CS2106: Lab 5

This lab is an experimental lab with programming which shows measuring process time and generating a specific time delay with the CPU. It is useful for generating small time delays which may not be feasible with interrupts or hardware timers. This question will also give you some insights into real-time code which may need to execute within X microseconds, e.g. flight control system.[1]

Important Note:
The results of this lab can depend on your actual machine and kernel, e.g. different processors and kernel versions can have different results. This means that the results can be dependent on the machine including virtual machine (if one is used) and version of Linux. As such, you should clearly indicate what machine, version of Linux, and any virtual machine information. To get the machine information and version, Have a look at the special files `/proc/cpuinfo` and `/proc/version`.[2] For a virtual machine, the cpuinfo may not exactly match your hardware. While you can test timing on a virtual machine, it is best to also test this on a real machine, i.e. the lab PCs, since timing on a virtual machine also virtualises the time which may lead to inaccuracies.

1 Exercise 1: Timing a CPU bound process (Not graded)

Goals: Understanding CPU-bound processes and timing on Unix

Consider the program `lab5ex1.c` which takes an argument which is a loop count. It runs the loop in `delay()` (in `lab5delay.c`) for the specified number of times. The `delay()` function is an example of a function which only does computation (although here it doesn't compute any useful answer).[3] The objective of `delay()` is to have a pure CPU-bound process which does computation. To be more precise, the `delay()` function only executes execute machine instructions without any system calls, so it is a CPU-bound loop. It is unlikely to have any page faults or exceptions.[4] Given the way the code is written, once the CPU is executing `delay()` then the code should continue running until the Linux scheduler causes another process to run. Note that the rest of `lab5delay.c` will make some system calls.

You can compile the code as follows:

```
$ gcc -O2 -c lab5delay.c # generates object file: lab5delay.o
$ gcc -o lab5ex1.c lab5delay.o -lm
```

We have used `-O2` to `gcc`, optimization level 2, for `gcc` to try to keep the resulting compiled instructions in registers. This example shows separate compilation, the `-c` which compiles the C code to object code

[1] The flight control system is being investigated in the Boeing 737 Max crashes. May not be a real-time/scheduling issue though but it does highlight the seriousness of bugs and timing issues in the software running critical system. It is difficult to ensure that code works exactly as intended without any bugs. Lab 5 is about the accuracy of the timing, so inaccurate timing may be considered to not meet the requirement though it may not be a bug in the input-output sense.

[2] At the same, take a look at the other files in `/proc`. `/proc` is a special "virtual" filesystem containing information from the kernel).

[3] There is no need to try to understand what is the meaning of the code, just the effect on CPU time. It is partly written this way to avoid `gcc` removing the loop since the optimizer can in some cases remove loops completely.

[4] There may be a possibility of arithmetic exception.

containing the machine instructions in a `.o` file. The `sqrt()` function comes from the math library (simply called `m`) so it is "linked" in with `-lm`.

The code also prints the current time of the day (real time) in seconds and microseconds (usec). You can compare the time with the real time returned by the `time` command. The `time` command in the `bash` shell on Linux runs a command[5] and measures the (**approximate**) real and user time taken, as well as, the amount of system time (the kernel time) used. There are three kinds of time here. Real time refers to wall-clock time. User time, on the other hand, refers to logical time used by the process, i.e. how much time was used in running the process. Essentially, how much time spent the process spends in running state. Real and user time are different because of concurrency. For example, on a single CPU with many processes sharing it, the real time would be expected to exceed the user time. System (sys) time refers to how much time is spent in the OS for this process. Note that the time measurement is often only approximate and is dependent on what hardware mechanisms are used by the kernel to measure time. See also `man 7 time`.

To understand what `lab5ex1` does, try timing `lab5ex1` with various loop arguments, e.g.

```
$ time ./lab5ex1 500
```

You can see that by adjusting the value of `D` (the loop iterations), its possible to get different timings. Try values of `D` such as 200, 500, 1000, etc.[6] Take note of all *three* times from the `time` program. Is the real time different from the user time? You may observe that the timings are not exactly the same and may have some variation — repeating the experiment can change the reported timing. Often, you will find that larger times tend to be more consistent than smaller times. Small times are also hard to measure.

2 Exercise 2: Calibrating computation and time (Not Graded)

Goals: An attempt to determine what value of `D` to give for `delay(D)` which will run for one second of CPU time

Please do Exercise 1 first so that you can understanding timing issues before starting on Exercise 2. Take note that your final results and timings should be done with the PC lab machines — this is to standardize grading as the results may be too variable otherwise.[7]

The following program, `lab5ex2.c` attempts to measure the value of D for `delay(`D`)` (from Exercise 1) which corresponds to approximately one second of CPU time. This means that the `User` time from the `time` command is expected to be be around 1 second.

`lab5ex2` uses `clock()` which returns a unit which corresponds to the CPU time usage of a process. The unit of measurement returned by `clock()` is not the same as time, rather there are `CLOCKS_PER_SEC` clock ticks. The ticks need not increase continuously and may only changes at certain times (may not be completely periodic). The way to think about `clock()` is that measures something like timer ticks. However the way the clock unit is adjusted depends on the Linux implementation.[8]

[5] A little like `lab4ex3` but performs a different function.

[6] Note the caveat that the effect of `D` is machine/OS dependent.

[7] It is possible to do some testing on a virtual machine or another machine but the final testing should be on the actual PC lab machines.

[8] While a timer interrupt is of the simplest ways of doing so, there are other ways, such as "tickless" kernel. We do not

Run `lab5ex2` (you can compile using similar procedure to Exercise 1) and you should run several times to understand how the estimated D value varies. Use Exercise 1 to check on the accuracy of D from `lab5ex2`. Does the D correspond to 1 second of user time as measured in Exercise 1?

To understand why `lab5ex2` is less accurate than desired. Imagine that timer interrupts.[9] Everything which is running on the CPU takes time (this should be implicit from CS2100/Computer Organization since every instruction takes some clock cycles). This includes the machine instructions from the running process, exception and interrupt mechanisms which occur, OS kernel code which is executed (which may be from a page fault and not system call), etc. So while a certain number of machine instructions will be executed between two timer interrupts, the question is how many of those are the instructions we are controlling in `lab5ex2` and how to get an accurate measurement without needing to understand the details of the kernel.

3 Exercise 3: More accurate calibration of CPU time (Graded)

Goals: To write a cpu-bound program which can accurately use one second of CPU time

You will have seen that while `lab5ex2` does give a CPU-bound computation which is approximately equivalent to 1 second of CPU time. This approximation is not so good. The goal in Exercise 3 is to write an improved program, `lab5ex3.c`, which outputs a more accurate value of D which is applied to `lab5ex1` to measure the CPU time taken by `delay(`D`)`. The objective is for the timing result of D to be within 1 ± 0.1 second of user time or *better*.

Grading will be based on the *accuracy* obtained – so a more accurate solution can get more marks. One use of `lab5ex3` is get a very short time delay, this delay may be shorter than what the hardware timers can achieve (this depends, on the particular hardware).

Important Notes

Obviously there can be interaction with the Linux scheduler. You should ensure the machine is not busy while doing the timing test. Timing on a loaded machine may be less accurate than on an unloaded machine.

This lab may also be unfamiliar since you may not have associated computation with precise CPU time. Here, the whole objective is to determine this association between CPU bound computation with a unit of time like 1 second. Obviously if one can do 1 second, then any desired time is also possible. Furthermore, it is feasible to go under the accuracy of a clock unit, e.g. 0.1 clock unit.

The Linux kernel does something similar. A discussion on BogoMIPS can be found on:
`https://en.wikipedia.org/wiki/BogoMips`
The BogoMIP value is similar to the delay value computed in Exercise 1 and 2 though the actual value is different since the details are different.

may any assumptions of the kernel implementation. You should read the `clock` man page with this perspective in mind.

[9] However, the actual Linux implementation may be more complex) are used to increment the `clock` unit.

The value of BogoMips can also be thought of as a measure of the CPU speed, the faster the CPU, the more loops can be run within 1s, hence the name **Bogus MIPS** rather than Machine Instructions per Sec. Note that the `clock()` function itself takes CPU time, so one should be careful in running it too long. Please write your code **only in C** yourself without using any of the Linux BogoMIPS code (the Linux code may contain machine instructions, i.e. assembly language code), and no `gcc` extensions are allowed.

Submission

Submit only `lab5ex3.c` to the appropriate workbin using the usual naming convention, e.g. *Surname*-*StudentID*-`lab5ex3.c`.

In your submission, you should give the CPU and Linux version information in the comment block at the start of `lab5ex3.c`. Give your experimental results which should include timing results from several runs using the `time` program on `lab5ex1` using the values from `lab5ex3`. You should also compare the timings with the values from `lab5ex2`. Discuss the accuracy of your answer focusing on why it is accurate. Your discussion must be supported by relevant experimental results. Please add the example comment block to the start of the file (substituting Name, ID (Student ID), CPUINFO, VERSION and your results):

```
/*
* Name: Dennis Ritchie
* ID: U123Y
* file: lab5ex3.c
*/

/* Lab 5 results
   DISCUSS YOUR RESULTS HERE:
   CPUINFO: from /proc/cpuinfo
   VERSION: from /proc/version

   1. your values of D from lab5ex3 and the timings using lab5ex1
   2. the results using D from lab5ex2
*/
```