Homework 3

CSCE 5640 Operating System Design, Fall 2024

Due 11/15/2024 at 11:59pm

All programs must compile and execute on the CSE machines!

1. (30) Using simple semaphores, synchronize 8 threads to perform the following scenario: Assume one of the 8 threads is a producer thread that generates one item per execution and places them into a bucket, i.e., there is one producer thread and 7 consumer threads. The producer thread will only be allowed to start if the bucket is empty. It will then execute 20 times, producing one item per iteration. The bucket can hold at most 20 items. Only after the bucket is filled with 20 items are the remaining 7 consumer threads allowed to consume the items from the bucket. Each consumer consumes one single item per execution cycle. Consumers continue to execute until all 20 items are consumed. They then have to wait until the producer has generated 20 new items before starting again. *Hint*: Recall the Producer-Consumer solution discussed in class and see Sections 7.1.1 (The Bounded-Buffer problem), 7.1.2 (The Readers-Writers problem) also.
   Use POSIX Pthreads and Semaphores (section 7.3.2) in your program.
   Comment your code and submit it in a file named **prodcons.cpp.**

2. (20) Homework 2 asked you to design a multithreaded program that estimated π using the Monte Carlo technique. In that exercise, you were asked to create a single thread that generated random points, storing the result in a global variable. Once that thread exited, the parent thread performed the calculation that estimated the value of π. Modify that program so that you create 15 threads, each of which generates random points and determines if the points fall within the circle. Each thread will have to update the global count of all points that fall within the circle. Protect against race conditions on updates to the shared global variable by using mutex locks.
   Comment your code and submit it in a file named **computepiMultiThread.cpp.**

3. (20) Write a program consisting of 3 processes (or threads) that deadlock.
   Comment your code and submit it in a file named **deadlock.cpp.**

4. (10) Consider a system consisting of two processes, P_0 and P_1, each accessing two semaphores, S and Q, set to the value 1:

```
  P0              P1
wait(S);        wait(Q);
wait(Q);        wait(S);
.               .
.               .
.               .
signal(S);      signal(Q);
signal(Q);      signal(S);
```

   In the above situation, there is a potential deadlock scenario involving processes P_0 and P_1 and semaphores S and Q. Draw the resource-allocation graph that illustrates deadlock under the scenario presented in this situation.

5. (10) Consider the following snapshot of a system:

	Allocation				*Max*			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
T_0	1	2	0	2	4	3	1	6
T_1	0	1	1	2	2	4	2	4
T_2	1	2	4	0	3	6	5	1
T_3	1	2	0	1	2	6	2	3
T_4	1	0	0	1	3	1	1	2

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (2, 2, 2, 3)
b. ***Available*** = (4, 4, 1, 1)
c. ***Available*** = (3, 0, 1, 4)
d. ***Available*** = (1, 5, 2, 2)

6. (10) Consider the following snapshot of a system:

	Allocation				*Max*				*Available*			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
T_0	3	1	4	1	6	4	7	3	2	2	2	4
T_1	2	1	0	2	4	2	3	2				
T_2	2	4	1	3	2	5	3	3				
T_3	4	1	1	0	6	3	3	2				
T_4	2	2	2	1	5	6	7	5				

Answer the following questions using the banker's algorithm:

a. Illustrate that the system is in a safe state by demonstrating an order in which the threads may be completed.
b. If a request from thread T_4 arrives for (2, 2, 2, 4), can the request be granted immediately?
c. If a request from thread T_2 arrives for (0, 1, 1, 0), can the request be granted immediately?
d. If a request from thread T_3 arrives for (2, 2, 1, 2), can the request be granted immediately?

Submission:

- We will be using an electronic homework submission on Canvas to make sure that all students submit their programming tasks on time. You will submit both the program source code file(s) to the **Homework 3 dropbox** on Canvas by the due date and time.
- Make sure you submit the following files for this homework (do not submit a zip file):
  - **prodcons.cpp**
  - **computepiMultiThread.cpp**
  - **deadlock.cpp**
  - **hw3sol.docx (or pdf)** (which contains answers to questions #4, 5, and 6)
- Program submissions will be checked using a code plagiarism tool against other solutions, including those found on the Internet, so please ensure that all work submitted is your own. Any student determined to have cheated will receive an **'F'** in the course and will be reported for an academic integrity violation.

- Your programs will be tested on our Linux CSE machines. So, make sure you test your programs on the CSE machines fully before you submit on canvas.
- Until you are comfortable working on our Linux CSE machines, as a safety precaution, do not edit your program (using vim or nano) after you have submitted your program where you might accidentally re-save the program, causing the timestamp on your file to be later than the due date. If you want to look (or work on it) after submitting it, make a copy of your submission and work off that copy. Should there be any issues with your submission, this timestamp on your code on the CSE machines will be used to validate when the program is completed.