Bandwidth and Latency Measurement on Inter-Process Communications

Kedar Bastakoti, Ruohui Wang

Abstract

There are multiple ways to do inter-process communications(IPC). For example, pipe is often used to communicate between processes, whereas TCP and UDP can also be used to communicate both locally and remotely. Due to their differences in design, they tend to have different latencies and througputs. Experiments to show

1 Clock Resolution

To begin with, how can we measure the time? We need clocks to do so. The Linux operating system, along with the underlying hardware, provides multiple methods for such purpose. But how can we choose one? One of the criteria is how precise they can be. And resolution is an important standard for the precision.

The resolution is the smallest possible increase of the clock. In order to measure the resolution, we tried to create minimal possible differences between two time measurements. We create such difference by inserting an line of assembly `nop` into the code.

As per the POSIX manual**?**, the function `gettimeofday` is obsolete, and the switch to `clock_gettime` is recommended, so we used `clock_gettime` instead. along with `clock_gettime`, function `clock_getres` is provided for user to query the resolution of time. As we will demonstrated, it produces the same result as ours.

Also, the x86 CPU provides `rdtscp` instruction to give the CPU timestamp, in

terms of TSC(Time Stamp Counter) cycles. As Sergiu and Terry pointed out, in order to use TSC as a reliable clocksource, it must be stable and have a constant rate, which can be examined using `\proc\cpuinfo`?.

It is worth noting that accuracy is also a key factor to consider, which we will leave to the discussion.

2 System Calls

Since we are measuring I/O performance, we cannot avoid doing a lot of system calls. System calls involving context switching, and trapping into the kernel, both of which might . In order to get an better idea of how these system calls take up time. We measured some trivial system calls to see if the influence is non-trivial.

In order to measure the impact of trapping, we repeatedly make system calls that do least extra work. Here, we chose `getpid` and `getuid` as our syscalls. In addition, to avoid function returns that may take up time, we directly used the inline assembly to call the `syscall` instruction. And we return the minimal result in order to eliminate the effects random events including page faults and preemptions. As the results shows, whether using the assembly `syscall` has a great effect on the time measured.

3 Pipe

4 TCP

5 UDP

6 Evaluation