



Join the Stack Overflow Community

Stack Overflow is a community of 7.1 million programmers, just like you, helping each other. Join them; it only takes a minute:

Sign up

gettimeofday() C++ Inconsistency




[Get started](#)

I'm doing a project that involves comparing programming languages. I'm computing the Ackermann function. I tested Java, Python, and Ruby, and got responses between 10 and 30 milliseconds. But C++ seems to take 125 milliseconds. Is this normal, or is it a problem with the gettimeofday() ? Gettimeofday() is in time.h.

I'm testing on a (virtual) Ubuntu Natty Narwhal 32-bit. I'm not short processing power (Quad-core 2.13 GHz Intel Xeon).


My code is here:

```
#include <iostream>
#include <sys/time.h>
using namespace std;
int a(int m,int n) {
    if (m == 0) {
        return n + 1;
    } else if (m > 0 and n == 0) {
        return a(m-1,1);
    } else if (m > 0 and n > 0) {
        return a(m-1,a(m,n-1));
    }
}

int main() {
    timeval tim;
    gettimeofday(&tim,NULL);
    double t1 = tim.tv_usec;
    int v = a(3,4);
    gettimeofday(&tim,NULL);
    double t2 = tim.tv_usec;
    cout << v << endl << t2-t1;
    return 0;
}
```


[time](#) [programming-languages](#) [benchmarking](#) [stopwatch](#)

edited Aug 22 '14 at 2:50

Jonathan Leffler

474k68550877

asked Oct 13 '11 at 22:01

singerng

3482413

- 1 What platform are you running on? And what actual code are you using to take the measurement? – [Clare Macrae](#) Oct 13 '11 at 22:03
- How did you test this? – [Foo Bah](#) Oct 13 '11 at 22:03
- Could you at least show us your code to see how you are doing it? – [K-ballo](#) Oct 13 '11 at 22:03

1 Answer

Assuming you're talking about the *resolution* of the data returned ^(a), [the POSIX specification](#) for gettimeofday states:

The resolution of the system clock is unspecified.

This is due to the fact that systems may have a widely varying capacity for tracking small time periods. Even the ISO standard clock() function includes caveats like this.

If you're talking about how *long* it takes to call it ^(a), the standard makes no guarantees about performance along those lines. An implementation is perfectly free to wait 125 *minutes* before giving you the time although I doubt such an implementation would have much market success :-)

As an example of the limited resolution, I typed in the following code to check it:

```
#include <stdio.h>
#include <sys/time.h>

#define NUMBER 30

int main (void) {
    struct timeval tv[NUMBER];
    int count[NUMBER], i, diff;

    gettimeofday (&tv[0], NULL);

    for (i = 1; i < NUMBER; i++) {
        gettimeofday (&tv[i], NULL);
        count[i] = 1;
        while ((tv[i].tv_sec == tv[i-1].tv_sec) &&
            (tv[i].tv_usec == tv[i-1].tv_usec))
        {
            gettimeofday (&tv[i], NULL);
            count[i]++;
        }
    }

    printf ("%2d: secs = %d, usecs = %6d\n", 0, tv[0].tv_sec, tv[0].tv_usec);
    for (i = 1; i < NUMBER; i++) {
        diff = (tv[i].tv_sec - tv[i-1].tv_sec) * 1000000;
        diff += tv[i].tv_usec - tv[i-1].tv_usec;

        printf ("%2d: secs = %d, usecs = %6d, count = %5d, diff = %d\n",
            i, tv[i].tv_sec, tv[i].tv_usec, count[i], diff);
    }

    return 0;
}
```

The code basically records the changes in the underlying time, keeping a count of how many calls it took to `gettimeofday()` for the time to actually change. This is on my new i7 grunter laptop so it's not short on processing power (the count indicates how often it was able to call `gettimeofday()` for each time quantum, around the 5,000 mark).

The output was:

```
0: secs = 1318554836, usecs = 990820
1: secs = 1318554836, usecs = 991820, count = 5129, diff = 1000
2: secs = 1318554836, usecs = 992820, count = 5807, diff = 1000
3: secs = 1318554836, usecs = 993820, count = 5901, diff = 1000
4: secs = 1318554836, usecs = 994820, count = 5916, diff = 1000
5: secs = 1318554836, usecs = 995820, count = 5925, diff = 1000
6: secs = 1318554836, usecs = 996820, count = 5814, diff = 1000
7: secs = 1318554836, usecs = 997820, count = 5814, diff = 1000
8: secs = 1318554836, usecs = 998820, count = 5819, diff = 1000
9: secs = 1318554836, usecs = 999820, count = 5901, diff = 1000
10: secs = 1318554837, usecs = 820, count = 5815, diff = 1000
11: secs = 1318554837, usecs = 1820, count = 5866, diff = 1000
12: secs = 1318554837, usecs = 2820, count = 5849, diff = 1000
13: secs = 1318554837, usecs = 3820, count = 5857, diff = 1000
14: secs = 1318554837, usecs = 4820, count = 5867, diff = 1000
15: secs = 1318554837, usecs = 5820, count = 5852, diff = 1000
16: secs = 1318554837, usecs = 6820, count = 5865, diff = 1000
17: secs = 1318554837, usecs = 7820, count = 5867, diff = 1000
18: secs = 1318554837, usecs = 8820, count = 5885, diff = 1000
19: secs = 1318554837, usecs = 9820, count = 5864, diff = 1000
20: secs = 1318554837, usecs = 10820, count = 5918, diff = 1000
21: secs = 1318554837, usecs = 11820, count = 5869, diff = 1000
22: secs = 1318554837, usecs = 12820, count = 5866, diff = 1000
23: secs = 1318554837, usecs = 13820, count = 5875, diff = 1000
24: secs = 1318554837, usecs = 14820, count = 5925, diff = 1000
25: secs = 1318554837, usecs = 15820, count = 5870, diff = 1000
26: secs = 1318554837, usecs = 16820, count = 5877, diff = 1000
27: secs = 1318554837, usecs = 17820, count = 5868, diff = 1000
28: secs = 1318554837, usecs = 18820, count = 5874, diff = 1000
29: secs = 1318554837, usecs = 19820, count = 5862, diff = 1000
```

showing that the resolution seems to be limited to no better than one thousand microseconds. Of course, your system may be different to that, the bottom line is that it depends on your implementation and/or environment.

^(a) If you're talking about something *else* that I haven't thought of, you should probably flesh out your question with a little more detail. We're pretty good here on SO, but we're not omniscient.

edited Oct 14 '11 at 1:15

answered Oct 13 '11 at 22:06



paxdiablo

528k 129 1059
1508