How does delay-loading use binding information?

devblogs.microsoft.com/oldnewthing/20100319-00

March 19, 2010

Raymond Chen

In the documentation for delay-loading, there's a remark that says that the call to `GetProcAddress` can be avoided if there is binding information. A customer who received the explanation of why you can't delay-load `kernel32` pointed out that paragraph and asked whether this means that you can delay-load `kernel32` if you bind to it. (Getting around to answering this question was the point of the past few days.) Let's take another look at what that `GetProcAddress`-avoidance optimization does. Actually, it's just another look at what the module loader does when it's time to resolve imports to a bound DLL: At build time, the actual function pointers are precomputed and cached, along with the timestamp of the DLL those precomputed values came from. At run time, the delay-load stubs check the timestamp of the target DLL and compare it against the timestamp that it had cached. If they are the same, then they skip the call to `GetProcAddress` and use the cached value. In other words, the delay-load stubs use binding information in exactly the same way the module loader does. Does this mean that you can now delay-load `kernel32`? No. First of all, if the timestamps don't match or if the target DLL was not loaded at its preferred address, then the binding information is of no use—you have a cache miss. In that case, the module loader (and the delay-load stubs) must obtain the function pointers the old-fashioned way. You can't assume that your binding information will always be accurate. (For example, after your module was bound to `kernel32`, there may have been a security update which modified `kernel32`, which invalidates your binding information.)

And besides, even if the binding information were used, you still have to call `LoadLibrary` to get the target DLL loaded in the first place. Even though binding may have optimized away one call to `kernel32`, you still have that `LoadLibrary` to deal with.

Raymond Chen

Follow