Why is there no way to add a permission to a page with VirtualProtect instead of replacing it?

devblogs.microsoft.com/oldnewthing/20171116-00

November 16, 2017

Raymond Chen

The `VirtualProtect` function lets you change the protection of a page, but all it can do is replace the current protections with the protections you specify, returning the old protections. There is no way to add a protection to a page. In other words, there is an `InterlockedExchange` for page protections, but no `InterlockedOr`. Why not?

The theory behind page protections is that you should be modifying protections of pages that you have control over, and not messing with protections of pages that are not yours. If you stick to that theory, then if you have to synchronize access to the protections for a page, you can impose your own locking policy to avoid thread races.

In other words, the code that allocated the pages can also set up a policy that any time you want to change the protection of the page, you have to take a particular critical section so you can change the protection *and* update whatever auxiliary data structures are necessary to keep track of the state of the page. Anybody who changes the protection of the page without taking the lock is violating the principle that you should not be modifying protections of pages that are not yours.

This principle of "Don't mess with it if it's not yours" means that if you want to have pages whose protections you want to manipulate, you should allocate them yourselves so that you are the one in control of their protections.

One might argue whether this principle is still valid, but that's the principle that went into the original design, so you'll just have to deal with it.

Even if there were some sort of `InterlockedOr`-like function that added a permission to a page, you still have a problem if the various pieces of code that update the protections of a page are not coördinating with each other.

Consider a page that starts out with `PAGE_READONLY` permission. Function 1 uses this imaginary `VirtualProtectOr` function to add write permssion, bringing the permission to `PAGE_READWRITE`. Function 2 does the same thing, but since the page already has write

permission, the request to add write permission is a nop.

Next, function 1 is finished writing to the page, so it uses the imaginary `VirtualProtect-And` function to remove write permission. Oops, function 2 just lost write permission to the page even though it had added it just moments ago.

In order for this to work, somebody somewhere would have to keep track of how many times somebody wanted to add write permission to the page and remove write permission only when the count drops to zero. And even then, that might not be the right thing. Maybe function 1 is removing write permission for some security purpose. If function 2 still had an outstanding "add write permission" and the page remained write-enabled, well, that's good for function 2, but it leaves function 1 vulnerable.

Since it's not clear what the correct behavior is, there's not much point in the kernel keeping a history of all the protection changes for every page in the system. If you want to have a policy for what should happen if two pieces of code change the protections of the same page, then you are welcome to write code that enforces that policy and make everybody go through you when they want to change the protection.

Bonus chatter: Note also that page permissions cannot be arbitrarily mixed and matched. For example, there are no write-only pages. This makes reconciling multiple page protections even more difficult when the result is something that doesn't exist.

Raymond Chen

Follow