If you're waiting for I/O to complete, it helps if you actually have an I/O to begin with

devblogs.microsoft.com/oldnewthing/20110303-00

March 3, 2011

Raymond Chen

We saw earlier the importance of waiting for I/O to complete before freeing the data structures associated with that I/O. On the other hand, before you start waiting, you have to make sure that you have something to wait *for*.

A customer reported a hang in `GetOverlappedResult` waiting for an I/O to cancel, and the I/O team was brought in to investigate. They looked at the I/O stack and found that the I/O the customer was waiting for was no longer active. The I/O people considered a few possibilities.

- The I/O was active at one point, but when it completed, a driver bug prevented the completion event from being signaled.
- The I/O was active at one point, and the I/O completed, but the program inadvertently called `ResetEvent` on the handle, negating the `SetEvent` performed by the I/O subsystem.
- The I/O was never active in the first place.

These possibilities are in increasing order of likelihood (and, perhaps not coincidentally, decreasing order of relevance to the I/O team).

A closer investigation of the customer's code showed a code path in which the `ReadFile` call was bypassed. When the bypass code path rejoined the mainline code path, the code continued its work for a while, and then if it decided that it was tired of waiting for the read to complete, it performed a `CancelIo` followed by a `GetOverlappedResult` to wait for the cancellation to complete.

If you never issue the I/O, then a wait for the I/O to complete will wait forever, since you're waiting for something that will never happen.

Okay, so maybe this was a dope-slap type of bug. But here's something perhaps a little less self-evident:

```
// there is a flaw in this code - see discussion
// assume operating on a FILE_FLAG_OVERLAPPED file
if (ReadFile(h, ..., &overlapped)) {
 // I/O completed synchronously, as we learned earlier
} else {
 // I/O under way
 ... do stuff ...
 // okay, let's wait for that I/O
 GetOverlappedResult(h, &overlapped, &dwRead, TRUE);
 ...
}
```

The `GetOverlappedResult` call can hang here because the comment “I/O is under way” is overly optimistic: The I/O may never even have gotten started. If it never started, then it will never complete either. You cannot assume that a `FALSE` return from `ReadFile` implies that the I/O is under way. You also have to check that `GetLastError()` returns `ERROR_IO_PENDING`. Otherwise, the I/O failed to start, and you shouldn’t wait for it.

```
// assume operating on a FILE_FLAG_OVERLAPPED file
if (ReadFile(h, ..., &overlapped)) {
 // I/O completed synchronously, as we learned earlier
} else if (GetLastError() == ERROR_IO_PENDING) {
 // I/O under way
 ... do stuff ...
 // okay, let's wait for that I/O
 GetOverlappedResult(h, &overlapped, &dwRead, TRUE);
 ...
} else {
 // I/O failed - don't wait because there's nothing to wait for!
}
```

Raymond Chen

Follow