# Why are some system functions exported as stubs instead as forwarders?

devblogs.microsoft.com/oldnewthing/20200826-00

August 26, 2020

Raymond Chen

If you do a little digging around inside some Windows system functions, you'll see that, for example, the `CreateProcessW` function looks like this:

```
kernel32!CreateProcessW:
6b819ef0 mov     edi,edi
6b819ef2 push    ebp
6b819ef3 mov     ebp,esp
6b819ef5 pop     ebp
6b819ef6 jmp     dword ptr [kernel32!kernelbase_CreateProcessW]
```

The first four instructions have no net effect, so basically this is just an indirect jump to the `kernelbase!CreateProcessW` function. In other words, it's a stub that forwards to the real implementation over in `kernelbase`.

Why is it done this way? Why isn't the `CreateProcessW` function just a forwarder to `kernelbase`? That would avoid having to travel through `kernel32` just to reach `kernelbase`.

Yes, this would normally be a forwarder, but it's not. For backward compatibility.

Wait, why is there a compatibility constraint that the `CreateProcessW` function cannot be a forwarder?

Set the time machine to 2001. The Microsoft Layer for Unicode (MSLU) was just released, also affectionately known as "Unicows", after the DLL component of MSLU: `unicows.dll`.

MSLU was a combination of a static library and a DLL. You wrote a Unicode application and linked it with the MSLU static library. This library contained its own definitions for a large number of functions, including `CreateProcessW`. When your Unicode application called the alternate version of `CreateProcessW`, the library checked whether it was running on a version of Windows that was ANSI-only (the Windows 95 series) or a version that supported Unicode (the Windows NT series).

If it was running on an ANSI-only system, then the stub loaded the `unicows.dll` library and forwarded the call to a helper function in that library which did the work of thunking the Unicode parameters to ANSI, and then calling the `CreateProcessA` function, and then converting the results back to Unicode, and returning that to the caller. If it was running on a Unicode system, then it forwarded the call to the operating system's `CreateProcessW` function.

In other words, the static library contained a stub that decided whether to allow the Unicode call to go straight to the Unicode version of the underlying function, or whether it should convert the call to ANSI and call the ANSI version of the underlying function.

Okay, great, so where do DLL forwarders come into the story?

After the MSLU static library decides which code path it should use, it goes back and patches the the caller's import table to point directly to the destination function. That way, the second and subsequent calls are direct and don't go through the evaluation step again. (This is the same sort of trick that the delay-load stubs use.)

In the case where the MSLU static library decided to pass the function straight to the Unicode version of the underlying function, it needs to get the address of that Unicode version of the underlying function. For reasons not entirely clear to me, it doesn't use the `GetProcAddress` function.[1] Instead it has a custom implementation of `GetProcAddress` which parses the DLL export table manually to find the function to forward to.

That custom implementation of `GetProcAddress` doesn't support forwarders. There's even a comment acknowledging as much:

```
// This is a forwarder - Ignore for now.
```

Therefore, any function supported by MSLU may not take the form of a DLL forwarder. It must be a stub. Just in case somebody runs a program from the early 2000s written with MSLU.

**Bonus chatter**: This requirement that the function be a stub and not be a forwarder applies only to the x86-32 version of Windows, since that's the only architecture supported by the Windows 95 series, and therefore the only one supported by MSLU. However, the functions are stubs on all architectures, presumably for simplicity of implementation.

[1] My suspicion was that it does this to avoid certain reentrancy issues in the loader, but I'm not sure.

Raymond Chen

**Follow**