Real-Time Systems: Testing Your System in Linux

Jiangnan Liu
CSE 520S Spring
02/13/2020

With adaptions from Haoran Li
What is meant by Time?

- **Absolute time** since some fixed past event, e.g.:
  - Seconds since start of the Unix Epoch
    (00:00:00 UTC on 1 January 1970)
  - Seconds since system boot

- **Relative time**, E.g.:
  - Seconds between two events
  - Ten seconds into the future (from now)
  - Execution time of a program segment

- **World time**, E.g.:
  - January 30\textsuperscript{th}, 9:00 AM

- An OS must approximate time to provide time-based functions for users.
Measure Elapsed Time
Why Elapsed Time First?

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

  ![Diagram of clock synchronization]

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

  ![Diagram of elapsed time measurement]

Synchronized / Asynchronized Clocks?

Start_time (clock 1)  
end_time (clock 2)

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 Elapsed Time: gettimeofday

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

---

Normal Evaluation (5 seconds)

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

Intentionally change the wall clock, when program is running

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

```
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
```
Don’t Relied on Wall Clock Time

- Shell built-in command “time”

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

- **Sources:** [https://linux.die.net/man/3/clock_gettime](https://linux.die.net/man/3/clock_gettime)
  - **CLOCK_REALTIME:**
    - Wall Time, affected by discontinuous jump
  - **CLOCK_MONOTONIC:**
    - **Not** affected by discontinuous jump but affected by incremental adjustments (e.g. NTP).
    - Clock cannot jump, but may skew
  - **CLOCK_MONOTONIC_RAW:**
    - **Not** affected by NTP
    - More accurate for very short intervals

- **Precision:** clock_getres()

For short latency measurement in a single host, try **CLOCK_MONOTONIC_RAW**
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)

- 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 End-to-End Latency?

End-to-End Latency

Contributor:
- propagation delay: static
- queuing delays
- node processing delays
- routing changes

Need synchronization between clocks
How to Get Clock Synchronized?

- **NTP (Network Time Protocol)**
  - Query NTP Server, get clock synchronized

![Diagram showing how to get clock synchronized using NTP](image-url)
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

Problems with NTP

- Loose granularity:
  - Tens of milliseconds in public internet
  - Around one millisecond in Local Area Network\textsuperscript{[1]}

- Leap second handling:
  - Detects the announcement at least a few \textit{seconds} after the refclock has started to announce the leap second

- Security:
  - NTP has been used in Distributed Denial of Service attacks (DDos) \textsuperscript{[2]}

Set up an NTP server: [https://ubuntuforums.org/showthread.php?t=862620](https://ubuntuforums.org/showthread.php?t=862620)  
On LAN:

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

- **NTP**
  - Hierarchical NTP servers
  - Redundancy: Client-Server, Peer-to-Peer, Multicast
  - Granularity: Tens of microseconds
  - Built-in in EC2

- **PTP**
  - Grand Master-Slave(Master)-Slave
  - Redundancy: Fail take over
  - Hardware slaves (e.g. PCIe card) when possible
  - Granularity: Sub-microseconds (when w/ hardware support)
  - Not built-in in EC2, normally used for LAN
Pointers

- **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**
Summary

- Elapsed time or end-to-end latency?
- What time is measured by the clock? (real, user, system, or, hopefully not, wall-clock?)
- What is the granularity requirement of time? (s, ms, µs, ns)
- Is the clock monotonic, or will it change with changes in the system time (via NTP, time zone, daylight savings time, by the user, etc.)?
- Does the program interference the time?
- Does your observation of the time interference the program?
Trace your System
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
Configure and Trace your Network

- **ifconfig**
  - Configure, manage and query network interface parameters through command line

- **netstat**
  - List network(socket) connections to the system

- **tcpdump**
  - Packet analyzer to trace packets transmitted over the network to which the computer is attached
    - Example:
      - `tcpdump -i eth0 dst 172.168.0.118 -w 001.pcap`

- **wireshark**
  - GUI based alternative to tcpdump
Real-Time Scheduler in Linux
Scheduling in Classes

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 ms, and each invocation has a deadline of 5 ms

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

**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

---

**Task 1**
- Deadline: 5
- Exec time: 4

**Task 2**
- Deadline: 12
- Exec time: 3

**Task 3**
- Deadline: 8
- Exec time: 2

---

**Miss Deadline When using SCHED_FIFO**

---

Scheduler Setup – Basic

- Two classes, would always schedule RT class first
  - **RT class**: static priority, 1 (lowest) to 99 (highest)
    - Preemptive scheduling
    - **SCHED_DEADLINE**: Exists in Kernel, however, no libc wrapper:
      - You need write a Syscall Wrapper by yourself
      - Or: [https://github.com/jlelli/schedtool-dl](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_s gettexter  [http://linux.die.net/man/2/sched_setscheduler](http://linux.die.net/man/2/sched_setscheduler)

  ```
  #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)
  
  ![Command Example](http://linux.die.net/man/2/sched_setscheduler)

  ```
  sudo taskset -c 2,3 4800 # pid 4800 runs on cores 2-3
  ```

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

  ```
  #include <sched.h>

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

- Linux schedulers

- Set priority
  - `sched_setscheduler`: [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)