Adding a new flag to enable behavior that previously was on by default

devblogs.microsoft.com/oldnewthing/20060419-14

April 19, 2006

Raymond Chen

One of the suggestions for addressing the network compatibility problem was to give up on fast mode and have a new "fast mode 2". (Equivalently, add a flag to the server capabilities that means "I support fast mode, and I'm not buggy.") This is another example of changing the rules after the game is over, by adding a flag to work around driver bugs. Consider a hypothetical program that uses fast mode on Windows XP. It runs against a Windows Server 2003 server and everybody is happy. Suppose you make a change to Windows Vista so that it requires that servers set a new "fast mode 2" flag in order to support fast mode. When the customer upgrades their client from Windows XP to Windows Vista, they would find that their hypothetical program ran much slower. Whose fault is it? Not the hypothetical program that was using fast mode on Windows XP; that program is using fast mode correctly. Not the Windows Server 2003 machine; that server supports fast mode correctly. Is it Windows Vista, then, that is at fault? "Hey, don't blame me," you answer. you answer. "It's that guy over there. That guy you've never heard of. He made me do it. Blame him!" To describe this sort of behavior I like to steal a phrase from Albert Einstein: "Spooky action at a distance". (Einstein used it to describe what in modern physics is known as quantum entanglement.) In this particular situation, we have a conversation between two participants (the client software and the server software) mediated by a third (Windows) which collapses due to the mere existence of a fourth party not involved in the conversation! It's as if your CD player suddenly lost the ability to play any of your music CDs because some company you've never heard of halfway around the world pressed a bunch of bad CDs for a few months earlier this year. Some people suggested, "Why not have a flag that says 'I support fast mode'?" Indeed that flag already exists; that's why Windows Vista was trying to use fast mode in the first place. The problem wasn't that the server didn't support fast mode. The problem was that the server had a bug in its fast mode implementation. "Okay, then add a new flag that says 'My fast mode isn't buggy.'" Consider also how this course of action would look after a few revisions of the specification:

> In response to the `QUERY_CAPABILITIES` request, the server shall return a 32-bit value consisting of zero of more of the following bits:
>
> | | |
> |---|---|
> | 0x00000001 | This server supports fast mode |
> | 0x00000002 | This server supports fast mode and doesn't have the bug where enumerating a directory with more than 128 files fails on the 129th query |
> | 0x00000004 | This server supports fast mode and doesn't have the bug where the long file name is reported incorrectly in the response packet |
> | 0x00000008 | This server supports fast mode and doesn't have the bug where directories whose names consist entirely of digits are misreported as files |
> | 0x00000010 | This server supports fast mode and doesn't have the bug where the enumeration resets if a file is created in the directory while the enumeration is in progress |
> | 0x00000020 | This server supports fast mode and doesn't have the bug where `FindNext` returns failure even though there are still files to be enumerated |
>
> …

If a new capabilities flag were created for every single server bug that was discovered, the capabilities mask would quickly fill up with all these random bits for bugs that were fixed ages ago. And each time a bug was found in any one server, **all** servers would have to be updated to add the new capabilities bit that says, "I'm not that buggy server you found on April 8th 2006," even the servers sitting in a locked closet whose operating systems are burned into EPROMs. And if you're the author of a new server, which capabilities bits do you set? Do you claim that you don't have the bug where `FindNext` returns failure even though the enumeration hasn't completed? What if, six months after you ship, somebody finds a bug in your server of exactly that sort? I guess this mean that the next revision of the protocol will have to have a new flag:

> | | |
> |---|---|
> | 0x00000020 | This server supports fast mode and doesn't have the bug where it claims that it doesn't have the " `FindNext` returns failure even though there are still files to be enumerated" bug, even though it actually does have the bug, but in a more subtle manner |

Or maybe you're convinced that you don't have any bugs in your "fast mode" implementation. Do you report `0xFFFFFFFF` to say "I have no bugs at all, not even the ones people might discover later in other implementations"? What happens when the 33rd "fast mode" bug is found? Do we have to have a `QUERY_CAPABILITIES2` function? If a capabilities bit is created for every single bug that ever existed in a networking protocol implementation, you'd have a few thousand capability bits all of whom mean "I don't have that bug where…"

Now, I'm not saying that this course of action is out of the question. Sometimes you have to do it, but you also have to realize that the cost for making this type of change is very high, and the benefit had better be worth it.

Raymond Chen

Follow