What was the difference between LocalAlloc and GlobalAlloc?

devblogs.microsoft.com/oldnewthing/20041101-00

November 1, 2004

Raymond Chen

Back in the days of 16-bit Windows, the difference was significant.

In 16-bit Windows, memory was accessed through values called "selectors", each of which could address up to 64K. There was a default selector called the "data selector"; operations on so-called "near pointers" were performed relative to the data selector. For example, if you had a near pointer `p` whose value was 0x1234 and your data selector was 0x012F, then when you wrote `*p`, you were accessing the memory at 012F:1234. (When you declared a pointer, it was near by default. You had to say `FAR` explicitly if you wanted a far pointer.)

Important: Near pointers are always **relative to** a selector, usually the data selector.

The GlobalAlloc function allocated a selector that could be used to access the amount of memory you requested. (If you asked for more than 64K, then something exciting happened, which is not important here.) You could access the memory in that selector with a "far pointer". A "far pointer" is a selector combined with a near pointer. (Remember that a near pointer is relative to a selector; when you combine the near pointer with an appropriate selector, you get a far pointer.)

Every instance of a program and DLL got its own data selector, known as the HINSTANCE, which I described in an earlier entry. The default data selector for code in a program executable was the HINSTANCE of that instance of the program; the default data selector for code in a DLL was the HINSTANCE of that DLL. Therefore, if you had a near pointer `p` and accessed it via `*p` from a program executable, it accessed memory relative to the program instance's HINSTANCE. If you accessed it from a DLL, you got memory relative to your DLL's HINSTANCE.

The memory referenced by the default selector could be turned into a "local heap" by calling the LocalInit function. Initialing the local heap was typically one of the first things a program or DLL did when it started up. (For DLLs, it was usually the **only** thing it did!) Once you have a local heap, you can call LocalAlloc to allocate memory from it. The LocalAlloc function returned a near pointer relative to the default selector, so if you called it from a program executable, it allocated memory from the executable's HINSTANCE; if you called it from a DLL, it allocated memory from the DLL's HINSTANCE.

If you were clever, you realized that you could use LocalAlloc to allocate from memory other than HINSTANCEs. All you had to do was change your default selector to the selector for some memory you had allocated via GlobalAlloc, call the LocalAlloc function, then restore the default selector. This gave you a near pointer relative to something **other than the default selector**, which was a very scary thing to have, but if you were smart and kept careful track, you could keep yourself out of trouble.

Observe, therefore, that in 16-bit Windows, the LocalAlloc and GlobalAlloc functions were completely different! LocalAlloc returned a near pointer, whereas GlobalAlloc returned a selector.

Pointers that you intended to pass between modules had to be in the form of "far pointers" because each module has a different default selector. If you wanted to transfer ownership of memory to another module, you had to use GlobalAlloc since that permitted the recipient to call GlobalFree to free it. (The recipient can't use LocalFree since LocalFree operates on the local heap, which would be the local heap of the recipient – not the same as your local heap.)

This historical difference between local and global memory still has vestiges in Win32. If you have a function that was inherited from 16-bit Windows and it transfers ownership of memory, it will take the form of an HGLOBAL. The clipboard functions are a classic example of this. If you put a block of memory onto the clipboard, it must have been allocated via HGLOBAL because you are transferring the memory to the clipboard, and the clipboard will call GlobalFree when it no longer needs the memory. Memory transferred via STGMEDIUM takes the form of HGLOBALs for the same reason.

Even in Win32, you have to be careful not to confuse the local heap from the global heap. Memory allocated from one cannot be freed on the other. The functional differences have largely disappeared; the semantics are pretty much identical by this point. All the weirdness about near and far pointers disappeared with the transition to Win32. But the local heap functions and the global heap functions are nevertheless two distinct heap interfaces.

I'm going to spend the next few entries describing some of the features of the 16-bit memory manager. Even though you don't need to know them, having some background may help you understand the reason behind the quirks of the Win32 memory manager. We saw a little of that today, where the mindset of the 16-bit memory manager established the rules for the clipboard.

[Raymond is currently on vacation; this message was pre-recorded.]

Raymond Chen

Follow