Why can’t you thunk between 32-bit and 64-bit Windows?

devblogs.microsoft.com/oldnewthing/20081020-00

October 20, 2008

Raymond Chen

It was possible to use generic thunks in 16-bit code to allow it to call into 32-bit code. Why can’t we do the same thing to allow 32-bit code to call 64-bit code?

It’s the address space.

Both 16-bit and 32-bit Windows lived in a 32-bit linear address space. The terms 16 and 32 refer to the size of the offset relative to the selector.

Okay, I suspect most people haven’t had to deal with selectors (and that’s probably a good thing). In 16-bit Windows, addresses were specified in the form of a selector (often mistakenly called a “segment”) and an offset. For example, a typical address might be `0x0123:0x4567`. This means “The byte at offset 0x4567 relative to the selector 0x0123.” Each selector had a corresponding entry in one of the descriptor tables which describes things like what type of selector it is (can it be used to read data? write data? execute code?), but what’s important here is that it also contained a base address and a limit. For example, the entry for selector `0x0123` might say “0x0123 is a read-only data selector which begins at linear address 0x00524200 and has a limit of 0x7FFF.” This means that the address `0x0123:n` refers to the byte whose linear address is `0x00524200` + n, provided that n ≤ `0x7FFF`.

With the introduction of the 80386, the maximum limit for a selector was raised from `0xFFFF` to `0xFFFFFFFF`. (Accessing the bytes past `0xFFFF` required a 32-bit offset, of course.) Now, if you were clever, you could say “Well, let me create a selector and set its base to `0x00000000` and its limit to `0xFFFFFFFF`. With this selector, I can access the entire 32-bit linear address space. There’s no need to chop it up into 64KB chunks like I had to back in the 16-bit days. And then I can just declare that *all addresses will be in this form* and nobody would have to bother specifying which selector to use since it is implied.”

And if you said this, then you invented the Win32 addressing scheme. It’s not that there are no selectors; it’s just that there is effectively only one selector, so there’s no need to say it all the time.

Now let’s look at the consequences of this for thunking.

First, notice that a full-sized 16-bit pointer and a 32-bit flat pointer are the same size. The value `0x0123:0x467` requires 32 bits, and wow, so too does a 32-bit pointer. This means that data structures containing pointers do not change size between their 16-bit and 32-bit counterparts. A very handy coincidence.

Next, notice that the 16-bit address space is still fully capable of referring to every byte in the 32-bit address space, since they are both windows into the same underlying linear address space. It's just that the 16-bit address space can only see the underlying linear address space in windows of 64KB, whereas the 32-bit address space can see it all at once. This means that any memory that 32-bit code can access 16-bit code can also access. It's just more cumbersome from the 16-bit side since you have to build a temporary address window.

Neither of these two observations holds true for 32-bit to 64-bit thunking. The size of the pointer has changed, which means that converting a 32-bit structure to a 64-bit structure and vice versa changes the size of the structure. And the 64-bit address space is four billion times larger than the 32-bit address space. If there is some memory in the 64-bit address space at offset ``0x000006fb`01234567``, 32-bit code will be unable to access it. It's not like you can build a temporary address window, because 32-bit flat code doesn't know about these temporary address windows; they abandoned selectors, remember?

It's one thing when two people have two different words to describe the same thing. But if one party doesn't even have the capability of talking about that thing, translating between the two will be quite difficult indeed.

P.S., like most things I state as "fact", this is just informed speculation.

Raymond Chen

Follow