McGill University
COMP 310/ECSE 427
Dr. Rola Harmouche

Assignment #3 **Winter 2018**
Due Date: Check mycourses
Drafted By: Aakash Nandi ,Dimitri Gallos and Dr. Rola Harmouche

1 Theoretical Questions

1. When we already use 2 semaphores in the producer-consumer problem why is there a need for mutex?
2. Is it possible that a consumer with lowest priority suffers from starvation in the 2 semaphore and 1 mutex setup for producer-consumer.Explain the situation.
3. Though binary semaphores avoid starvation and mutexes don't, why is it recommeded to use mutex in producer-consumer to secure the critical section?
4. Why do producer-consumer **need to** have 2 semaphores namely "Full" and "empty". What complications may arise if we use only one semaphore for "Full" and rely on computed complementary value for "empty".

2 Disk Scheduling Algorithm

In the given skeleton for the disk scheduling part, your task is to implement the following algorithm for a single batch.

1. FCFS (Implemented)
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. C-LOOK

FCFS is the most basic algorithm which has been pre-implemented in the skeleton. You are supposed to modify only the contents of accessPOLICY() functions corresponding to each policy.

In some policies where the disk head jumps to extremities, introduce it in the output sequence.Note that these extremities might not be present in the output sequence depending on the input given.

The printSeqNPerformance() function takes your output sequence as an array of integers and prints it. It also computes the number of cylinders traversed considering the cylinders traversed even when there is a jump back of head, as a result this metric will be different from what was there in the lecture slides.This metric is only for assignment-evaluation purpose.In general cylinder traversed in jump back is not a part of performance.

3 Bankers Algorithm

This part of the assignment consists of implementing a system that simulates allocating resources to concurrent processes requesting them in such a way as to avoid deadlocks. Processes will request resources as they need them, and will finish when they are done requesting and being granted all their resources. If a process cannot be granted the requested resources, it will block until another process terminates and relinquishes all their resources.

1) Question 1

You are asked to first design a system that takes as input the number of processes, the number of district resources, the amount of each resource in the system, the maximum resource claim per process / resource. The system will then simulate each process using a thread, and all threads will run concurrently until they have used all the resources they require, after which they will terminate. Every time a process thread runs, it requests i_j instances of each resource j, where i is randomly selected as a value between 0 and the remaining amount of needed resources to terminate. It then checks if granting the resource request will result in a safe state. If so, the resources are granted to the process and the process continues (by sleeping for 3 seconds). Otherwise, the process is blocked. You are to ensure that mutual exclusion / synchronization are respected at the appropriate locations using semaphores and mutexes. The flow for each process thread is thus as follows:

1. Processes begin
2. while a process executes, a resource request vector is generated for that process. This request vector contains i_j instances of each resource j, where i is randomly selected as a value between 0 and the remaining amount of needed resources to terminate.
3. A deadlock avoidance algorithm is then run (Bankers) in order to check whether allocating these resources will result in a safe state.
4. if allocation is safe, then the process acquires the resources and the can continue executing.
5. it checks at this point if there are remaining requests to be made by that process, if not, then the process can terminate. If there are remaining requests, then simulating continued execution is done by sleeping for 3 seconds and then making a new request.
6. if allocation is unsafe, then the process blocks until another process finishes and relinquishes its resources.

2) Question 2

The above system is working in ideal conditions, where the number of resources is not changing throughout. Banker's algorithm works under the assumption that the number of instances of each resource does not change. We would like to see what happens when this condition gets violated. To do so, we will now introduce faulty resources to the system. This will be done through a thread called "faulty_thread", which will run every 10 seconds, and at each run create a fault in only one of the resources (the resource is selected randomly with a uniform probability) with a probability of 50%. If a fault occurs for a particular resource, then the number of available instances for that particular resource is decreased by 1. Now given that the number of resources is changing over time, Banker's algorithm may not be able to avoid deadlock in this case.

So for this situation, though the same scheme as above will be implemented (i.e. you will still use Banker's algorithm to allocate resources), deadlock might still occur. You will thus need to detect deadlock. The deadlock detection scheme we will use in this case will be run every 10 seconds, and will check if the process needs are higher than the available resources for all processes. If no process can acquire all the resources it needs to complete, then deadlock is inevitable. In this case, you should output the following and then exit:

Deadlock will occur as processes request more resources, exiting...

4 Submission guidelines

1) Theoretical Section

Please place Mcgill_Id_part1.pdf (ex 260123456_part1.pdf) with your answers written against the corresponding question number, inside submission folder.

2) Disk Scheduling Algorithm

Rename your C file as Mcgill_Id_part2.c and place it in the submission folder.

3) Bankers Algorithm

The following files need to be inserted in the submission folder

1. program simulating ideal conditions called resource_request_simulator.c (for question 1)
2. program with faulty conditions called faulty_resource_request_simulator.c (for question 2)

Finally rename your submission folder to your McGill-Id, compress it to zip format and upload it to MyCourses.

————XX————