The subtleties of CreateStreamOnHGlobal, part 3: Suppressing the deletion of a shared HGLOBAL

devblogs.microsoft.com/oldnewthing/20210930-00

September 30, 2021

Raymond Chen

Last time, we looked at what happens if you suppress the deletion of a `HGLOBAL` that the stream created on your behalf. This time, we'll look at what happens if you provide an explicit initial `HGLOBAL`.

If you provide an explicit initial `HGLOBAL` and also pass `fDeleteOnRelease = FALSE`, you create a sort of shared ownership of the `HGLOBAL`. The ownership is shared in the sense that the stream won't delete it when it's done. But really, it's not shared ownership. The stream will manage the `HGLOBAL` as if it owned it. It just skips the part where it frees the memory at destruction. All of the other memory management operations *are still in effect*.

In particular, the stream assumes that it is the only one who will lock and unlock the memory block.

And it's not really sharing. It's more like a loan: When you loan somebody a book, you don't get to read it over their shoulder. You just have to wait until they return it, and then the book is yours again.

Now, if all of the operations performed on the stream are non-mutating operations, like `Seek` and `Read`, then things are pretty manageable, because the stream object won't make any changes to the `HGLOBAL`. But things get weirder if there's a `Write` or a `SetSize`.

The `Write` will update the bytes in the `HGLOBAL`, which might or might not be surprising, depending on whether you expected the memory to be read-write or read-only.

Far more surprising is if a `Write` extends past the end of the stream, or if a `SetSize` operation is performed. These will change the size of the stream, which means that the stream object will reallocate the `HGLOBAL` memory block. If you had performed a `GlobalLock` of your own in order to access the memory, and a reallocation is performed on the stream from another thread, then that reallocation may fail.

Let's walk through the consequences.

One usage pattern for suppressing automatic cleanup is if you have a `HGLOBAL` representing a block of memory, and you want to create multiple streams from it, so that all the streams represent the same underlying data. You wait for all of those streams to be destroyed, and then you can free the underlying `HGLOBAL`.

This pattern works provided you can guarantee three things:

1. Nobody extends the stream via `IStream::Write`.[1]
2. Nobody resizes the stream via `IStream::SetSize`.
3. You know when all of the streams have been destroyed for sure.

For the first two points: Each stream internally keeps track of the amount of valid data in its associated `HGLOBAL`. The amount of valid data can be less than the size of the `HGLOBAL`, thanks to allocation granularity. If one stream changes the size of the memory block, the others won't know about it. This could mean that data written to one stream ends up not readable by another stream constructed from the same `HGLOBAL`, because the data got put beyond what the other stream believed to be the end of the data.

Even worse, if one stream shrinks the memory block, none of the other streams will know about this, and you will get buffer overflows.

Another problem is that locking the memory block can collide with a reallocation. If one of the modifications to the stream triggers a reallocation of the `HGLOBAL`, this requires that the memory block be unlocked. If any other stream object happens to be reading from the memory block at the same time, then that other stream object will possess a lock on the `HGLOBAL` and prevent the reallocation.

This is a race condition that causes `Write` and `SetSize` operations to fail randomly. You're going to have a hard time reproducing this bug, but it'll leave you scratching your head for a long time.

The root cause is that the stream objects cannot coordinate their efforts because they were created independently from each other. None of them know that the `HGLOBAL` they were given is two-timing with another stream behind their back.

To solve this problem, don't call `CreateStreamOnHGlobal` multiple times with the same `HGLOBAL` memory block. Instead, call it once to create the stream, and then call `IStream::Clone` to create clones of the stream from the first stream. `HGLOBAL`-based streams that are clones of each other share the same underlying data, but they also are aware of each other, in that weird sci-fi way that clones are psychically connected to each other.

Okay, so you switched to using clones rather than independent streams all based on the same `HGLOBAL`.

The third issue is that you need to know when all of the streams have been destroyed, because that's when you can finally clean up the `HGLOBAL`. Identifying when this has occurred may be tricky because the lifetime of any COM object can, in principle, be extended by calling `AddRef`, and you have to wait for all of those references to be `Release`d before you can clean up. Knowing when this has occurred relies on knowing the behavior of each of the components you give the streams to, and in turn, the behavior of each of the components *they* give the streams to, and so on. And woe unto you if any of those components hand the stream to a garbage-collected language, because the stream won't be released until the next GC![2]

If you're not scared enough yet, next time we'll look at the even crazier things that happen if the `HGLOBAL` you provide is fixed rather than movable. Yes, the documentation says that you need to pass a movable block, but that never stopped people from passing fixed blocks anyway.

[1] As noted earlier, a non-extending write modifies the underlying data for all of the streams that share the `HGLOBAL`. That may cause the other stream consumers to get confused: "I just read a byte, and then I went back to read that same byte, and it's *different*!" This may be something the clients are not expecting.

[2] Now, even though the stream is stuck waiting for GC, it's technically okay to steal the `HGLOBAL` out from under it, provided you are absolutely sure that the only thing that will happen to the stream is its `Release`. If any other operation occurs, it will operate on a freed `HGLOBAL`, and you have a use-after-free bug on your hands, which will manifest itself as a denial of service, information disclosure, or possibly even remote code execution. At any rate, you're going to be scrambling a security fix. That's not fun.

Raymond Chen

Follow