Is DateTime.Now an I/O bound operation?

Asked 8 years, 6 months ago Modified 8 years, 6 months ago

Viewed 2k times

19

What happens when you call `DateTime.Now`?

I followed the property code in Reflector and it appears to add the time zone offset of the current locale to `UtcNow`. Following `UTCNow` led me, turn by turn, finally to a Win32 API call.

I reflected on it and asked a related question but haven't received a satisfactory response yet. From the links in the present comment on that question, I infer that there is a hardware unit that keeps time. But I also want to know what unit it keeps time in and whether or not it uses the CPU to convert time into a human readable unit. This will shed some light on whether the retrieval of date and time information is I/O bound or compute-bound.

c# .net datetime

Share

Improve this question

Follow

edited May 23, 2017 at 12:08

Community Bot

1 ●1

asked Jun 5, 2016 at 15:16

Water Cooler v2
33.8k ● 61 ● 181 ● 362

IO and compute bound are quite undefined terms. Can you clarify what properties you are interested in or what you want to achieve? – usr Jun 5, 2016 at 15:19

I think bringing date is I/O bound, converting it with DateTime.Now (it returns readable value right?) is compute-bound. So, both of them? – Ali1928 Jun 5, 2016 at 15:20

@usr I am unable to answer your probe due to my limited understanding of your question and of perhaps the very concept of I/O and compute bound to the depths your question indicates you do. I am not sure of what I want. I just want to know what happens so I can plan how to call `DateTime.Now` the next time so that I don't block on it, if at all. It is just one of the things I was left pondering over recently. – Water Cooler v2 Jun 5, 2016 at 15:23

@Ali I would imagine you are right. However, I would like more accurate details. – Water Cooler v2 Jun 5, 2016 at 15:24

1 If you'd tell us what Win32 API call it went to you could help us give you better answers. Tease. – Bruce Dawson Jun 6, 2016 at 4:26

3 Answers

Sorted by: Highest score (default)

35

You are deeply in undocumented territory with this question. Time is provided by the kernel: the underlying native API call is `NtQuerySystemTime()`. This *does* get tinkered with across Windows versions - Windows 8 especially heavily altered the underlying implementation, with visible side-effects.

It is I/O bound in nature: time is maintained by the RTC (Real Time Clock) which used to be a dedicated chip but nowadays is integrated in the chipset. But there is very strong evidence that it isn't I/O bound in practice. Time updates in sync with the clock interrupt so very likely the interrupt handler reads the RTC and you get a copy of the value. Something you can see when you tinker with `timeBeginPeriod()`.

And you can see when you profile it that it only takes ~7 nanoseconds on Windows 10 - entirely too fast to be I/O bound.

Share Improve this answer Follow

edited Jun 6, 2016 at 7:23
psmears
27.9k ● 4 ● 41 ● 49

answered Jun 5, 2016 at 15:48
Hans Passant
940k ● 148 ● 1.7k ● 2.6k

---

2 It is in the kernel, the RTC is the likely interrupt source. Also heavily affects the thread scheduler, context switches occur at the interrupt rate. A basic reason why a Sleep() can never be more accurate than the rate. More deeply undocumented details that should not affect the way you code.
– Hans Passant Jun 5, 2016 at 16:11

---

1 Actually, I believe the RTC is only used when booting and then the kernel keeps time on its own using some other

timers (possibly the main CPU clock). – André Borie Jun 6, 2016 at 4:56

@AndréBorie The HPET uses different timing mechanisms, nowadays in hardware. When that's not available (old hardware) a combination of RTC and CPU timers. AFAIK things like `DateTime.Now` and `Sleep` use the RTC. You can actually test it in a loop and see the values jump in intervals of (iirc) 15 ms, typical for RTC. – atlaste Jun 6, 2016 at 6:52

If you're going to class accessing the RTC as I/O, where do you stop? Is accessing memory also I/O? – Damien_The_Unbeliever Jun 6, 2016 at 7:46

I have benchmarked(benchmarkdotnet) `DateTime.Now` and it didn't take 7ns but 627ns in average on windows 10(CPU E3-1270 v5 3.60GHz, ProcessorCount=8). Returning a cached value took 0.3ns. – Tim Schmelter Feb 8, 2017 at 15:44

4

You seem to be concerned with blocking. There are two cases where you'd want to avoid that.

1. On the UI thread it's about latency. It does not matter what you do (IO or CPU), it can't take long. Otherwise it freezes the UI thread. `UtcNow` is super fast so it's not a concern.
2. Sometimes, non-blocking IO is being uses as a way to scale throughput as more load is added. Here, the only reason is to save threads because each thread consumes a lot of resources. Since there is no async way to call `UtcNow` the question is moot. You just have to call it as is.

Since time on Windows usually advances at 60 Hz I'd assume that a call to `UtcNow` reads from an in-memory variable that is written to at 60 Hz. That makes is CPU bound. But it does not matter either way.

Share Improve this answer Follow

answered Jun 5, 2016 at 15:27

usr
171k ● 36 ● 247 ● 377

Interesting! Do you know any articles on `~60Hz` Windows time progress? I googled but all I got was about monitor refresh rate. – Isaac Jun 5, 2016 at 15:44

1 It is 64 Hz, never 60. Tends to be 100 Hz when you use SO, browsers tinker with it :) – Hans Passant Jun 5, 2016 at 15:51

1 It varies quite a bit on modern computers. Energy management and applications can adjust it. SysInternals have developed a tool to monitor it: technet.microsoft.com/en-us/sysinternals/bb897568.aspx – Cee McSharpface Jun 5, 2016 at 15:55

Built into Windows, `powercfg.exe /energy` is pretty slick. – Hans Passant Jun 5, 2016 at 16:06

3

.NET relies on the API. MSDN has to say this about the API:

https://msdn.microsoft.com/de-de/library/windows/desktop/ms724961(v=vs.85).aspx

> When the system first starts, it sets the system time to a value based on the real-time clock of

> the computer and then regularly updates the time [...] GetSystemTime copies the time to a SYSTEMTIME [...]

I have found no reliable sources to back up my claim that it is stored as `SYSTEMTIME` structure, updated therein, and just copied into the receiving buffer of `GetSystemTime` when called. The smallest logical unit is 100ns from the NtQuerySystemTime system call, but we end up with 1 millisecond in the CLR's `DateTime` object. Resolution is not always the same.

We might be able to figure that out for Mono on Linux, but hardly for Windows given that the API code itself is not public. So here is an assumption: Current time is a variable in the kernel address space. It will be updated by the OS (frequently by the system clock timer interrupt, less frequently maybe from a network source -- the documentation mentions that callers may not rely on monotonic behavior, as a network sync can correct the current time backwards). The OS will synchronize access to prevent concurrent writing but otherwise it will not be an I/O-expensive operation.

On recent computers, the timer interval is no longer fixed, and can be controlled by the BIOS and OS. Applications can even request lower or higher clock rates: https://randomascii.wordpress.com/2013/07/08/windows-timer-resolution-megawatts-wasted

Share Improve this answer edited Jun 6, 2016 at 9:24

Follow

answered Jun 5, 2016 at 15:36

Cee McSharpface
8,728 ● 3 ● 39 ● 85

1 The smallest logical unit is 100 nanoseconds, actually.
– Joker_vD Jun 6, 2016 at 5:53

well, there is the `DateTime.Millisecond` member in the CLR which is documented as "The milliseconds component, expressed as a value between 0 and 999" and the `_SYSTEMTIME.wMilliseconds` field documented as "The millisecond. The valid values for this member are 0 through 999". We would have to look into `GetSystemTimePreciseAsFileTime` or this for ns or even µs, enough material for a question on its own
– Cee McSharpface Jun 6, 2016 at 9:07