CPSC 3500 (HW-3)
Programming Assignment: Multithreaded Programming
Due: May 5, 11:59 PM (on Canvas)

NOTE: Please submit your work by May 5 on Canvas. This assignment is to be done individually; you can discuss the problem with your classmates, but you should code your own solutions independently. This programming assignment would be 25% of the total assignment grade across all homeworks.

Directions:

1. Use the provided "city##.txt" files (at the CS1 folder specified below) for Part A of this assignment.
2. For final evaluation of your code, **we will be using a different set of input files**, which will be in the same format as the "city##.txt" files provided with the assignment but will have different values/content. For full credit, your code should work correctly on different input files.
3. Your programs will primarily be evaluated for functionality. The greater the number of test cases they pass, the greater will be the score.
4. Partial credit can be given. However, a program failing several test cases will receive a very low grade.
5. **No changes to the submitted programs** (however minor) will be accepted after the submission. Please test your programs thoroughly before submission and ensure that you are submitting the best version.
6. **Your programs will be evaluated on the CS1 server.** Please ensure they compile and run as expected on the CS1 server. Programs that fail to compile or run on the CS1 server will receive a zero.
7. Testing and debugging are integral part of programming. Before submission, you are responsible for creating test cases and testing your programs.
8. Sharing of code in any form is strictly prohibited. You can share the test cases, though.

Part A – Introduction to Parallel Programming (35 points)

It is vaccine season! COVID19 vaccines are out and everyone is rushing to get one. Vaccines are available from six different brands and you have been entrusted to write a program to count the total number of vaccines of each brand administered across 21 cities in the state. Write a multithreaded program (in the file `vaccines.cpp`) that counts the number of vaccines in 21 different files of numbers. Each line in each of the files will contain either one of the following brand names: v1, v2, v3. Your program must count the number of units for each vaccine brand administered, across all 21 cities, and display the total number of units for each brand.

The files are named "`city1.txt`", "`city2.txt`", … , and " `city21.txt`". They are available from the directory: `/home/fac/mishraa/cpsc3500/pa3a`

Illustrative Toy Example:

If there were only three cities, with the contents of the files as below, your program should print the following.

"city1.txt"	"city2.txt"	"city3.txt"
v1	v2	v1
v2	v3	v2
v3	v1	v2
v1	v3	v2
v2	v3	v3
v3	v2	v3

Total vaccine counts across 3 cities:

v1 = 4

v2 = 7

v3 = 7

Additional Requirements

- The program should create 21 threads, each thread should process a different input file.
- Your program must read the files from the directory `/home/fac/mishraa/cpsc3500/pa3a`.
    - Do not assume the input files are in the current directory.
    - The directory and file names must be hard-coded into the program. Do NOT prompt the user for this information.
- Assuming error-free execution, the only output the program should produce is the cumulative unit count for each vaccine brand. Do NOT print out the individual counts for each file.
- Your program must use appropriate data structures and loops to avoid redundant code. Code that replicates something 21 times (such as 21 `pthread_create` calls) will be penalized.
- Global variables are not permitted! You must properly pass parameters into the thread function and use `pthread_join` to return the tally from each file. Using a global variable for this purpose will result in a minimum 12 point penalty.
- You must use Pthreads to implement the threading. You are not permitted to use the threading features present in C++ 11.
- No makefile is required for this assignment. Your program should compile using the following g++ command on CS1 server (cs1.seattleu.edu):

```
g++ vaccines.cpp -std=c++11 -lpthread -o vaccines
```

Error Checking

- If a pthread function returns an error, abort the program with an error message.

Assumptions

- Each input file will contain at least one line. Beyond this, you cannot make any assumptions on how long the input file is.
- You may assume that each line only contains one of the three names noted in the description.

Part B – Synchronization (Social Distancing Problem) (60 points)

At a playground, kids love to play on a play play structure. However, due to an ongoing pandemic, social distancing needs to be enforced. There are two types of kids—vaccinated, and non-vaccinated. The vaccinated kids have immunity against the virus hence don't need social distancing. The non-vaccinated kids needs to maintain social distancing for their safety. The park managers came up with a solution that both the vaccinated and non-vaccinated kids cannot play at the play structure simultaneously.
Below is a pseudocode for a solution using semaphores that allows any number of vaccinated kids on the play structure at the same time, but only one non-vaccinated kid. (All the kids are represented as threads.) If a non-vaccinated kid is playing at the structure, no other kid (either vaccinated or non-vaccinated) can be allowed. Any number of vaccinated kids can be at the play structure together. The solution must implement the required mutual exclusion and avoid deadlock. However, the solution does not need to be fair.

The threads share the following data structures (the following three can be global variables in your program):

Int vaccinated_kids_count = 0; //the number of vaccinated kids currently playing at the structure
Semaphore mutex = 1 // Ensures mutual exclusion when vaccinated_kids_count is updated
Semaphore play_mutex = 1 // provides mutual exclusion for accessing the play structure

<u>Below if the pseudocode for the non-vaccinated kids threads:</u>

```
do {
        wait(play_mutex);
            ...
        /* play at the structure */
            Play(…);
           …
         signal(play_mutex);
    } while (true);
```

Below is the pseudocode for the Vaccinated-kids threads:

```
do {
        wait(mutex);
        vaccinated_kids_count++;
        if (vaccinated_kids_count == 1) {
          wait(play_mutex);
          }
         signal(mutex);
                    ...
          /* play at the structure */
          Play(…);
                    ...
           wait(mutex);
        vaccinated_kids_count --;
        if (vaccinated_kids_count == 0)
                    signal(play_mutex);
         signal(mutex);
     } while (true);
```

Here is the pseudocode for the Play():

```
Play(…){
  Print 1 //see the note below for details on the print statement
  sleep(1); //for the sleep() use #include <unistd.h>
  Print 2 //see the note below for details on the print statement
}
```

The print statements should print the following information:

- For non-vaccinated (NV) kids thread, assuming thread Id is 1:
    - Print 1: NV Thread 1 playing!
    - Print 2: NV Thread 1 playing done!
- For vaccinated (V) kids thread, assuming thread Id is 3:
    - Print 1: V Thread 3 playing!
    - Print 2: V Thread 3 playing done!

You can decide the function prototype for Play() as needed for your program, but the Play() function must: 1) have a sleep() for 1 second as shown above; 2) print the thread Id and type of thread (vaccinated or non-vaccinated kid thread) executing in play, as illustrated above.

Additional directions:

NOTE: Use "printf(…)" statements for printing; do not use "cout". Otherwise, your output may not get printed together.

Add the following print statements in the non-vaccinated kids threads. You will need to print the thread Id of the thread with every statement. The example statements below are assuming the thread Id of the non-vaccinated kids thread is 1. "NV" is abbreviation for Non-Vaccinated.

1. Just before the "wait(play_mutex)" print: "NV Thread 1 trying to acquire play_mutex."

Add the following print statements in the vaccinated kids threads. You will need to print the thread Id of the thread with every statement. The example statements below are assuming the thread Id of the vaccinated kids thread is 3. "V" is abbreviation for Vaccinated.

1. Immediately after the first wait(mutex) print: "V Thread 3 acquired mutex for incrementing the count."
2. Just before the "wait(play_mutex)" print: "V Thread 3 trying to acquire play_mutex."
3. Just before the first signal(mutex) print: "V Thread 3 about to release mutex after incrementing."
4. Immediately after the second wait(mutex) print: "V Thread 3 acquired mutex for decrementing the count."
5. Just before the signal(play_mutex) print: "V Thread 3 about to release play_mutex."
6. Just before the second signal(mutex) print: "V Thread 3 about to release mutex after decrementing."

- The above print statements will help you debug your code and verify its correctness.
- Include all the above print statements as directed.
- In your final submission, do not include any other print statements besides the ones mentioned above.
- If we are unable to verify the correctness of your implementation, for instance, due to missing or unexpected or inaccurate print (and sleep()) statements, you will not get credit for the unverified features.

Compiling and running your program of CS1:

- Your program should accept the following two integers as command line inputs: first, number of vaccinated kids threads to be created; second, number of non-vaccinated kids threads to be created.
    - If less than two inputs are supplied or unexpected input (e.g., floats instead of int) is supplied, print out an informative message asking for the correct inputs and terminate the program.

    - Tutorial on command line parameters: http://www.cplusplus.com/articles/DEN36Up4/
- No makefile is required for this assignment. Name your program "social_distancing.cpp". Your program should compile using the following g++ command on CS1 server (cs1.seattleu.edu):
    - g++ social_distancing.cpp -std=c++11 -lpthread -o social_distancing
- for executing the program on CS1:
    - ./social_distancing 3 6
    - Above sample execution command will create three vaccinated kid threads and six non-vaccinated kid threads in your process.
- As indicated by the logic in the pseudocodes, both kinds of threads will run indefinitely, use Ctrl+C to terminate/kill the program.

Testing your code:

Add a main method to test your code.

A) First, simply create and run your program with only one non-vaccinated kid thread to check if all the print statements are included as expected.
B) Second, simply create and run your program with only one vaccinated kid thread to check if all the print statements are included as expected.
C) Next, create one vaccinated kid thread and one non-vaccinated kid thread. Make sure both are not playing at the structure together, there are no deadlocks, and both kids are able to play at least once.
D) Finally test your code for various combinations of multiple threads of each kind; for instance, three vaccinated and three non-vaccinated kids, six vaccinated and four non-vaccinated kids, etc.

Notes:

1. Your program **must compile and run on CS1**.seattleu.edu. Programs not compiling on CS1 will receive a zero.
2. The output of your program should be intelligible/readable. If multiple threads run concurrently, their outputs can be interleaved, however, to make the output readable, make sure the output of each print statement is displayed on a new line and at one place.
3. You may find the following implementation of reader-writer problem useful reference: https://iq.opengenus.org/reader-writer-problem-cpp/

3) Prepare a “README.txt” file for your submission. The file should contain the following: (5 points)

a) Instructions for compiling and executing your program(s). Include example command lines for the same.
b) If your implementation does not work, you should also document the problems in the README file, preferably with your explanation of why it does not work and how you would solve it if you had more time.

What should you submit?

Upload your submission as a zip file on Canvas. The zip file should contain:

1. All your code files and any other files that might be needed for executing your code.
    a. For Part A, submit vaccines.cpp (plus any other supporting files needed for your program)
    b. For Part B, submit social_distancing.cpp (plus any other supporting files needed for your program)
2. README.txt.