Name: Tanisha Rana
Section: CSE4
Roll no.: SE20UCSE202

README ASSIGNMENT 5

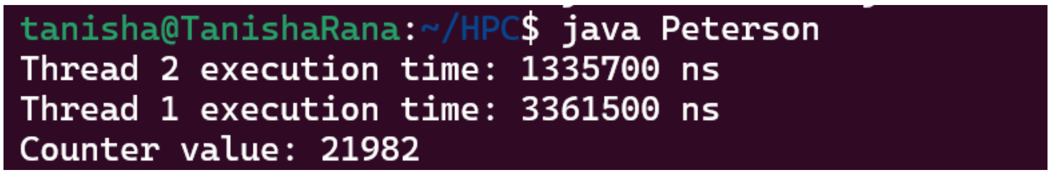
About the code:
This code implements Peterson's algorithm, which is a mutual exclusion algorithm for two threads. It ensures that only one thread can access a critical section of code at any given time, preventing race conditions.
The code consists of a main method and two helper methods: incrementCounter and PetersonLock. The incrementCounter method increments a shared counter variable while using the PetersonLock object to ensure that only one thread can access the counter at a time. The PetersonLock object implements the algorithm by using two boolean flags and a victim variable.
The main method creates two threads and starts them. Each thread calls the incrementCounter method with a thread id and a number of iterations to perform. After both threads have completed, the main method outputs the final value of the counter variable.

Cases Taken for Different Workloads:

Case 1:

Thread t1 = new Thread(() -> incrementCounter(0, 20000), "Thread 1");

Thread t2 = new Thread(() -> incrementCounter(1, 2000), "Thread 2");

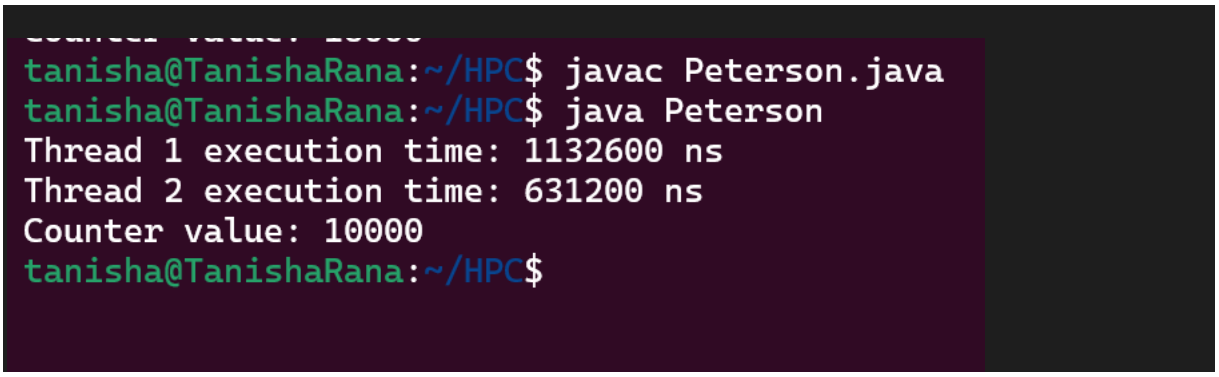


```
tanisha@TanishaRana:~/HPC$ java Peterson
Thread 2 execution time: 1335700 ns
Thread 1 execution time: 3361500 ns
Counter value: 21982
```

Case 2:

Thread t1 = new Thread(() -> incrementCounter(0, 5000), "Thread 1");

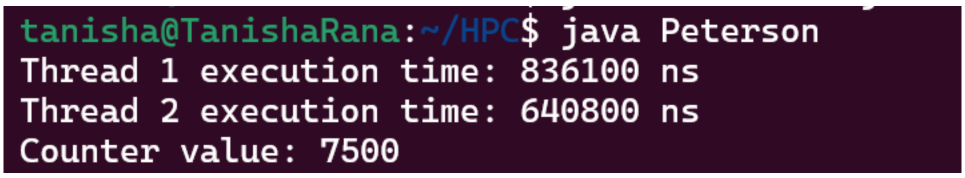
Thread t2 = new Thread(() -> incrementCounter(1, 5000), "Thread 2");



```
tanisha@TanishaRana:~/HPC$ javac Peterson.java
tanisha@TanishaRana:~/HPC$ java Peterson
Thread 1 execution time: 1132600 ns
Thread 2 execution time: 631200 ns
Counter value: 10000
tanisha@TanishaRana:~/HPC$
```

Case 3:
Thread t1 = new Thread(() -> incrementCounter(0, 2500), "Thread 1");

Thread t2 = new Thread(() -> incrementCounter(1, 5000), "Thread 2");

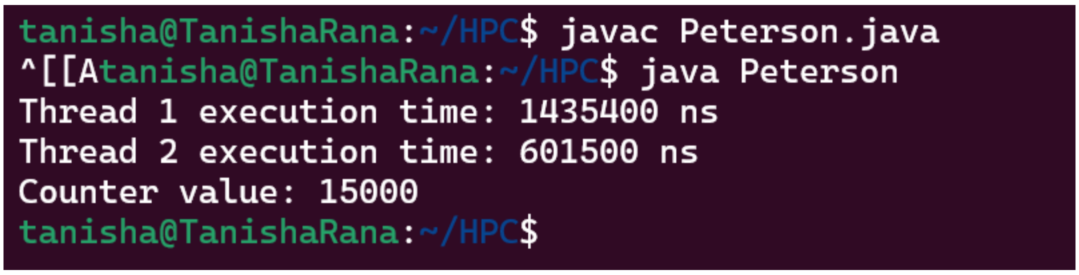


```
tanisha@TanishaRana:~/HPC$ java Peterson
Thread 1 execution time: 836100 ns
Thread 2 execution time: 640800 ns
Counter value: 7500
```

Case 4:

Thread t1 = new Thread(() -> incrementCounter(0, 10000), "Thread 1");

Thread t2 = new Thread(() -> incrementCounter(1, 5000), "Thread 2");

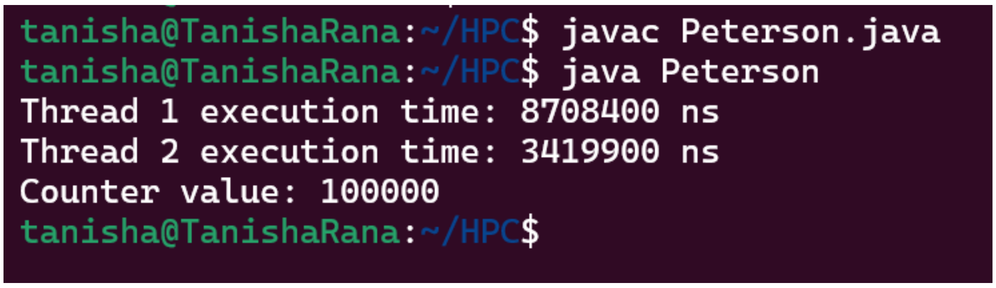


```
tanisha@TanishaRana:~/HPC$ javac Peterson.java
^[[Atanisha@TanishaRana:~/HPC$ java Peterson
Thread 1 execution time: 1435400 ns
Thread 2 execution time: 601500 ns
Counter value: 15000
tanisha@TanishaRana:~/HPC$
```

Case 5:

Thread t1 = new Thread(() -> incrementCounter(0,50000), "Thread 1");

Thread t2 = new Thread(() -> incrementCounter(1, 50000), "Thread 2");

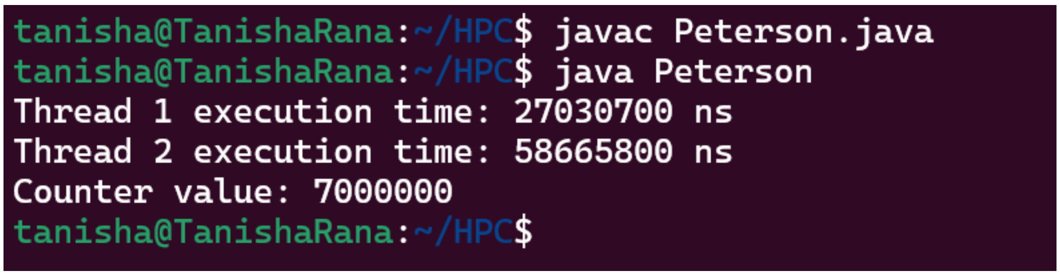


```
tanisha@TanishaRana:~/HPC$ javac Peterson.java
tanisha@TanishaRana:~/HPC$ java Peterson
Thread 1 execution time: 8708400 ns
Thread 2 execution time: 3419900 ns
Counter value: 100000
tanisha@TanishaRana:~/HPC$
```

Case 6:

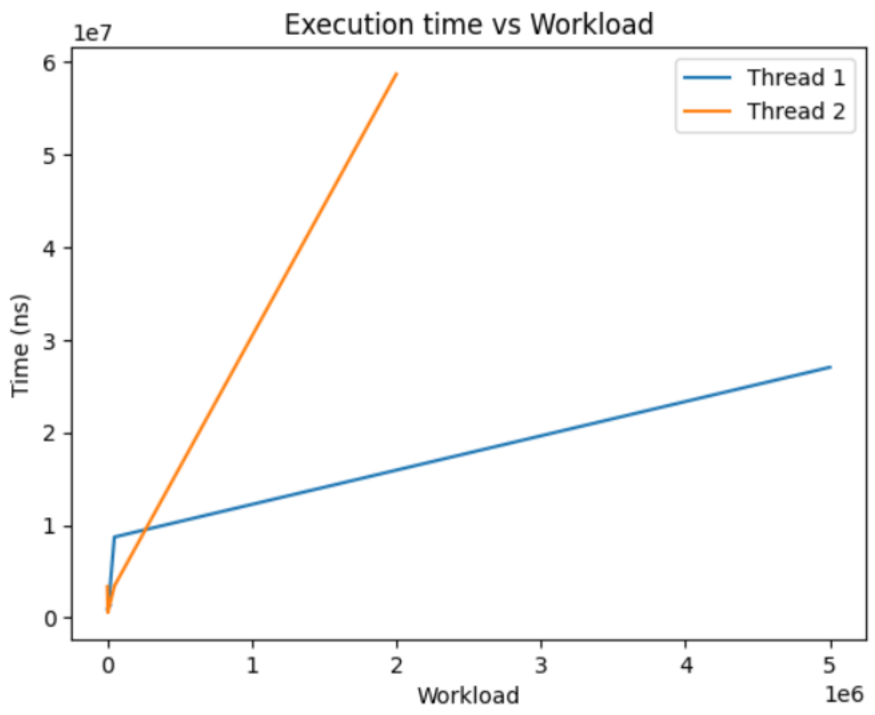
Thread t1 = new Thread(() -> incrementCounter(0,5000000), "Thread 1");

Thread t2 = new Thread(() -> incrementCounter(1, 2000000), "Thread 2");



```
tanisha@TanishaRana:~/HPC$ javac Peterson.java
tanisha@TanishaRana:~/HPC$ java Peterson
Thread 1 execution time: 27030700 ns
Thread 2 execution time: 58665800 ns
Counter value: 7000000
tanisha@TanishaRana:~/HPC$
```

Graph:


Execution time vs Workload
1e7
6
5
4
3
2
1
0
Time (ns)
Thread 1
Thread 2
0
1
2
3
4
5
Workload
1e6