I can create a read-only page, but why not a write-only page?

devblogs.microsoft.com/oldnewthing/20230306-00

March 6, 2023

Raymond Chen

There is an interesting hole in the diagram of page protections supported by the `VirtualAlloc` function:

	Deny write	Allow write
Deny read	`PAGE_NOACCESS`	???
Allow read	`PAGE_READONLY`	`PAGE_READWRITE`

The missing value is `PAGE_WRITEONLY`. Why is there no `PAGE_WRITEONLY`?

The short answer is "Because no processor supports it."

The page protections in most processors are not three bits, one for read, one for write, and one for execute. Different processors do things differently.[1] Typically, they start with a *valid* bit, which says whether any access is allowed at all. If the *valid* bit is not set, then you immediately get `PAGE_NOACCESS`, and the rest of the page table entry is ignored.[2] If the *valid* bit is set, then other bits are used to configure the details of the protection. One way is to have separate bits for controlling write and execute access independently.[3] Another is to take a few bits and treat them as an enumeration which selects from a list of possible page protection combinations, and "write-only" is not on the list of possibilities. Either way, there is no separate *read* bit; the *read* bit overloaded onto the *valid* bit: Every valid page is implicitly readable.

Naturally, if no processors support an operation, there's no point adding a flag for it to the operating system. "Hi, here's a flag that is not supported by anyone. If you set it, the operation always fails. Good luck with that!"

It's also unclear how write-only pages would work anyway. CPUs nowadays typically do not issue precise writes. Rather, when you issue a write, the CPU loads the entire enclosing cache line, modifies the written bytes, and then writes the updated cache line back to memory (usually lazily). If the page were write-only, then the attempt to load the enclosing cache line would fail with an access violation,[4] and you'd never get around to writing the updated cache line.

[1] For the purpose of this discussion, we're looking only at user mode access.

[2] Operating systems often use the ignored bits to record information about the invalid page, so that if a page fault occurs, the operating system can quickly look up what it should do next. For example, the unused bits could be a pointer to a kernel data structure that says, "If somebody tries to access this page, then instead of raising an access violation, try to page the data in from the page file. The data is stored in frame *N*. After you read the data, change the protection to `PAGE_READONLY` and restart the instruction."

[3] On some processors, the control is done by setting the bit to allow access. On other processors, setting the bit denies access.

[4] Alternatively, the CPU might load the cache line, bypassing the write-only protection, but try to prevent the code from observing any of the values that were loaded into that cache line. This could end up vulnerable to a side-channel attack.