Why was the replacement installer for recognized 16-bit installers itself a 32-bit program instead of a 64-bit program?

devblogs.microsoft.com/oldnewthing/20150211-00

February 11, 2015

Raymond Chen

Even though 64-bit Windows does not support 16-bit applications, there is a special case for 16-bit installers for 32-bit applications. Windows detects this scenario and substitutes a 32-bit replacement installer which replicates the actions of the 16-bit installer. Commenter Karellen is horrified at the fact that the replacement installer is a 32-bit program. "You're writing a program that will run exclusively on 64-bit systems. Why not built it to run natively on the OS it's designed for? Why is this apparently not the "obvious" Right Thing(tm) to do? What am I missing?"

Recall that a science project is a programming project that is technically impressive but ultimately impractical. For example it might be a project that nobody would actually use, or it attempts to add a Gee-Whiz feature that nobody is really clamoring for.

But at least a science project is trying to solve a problem. This proposal doesn't even solve any problems! Indeed, this proposal *creates* problems. One argument in favor of doing it this way is that it is satisfies some obsessive-compulsive requirement that a 64-bit operating system have no 32-bit components beyond the 32-bit emulation environment itself.

> Because! Because you're running a 64-bit system, and running apps native to that system is just more elegant.

Okay, it's not obsessive-compulsive behavior. It's some sort of aesthetic ideal, postulated for its own sake, devoid of practical considerations.

Remember the problem space. We have a bunch of 32-bit applications that use a 16-bit installer. Our goal is to get those applications installed on 64-bit Windows. By making the replacement installer a 32-bit program, you get the emulator to do all the dirty work for you. Things like registry redirection, file system redirection, and 32-bit application compatibility.

Suppose the original installer database says

- Copy X.DLL file into the `%ProgramFiles%\AppName` directory.
- Copy Y.DLL into the `%windir%\System32` directory.
- If the current version of `C:\Program Files\Common Files\Adobe\Acrobat\ActiveX\AcroPDF.dll` is 7.5 or higher, then set this registry key.

If you write the replacement installer as a 32-bit program, then other parts of the 32-bit emulation engine do the work for you.

- The environment manager knows that 64-bit processes get the environment variable `ProgramFiles` pointing to `C:\Program Files`, whereas 32-bit processes get `ProgramFiles` pointing to `C:\Program Files (x86)`.
- The file system redirector knows that if a 32-bit process asks for `%windir%\System32`, it should really get `%windir%\SysWOW64`.
- The registry redirector knows that if a 32-bit process tries to access certain parts of the registry, they should be sent to the `Wow6432Node` instead.

If you had written the replacement installer as a 64-bit program, you would have to replicate all of these rules and make sure your copy of the rules exactly matched the rules used by the real environment manager, file system redirector, and registry redirector.

Now you have to keep two engines in sync: the 32-bit emulation engine and the 64-bit replacement installer for 32-bit applications. This introduces fragility, because any behavior change in the 32-bit emulation engine must be accompanied by a corresponding change in the 64-bit replacement installer.

Suppose the application compatibility folks add a rule that says, "If a 32-bit installer tries to read the version string from `C:\Program Files\Common Files\Adobe\Acrobat\ActiveX\AcroPDF.dll`, return the version string from `C:\Program Files (x86)\Common Files\Adobe\Acrobat\ActiveX\AcroPDF.dll` instead." And suppose that rule is not copied to the 64-bit replacement installer. Congratulations, your 64-bit replacement installer will incorrectly install any program that changes behavior based on the currently-installed version of AcroPDF.

I don't know for sure, but I wouldn't be surprised if some of these installers support plug-ins, so that the application developer can run custom code during installation. It is possible for 16-bit applications to load 32-bit DLLs via a technique known as generic thunking, and the 16-bit stub installer would use a generic thunk to call into the 32-bit DLL to do whatever custom action was required. On the other hand, 64-bit applications cannot load 32-bit DLLs, so if the 64-bit replacement installer encountered a 32-bit DLL plug-in, it would have to run a 32-bit helper application to load the plug-in and call into it. So you didn't escape having a 32-bit component after all.

And the original obsessive-compulsive reason for requiring the replacement installer to be 64-bit was flawed anyway. This is a replacement installer *for a 32-bit application*. Therefore, the replacement installer is part of the 32-bit emulation environment, so it is allowed to be written as a 32-bit component.

Let's look at the other arguments given for why the replacement installer for a 32-bit application should be written as a 64-bit application.

> Because complexity is what will be our undoing in the end, and reducing it wherever we can is always a win.

As we saw above, writing the replacement installer as a 64-bit application *introduces* complexity. Writing it as a 32-bit application *reduces* complexity. So this statement itself argues for writing the replacement installer as a 32-bit application.

> Because we can't rewrite everything from scratch at once, but we can create clean new code one small piece at a time, preventing an increase to our technical debt where we have the opportunity to do so at negligible incremental cost to just piling on more cruft.

As noted above, the incremental cost is hardly negligible. Indeed, writing the replacement installer as a 64-bit application is not merely more complex, it *creates* an ongoing support obligation, because any time there is a change to the 32-bit emulation environment, that change needs to be replicated in the 64-bit replacement installer. This is a huge source of technical debt: Fragile coupling between two seemingly-unrelated components.

And writing the replacement installer as a 32-bit application does not create a future obligation to port it to 64 bits when support for 32-bit applications is dropped in some future version of Windows. Because when support for 32-bit applications disappears (as it already has on Server Core), there will be no need to port the replacement installer to 64-bit because there's no point writing an installer for a program that cannot run!

Writing the replacement installer as a 32-bit program was the right call.

Raymond Chen

Follow