Why doesn't Windows use the 64-bit virtual address space below 0x00000000`7ffe0000?

devblogs.microsoft.com/oldnewthing/20221216-00

December 16, 2022

Raymond Chen

A customer used VMMap and observed that for all of their 64-bit processes, nothing was allocated at any addresses below `0x00000000`7ffe0000`. Why does the virtual address space start at `0x00000000`7ffe0000`? Is it to make it easier to catch pointer truncation bugs? And what's so special about `0x00000000`7ffe0000`?

Okay, let's go through the questions one at a time.

First, is it even true that the virtual address space starts at `0x00000000`7ffe0000`?

No. The virtual address space starts at the 64KB boundary. You can confirm this by calling `GetSystemInfo` and checking the `lpMinimumApplicationAddress`. It will be `0x00000000`00010000`.

If the address space starts at 64KB, why is the lower 2GB pretty much ignored?

Because it turns out that the total address space is really big.

Address Space Layout Randomization (ASLR) tries to put things at unpredictable addresses. The full user-mode address space on x86-64 is 128TB, and a randomly-generated 47-bit address is very unlikely to begin with 15 consecutive zero bits. The first 2GB of address space is only 0.003% of the total available address space, so it's a pretty small target.

But why is there a page of memory consistently allocated at exactly `0x00000000`7ffe0000`?

That is a special page of memory that is mapped read-only into user mode from kernel mode, and it contains things like the current time, so that applications can get this information quickly without having to take a kernel transition. This page is at a fixed location for performance reasons.

If a 64-bit application is not marked `/LARGEADDRESSAWARE`, then it receives a shrunken address space of only 2GB so that it doesn't have to deal with any "large addresses" (namely, those above 2GB). These "64-bit processes that don't understand addresses above 2GB" (also

known in my kitchen as “really stupid 64-bit processes”) still need to access the shared data, so the shared data must go below the 2GB boundary.

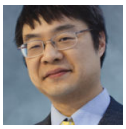
Since virtual address space granularity on Windows is 64KB, reserving a single page actually reserves an entire 64KB block, from `0x00000000`7ffe0000` to `0x00000000`7fffffff`. Also in this range is a second read-only page mapped from kernel mode, this time for sharing information with the hypervisor, specifically the partition reference time stamp counter. This second page is placed at a random location inside the 64KB range, not so much for ASLR reasons (since there are only 15 choices, which isn’t very random), but just to make sure that nobody accidentally takes a dependency on a fixed address. As noted in the documentation, you find this partition reference time stamp counter page by reading the `HV_X64_MSR_REFERENCE_TSC` model-specific register.

Okay, but why are these special shared pages in the range `0x00000000`7ffe0000` to `0x00000000`7fffffff`? Why not put them right up at the 2GB boundary of `0x00000000`7fff0000` to `0x00000000`7fffffff`?

One reason is that the 64KB region immediately above and below the 2GB boundary are marked permanently invalid in order to simplify address validity checks: On 32-bit systems, the 2GB boundary marks the traditional boundary between user mode and kernel mode. If you put a “no man’s land” between user mode and kernel mode, then you can validate a memory range by checking that it starts in user mode, and verifying that every page is accessible. You’ll run into the inaccessible page before you get to the kernel mode addresses.

Okay, that explains why there’s a no man’s land on 32-bit systems, but why do we also have it on 64-bit systems?

On the Alpha AXP, most 32-bit constants can be generated in at most two instructions. But there’s a range of values that requires three instructions: `0x7fff8000` to `0x7fffffff`. Blocking off that region means that you never have to provide a relocation fixup that targets the problematic memory range. And the initial target of 64-bit Windows was the Alpha AXP.



Raymond Chen

Follow