Doruk Benli
29182

PA3 Report

Introduction

We are asked to implement a C/C++ program that simulates a demo room for our school projects. In the program we simulate a demo session with 2 students and an assistant in classroom. There are certain limitations for example students must wait assistants to enter the classroom, only after they can enter. Or the student count must be twice of assistant count. This simulation requires concurrency and synchronization and we utilize 5 semaphores 1 mutex and barrier to fully simulate the demo session. For each student we have arrive enter participate and leave and for each assistant we have enter demo and leave.

Pseudo code for the Threads

Assistant thread:

1. Enter the classroom
2. Post to sem_ast_in to allow student threads to enter
3. Increment assistant count
4. Notify if demo conditions are met
5. Wait for the opportunity to participate
6. Participate in demo
7. Leave the classroom
8. Check the remaining student count. If too high, wait for a student to exit
9. Leave the classroom

Student thread:

1. Express desire to enter the classroom
2. Wait for an assistant to be present in the classroom
3. Enter the classroom
4. Increment student count
5. Notify if demo conditions are met
6. Wait for the opportunity to participate
7. Participate in demo session.
8. Leave the classroom
9. Decrement student count

Doruk Benli
29182

Explanation and Synchronization techniques

To begin with barrier is used to sort of hold each demo session, thats why we initialize it to 3, because in each demo session we have 2 students and 1 assistant so we have to see that both assistant and student threads reach a certain point in the execution. We use this barrier. We use this barrier to be sure that we do not end the demo session before all of the 3 threads participate in that session so that no other demo starts before one of them participates.

Sem_ast_in semaphore: this semaphore is used to be sure that students wait for assistants before entering the classroom if they arrive earlier than the assistants. When an assistant enters the room and posts to the sem_ast_in, student threads proceed.

Sem_in: This semaphore checks the access of students and assistants to the classroom. This actually ensures that only 1 thread(assistant or student) can enter to a classroom at a given time. In my code sem_out also has the similar approach but with reverse logic, it ensures that only 1 thread can exit in a given time.

Sem_out: This semaphore, as its described above works in reverse logic of sem_in. It ensures that only 1 thread can exit the classroom at any given time. A thread performs sem_post() and wakes one thread up to signal that they can leave.

sem_Stud: this semaphore ensures that students can only participate if there exists an assistant in the classroom. This semaphore is set to 0 initially since they must wait for an assistant to enter a classroom.

Sem_ast: Similar to the sem_Stud, this semaphore is used to guarantee that assistants participate if there exists a student in the classroom.

Functions:

My flow of my program sort of mimics a pattern called observer pattern, where we have functions for threads for notifying that they can precede once an event takes place.

2 functions named astNotify and stdNotify checks if there are enough students or assistants for a demo to take place, if there are then they simply notify the threads(basically call sem_post for corresponding threads) and adjust the counts.

Doruk Benli
29182

2 functions as Participate and stdParticipate: these 2 functions simply prints that corresponding threads are participating, they call sem_wait() on required semaphores and print then they also call pthread_barrier_wait() for the barrier to ensure that all the threads reach to that point.

Lastly we have thread routines for the assistants and students, student threads use sem_wait on sem_ast_in to wait for assistants to enter before they do even if they arrive early. Similarly our assistant thread uses sem_post() to wake student threads up. Both threads then use their corresponding notify and participate functions in that order and and after making correct changes to the students or assistants in classroom and their counts and after printing they are done, they return.

We have main function or the main thread lastly, where we initialize and join our threads and semaphores, mutexes and barriers. But before those we first make necessary checks for compilation. If the nımber of arguments (argc) is smaller than 3, since we do not know the count of students we cannot compile hence we print an error message and directly terminate the thread with return1; after that we check if the number of assistants is positive and do something similar. Lastly we check if the amount of students the 2 times of the assistants. If all of these are met, we can run the program and say “my program compiles with all conditions”.

Summary

This program simulates a demo session by using various synchronization mechanisms. For this program to work concurrently, i have utilized semaphores, barriers and mutexes. They are used to make this demosim process synchronized and orders of actions are satisfied correctly, i.e a student cannot enter to the classroom before an assistant does.