A history of GlobalLock, part 3: Transitioning to Win32

devblogs.microsoft.com/oldnewthing/20041108-00

November 8, 2004

Raymond Chen

Now that you know how the 16-bit memory manager handled the global heap, it's time to see how this got transitioned to the new 32-bit world.

The `GlobalAlloc` function continued to emulate all its previous moveability rules, but the return value of `GlobalAlloc` was no longer a selector since Win32 used the processor in "flat mode".

This means that the old trick of caching a selector and reallocating the memory out from under it no longer worked.

Moveability semantics were preserved. Memory blocks still had a lock count, even though it didn't really accomplish anything since Win32 never compacted memory. (Recall that the purpose of the lock count was to prevent memory from moving during a compaction.)

Moveable memory and locking could have been eliminated completely, if it weren't for the `GlobalFlags` function. This function returns several strange bits of information—now entirely irrelevant—the most troubling of which is the lock count. Consequently, the charade of locking must be maintained just in case there's some application that actually snoops at the lock count, or a program that expected the `GlobalReAlloc` function to fail on a locked block.

Aside from that, moveable memory gets you nothing aside from overhead.

The `LocalAlloc` function also carries the moveability overhead, but since local memory was never passed between DLLs in Win16, the local heap functions don't carry as much 16-bit compatibility overhead as the global heap functions. `LocalAlloc` is preferred over `GlobalAlloc` in Win32 for that reason. (Of course, many functions require a specific type of memory allocation, in which case you don't have any choice. The clipboard, for example, requires moveable global handles, and COM requires use of the task allocator.)

Next time, an insight into how locking is implemented (even though it doesn't do anything).

Raymond Chen

Follow