The mental model for StartThreadpoollo

devblogs.microsoft.com/oldnewthing/20211117-00

November 17, 2021

Raymond Chen

A customer was having trouble using asynchronous I/O with the Windows thread pool. Their proof-of-concept program was crashing once they issue their second write. Here's a sketch:

```
auto io = CreateThreadpoolIo(fileHandle, callback, nullptr, nullptr);
StartThreadpoolIo(io);

OVERLAPPED pending[NUMBER] = {};

for (int i = 0; i < NUMBER; i++) {
    pending[i].Offset = offset[i];
    pending[i].OffsetHigh = 0;
    bool result = WriteFile(fileHandle, data[i], size[i],
            &bytesWritten, &pending[i]);

    if (!result && GetLastError() != ERROR_IO_PENDING) {
        CancelThreadpoolIo(io);
    }
}
```

They found that if `NUMBER` is 1, then everything works great. If `NUMBER` is greater than 1, then the first I/O completion is successful, but the second one crashes.

The confusion is over what `StartThreadpoolIo` does. The customer thought that it needed to be called once for each file handle. But really, it needs to be called once for each I/O operation. If you issue ten writes against a file handle, you need to call `StartThreadpoolIo` ten times, once before each call.

The point of `StartThreadpoolIo` is to tell the thread pool to prepare for an incoming completion against the file handle. If you issue an I/O without first preparing the thread pool, then the completion arrives and the thread pool doesn't know what to do with it.

The fix is to move the call to `StartThreadpoolIo` to immediately before issuing the I/O operation:

```
auto io = CreateThreadpoolIo(fileHandle, callback, nullptr, nullptr);
// StartThreadpoolIo(io); // from here

OVERLAPPED pending[NUMBER] = {};

for (int i = 0; i < NUMBER; i++) {
    pending[i].Offset = offset[i];
    pending[i].OffsetHigh = 0;
    StartThreadpoolIo(io); // to here
    bool result = WriteFile(fileHandle, data[i], size[i],
            &bytesWritten, &pending[i]);

    if (!result && GetLastError() != ERROR_IO_PENDING) {
        CancelThreadpoolIo(io);
    }
}
```

If you discover that the I/O won't generate a completion after all (because it failed synchronously, or because it succeeded synchronously on a handle that is marked as `FILE_SKIP_COMPLETION_PORT_ON_SUCCESS`), then you need to call `CancelThreadpoolIo` to say, "Um, it looks like there won't be any completion at all. Sorry." That way, the thread pool knows that it shouldn't be expecting one.

Raymond Chen

Follow