Why can't I use the linker to delay-load a function from kernel32?

devblogs.microsoft.com/oldnewthing/20100201-00

February 1, 2010

Raymond Chen

For some time (I am too lazy to look up when it was introduced), the Visual Studio linker has supported a feature known as delay-loading. But why can't you use this feature to delay-load a function from `kernel32`? It would be very handy: If you write

```
if (CurrentWindowsVersionSupportsKernelFunctionXyz())
{
  Xyz(...);
}
```

the program fails to load on versions of Windows which do not support the function `Xyz` because the Win32 load rejects loading a module that contains unresolved references. On the other hand, if you could mark `kernel32` as delay-loaded, then the code above would work, since the call to `Xyz` would be redirected to a stub that calls `GetProcAddress`. Since the `GetProcAddress` is performed only when the code path is hit, the loader won't complain at load time. But if you try to delay-load `kernel32`, the linker gets upset at you. Why won't it let me delay-load `kernel32`?

The linker delay-load feature operates on the DLL level, not on the function level. When you put a DLL on the `/DELAYLOAD` list, the linker changes all calls to functions in that DLL into calls to linker-generated stubs. These stubs load the target DLL, call `GetProcAddress`, then resume execution at the target function.

Since the delay-load feature operates on the DLL level, if you put `kernel32` on the delay-load list, then *all* calls to functions in `kernel32` turn into calls to stubs.

And then you are trapped in this Catch-22.

When a function from `kernel32` gets called, transfer goes to the stub function, which loads the target DLL (`kernel32`) to get the target function. Except that loading the target DLL means calling `LoadLibrary`, and finding the target function means calling `GetProcAddress`, and these functions *themselves reside in* `kernel32`.

Now you're trapped. To load `kernel32`, we need to call `LoadLibrary`, but our call to `LoadLibrary` was redirected to a stub which... calls `LoadLibrary`.

Sure, the linker folks could have added special casing for `kernel32`, say, having a list of core functions like `InitializeCriticalSection` which are never delay-loaded and always go directly into `kernel32`. But that's really out of scope for the `/DELAYLOAD` feature, whose purpose is not to make it easier to call functions which might not be there, but rather to assist in application startup performance by avoiding the cost of loading the target DLL until a function from it is called. If there were functions that went directly into `kernel32`, then the stated purpose of delay-loading fails: that import of `InitializeCriticalSection` forces `kernel32` to be loaded when the module is loaded, completely contrary to the aim of delay-loading to avoid loading `kernel32` at module load time.

Now, it's certainly a nice feature to be able to perform delay-loading on a per-function level, in order to make it easier to write code which changes behavior based on the current version of Windows, but that's a different problem from what the `/DELAYLOAD` switch was created to solve.

Raymond Chen

Follow