How are DLL functions exported in 32-bit Windows?

devblogs.microsoft.com/oldnewthing/20060718-32

July 18, 2006

Raymond Chen

The designers of 32-bit Windows didn't have to worry quite so much about squeezing everything into 256KB of memory. Since modules in Win32 are based on demand-paging, all you have to do is map the entire image into memory and then run around accessing the parts you need. There is no distinction between resident and non-resident names; the names of exported functions are just stored in the image, with a pointer (well, a relative virtual address) to the name stored in the export table. Unlike the 16-bit ordinal export table, the 32-bit ordinal export table is not sparse. If your DLL exports two functions, one as ordinal 10 and one as ordinal 1000, you will have a 991-entry table that consists of two actual function pointers and a lot of zeros. Thus, you should try not to have large gaps in your ordinal exports, or you will be wasting space in your DLL's export table. As I noted above, there is only one exported names table, so you don't have to distinguish between resident and non-resident names. The exported names table functions in the same manner as the exported names table in 16-bit Windows, mapping names to ordinals. Unlike the 16-bit named export tables, where order is irrelevant, the exported names table in 32-bit Windows is kept sorted so that a more efficient binary search can be used to locate functions. As with 16-bit Windows, every named function is assigned an ordinal. If the programmer didn't assign one in the module definition file, the linker will make one up for you, and as with 16-bit Windows, the value the linker makes up can vary from build to build. However, there is a major difference between the two models: Recall that named exports in 16-bit Windows were discouraged (on efficiency grounds), and as a result, every exported function was explicitly assigned an ordinal, which was the preferred way of linking to the function. On the other hand, named exports in 32-bit Windows are the norm, with no explicit ordinal assignment. This means that **the ordinal for a named export is not fixed**. For example, let's look at the ordinal that got assigned to the kernel32 function `LocalAlloc` in the early years:

Windows NT 3.1	314
Windows NT 3.5	372
Windows 95	501

Windows NT 4.0 407

Now, some people are in the habit of reverse-engineering import libraries, probably because they can't be bothered to download the Platform SDK and get the **real** import libraries. The problem with generating the import library manually is that you can't tell whether the ordinal that was assigned to, say, the `LoadLibrary` function was assigned by the module definition file (and therefore will not change from build to build) or was just auto-generated by the linker (in which case the ordinal **will** change). The import library generation tools could just play it safe and use the named export, since that will work in both cases, but for some reason, they use the ordinal export instead. (This is probably a leftover from 16-bit Windows, where ordinals were preferred over names, as we saw earlier.) This unfortunate choice on the part of the import library generation tools to live dangerously has created compatibility problems for the DirectX team. (I don't know why DirectX got hit by this harder than other teams. Perhaps because game developers don't have the time to learn the fine details of Win32; they just want to write their game.) Since they used one of these tools, they ended up linking to DirectX functions like `DirectDrawCreate` by ordinal rather than by name, and then when the next version of DirectX came out and the name was assigned a different ordinal by the linker, their programs crashed pretty badly. The DirectX team had to go back to the old DLLs, write down all the ordinals that the linker randomly assigned, and explicitly assign those ordinals in the module definition files so they wouldn't move around in the future. There are other reasons why you cannot generate an import library from a DLL; I'll pick up those topics later when I talk about import libraries in more detail.

Next time, forwarders.

Raymond Chen

Follow