On memory allocations larger than 64KB on 16-bit Windows

devblogs.microsoft.com/oldnewthing/20171113-00

November 13, 2017

Raymond Chen

Allocating memory blocks larger than 64KB was tricky in 16-bit Windows because the nature of 16-bit segment:offset addressing meant that you could access the memory only 64KB at a time. Global memory allocations returned you a segment (or selector, if running protected mode Windows), and the memory started at offset zero in that selector. Things got complicated once you needed to read the byte that comes after offset `0xFFFF`.

For the purpose of discussion, let's say that the value returned from `GlobalAlloc` was `0x1234`. The first 64KB of the allocated memory are accessible as `1234:0000` through `1234:FFFF`.

In real mode, linear addresses are calculated by taking the segment number, multiplying by 16, and adding the offset. This means that `1234:0000` refer to linear byte `0x12340`, and `1234:FFFF` refer to linear byte `0x12340` + `0xFFFF` = `0x2233F`. The next linear byte is `0x22340`, which you could access as `2234:0000`.

Conclusion: When the offset wraps around, you add `0x1000` to the segment.

In standard mode, linear addresses are calculated by looking up the base address of the selector in the descriptor table, and adding the offset. When Windows allocated a block larger than 64KB, it allocated a block of consecutive selectors, so that the first selector pointed to the first 64KB of the allocated memory, the second selector pointed to second 64KB of the allocated memory, and so on.

Now, consecutive selectors do not have consecutive values, however. On the 80286, the bottom three bits of the selector are used for other purposes, so the numeric difference between consecutive selectors is actually 8. The first 64KB of the allocated memory are accessible as `1234:0000` through `1234:FFFF`, and the next byte after that is available as `123C:0000`.

This makes for a bit of trouble if you're writing a program that needs to run in both real mode and protected mode. When you reach the end of the first 64KB block, how much do you increment the segment/selector by to reach the next 64KB block?

Enter the `__AHINCR` variable.

The `__AHINCR` variable is a variable exported from `KERNEL`. In real mode Windows, the value is `0x1000`. In protected mode Windows, the value is `0x0008`. When your program reaches the end of a 64KB block, it uses the `__AHINCR` value to decide how much to increment the segment/selector by in order to reach the next 64KB block.

Most programmers never saw this variable. It was hidden inside the code generated by the compiler.

With the introduction of enhanced mode Windows, the memory manager did a little more. Enhanced mode Windows used the 80386, "Now with 32-bit registers!" This means that the offset portion of a selector:offset address can be a 32-bit value.

The Windows memory manager assigned the selectors to the different 64KB chunks of data in the same way that the standard mode memory manager did, but instead of setting the selector limit to `0xFFFF`, it set the selector limit to extend to the entire remainder of the block. The first selector's limit was the entire memory block. The second selector's limit was the memory block minus 64KB. The third selector's limit was the memory block minus 128KB. And so on until all the selectors were exhausted.

This arrangement meant that if you could convince your compiler to do it (or if you wrote code in assembly language directly), you could leave the selector alone and operate solely on the offset portion.

Windows 95 took advantage of this. The languages team produced a special version of the compiler that, with proper coaxing and appeasement, could be convinced to access memory using 32-bit offsets from a 16-bit selector, provided you declared the selector and the pointer in just the right way.

No lesson today. Just some reminiscing.

Raymond Chen

Follow