A history of GlobalLock, part 1: The early years

devblogs.microsoft.com/oldnewthing/20041104-00

November 4, 2004

Raymond Chen

Once upon a time, there was Windows 1.0. This was truly The Before Time. 640K. Segments. Near and far pointers. No virtual memory. Co-operative multitasking.

Since there was no virtual memory, swapping had to be done with the co-operation of the application. When there was an attempt to allocate memory (either for code or data) and insufficient contiguous memory was available, the memory manager had to perform a process called "compaction" to make the desired amount of contiguous memory available.

- Code segments could be discarded completely, since they can be reloaded from the original EXE. (No virtual memory – there is no such thing as "paged out".) Discarding code requires extra work to make sure that the next time the code got called, it was re-fetched from memory. How this was done is not relevant here, although it was quite a complicated process in and of itself.
- Memory containing code could be moved around, and references to the old address were patched up to refer to the new address. This was also a complicated process not relevant here.
- Memory containing data could be moved around, but references to the old addresses were not patched up. It was the application's job to protect against its memory moving out from under it if it had a cached pointer to that memory.
- Memory that was locked or fixed (or a third category, "wired" — let's not get into that) would never be moved.

When you allocated memory via GlobalAlloc(), you first had to decide whether you wanted "moveable" memory (memory which could be shuffled around by the memory manager) or "fixed" memory (memory which was immune from motion). Conceptually, a "fixed" memory block was like a moveable block that was permanently locked.

Applications were strongly discouraged from allocating fixed memory because it gummed up the memory manager. (Think of it as the memory equivalent of an immovable disk block faced by a defragmenter.)

The return value of GlobalAlloc() was a handle to a global memory block, or an HGLOBAL. This value was useless by itself. You had to call GlobalLock() to convert this HGLOBAL into a pointer that you could use.

GlobalLock() did a few things:

- It forced the memory present (if it had been discarded). Other memory blocks may need to be discarded or moved around to make room for the memory block being locked.
- If the memory block was "moveable", then it also incremented the "lock count" on the memory block, thus preventing the memory manager from moving the memory block during compaction. (Lock counts on "fixed" memory aren't necessary because they can't be moved anyway.)

Applications were encouraged to keep global memory blocks locked only as long as necessary in order to avoid fragmenting the heap. Pointers to unlocked moveable memory were forbidden since even the slightest breath — like calling a function that happened to have been discarded — would cause a compaction and invalidate the pointer.

Okay, so how did this all interact with GlobalReAlloc()?

It depends on how the memory was allocated and what its lock state was.

If the memory was allocated as "moveable" and it wasn't locked, then the memory manager was allowed to find a new home for the memory elsewhere in the system and update its bookkeeping so the next time somebody called GlobalLock(), they got a pointer to the new location.

If the memory was allocated as "moveable" but it was locked, or if the memory was allocated as "fixed", then the memory manager could only resize it in place. It couldn't move the memory either because (if moveable and locked) there were still outstanding pointers to it, as evidenced by the nonzero lock count, or (if fixed) fixed memory was allocated on the assumption that it would never move.

If the memory was allocated as "moveable" and was locked, or if it was allocated as "fixed", then you can pass the GMEM_MOVEABLE flag to *override the "may only resize in place" behavior*, in which case the memory manager would attempt to move the memory if necessary. Passing the GMEM_MOVEABLE flag meant, "No, really, I know that according to the rules, you can't move the memory, but I want you to move it anyway. I promise to take the responsibility of updating all pointers to the old location to point to the new location."

(Raymond actually remembers using Windows 1.0. Fortunately, the therapy sessions have helped tremendously.)

Next time, the advent of selectors.

Raymond Chen

Follow