Why not use weak linking to solve the retargetable library problem?

devblogs.microsoft.com/oldnewthing/20160317-00

March 17, 2016

Raymond Chen

In response to the problem of creating a retargetable library, there was disbelief that the Windows linker doesn't support weak symbols. (I'm assuming they meant "loader", not "linker".)

Back in the days of 16-bit Windows, the loader used weak linking. If you imported a function and it didn't exist, you got a null pointer. You were then on the hook not to call the function unless you first checked that it was there. And it was a disaster. Programs crashed left and right because they didn't check whether the function actually existed before calling it. The designers of Win32 decided that if you wanted weak linking, you had to do it explicitly via `LoadLibrary` and `GetProcAddress`.

Okay, but now we've come full circle, because `LoadLibrary` and `GetProcAddress` of system DLLs is not permitted in universal Windows apps. Why not let universal Windows apps link to nonexistent functions, and put the burden on them to check the pointer before calling it?

Well, first of all, that's sort of taking a step backward. "Hey, here's a new programming platform. It's harder to use than the old one."

Second, how would you enforce this policy? The Windows App Certification Kit acceptance test would see that there is an attempt to import the `InitializeCriticalSection` function. Now it needs to reverse-engineer the code to verify that the program never calls `InitializeCriticalSection`. This eventually turns into the Halting Program, which is unsolvable.

Furthermore, the issue isn't that the `InitializeCriticalSection` function doesn't exist. The function exists just fine. It's just that universal Windows apps in the Windows Store are not allowed to call it. This means that you have to come up with a way for the operating system to decide at run time whether the import table entry for `InitializeCritical-`

`Section` should be null or not. This means that the loader must now be aware of Windows Store policies, and whenever the Windows Store changes its policy, a system update needs to be delivered in order to implement that policy.

And even if you somehow got the loader to enforce Windows Store policies, you have to tell the loader, “Oh wait, this program didn’t get installed via the Windows Store.” If a program gets installed by means other than the Windows Store, then Windows Store policies don’t apply. The loader would have to check somehow whether Windows Store policies are in effect for the app.

Therefore, the answer to the original question is twofold. First of all, no, we don’t have weak symbols, on purpose; if you want weak symbols, you can do it yourself with `LoadLibrary` and `GetProcAddress`. Second, weak symbols don’t actually solve the problem, because it leads to making low-level components enforce a high-level policy.

Raymond Chen

Follow