Assignment 5

The goal of this assignment is to solve some problems involving process co-ordination, in C using pthreads and semaphores. Grading of this project will be based on two criteria, correctness and presentation of code, and report that you will submit.

For each problem, you have to do the following in your report:

1. Identify the correctness constraints for each problem
2. Specify the conditions where wait should happen and justify your reasons for them.
3. Also specify, when your algorithm can lead to starvation, deadlock or race condition etc, and if not why
4. Implement the solution using semaphores only.

Problem 1 Life on a Chip

In future, water will be a scarce resource and most of the humans will have to generate water on their own.
For this purpose, a chip was developed that will combine Hydrogen and Oxygen to make water. This chip
has *n* sites where the reaction will take place, and each reaction will generate 1 E Mj of energy. For safety
purpose, no two consecutive sites should be used for the reaction, and total energy generation at any moment
should not exceed than some threshold value. As for reaction, each hydrogen atom must combine with another hydrogen atom and an oxygen atom at a site.

For the purpose of this problem, each atom can be assumed to be a thread, which will be in sleep state and at every state of reaction you have to wake up the corresponding threads (atom) at particular site, and join them. Assume that a reaction take sleep (3) time to complete.

Your task is to write an algorithm using semaphores, which efficiently uses atoms, and gives a faster way to generate water.

Input:
./a.out <No of H atoms> <No of O atoms> <No of sites> <Threshold Energy(in Units)>
./a.out 12 6 3 2

Output:
Write lines, such that ith line represents a reaction giving the details of site and total energy generated. Each line also specifies number of H and O atoms left (indexing from 1 to m).

Problem 2 GHAC Old Bridge Trip

In Hyderabad, GHAC organizes Trek and other adventurous stuff for people. However, recently there are
many geek people who have tried on these adventure trips. As a result there had been two major classification
of people, **geeks and non-geeks**. In a particular trek, where they have to cross an old bridge which can allow maximum 4 people, with the condition that bridge can only hold weight at one point i.e all 4 of them have to be together on bridge and it can only be used one at a time; a next pass on bridge should only happen when
previous pass is complete. All other combination are safe. Furthermore, this bridge is located near to the
village of singers. So Trip Leader decided to invite some singers on the trip to bridge.

In order to maintain the spirit of adventure, you cannot put three geeks with one non-geek or vice-versa.
But if boarding group has a singer, then he can calm down even unbalanced group. But you can never allow
two singers on same pass, as they have jealousy issues. Two procedures are needed, *GeekArrives()* and *NonGeekArrives()*, called by a geek or non-geek when he/she arrives at the end of bridge. The procedures
arrange the arriving geeks and nongeeks into safe bridgeloads. To get on the bridge, a thread calls
BoardBridge(); once the bridge is at its full capacity, *one* thread calls GoBridge(). GoBridge() does not return
until the members have left the end. Assume BoardBridge() and GoBridge() are already written. Implement GeekArrives() and NonGeekArrives(). These methods should not return until after GoBridge() has been called
for the bridgeload. Any order is acceptable (again, don't worry about starvation), and there should be no
busy-waiting and no undue waiting (geeks and nongeeks should not wait if there are enough of them for a safe bridgeload).

Input:
./a.out <#of geeks> <#of non-geeks> <#of singers>
./a.out 12 15 3

Output:
Lines giving details about transactions of the 4 people that are on a bridge.

Problem 3 Courses Allocation Algorithm

LSR College offers a great deal of varied courses to its students. However, allocation, this time was a mayhem, and for that reason their administrative staff approached ICS231 to get this problem solved. Before we get into details, here is the structure of any course in LSR.

- Each course allows up to 60 students to enroll.
- Each course belongs to one of 4 group, also known as *Knowledge Spectrum*. For eg. Commerce,
  Humanities, Management, Arts etc.
- Each course allows a certain quota to respective branch of students. For eg. M.Com, PHD, B.Com etc.
  (Here Quota defines maximum number of people from each branch that can enroll to any course)

Students too have a structure.

- Each student belongs to one of four branch, i.e M.Com, PHD, B.Com and Arts.
- Each student can take exactly 4 courses in one semester.
- Each student is allowed to make a list of preferences to the courses they like, such that size of list is 8.

Now Administrative head of LSR have asked us to impose certain criterias

1. Each student must pick at least one course from each Knowledge spectrum, i.e Commerce, Arts etc.
2. Each student should get 4 courses from the preference of 8.
3. Quota for each branch is in ratio *1:1:2:1* i.e out of 60 students, 12 belong to M.com, 12 to PHD, 24 to B.com
   and 12 to Arts

Your task is to write an suitable algorithm, which maintains certain data-structure for the process of allocation.
For eg. Queues. You can think of a distributed or centralized approach to this problem. Both approaches will work, and have their pits and falls. In case, of starvation, i.e when some student doesn't get their 4 opted courses, you have to report that student. You can generate preference on random, and #of courses can be equally divided into 4 knowledge spectrum. Branch capacity can be determined by the quota mentioned above

Input
./a.out <#of students> <#of courses>

Output
Report the students who got left.(*Print it to stdout*)
Write allocation in a file name *allocation.txt*, with each course number containing number of actual students who enrolled into it.

Deadline: 26th October, Saturday, 5:00PM

Upload Format: .tar.gz

Create a folder named your roll number. Copy all the source files and report in this folder. Create a tar.gz named “Assignment5.tar.gz” and upload it.