (!writeRc && GetLastError() != ERROR_IO_PENDING) {
        printf("I/O failed\n");
    } else {
        printf("Completed synchronously? %d\n", writeRc);
        GetOverlappedResult(write, &oWrite, &actual, TRUE);
    }

    // Clean up the pending read.
    CancelIo(read);
    GetOverlappedResult(read, &oRead, &actual, TRUE);

    CloseHandle(write);
    CloseHandle(read);

    return 0;
}
```

We create a named pipe and obtain handles to the read and write ends of the pipe. We set up an overlapped read on the pipe so that the pipe is ready to accept data. Next, we perform the write, and report whether it completed synchronously. (We expect it to.) Finally, we clean up.

This also shows how we can force a `ReadFile` to hang: The `ReadFile` we perform on the pipe will remain pending until something is written into the pipe.

This pattern also shows how we can force a write to pend: Create a named pipe with no buffering (pass zero as the buffer sizes), issue a read for 1 byte so that the pipe can be connected, and then try to write 10 bytes. The write will pend until all 10 bytes are read out.

And you can force a read to complete synchronously by reading from a pipe that already has data in its read buffer.

Named pipes are a pretty handy way of forcing these sorts of edge cases, since you can control how the other end of the pipe behaves.

Raymond Chen

Follow