Why do we need atomic operations on the 80386, if it doesn’t support symmetric multiprocessing anyway?

devblogs.microsoft.com/oldnewthing/20190404-00

April 4, 2019

Raymond Chen

The 80386 processor did not support symmetric multiprocessing, yet we discussed atomic operations when in our overview of the processor. If the processor doesn’t even support symmetric multiprocessing, why does it matter?

Well, one reason is that the 80386 processor does support *asymmetric* multiprocessing. Floating point operations are performed by a coprocessor, and the main processor and coprocessor are both accessing the same memory. Another source of competing memory access is from hardware devices that are using Direct Memory Access (DMA).

Even within the processor, you have to worry about races, because you might be racing with *yourself*.

The 80386 did not support symmetric multiprocessing, but it did support pre-emptive multitasking, which means that any multi-instruction sequence is at risk of being interrupted, and at the worst possible time.

```
; decrement the variable and check against zero
mov     eax, [var]
dec     eax
mov     [var], eax
je      zero
```

If the threads gets pre-empted between the first and third instructions, then the contents of the variable may be changed by another thread, and the decrement operation becomes non-atomic. To ensure atomicity, you need to force the compiler to generate a single `dec` instruction, and then to test the flags directly from the decrement.

```
; decrement the variable and check against zero
dec     [var]
jz      zero
```

There was no way to express this level of detail to compilers of that era, so you had to hide it behind a function call.

And if your operation cannot be expressed in a single instruction, then you're out of luck. Increment and compare against 10? Compare and exchange if equal? Nope, you can't do those things, at least not without some help from the operating system.

Raymond Chen

Follow