sameera Oblig 5 report

Introduction:

This report is for oblig 5 “Synchronization” in IN3030 2024.
I have tested my code on a laptop with a 2,3 GHz to core Intel Core i5, but it has Hyper-Threading, so I have 4 threads.

User guide:

The program takes no parameters.

Example run:

java Oblig5

The program executes both the waitAndSwap and TestwaitAndSwap.

Semaphore Configurations:

waitSemaphore is crucial for making odd numbered threads wait. Starting with zero permits ensures that any odd-numbered thread entering the method cannot proceed until an even numbered thread releases a permit to it, effectively putting it in a wait state.

signalSemaphore is used to control the sequence of thread execution. The semaphore starts with one permit, allowing the first even numbered thread to proceed without waiting. Other even numbered threads must acquire this semaphore, ensuring they can only proceed after the previous odd-numbered thread has executed its wait operation and released this semaphore.

countSemaphore serves to protect the access to the shared count variable, which tracks the sequence number of threads. By locking and releasing this semaphore around the increment of count, the method ensures that thread counts are accurately tracked and incremented in a thread-safe manner. It also used to add to the inlist of threads going into the method, and the outlist of threads going out. They are later used to test the method.

entrySemaphore limits the number of threads that can enter the waitAndSwap method to two at a time. This restriction helps manage resource contention and ensures that threads enter the method in a controlled way.

Method Flow:

Threads acquire the entrySemaphore before entering the method, ensuring no more than two threads engage in the method simultaneously. countSemaphore is used to safely increment the thread counter and add the thread's identifier to the inList. Depending on whether the thread's turn is odd or even, it will either wait or proceed. Before exiting, each thread registers its identifier in the outList, showing the order in which threads complete the method. The entrySemaphore is released as the thread exits the method, allowing other queued threads to enter.

TestwaitAndSwap:

The Test method validates the effectiveness of the waitAndSwap method by checking the order in which threads were expected to be swapped. It iterates through the lists of thread entries and exits, comparing their indices to ensure that odd-indexed threads in inList exit in the position one place ahead in outList, and even-indexed threads exit one place back, reflecting the correct swap operation. If any thread does not match this expected pattern, the method returns false, indicating an error in the thread swapping process. It returns true, confirming the correctness of the swapping logic.

Conclusion:

We can see that it is possible to swap these threads pair-wise with only using semaphores.