Providing a small amount of additional control over what happens when an asynchronous I/O completes synchronously

devblogs.microsoft.com/oldnewthing/20190719-00

July 19, 2019

Raymond Chen

The `SetFileCompletionNotificationModes` function gives you some control over how the system behaves when an overlapped I/O completes synchronously.

The `FILE_SKIP_COMPLETION_PORT_ON_SUCCESS` flag specifies that a completion is not queued to the associated I/O completion port in the case the call completes synchronously. In that case, you can reclaim the `OVERLAPPED` structure immediately, since that is the last you will ever know about the I/O. Normally, there would be a completion queued to the completion port, but since you suppressed even that, there's nothing more going on.

The other flag is more confusing. The `FILE_SKIP_SET_EVENT_ON_HANDLE` flag specifies that the I/O Manager "does not set the event for the file object."

The phrase "the event for the file object" was written by someone wearing kernel-colored glasses.

There is a secret event inside every file object. You cannot access this event directly, but you can observe it by calling a function like `WaitForSingleObject` and passing it a file handle. When you ask to wait on a file handle, the kernel waits on the secret event hiding inside the file object.

This secret event is reset when the file operation starts, and it is set when the file operation completes. This secret event is internally how the kernel implements I/O to synchronous file handles: It issues an asynchronous I/O operation, and then waits on the file object.

You too can use this secret event by issuing an asynchronous I/O operation with `OVERLAPPED.hEvent = nullptr`, and then waiting directly on the file handle. This is also why the `GetOverlappedResult` function takes a handle as its first parameter: It waits on the handle if the `OVERLAPPED` structure didn't have an event handle in it.

Of course, if you choose to rely on the secret kernel event, it is your own responsibility to make sure you don't have more than one I/O operation outstanding at a time, because there is only one secret event. If you mistakenly have two I/O operations outstanding, then the first one to complete will set the secret event, and the second one to complete will set the already-set event, which has no effect.

The documentation for the `FILE_SKIP_SET_EVENT_ON_HANDLE` flag is talking about the secret event. It says that if you enable this feature on a handle, then the secret event will not be set if the operation returns success (indicating synchronous completion) or if it returns `ERROR_IO_PENDING` (indicating that the operation is continuing asynchronously).

It's hard for me to come up with a scenario where you would even need to do something like this. The secret event is not something that is widely used, and given the fact that you have to exercise special care to ensure you have at most one outstanding operation at a time, using this secret event seems to be more trouble than it's worth.

Bonus chatter: The documentation continues: "If an explicit event is provided for the request, it is still signaled." The explicit event being referred to here is the event passed in the `OVERLAPPED.hEvent` member. That event is always set, regardless of whether the operation completed synchronously or asynchronously, and regardless of whether the `FILE_SKIP_SET_EVENT_ON_HANDLE` feature was enabled on the file handle.

Raymond Chen

Follow