Why couldn't you have more than one instance of a 16-bit multi-DS program?

devblogs.microsoft.com/oldnewthing/20080208-00

February 8, 2008

Raymond Chen

Recall that the `HINSTANCE` identified a set of variables. This causes a bit of a problem if your program has multiple data segments; in other words, multiple sets of variables. In such a program, the code would load the data segment of whatever variable it needed each time it needed to access a variable from a different segment. This was no problem at all for a DLL, since 16-bit DLLs were single-instance. Go ahead, load your selectors whenever you want. Since there's only one copy of each data segment, you can just use them in your code and let the loader fix them up. No matter which processes calls your DLL, you're still good. But if you are doing this in a program, you run into trouble once the user runs a second copy of the program. All you get is an `HINSTANCE` to pass to `MakeProcInstance` (or to infer from your stack selector). In other words, you get one set of variables. If your program uses multiple sets of variables, you don't have a way to access those other variables, and the operating system has no way of telling you where they are. Now, a sufficiently clever compiler could work around this failure of mathematics. It could store the selectors of the extra data segments into the data segment specified by the `HINSTANCE`. When the program needed to access a variable from another data segment, it could access them by loading the appropriate selector from the stack segment register (since SS == DS). I don't know whether anybody actually bothered to write a compiler that did this.

Not that writing one today will win you any accolades since nobody writes 16-bit Windows programs any more. It's one of those things that may have been a neat idea back in its day but today will just get you quizzical looks. Think of it as the computer version of inventing a higher-capacity eight-track cartridge.

Raymond Chen

Follow