Why doesn’t my asynchronous read operation complete when I close the handle?

devblogs.microsoft.com/oldnewthing/20210730-00

July 30, 2021

Raymond Chen

A customer was using asynchronous I/O with an I/O completion port. At the time they wanted to shut things down, they still had an outstanding asynchronous read. To get things to clean up, they closed the file handle, expecting it to cause the `ReadFile` to complete and fail with an error like `ERROR_INVALID_HANDLE`. But instead, what they found was that the read operation remained outstanding, and nothing completed.

What’s going on?

What’s going on is that when you close the file handle, that decrements an internal reference count on the underlying file object, and that internal reference count is not yet zero, so the file is still open. And where did that extra reference count come from?

From the `ReadFile` operation itself!

In the kernel, one of the things that happens when you pass a handle from an application is that the kernel validates the handle and obtains a reference-counted pointer to the underlying kernel object, which temporarily bumps the object reference count by one. If you close the handle, that drops it back down, but there is still the outstanding reference from the I/O operation, and that outstanding reference won’t go away until the I/O operation completes.

As a side note, closing the handle to an object while there is still outstanding work on that object feels really sketchy to me. It’s getting dangerously close to “destroying an object while simultaneously using it”.

What you need to do is cancel the I/O by calling a function like `CancelIo` or `CancelIoEx`. And in order to cancel the I/O, you need to proide a handle to the file whose I/O you want to cancel.

Another reason not to close that handle yet.

When you cancel the I/O, the I/O will complete with an error code saying that it was cancelled. At that point, you can close the file handle and clean up.

Raymond Chen

Follow