Does the page table entry really have a sad-face for pages that are reserved?

devblogs.microsoft.com/oldnewthing/20160916-00

September 16, 2016

Raymond Chen

Commenter Mc asked whether the memory manager really uses the Unicode sad face emoticon to represent not-present pages, or whether it just uses some boring sentinel value like –1.

This is actually an insightful question disguised as a joke.

Each page table entry is 8 bytes long.[1] One of the bits in the page table entry represents whether the page is present. If the page is present, then the P flag is set, and other parts of the page table entry describe various attributes of the memory, such as whether it is read-only, and (perhaps most important) where to find that memory.

But if the page is not present, then the other 63 bits of the page table entry are not used by the processor and are explicitly documented as available for the operating system.

Pages can be not-present for a number of reasons. The memory for the page may have been paged out, or maybe it's a memory-mapped file, or it's a demand-paged executable with on-the-fly fixups, or it might be delay-zero-initialized memory. Or maybe it's just plain invalid after all.

Historically, operating systems took advantage of the fact that it had 63 free bits in the page table entry of non-present pages. Those bits were used to help answer the question, "So if the CPU tells me that somebody tried to access this page, what should I do?" Those bits might tell the memory manager, "This is a reserved page. Just raise an access violation." Or maybe "This is a read-only page. Raise an access violation if the operation was a write. Otherwise, try to page it in." And if the conclusion was "Page it in," those other bits helped the memory manager figure out what to page in.

Presumably this is why the Windows NT memory manager created a page table full of sad faces. They weren't the same sad face repeated 512 times; rather, it was 512 slightly different sad faces that helped the memory manager figure out what each page was for. (I'm basing my

assumption on the fact that this is what the Windows 95 memory manager did, and it seems like a reasonable thing to do because, hey, free memory.)

What changed is that with the advent of 64-bit Windows, it became not unreasonable to reserve quantities of memory that in 32-bit Windows would have been considered absurd. I mean, 32-bit Windows had only 4GB of address space to begin with, so reserving 100GB of address space is fundamentally impossible. And you don't really worry too much about the most efficient way to do something that is fundamentally impossible.

Not allocating a page table entry for every page in a large reserved region saves memory because you don't need to allocate page tables any more. The cost is that in order to decide what to do in response to a page fault, you can't rely on the free information in the page table entry. You have to go look it up yourself.

[1] In the original 80386, page table entries were only 4 bytes long, but they grew to 8 bytes with the introduction of PAE. For over a decade, all versions of Windows enable PAE, even on machines that have less than 4GB of physical memory, because PAE is a prerequisite for NX.

Raymond Chen

Follow