CS-431 PROGRAMMING LANGUAGES LAB ASSIGNMENT-1

-MAYANK WADHWANI (170101038)

TASK-1 SOCK MATCHING ROBOT

a) The Role of Concurrency and Synchronization in the above system

Concurrency

As far as concurrency is concerned, we would want to have our system concurrent. Since there are multiple robotic arms and a match making machine, which are all independent of each other. Like the robotic arm would simply put the sock in the correct queue of the matching machine and the machine can then run and match pairs. So if we have a greater deal of concurrency in the system, our system would be much efficient and would have a higher throughput as opposed to the non-concurrent one where first the robotic arms would keep everything on the match making queue and then the match maker would start matching pairs.

Synchronization

We would also want our system to be synchronous. Synchronous systems are those which do not suffer from race conditions. Since we do not get to choose the order of thread execution, if the system is not synchronized, we may end up in some cases where one thread was changing a global variable and gets context switched to another thread which changes this global variable to some other value and then when we come back, we have lost information. In simple words, if multiple threads operate on a shared data, it can lead to race conditions. This can be restricted by the programmer if he/she ensures that the system is synchronous (I have discussed how I achieved it in the next section).

b) How you handled it?

Concurrency

Concurrency was handled by creating multiple threads for each robotic arm and Match Making machine. So all these were executed on separate threads. As a result of which, we see high throughput in our system, when the output is generated, we see how the threads and match making machines produce output (interleaved fashion).

Synchronization

Synchronization was achieved by using the keyword **synchronized** wherever necessary. So essentially, if we declare a block to be synchronized, then only one object can execute that block at a time. So even if this thread let's say gets context switched to another thread, the new thread will not be able to enter the synchronized block since the lock is held by the earlier block.

Here the queue of socks was the main shared resource and the threads operating on it were the threads trying to acquire the lock. Also, the dictionary that contains the current amount of socks in the match making machine was the shared resource and the threads which tried to access it were the match making machine and the robotic arms.

If we go into the implementation details, what happens under the covers is that synchronized keyword is changed by the compiler to monitorEnter and monitorExit system calls. So this is implemented using a monitor. An object trying to enter the critical section will keep on spinning if the monitor lock is currently acquired by some other thread. There are many other methods such as wait or notify or notifyall which is typically used to increase efficiency. If we find that the monitor is with some other thread, we use the wait system call to delay the current execution so that we don't waste processor cycles. And then after some time, check again if the lock is available. Notify and notifyall is used by the owner of the lock to notify and wake up sleeping threads if any if they are waiting for the lock.

TASK-2 DATA MODIFICATION IN DISTRIBUTED SYSTEM

I) Why concurrency is important here?

Concurrency is important so that if I being a user has to change say Mayank's marks and then change Aman's marks. Then after giving the request to change Mayank's marks, and if the file lock is not available at that time, we shouldn't go into the blocking state. We should be taking in more inputs. So lets say the file lock is acquired by thread B and has it for 100 seconds. Now I being the user has to change Mayank's, Aman's and Manan's marks. So I don't wait after giving request to change Mayank's marks. I keep on giving requests to change everyone's marks and then after some time when the file lock becomes available, we process all the queued up requests. This shows the importance of concurrency.

Also it is important so that all the instructors (TA1, TA2, CC) can access our programs at a single time simultaneously. Moreover anyone can read the files like sorted files by name and roll number at a single time. So we create concurrent threads that achieve the same. Our data can be read by multiple people simultaneously.

II) What are the shared resources?

The given file **Stud_Info.txt** is a shared resource in our application. In other words, all the records which contain data of students are the shared resource.

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

If synchronization is not taken care of, we may end up getting race conditions in the program.

So lets consider the case when TA1 wished to increase Mayank's marks by 20 and TA2 wishes to increase the marks by 10.

Then these are the operations carried by the processes (let P1 be the process for TA1 and P2 be the process for TA2)

P1:

Marks += 20

P2:

marks += 10

In assembly language, this gets converted to:

P1:

load r1, M1

add r1,20

store r1

P1:

load r1, M1

add r1,10

store r1

Now since we have not synchronized our program, we may find ourselves in a situation where we get context switched when we just finish execution of add r1,20. In this case, P2 will dominate since the result of marks+20 was not stored (it got context switched before it could store it) and the resultant stored will be marks+10 instead of marks+30.

Mayank just lost 20 marks because the program was not synchronized!

So this is the importance of synchronization, it helps in preventing race conditions. So had the marks += val been synchronised, then the full assembly code would have been excecuted, that is the marks would have been stored in the correct place and when the other process starts, it would have read the updated marks and not the earlier one.

IV) How you handled concurrency and synchronization?

Concurrency

Concurrency was handled by simply creating multiple threads. Each thread for a query. So if want to update marks of two students, two separate threads will be created for both the queries. Moreover TA1, TA2, CC run on their respective main threads. So we have used multithreading to achieve concurrency.

Synchronization

Synchronization was achieved by using File Locks which are a part of File Channels. What happens under the covers is that when a thread takes this file lock, the others busy wait. They keep on running in an infinite while loop till they eventually get the lock. Once a thread has a lock, it can make all the changes it wants to make, like update marks of students, etc. And after making these changes, they write the data back to our file and then while terminating they release their lock giving a chance to other threads to acquire the lock and make their respective changes.
So in this manner, we achieve synchronization in our program.

END OF REPORT