CSCI 4404/5401 - Assignment-2
Due: 25th March, 2019 (Monday)

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

1. All students can score 120 points in this assignment.
2. This Assignment is for **100 points** for both **CSCI 4401** and **CSCI 5401** students. It is highly recommended that students work on all the questions in the assignment as this will increase your chances of getting the full 100 points.
3. 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.
4. You will need a Linux OS to do the questions in the assignment. Use a VM application (for example: VirtualBox) to install one if you have a non-Linux system such as a Windows or a Mac. Personally, I prefer to use Debian-based OSs such as Ubuntu.
5. All answers such as output of programs you write and text should be added to a **single PDF** file titled **YOUR_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.
6. 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**.
7. Put all the files in a directory named it **YOUR_LAST_NAME**, compress it and upload to Moodle.
8. There is a **strict late submission policy** for the assignment: **15% points** will be **deducted** for **every day** that the assignment is late.
9. The due date for the assignment is **25th March 2019 (Monday), 11:55 PM**
10. 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).

Question-1 [40 points]:

Skills tested:

- Ability to use multiple threads in a programming language of your choice.

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:

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 go over each word in the line and update the word counts. At the end, the code should use the word count to print the frequencies of various lettered words.

You should write 2 versions of this code:

1. One version should be written without using threads.
2. Write another version of the above code using threads. You should try to make sure that each thread does the work of both text preparation (i.e. replacing characters with spaces) and word counting.

For both versions, make sure to measure the time taken by each version. You need to measure this time accurately with time() function calls in your language of choice.

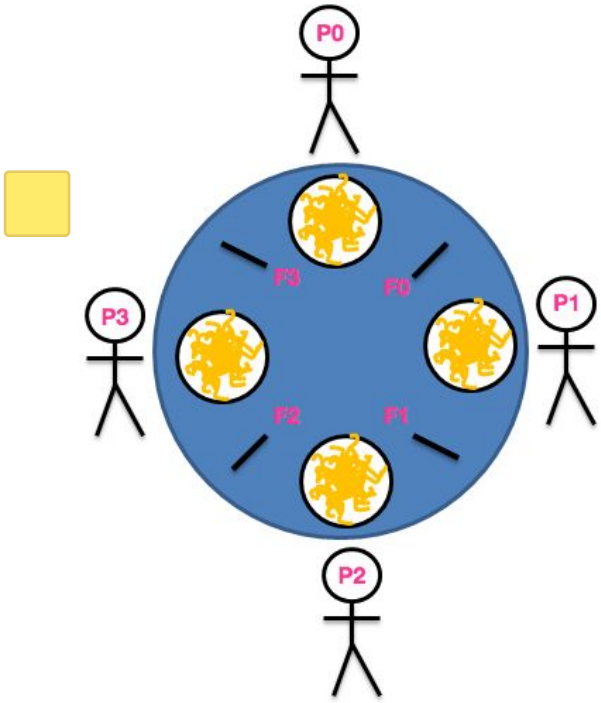
Submission Requirements:

1. The source code for version-1 which measures the word frequencies without using threads. This file name should be mentioned in the main PDF file.
2. The source code for version-2 which measures the word frequencies by using threads. This file name should be mentioned in the main PDF file.
3. The outputs (frequencies table) of version-1 and version-2 should match. This output should be reported in the main PDF file.
4. The time measured for running version-1 and version-2 should be mentioned in the main PDF file. Ideally, version-2 should take less time to run than version-1. If that's not the case, then you should reason about why that might be.

Common instructions for both Questions 2 & 3:

The following pertains to both questions 2 and 3. In both Questions 2 & 3, your task is to solve the Dining Philosophers problem by writing a program. 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).

Assume that there are 4 philosophers and 4 forks on the table. The 4 philosophers can be numbered from P0 to P3. Similarly the 4 forks can be numbered from F0 to F3. This is depicted in the diagram below. For both questions, you will need to represent the 4 philosophers by 4 different threads. Each philosopher will be thinking and eating alternatively. The thinking state lasts for a random time between 10 to 40 milliseconds (ms). The eating state lasts for a constant 10 ms.



Questions - 2 [40 points]

Skills tested:

- Use of mutexes in solving simple synchronization problems.

Implement the following solution to the dining philosophers problem:

1. Each fork should be protected by a mutex
2. Whenever a philosopher wants to eat, he/she tries to pick up the fork to their left first (i.e. P3 picks up F3 first). If available, the philosopher will pick it up.
3. If left fork is not available, the philosopher will sleep for a random time between 50 to 100 ms and then try again.
4. If available, the philosopher will pick up the left fork. If not, go to step 3.
5. If right fork is available, the philosopher will pick it up.
6. If not available, the philosopher will sleep for a random time between 50 to 100 ms and then try again.
7. If right fork is available, the philosopher will pick it up. If not, the philosopher will drop the left fork, sleep for a random time between 50-100 ms and go back to step 2. This is done to avoid deadlocks.
8. After picking up both forks, the philosopher should sleep for a random time between 10-40 ms to simulate eating.

9. Whenever a philosopher wants to think, they can think right away. To simulate this, the philosopher thread should sleep for 10 ms.

So, at any point philosopher can be either eating, waiting to eat (i.e. waiting to acquire forks) or thinking. We can call the “waiting to eat” state as “hungry” state. Each philosopher (or thread) should keep track of how long they are in hungry / eating / thinking states.

Your program should simulate this scenario for 60 seconds. Apart from the simulation, your program should do the following:

1. Each philosopher thread should write to a shared open file every time an event of interest happens. For example, these are some events that can be logged.
    - Philosopher: 0, Time: 2 ms, entering hungry state
    - Philosopher: 0, Time: 2 ms, picked up fork 0
    - Philosopher: 0, Time: 2 ms, tried to pick up fork 3, its unavailable
    - Philosopher: 0, Time: 70 ms, picked up fork 3
    - Philosopher: 0, Time: 70 ms, entering eating state. Will eat for 32 ms.
    - Philosopher: 0, Time: 102 ms, entering thinking state
    - Philosopher: 0, Time: 112 ms, entering hungry state

    Note that the above log lines will be mixed up with the log lines of other philosophers as well. Time should start from 0 ms and go up to 60000 ms (60 s).
2. Print the time that each philosopher spends in hungry, eating and thinking states respectively.
3. Compute and print the average time that all philosophers spend in hungry, eating and thinking states separately.

Submission Requirements:

1. The entire source code that you write to simulate the above solution should be submitted and the file name should be mentioned in the main PDF file.
2. The event logs from all the philosophers should be in a single file (in time order). This file should be submitted and be referenced in the main PDF file.
3. The time spent in each state by each philosopher and the average times (2 and 3 above) should be included in the main PDF file.

Questions - 3 [40 points]

Skills tested:

- Understanding and implementing an efficient solution to the dining philosophers problem.

Implement a more optimum solution to the problem mentioned in Question-2 above using mutexes. You should use the algorithm for solving the Dining Philosophers problem that was mentioned in the textbook. Similar to Question-2, you should run your program for 60 seconds.

Submission Requirements:

1. The entire source code that you write to simulate the textbook solution should be submitted and the file name should referenced in the main PDF file.
2. The event logs from all the philosophers should be in a single file (in time order). This is similar to the event logs file that was mentioned in Question-2. This file should be submitted and be referenced in the main PDF file.
3. The time spent in each state by each philosopher and the average times should be included in the main PDF file.
4. Compare these times in 3) above to the times in 3) mentioned in your answers for Question-2 and explain the difference.