Problems on Process Co-ordination

Project No: 7

Team Name: Triple Thread

Team Members and their contribution:

Roll No	Name	Contribution
2022201073	Vedashree Ranade	Q2 - GHAC Old Bridge Trip
2022201079	Priyanshu Bansal	Q3 - Courses Allocation Algorithm
2022202020	Shivani Sinha	Q1 - Life on a chip
201050013	Kiran Kumar Gowrarajula	Q3 - Courses Allocation Algorithm

Mentor: Ira Tyagi

Instructor: Manish Shrivastava

GitHub Repository Link: https://github.com/priyanshu988/Process_Cordination_7.git

Introduction: Process coordination or concurrency control deals with mutual exclusion and synchronization. Mutual exclusion ensures that two concurrent activities do not access shared data (resource) at the same time and critical region is executed by only one process. Synchronization refers to using a condition to coordinate the actions of concurrent activities. Implementation of solutions to the given 3 problems on process coordination requires use of pthreads for process synchronization.

Problem 1: Life on a Chip

Problem Description:

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.

Solution:

- Number of hydrogens, oxygen, threshold energy and sites are given as input in command line.
- The total number of usable hydrogen, oxygen and operable sites is calculated from given input.
- A thread is created for each operable site, which will create 3 more threads i.e for oxygen atom and two more for hydrogen atom
- Now each thread will call a function for updating the number of hydrogen and oxygen items using pthread_mutex_lock on both hydrogen and oxygen items as required to ensure mutual exclusion.
- The formation of H2O takes sleep(3) time with a restriction that no two consecutive sites should be used for the reaction.

Implementation Details:

1. Thread implementation:
    - pthread_t: used to create a thread for each of the operable sites and each hydrogen and oxygen atom.
    - pthread_mutex_t: used to lock and unlock in order to ensure mutual exclusion.
2. Function implementation:
    - reaction(): This function is used to create 3 threads i.e one for oxygen and two for hydrogen, and those threads will call a function for updating the number of hydrogen and oxygen atoms.
    - update_Oxygen(): It is called by an oxygen atom thread to update the count of oxygen atoms when they participate in reaction.
    - update_Hydrogen(): It is called by a hydrogen atom thread to update the count of hydrogen atoms when they participate in reaction.
3. Deadlock / Starvation / Race-condition: Number of usable hydrogen and oxygen atoms are pre-computed, which might have led to deadlock or starvation.

Result:

H2O formation will take sleep(3), utilizing the given hydrogen and oxygen atoms optimally, The final output will print Site Number, Energy, remaining hydrogen and remaining oxygen.

Problem 2 : GHAC Old Bridge Trip

Problem Description:

A group of geeks and non-geeks have to cross an old bridge that allows a maximum of 4 people in each pass. In addition to this, three geeks with one non-geek or vice-versa can not be put together in one group. But if a boarding group includes a singer, he can calm down an unbalanced group. There is an additional constraint that no 2 singers can be present in the same pass. The task is to design an algorithm that arranges the arriving geeks and non geeks into safe bridge loads in an efficient manner.

Solution:

- The problem is divided into different cases and threads of geeks, non-geeks and singers are created accordingly, which will in turn call GeekArrives(), NonGeekArrives() and SingerArrives() respectively.
- The cases are:

Case 1: #singers = 0 and (#geeks != 0 or #non geeks !=0)

Case 1.1: #geeks >= 4
Case 1.2: #non geeks >= 4
Case 1.3: #geeks = 3 and (#non geeks = 2 or #non geeks = 3)
Case 1.4: #geeks = 2 and (#non geeks = 2 or #non geeks = 3)

Case 2: #singers > 0 and (#geeks != 0 or #non geeks !=0)

Case 2.1: #geeks >= 3
Case 2.2: #non geeks >= 3
Case 2.3: #geeks = 1 and #non geeks = 2
Case 2.4: #geeks = 2 and #non geeks = 1

Implementation Details:

1. Thread Implementation:

pthread: to create thread for each geek, non-geek and singer

2. Functions:

- GeekArrives(): If constraints to cross the bridge are satisfied, updates the count of geeks and calls crossed_bridge().
- NonGeekArrives(): If constraints to cross the bridge are satisfied, updates the count of non geeks and calls crossed_bridge().
- SingerArrives() : If constraints to cross the bridge are satisfied, updates the count of singers and calls crossed_bridge().

3. Deadlock / Starvation / Race-condition:

If any of the conditions at any point of time is not satisfied, then threads are being closed.

4.Logic details:

Singers are given priority to form groups so that any possibilities of unbalanced groups can be handled first. Geeks are chosen after singers while forming groups. After all the possible groups are formed and crossed the bridge, the count of remaining people who could not cross the bridge is also displayed at the end in output.

Result:

The bridge is crossed by the boarding group in a number of passes in an optimal manner, giving information of the category of people (i.e. geeks / non-geeks / singers) in that particular pass.

Problem 3: Courses Allocation Algorithm

Problem Description:

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 upto 60 students to enroll.
- Each course belongs to one of 4 groups, also known as Knowledge Spectrum. For eg. Commerce, Humanities, Management, Arts etc.
- Each course allows a certain quota to each 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 branches, 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 the size of the list is 8.

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

- Each student must pick at least one course from each Knowledge spectrum, i.e Commerce,Arts etc.
- Each student should get 4 courses from the preference of 8.
- 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

Solution:

● Initially the total number of students allowed in each knowledge spectrum are calculated.
● Thread is created for each student to determine the courses that will be allocated to him/her.
● Created a course preference list for each student according to the given constraints.
● The allocation array is created and populated in such a way that each student has courses from each of the knowledge spectrum and each course has a quota of students in ratio of 1:1:2:1 from the respective streams.
● If all criteria for course allocation of a student is fulfilled then he/she will be allocated the 4 courses.
● Then the total number of seats left in the streams(and courses) are updated for further allocation.
● Information about the remaining seats of every stream is hence written into allocation.txt
● Else if any of the criteria for course allocation of a student is not fulfilled, then student is not allocated a seat.
● At the end, the total number of students, who are not allocated seats are displayed.

Implementation Details:

1. Data structures used:
    - bitmap - to ensure the ratio of student category
    - vector - to store the courses for each knowledge stream
    - array - to store preference list and allocation list

2.Thread Implementation:
    - pthreads_t - to create a thread for each student.

3.Functions:
    - studentAllocation() : allocates courses to students, if given constraints are satisfied.
    - printToFile() : Writes information about the remaining seats of every stream is into the allocation.txt

4. Deadlock / Starvation / Race-condition: Proper thread synchronization using locks or atomic variables to prevents race conditions

Result:

The courses are allocated to students, by following the constraints and the total number of students, who are not allocated seats are displayed. Information about the remaining seats of every stream is provided in the allocation.txt.