CS3500 - Operating System, August 2022

Lab 5: Thread Synchronization

Due Date: Friday, 26th September 11.59PM in Moodle.
Demo Date (if required): 30/09/2022

1. **Dining Philosopher Problem**

The task is to implement the classic synchronization - Dining philosopher problem.

Introduction
Say there are N (the value of N will be given as command line argument) philosophers who spend their time either thinking or eating. They sit around a circular table. Food is kept on the middle of the table and a chopstick is placed in between each philosopher. Philosophers think for some time during which they do not interact with others. Once they are hungry, they pick up the chopsticks on the right and left to eat (one after another if both are available). If any of the chopsticks are not available, they go back to thinking and check after some time. When a hungry philosopher has both chopsticks at the same time, he eats without releasing the chopsticks. When he is finished eating, he puts down both chopsticks and starts thinking again.

Implementation
Each philosopher should be implemented as a thread. Each philosopher thinks for 50-100 milliseconds at a time (you can pick a random number from this range in every iteration) and eats for 100 milliseconds. Each philosopher must eat 5 times. Use the concept of semaphores for the synchronization between the threads. And you can use mutex to execute a set of instructions atomically.

Print the messages: (Print the messages as it is, It will be easier for TAs to evaluate).
<time> Philosopher i in THINKING state
<time> Philosopher i in EATING state - <count>
where <time> is time with respect to start time of program and <count> is the count Philosopher i is eating.

Program will be run as follows:
$ ./phil N

The value of N will be passed as a command line argument.
Sample Out for N = 3
<0> Philosopher 0 in THINKING state
<0> Philosopher 1 in THINKING state
<0> Philosopher 2 in THINKING state
<100> Philosopher 0 in EATING state - 1
<190> Philosopher 0 in THINKING state
<290> Philosopher 2 in EATING state - 1
<391> Philosopher 2 in EATING state - 2
<461> Philosopher 2 in THINKING state
<561> Philosopher 1 in EATING state - 1

<631> Philosopher 2 in THINKING state
<727> Philosopher 1 in THINKING state
<827> Philosopher 0 in EATING state - 2
<918> Philosopher 0 in THINKING state
<1018> Philosopher 2 in EATING state - 3
<1085> Philosopher 2 in THINKING state
<1186> Philosopher 1 in EATING state - 2
<1281> Philosopher 1 in THINKING state
<1382> Philosopher 0 in EATING state - 3
<1482> Philosopher 0 in THINKING state
<1582> Philosopher 2 in EATING state - 4
<1640> Philosopher 2 in THINKING state
<1741> Philosopher 1 in EATING state - 3
<1813> Philosopher 1 in THINKING state
<1913> Philosopher 0 in EATING state - 4
<1969> Philosopher 0 in THINKING state
<2069> Philosopher 2 in EATING state - 5
<2142> Philosopher 2 in THINKING state
<2243> Philosopher 1 in EATING state - 4
<2297> Philosopher 1 in THINKING state
<2397> Philosopher 0 in EATING state - 5
<2466> Philosopher 0 in THINKING state
<2566> Philosopher 1 in EATING state - 5
<2665> Philosopher 1 in THINKING state

Please submit a C program with the name ***phil.c*** and a Makefile which clearly specifies how the executable ***phil*** is being generated.

Follow the Instructions strictly so that the evaluation can be done easily.

2. **Multiple Producer-Consumer Problem**

Introduction
There are P number of producers and C number of consumers. The capacity of each producer is given by PC and capacity of each consumer is given by CC. All these values will be given as command line arguments. All the producers and consumers will have a shared buffer which you can consider to be a stack like data structure. Whenever a producer produces an item (a number in our case), it is placed on top of the stack. And similarly, whenever a consumer wants to consume an item, it picks a number from the top of the stack. All the producers and consumers should be implemented as threads. Each producer will only produce a PC number of items and then stop producing. Each consumer will try to consume a CC number of items and then stop consuming. The program ends once all producers are done with producing a PC number of items and all consumers are done consuming a CC number of items. You can assume that (P * PC >= C * CC).

Implementation
Each producer and consumer should be implemented as a thread. Use semaphores to synchronize between the producers and consumers. You can consider the items generated by the producers to be random numbers.

The program will be run as follows:
$ ./multi_pc -p 4 -pc 2 -c 2 -cc 2

Output for above execution is:

Producer 1 produced 434
Producer 4 produced 435
Producer 3 produced 118
Producer 2 produced 402
Consumer 1 consumed 402
Consumer 2 consumed 118
Producer 2 produced 847
Consumer 1 consumed 847
Consumer 2 consumed 435
Producer 3 produced 624
Producer 4 produced 427
Producer 1 produced 475

Please submit a C program with the name ***multi_pc.c*** and a Makefile which clearly specifies how the executable ***multi_pc*** is being generated.

2.2 **Answer the following question:**
Suppose if you start the consumer threads, but put all producer threads to sleep for (say) 5 seconds, then will the output of your program change? Explain briefly why it will or will not change your output of the program. You can make changes to your program and see what happens with this change.
Submit the answer as a report