CS431 PROGRAMMING LAB

Assignment 1 : Concurrent Programming

Group 27
150101045 : Patoliya Meetkumar Krushnadas
150101072 : Shubham Singhal

Problem 1: Sock Matching

1. The role of concurrency and synchronization in the above system.

a. Concurrency

- Concurrency comes into picture in this problem since there are **n number of socks** arms that are trying to pick socks from a heap at a single time.

b. Synchronization

- **One sock to be picked by only one arm at a time :**
  - If proper synchronization is not taken care of, it might happen that two different arms try to access same sock while picking sock from heap.
  - If this is not taken care of, two picking arms might report same sock which is an impossible event.
- **Only two socks can be transferred to matching machine at a time :**
  - Since there are n number of sock picking arms but the matching machine can match only two socks at a time, synchronization is required to properly handle the action of passing socks to matching machine.
  - If this is not taken care of, more than two picking arms will try to pass the sock to matching machine and the matching machine will be overloaded (return an error).

2. How you handled it?

a. Concurrency

- n number of threads were **simultaneously created** for given no. of sock picking arms. The number of picking arms were taken input from the command line.

b. Synchronization

- **One sock to be picked by only one arm at a time :**
  - **'getSock'** method in the class Sock is defined to be 'synchronized'.
  - The available property of sock class is controlled via a **Semaphore.**
  - This ensures that when any thread tries to pick a sock(thread represent a picking arm), the process is synchronized and no two thread will try to 'getSock' at same time ensuring that one sock is picked by one arm at a time.
- **Only two socks can be transferred to matching machine at a time :**
  - '**recieveSock'** method in class MatchingMachine is defined to be 'synchronized'.
  - The available property of MatchingMachine is controlled via a **Counting Semaphore** with n=2.
  - This ensures that matching machine will receive a sock one at a time by any thread(picking arm) and hence removing the possibility of being passed more than two sock at a time.

Problem 2: Data Modification in Distributed System

1. Why concurrency is important here?

- Concurrency is making those tasks parallel whose relative ordering would not change outcome of the overall system. More concurrent the program is, less its time of execution. That's why it is important. In this problem, there are two cases:

  a. Files are being updated with synchronization:
  In this case we can update marks of two different students parallelly, since the tasks involving different students are unaffected by relative ordering. This will increase concurrency of the system, thereby making overall system faster. Note that we can not process two updates on same student's marks concurrently as this will violate synchronization.

  b. Files are being updated without synchronization:
  Here as we are not maintaining synchronization, so we can ignore relative ordering of two updates of same student's marks. Therefore, we can update all students' marks concurrently.

2. What are the shared resources?

- Individual student record is the shared resource. Updating this shared record should be handled with care.
- Overall, the file we are updating is the shared resource.

3. What may happen if synchronization is not taken care of? Give examples.

- If synchronization is not taken care of, then the relative ordering of two updates might change the overall outcome.
- For example, if marks of "student1" are incremented by 5 by TA1 followed by decrement of 3 by CC. If synchronization is taken care of will result in overall increment by 2 in "student1"'s marks.
- On the other hand, without synchronization, overall marks either incremented by 2 (TA1→CC) or decremented by 3 (CC→TA1). In second case as CC has updated the marks first then TA1 could not update them despite of the fact that he made update request before CC.

4. How you handled concurrency and synchronization?

a. Concurrency:

- When taking care of synchronization, for different students, files are concurrently updated. We have made thread for each of distinct student whose data is going to be modified and those thread are running parallelly.
- In case when we are not taking care of synchronization, we have made overall process even more concurrent by making single thread for each update.

b. Synchronization:

- We are handling synchronization by sequentially making all updates in marks of given student by one thread. We are applying **lock** to student's record while it is being updated.

- Also, before generating updated files sorted by names and roll number, we need to ensure that all threads are finished. So we are using **“CountDownLatch”** to assure that all threads are finished before file generation.

Problem 3: Room Delivery Service of Tea/Snacks.

1. What role concurrency plays here?

- We have made **multithreaded server** to have concurrent connection of multiple clients with server.
- If the server is Single Threaded, only one client can give a order at a time. The process for another client will start only once the other client thread is over and hence a unrealistic scenario to handle multiple requests.

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

- When multiple request are being handled by server, the server should ensure that the **stocks file must be accessed with proper synchronization so as to prevent the clients from giving wrong status about the availability of the items.**
- For instance, 10 clients are connected to server and try to order at same time for biscuits whose available quantity is 6 at current moment. Actually 6 orders should be accepted and rest be declined due to non-availability.
- However, if access to stocks file is not synchronized all of the threads (client connections) will see that remaining quantity is 6 and will place successful orders for all clients which is not accepted.

3. How you handled both concurrency and synchronization?

a. Multithreaded Server for Concurrency

- Socket Connections were made using the *java.net.Socket* package of java.
- Multithreaded Server spawns a new thread on every client connection it receives.
- The main thread of a multithreaded server is a while loop in ‘accepting’ state of connection.
- As a new connection is made, it spawns a new thread for handling that client.
- Client connections are as usual. It requires the server address and a port number to connect to.

b. ‘Thread.join()’ for Synchronization

- The orders are needed to be processed in **First Come First Serve (FCFS)** manner.
- Thus the thread for the order that was received earlier will access the stocks file first.
- Hence **join** method from thread class ensures that some thread starts it’s **run** method only after a thread which calls join method has finished it’s execution.