Measuring time seems like it should be simple. We have been designing clocks for at least 4000 years. Modern computer systems have multiple timers used for, among other purposes, time-keeping. These timers have different purposes and thus different trade-offs in features. Further complicating matters is the plethora of APIs to access the clocks.

Clocks are really just counters. Something generates events and the counter counts them. Generally, some circuit or crystal osculates at a fixed frequency. Each osculation is counted by the counter. Thus if you know the frequency, you can compute the time elapsed between two measurements of the counter.

For at least: `gettimeofday`, `RDTSC`, and `clock_gettime` (all supported clocks) measure the precision and overhead of timing a region of code. How long does it take to measure a region (overhead of polling the timer)? What is the smallest time difference in time reportable with the timer (note that this could be shorter than the time it takes to retrieve the timer value)? How does this compare with the units used to report? E.g. just because a clock reports time in units of nanocenturies doesn’t mean it is incremented more than every day.

What are the limitations of each clock? Some drawbacks of each clock may not be obvious just from experimentation. For example, some clocks may not be synchronized between cores, some clocks may be measuring ticks of a non-constant frequency sources, some clocks may be of program time, while others are of wall-clock time, some clocks may be affected by power-savings (either hibernation of simply frequency scaling), etc.

This homework may be done on any x86 Linux system. Provide the model of the CPU you use, as well as the kernel version. Explain the methodology you use to make your measurements, as well as present the results. Turn in by emailing a pdf and the source to the TA (lijialiucs@gmail.com) with subject line [cs378 HW1] and your UTID.

Note: inline assembly is required to issue a rdtsc instruction, but numerous examples of how to do this are easy to find.

**Bonus Questions (Extra Credit)**

Many operating systems, Linux included, try to synchronize the various per-CPU clocks to make them usable for processes even if the process is migrated to a new core. For the highest resolution clock you found, propose a way of synchronizing the clocks (no code required), and compute the maximum error between the clocks after synchronization. You cannot computed a single number as an answer, you will need to phrase it as a function of some cost of your synchronization.

Why does Linux kernel version and CPU model matter? Have there been changes in what guarantees the kernel provides for each type of timer? How has Intel’s implementation of CPU timers changed in terms of how they behave?

The C++11 standard defines 3 types of clocks. How are they implemented in Linux?