Programming Languages Lab

Group members:-

Udayraj Deshmukh
150101021

Anurag Ramteke
150101010

Problem 1: Sock Matching

The role of concurrency and synchronization in the system

Since the task was to **simulate** the sock sorting system, we need concurrent programming. The role of concurrency here is to simulate arms picking up socks from the heap of socks, a matching machine finding the socks of same color and a shelf manager robot to organize them in the shelf.

Synchronization is needed in the system to ensure correct execution of these parallelly operating machines. The arms should not pass sock to the matching machine unless it is ready to accept it. Similarly, matching machine should pass sock pair only when shelf manager has done processing previously passed pair.

How we handled it?

We divided the problem into two synchronization subproblems. Both are treated similar to the producer-consumer problem.

In first subproblem, the arms are the producers and matching machine is the consumer. To allow the "passing" of socks from robotic arms to matching machine, we allotted a **shared queue** in which the arms may put the socks in and matching machine may pick up from it.

In second subproblem, the matching machine becomes producer and shelf manager is the consumer. To let matching machine pass the pair of socks to shelf manager, they use a **shared variable** to contain color of matched pair.

In java terms:

Arm ← [sharedQueue]→ MatchingMachine ← [matchedColor] → ShelfManager

We created shared variables "sharedQueue" and "matchedColor" on which the **synchronized** keyword is used appropriately to make sure no machines try to read/write on the variables when there is nothing to read/write from.

The Arms call .**wait**() on the sharedQueue when it is full.

The MatchingMachine waits while the sharedQueue is empty.

Once matching machine finds a color pair, it writes it on the matchedColor and calls

notifyAll() to notify the ShelfManager. The ShelfManager reads the value and calls notifyAll to inform MatchingMachine that the variable is free to write on.

Problem 2: Data Modification in Distributed System

Why concurrency is important here?
As the requirement of this problem statement is that the file can be accessed and updated by TA1, TA2 or CC simultaneously, we require concurrency. And to allow update of same record by multiple people simultaneously, we also need synchronization to ensure correct updates.

Shared resources used:
The shared resources in this problem are between the threads CC, TA1 and TA2.
Another shared resource which will be used by them will be the **vector of records**. There will be have an **individual lock for each record** which is also shared across the threads**.** When one of them is using to modify that share resource and other will wait till the thread which has acquired the lock frees the shared resource.

What may happen if synchronization is not taken care of? Give examples.
If Synchronization is not taken care of then multiple threads access the shared resources at the same time and modify it at the same time. This will create havoc in the database, if TA1 and TA2 are trying to modify the same record, only the most recent modification will be reflected. Thus causing incorrect storage of records.

Examples:

1) *Let's say we have a person with roll number 174101055 with marks 75. Now, say TA1 wants to increase his marks by 3 and TA2 wants to decrease his marks by 1. So, now TA1 reads the his score to be 75 and TA2 also reads his score to be 75. Now, TA1 increases his marks and his marks becomes 78 and TA2 decreases his marks and his marks becomes 74. Now, ideally, his marks should be 77. But lets say TA1 writes the data to the file first and then TA2 writes so the data reflected will be 74. If TA1 writes data to the file later, then data reflected will be 77. The only condition when 77 will be reflected in the database will be when one of TA reads the data first and writes and then another one reads and then writes. At that time marks reflected will be 77 as expected. However, without synchronization this outcome is highly unlikely.*
2) *Let's say we have another scenario with same roll number and with marks 75. Now, CC wants to increase his marks by 5 and TA1 wants to decrease his marks by 2. So, CC will have access to the data first once CC access the data and modify it TA1 should not be able to do anything modification with that record. So, ideally students marks should be reflected 80. However, situation may happen that instead of CC first accessing and then TA1. Due to not synchronization applied CC and TA1 both accessed the data and read 75. CC will write 80 and TA1 will write 74 and the one which wrote later will be reflected in the database. If CC does it later then the modified value is expected otherwise we get the wrong one. TA1 access it first and*

then change the value and then CC reads it then the value will be 79 which is also unexpected.

How you handled concurrency and synchronization?
Concurrency is used in file level modification and synchronization is used in record level modification. When file level modification is done two records are used against two threads and both are locked for their respective thread and then modified here concurrency is used. When record level modification is done one record is modified by two threads both read the record at the same time using read lock and then one of them enters the write lock and other stays waiting until the write lock is released from that record. The database is updated by the modified value of the first thread and the second thread read value is updated accordingly before entering into the write lock and then its performs its modify operation and write it to file and releases the lock.

Problem 3: Room Delivery Service of Tea/Snacks.

Assumptions: For calculation of expected delivery times, we assumed that there is only 1 tea machine and 1 coffee machine(the two can run parallely) and that the delivery time is fixed for each order.

Role of concurrency:
Since here we are creating a single-server, multiple-clients mechanism, there's a need of doing handling the connections parallelly. To accept and process multiple orders simultaneously, the system needs concurrency. On the client side also, the GUI implementation allows many events(like mouse move and keyboard input) to happen simultaneously, thus internally making use of concurrency.

Do we need to bother about synchronization? Why? Illustrate with example.
Yes. Because without proper synchronization, the stock quantities as well as expected delivery times will not get updated properly, leading to incorrect order invoices and causing losses to the vendor which is highly undesirable.
This can be illustrated using following scenario.
Say Customers C1, C2 are connected to the server in threads T1 and T2 respectively. Let the available quantity for Cookies be 20.

T1: C1 sees 20 available cookies, orders 15 Cookies.
T2: C2 sees 20 available cookies, orders 15 Cookies.

If the threads are not synchronized,

T1: Order satisfied, updates cookie quantity to 20-15=5
T2: Order satisfied, updates cookie quantity to 20-15=5

Here total amount ordered is greater than total available amount. The orders for both threads will be accepted and the quantity for Cookies will remain 5 which is incorrect.

By synchronization, we ensure T2 sees updated value of available cookies before accepting an order. The expected execution is observed-

T1: Order satisfied, updates cookie quantity to 20-15=5
T2: Order not satisfied, available cookies = 5 < 15

Handling of concurrency and synchronization:
For synchronization in this problem, Java's Semaphore class was used as it offers to schedule the threads in FCFS order via one of its arguments. We used **one Semaphore for each type** of item(Tea, Coffee, Cookies and Snacks). Also, two more semaphores were used to maintain count of tea/coffees in preparation, updation of the count at fixed intervals was achieved using a thread for each - Tea and Coffee machine.

When a user places an order for a subset of these items, only those semaphores would be acquired. This way, **simultaneous orders** are possible for mutually exclusive orders.

Example 1: If at an instant of time, Customers C1, C2 and C3 place orders for 1 Tea, 1 Coffee and 1 Cookie respectively. The semaphores will get acquired individually and all three orders will get processed simultaneously. Thus providing for the **concurrency** requirement**.**

Example 2: If C1, C2 and C3 all three order for Tea and Coffee one after other (before any of them is delivered), then the semaphores would be acquired in the same order as customers thus the orders will be delivered **in FCFS manner**.