CS431: Programming Languages Laboratory

Assignment 1: Concurrent Programming

Anirudh Sharma (150101006)

Saket Sanjay Agrawal(150101054)

QUESTION 1:

a) **What role concurrency plays here?**
Concurrency help multiple arms to pick the socks from the pile at same time without waiting for other to finish. For each machine arm we are creating a thread that will pick a sock from the pile and give it to matching machine. With the help of concurrency multiple arms can work at same time.

b) **Do we need to bother about synchronization? Why? Illustrate with example.**

Yes ,we need to insure that no two robotic arms are trying to pick the same sock. Otherwise none of them will be able to pick it and deadlock situation might happen. For that we need synchronization.

c) **How you handled both ?**

- **Concurrency** : Is handled by creating thread for each order.
- **Synchronization :** With each sock, we are associating a permanent lock and synchronization lock with it. If some arm is trying to pick the sock, it will first check the permanent lock status, if it is 0, it will acquire the lock(make it 1) and the value of this lock will persist permanently since a sock when picked up can never be picked up again. To insure that no two arms are picking the same sock, we have sync lock. When a thread is picking a sock, it will need to first acquire this lock and when done will release it.

QUESTION 2:

a) **What role concurrency plays here?**

Concurrency helps in modifying the record of a student even when someone else is modifying the record. It will help the modifier(TAs or CC)

to update the marks of any student (given currently no one is modifying the marks of the same student) without waiting in a queue for its turn. For each request (to modify the mark of a given student by TAs or CC), we are creating a thread.

b) Do we need to bother about synchronization? Why? Illustrate with example.

Yes, as for each request to modify marks, a new thread is created .This threads if not synchronized will update marks of any given student in undetermined manner. Synchronization makes updating marks atomic hence preventing multiple modifiers (TAs or CC) to update the marks at same time leading to inconsistency .

For example, let's say TA1 and TA2 wants to update the marks of any given student at same time:

Let say the current marks of a given student is 75.

TA1 wants to increase the marks by 5 and TA2 wants to decrease the marks by 3.

Now if both of them simultaneously update the marks then final marks can be 72,80 or 77 depending upon what marks each thread read and who updates the marks finally. While with synchronization the final marks will be definitely 77.

c) How you handled both ?

- **Concurrency** : Is handled by creating thread for each order.
- **Synchronization :** We associate a semaphore with each student record. Whenever a modifier(TAs or CC) wants to update the marks of any given student, the thread will acquire the semaphore for that student and after updating it will release the semaphore hence ensuring that no two thread is accessing (or modifying)the student record at same time.

QUESTION 3:

a) What role concurrency plays here?

Concurrency help in taking orders even during time-event in which a previous order is being processed (Cooked or delivered). Concurrency

enables to decide whether orders could not be completed because of shortage of food products. For each customer visiting platform, a new thread is created which takes their order. Hence concurrency is needed to give a real time experience to customers.

b) Do we need to bother about synchronization? Why? Illustrate with example.

Yes, as for each customer visiting , a new thread is created .These threads if not synchronized will update values of snacks and cookies in undetermined manner. synchronization makes updating stock atomic hence preventing multiple customer editing at same time leading to inconsistency .

For example, let's say t1 and t2 be two threads which update values of food items present . If update step is not atomic :

t1 calls -> update (snacks:4 , cookies:5) // snacks and cookies present at when t1 tried to update .

t2 calls -> update(snacks:4 , cookies:5) //as t1 has not made any update yet therefore t2 also get same value of snacks and cookies .

t1 orders(snacks 3 , cookies 4) -> remaining (snacks 1 , cookies 1)

t2 orders (snacks 3 , cookies 3) -> remaining (snacks 1 , cookies 2) // as value of items didn't update in t2 thread . this created problem as t2 must be able to make order due to lack of quantity .

c) How you handled both ?

- **Concurrency** : Is handled by creating thread for each order.
- **Synchronization :** Before calling Order function it is defined as synchronized hence only one thread will be able to make order at a time , in other words by synchronized only single thread at a time is given write to update values of food item hence making an order.