Lavish Gulati - 170101082

CS431 - Programming Languages Lab - Assignment 1

1. Sock Matching Robot

a. Role of concurrency and synchronization

Concurrency:

i. There are several robotic arms working simultaneously to pick a sock from the heap and pass it to the matching machine.

ii. The matching machine and shelf manager robot are two other entities which are also working simultaneously with each other and with all the robotic arms. The matching machine picks a sock from the robotic arm and pairs them with matching coloured sock and passes it to the shelf manager robot. The shelf manager robot increments the count of the colour of which the sock pair belonged.

iii. This results in reduced waiting time for the execution of the program and increased throughput or resource utilization in terms of multi-processing cores in the CPU.

Synchronization:

i. The heap of socks is accessed simultaneously by all robotic arms which can lead to two robotic arms picking up the same sock. Thus, synchronization is important to ensure that a sock is picked by only one robotic arm at a single point of time.

ii. The pairing of the socks by the matching machine and incrementing the count of pairs by the shelf manager robot also needs to be done synchronously (or else it would create data incoherency issues). For example, if the current number of white pairs is 10, and shelf manager robot receives two pairs of socks in separate instances, the final answer could be 11 when the second instance reads the input as 10. So synchronization is important to ensure increment is done by one instance at a time.

b. How did you handle it?

Concurrency: Concurrency is achieved by multi-threading. The Robotic Arm class extends the Thread class and multiple instances of robotic arms are executed as multiple threads running concurrently.
Synchronization: Synchronization for picking socks by the robotic arms is handled by the construct Semaphore. A semaphore controls access to a shared resource through the use of a counter. If the counter is greater than zero, then access is allowed. If it is zero, then access is denied. What the counter is counting are permits that allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore. In our case, socks at each index form the shared resource and Semaphore with counter value 1 is used for each individual sock.

2. Data Modification in Distributed System

a. Importance of concurrency

CC, TA1 and TA2 are independent entities and can update the record simultaneously. Concurrency is important in such distributed database or system because multiple clients can send read/write requests to the program simultaneously. In absence of concurrency, only one request is handled at a time which can lead to increased waiting time for each client. Concurrency allows many requests to be executed simultaneously thus reducing waiting time and increasing throughput.

b. What are the shared resources?

The input files containing the record of each student are the shared resources in this case. The record of each student forms the shared resource during row-level updation and needs to be handled carefully.

c. Importance of synchronization

i. **Record level updation:** In this case, two entities can update the same record simultaneously resulting in data incoherency.
For example, let there be a record with Roll Number 170101082 and marks 10.
The program receives two queries: Q1 - TA1 wants to increase marks by 5 and Q2 - TA2 wants to decrease marks by 3. If both queries are received simultaneously, it could lead to a race condition where:
Case 1: Both TA1 and TA2 view the marks to be 10. TA1 increases marks to 15 and writes it to the file (where TA2 has not completed the operation yet). TA2 decreases the marks to 7 and writes to the file (over-writing the updated marks by TA1). The final output would be 7 marks (which is incorrect).
Case 2: Both TA1 and TA2 view the marks to be 10. TA2 decreases marks to 7 and writes it to the file (where TA1 has not completed the operation yet). TA1 increases the marks to 15 and writes to the file (over-writing the updated marks by TA2). The final output would be 15 marks (which is incorrect).

ii. **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: Roll Number 170101082 with marks 10, and Roll Number 170101081 with marks 15.
TA1 wants to update the marks of Roll Number 170101082 to 7 and TA2 wants to update the marks of Roll Number 170101081 to 19. Both open a cached copy of the same file simultaneously.
Depending on the race condition, let's say TA2's cache is written back to the original file after TA1, it can overwrite the updated marks by TA1. The result would be Roll Number 170101082 with marks 10, and Roll Number 170101081 with marks 19 (which is incorrect).

d. How you handled concurrency and synchronization?

Concurrency: Concurrency is achieved by multi-threading. A separate thread is created for each Teacher instance (CC, TA1, TA2) using the Thread class extension and executed concurrently to update

entries.

Synchronization: Synchronization for marks updation is handled by the construct Reentrant Lock.
The ReentrantLock class implements the Lock interface and provides synchronization to methods while accessing shared resources. ReentrantLock allows threads to enter into a lock on a resource more than once. When the thread first enters into lock, a hold count is set to one. Before unlocking the thread can re-enter into lock again and every time hold count is incremented by one. For every unlock request, hold count is decremented by one and when hold count is 0, the resource is unlocked.
In our case, lock is acquired for each student entry while updating it to ensure synchronization.

3. Calculator for Differently Abled Persons

a. Using single SwingWorker thread

Since we only need to input a single digit before moving onto the function area and vice versa, there is no requirement for multi-threading. Hence, a single SwingWorker thread is executed which highlights the area specified by the variable ‘stage’ (0 means number area and 1 means function area), and switches the value after a single character is inputted (or differently in special cases ‘evaluate’ and ‘clear’).

b. Using two SwingWorker threads

Since we need to form multi-digit numbers, the user can input a digit or operator at any point of the execution. So we need two SwingWorker threads: number scanner handles the scanning of input in the number area, and function scanner handles the scanning of input in the function area. Both worker threads are executed simultaneously.