CS 431
Programming Languages Lab
Assignment 1

Chirag Gupta
170101019

1) Sock Matching Robot

a) Role of concurrency and synchronization

Concurrency

- As per the problem statement, many robotic arms work together and concurrently to pick up a sock from the heap of socks and pass it to the matching machine.
- Now the matching machine segregates the sock of similar colour and passes it to the appropriate colour shelf. The matching machine picks a sock from the robotic arm and pairs them with the same coloured sock and passes it to shelf manager.
- The shelf manager robot increments the count of the colour of which sock pair belonged.
- All this results in lower waiting time for execution of the program as multiple things can be done at the same time with help of multi-threading ability of the CPU.

Synchronization

- Now as the heap of socks(shared data) is accessible to all the robotic arms, it is highly possible that two robotic arms want to pick the same sock at the same time.
- To contour this, synchronization must be integrated into our code to ensure that a sock is picked by only a single robotic arm, and if another one tries to pick that, it should be unable to pick.
- The pairing of socks and incrementing the count of pairs of individual coloured socks must be done in a synchronous manner.
- If synchronization is not integrated, then we may have the case of data incoherency. Let's say if currently, total pairs of socks are 5, and the shelf manager receives two pairs of socks. Then ideally final answer should be 7, but if for both the pairs, value is incremented before the previous increment is complete, then the final answer would be 6.

b) How did you handle it?

- **Concurrency** is achieved with help of **multi-threading**. The Robotic Arm Class extends the Thread class and multiple robotic arms are running multiple threads simultaneously.
- **Synchronization** for picking a single sock is implemented by using **method synchronization** on a Sock object so that only one Robotic Arm can pick up a sock at a time.

2) Data Modification in a Distributed System

a) Why concurrency is important?

- Here all the three teachers (TA1, TA2 and CC) are independent of each other and hence can simultaneously update the marks of the students.
- If concurrency is not implemented, then if one is updating the marks of a particular student, then **others need to wait.**
- But with concurrency, the three **teachers can simultaneously update** the marks of different students to speed up the execution process.
- In nutshell, concurrency helps to execute many requests simultaneously thus **reducing waiting time** and **increase in overall efficiency.**

b) What are the shared resources?

- **The input files contain the record of each student** and this particular file is being shared among all the clients(Teachers). The record of each student forms the shared resource during row-level updating in the file and hence needs to be handled carefully.

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

Record-level update

- Here two entities can **update the same record simultaneously** and hence resulting in data corruption. This all could lead to **race-condition**.

- Let's say TA1 wants to increase marks by 7 while TA2 wants to decrease it by 4 marks. Initially, both view the marks in shared resources as 20 marks. Now let's say TA1 increase marks to 27

but before writing it to file, TA2 also decreases the mark to 16. Now let's say the first TA1 writes 27 marks to a shared file, but after few moments TA2 updates it to 16. So the final answer present in the file would be 16 (which is incorrect as it should be 20 + 7 - 4 = 23 marks ideally)

File-level update

- Here the issue is that because multiple entities have opened a **cached copy of the same shared** file simultaneously, the race-condition on who writes it back first might lead to corrupted data.
- Let's say TA1 wants to update marks of student A and TA2 wants to update marks of student B and both of the clients have opened the cache copy of shared file together and updated the marks in a cached copy.
- Now while writing back it to a real copy, let's say TA2 writes it first and hence marks of student B have been updated in the database but not for student A till now. Now when the CPU is given to TA1, he also copies his cached copy to the shared file, which would lead to updated marks of student A but not of student B.
- Finally, we have updated the marks of student A but not of student A which is a clear example of a **corrupt shared resource**.

d) How you handled concurrency and synchronization?

- **Concurrency** is handle by **multithreading** where a single thread is created for each teacher (CC, TA1 and TA2) and these threads are being executed simultaneously to update the marks.
- For **synchronization** and locks, I have used **ReentrantLock**. This allows threads to enter into a lock on a resource more than once. When the thread first enters into the lock, a hold count is set to one. Before unlocking, the thread can re-enter into the lock again and every time hold count is increased by one.
- For every unlocks request, hold count is decreased by one and when hold count becomes 0, the resource is unlocked.
- Each unique roll number has its own lock. A thread(client) has to acquire the lock corresponding to the roll number of the student if he/she wants to update the marks of that particular student.
- This ensures that any important updates are **made only by a single thread** at any time till the lock is removed.