Assignment 4: Multi-threaded Producer Consumer Counter

Due Nov 21, 2022 by 11:59pm **Points** 75 **Submitting** a file upload
Available Oct 31, 2022 at 12:01am - Nov 23, 2022 at 11:59pm

This assignment was locked Nov 23, 2022 at 11:59pm.

Introduction

In this assignment, you'll write a C program called "myCounter" that will get you familiar with the use of threads, mutual exclusion and condition variables.

Learning Outcomes

- Describe what is mutual exclusion and why is it an important property to maintain when developing programs that may concurrently access shared resources (MLO 2)
- Describe the API you can use to create threads and wait for the termination of a thread (MLO 4)
- Describe what are condition variables and the API related to their use (MLO 4)

Instructions

Use slides 50 and 51 from Lecture 2.3 to create a program that uses two threads: one thread (the one that your program starts out with) will be the Producer, while the other thread (that you'll create) will be the Consumer.

Your program must use these 2 threads to communicate with each other using a Producer-Consumer approach, as described in this lecture.

Objective and Variables

Both threads must share one mutex and two condition variables to control and protect the counting of a number. The number must count from a starting value of 0 to 10, by ones, at which point the program will end. You get to decide which parts of the incrementation and printing go in each thread.

Your variables must be named as follows:

- Your mutex must be named "myMutex".
- Your two conditions variables must be named "myCond1" and "myCond2".
- Your counting variable must be named "myCount".

Output

Your program must output lines that contain the following text exactly as written, at the following times. No other lines are allowed to be in the output:

- When your program begins:

```
PROGRAM START
```

- When thread 2 is created:

```
CONSUMER THREAD CREATED
```

- When myCount changes value:

```
myCount: <PREVIOUS#> -> <NEW#>
```

  Example:

```
myCount: 1 -> 2
```

- When myMutex is unlocked:

```
<THREAD>: myMutex unlocked
```

  Example:

```
CONSUMER: myMutex unlocked
```

- When myMutex is locked:

```
<THREAD>: myMutex locked
```

  Example:

```
CONSUMER: myMutex locked
```

- When myCond1 or myCond2 has pthread_cond_wait() called on it:

```
<THREAD>: waiting on <CONDITION VAR>
```

  Example:

```
PRODUCER: waiting on myCond1
```

- When myCond1 or myCond2 has pthread_cond_signal() called on it:

```
<THREAD>: signaling <CONDITION VAR>
```

  Example:

```
CONSUMER: signaling myCond1
```

- When your program ends:

```
PROGRAM END
```

What to turn in?

- You can only use C for coding this assignment and you must use the gcc compiler.
- You can use C99 or GNU99 standard or the default standard used by the gcc installation on os1.

- Your assignment will be graded on os1.
- Submit a single zip file with all your code, which can be in as many different files as you want.
- This zip file must be named `youronid_program4.zip` where youronid must be replaced by your own ONID.
  - E.g., if chaudhrn was submitting the assignment, the file must be named `chaudhrn_program4.zip`.
- In the zip file, you must include a text file called `README.txt` that contains instructions on how to compile your code using gcc to create an executable file that must be named `myCounter`.
- When you resubmit a file in Canvas, Canvas can attach a suffix to the file, e.g., the file name may become `chaudhrn_program4-1.zip`. Don't worry about this name change as no points will be deducted because of this.

Grading Criteria

- The points for the assignment and the break-up for items is described in the grading rubric.
- The "waiting on" lines can be missing, or only sporadically appear, as the code doesn't always need to use pthread_cond_wait(). This can happen because your own internal logic might cause the wait to be unnecessary at least some of the time, and potentially all of the time.
- The order of the lines displaying the mutex and condition variables changing is allowed to change, and extra lines displaying these changes as the program initializes are acceptable.

Assignment 4 - Counter

Criteria	Ratings		Pts
PROGRAM START	**5 pts** **Full Marks** The very first line of output is exactly: PROGRAM START	**0 pts** **No Marks** The very first line of output is NOT exactly: PROGRAM START	5 pts
CONSUMER THREAD CREATED	**5 pts** **Full Marks** The second line of output is exactly: CONSUMER THREAD CREATED	**0 pts** **No Marks** The second line of output is NOT exactly: CONSUMER THREAD CREATED	5 pts
Counter Changes	**20 pts** **Full Marks** Each counter change is displayed using the provided example. All counter changes are displayed, including the 9 -> 10 change. All counter changes are done in concert between the two threads, mutex, and two condition variables.	**0 pts** **No Marks** Any one counter change is not displayed, or the changes are being done without the required methods.	20 pts
mutex Locking	**20 pts** **Full Marks** The mutex is locked before the count is updated, each time, and the mutex is after the count is updated, each time. Exception: you do not have to display the final mutex unlock line, if you do not want to (i.e. your program terminates as soon as myCount = 10).	**0 pts** **No Marks** At least once, the mutex is NOT locked before the count is updated.	20 pts
condition Variables	**20 pts** **Full Marks** The two condition variables are toggled on and off as per the Lectures, in a way that protects the counter. This will be demonstrated by your code displaying the "...waiting on..." and "...signaling..." lines, and by examining your code.	**0 pts** **No Marks** At least once, your code alters the counting variable without using the two condition variables, and/or your program doesn't display the state of these condition variables correctly.	20 pts
PROGRAM END	**5 pts** **Full Marks** The very last line of output is exactly: PROGRAM END	**0 pts** **No Marks** The very last line of output is NOT exactly: PROGRAM END	5 pts
			Total Points: 75