CS-431 | PROGRAMMING LANGUAGES LAB REPORT
ASSIGNMENT-1
VAKUL GUPTA | 170101076

NOTE- Each Problem Folder contains the corresponding **README** file. Check that for compilation and execution.

PROBLEM 1 - Sock Matching Robot

a) The role of concurrency and synchronization in the above system

Concurrency -

- There are multiple robotic arms available in the system and they are executing concurrently to pick a sock from the heap and pass it to the matching machine.
- Similarly, the matching machine and the shelf manager robot are also working simultaneously with the robot arms where the matching machine picks up a sock from the robotic arm, matches it with the same coloured sock, and passes it to the shelf manager robot, which further increments the sock pair count of the corresponding colour.
- As multiple threads are performing the work concurrently, thus it increases the system throughput, and reduces the waiting time.

Synchronization -

- The heap of socks can be accessed simultaneously by all the available robotic arms but no two robotic arms should pick the same sock. Hence picking up a sock by a robotic arm has been synchronized.
- Matching of the socks present in the matching machine's buffer and addition of a new sock in the matching machine's buffer has also been done synchronously.
- Arrangement of the pair of socks from the shelf manager's buffer to the correct shelf and addition of a new sock (thus incrementing the sock pair count) to the shelf manager's buffer has also been done synchronously.

b) How did you handle it?

Concurrency -

Concurrency is achieved by multithreading. A thread is created for each robotic arm and is executed concurrently. (The class RoboArm extends the thread so that multiple instances of it can be run concurrently)

Synchronization -

Synchronization for accessing the socks by the multiple robotic arms is done by using **semaphores** . A semaphore controls access to a shared resource through the use of a counter.

It allows the access when the counter is greater than zero and denies if the counter is zero. Thus, to access the resource, a thread must be granted a permit from the semaphore. Here socks are the shared resources and we have used individual semaphores to lock each of the sock and set the semaphore counter value equal to 1 i.e. each sock can be picked up by only one robotic arm.

PROBLEM 2 - Data Modification in Distributed System

a) Why concurrency is important here?

Since CC, TA1, and TA2 act independently, they can simultaneously update the marks of the students. In absence of concurrency, while one of them is updating the marks of any one student, others have to wait for even if they want to update the marks of some other student which can be done independently. So concurrency is required for allowing them to concurrently update the marks of different students simultaneously. Thus, concurrency would speed up the process and would result in increased throughput and reduced waiting times.

b) What are the shared resources?

The input file “Stud_Info.txt” containing the student records (in which marks are present at a particular field) are the shared resources here. During row level updation, the marks of the individual students are needed to be handled carefully.

c) What may happen if synchronization is not taken care of? Give examples.

If synchronization is not taken care of, some updation of marks may happen incorrectly because of independent threads for CC, TA1, and TA2.

EXAMPLE -
RECORD LEVEL UPDATION -

Suppose A’s initial marks = 40 and there are two queries

Query-1: TA1 increasing A’s marks by 10
Query-2: TA2 increasing A’s marks by 20

If both Queries are executed simultaneously then there would be a race condition leading to data inconsistency.

TA1 will see the initial marks of A’s to be 40 and increase it by 10. Now TA2 will also see the initial marks of A’s to be 10 as TA1 haven’t yet written back the updated marks. And TA2 increments A’s marks by 20. Both TA1 and TA2 write back their updated marks for the student.

Finally A's marks in the records will be 60 (TA2 writes back the updated marks after TA1) , but actually it should be 70. So the Query-1 execution hasn't affected the A's marks. If TA1 updates the marks later on, then the output would be 50 which is also incorrect.

FILE LEVEL UPDATION -

In this case, an entity might view the dirty record because multiple entities have opened a cached copy of the same file simultaneously.
For example, let there be two records: A1 with marks 20, and A2 with marks 15.
TA1 wants to update the marks of A1 to 7 and TA2 wants to update the marks of A2 to 19. Both open a cached copy of the same file simultaneously. Now, depending on the race condition, let's say TA1's cache is written back to the original file after TA2, it can overwrite the updated marks by TA1. The result would be A1 with marks 7, and A2 with marks 15 (which is incorrect).

Hence synchronization is required here to maintain correct results.

d) How you handled concurrency and synchronization?

Concurrency -

Concurrency is achieved by multithreading. Individual threads are created for TA1, TA2 and CC and are being executed concurrently for updating the marks of the students. (The class Instructor extends the thread so that multiple instances of it can be run concurrently)

Synchronization -

I have used **reentrantlock** for synchronization. A thread has to acquire a reentrantlock corresponding to the record of the student it wants to modify. This ensures that there is only one thread modifying a particular student's record.

WORKING - ReentrantLock allows threads to enter into a lock on a resource more than once. When the thread first enters into lock, a counter is set to one. Before unlocking, the thread can re-enter into lock again and every time counter is incremented by one. For every unlock request, counter is decremented by one and when counter count is 0, the resource is unlocked.
