A history of GlobalLock, part 4: A peek at the implementation

devblogs.microsoft.com/oldnewthing/20041109-00

November 9, 2004

Raymond Chen

On one of our internal discussion mailing lists, someone posted the following question:

> We have some code that was using DragQueryFile to extract file paths. The prototype for DragQueryFile appears as follows:
>
> ```
> UINT DragQueryFile(
>     HDROP hDrop,
>     UINT iFile,
>     LPTSTR lpszFile,
>     UINT cch
> );
> ```
>
> In the code we have, instead of passing an HDROP as the first parameter, we were passing in a pointer to a DROPFILES structure. This code was working fine for the last few months until some protocol changes we made in packet layouts over the weekend.
>
> I know that the bug is that we should be passing an HDROP handle instead of a pointer, but I am just curious as to why this worked so flawlessly until now. In other words, what determines the validity of a handle and how come a pointer can sometimes be used instead of a handle?

GlobalLock accepts HGLOBALs that refer to either GMEM_MOVEABLE or GMEM_FIXED memory. The rule for Win32 is that for fixed memory, the HGLOBAL is itself a pointer to the memory, whereas for moveable memory, the HGLOBAL is a handle that needs to be converted to a pointer.

GlobalAlloc works closely with GlobalLock so that GlobalLock can be fast. If the memory happens to be aligned just right and pass some other tests, GlobalLock says "Woo-hoo, this is a handle to a GMEM_FIXED block of memory, so I should just return the pointer back."

The packet layout changes probably altered the alignment, which in turn caused GlobalLock no longer to recognize (mistakenly) the invalid parameter as a GMEM_FIXED handle. It then went down other parts of the validation path and realized that the handle wasn't valid at all.

This is not, of course, granting permission to pass bogus pointers to GlobalLock; I'm just explaining why the problem kicked up all of a sudden even though it has always been there.

With that lead-in, what's the real story behind GMEM_MOVEABLE in Win32?

GMEM_MOVEABLE memory allocates a "handle". This handle can be converted to memory via GlobalLock. You can call GlobalReAlloc() on an unlocked GMEM_MOVEABLE block (or a locked GMEM_MOVEABLE block when you pass the GMEM_MOVEABLE flag to GlobalReAlloc which means "move it even if it's locked") and the memory **will move**, but the handle will continue to refer to it. You have to re-lock the handle to get the new address it got moved to.

GMEM_MOVEABLE is largely unnecessary; it provides additional functionality that most people have no use for. Most people don't mind when Realloc hands back a different value from the original. GMEM_MOVEABLE is primarily for the case where you hand out a memory handle, and then you decide to realloc it behind the handle's back. If you use GMEM_MOVEABLE, the handle remains valid even though the memory it refers to has moved.

This may sound like a neat feature, but in practice it's much more trouble than it's worth. If you decide to use moveable memory, you have to lock it before accessing it, then unlock it when done. All this lock/unlock overhead becomes a real pain, since you can't use pointers any more. You have to use handles and convert them to pointers right before you use them. (This also means no pointers into the middle of a moveable object.)

Consequently, moveable memory is useless in practice.

Note, however, that GMEM_MOVEABLE still lingers on in various places for compatibility reasons. For example, clipboard data must be allocated as moveable. If you break this rule, some programs will crash because they made undocumented assumptions about how the heap manager internally manages handles to moveable memory blocks instead of calling GlobalLock to convert the handle to a pointer.

A very common error is forgetting to lock global handles before using them. If you forget and instead just cast a moveable memory handle to a pointer, you will get strange results (and will likely corrupt the heap). Specifically, global handles passed via the `hGlobal` member of the `STGMEDIUM` structure, returned via the `GetClipboardData` function, as well as lesser-known places like the `hDevMode` and `hDevNames` members of the `PRINTDLG` structure are all potentially moveable. What's scary is that if you make this mistake, you might actually get away with it for a long time (if the memory you're looking at happened to be allocated as GMEM_FIXED), and then suddenly one day it crashes because all of a sudden somebody gave you memory that was allocated as GMEM_MOVEABLE.

Okay, that’s enough about the legacy of the 16-bit memory manager for now. My head is starting to hurt…

Raymond Chen

Follow