Real-Time System:

Time, Schedule and Measurement

02/17/2022
Ruixuan Dai
Time in Real Time system

How to measure time in the context of real time?

- Just use a stopwatch?
- Precision?
- Overhead?

Two situations:

- A single host: elapsed time
- Multiple hosts: end-to-end time
Measure time in a single host

- Elapsed Time:
  - Same Clock Source
  - E.g.
    - Program running time
    - Round Trip Time (e.g., ping)

\[
\text{Elapsed\_time} = \text{end\_time} - \text{start\_time}
\]
How to measure elapse time of a program

- Use **Linux built-in command** (A coarse-grained estimation)
  - `time`

```bash
ubuntu@ip-172-31-11-243:~/demo_code$ time ./print_loop
1 2 3 4 5 6 7 8 9 10
real    0m6.008s
user    0m5.964s
sys     0m0.000s
```

- “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**, e.g., `print to stdout`)

Get an overview of your program’s response / execution time.
Don’t Relied on Wall-Time Clock

- Shell built-in command “time”

```bash
ubuntu@ip-172-31-11-243:~/demo_code$ time ./print_loop
real 525592m14.883s
user 0m5.923s
sys  0m0.000s
```

**When Measuring Elapse Time, don’t rely on WTC.**
Measure Elapse Time programmatically

- **C function: gettimeofday()**
  - `return struct timeval`,
  - **Wall clock time** may change
    - User/other program (NTP) changes clock

```
ubuntu@ip-172-31-11-243:~/demo_code$ gcc -o print_timeofday.c -o print_timeofday
```
```
1 2 3 4 5 6 7 8 9 10
Elapsed seconds : 5.854653
```
```
ubuntu@ip-172-31-11-243:~/demo_code$ ./print_timeofday
1 2 3 4 5 6 7 8 9 10
Elapsed seconds : -15483546.304753
```
```
ubuntu@ip-172-31-11-243:~/demo_code$
```
```
ubuntu@ip-172-31-11-243:~$ sudo timedatectl set-ntp 0
```
```
ubuntu@ip-172-31-11-243:~$ timedatectl status
    Local time: Wed 2022-02-09 04:58:46 UTC
    Universal time: Wed 2022-02-09 04:58:46 UTC
    RTC time: Wed 2022-02-09 04:58:46
    Time zone: Etc/UTC (UTC, +0000)
System clock synchronized: yes
    NTP service: inactive
    RTC in local TZ: no
```
```
ubuntu@ip-172-31-11-243:~$ sudo timedatectl set-time 2021-08-14
```
Use POSIX `clock_gettime()`

- **Sources:**
  - **CLOCK_REALTIME:** Wall Time, affected by discontinuous jump. (like `gettimeofday()`)
  - **CLOCK_MONOTONIC:** Not affected by discontinuous jump, but affected by incremental adjust to **compensate the drift** (e.g. NTP).
    - Clock cannot jump, but may skew due to frequency shift from NTP adjustment
  - **CLOCK_MONOTONIC_RAW:** Not affected by NTP, but **drift is not handled**
    - More accurate for very short intervals

- **Precision:** `Clock_getres()`

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

- **RDTSC** *(A CPU instruction to return the number of CPU cycles since its reset)*
  - 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

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

What if on a multi-CPU (multi-core) system?
Inter-domain TSC Synchronization (x86 x64)

- Issues: Non-Synchronized Clocks / Offsets
  - Clock Tree
  - Workaround:
    - Enable **Invariant TSC** (INTEL, since **Haswell**)
    - `rdtsc()`\[^1\]

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

RTC (Real-Time Clock)
- Available on most computers
- 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 and x64, 64-bit
- Cycle Counter (CCNT) on ARM, 32-bit / 64-bit
  - 64-cycle divider, not accessible in user mode
- Potentially very high accuracy

Source: CSE 422S [https://www.cse.wustl.edu/~cdgill/courses/cse422_fl20/slides/07_timers_timing.pptx](https://www.cse.wustl.edu/~cdgill/courses/cse422_fl20/slides/07_timers_timing.pptx)
Elapsed Time in multiple hosts?

- **End-to-End Latency**

  
  \[
  \text{Elapsed time} = \text{end}_{\text{host2}} - \text{start}_{\text{host1}}
  \]

- **Contributor:**
  - propagation delay: signal travel time
  - queueing delays
  - node processing delays
  - routing changes

The time is not always synced between the hosts!
Get Clock Synchronized?

- Query NTP Server, get clock synchronized
  - NTP (Network Time Protocol)
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 is not so precise

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

[^1]: Set up an NTP server: https://ubuntuforums.org/showthread.php?t=862620
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.
# PTP vs. NTP

<table>
<thead>
<tr>
<th>Feature</th>
<th>NTP</th>
<th>PTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay correction</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Transmit timestamp correction</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Client-side source selection</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multiple sources</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Estimation of maximum error</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Authentication</td>
<td>Yes</td>
<td>Experimental</td>
</tr>
</tbody>
</table>
Resources

- **Linux PTP**
  - `SO_TIMESTAMP`
  - 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 Scheduling Classes

Multiple schedulers are implemented as different scheduling classes.

Normal:

- **SCHED_NORMAL/SCHED_OTHER**: regular, Completely Fair Scheduler (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)
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 (cannot be assigned in code):
      - You need write a Syscall Wrapper by yourself
      - Or: [https://github.com/jlleli/schedtool-dl](https://github.com/jlleli/schedtool-dl)

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

- chrt command (can also check task priorities)  

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

- sched_scheduler  
  `[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);
    }
  }
  ...}
  ```
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 release the resource 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

Source: [https://www.cse.wustl.edu/~cdgill/courses/cse422_f19/slides/09_real_time_sched.pptx](https://www.cse.wustl.edu/~cdgill/courses/cse422_f19/slides/09_real_time_sched.pptx)
Scheduler Setup – Affinities

- Taskset command (can also check task affinities) 
  - sudo taskset -c 2,3 4800  # pid 4800 runs on cores 2-3

- sched_setaffinity

```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**
  - A **jiffy** is a kernel unit of time declared in `<linux/jiffies.h>`
  - **HZ** is the number of times jiffies is incremented in one second

- 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`  
  - [http://elinux.org/High_Resolution_Timers](http://elinux.org/High_Resolution_Timers)

- Can recompile kernel to change these values
Trace Your System 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
Resource

- **CSE 422S Lecture: Timers and Timing / Scheduling**
  - [https://www.cse.wustl.edu/~cdgill/courses/cse422_fl20/slides/07_timers_timing.pptx](https://www.cse.wustl.edu/~cdgill/courses/cse422_fl20/slides/07_timers_timing.pptx)
  - [https://www.cse.wustl.edu/~cdgill/courses/cse422_fl19/slides/09_real_time_sched.pptx](https://www.cse.wustl.edu/~cdgill/courses/cse422_fl19/slides/09_real_time_sched.pptx)

- **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/)
Thank You!