CS307 PA3 REPORT

In the code, I've used 5 semaphores, 1 barrier 2 mutexes and 6 counters in total. Student semaphore is for entering and leaving conditions, demoassistant and demostudent semaphores are for forming the demo sessions and finishassistant, finishstudent semaphores are for leaving the classroom.

Before the pseudocodes, I would like to point out that I believe that in the thread implementations I have 1 while loop which could cause the busy-waiting problem. The loop is in assistant thread, specifically for the condition where the assistant has to wait for the 2 students in his/her demo session to print out I am leaving before the assistant himself/herself prints that he/she will leave the classroom. I have other while loops in the code however I've used semaphores inside those loops to prevent busy-waiting.

Assistant thread function pseudo code:

- Prints that it has entered the classroom.
- Calls sem_wait(&student) twice to indicate that it is going to allow 2 student threads to join the class.
- Initializes a boolean called waker which would check if this thread will be the waker of the other 2 students for the demo sessions.
- Locks a mutex, increases the enterAssistants count by 1. (enterAssistants represent the amount of assistants that are inside the room.)
- Checks if there are >= 2 students that are waiting to form a demo group.
- If so, changes waker to true, wakes up 2 demostudent semaphores, decreases demoStudents count by 2. (DemoStudents count represents the students that are currently waiting to join a demo session.)
- Else, it changes waker to false, increases the demoAssistants count by 1, releases the lock and sleeps (sem_wait(&demoassistant)).
- After it is waken up by a student thread, it encounters the barrier which is set to value 3 indicating that it is waiting for 3 threads to arrive.
- After passing the barrier, it prints the I am now participating line.
- It checks if it was the waker thread, if so it releases the lock. (since we only released the lock for the thread that was going to sleep.)
- After that it acquires the lock again to increment the Aparticipate counter by 1 and releases the lock. This counter is keeping track of the assistants that have printed out the I am now participating line.
- It checks if the Aparticipate + Sparticipate equals to something that is divisible by 3, checks if the Aparticipate is more than 0, Sparticipate is more than 1. If that is the case it wont go into the while loop to sleep. (Lets say we had Aparticipate = 1 and Sparticipate = 2 here, it wouldn't go into the while loop, meaning that everybody in that demo group printed their participation line.) If it goes inside the while loop it sleeps (sem_wait(&finishassistant)).
- After it wakes up/everybody already printed the participation line, it prints out "Demo is over" line.
- This is the part where I've talked about in my second paragraph. It goes into a while loop and checks if the current students and assistants would equal to zero if he left the class. If so the loop terminates. If not it checks if the number of the remaining students would be less than 3 times the assistants in the class if he left. If so the loop also terminates. Else it keeps going.

- After the loop has been terminated, we decrement the enterAssistants counter by 1 indicating that 1 assistant has left the room, print the I left the classroom line, wake up 2 finishstudent threads that might be waiting for the assistant's participation line and then release the mutex.

Student thread function pseudo code:

- Prints that it wants to enter the classroom.
- Goes into a while loop and sleeps (sem_wait(&student)). If there has been allocated space in the classroom already for it, it doesn't sleep and checks the conditions. Conditions are the same as the assistant leave conditions. It checks if the enterStudent and enterAssistant counts are 0. If so the loop terminates. Else if it checks if this student joining would make the student count in the class still lower than 3 times the assistant count. If that is the case loop terminates. Else it goes back to the start of the loop and sleeps.
- After the loop it prints the "I entered the classroom" line.
- We have the same waker variable for the same purpose in student thread as well.
- It locks the mutex, increases the enterStudents by 1.
- Checks if there are >=1 assistants and >=1 students that are already waiting to form a group. If so this thread is the waker thread. It wakes 1 student and 1 assistant up. (sem_post(&demostudent), sem_post(&demoassistant)). Decreases the demoassistant and demostudent count by 1 indicating that they are no longer waiting for a thread to form a group.
- If that is not the case, it increases demoStudents count by 1, sets waker to false and releases the lock. Then it sleeps.
- When it proceeds from this point it encounters the barrier.
- After the barrier it prints the participation line.
- It checks if it was the waker thread. If so it releases the lock. If not it doesn't do anything.
- It locks the mutex again, increments the Sparticipate by 1 indicating that it has printed its participation line.
- After that it checks if the Aparticipate + Sparticipate equals to something that is divisible by 3, checks if the Aparticipate is more than 0, Sparticipate is more than 1. If that is the case it wont go into the while loop to sleep. (Let's say we had Aparticipate = 1 and Sparticipate = 2 here, it wouldn't go into the while loop, meaning that everybody in that demo group printed their participation line.) If it goes inside the while loop it sleeps (sem_wait(&finishstudent)).
- After it wakes up/everybody in its group already printed their participation it acquires a lock again.
- It prints out that it is leaving the classroom, decreases the enterStudents count by 1, wakes up 2 possibly sleeping assistant and student threads that are waiting in the participation line loop (finishassistant, finishstudent). Lastly it wakes up one student semaphore which could be waiting to get into the class after all the changes.
- It releases the mutex.

For the design choices of semaphores:

The student, demoassistant and demostudent semaphore's have the initial value of 0 since I want them to sleep whenever there is no space allocated for them.

However for the finishassistant and finishstudent semaphore I want them to be able to check their loop's condition once before they go to sleep. Therefore I assigned an initial value of 1 for those.

I've also declared 3 as the initial value for the barrier. (I've talked about it in assistant pseudocode.) The reason is that we need 3 threads to participate in each demo session.

Ural Sarp Sipahi 28093