If FlushInstructionCache doesn’t do anything, why do you have to call it, revisited

devblogs.microsoft.com/oldnewthing/20190902-00

September 2, 2019

Raymond Chen

You are supposed to call the `FlushInstructionCache` function when you generate or modify code at runtime, so that when the CPU tries to execute the newly-generated or newly-modified modified code, it will read the instructions you wrote, rather than any instructions that may be hanging around in the instruction cache.

Some time ago, we saw that on Windows 95, the `FlushInstructionCache` function didn’t do anything aside from returning. That’s because the mere act of calling a function was sufficient to flush the instruction cache.

On Windows NT, however, the `FlushInstructionCache` function does actual work, since it needs to notify all the other processors of the need to flush their instruction caches, too.

But if you look at Windows 10, you may find that the `FlushInstructionCache` function looks like the Windows 95 version: It doesn’t do anything.

What’s going on?

Whether the `FlushInstructionCache` function “does anything” depends on which processor you’re using. Some processors have an integrated data and instruction cache, in which case the `FlushInstructionCache` function doesn’t need to do anything. Others such as ARM still have separate instruction and data caches, and in those cases, flushing does real work. Whether the `FlushInstructionCache` function “does anything” depends on the processor architecture it was compiled for.

As a programmer, you should just call the `FlushInstructionCache` function and let the operating system figure out whether flushing will actually need to “do anything”.

Raymond Chen

Follow