What happens if you forget to pass an OVERLAPPED structure on an asynchronous handle?

devblogs.microsoft.com/oldnewthing/20121012-00

October 12, 2012

Raymond Chen

A customer noticed that they were accidentally issuing some I/O's against an overlapped handle without using an `OVERLAPPED` structure. The calls seemed to work, and the program did not appear to suffer any ill effects, but they were worried that they just being lucky and that eventually the error will come back to bite them. So what really happens if you forget to pass an `OVERLAPPED` structure on an asynchronous handle? Well, the layer of the kernel that deals with `OVERLAPPED` structures doesn't know whether then handle is synchronous or asynchronous. It just assumes that if you don't pass an `OVERLAPPED` structure, then the handle is synchronous. And the way it deals with synchronous I/O without an `OVERLAPPED` is that it creates a temporary `OVERLAPPED` structure on the stack with a null `hEvent`, issues an asynchronous I/O with that temporary `OVERLAPPED` structure, and then waits for completion with `GetOverlappedResult(bWait = TRUE)`. It then returns the result. What does this mean for you? Well, what happens if the `hEvent` is null?

> If the **hEvent** member of the **OVERLAPPED** structure is NULL, the system uses the state of the **hFile** handle to signal when the operation has been completed.

Okay, let's step back and look at what's going on here. First of all, a file handle is a waitable object: It becomes unsignaled when an I/O operation begins, and it becomes signaled when an I/O operation ends. Second of all, this behavior is not useful in general. If you are operating on a synchronous handle, you already know when the I/O operation ends: It ends when the synchronous I/O call returns. And if you are operating on an asynchronous handle, all the `hFile` tells you is that *some* I/O completed, but you don't know which one it was. That's why the documentation also says

> Use of file, named pipe, or communications-device handles for this purpose is discouraged. It is safer to use an event object because of the confusion that can occur when multiple simultaneous overlapped operations are performed on the same file, named pipe, or communications device. In this situation, there is no way to know which operation caused the object's state to be signaled.

What's more, if somebody initiates a new I/O after your asynchronous I/O completed, the file object becomes unsignaled, and there's a possibility that this happened before you got a change to call `WaitForSingleObject`. So why have this weird behavior if it's not useful in general? Because it's what the system itself uses internally to implement synchronous I/O! It issues the I/O asynchronously, then waits on the handle. Since the handle is synchronous, the system already knows that there can be only one I/O in progress at a time, so it can just wait on the `hFile` to know when that I/O is complete. Okay, so let's look again at the case of the overlapped I/O issued with no `OVERLAPPED` structure. The layer that deals with `OVERLAPPED` structure assumes it has a synchronous handle and issues an asynchronous I/O, then waits until the handle is signaled, under the mistaken belief that the handle will be signaled when that I/O completes (since it "knows" that that's the only outstanding I/O request). But if your handle is actually asynchronous, what happens is that the `OVERLAPPED` layer waits on the `hFile`, and the call returns when *any* I/O on that handle completes. In other words, you're in the "… is discouraged" part of the documentation. Theoretically speaking, then, it is legal to pass `NULL` as the `lpOverlapped` to `ReadFile` when the handle is asynchronous, but the results may not be what you want, since the call may return prematurely if there is other I/O going on at the same time. And then when the I/O actually completes, it updates the `OVERLAPPED` structure that was created temporarily on the stack, and we saw that that leads to memory corruption that goes away when you try to debug it. There are those who argue that the documentation for `ReadFile` is overly cautious when it outright bans the use of a null `lpOverlapped` on asynchronous handles, because if you are *really careful*, you can get it to work, if you can guarantee that no I/O is outstanding on the handle at the time you issue your I/O call, and no other I/O will be issued against the handle while you're waiting for your call to complete. I'm of the camp that it's like telling people that it's okay to change gears on your manual transmission by just slamming the gear stick into position without using the clutch. Yes, you can do it if you are really careful and get everything to align just right, but if you mess up, your transmission explodes and spews parts all over the road. In the customer's case they were issuing the I/O without an `OVERLAPPED` structure after the handle was created and before asynchronous operations began, so it was indeed the case that nobody else was using the handle.[1] The usage was therefore technically safe, but the customer nevertheless chose to switch to using an explicit `OVERLAPPED` structure with an explicit `hEvent`, just in case future code changes resulted in asynchronous operations being performed on the handle at an earlier point. (Wise choice on the customer's part. Safety first!)

[1] We're assuming that there aren't any bugs that result in somebody using a handle after closing it or using an uninitialized handle variable. Even if that assumption isn't true, it would also cause problems even in the case where we we passed an explicit `OVERLAPPED` structure, so it's no buggier than it was before.

Raymond Chen

Follow