What does the SizeOfImage mean in the MODULEINFO structure?

devblogs.microsoft.com/oldnewthing/20211015-00

October 15, 2021

Raymond Chen

A customer had a program that supported a plug-in model, and they wanted to do some analysis of how much memory those plug-ins were costing. They can already track the dynamic memory allocations performed by those plug-ins (because the plug-in model is very restrictive), but they were interested in the memory footprint of the module's code and static data segments. They were getting information about the plug-in by calling `GetModule-Information` and looking at the `SizeOfImage`, but they wanted some guidance on how to interpret the value. Would a large zero-initialized static array, for example, count toward the `SizeOfImage`? They know that zero-initialized static data tends not to occupy space in the file itself, but do they show up in the `SizeOfImage`? What is that the size of, anyway?

Recall that the `MODULEINFO` structure gets its information from the loader, and the loader's job is managing the modules loaded by a process. The `SizeOfImage` in is paired with the `lpBaseOfDll` to describe the virtual address range that the DLL's image occupies after being loaded from disk. The loader uses this information for various things, including responding to the `GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS` flag that is passed to the `GetModuleHandleEx` function.

The size of the image in the virtual address space includes its code, static data, and resources. This could be bigger or smaller than the size on disk: The file on disk may contain bytes that are not mapped into memory, like the digital certificate, or any additional payload appended to the end of the file. On the other hand, there are things that occupy space in the virtual address space that aren't in the file, like zero-initialized data or inter-section padding. A module could even intentionally lay out its address space with gaps between sections larger than the minimum required to maintain alignment.

If you are looking to measure the memory footprint of the plug-in, you'll have to take these things into account. Do you want to count read-write data differently from code, resources, and read-only data? Not all of the code, resources, read-only and read-write data are likely to be read from disk, although they do consume space in the virtual address space. In particular, the relocation table is almost certainly not ever going to be used.

Deciding which sections to count and which not to count is something you need to decide for yourself. The module itself does help you out by providing `SizeOfCode`, `SizeOfData`, and `SizeOfUninitializedData` to save you the trouble of having to calculate those values yourself. Those three values cover code, data (both read-only and read-write), and zero-initialized data, respectively. They're probably a good start if you're not interested in relocations, resources, or exception tables.

Raymond Chen

Follow