Measure time in Linux - time vs clock vs getrusage vs clock_gettime vs gettimeofday vs timespec_get?

Asked 11 years ago Modified 6 months ago Viewed 82k times

175

Among the timing functions, `time`, `clock` `getrusage`, `clock_gettime`, `gettimeofday` and `timespec_get`, I want to understand clearly how they are implemented and what are their return values in order to know in which situation I have to use them.

First we need to classify functions returning **wall-clock values** compare to functions returning **process or threads values**. `gettimeofday` returns wall-clock value, `clock_gettime` returns wall-clock value **or** process or threads values depending on the `Clock` parameter passed to it. `getrusage` and `clock` return process values.

Then the second question regards the implementation of these functions and as a consequence, their accuracy. Which hardware or software mechanism does these functions use.

It seems that `getrusage` uses only the kernel tick (usually 1ms long) and as a consequence can't be more accurate than the ms. Is it right? Then the `gettimeofday` function seems to use the most accurate underlying hardware available. As a consequence its accuracy is usually the microsecond (can't be more because of the API) on recent hardware. What about `clock`, the man page speak about "approximation", what does it mean? What about `clock_gettime`, the API is in nanosecond, does it means that it's able to be so accurate if underlying hardware allows it? What about monotonicity?

Are there any other functions?

c linux time linux-kernel

Share Follow

edited Mar 28 at 13:18
Patrizio Bertoni
2,582 ● 31 ● 43

asked Sep 12, 2012 at 16:05
Manuel Selva
18.6k ● 22 ● 89 ● 136

2 Answers

Sorted by: Highest score (default)

Não encontrou uma resposta? Pergunte em Stack Overflow em Português. ×

224

The problem is that there are several different time functions available in C and C++, and some of them vary in behavior between implementations. There are also a lot of half-answers floating around. Compiling a list of clock functions together with their properties would answer the question properly. For starts let's ask what the relevant properties are that we're looking for. Looking at your post, I suggest:

- What time is measured by the clock? (real, user, system, or, hopefully not, wall-clock?)
- What is the precision of the clock? (s, ms, µs, or faster?)
- After how much time does the clock wrap around? Or is there some mechanism to avoid this?
- Is the clock monotonic, or will it change with changes in the system time (via NTP, time zone, daylight savings time, by the user, etc.)?

- How do the above vary between implementations?
- Is the specific function obsolete, non standard, etc.?

Before starting the list, I'd like to point out that wall-clock time is rarely the right time to use, whereas it changes with time zone changes, daylight savings time changes, or if the wall clock is synchronized by NTP. None of these things are good if you're using the time to schedule events or to benchmark performance. It's only really good for what the name says, a clock on the wall (or desktop).

Here's what I've found so far for clocks in Linux and OS X:

- `time()` returns the wall-clock time from the OS, with precision in seconds.
- `clock()` seems to return the sum of user and system time. It is present in C89 and later. At one time this was supposed to be the CPU time in cycles, but modern standards like POSIX require CLOCKS_PER_SEC to be 1000000, giving a maximum possible precision of 1 µs. The precision on my system is indeed 1 µs. This clock wraps around once it tops out (this typically happens after ~2^32 ticks, which is not very long for a 1 MHz clock). `man clock` says that since glibc 2.18 it is implemented with `clock_gettime(CLOCK_PROCESS_CPUTIME_ID, ...)` in Linux.
- `clock_gettime(CLOCK_MONOTONIC, ...)` provides nanosecond resolution, is monotonic. I believe the 'seconds' and 'nanoseconds' are stored separately, each in 32-bit counters. Thus, any wrap-around would occur after many dozen years of uptime. This looks like a very good clock, but unfortunately it isn't yet available on OS X. POSIX 7 describes `CLOCK_MONOTONIC` as an optional extension.
- `getrusage()` turned out to be the best choice for my situation. It reports the user and system times separately and does not wrap around. The precision on my system is 1 µs, but I also tested it on a Linux system (Red Hat 4.1.2-48 with GCC 4.1.2) and there the precision was only 1 ms.
- `gettimeofday()` returns the wall-clock time with (nominally) µs precision. On my system this clock does seem to have µs precision, but this is not guaranteed, because "the resolution of the system clock is hardware dependent". POSIX.1-2008 says that. "Applications should use the `clock_gettime()` function instead of the obsolescent `gettimeofday()` function", so you should stay away from it. Linux x86 and implements it as a system call.
- `mach_absolute_time()` is an option for very high resolution (ns) timing on OS X. On my system, this does indeed give ns resolution. In principle this clock wraps around, however it is storing ns using a 64-bit unsigned integer, so the wrapping around shouldn't be an issue in practice. Portability is questionable.
- I wrote a hybrid function based on this snippet that uses clock_gettime when compiled on Linux, or a Mach timer when compiled on OS X, in order to get ns precision on both Linux and OS X.

All of the above exist in both Linux and OS X except where otherwise specified. "My system" in the above is an Apple running OS X 10.8.3 with GCC 4.7.2 from MacPorts.

Finally, here is a list of references that I found helpful in addition to the links above:

- http://blog.habets.pp.se/2010/09/gettimeofday-should-never-be-used-to-measure-time
- How to measure the ACTUAL execution time of a C program under Linux?
- http://digitalsandwich.com/archives/27-benchmarking-misconceptions-microtime-vs-getrusage.html
- http://www.unix.com/hp-ux/38937-getrusage.html

Update: for OS X, `clock_gettime` has been implemented as of 10.12 (Sierra). Also, both POSIX and BSD based platforms (like OS X) share the `rusage.ru_utime` struct field.

Share Follow

edited May 23, 2017 at 12:18
Community Bot
1 ● 1

answered Sep 18, 2012 at 15:45
Douglas B. Staple
10.6k ● 8 ● 32 ● 58

Mac OS X doesn't have `clock_gettime`, hence the use of `gettimeofday()` being a little more versatile than `clock_gettime()` – bobobobo Apr 17, 2013 at 4:30

1 You haven't mentioned `times()` (with an s), which has existed in POSIX since Issue 1. Concerning GNU/Linux: According to the clock(3) man page, `clock()` from glibc 2.17 and earlier was implemented on top of it, but for improved precision, it is now implemented on top of `clock_gettime(CLOCK_PROCESS_CPUTIME_ID,...)`, which is also specified in POSIX but is optional. – vinc17 Sep 11, 2014 at 15:02

2 @starflyer The precision of the clock is partly limited by the amount of time it takes to poll the clock. This is because, if I call the clock and it takes 1 µs to return, then the time the clock reports will be "off" by 1 µs from the perspective of the caller. This means that a highly accurate clock must also be low-latency. So typically one will not have the trade-off you're talking about: the cheapest clocks will also be the most accurate. – Douglas B. Staple Mar 23, 2015 at 23:53

3 Also most clocks do not care about daylight saving time / time zones, even if they are considered to be *wall clocks*. Both `time` and `gettimeofday` return, at least nowadays, seconds since epoch (a.k.a. unix-timestamps). This is independent of time zones / DST. Leap seconds are another story... – Zulan May 7, 2016 at 7:18

2 For Android users, using CLOCK_MONOTONIC may be problematic since the app may get suspended, along with the clock. For that, Android added the ANDROID_ALARM_ELAPSED_REALTIME timer that is accessible through ioctl. some information on these and other suspend related information can be found here – Itay Bianco Jul 11, 2016 at 7:15

24

C11 `timespec_get`

Usage example at: https://stackoverflow.com/a/36095407/895245

The maximum possible precision returned is nanoseconds, but the actual precision is implementation defined and could be smaller.

It returns wall time, not CPU usage.

glibc 2.21 implements it under `sysdeps/posix/timespec_get.c` and it forwards directly to:

```
clock_gettime (CLOCK_REALTIME, ts) < 0)
```

`clock_gettime` and `CLOCK_REALTIME` are POSIX http://pubs.opengroup.org/onlinepubs/9699919799/functions/clock_getres.html, and `man clock_gettime` says that this measure may have discontinuities if you change some system time setting while your program runs.

C++11 chrono

Since we're at it, let's cover them as well: http://en.cppreference.com/w/cpp/chrono

GCC 5.3.0 (C++ stdlib is inside GCC source):

- `high_resolution_clock` is an alias for `system_clock`
- `system_clock` forwards to the first of the following that is available:
  - `clock_gettime(CLOCK_REALTIME, ...)`
  - `gettimeofday`

  - `time`
- `steady_clock` forwards to the first of the following that is available:
  - `clock_gettime(CLOCK_MONOTONIC, ...)`
  - `system_clock`

Asked at: Difference between std::system_clock and std::steady_clock?

`CLOCK_REALTIME` vs `CLOCK_MONOTONIC`: Difference between CLOCK_REALTIME and CLOCK_MONOTONIC?

Share Follow edited Jul 6, 2017 at 10:05 answered Apr 18, 2016 at 17:14

Ciro Santilli OurBigBook.com
350k ● 103 ● 1203 ● 991

1 Great answer that demystifies the typical implementations. That's what people really need to know. – Beeeaaar Aug 24, 2018 at 21:56