Hidden compatibility constraints of redirecting program execution via a stub

devblogs.microsoft.com/oldnewthing/20110502-00

May 2, 2011

Raymond Chen

One of the “obvious” solutions to the issue of how much work you’re willing to do to save 68KB of disk space was to replace one of the copies with a stub that launches the other copy. If you try this obvious solution, you may run into some compatibility issues. First of all, there are programs which launch Notepad and then wait on the process handle so they can wait until the user closes Notepad. Your stub program cannot just do a `CreateProcess` on the target, because programs which perform a wait will find the wait satisfied when your stub program exits. Okay, so your stub program has to wait for the real copy of Notepad to exit before it can exit itself. Once you fix that, you’ll find another problem: Programs call `GetExitCodeProcess` to see how Notepad exited. Your stub program therefore cannot just perform an `ExitProcess`; it has to do a `GetExitCodeProcess` on the real Notepad and pass that exit code to your own `ExitProcess`. Once you fix that, you’ll find another problem: There are programs which execute a process and then look for windows owned by that process. (Yes, there can be more than one, but Notepad is a simple program that creates only one top-level unowned window.) Those programs will get the process ID of your stub program and be unable to find the Notepad window (since it belongs to the real Notepad program, which has a different process ID). I’m not sure how to fix that one. Yes, you can write a stub that launches another program, but that solves the “save disk space” problem by introducing other problems.

Remember, even though people are supposed to stick to documented behavior (since that is all that is contractual), in practice any implementation detail becomes a compatibility constraint.

Raymond Chen

Follow