# Lab1

Name
Nico Ferrari (nife1600)
Erick 'Xiao' Guan (xigu1500)

September 2017

## 1 Setting up the system

We installed the RASPBIAN STRETCH LITE image released on 2017-0907. Then we upgraded the system to the latest as of execution. We used Raspberry Pi 2 Model B (called *Pi* later) for the lab. The hardware utilizes the Broadcom BCM2836 package with 900 MHz 32bit quadcore ARM CortexA7 CPU as well as 1 GB RAM.

We connected the Raspberry Pi to a router along with a Mac. Then we use ssh and mosh to login into the machine.

To make sure everything runs at it should, we installed the software as listed:

1. Linux: 4.9.41
2. git: 2.11
3. SSH: OpenSSH
4. mosh: 1.2.6
5. Wiring Pi at 96344ff7125182989f98d3be8d111952a8f74e15

## 2 Running the simple program with Wiring Pi

We ran the command `gpio -v`. It returns:

```
gpio version: 2.44
Copyright (c) 2012-2017 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Pi 2, Revision: 01, Memory: 1024MB, Maker: Sony
  * Device tree is enabled.
  *--> Raspberry Pi 2 Model B Rev 1.1
  * This Raspberry Pi supports user-level GPIO access.
```

We ran the command `gpio readall`. It returns:

```
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
```

Then we ran the blink example to verify the environment [1]. It worked as Figure 1. The source is in the `0_blink.c`.

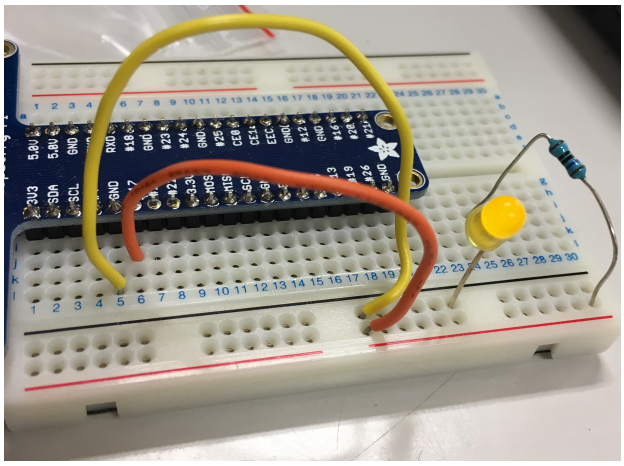

**Figure 1:** Blinking

# 3 C programming toggling BCM 17 (pin11, GPIO0) every 20ms

The file is listed in the `1_toggle_gpio.c`.


**Figure 2:** Interference

We brought the setup from the previous sample. Starting from the long cables and $160\Omega$. It has a lot of interference per Figure 2.

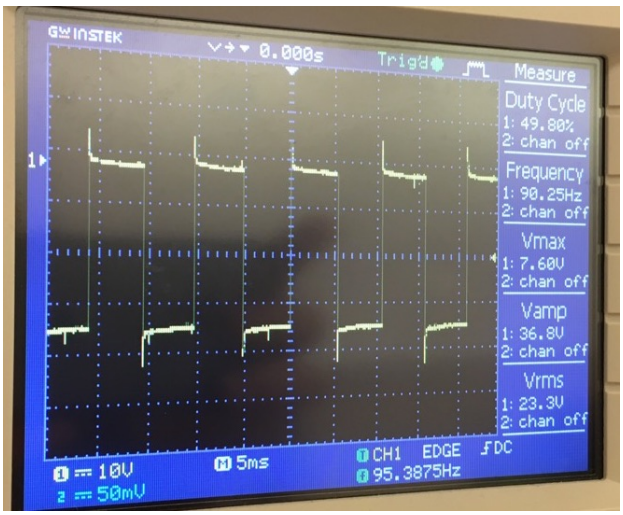


**Figure 3:** Realtime

So we changed the resistor to $220\Omega$ and shorter cables but the problem remains. Even after we changed the pin to input mode. Short after, we used the channel 2 on our oscilloscope, it's better with less interference. In the end, we manage to get a smooth scope with the channel 1 as Figure 3 shows.

# 4 Improving performance

We followed the instruction of lab and did our experiments along the way.

## 4.1 Disabiling swap

We tried to push our program into limit. We tried to output at 1ms interval which is minimal and it was running smoothly than we expected. Anyhow, the intermediate action we took was disabling the paging and swapping. This is done by `swapoff /var/<swapfile>`. There isn't much difference in our observation. This led to the first conclusion that our program is simply too trivial for CPU to handle. Linux doesn't do paging or swapping to such simple program at all.

## 4.2 Underclocking

We started to think differently. We tried to underclock the device since it should decrease the performance of CPU. We pined the CPU frequency at 500 MHz, SDRAM stopped at 250 MHz and the core frequency at 250 MHz. We ran our program again. It didn't show visible difference either.

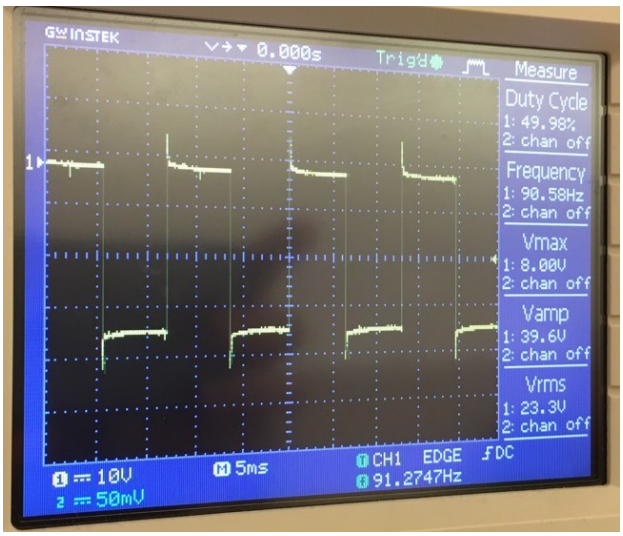


**Figure 4:** Without realtime scheduling

Figure 4 shows the oscilloscope graph from running program without realtime scheduling.

## 4.3 cgroups

So we continued to look into ways to torture our CPU. We thought cgroups in Linux could be a viable choice since it's designed to split the cpu as shares and allot them to processes. We followed [2][3] to set up two groups. They have 1024 and 16 shares of CPU. And then we needed to

find another program that can use CPU. We expected a neighbor who can give us some disruption. We introduced [4]a mathematic program that calculates prime and gives it the larger share of CPU. And we gave our program only 16 shares of CPU. We saw a huge jitter in the oscilloscope. We tried to confirm the behavior by not running programs with cgroups. We didn't see any difference. Here we also switched back to 5ms interval since the interval doesn't matter that much but a bad neighbor does. We concluded that cgroups doesn't affects realtime scheduling much but a loud neighbor does.

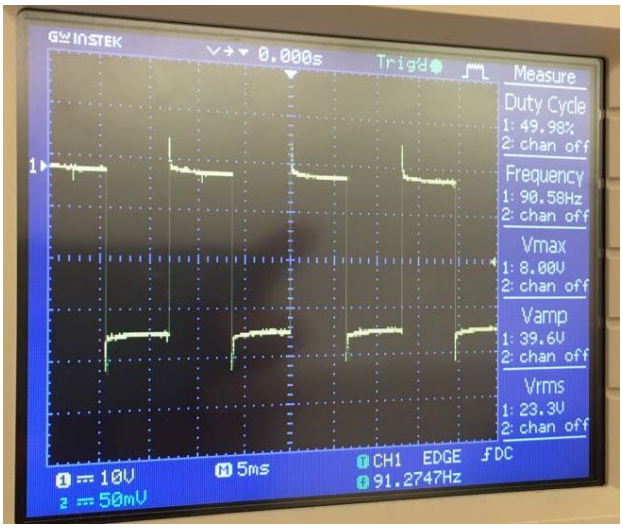


**Figure 5:** Bad neighbor gives a lot of jitters

As Figure 5 shows, the bad neighbor gives us a lot of jitters. The pulse should be delivered at 5ms interval but it isn't.

## 4.4 Monotonic clock ticks

We started to measure the interval by logging the result by a monotonic clock tick. Nothing special here. syslog are used.

## 4.5 Realtime scheduling

We ran our program by using `chrt --rr 99` and the result was astonishing. The spike went away. Every pulses are almost at its 5ms interval with maybe 10 nanoseconds deviation. In the meantime, the math program was still computing at its maximum speed in a limit of Linux's processes scheduler allows.

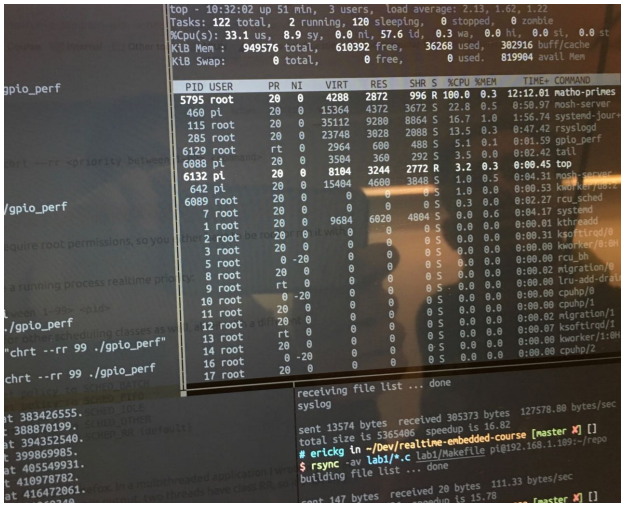


**Figure 6:** Primer program running in the backgroud

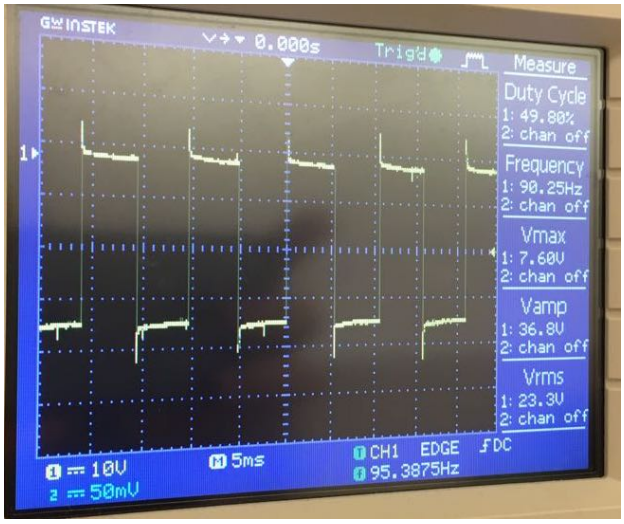


**Figure 7:** Realtime scheduling works

Figure 6 shows that the primer program take a lot of CPU time. In the meantime, as the Figure 7 shows, the realtime scheduler give us almost ideal output but it's not perfect.

We concluded that realtime scheduling algorithm made all the difference here.

# 5 Conclusion

We have three conclusions:

1. Conclusion 4.1 is that a small program don't take too much resource which means it can run on general purpose operating system usually with no jitters.

2. Conclusion 4.3 is that cgroups is not useful to help realtime program.

3. Conclusion 4.5 is that realtime scheduler takes care of timeliness of program.

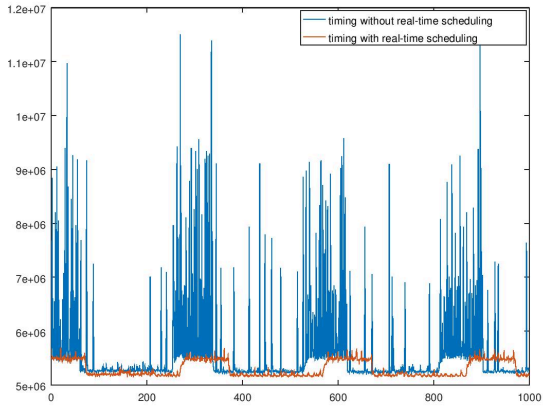


**Figure 8:** Realtime scheduling vs non-realtime scheduling

We also draw a graph as Figure 8 to show the interval difference over their behaviors. The x-axis represent event id while the y-axis represents the time interval with previous event. We can observe that realtime schedulding does help the program running something at the given interval as our conclusion says.

# 6 Reference


[1] *Blink example on wiring pi*, `http://web.archive.org/web/20170929074233/http://wiringpi.com/examples/blink/`, visited 2017-09-29.

[2] *Arch linux: Cgroups*, `http://web.archive.org/web/20170929074336/https://wiki.archlinux.org/index.php/cgroups`, visited 2017-09-29.

[3] *Restricting process cpu usage using nice, cpulimit, and cgroups*, `http://web.archive.org/web/20170929074410/http://blog.scoutapp.com/articles/2014/11/04/restricting-process-cpu-usage-using-nice-cpulimit-and-cgroups`, visited 2017-09-29.

[4] *Mfillpot/mathomatic on github*, `https://github.com/mfillpot/mathomatic/commit/6eb4cc1674c2aa30a514e850aa0663b7bbc060de`, visited 2017-09-29.