Linux Tutorial

TA for class CSE 520S, Fall
09/20/2017
Haoran Li
Outline

- Latency Measurement
  - in a single machine
  - between machines

- Real-Time Schedulers
Measure Elapsed Time
Why Elapsed Time First?

- **End-to-End Latency**
  - Get absolute time on each end?

- **Elapsed Time:**
  - Round Trip Time (ping)
  - Same Clock Source

---

Synchronized / Asynchronized Clocks?

- Clock 1
  - Start_time (clock 1)
- Clock 2
  - end_time (clock 2)

Measure Time on a single Host

- Clock 1
  - Start_time (clock 1)
  - End_time (clock 1)
Get a coarse grained estimation

- Use shell built-in command
  - `time`

  - “real”: Wall Time Elapsed (Just an estimation, Don’t rely on it)
  - “user”: Execution Time in User Space
  - “sys”: Execution Time in Kernel Space (syscall)

Get an overview of your program’s response / execution time.
Measure Elapse Time: gettimeofday()

- gettimeofday() [http://linux.die.net/man/2/sched_setscheduler]
  - return struct timeval, includes tv_sec and tv_usec
  - **NOT** ok for measuring **overhead** on standard kernel configuration
  - Wall clock time may change
    - User/other program (NTP) changes clock

```
lihaoran@MBP15:Latency_Measurement$ ./gettod
4978670
lihaoran@MBP15:Latency_Measurement$ ./gettod
-86427362948
lihaoran@MBP15:Latency_Measurement$
```

Normal Evaluation (5 seconds)

```
lihaoran@MBP15:Latency_Measurement$ timedatectl status
Local time: Tue 2017-09-19 00:00:25 CDT
Universal time: Tue 2017-09-19 05:00:25 UTC
RTC time: Tue 2017-09-19 05:00:25
Time zone: America/Chicago (CDT, -0500)
Network time on: no
NTP synchronized: no
RTC in local TZ: no
lihaoran@MBP15:Latency_Measurement$ timedatectl set-time 2017-09-18
```

Intentionally change the wall clock, when program is running
Don’t Relied on Wall-Time Clock

- Shell built-in command “time”

When Measuring Elapse Time, don’t rely on WTC.
Use POSIX clock_gettime()

- Sources:
  - CLOCK_REALTIME: Wall Time, affected by discontinuous jump
  - CLOCK_MONOTIC: **Not** affected by discontinuous jump, but affected by incremental adjust (e.g. NTP).
    - Clock cannot jump, but may skew
  - CLOCK_MONOTIC_RAW: Not affected by NTP
    - More accurate for very short intervals

- Precision: Clock_getres()

For short latency measurement in a single host, try CLOCK_MONOTIC_RAW
Use Processor Cycles (x86 x64)

- FIX CPU Frequency
- Disable Hyperthreading

- Rdtsc
  - read CPU cycles directly (need to fix CPU frequency)
  - cat /proc/cpuinfo to get CPU frequency
  - on a 1GHZ CPU, ticks 1,000,000,000 times per second
    • if you use rdtsc to record time, pay attention to this value
    • cat /proc/cpuinfo  # get CPU frequency
    • cat /sys/devices/system/clocksource/clocksource0/current_clocksource

Generally speaking, rdtsc() gives you best accuracy and trivial overhead. However, in multi-core system, Cycle counters for different cores are not always synchronized!
Inter-domain TSC Synchronization
(x86 x64)

- Issues: Non-Synchronized Clocks / Offsets
  - Clock Tree + DPLL

- Workaround:
  - Enable **Invariant TSC** (INTEL, since **Haswell**)
  - `rdtscp()`[1]

---

[1] How to benchmark code execution Times on Intel:
FYI: Other Time Sources

RTC (Real-Time Clock)
- Available on most computers (not on RPi 2 or 3 unless you add it)
- Low precision (as low as 0.5 seconds)

Hardware Timers
- Might be used to generate interrupts, might be queryable
- Run at a variety of frequencies
- Programmable Interval Timer (PIT)
- High-Performance Event Timer (HPET)
- Programmable Interrupt Controller (PIC)
- Advanced Programmable Interrupt Controller (APIC)

Processor Cycles
- Timestamp Counter (TSC) on x86, 64-bit
- Cycle Counter (CCNT) on ARM, 32-bit, 64-cycle divider, not accessible in user mode
- Potentially very high accuracy

Pointers

- **Time related system calls in the Linux kernel**
  - [https://0xax.gitbooks.io/linux-insides/content/Timers/timers-7.html](https://0xax.gitbooks.io/linux-insides/content/Timers/timers-7.html)

- **Clock_gettime()**
  - [Man 2 clock_gettime](https://linux.die.net/man/2/clock_gettime)
  - [https://linux.die.net/man/2/clock_gettime](https://linux.die.net/man/2/clock_gettime)

- **RDTSC**

- **RDTSCP**

- **Invariant TSC**
  - Pitfall of TSC usage

- **(ARM) High Resolution Timing on Raspberry Pi**
  - [https://blog.regehr.org/archives/794](https://blog.regehr.org/archives/794)
Measure End-to-End Latency
Why Elapsed Time First?

End-to-End Latency

Contributor:
- propagation delay: static
- queuing delays
- node processing delays
- routing changes
Get Clock Synchronized?

- Query NTP Server, get clock synchronized
Background: NTP

- Hierarchical NTP servers: Clock Strata
- Stratum 0: reference clock
- Stratum 1: primary time servers

The **U.S. Naval Observatory** Alternate Master Clock
Stratum 0: high-precision timekeeping devices
*atomic (cesium) clocks*

Sadly, Wall Time

- timedatectl: ubuntu
- ntpd: POSIX
- “Low” Precision:
  - Milliseconds ~ Seconds
  - ~1ms in Local Area Network, using your own NTP server\[1\]

Set up an NTP server: [https://ubuntuforums.org/showthread.php?t=862620](https://ubuntuforums.org/showthread.php?t=862620)
Precision Time Protocol: Better Accuracy

On LAN:

- sub-microsecond range
- making it suitable for measurement and control systems
- PTP v.s. NTP
  - hardware support present in various network interface controllers (NIC) and network switches.

Warning: We never tried to use PTP on EC2. I am asking you to do some research.
PTP vs. NTP

- **NTP**
  - NTP servers with hardware timestamping and a good oscillator
  - Server located as close to clients as possible
  - Switches are low latency and lightly loaded
  - Redundancy: Clients use multiple servers
  - Monitoring: servers monitor each other (peer stats)

- **PTP**
  - Switches are transparent clocks or boundary clocks
  - Or use telecom profile technology (ITU-T G.8265.1 or G. 8275.2)
  - Redundant Grand Masters monitor each other
  - Hardware slaves (e.g. PCIe card) when possible
Linux PTP

- `SO_TIMESTAMPS`
- Supports the Linux PTP Hardware Clock (PHC) subsystem by using the `clock_gettime` family of calls, including the new `clock_adjtimex` system call.

PTP daemon man page

- `sudo apt-get install ptpd`
- `man ptpd`

NTP v.s PTP: How do you get accuracy

Real-Time Scheduler in Linux
Multiple schedulers are implemented as different scheduling classes.

Normal:

- SCHED_NORMAL/SCHED_OTHER: regular, interactive CFS tasks
- SCHED_BATCH: low priority, non-interactive CFS tasks
- SCHED_IDLE: very low priority tasks

Real-time:

- SCHED_RR: round-robin
- SCHED_FIFO: first-in, first-out
- SCHED_DEADLINE: earliest deadline first
Real-time tasks execute repeatedly (usually are periodic) under some time constraint.

E.g., a task is released to execute every 5 msec, and each invocation has a deadline of 5 msec.

Separate priority range from nice:
- Priorities range from 1 (low) to 99 (high)
Real-Time OS Support

Goal is to achieve predictable execution:

Sources of uncertainty (and solutions):

- Scheduling preemptions (real-time scheduling)
- Interrupts (can mask interrupts)
- Migrations (can pin tasks to cores)
- OS latency & jitter (RT_PREEMPT patch set)

Round-robin scheduling

Among tasks of equal priority:
- Rotate through all tasks
- Each task gets a fixed time slice

Cannot run if higher priority tasks are runnable
**SCHED_FIFO**

*First-in, First-out* scheduling

- The first enqueued task of highest priority executes to completion.
- A task will only relinquish a processor when it completes, yields, or blocks.

Earliest Deadline First (EDF) scheduling

- Whichever task has next deadline gets to run
- Theory exists to analyze such systems
- Linux implements bandwidth reservation to prevent deadline abuse

Scheduler Setup – Basic

Two classes, would always schedule RT class first

- RT class: static priority, 1 (lowest) to 99 (highest)
  - Preemptive scheduling
  - SCHED_FIFO, SCHED_RR to schedule processes with same priority
  - Can be used to implement static priority (like rate monotonic)
  - SCHED_DEADLINE: Exists in Kernel, however, no libc wrapper:
    - You need write a Syscall Wrapper by yourself
    - Or: https://github.com/jlelli/schedtool-dl
  - Default Reserve 5% for other classes (50ms every 1s)
    - /proc/sys/kernel/sched_rt_period_us 1000000
    - /proc/sys/kernel/sched_rt_runtime_us 950000

- Non-RT class: SCHED_OTHER with Complete Fair Scheduler
Scheduler Setup – Priorities

- **chrt command** (can also check task priorities)
  

  ```
  sudo chrt -f -p 99 4800  # pid 4800 with priority 99 and fifo
  ```

- **sched_setscheduler**  
  
  [http://linux.die.net/man/2/sched_setscheduler](http://linux.die.net/man/2/sched_setscheduler)

```c
#include <sched.h>

int main() {
  ...
  struct sched_param sched;
  sched.sched_priority = 98;
  if (sched_setscheduler(getpid(), SCHED_FIFO, &sched) < 0) {
    exit(EXIT_FAILURE);
  }
  ...
}
```
Scheduler Setup – Affinities

- taskset command (can also check task affinities) [http://linux.die.net/man/2/sched_setscheduler]
  - sudo taskset -c 2,3 4800  # pid 4800 runs on cores 2-3

- sched_setaffinity [http://linux.die.net/man/2/sched_setscheduler]

```c
#include <sched.h>

int main() {
    ...
    unsigned long mask = 1;
    if (sched_setaffinity(getpid(), sizeof(mask), &mask) < 0) {
        exit(EXIT_FAILURE);
    }
    ...
}
```
Scheduler Setup – Preemptive

- Scheduler is triggered every HZ quantum

- cat /boot/config-* | grep CONFIG_HZ
  - For most desktops, value is 1000. ticked every 1ms

- CONFIG_NO_HZ = y
  - Temporarily disable timer interrupt when system is idle or there is only single task running

- CONFIG_HIGH_RES_TIMERS = y
  - [Link](http://elinux.org/High_Resolution_Timers)

- Can recompile kernel to change these values
Trace Your System by using ftrace

- **ftrace**
  - Traces the internal operations of the kernel
  - Static tracepoints within the kernel (event tracing)
    - Scheduling
    - Interrupts

- **Trace-cmd**
  - Front-End (user-level) utility for ftrace
  - Example:
    - `sudo trace-cmd record -e sched_switch .myapp`
    - Dump trace.dat

- **Kernel Shark**
  - GUI trace-cmd reader
    - `kernelshark trace.dat`

Kernel Shark: http://rostedt.homelinux.com/kernelshark/
A Typical Trace

[Diagram showing a trace with timeline and CPU events]
Generate Periodic Tasks

- Video decoding, sensor processing, etc.
- Use Signals and Timers

```c
struct sigaction sa;
...
sa.sa_sigaction = work;
sigaction(SIGRTMIN, &sa, NULL);
...
struct sigevent timer_event;
timer_event.sigev_signo = SIGRTMIN;
...
timer_create(CLOCK_REALTIME, &timer_event, &timer);
timer_settime(timer, TIMER_ABSTIME, &timerspec, NULL);
...```

- Many other approaches in pointers
Pointers

- Periodically running a task
  - [http://www.embedded-linux.co.uk/tutorial/periodic_threads](http://www.embedded-linux.co.uk/tutorial/periodic_threads)

- Video players
  - [https://wiki.litmus-rt.org/litmus/Publications](https://wiki.litmus-rt.org/litmus/Publications)

- Get time in Linux
  - gettimeofday: [http://linux.die.net/man/2/gettimeofday](http://linux.die.net/man/2/gettimeofday)

- Fix CPU frequencies
  - [http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt](http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt)
  - [https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling](https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling)
Pointers

- Linux schedulers

- Set priority
  - sched_setcheduler: [http://linux.die.net/man/2/sched_setscheduler](http://linux.die.net/man/2/sched_setscheduler)

- Set CPU affinity on multi-core:
  - taskset: [http://linux.die.net/man/1/taskset](http://linux.die.net/man/1/taskset)

- Linux real-time patches:
  - RTAI: [https://www.rtai.org/](https://www.rtai.org/)
  - SCHED_DEADLINE: [http://gitorious.org/sched_deadline](http://gitorious.org/sched_deadline)
Thank You for your attention!

TA for class CSE 520S, Fall
09/20/2017
Haoran Li