Zolile Zoko ZKXZOL001

Parallel and Concurrent Programming Assignment 2

In a 4x100 medley relay swimming race, each team has four swimmers, each assigned a specific stroke—backstroke, breaststroke, butterfly, or freestyle. Each swimmer must complete their leg before the next swimmer in their team can start their leg of the race. This scenario is a real-life example of controlled resource sharing, where thread safety is crucial.

In this assignment, I implemented safety measures to ensure safe resource sharing and adherence to race rules. Thread safety was achieved through mutual exclusion mechanisms like synchronization methods, blocks, atomic objects and thread signaling. Synchronization methods and blocks prevent race conditions and data races by allowing only one thread to access critical code at a time. The Java monitor pattern was followed to synchronize methods in classes sharing resources.

Thread signaling was employed in two ways: first, to start the race when the user clicks the start button, allowing swimmer threads to run; second, to manage a baton system that enforces the correct swimming order. Each swimmer, upon entering the stadium, received the baton from the previous swimmer, ensuring they followed the stroke order. A CountDownLatch at the starting blocks made all 10 backstroke swimmers wait until they were all present before any could start swimming.

The baton system was also used to control the race, ensuring swimmers waited their turn and performed legal changeovers. After completing their leg, each swimmer passed the baton to the next, while the last swimmer remained in the pool. Atomic objects in this assignment refers to the use of AtomicBoolean and AtomicInteger as opposed to primitive types such as int and boolean, these atomic types help in preventing bad interleavings in the code in cases where a check and act mechanism is used, a basic example of a check and act mechanism is a getting and setting sequence of statements were getting the value of a primitive variable is done and then the value is used again. A bad interleaving result in stale values when getters and setters are not synchronized.

To prevent deadlocks, signaling rules ensured that no more than one swimmer thread could occupy a pool lane at a time, promoting orderly and fair racing, synchronizing the Grid class methods allowed for this to be achieved. Another modification to the provided code was the replacement of busy waiting with a signaling mechanism. This change allowed threads to sleep and only be notified when a specific condition was met, reducing CPU usage by avoiding continuous checks for the condition.

The conversion from busy waiting ("spinning") to a signaling mechanism also prevents starvation and introduces fairness. Busy waiting can consume CPU time continuously, potentially preventing other threads from running. The signaling mechanism, in contrast, ensures fairness and order, allowing every thread an equal opportunity to run.

This assignment provided valuable insight into the complexities of ensuring thread safety in a program where threads share resources. Trade-offs, such as increased program execution time due to mutual exclusion mechanisms making programs slightly slower and potentially reduced thread liveness, are often necessary to maintain thread safety.