CS-350 - Fundamentals of Computing Systems
Homework Assignment #1 - EVAL

Due on September 21, 2023 — Late deadline: September 23, 2023 EoD at 11:59 pm

Prof. Renato Mancuso

Renato Mancuso

EVAL Problem 1

In this EVAL problem, you will evaluate the result of the clock measuring functions developed in the corresponding BUILD assignment. In particular, we will estimate which method (SLEEP vs. BUSYWAIT) works best for stable CPU speed estimates, and whether the resulting measurements match the CPU specifications.

a) First, estimate the CPU clock speed on your own machine using the SLEEP method. To do this, collect the results returned by the `clock` executable you implemented by running it 10 times. Start with a wait time of 1 second, all the way up to (and including) 10 seconds, increasing the wait time by 1 second at each step.

From the 10 CPU clock speed measurements you have obtained, compute and report the max, min, average, and standard deviation.

A	B	C
ARRIVAL_RATE	SLEEP	BUSYWAIT
1	WaitMethod:SLEEP WaitTime:1 0 ClocksElapsed:1004255584 ClockSpeed:1004.26	WaitMethod:BUSYWAIT WaitTime:1 0 ClocksElapsed:1000040667 ClockSpeed:1000.04
2	WaitMethod:SLEEP WaitTime:2 0 ClocksElapsed:2003230417 ClockSpeed:1001.62	WaitMethod:BUSYWAIT WaitTime:2 0 ClocksElapsed:2000007792 ClockSpeed:1000.00
3	WaitMethod:SLEEP WaitTime:3 0 ClocksElapsed:3005198377 ClockSpeed:1001.7	WaitMethod:BUSYWAIT WaitTime:3 0 ClocksElapsed:3000028126 ClockSpeed:1000.01
4	WaitMethod:SLEEP WaitTime:4 0 ClocksElapsed:4002025377 ClockSpeed:1000.51	WaitMethod:SLEEP WaitTime:4 0 ClocksElapsed:4003064460 ClockSpeed:1000.77
5	WaitMethod:SLEEP WaitTime:5 0 ClocksElapsed:5001534169 ClockSpeed:1000.31	WaitMethod:BUSYWAIT WaitTime:5 0 ClocksElapsed:5000043503 ClockSpeed:1000.01
6	WaitMethod:SLEEP WaitTime:6 0 ClocksElapsed:6002245336 ClockSpeed:1000.37	WaitMethod:BUSYWAIT WaitTime:6 0 ClocksElapsed:6000003836 ClockSpeed:1000.00
7	WaitMethod:SLEEP WaitTime:7 0 ClocksElapsed:7001345670 ClockSpeed:1000.19	WaitMethod:BUSYWAIT WaitTime:7 0 ClocksElapsed:7000031545 ClockSpeed:1000.00
8	WaitMethod:SLEEP WaitTime:8 0 ClocksElapsed:8000709462 ClockSpeed:1000.09	WaitMethod:BUSYWAIT WaitTime:8 0 ClocksElapsed:8000035712 ClockSpeed:1000.00
9	WaitMethod:SLEEP WaitTime:9 0 ClocksElapsed:9001339004 ClockSpeed:1000.15	WaitMethod:BUSYWAIT WaitTime:9 0 ClocksElapsed:9000044587 ClockSpeed:1000.00
10	WaitMethod:SLEEP WaitTime:10 0 ClocksElapsed:10000931505 ClockSpeed:1000.09	WaitMethod:BUSYWAIT WaitTime:10 0 ClocksElapsed:10000049296 ClockSpeed:1000.00

Figure 1: 1a, 1b: measures of clock speeds by SLEEP and BUSYWAIT methods

BUSYWAIT's measures of clock speeds:

- **max: 1000.04 MHz**
- **min: 1000.00 MHz**
- **mean: about 1000.01 MHz**
- **standard deviation: about 0.01 MHz**

b) Repeat the same procedure above where instead of the SLEEP method, you use the BUSYWAIT method for all the measurements. Then, compute and report the same statistics on the measured CPU clock speed.

SLEEP's measures of clock speeds:

- **max: 1004.26 MHz**
- **min: 1000.09 MHz**
- **mean: about 1000.35 MHz**
- **standard deviation: about 2.59 MHz**

c) Based on what you have acquired in the previous steps, compare the SLEEP and BUSYWAIT methods to estimate the CPU speed of your processor. Which method yielded more precise and/or stable results? Take also a look at the specifications of your CPU. Which method produced a result that is closer to the specifications provided by the manufacturer of your CPU? Comment on whether the results are in a match with said specifications or not, and why.

- **The SLEEP method's clock speeds have a higher standard deviation, which indicates that it might not be as precise or stable in measuring the CPU clock speed. The BUSYWAIT method, on the other hand, measured clock speeds with consistency and a relatively low standard deviation (i.e., close to 1000.00 MHz for all wait times), suggesting greater precision and stability.**
- **The processor of my machine, Apple's M2 Pro processor (10 core), has a clock speed of 3.49 GHz = 3490 MHz. Although the maximum speed measured by SLEEP method is numerically closer to the specifications provided by M2 Pro processor, and that the BUSYWAIT method is more stable to produce reliable estimate of cpu frequency, that both methods produced results too far away from the official cpu frequency makes it difficult to determine which method is better in terms of approaching the the specifications of M2 Pro CPU. The noticeable discrepancy can be accounted by the following factors:**
  - (a) **Apple's M2 Pro processors consist of multiple types of cores, such as high-performance cores (my machine has 6 of them) and power-efficient cores (my machine has 4 of them). The clock speeds of different cores may be different, and the clock speed reported in official specifications might relate to the high-performance cores, which are not always active. The SLEEP and BUSYWAIT methods may not fully utilize these high-performance cores, leading to low clock speed measurements.**
  - (b) **Apple's M2 Pro processors use ARM-based Reduced Instruction Set Computer (RISC) architecture, which is different from the x86 CISC architecture implemented in many traditional laptops. Clock speed measurements can vary between ISAs, which introduces the great discrepancy between official specs number and my actual measurement.**
  - (c) **3. Due to Apple's co-processors, it has integrated the Neural Engine and GPU. These co-processors offload specific tasks from the CPU, improving overall system performance and efficiency. The clock speed of these co-processors may not be reflected in the SLEEP and BUSYWAIT measurements.**
  - (d) **// sought help from ChatGPT; https://www.macworld.com/article/670873/which-mac-processor-apple-processor-comparison-m1-vs-intel.html; https://www.quora.com/Why-are-ARM-processors-more-efficient-than-x86-Are-there-any-benefits-to-x86; https://www.andro vs-x86-key-differences-explained-568718/; https://www.apple.com/newsroom/2023/06/apple-introduces-m2-ultra/**

EVAL Problem 2

In this EVAL problem, we will start to examine the behavior of the `server` process in response to variable traffic conditions generated by the `client`.
HINT: many of the questions below, will ask you to determine a lapse of time in which the server was active. You can compute that time window in post-processing as the lapse of time between the `<receipt timestamp>` of the very first request and the `<completion timestamp>` of the last request processed by the server.

a) First, launch the client with the following parameters: `client -a 10 -s 12 -n 500 <some port>` where `<some port>` is the same port (of your choice) used for the `server`. Allow the client and server to exchange data and terminate, then post-process the output produced by the server.

From the server output, compute the average throughput of the server in terms of requests per second. Walk us through the steps you performed to accomplish this.

The average throughput of the server in terms of requests per second is given by:

$$\text{Average Throughput} = 9.748335603298035$$

where the time lapsed for all 500 request is derived by subtracting `receipt timestamp` of the first process from `completion timestamp` of the last process.

$$\begin{aligned}\text{Average Throughput} &= \frac{500}{\text{process}_{499}.\text{completion\_time} - \text{process}_0.\text{receipt\_time}} \\ &= \frac{500}{51.29080700000122} \\ &\approx 9.748335603298035\end{aligned}$$

b) Follow the same procedure above and once again post-process the output of the server. This time, compute the utilization of the server as a percentage. Walk us through the steps you performed to accomplish this.

$$\begin{aligned}\text{total\_time\_elapsed} &= \text{process499.completion\_time} - \text{process0.receipt\_time} \\ &= 19785.032448 - 19733.730446 \\ &= 51.30200200000036 \text{ seconds} \\ \text{active\_time} &= \sum_{i=0}^{499}(\text{process}_i.\text{completion\_time} - \text{process}_i.\text{receipt\_time}) \\ &= 40.67126399999324 \\ \text{utilization} &= \frac{\text{active\_time}}{\text{total\_time\_elapsed}} \\ &= \frac{40.67126399999324}{51.30200200000036} \\ &\approx 0.7927875\end{aligned}$$

c) Repeat the experiment above but by running the server and client back-to-back, while at the same time extracting some statistics from the server run from the OS. To do so, move to the build directory and use the following command line:

```
/usr/bin/time -v ./server 2222 & ./client -a 10 -s 12 -n 500 2222
```

NOTE: by doing so, BOTH client and server will run in the same terminal and their output will be interleaved on the console. If you want to suppress the output of the client, redirect the stdout of the client to the null device using:

```
/usr/bin/time -v ./server 2222 & ./client -a 10 -s 12 -n 500 2222 > /dev/null
```

Also note that the server will run in background. You can use the following command to kill it if it does not terminate:

```
killall server
```

After the server has run successfully, the `time` utility will print a host of statistics acquired from the OS. Take a look at the entry "Percent of CPU this job got:". Does it match with what you computed in the previous question?

Percent of CPU this job got: 79%. Exactly what I got in part 2.

d) Now let us repeat the computation of the utilization of the server as in Qb) but this time sweep through the `-a` parameter passed to the server. In particular, run the first experiment for a value of 1; then a second time with a value of 2; and so on until and including the case where the value is 12. Thus, you will run 12 experiments in total.

Produce a plot that depicts the trend of the resulting server utilization (y-axis) as a function of the arrival rate (`-a`) parameter (x-axis). Is there any correlation between the two values?

$$\text{Arrival rates} = [1, 2, \ldots, 12]$$

Utilization = [0.07990854200303942, 0.15978851574975828,
0.23963920026085142, 0.3194657248494175, 0.399276819875939,
0.47906471138070345, 0.5588362268081839, 0.6378679993547415,
0.7158230949952732, 0.7928351565990366, 0.8693743028659328,
0.9454279142970762]

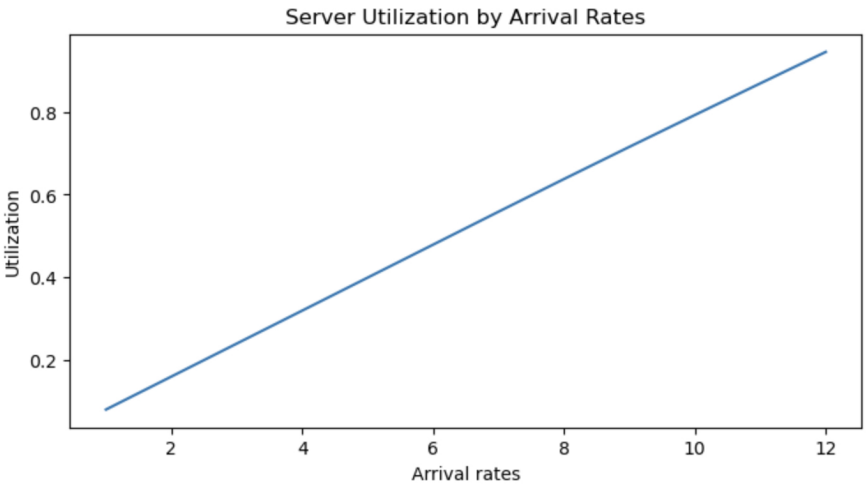

Figure 2: 2d. Server Utilization by Arrival Rates

Correlation between arrival rates and server utilization: Arrival rates are positively correlated with server utilization. As arrival rates increase, utilization increases steadily accordingly.

e) By reusing the same exact output files produced as part of the previous question, let us reason about the response time of the various requests. First, compute the average response time for all the 500 requests as observed in the case when the parameter `-a` was set to 10. Also compute max, min, and std. deviation.

For an arrival rate of 10:

- the max response time is the maximum difference between `completion_time` and `sent_time` = 0.5166409999983443;
- the min response time is the minimum difference between `completion_time` and `sent_time` = 0.0002640000020619482;
- the mean response time is 0.08133030999987387;
- the standard deviation is 0.07785404864502665.

f) Now, repeat the average response time calculation individually for the various runs with the `-a` parameter from 1 to 12. Finally, produce a plot that depicts the trend of the average response time y-axis as a function of the server utilization x-axis. What relationship do you discover between the two quantities?

Zipped tuples of (utilization, `average_response_time`) for each
arrival rate from 1 to 12: [(0.07990854200303942, 0.0869059960000377),
(0.15978851574975828, 0.09569467599999189),
(0.23963920026085142, 0.10479275199996482), (0.3194657248494175, 0.11509702000001562),
(0.399276819875939, 0.12863635999995313), (0.47906471138070345, 0.14517568399994343),
(0.5588362268081839, 0.1684361699999208), (0.6378679993547415, 0.1977681400000147),
(0.7158230949952732, 0.25027485200004596), (0.7928351565990366, 0.3288346560000064),
(0.8693743028659328, 0.4425892380000878), (0.9454279142970762, 0.8276834020000169)]

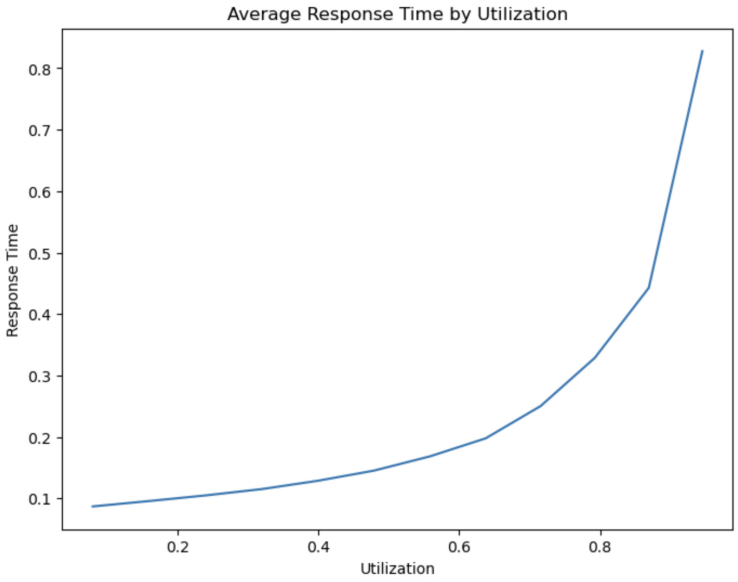

Figure 3: 2f. Average Response Time by Utilization

Relationship: As server utilization increases, the average response time increases exponentially.