Assignment 5-1: Working with Pthreads - Strict Alternation

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:

- Understand the functionality of the **`pthread_join`** function
- Practice the use of the **`pthread_join`** function in a program
- Apply the concept of **mutual exclusion** in the context of a C++ program
- Better understand solutions to the **mutual exclusion** problem
- Apply **strict alternation** to address the problem of **mutual exclusion**

Assignment Tasks:

The goal of the assignment is for you to practice and use functions related to **POSIX threads or Pthreads creation & handling** in Unix based OSs. An online version of the Linux manual can be found here:**http://linux.die.net/man/**.

For this activity, you will need to refer to the **pthreads section** of the Linux manual, available here:https://linux.die.net/man/7/pthreads

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 a small program 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/jbahamon/ITSC_3146_A_5_1.git
```

3. Change directory into the newly created directory (folder) named ITSC_3146_A_5_1

4. Issue the following command to compile one of the programs:

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

5. Issue the following command to execute the program:
   `./pthread_join`

Note:

- When compiling each of the programs in this assignment if you get the following error:

```
error: 'rand' is not declared in this scope
```

Simply add `#include <stdlib.h>` to the top of the file

Part 1: Waiting for a Thread to Terminate (5 points)

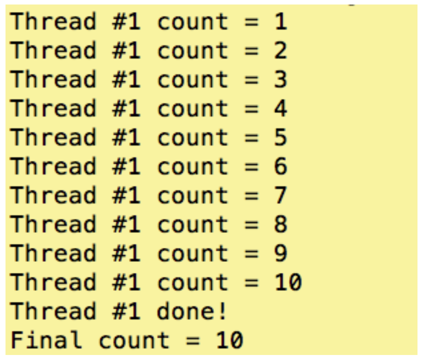
1. **Execute** the **`pthread_join`** program **several** times.
2. Examine the output carefully. You should notice a problem in the implementation. Make sure to follow the logic in **main()** and to read the comments carefully.
3. Read the Linux manual page that describes the **pthread_join** function [URL: https://linux.die.net/man/3/pthread_join] and make sure that

you understand the functionality that **pthread_join** provides and how to call (invoke) this function.

4. **Correct the problem**. Look for the **// TODO** comment and address it (i.e., implement the functionality described in the comment). *Hint: The solution requires only one (1) line of code.*
5. Build and run your program and make sure that it works correctly.

Expected Output:

Your program should produce the following output



```
Thread #1 count = 1
Thread #1 count = 2
Thread #1 count = 3
Thread #1 count = 4
Thread #1 count = 5
Thread #1 count = 6
Thread #1 count = 7
Thread #1 count = 8
Thread #1 count = 9
Thread #1 count = 10
Thread #1 done!
Final count = 10
```

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

Part 2: Pthread Data Sharing (5 points)

1. A file named **`pthread-data-sharing-mutex.cpp`** has been provided to you in the same project.
2. Compile the program and **execute it several times,** at least 10. Make sure to **pay close attention to the output that the program produces**.
   a. Create a Word or Google Docs document.
   b. In this document, answer the following questions about the program's behavior:
      i. What does it do?
      ii. What output does it produce?
      iii. Examine the program code carefully. Is the program functioning correctly?
      iv. If you do not think that the program is working correctly, describe why?

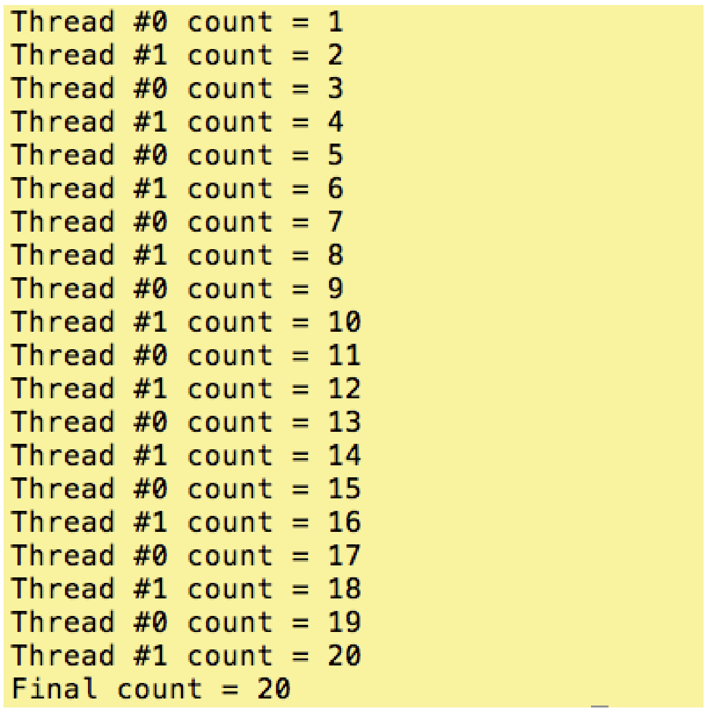
Take a screenshot of a sample output and upload the picture as part of your assignment submission.

Part 3: Strict Alternation (10 points)

1. A file named **`pthread-data-sharing-mutex-strict-alternation.cpp`** has been provided to you in the same project.
2. Compile the program and make sure that it executes.
3. Examine the program code. **Note that except for some minor changes, this program is identical to the one you used in Part 2 of this assignment.**
4. Modify the program to implement the **strict alternation** solution to achieve mutual exclusion (*refer back to the relevant prep work video/slides if you need to;* ***IMPORTANT NOTE:*** *the* ***outer, infinite while loop*** *in the prep work video/slides is just an example and is* ***not*** *part of the strict alternation solution; only the empty while loop before the call to the critical region and the line immediately following the call to the critical region are part of the strict alternation solution*).
5. Build and execute the updated program several times.

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

   Expected Output:
   Your program should produce the following output (Note: it does not matter whether Thread #0 goes first or Thread #1, but it is important that the threads ***strictly alternate***):



```
Thread #0 count = 1
Thread #1 count = 2
Thread #0 count = 3
Thread #1 count = 4
Thread #0 count = 5
Thread #1 count = 6
Thread #0 count = 7
Thread #1 count = 8
Thread #0 count = 9
Thread #1 count = 10
Thread #0 count = 11
Thread #1 count = 12
Thread #0 count = 13
Thread #1 count = 14
Thread #0 count = 15
Thread #1 count = 16
Thread #0 count = 17
Thread #1 count = 18
Thread #0 count = 19
Thread #1 count = 20
Final count = 20
```

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:

- pthread_join.cpp
- Document/text file that contains the answers to the questions in part two of the assignment.
- pthread-data-sharing-mutex-strict-alternation.cpp
- The necessary output screenshots for both cpp files.

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