What are the potentially-erroneous results if you don't pass NULL as the lpNumberOfBytesRead when issuing overlapped I/O?

devblogs.microsoft.com/oldnewthing/20230215-00

February 15, 2023

Raymond Chen

The documentation for many I/O functions that read or write bytes recommend that you pass `NULL` as the `lpNumberOfBytesRead` parameter when issuing overlapped I/O to avoid "potentially erroneous results". What are these potentially erroneous results the documentation is trying to warn against?

For the purpose of this discussion, let's use `ReadFile` as our example, even though the same argument applies to `WriteFile`, `WSASend`, and other functions which follow the same pattern.

The race condition here is a race between the code that calls `ReadFile` and the code that handles the asynchronous completion. If the variable passed as the output for `ReadFile`'s `lpNumberOfBytesRead` parameter is the same variable used as the output for `GetOverlappedResult`'s `lpNumberOfBytesTransferred` parameter, then there is a race because the completion might run concurrently with the exit out of `ReadFile`.

Thread 1	Thread 2
`ReadFile(..., &byteCount, ...);`	
`ReadFile` begins	
I/O initiated asynchronously	
	I/O completes asynchronously
	`GetOverlappedResult(..., &byteCount, ...)`
	`GetOverlappedResult` sets `byteCount = result`
set `byteCount = 0`	

```
    SetLastError(ERROR_IO_PENDING);
```

```
    return FALSE;
```

If the I/O completes very quickly, then the completion routine can run before `ReadFile` returns. And then when `ReadFile` tries to report the fact that the I/O was initiated asynchronously, it overwrites the `byteCount` that the completion routine had calculated.

So it's okay to pass a non-null `lpNumberOfBytesRead` to `ReadFile`, even when issuing asynchronous I/O, provided that you do so into a different variable from the one that the completion routine uses.

Normally, however, there's no reason to pass a non-null `lpNumberOfBytesRead` because the result of the operation is going to be handled by the completion function. But there's a case where it is advantageous to use a non-null value, and that's where you have used `SetFileCompletionNotificationModes` to configure the handle as `FILE_SKIP_COMPLETION_PORT_ON_SUCCESS`. In that case, a synchronous completion does not queue a call to the completion function on the I/O completion thread. Instead, the code that called `ReadFile` is expected to deal with the synchronous completion inline. And one of the things it probably wants to know is how many bytes were read, so it would normally call `GetOverlappedResult` to find out. You can avoid that extra call to `GetOverlappedResult` by passing a non-null pointer to `ReadFile` so that in the case of a synchronous completion, you have your answer immediately.

This is admittedly a micro-optimization. One of my colleagues was not aware of this trick and just followed the guidance in the documentation by passing `NULL` and calling `GetOverlappedResult`, and he says that his code can still stream data at 100Gbps over the network despite doing things "inefficiently".

So maybe you're better off not knowing and just following the simple rule of "Use `NULL` when issuing asynchronous I/O." It's easier to explain and easier to remember.