CS431 Programming Lab Assignment 1

Udbhav Chugh 170101081

Note: For each question, steps to run the code and input-output format are provided in the README.txt file of each folder.

1) Sock Matching Robot

The folder Question1 contains the code for the Sock Matching Robot. Details about steps to run the code and input and output format are provided in the README file present in the Question1 folder. Answer to the assignment questions:

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

Concurrency: The role of concurrency in the above system is as follows:

- There are multiple robotic arms in the system and they are executing concurrently. All of the robotic arms have the job of picking up a sock from the heap and passing it to the matching machine which they perform concurrently.
- Along with this, the sock matcher(to match a pair of socks) and shelf manager(to put a matched pair in the appropriate shelf) are working simultaneously with the robot arms.

Hence the robotic arms, matching machine and shelf manager are all performing their roles concurrently.

Synchronization: There role of synchronization in the above system is as follows:

- The heap of socks can be accessed simultaneously by all the available robotic arms but no two robotic arms can pick up the same sock simultaneously. Hence picking up of a sock by the robotic arms is being synchronized.
- Matching of the socks present in the matching machine's buffer and addition of a new sock in the matching machine's buffer is done synchronously. It is important that while a matching machine is trying to match, say two white socks, no other robot arms can alter any variable related to pairing by matching machine.
- Arrangement of the pair of socks from the shelf manager's buffer to the correct shelf and addition of a new sock to the shelf manager's buffer is done synchronously.

b) **How did you handle it?**

Concurrency: Concurrency is maintained by using multithreading. A thread is created for each robotic arm and each of them execute concurrently. Each thread operates independent of other threads which is just like how the Robotic arms in the system works.
Synchronization: Synchronization is handled using Reentrant lock for each sock and synchronization blocks for Machine Machine and Shelf Manager as explained below:

- I have used Reentrant lock for acquiring the lock on each sock present in the heap so that when multiple arms try to access the same sock, only one of them can acquire it and pick it. The ReentrantLock class implements the Lock interface and provides synchronization to methods while accessing shared resources. I have used individual ReentrantLock to lock each of the sock (which is a shared resource) and by acquiring the ReentrantLock, it is ensured that each sock can be picked up by only one robotic arm.

- I have used a block synchronization method to put a sock in the matching machine's buffer and similarly to put a sock in the shelf manager's buffer. This is to ensure that when Matching Machine is executing the segment of code when it finds out if it already has a particular colored sock and matches it with a new sock of the same color, the variable isSockPresent (for different colors) is not updated by multiple threads which otherwise will make the code wrong. Similarly synchronization when Shelf manager is placing a particular sock in the corresponding shelf, is needed so that the count of socks in each shelf doesn't get an incorrect value.

2) Data Modification in Distributed System:

The folder Question2 contains the code for the Data Modification in the Distributed System. Details about steps to run the code and input and output format are provided in the README file present in the Question2 folder.
Answer to the assignment questions:

I. **Why concurrency is important here?**
While we are doing file level modification the two changes made by the TA's or the CC must happen concurrently. So to do these changes concurrency is used.
There are 3 people who can access the file simultaneously: CC, TA1, TA2. They are independent of each other and they can simultaneously update the marks of the students. If concurrency is not maintained, only one of them can update marks at a particular time even when they want to update marks of separate records which is inefficient as it can be done independently using concurrency. So concurrency is required to allow simultaneous updation of the marks of different students to make the updation process efficient.

II. **What are the shared resources?**
The file Stud_Info.txt which contains the student marks is the shared resource here. A student object is made for each record which can be accessed by CC, TA1 and TA2. Hence each student record is a shared resource with the marks and teacher column being the shared resource that can be updated for each individual student record.

III. **What may happen if synchronization is not taken care of? Give examples.**
If synchronization is not taken care of, it is possible when two evaluators try to update records simultaneously, we may not get the desired result which will basically corrupt the entire students record as it can no longer be trusted.
For example consider the initial Stud_Info.txt as follows
174101055,Amit Kumar Sharma,amit55@iitg.ac.in,75,TA1,
174101012,Abdul Najim,abdul12@iitg.ac.in,29,TA2,
174101058,Kunal Kishore,Kunal58@iitg.ac.in,67,TA2,
174101033,Subhra Shivani,subhra33@iitg.ac.in,53,CC,

174101035,Savnam Khatam,savanam35@iig.ac.in,88,TA1,

Consider TA1 wants to increase the marks of 174101055 by 10 and TA2 wants to decrease the marks of 174101055 by 5.
Now the output can vary based on the following cases:

- If while TA1 is updating marks, TA2, also starts updating marks of 174101055. And hence TA1 changes data from 75 to 85 and TA2 changes data from 75 yo 70 and later if TA1 updates data before TA2, the latter value of 70 will remain.
- If while TA1 is updating marks, TA2, also starts updating marks of 174101055. And hence TA1 changes data from 75 to 85 and TA2 changes data from 75 yo 70 and later if TA2 updates data before TA1, the latter value of 85 will remain.
- If by chance TA2 threads occurs after TA1 is completed, then output can be correct(80) but this can never be ensured without synchronization.

To see the effect of asynchronous update, one can comment the part where the thread acquires Semaphore locks in updateSyncData() function and instead directly call updateData() function. In that case the output may or may not be as desired.

Only when we have synchronization such that one acquires lock of the record only after another has left, we will be able to see correct output with surety.

IV. **How you handled concurrency and synchronization? (You must use a different synchronization technique than you used in Q1)?**
Concurrency is maintained by using multithreading. A thread is created for each of CC, TA1, and TA2 and each of them execute concurrently. Each thread completes the tasks assigned to each particular evaluator simultaneously.
Synchronization:

- I have used **Semaphore locks** for maintaining synchronization(record level) in the student record updation. A semaphore controls access to a shared resource through the use of a counter. I have used individual semaphores for each of the record (which is a shared resource) to ensure synchronous update.
- Whenever an evaluator wants to update a particular record, it must acquire the corresponding semaphore lock associated with that particular record. This ensures that only one evaluator is updating that record at a time to ensure synchronization.
- And since it is possible the evaluators are updating different records, the program waits for the three threads to execute(using join function) their roles and then write to file for a synchronous file level updation.
- Also CC thread is given higher priority than TA1, and TA2 to ensure it performs updation before any of the TAs. And once CC has performed the update on a particular record, TA1 and TA2 cannot alter it since the Teacher column is updated.

3) Calculator for Differently Abled Persons

Here I have developed four different types of calculator covering the following cases:

1. 3a1_singleDigitSingleOperand contains answer of part a of ques3 with **single operator** input at a time and **single digit operands**.(Eg: 2+3).

2. 3a2_singleDigitMultipleOperands contains an answer of part a of ques3 with **multiple operators** input in a single expression and **single digit operands**. (Eg: 3+2*5-1)

3. 3b1_multipleDigitSingleOperand contains answer of part b of ques3 with **single operator** input at a time but **multi digit operands**.(Eg: 324+271)

4. 3b2_multipleDigitMultipleOperand contains answer of part b of ques3 with **multiple operator** input in a single expression and **multi digit operands**.(Eg: 32+24-11)

The steps to run each individual code are provided inside each corresponding folder. The calculators have a clear option to clear input and restart. The "Evaluate" button is provided in cases when multiple operators or multiple digit input is present clicking which evaluates the answer until that point and then the user can continue with the new answer on the screen.

Note for the cases with multiple operators in a single expression, while calculating the value, * and / are given higher precedence than and + and -. And between the same precedence, left to right evaluation is done. I have implemented this using postfix expression evaluation.

Also in case the expression is invalid (Eg: 3*/4 or 7/0), Invalid Expression is displayed on the screen.

Swing library is used for GUI. SwingWorker (which allows users to schedule the execution of background tasks on Worker Thread) is used for running threads in background to periodically change the highlighted color for both numbers and operators. **doInBackground(), publish()** and **process()** functions from SwingWorker are used for running background thread. doInBackground() contains the logic of the background task and it issues publish() which delivers intermediate results for processing on the Event Dispatch Thread. process() receives data chunks from the publish method asynchronously on the Event Dispatch Thread and uses it to change the color of the highlighted cell accordingly.

For the first two calculators, at one point only one background thread is running (since either keys are to be highlighted periodically or operators but not both).

For the last two calculators, keys thread and operators thread both run in background changing the highlight. The difference between third and fourth calculator is once an operator is chosen in the third calculator that thread is stopped since this calculator allows only one operator expression.

For the last calculator the two threads run to allow multiple digits and multiple operators input for the expression. On choosing Evaluate, the current expression is evaluated as explained above and on choosing clear, input is cleared and one can start again.