ECE 1175
Embedded Systems Design

Lab 5 – CPU Frequency Control over Raspberry Pi

ECE 1175 – Lab 5 (the last one)

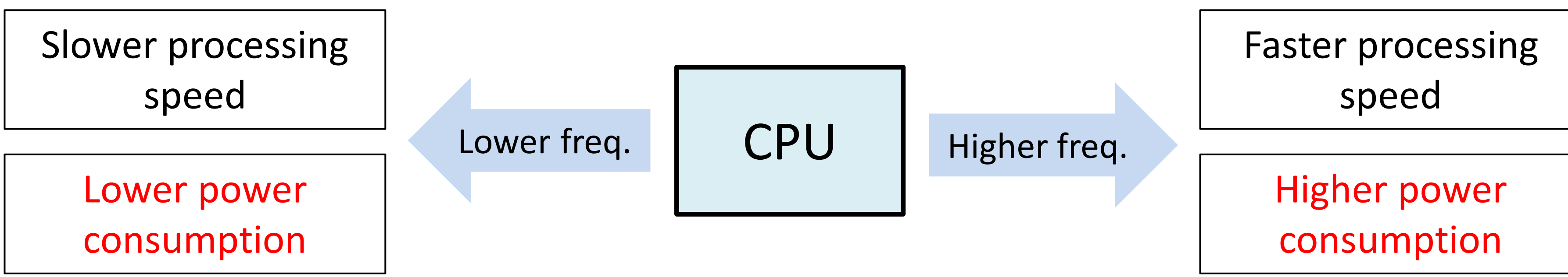
- **Prerequisites**
  - CPU frequency vs. power consumption
  - CPU frequency governor
- **Manipulating CPU Freq. on Raspberry Pi**
  - Access CPU frequency info
  - Change CPU frequency
  - <u>Lab task 1: warm-up of CPU freq. manipulation</u>
- **Customize CPU Freq. Governor**
  - “schedutil” governor
  - <u>Lab task 2: implement “schedutil” from userspace</u>

Prerequisites

- **CPU Frequency vs. Power Consumption**

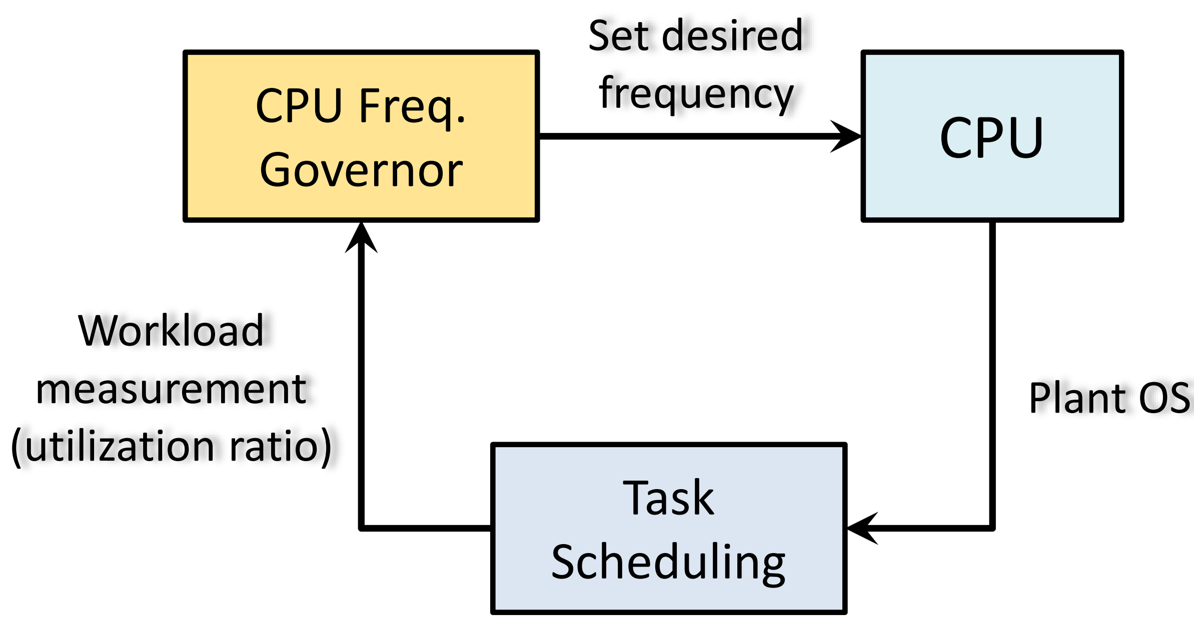
Dynamic CPU power consumption is proportional to the CPU frequency.

$$P_{dyn} = CV^2 f$$



Prerequisites

- **CPU Frequency Governor**
  - Control CPU frequency based on different workloads
  - Basic idea: heavy load → high freq. Light load → low freq.



Access CPU Frequency Info

- **Sysfs**
  - Linux kernel provides an interface via sysfs pseudo filesystem.
  - We can get access under: /sys/devices/system/cpu/cpu[no.]/cpufreq/

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0 $ cd cpufreq
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ ls
affected_cpus               related_cpus                    scaling_governor
cpuinfo_cur_freq            scaling_available_frequencies   scaling_max_freq
cpuinfo_max_freq            scaling_available_governors     scaling_min_freq
cpuinfo_min_freq            scaling_cur_freq                scaling_setspeed
cpuinfo_transition_latency  scaling_driver                  stats
```

Access CPU Frequency Info

- **Useful CPU Frequency Info**
  - cpuinfo_max_freq & cpuinfo_min_freq
  - cpuinfo_cur_freq

Note: the frequency unit is kHz by default.

Access CPU Frequency Info

- **Useful CPU Frequency Info**
  - scaling_available_governors
  - scaling_available_frequencies

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_gove
rnors
conservative ondemand userspace powersave performance schedutil
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_freq
uencies
600000 750000 1000000 1500000
```

Note: check the scaling available frequencies on Raspberry Pi.

Change CPU Frequency

- **We can manually change CPU freq. to any available values. To do so,**
  - Switch the governor to the userspace mode
  - Write your freq. value to scaling_setspeed file

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ sudo su
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo userspace > scali
ng_governor  1. Switch to userspace.
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# cat scaling_governor
userspace
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# cat cpuinfo_cur_freq
600000
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo 1000000 > scaling
_setspeed  2. Write any of available freq. values.
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# cat cpuinfo_cur_freq
1000000
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0#
```

Note: you must run these commands as root.

Lab Task 1

- **A Warm-up of CPU Freq. Manipulation**
  - Get familiar with how to access CPU freq. info
  - Measure runtime of the test program under different CPU frequencies.

```
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo userspace > scali
ng_governor
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo 1500000 > scaling
_setspeed
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo 600000 > scaling_
setspeed
```

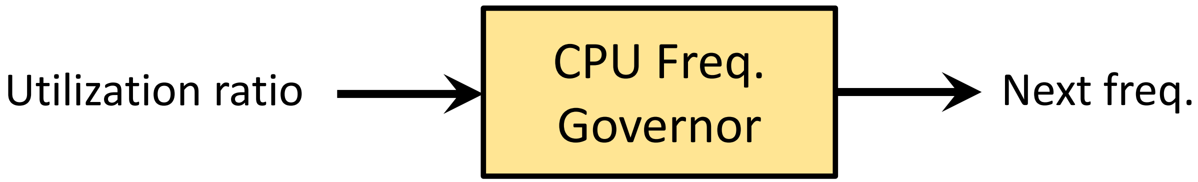
```
pi@raspberrypi:~ $ sudo taskset -c 0 ./lab5_1
Execution time: 2.670180s
pi@raspberrypi:~ $ sudo taskset -c 0 ./lab5_1
Execution time: 6.813416s
```

Note: Run your program on single core.

Schedutil Governor

- **Schedutil**
  - Already implemented in the kernel
  - For details: https://lkml.org/lkml/2016/3/17/420

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_governors
conservative ondemand userspace powersave performance schedutil
```



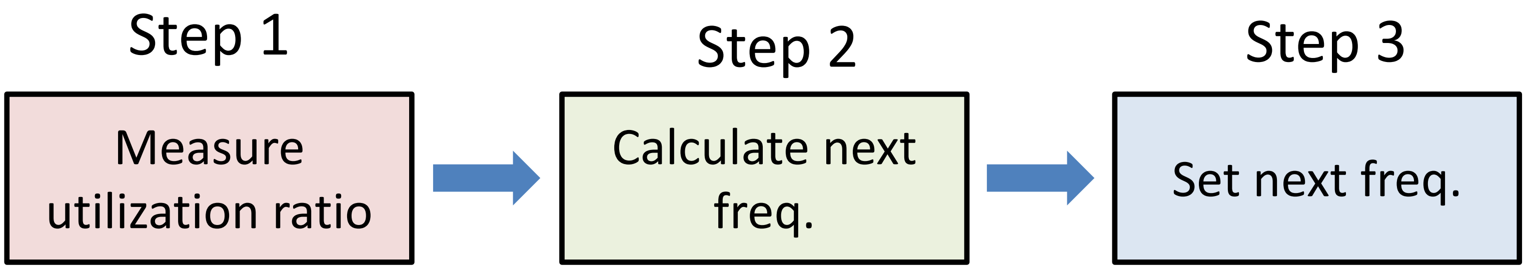
If utilization is frequency-invariant, schedutil says

$$\boxed{\text{desired\_freq} = 1.25 * \text{max\_freq.} * \text{util\_ratio}}$$

Next freq. is the available frequency closest to desired_freq

Lab Task 2

- **Write your own program to Implement schedutil from userspace**



- **We are very clear about how to do step 2 and 3.**
  - Step 2: just apply a formula
  - Step 3: write the value to scaling_setspeed file

But how to get utilization ratio in real-time?

Lab Task 2

- **Get per-CPU utilization ratio in real-time**
  - Read /proc/stat file

pi@raspberrypi: /proc

File Edit Tabs Help

```
pi@raspberrypi:/proc $ cat stat
cpu  51869 0 6607 4803123 4126 0 228 0 0 0
cpu0 28376 0 2787 1157054 1996 0 193 0 0 0
cpu1 8942 0 1250 1214468 635 0 22 0 0 0
cpu2 6293 0 1412 1216430 717 0 8 0 0 0
cpu3 8258 0 1158 1215171 776 0 5 0 0 0
intr 28774324 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 304386 0 0 0 0 0 6793 14 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 931 0 0 0 0 16619 0 0 16244 0 0 0 0 0 0 0 276
56723 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4545 5124 0 0 0 0 289289 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 1743013
btime 1604165995
processes 3923
procs_running 1
procs_blocked 0
softirq 842530 52919 213442 9 18098 0 0 270344 181986 0 105732
```

Lab Task 2

- **Get per-CPU utilization ratio in real-time**
  - Interpret entries in /proc/stat
  - For details: [Link]

For each cpu[no.],

Total idle time since boot: sum 4th & 5th columns

```
cpu  51869 0 6607 4803123 4126 0 228 0 0 0
cpu0 28376 0 2787 1157054 1996 0 193 0 0 0
cpu1 8942 0 1250 1214468 635 0 22 0 0 0
cpu2 6293 0 1412 1216430 717 0 8 0 0 0
cpu3 8258 0 1158 1215171 776 0 5 0 0 0
```

Total time since boot: sum 1st to 8th columns

```
The very first  "cpu" line aggregates the  numbers in all  of the other "cpuN"
lines.  These numbers identify the amount of time the CPU has spent performing
different kinds of work.  Time units are in USER_HZ (typically hundredths of a
second).  The meanings of the columns are as follows, from left to right:

- user: normal processes executing in user mode
- nice: niced processes executing in user mode
- system: processes executing in kernel mode
- idle: twiddling thumbs
- iowait: In a word, iowait stands for waiting for I/O to complete. But there
  are several problems:
  1. Cpu will not wait for I/O to complete, iowait is the time that a task is
     waiting for I/O to complete. When cpu goes into idle state for
     outstanding task io, another task will be scheduled on this CPU.
  2. In a multi-core CPU, the task waiting for I/O to complete is not running
     on any CPU, so the iowait of each CPU is difficult to calculate.
  3. The value of iowait field in /proc/stat will decrease in certain
     conditions.
  So, the iowait is not reliable by reading from /proc/stat.
- irq: servicing interrupts
- softirq: servicing softirqs
- steal: involuntary wait
- guest: running a normal guest
- guest_nice: running a niced guest
```

Lab Task 2

- **Get per-CPU utilization ratio in real-time**
  - Calculate utilization ratio for each cpu core

// Get accumulative t_total & t_usage

$$t_{total} = user + nice + system + idle + iowait + irq + softirq + steal$$
$$t_{idle} = idle + iowait$$
$$t_{usage} = t_{total} - t_{idle}$$

// Get real-time util ratio (from moment τ_1 to τ_2)

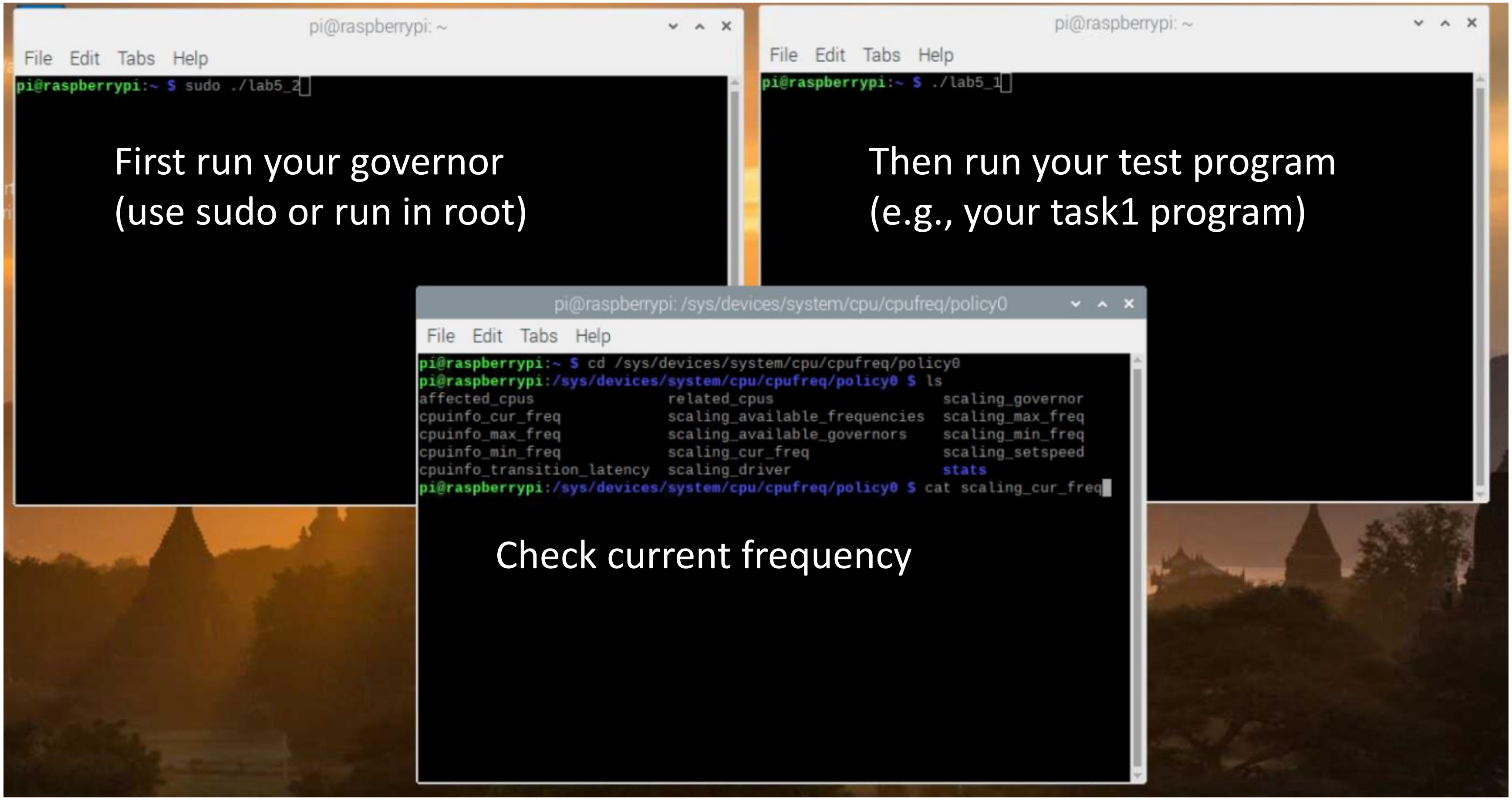
$$\Delta t_{total} = t_{total}(\tau_2) - t_{total}(\tau_1)$$
$$\Delta t_{usage} = t_{usage}(\tau_2) - t_{usage}(\tau_1)$$
$$\%Util = \frac{\Delta t_{usage}}{\Delta t_{total}} \times 100\%$$

One more step: find the highest util ratio across all cores as the final util ratio fed to the formula.

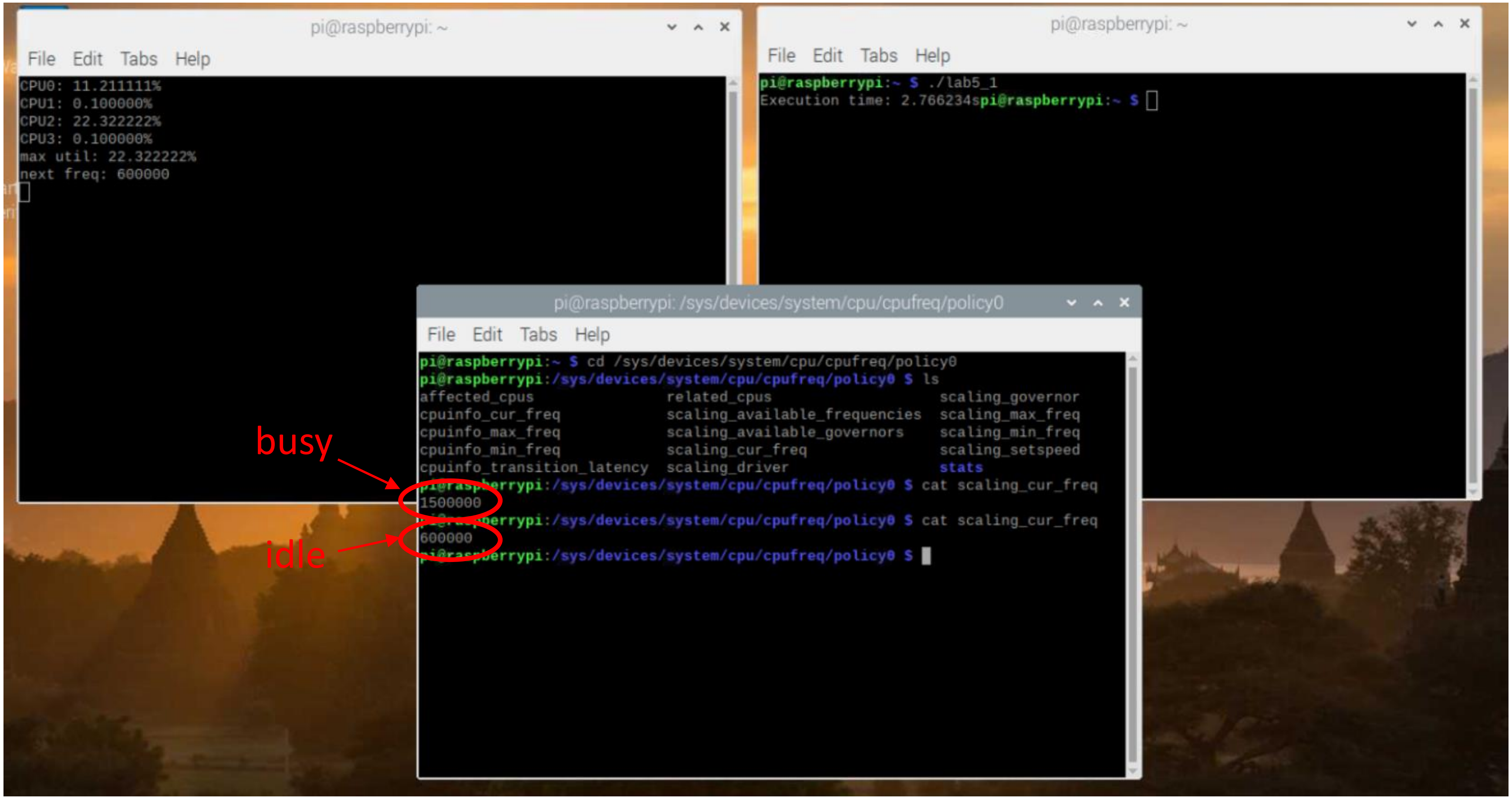
Lab Task 2

- **Use programming language of your choice**
  - C/C++: faster, most familiar
  - Python: easier to code
  - Others …
- **To demonstrate your implementation**
  - Switch to userspace mode
  - Keep running your governor program in the background
  - Run test programs with different workloads
  - TA will check how cpu frequency changes
  - Submit your code to Canvas

Lab Task 2


pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ sudo ./lab5_2
First run your governor
(use sudo or run in root)
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ ./lab5_1
Then run your test program
(e.g., your task1 program)
pi@raspberrypi: /sys/devices/system/cpu/cpufreq/policy0
File Edit Tabs Help
pi@raspberrypi:~ $ cd /sys/devices/system/cpu/cpufreq/policy0
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ ls
affected_cpus related_cpus scaling_governor
cpuinfo_cur_freq scaling_available_frequencies scaling_max_freq
cpuinfo_max_freq scaling_available_governors scaling_min_freq
cpuinfo_min_freq scaling_cur_freq scaling_setspeed
cpuinfo_transition_latency scaling_driver stats
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ cat scaling_cur_freq
Check current frequency

Lab Task 2


pi@raspberrypi: ~
File Edit Tabs Help
CPU0: 11.211111%
CPU1: 0.100000%
CPU2: 22.322222%
CPU3: 0.100000%
max util: 22.322222%
next freq: 600000
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ ./lab5_1
Execution time: 2.766234spi@raspberrypi:~ $
pi@raspberrypi: /sys/devices/system/cpu/cpufreq/policy0
File Edit Tabs Help
pi@raspberrypi:~ $ cd /sys/devices/system/cpu/cpufreq/policy0
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ ls
affected_cpus related_cpus scaling_governor
cpuinfo_cur_freq scaling_available_frequencies scaling_max_freq
cpuinfo_max_freq scaling_available_governors scaling_min_freq
cpuinfo_min_freq scaling_cur_freq scaling_setspeed
cpuinfo_transition_latency scaling_driver stats
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ cat scaling_cur_freq
1500000
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ cat scaling_cur_freq
600000
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $
busy
idle

Thanks for your participation in ECE II75 lab!