Assignment 5

Operating System Lab (**CS341**)

Department of CSE, IIT Patna

Date:- 5-Feb-2019 **Time:-** 3 hours

Instructions:

1. All the assignments of part-I should be completed and uploaded by 5 pm. Marks will be deducted for the submissions made after 5 pm.
2. Markings will be based on the correctness and soundness of the outputs. Marks will be deducted in case of plagiarism.
3. Proper indentation and appropriate comments are mandatory.
4. You should zip all the required files and name the zip file as ***roll_no*.zip**, eg. **1501cs11.zip.**
5. Upload your assignment (**the zip file**) in the following link: https://www.dropbox.com/request/sf8OHE8sq1NbygTOSOsb

1. In this assignment, you implement ***diners philosophers problem***. Here each philosopher grabs the two forks one by one – first the left fork, and then after some waiting the right fork. The parent process checks at regular intervals whether a deadlock has occurred. If so, it chooses a philosopher randomly and releases the fork (***the left one actually***) grabbed by him. Maintain a ***resource graph*** using shared memory. The parent process periodically checks for a deadlock (cycle) in the shared resource graph. Use ***semaphores*** for synchronization and mutual exclusion.

In both the programs, print suitable diagnostic messages, like the following:

Philosopher 3 starts thinking
Philosopher 2 starts eating
Philosopher 0 grabs fork 0 (left)
Philosopher 4 ends eating and releases forks 4 (left) and 0 (right)
Parent detects deadlock, going to initiate recovery
Parent preempts Philosopher 1

Also for each step, print the ***allocation matrix*** and ***request matrix*** for each process.

2. In this assignment, you implement a **semaphore-based** solution to the **bounded buffer producer/consumer** problem.

The buffer is manipulated with two functions, ***insert_item()*** and ***remove_item()***, which are called by the producer and consumer threads, respectively. After you complete (by using semaphores) the ***insert_item()*** and ***remove_item()***functions, these functions will synchronize the producer and consumer threads. You must use three semaphores: *empty and full*, which count the number of empty and full slots in the buffer, and *mutex*, which is a binary (or mutual exclusion) semaphore that protects the actual insertion or removal of items in the buffer (i.e., the critical section).

The ***main()*** function initializes the buffer and creates separate producer and consumer threads. Once it has created the producer and consumer threads, the ***main()*** function will sleep for a period of time and, upon awakening, will terminate the application. The ***main()*** function is passed three parameters on the command line:

1. How long to sleep before terminating
2. The number of producer threads
3. The number of consumer threads

The producer thread alternates between sleeping for a random period of time and inserting a random integer into the buffer. The consumer thread sleeps for a random period of time and, upon awakening, attempts to remove an item from the buffer.