CSCI 4404/5401 - Assignment-2

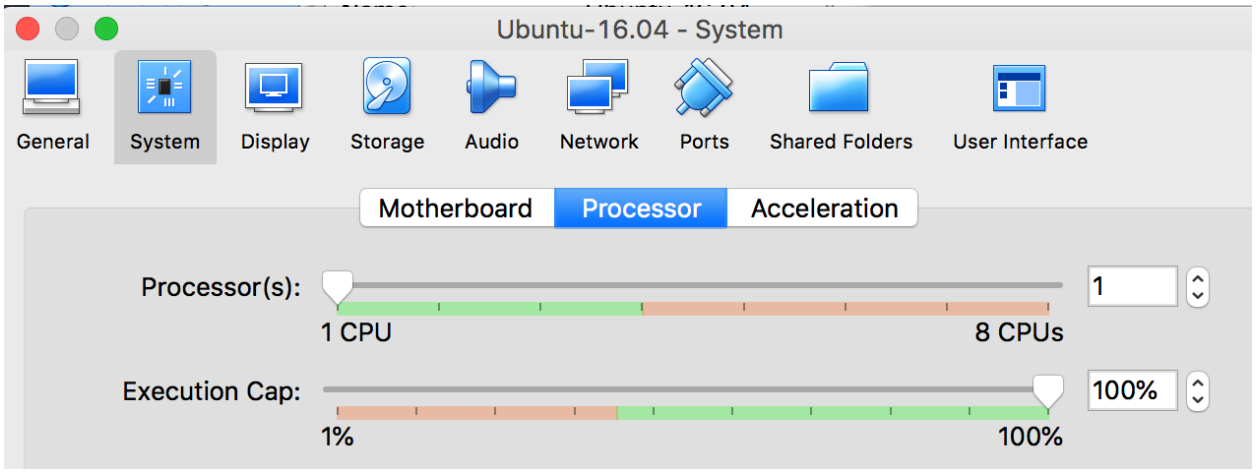
Please carefully read and adhere to **all** the below guidelines:

- This Assignment is for **100 points** for both **CSCI 4401** and **CSCI 5401** students.
- The Assignment scores will be reduced to 10 points for the final grade. For example, if a student scores 95 points in this assignment, the student will get 9.5 points credit towards the course grade.
- All answers such as output of programs you write and text should be added to a **single PDF** file titled **YOUR_FIRST_NAME_LAST_NAME.pdf**. For example, my file should be titled vadrevu.pdf. This file will be referred to as "**Main PDF**" file through out the rest of this document.
- Source code should be in **separate files**. (Not in Main PDF file). **Make sure to include the question numbers in the names of all the source code files.** These **files should also be referenced in the Main PDF file**.
- Put all the files in a directory named it **YOUR_FIRST_NAME_LAST_NAME**, compress it and upload to Moodle.
- There is a **strict late submission policy** for the assignment: **15% points** will be **deducted** for **every day** that the assignment is late.
- Pick a language of your choice among the following: C, C++, Java, Python. (If you use C/C++, please use pthread or std::thread libraries for thread-related tasks).

Skills tested:

- Use concurrency to speed up a CPU-bound task and an I/O-bound task.
- Interpret the difference in how these two tasks differ when using concurrency.

Note: Some parts of this assignment requires you to restrict the number of cores your code runs on. In order to do this effectively, I highly recommend you to use a VM such as VirtualBox. Make sure you install a guest OS that you like on VirtualBox and **use that guest OS for ALL your development efforts** in this assignment. This allows you to be consistent with your results. With this set up, you can easily change the number of CPU cores that your code has access to. You simply have to change the number of Processors (Virtual CPUs) in System settings of your VM to change the number of cores on the host machine that you want your code to access. This is because each Processor in VirtualBox corresponds to one kernel-thread in the host OS. Be sure to leave the Execution Cap as is at the default value of 100%.



Question-1 [50 points]:
In this question, your task is to evaluate the effects of using different concurrency configurations in solving a task that is CPU-bound. Here's an example of a CPU-bound task (you can either use this task or create a similar CPU-bound task of your own):

> Given a number `n`, find out if it's a prime or not. For this, you can check if any of the numbers from 2 to about `sqrt(n)` is a factor of `n` or not. If none of those are factors, you can safely conclude that it's a prime.

For this question, you can hard-code the value of `n` to a prime number. What prime number do you use? Make sure you use a prime number that is large enough to make a single-threaded (and single-processed) version of your code run for about 1 to 2 minutes in your computing set up. Let's call this **Version-1.**

For example, my quick and dirty version of the single threaded code in Python is as follows:

```
import math
import time
start_time = time.time()
n = 69881631850817231
sqrt_n = int(math.ceil(math.sqrt(n)))
for  i in range(2, sqrt_n+1):
    if n % i == 0:
        print "%s is not a prime :(" % (n,)
        break
else:
    print "%s a prime!!" % (n,)
end_time = time.time()
print "This took %.2f seconds" % (end_time - start_time)
```

I hard coded the value of n to a 17-digit prime: "69881631850817231". The reason is that this took about 85 seconds to run on my machine:

```
69881631850817231 a prime!!
This took 85.02 seconds
```

Next question, might be: "how to find these large primes?". I have 2 resources for you:

- Directly ask the awesome Wolfram Alpha tool what you want: https://www.wolframalpha.com/input/?i=random+prime+with+17+digits
- Use this great website to get some very interesting prime numbers with the number of digits that you are looking for:

https://primes.utm.edu/curios/index.php?start=16&stop=19

Once you have the single-threaded version ready, you should next develop a multi-threaded version of your code. Let's call this **Version-2.**

- For reasons discussed in class as well as here (https://melvinkoh.me/concurrent-programming-in-python-is-not-what-you-think-it-is-cjn39wijd009e25s19bb6pb17), **Python programmers should use the `multiprocessing` module and not the `threading` module for this Question**.

After this, you should be able to run both versions of the code in the following configurations:

- Version-1 on a VM with 1 processor.
- Version-2 (with 2 threads) on a VM with 1 processor.
- Version-2 (with 2 threads) on a VM with 2 processors.
- Version-2 (with 3 threads) on a VM with 3 processors.

You should run your code **5 times** in each of the above configurations. Present your results in a tabular format. Each configuration should be a column in your table. Each row should refer to one of the 5 attempts you made with all the configurations. Each cell should contain the time it took for the code run in a particular configuration (in seconds). There should also be 2 more additional rows giving the mean and median values of each column. (**Tip**: Try to automate the parsing of output from your code and computation of the mean and median values. This will save a lot of manual effort). In the end, your table should have 7 rows and 4 columns.

Question-2 [50 points]:

The below zip file contains the first 10^9 bytes of the English Wikipedia dump made on Mar. 3, 2006.
http://mattmahoney.net/dc/enwik9.zip

Your task is to find the frequency of each word length in that text. As an example, a sample text (with 1000 words) could have the following frequencies of words (frequencies table):

1 letter - 100 words, 10%
2 letters - 150 words, 15%

3 letters - 100 words, 10%
4 letters - 100 words, 10%
5 letters - 150 words, 15%
6 letters - 100 words, 10%
7 letters - 100 words, 10%
8 or more letters - 200 words, 20%

Your code should compute similar frequencies for the given file. Before counting the words in a given line, the code should first replace all characters in that line other than A-Z, a-z, 0-9, - and _ characters with spaces. The code should then split each line by space and convert it into a list of words. It should then go over each word in the line and update the word counts. At the end, the code should use the word counts to print the frequencies of various lettered words.

After you build a single-threaded version of the above, try playing around with the size of the file that I linked you to. You should strip the file down to a size such that the single-threaded version takes about 1 to 2 minutes to run. You can use the `head` command with `-n` flag to strip the file to a suitable size for you. Be sure to let me know in your write up the parameter value of `-n` you used for cutting down the file.

Tip: When initially developing your code, work with a very small sized version of the text file such that your code can run through it in seconds. This will help you to fix bugs in your code and you can then use the bigger version for your experiments.

You should write 2 versions of this code:

- One version should be written without using any concurrency - this is the baseline. All your other solutions should run faster than this. Let's call this **Version-1**.
- Another version using 2 threads. Let's call this **Version-2**. It is very important to figure out how you are going to use threads to break up this task. You should have one thread that exclusively handles File I/O tasks like reading the lines from the file. You should have another thread that does the CPU-oriented tasks like breaking the line into words and counting of words. You can use a "thread-safe" queue to let the 2 threads communicate with one another. **Python programmers should actually code Version-2 using `threading` module instead of `multiprocessing` module.**
- Next, develop another version that uses 3 threads. Let's call this **Version-3.** This should have a single File I/O thread. But the above CPU-oriented tasks of

breaking each line into words and counting them should be broken down into 2 threads. **Pythonistas can go back to using `multiprocessing` again for this version.**

Please note that all the 3 versions of the code should use the same file as input. So, the resulting counts from all 3 versions of the code should be exactly the same. If it's not the same, then, that means there is an issue in your code that you need to fix before proceeding further.

After this, you should be able to run both versions of the code in the following configurations:

- Version-1 on a VM with 1 processor.
- Version-2 (with 2 threads) on a VM with 1 processor.
- Version-2 (with 2 threads) on a VM with 2 processors.
- Version-3 (with 3 threads) on a VM with 3 processors.

You should run your code **5 times** in each of the above configurations. Present your results in a tabular format. Each configuration should be a column in your table. Each row should refer to one of the 5 attempts you made with all the configurations. Each cell should contain the time it took for the code run in a particular configuration (in seconds). There should also be 2 more additional rows giving the mean and median values of each column. (**Tip**: Try to automate the parsing of output from your code and computation of the mean and median values. This will save a lot of manual effort). In the end, your table should have 7 rows and 4 columns.

For all 3 versions, make sure to measure the time taken by each version. You should make sure that (1), (2) and (3) take decreasing amount of times. If not, it means there is something wrong with your code, that needs to be fixed. You need to measure this time accurately with time() function calls in your language of choice.

Submission Requirements for the Assignment:

1. The source code for version-1 and version-2 of Question-1
2. The time matrix (of size 7 by 4) for Question-1
3. Your comments on why you think the values are varying from column-1 to column-4 in the above matrix table. For example: which column is lowest? Which one is highest? Are any columns very close to one another? If so, why?
4. The source code for versions-1, version-2 and version-3 of Question-2

5. The parameter `n` you used with head to cut down the input file. Please don't include the
6. The output (frequencies table) for all three versions of Question-2. Note that they should all match.
7. The time matrix (of size 7 by 4) for Question-2
8. Your comments on why you think the values are varying from column-1 to column-4 in the above table. For example: which column is lowest? Which one is highest? Are any columns very close to one another? If so, why?
9. How do the 2 matrices of Question-1 and Question-2 compare? Please include your comments on how and why you think those two matrices varied differently across the 4 columns.
10. You should also include any auxiliary scripts you have such as scripts that parse your program outputs and compute the mean and median values.