Assignment 6-1: Working with Pthread Condition Variables in C++

Assignment Instructions:

- You must use the virtual environment that you set up in Exercise 1-1-3 for this assignment.
- Be sure to compile and test each program to be certain it works as expected. If you aren't sure how to compile and run a C++ program, refer to the Build and Execute Program section of the setup instructions document.

Important notes:

- At the top of your .cpp file, please include a comment with your **full name**. If your section uses **Lightweight Teams**, add the names of the teammates whom you worked with to the same comment.
- Add your own **individual comment** for each function / major portion of code that you add, briefly explaining what that part does.
- If you are asked to submit screenshots and your submitted screenshots do not match with your program's actual behavior, we will consider that to be a **violation of academic integrity** and pursue it accordingly.
- Make sure to **organize and format** your code in a consistent way.
- If you refer to any online resource to understand a concept, see examples of the use of a particular syntax, etc., add a comment **citing** that resource (i.e., specify website name and link).
- You must only submit **.cpp** files. If you have multiple .cpp files, upload them individually and **not** as a zip / compressed file.
- No screenshot(s) will mean no grade for this assignment.

Assignment Objectives:

- Practice the use of pthread **condition variables** in the context of a **producer-consumer** problem

Assignment Tasks:

The goal of the assignment is for you to practice and use functions related to **Pthread condition variables** in Unix based OSs.

For your reference, an online version of the Linux manual can be found here: **http://linux.die.net/man/**. Another useful resource is the POSIX Threads Programming page at Lawrence Livermore National Laboratory by Blaise Barney [URL: https://computing.llnl.gov/tutorials/pthreads/].

If you need help with **navigating** the file system through a command line terminal, refer to this: http://linuxcommand.org/lc3_lts0020.php

Assignment Setup (0 points)

Note:

- *You need to use the terminal to compile and run the program. Do not use your IDE's GUI.*

1. You will need to download, compile, and execute programs using your virtual environment.

2. Type the following command ***into the terminal window*** to pull the project repository from GitLab:

```
git clone https://cci-git.charlotte.edu/hramapra/ITSC_3146_A_6_1.git
```

3. Change directory into the newly created directory (folder) named ITSC_3146_A_6_1. You will work in this directory for the rest of this assignment.

Note:

1. Issue the following command to compile the program in this directory:

```
g++ producer_consumer.cpp -o producer_consumer -lpthread
```

2. Issue the following command to execute the program built above:
   `./producer_consumer`

Part 1: Producer-Consumer - execution (0 points)

1. The producer_consumer.cpp program given to you simulates a highly simplified restaurant order taking and processing system using pthreads. Study this program, using the comments in the code to help you understand the functionality.

2. Build and execute this program multiple times. You will observe that the program behaves strangely:
    a. Some random, dummy orders that were never placed may get processed.
    b. Some orders entered by the user may not be processed at all.
    c. The message "Phew! Done with orders for today!" may get printed even before orders are placed.

    All these problems arise because of lack of synchronization among the *processOrders* and *takeOrders* threads.

    Take a screenshot of a sample output and upload the picture as part of your assignment submission.

Part 2: Producer-Consumer - modification (15 points)

Copy producer_consumer.cpp into a new file named producer_consumer_with_synchronization.cpp and modify the copy as instructed below (*refer to your prep work videos/slides if you need to help you complete this assignment*).

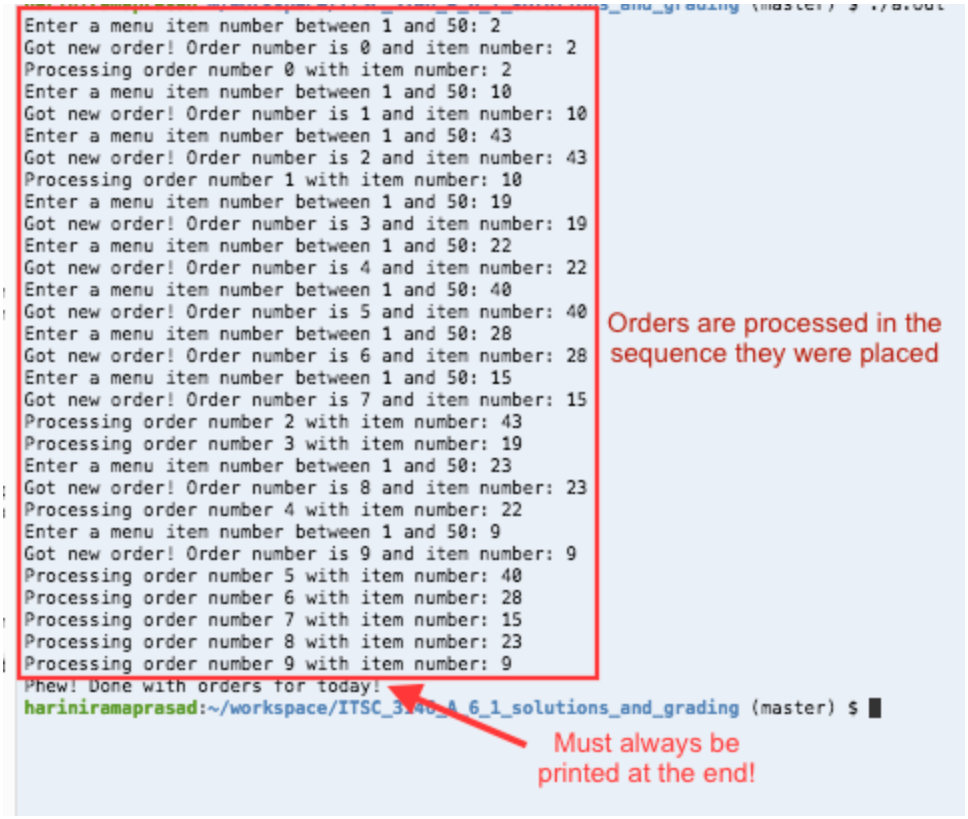
1. Protect access to shared (i.e., global) data variables used in the *processOrders* and *takeOrders* threads. As part of this:
    a. Create a global pthread mutex variable named **data_mutex** and initialize it to default attributes using the macro provided by the pthread library.
    b. Insert mutex lock/unlock statements before/after regions of code using shared data in the two pthread functions. Note: the beginning and end of all critical regions are marked. Identify and protect critical region(s) that are relevant to this task.
2. Protect access to the console in the *processOrders* and *takeOrders* threads (*remember that the console that cin & cout access is a shared resource*). As part of this:
    a. Create a global pthread mutex variable named **console_mutex** and initialize it to default attributes using the macro provided by the pthread library.
    b. Insert mutex lock/unlock statements before/after regions of code where the console is accessed in the two pthread functions. Note: the beginning and end of all critical regions are marked. Identify and protect critical region(s) that are relevant to this task.

3. Use condition variables to synchronize operations between the *processOrders* and *takeOrders* threads. As part of this:
    a. Create necessary global pthread condition variables and initialize all of them to default attributes using the macro provided by the pthread library.
    b. Insert pthread condition wait statements in *processOrders* and *takeOrders* threads such that:
        i. no new orders are put into the buffer unless there is at least one space in the buffer, i.e., no new orders are put into the buffer if the buffer is full;
        ii. orders are not retrieved from the buffer unless there is at least one new order, i.e., orders are not retrieved if the buffer is empty.
    c. Insert corresponding condition signal statements in *processOrders* and *takeOrders* threads so that the above waits will end when the conditions being waited for are satisfied.
4. In the *main* function, after both threads are created and before printing the goodbye message, insert code to ensure that the main thread waits for both pthreads to complete.
5. Build and run your program. When prompted, enter 10 unique integers in the range of 1 to 50. If you have put in all synchronization statements correctly, your program must adhere to the following:
    a. Print statements must not be interleaved (*i.e., a cout statement in a given thread must not be interrupted by a cout statement in a different thread*).
    b. Only orders that have been placed must get processed.
    c. All orders that are placed must get processed, in the order in which they were placed.
    d. Multiple orders may be placed before the first one gets processed.
    e. The message "Phew! Done with orders for today!" must always be printed and printed last.

Sample Output:

Below is a sample of what your output should look like after adding appropriate synchronization constructs:

Take a screenshot of a sample output and upload the picture as part of your assignment submission.



Caution: Before you submit, make sure that you have followed all the instructions under Assignment Tasks and Important notes and that you have taken screenshots as indicated in the assignment.

Assignment Submission Items:

The files that need to be submitted for this assignment are the following:

- Producer_consumer_with_synchronization.cpp
- The necessary output screenshot for all cpp files.

Note: No screenshot(s) will mean no grade for this assignment.