How a less naive compiler calls an imported function

devblogs.microsoft.com/oldnewthing/20060724-00

July 24, 2006

Raymond Chen

If a function is declared with the `dllimport` declaration specifier, this instructs the Visual Studio C/C++ compiler that the function in question is an imported function rather than a normal function with external linkage. With this additional information, the compiler generates slightly different code when it needs to reference an imported function, since the compiler is aware of the special way imported functions are implemented.

First, there is no need for the stub function any more, because the compiler can generate the special `call [__imp__FunctionName]` instruction inline. Furthermore, the compiler knows that the address of an imported function never changes, and consequently it can optimize away multiple loads of the function pointer:

```
mov   ebx, [__imp__FunctionName]
push  1
call  ebx ; FunctionName(1)
push  2
call  ebx ; FunctionName(2)
```

(Note to crazy people: This optimization means that you can run into problems if you patch a module's import table once it has started running, because the function pointer may have been optimized into a register before you patched the import. Consider, in the above example. that you patched the `__imp__FunctionName` table entry after the `mov ebx, [__imp__FunctionName]` instruction: Your replacement import table entry won't be called since the old function address was cached in the `ebx` register.)

Similarly, if your program tries to take the address of an imported function that has been declared with the `dllimport` declaration specifier, the compiler recognizes this and converts it to a load from the imported function address table.

As a result of this extra knowledge imparted to the compiler, the stub function is no longer needed; the compiler knows to go straight to the imported function address table.

Note that there are still occasional circumstances wherein you can induce the stub function to be created. We'll take a look at them (and related dangers) next time.

Raymond Chen

Follow