UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Practical 3

- **Date issued:** 29 August 2022
- **Deadline:** 08 September 2022, 8:00 PM
- This practical consists of 2 task. Read each task **carefully!**

1 Introduction

1.1 Objectives and Outcomes

This practical aims to explore wait-free methods of concurrency (consensus protocol) and futher experiment on Spin Locks and Contention.

You must complete this assignment individually. Copying will not be tolerated.

1.2 Submission and Demo Bookings

You will be provided with some skeleton code for task 1, consisting of the following Java classes: **Consensus.java, ConsensusThread.java, ConsensusProtocol.java**

NO skeleton code has been provided for task 2.

Submit your code to **clickup** before the deadline.

You will have to demonstrate each task of this practical during the **physical** practical lab session. So be sure to create copies of your source code for each task separately. Booking slots will be made available for the practical demo.

1.3 Mark Allocation

For each task in this practical, in order to achieve any marks, the following must hold:

- Your code must produce console output. (As this is not marked by fitchfork, formatting is not that strict)
- Your code must not contain any errors. (No exceptions must be thrown)
- Your code may not use any external libraries apart from those highlighted in the textbook.
- You must be able to explain your code to a tutor and answer any questions asked.

The mark allocation is as follows:

Task Number	**Marks**
Task 1	5
Task 2	5
Total	10

2 Practical Requirements

2.1 Task 1 - Read-Modify-Write Consensus

For this task, two friends are deciding how much to spend on a night out, you must simulate their decision by performing a RMWConsensus protocol:

2.1.1 Implementation

You must implement the following:

- ConsensusProtocol.java
  - This class is given.
  - Implement the **propose()** method.
- ConsensusThread.java
  - This class is given.
  - Implement the **run()** method
- RMWConsensus.java
  - You must create this class to extend ConsensusProtocol
  - Implement the **decide()** method.

2.1.2 Notes

main

- There should be two threads for this task, each must do the following:
  - Each thread must propose an amount to spend between 100 and 200. main/thread
  - The threads must then wait for a random amount of time between 50 and 100 ms. thread
  - Each thread must then decide on the same chosen amount. thread
  - This must be repeated 5 times. thread
- The threads must then wait for a random amount of time between 50 and 100 ms.
- Be sure that the value decided is the same for both threads

2.1.3 Output

The following output needs to occur:

- Output the value that the thread proposes to spend when **propose()** is called
- Output the value of the register when **decide()** is called
- Output the value each thread decided on.

2.2 Task 2 - Spin Locks and Contention

Performance is highlighted as the main factor in chapter 7. In this task you will perform an experiment to analyse the performance of three locks:

- Test-and-Set Lock
- Test-and-Test-and-Set Lock
- Exponential Backoff Lock

2.2.1 Notes

The following needs to be completed:

- Iteratively test the instance of the locks with different number of threads. All the locks must be tested, with the same number of threads to compare performance.
  - After a thread acquirers the lock it sleeps for 100 milliseconds
  - Each thread must access the critical section for a variable number of times. You are free to play around with this attribute.
  - For each iteration i.e. number of threads, record on the execution time for each lock.
- Report on the gradual (if not exponential) increase in the execution time for the different locks based on the number of threads .e.g number of threads: [1,2,3,4,7,10,14,19,25,35] x-axis and time(ms): [t,t,t,t,t,t,t,t,t,t] y-axis
  - You may have a minimum length of 5 for number of threads array

2.2.2 Output

- Example output:

```
Number of threads: [1,2,3,4,7,10,14,19,25,35]
-------------------------------------------------------------
TASLock: [t,t,t,t,t,t,t,t,t,t] time in [units]
TTASLock: [t,t,t,t,t,t,t,t,t,t] time in [units]
BackoffLock: [t,t,t,t,t,t,t,t,t,t] time in [units]
```

- Note, the length of the output array for the different Locks, should be the same as that of the number of threads. The array should be a minimum length of 5.
- Each value, t, reports on the execution time for the corresponding number of threads in the number of threads array.