Time

Time measurements

- World time
- An interval of time
- An alarm

Time measurements

Time measurements are never accurate

- Clock granularity
- Latency

Where it matters (in benchmarks), also report the accuracy of measurements

Beware, measurements themselves affect the timing results

Multiple time sources

Better to rely on a single time source than a combination of several time sources

- Independent clocks show different times and advance with different paces. They may also differ in granularity and latency
- It is very difficult to reasonably combine several clocks (except of world time synchronization)

Hardware time sources (PC)

- Real-time clock (RTC)
  - An independent processor + a small CMOS RAM
  - Keeps the world time
  - Ticks even when the PC is unplugged (battery)
  - Can issue periodic interrupts (IRQ 8) at 2 Hz-8192 Hz, these are rarely used in current system configurations
  - OS can read and set time via designated I/O ports (0x70, 0x71)

Hardware time sources (PC)

- Programmable Interval Timer (PIT)
  - Intel 8254
  - Ticks at ca. 1 MHz
  - Used to generate alarms (IRQ 0), either single or periodic ones
  - In Linux, the periodic alarm frequency is ca. 1000 Hz, i.e. 1 millisecond (not long ago, it used to be 100 Hz). In Windows, this frequency can be set in run-time, but it is usually 1000 Hz
- PIT is being replaced in PCs with HPET (High Precision Event Timer) with finer granularity (100 ns)

Hardware time sources (PC)

- Time Stamp Counter (TSC)
  - Does not measure world-time
  - A 64-bit register counting processor cycles, can be read via the instruction rdtsc
  - Based either on periodic CLK input signal (from an external oscillator) or a built-in oscillator (cores running at different frequencies)
  - The frequency is ca. GHz

Hardware time sources (PC)

In PCs with several processors (cores):

- PIT is being replaced with APIC which generates alarms locally, not system-wide
- TSC is always local to one core

So, if time-stamps must be monotonic (always increasing), a system-wide synchronization of TSCs is required

Programmer‘s view (Linux): world time

time() sec

gettimeofday() usec

clock_gettime(CLOCK_REALTIME) usec

clock_gettime(CLOCK_REALTIME_COARSE) usec

Reads a non-monotonic real-time clock

Reflects adjustments such as NTP corrections, time zone changes, summer/winter time, …

Used to generate real-time stamps (e.g. file modification)

COARSE_REALTIME is a faster version of REALTIME, it avoids the assistance of the kernel using VDSO (Virtual Dynamic Shared Object) which has replaced vsyscall()

Programmer‘s view (Linux): alarm

select() usec

sleep() sec

usleep() usec

nanosleep() nsec

Deschedules the caller for the given amount of time (unless the alarm is very soon)

The accuracy depends on the granularity of the time source. E.g. nanosleep() is not necessarily more precise than usleep()

Programmer‘s view (Linux): time interval

somehow_get_time(&start);

work();

somehow_get_time(&end);

elapsed_time = end – start;

Assuming that somehow_get_time() reads a clock with a real-time pace, this measures the real-time spent in work()

But sometimes we may not be interested in the elapsed real time. If work() contains system calls, we may be interested only in the time spent in work() while it was on the CPU

Programmer‘s view (Linux): time interval

getrusage() usec

Integrated time used by the calling process (user and system time separated)

gettimeofday() usec, ftime() msec

Always advances, regularly (but subject to e.g. NTP adjustments, …)

clock_gettime(CLOCK_MONOTONIC) nsec

Always advances. But not regularly (subject to e.g. NTP adjustments, …)

clock_gettime(CLOCK_MONOTONIC_COARSE) nsec

A faster, VDSO version

clock_gettime(CLOCK_MONOTONIC_RAW) nsec

Always advances. But not regularly (although not subject to NTP adjustments)

Programmer‘s view (Linux): time interval

getrusage() usec

Integrated time used by the calling process (user and system time separated)

clock_gettime(CLOCK_PROCESS_CPUTIME_ID) usec

Integrated user time of this process (over all threads)

clock_gettime(CLOCK_THREAD_CPUTIME_ID) usec

Integrated user time of this thread

Programmer‘s view

There is no simple answer to how to measure time. Aspects such as portability may be very important (besides technical issues such as wrapping clock around, high latency etc.)

Designing an appropriate benchmark is art (a one-time project). It is important to understand what is being measured (and why)

Advice: do not fiddle with time in production code (especially in higher levels, e.g. Java)

If your program’s correct behaviour depends on real time, then you do something wrong, perhaps except of when the program directly controls a hardware device with real time constraints

This also applies to the design of operating systems (and processors)