Why are the alignment requirements for SLIST_ENTRY so different on 64-bit Windows?

devblogs.microsoft.com/oldnewthing/20110819-00

August 19, 2011

Raymond Chen

The `InterlockedPushEntrySList` function stipulates that all list items must be aligned on a `MEMORY_ALLOCATION_ALIGNMENT` boundary. For 32-bit Windows, `MEMORY_ALLOCATION_ALIGNMENT` is 8, but the `SLIST_ENTRY` structure itself does not have a `DECLSPEC_ALIGN(8)` attribute. Even more confusingly, the documentation for `SLIST_ENTRY` says that the 64-bit structure needs to be 16-byte aligned but says nothing about the 32-bit structure. So what are the memory alignment requirements for a 32-bit `SLIST_ENTRY`, 8 or 4? It's 8. No, 4. No wait, it's both. Officially, the alignment requirement is 8. Earlier versions of the header file did not stipulate 8-byte alignment, and changing the declaration would have resulted in existing structures which (inadvertently) misaligned the field changing size and layout when the new requirement was imposed. So the 32-bit structure was sort-of grandfathered in. You should still align it on 8-byte boundaries, but the header file doesn't enforce it to avoid breaking existing code. Fortunately, when the 64-bit version was introduced, the proper alignment directive was introduced right off the bat. How about that: sometimes Microsoft learns from its mistakes after all. Why are the alignment requirements greater than the natural word size? To avoid the ABA problem. A standard workaround for the ABA problem is to append additional information (a "tag") to the pointer so that when the value changes from B back to A, the tag ensures that the second A still looks different from the first one. Many CPU architectures have a "double-pointer-sized atomic compare-and-swap" instruction, and some of them have the additional requirement that the double-pointer needs to be on a double-pointer boundary (8 bytes for 32-bit pointers and 16 bytes for 64-bit pointers). "But wait, the double-pointer compare-and-swap is used on the `SLIST_HEADER`, not on the `SLIST_ENTRY`. Why does the `SLIST_ENTRY` need to be double-pointer aligned, too?"

While it's true that many CPU architectures have a "double-pointer-sized atomic compare-and-swap" instruction, some support only a "pointer-sized atomic compare-and-swap". For example, the original AMD64 architecture did not have a CMPXCHG16B instruction; the largest data size for an atomic compare-and-swap was 8 bytes. As a result, the Slist functions

need to pack a 64-bit pointer, a list depth, *and* tag information into a single 64-bit value. One of the tricks they used was imposing a memory alignment of 16 bytes. This freed up four bits in the pointer for use as a tag.

Raymond Chen

Follow