Oblig5 IN3030 – Synchronization with Semaphores

Introduction

The report is about synchronization is done using semaphores. In the report I am talking about how the program is used, a little more about the synchronization, some about the implementation and at the end some result of testing and a conclusion.

User guide

Compile the with; **javac *.java**

And then run the program with; **java WaitAndSwap {number of threads}**

You can also run the program with number of iterations which is put behind the number of threads, eks.; **java WaitAndSwap {number of threads} {number of iterations}**

Number of threads must be an even number, because of how the swapping work between threads work. Pair of threads are swapping the number of calls in the waitAndSwap()-function. So, if there is an uneven number of threads the last threads will just wait for the next thread, but it will never come.

How synchronization is done

I am using two semaphores, where one is starting with the value -1 and the other with 1. The reason one of them are -1 is because of the first call where we need a way to stop the call without releasing the first call immediately. The synchronization is done where the odd call is waiting for the next odd number to release it but after the even number in between has passed.

Implementation

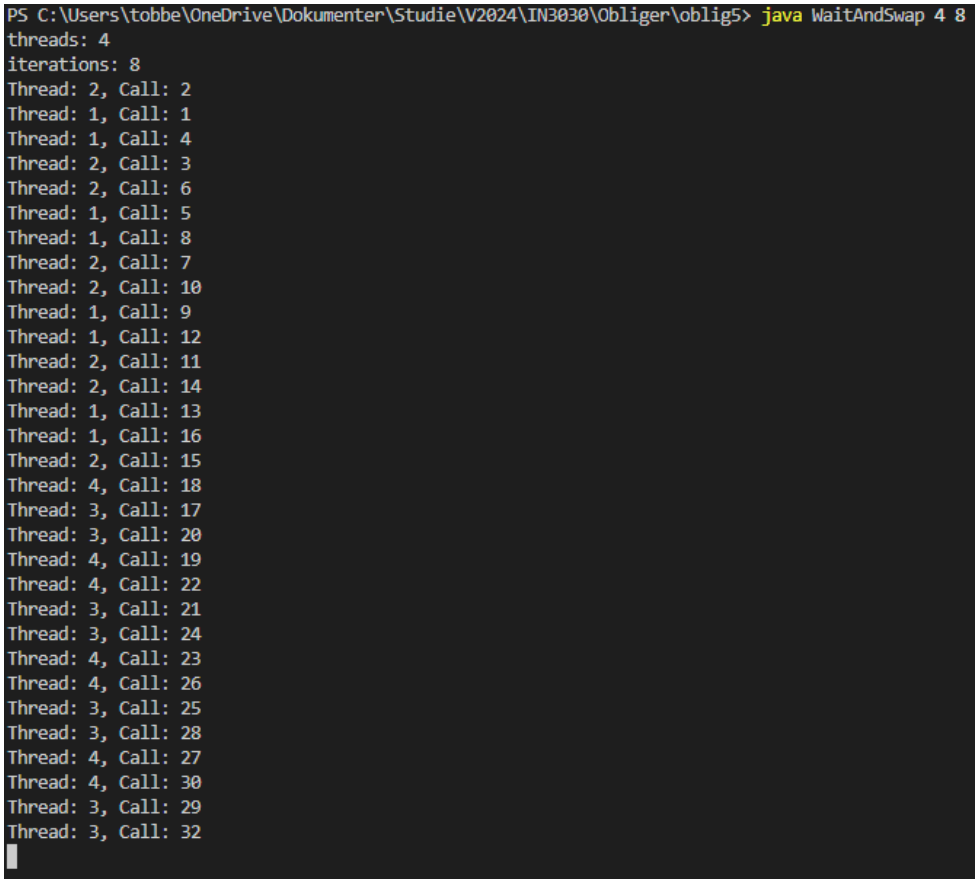
My java program is starting with taking in some arguments from the user and then setting them to some variables. Then it is starting a loop where all the threads are created and sent to the Worker class. There each thread is doing N iterations where every iteration is doing a call on the waitAndSwap()-function where the synchronization is done and each call is printing out in the order I want. The synchronization part is described better above. The program will never terminate because when it is on the last call of the function it is waiting for the next call to the function, but it is never coming.

Testing

I tested it with running the program with different arguments. Here are some examples, two with different number of threads and one where I am changing the number of threads and the number of iterations.

```
PS C:\Users\tobbe\OneDrive\Dokumenter\Studie\V2024\IN3030\Obliger\oblig5> java WaitAndSwap 2
threads: 2
Thread: 2, Call: 2
Thread: 1, Call: 1
Thread: 1, Call: 4
Thread: 2, Call: 3
Thread: 2, Call: 6
Thread: 1, Call: 5
Thread: 1, Call: 8
```

```
PS C:\Users\tobbe\OneDrive\Dokumenter\Studie\V2024\IN3030\Obliger\oblig5> java WaitAndSwap 12
threads: 12
Thread: 2, Call: 2
Thread: 1, Call: 1
Thread: 1, Call: 4
Thread: 2, Call: 3
Thread: 2, Call: 6
Thread: 1, Call: 5
Thread: 1, Call: 8
Thread: 2, Call: 7
Thread: 4, Call: 10
Thread: 3, Call: 9
Thread: 3, Call: 12
Thread: 4, Call: 11
Thread: 4, Call: 14
Thread: 3, Call: 13
Thread: 3, Call: 16
Thread: 4, Call: 15
Thread: 6, Call: 18
Thread: 5, Call: 17
Thread: 5, Call: 20
Thread: 6, Call: 19
Thread: 6, Call: 22
Thread: 5, Call: 21
Thread: 5, Call: 24
Thread: 6, Call: 23
Thread: 8, Call: 26
Thread: 7, Call: 25
Thread: 7, Call: 28
Thread: 8, Call: 27
Thread: 8, Call: 30
Thread: 7, Call: 29
Thread: 7, Call: 32
Thread: 8, Call: 31
Thread: 10, Call: 34
Thread: 9, Call: 33
Thread: 9, Call: 36
Thread: 10, Call: 35
Thread: 10, Call: 38
Thread: 9, Call: 37
Thread: 9, Call: 40
Thread: 10, Call: 39
Thread: 12, Call: 42
Thread: 11, Call: 41
Thread: 11, Call: 44
Thread: 12, Call: 43
Thread: 12, Call: 46
Thread: 11, Call: 45
Thread: 11, Call: 48
```



Conclusion

It is working as intended.