Why is 0x00400000 the default base address for an executable?

devblogs.microsoft.com/oldnewthing/20141003-00

October 3, 2014

Raymond Chen

The default base address for a DLL is 0x10000000, but the default base address for an EXE is 0x00400000. Why that particular value for EXEs? What's so special about 4 megabytes

It has to do with the amount of address space mapped by a single page directory entry on an x86 and a design decision made in 1987.

The only technical requirement for the base address of an EXE is that it be a multiple of 64KB. But some choices for base address are better than others.

The goal in choosing a base address is to minimize the likelihood that modules will have to be relocated. This means not colliding with things already in the address space (which will force you to relocate) as well as not colliding with things that may arrive in the address space later (forcing *them* to relocate). For an executable, the *not colliding with things that may arrive later* part means avoiding the region of the address space that tends to fill with DLLs. Since the operating system itself puts DLLs at high addresses and the default base address for non-operating system DLLs is 0x10000000, this means that the base address for the executable should be somewhere below 0x10000000, and the lower you go, the more room you have before you start colliding with DLLs. But how low can you go?

The first part means that you also want to avoid the things that are already there. Windows NT didn't have a lot of stuff at low addresses. The only thing that was already there was a `PAGE_NOACCESS` page mapped at zero in order to catch null pointer accesses. Therefore, on Windows NT, you could base your executable at 0x00010000, and many applications did just that.

But on Windows 95, there was a lot of stuff already there. The Windows 95 virtual machine manager permanently maps the first 64KB of physical memory to the first 64KB of virtual memory in order to avoid a CPU erratum. (Windows 95 had to work around a lot of CPU bugs and firmware bugs.) Furthermore, the entire first megabyte of virtual address space is mapped to the logical address space of the active virtual machine. (Nitpickers: actually a little more than a megabyte.) This mapping behavior is required by the x86 processor's virtual-8086 mode.

Windows 95, like its predecessor Windows 3.1, runs Windows in a special virtual machine (known as the System VM), and for compatibility it still routes all sorts of things through 16-bit code just to make sure the decoy quacks the right way. Therefore, even when the CPU is running a Windows application (as opposed to an MS-DOS-based application), it still keeps the virtual machine mapping active so it doesn't have to do page remapping (and the expensive TLB flush that comes with it) every time it needs to go to the MS-DOS compatibility layer.

Okay, so the first megabyte of address space is already off the table. What about the other three megabytes?

Now we come back to that little hint at the top of the article.

In order to make context switching fast, the Windows 3.1 virtual machine manager "rounds up" the per-VM context to 4MB. It does this so that a memory context switch can be performed by simply updating a single 32-bit value in the page directory. (Nitpickers: You also have to mark instance data pages, but that's just flipping a dozen or so bits.) This rounding causes us to lose three megabytes of address space, but given that there was four gigabytes of address space, a loss of less than one tenth of one percent was deemed a fair trade-off for the significant performance improvement. (Especially since no applications at the time came anywhere near beginning to scratch the surface of this limit. Your entire computer had only 2MB of RAM in the first place!)

This memory map was carried forward into Windows 95, with some tweaks to handle separate address spaces for 32-bit Windows applications. Therefore, the lowest address an executable could be loaded on Windows 95 was at 4MB, which is 0x00400000.

Geek trivia: To prevent Win32 applications from accessing the MS-DOS compatibility area, the flat data selector was actually an expand-down selector which stopped at the 4MB boundary. (Similarly, a null pointer in a 16-bit Windows application would result in an access violation because the null selector is invalid. It would not have accessed the interrupt vector table.)

The linker chooses a default base address for executables of 0x0400000 so that the resulting binary can load without relocation on both Windows NT and Windows 95. Nobody really cares much about targeting Windows 95 any more, so in principle, the linker folks could choose a different default base address now. But there's no real incentive for doing it aside from making diagrams look prettier, especially since ASLR makes the whole issue moot anyway. And besides, if they changed it, then people would be asking, “How come some executables have a base address of 0x04000000 and some executables have a base address of 0x00010000?”

TL;DR: To make context switching fast.

Raymond Chen

Follow