Опять про программное измерение времени.

Попытка сведения характеристик разных функций получения времени в одном месте. Что интересовало: накладные расходы, реальная точность, особенности. Были померены около 40 функций в Windows и Linux.

Disclaimer: этот текст был написан как результат удовлетворения личного любопытства автора, и не претендует на точность и полноту. Автор готов присоединиться к общим шуткам по поводу использования нано- и ПИКОсекунд в бытовых, не Realtime операционных системах. Измерять время в обычных многопоточных ОС дело неблагодарное. В любой момент процесс может быть прерван планировщиком потоков или прерыванием. Кроме того, современные компиляторы могут сильно изменить код в зависимости от его контекста, да и сам процессор имеет право менять порядок выполнения некоторых инструкций. Так же на скорость выполнения кода огромное влияние оказывает использование кэша процессора.

Описанные эксперименты не привели к каким-либо открытиям, все выводы широко известны, и всё же надеюсь, что кому-то они могут оказаться полезны.

Получаем два значения времени в начале и конце измеряемого интервала и путём простого вычитания узнаём искомую длительность. Это же элементарно! Вот только какую ф-ю использовать? С первых версий C описывает ф-ии clock() и time(), в С11 добавили ещё timespec\_get(). С++ с 17-го стандарта получил методы now() трёх классов часов… А может правильнее использовать API-шные ф-ии, не зря же их столько написали. В Windows документированы более 18 способов получения времени непереносимым способом, в Linux их более 11.

В стандарте C++ описываются как минимум 6 функций для получения времени. В дополнение к ним WinAPI добавляет ещё 18, а LINUX 11 способов и это не всё. Объясняется это многообразие тем, что часть функций унаследованы ради обратной совместимости,

 The primary API for native code is [QueryPerformanceCounter (QPC)](https://learn.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter). < https://learn.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps >

На моей системе с Win10 high\_resolution\_clock::now() вызывает QueryPerfomanceCounter()

Часы high\_resolution\_clock могут быть не монотонными (например в Linux)! Это может означать, что часы могут подводиться не только при сезонном переводе часов, но и при коррекции ухода часов от эталона. Коррекция спешащих часов может привести к вычислению отрицательной или нулевой величины интервала.

Прогрев ядер процессора: частота ядра может понижаться при простое или при достижении предела, определённого тепловым пакетом.

В ОС Windows ф-я count() нарушает стандарт и такты не зависимо от сна процесса.

TSC - the processor’s time stamp counter.

Q: Is the performance counter monotonic (non-decreasing)?

A: Yes. QPC does not go backward.

UTC-synchronized time stamps functions:  [GetSystemTimeAsFileTime](https://learn.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemtimeasfiletime), QueryInterruptTimePrecise

 The system clock timer is the hardware timer that periodically generates interrupts for the system clock. The uniform period between system clock timer interrupts is referred to as a system clock tick, and is typically in the range of 0.5 milliseconds to 15.625 milliseconds, depending on the hardware platform.

С первых версий языка C были описаны ф-ии time() и clock(), первая возвращала время, а вторая тики пригодные для замеров затрат ресурсов CPU процессом. Поскольку в те времена MS DOS не поддерживала многопоточность, то и реализация clock() у Microsoft просто возвращала число миллисекунд с момента запуска системы, так и осталось до си пор – реализация clock в Windows не соответствует стандарту.

The computational calling cost of [QPC](https://learn.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter) is determined primarily by the underlying hardware platform. If the TSC register is used as the basis for QPC, the computational cost is determined primarily by how long the processor takes to process an RDTSC instruction. This time ranges from 10s of CPU cycles to several hundred CPU cycles depending upon the processor used. If the TSC can't be used, the system will select a different hardware time basis. Because these time bases are located on the motherboard (for example, on the PCI South Bridge or PCH), the per-call computational cost is higher than the TSC, and is frequently in the vicinity of 0.8 - 1.0 microseconds depending on processor speed and other hardware factors. This cost is dominated by the time required to access the hardware device on the motherboard.

Absolute Clocks and Difference Clocks. Первые предоставляют время, синхронизированное с мировым (UTC), потому нуждаются в периодической корректировке. Вторые монотонны и обеспечивают более точные измерения интервалов времени.

Для некоторых функций пренебрегать ошибкой квантования нельзя, например time().

Оптимизатор может выкинуть (и часто выкидывает) «лишний» код, это затрудняет написание искусственных тестов для измерения производительности. Например, простейший приём – выполнить N раз измеряемый код и потраченное время поделить на N, оптимизатор может сократить до однократного выполнения кода, сочтя, что остальные не приводят к видимому эффекту, а то и вовсе выкинуть измеряемый код.

1. Не монотонное;
2. Монотонное;
   1. Мировое время;
   2. Время Процесса;
   3. Время потока;
3. Разрешение;
4. Точность;
5. Дороговизна;
6. Переносимость;

time-of-day clock

“time slice is small (approximately 20 milliseconds)” <https://learn.microsoft.com/en-us/windows/win32/procthread/multitasking>

Функция clock\_gettime в LINUX поддерживает все виды времени:

1. **CLOCK\_REALTIME**
2. **CLOCK\_MONOTONIC**
3. **CLOCK\_MONOTONIC\_RAW**

**И т.д.**

О времени: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/about-time>

CreateTimerQueueTimer

NtDelayExecution