2023 – 2024 Spring Term

CS307 – Operating Systems Course

Programming Assignment 3 Report

-

Berke Ayyıldızlı

31018

In your report, you must present the flow of your enter and leave methods as a pseudo code. You discuss which synchronization mechanisms (semaphores, mutexes and barriers) you have chosen, how you implemented, used or adapted them to suit your needs and provide formal arguments on why your code satisfies the correctness criteria described above.

1) Pseudo Codes

   a. enter() method

      i. First get the thread id.
      ii. Lock the mutex.
      iii. Checks if there is a game, if true, unlock mutex, wait the thread and lock the mutex.
      iv. Increase the player count by 1.
      v. Check if (we have a referee and player count is equal to 1 more of required players ORD we don't have a referee and player count is equal to the required players.
      vi. If so, make the match in progress bool true.
      vii. Print the match starting string.
      viii. If we have a referee, make the thread id referee id, and make the bool for checking if the referee has left false.
      ix. If step v fails, print passing some time, and release the mutex.

b. leave() method

    i. Get the thread id
    ii. Lock the mutex
    iii. Cheks if the match is in progress,
        1. Checks if the thread accessing this function is the referee, if so, print the appropriate statement, and for each player in the court, signal the semaphore.
            a. Make the bool for referee left true
            b. Signal that the match is over
        2. If not true,
            a. Unlock the mutex,
            b. Wait the threads until the referee is out
            c. Lock and print the appropriate statement
    iv. If not, print that no match can be found
    v. Decrease the player count
    vi. Check if player count is 0
        1. Make the match in progress bool false
        2. For each player in the court, signal that the match is ended.
        3. Since a game is played and ended, print the everybody left statement.
        4. Signal that the match is over.
    vii. Unlock the mutex

2) Selection of Synchronization Mechanism

For my programming assignment 3 code, I selected to use semaphores and mutexes. The implementation of a barrier seemed to be unnecessary, as correctly implementing the semaphores proves to be sufficient when combined with mutexes to make sure that only the wanted (or one) thread can access to the critical sections. Mutexes were particularly important to ensure the mutual exclusion when accessing and modifying shared resources. Semaphores were used as a conditional variable, signaling when and where to wait and continue thorought the code.

3) Implementation

Starting from the mutexes, I used a single mutex called *lock* to protect the critical sections in both enter and leave methods. The usage of mutex, ensures that the if statements that check if a match is in progress progresses correctly.

As for the usage of semaphores, since their primary purpose is to signal between threads, I implemented 3 of them.

accessToCourt semaphore is used to arrange the entering of the threads to the court. When a match is in progress, the threads that come to the game wait at the respected semaphore.

refereeFirst is to ensure that if a game has been initialized with a referee, the thread with the referee id exits the game first, in front of other player threads.

matchOver is used to signal for the last print statement. If a game is played and ended, the statement of "everybody left, letting any waiting people now" is printed.

Lastly, I need to mention why I did not use barriers in my homework. When used with mutexes, semaphored proved to be more than enough to ensure the dynamism of the synchronization requirements.

4) Correctness Criteria

To ensure that my court class satisfies the correctness criteria, we need to first know the correct implementation of my synchronization mechanisms. Since passing from all test cases, using mutexes with properly defined semaphores, ensures that mutual exclusion of threads trying to access to the critical sections of the enter and leave methods. Here, placing the semaphores' signals and waits carry an upmost value to prevent deadlocks, as I believe I have done on my implementation. I tried to also cover each method with mutex lock and unlock statements before the semaphore signals and waits, for the aim of preventing threads from data race. I also used printf statements rather than couts, to ensure that printings on the terminal do not clash out with each other. Overall, the test cases from both when the threads come at the same time and when the threads come at different times shows that the implementation of programming assignment 3, lies within the rules of concurrency and thread use.