Ipek Uzun 30837

CS307 PA3 Report

1. Pseudocode for enter() and leave() Methods

enter():

```
    print "I have arrived at the court"

  while True:
    wait(court_available)  // wait for a slot on the court

    lock(mutex)
    if matchGoing == true:
      unlock(mutex)
      signal(court_available)
      wait(match_end)    // wait for ongoing match to end
      continue

    inside += 1

    if enough players (and referee if required):
      matchGoing = true
      formed = true
      if referee is required:
        refereeId = current_thread
    else:
      formed = false

    if formed:
      print "There are enough players, starting a match"
    else:
      print "There are only X players, passing some time"

    unlock(mutex)
    break
```

leave():

```
lock(mutex)
  playing = matchGoing
  unlock(mutex)

  if not playing:
    print "Thread ID: <tid>, I was not able to find a match and I have to leave"
    lock(mutex)
```

```
        inside -= 1
        unlock(mutex)
        signal(court_available)
        return

    wait(barrier)  // synchronize with other participants

    if hasReferee and current_thread == refereeId:
        print "Thread ID: <tid>, I am the referee and now, match is over. I am leaving"
        repeat neededPlayers times:
            signal(match_end)  // wake up all players
    else:
        if hasReferee:
            wait(match_end)
        print "Thread ID: <tid>, I am a player and now, I am leaving"

    lock(mutex)
    inside -= 1
    last = (inside == 0)
    unlock(mutex)

    if last:
        lock(mutex)
        matchGoing = false
        refereeId = 0
        unlock(mutex)

        print "Thread ID: <tid>, everybody left, letting any waiting people know"
        repeat capacity times:
            signal(court_available)
```

2. Synchronization Mechanisms

2.1 Semaphores

First of all, I didn’t implemented semaphores myself I used the ones from semaphore.h library. I used 2 semaphores in total:

- **court_available**: I initialized a counting semaphoreto the total match capacity (players + referee). This semaphore controls how many threads are allowed to enter the court simultaneously. It prevents overfilling the court and ensures that no more than the required number of players participate in any match.
- **match_end**: I implemented this semaphore for the referee to notify all participating players that the match has ended. It ensures that no player leaves the court before the

referee, meeting the requirement that ″referee leaving″ must occur before any "player leaving".

2.2 Mutex

- I used one mutex that protects all shared variables (inside, matchGoing, refereeId). Every read and update to these variables is done inside critical sections protected by this mutex.

2.3 Barrier

I used a barrier is with a capacity equal to the addition of referee and players. I didnt implemented myself and used pthread_barrier_t from pthread library. It guarantees that all players and the referee begin the match simultaneously.

The barrier ensured that:

- The match only starts when everyone is ready.
- All threads that are part of a match leave after play() and barrier_wait() completes.
- The correct interleaving of messages between match starting and match leaving is respected.

Additionally I used barrier in leave() instead of enter(). It took me a lot of time to realize that but the main reason is to synchronize all participants after the match has started, not before. Using the barrier in enter() blocked threads and I encountered a deadlock.

3. Deadlock & Starvation Avoidance

- The program that I implemented blocking semaphores (not busy-waiting), ensuring no CPU cycles are wasted.
- All semaphore waits are paired with eventual post() calls, guaranteeing progress.
- No thread is indefinitely blocked without a matching signal, thus avoiding deadlock.
- Players who cannot join a match eventually leave with a ″no game″ message, avoiding starvation.

4. Why Is it Correct?

- Each thread prints exactly one init string:
  Every thread begins by calling enter() and prints "Thread ID: <tid>, I have arrived at the court." exactly once.

- Correct number of enter passtime and enter game strings:
  Threads that cannot form a match yet print "There are only <x> players...", while exactly one thread per game prints "There are enough players, starting a match.". This is ensured by using a mutex to guard the shared counter inside and the matchGoing flag.

Ipek Uzun 30837

- Correct number of no game strings:
  Threads that cannot form a match exit early with the message "I was not able to find a match and I have to leave." This is printed in the leave() method after checking matchGoing == false.

- Referee leaves before players:
  Within each match, we use a barrier in leave() to ensure all participants synchronize at the end of play(). The referee prints "match is over" and then signals players using a counting semaphore (match_end). Only after this signal do the players proceed to leave, preserving the required order.

- Exactly one everybody left message per game:
  The last participant to leave the court checks if inside == 0, resets the match state, and prints "everybody left, letting any waiting people know." This guarantees one and only one such message per match.

- No interleaved enter game before everybody left of previous match:
  A match cannot begin until all participants of the previous one leave. This is enforced by holding back new entries using the matchGoing flag and court_available.