Footnotes in Win32 history: VLM (Very Large Memory) support

devblogs.microsoft.com/oldnewthing/20070801-00

August 1, 2007

Raymond Chen

A long-forgotten footnote in Win32 history is the set of functions known as “VLM” for “Very large Memory”. To understand VLM, you first need to understand the Alpha AXP.

The Alpha AXP was a wonderful architecture, and I was sad to see it go. Partly because it meant that the years I spent learning the ins and outs of the processor were now just wasted space in my brain, good only for muttering incoherently during meetings and blog entries. Hang on a second…

The Alpha AXP was a 64-bit processor. None of this “64-bit mode” versus “32-bit mode” that we have with the AMD64 and EM64T (and early versions of the Itanium). It was 64-bit all the time. Now, the instruction set did provide a few arithmetic instructions which operated on 32-bit values, but the results were always sign-extended back up to 64 bits. This one concession to 32-bit code meant that you could run code that was conceptually 32-bit on this 64-bit CPU: All the operating system has to do is treat the 32-bit addresses 0x00000000 through 0x7FFFFFFF as 64-bit addresses 0x00000000`00000000 through 0x00000000`7FFFFFFF, and treat 32-bit addresses 0x80000000 through 0xFFFFFFFF as 64-bit addresses 0xFFFFFFFF`80000000 through 0xFFFFFFFF`FFFFFFFF, The processor’s natural sign extension did the rest.

(And now you can see another reason why there is a no-man’s land around the 2GB boundary. If objects were allowed to cross the 2GB boundary, they would end up being split up when converted to the Alpha AXP’s 64-bit address space.)

This is sort of analogous to running 16-bit Windows programs on an 80386 processor. Your 16-bit program could still use those 32-bit registers if it only knew they were there.

This clever design of the Alpha AXP meant that you could read through quite a bit of Alpha AXP assembly language without being able to tell whether the code was designed as 32-bit or 64-bit code since it all looked the same. The only giveaway would be when the code loaded a pointer from memory.

Anyway, back to VLM. Windows NT on an Alpha AXP was a 32-bit operating system on a 64-bit processor. The 32-bit address space was only a tiny fraction of the full 64-bit address space available to the processor. And VLM gave you access to that space that would otherwise go wasted.

You allocated memory in the 64-bit address space with functions like `VirtualAllocVlm`. All of the VLM functions operated on 64-bit pointers (called `PVOID64`). Allocating memory via VLM returned you a `PVOID64`, a 64-bit pointer to the memory, and your program had to use these 64-bit pointers to access the memory. And like its successor AWE, VLM allocated non-paged memory.

In addition to allocating memory, there were special functions for memory-mapping a file into the 64-bit address space and performing disk I/O into and out of these 64-bit addresses. But that's about it. The rest of Win32 still used 32-bit pointers, so you couldn't pass these 64-bit addresses to functions like `lstrcmpi`. You were given the raw materials for allocating memory in the 64-bit address space and the rest was up to you.

These functions were designed for high-end database programs which required enormous quantities of memory and (more importantly) address space. The memory wasn't pageable because high-end database programs invariably perform their own highly specialized memory management, and paging just gets in the way.

With the death of the Alpha AXP also came the death of VLM, since the Alpha AXP was the only architecture that supported 64-bit addresses in 32-bit code.

This model of programming is not dead, however. It was seen in Windows 3.1 with the `WINMEM32` library (which even let you create 32-bit code segments!), and a more restricted version of it lives on in AWE.

Raymond Chen

Follow