UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Practical 4

- **Date issued:** 10 September 2022
- **Deadline:** 29 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 Spin Locks and Contention.

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

1.2 Submission and Demo Bookings

You will are provided with some skeleton code, consisting of the following Java classes: **Main.java, Marshal.java, VotingStation.java, MCSQueue.java** and **Timeout.java**

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

Your task is to simulate a voting protocol using concurrent programming. The elections follows a specific procedure to ensure the integrity of the election results. You will simulate this protocol which the electoral committee has adopted by means of threading.

2.1 Task 1 - MCS Queue Lock

In order to mitigate the issue of spoilt ballots and to save time, ballot papers are released a week before the election day. During the election voters only have to submit the ballot paper. The process of voting is as follow:

- All voters have to stand in a queue. There are 5 different queues designated to different types of people, e.g civil servants, the elderly etc. Assume that there is 5 people in each queue.
- Each line is allocated a voting marshal who will escort 1 person at a time into the voting station.
- Once a person has entered the voting station the marshal ensures that they stand in a queue if there is one, otherwise they can just submit the ballot paper into the ballot box. This means that at most only 5 people can be in the voting station excluding the marshals.

Note:

- A thread will simulate a marshal and the ballot box will be the critical section.
- Once a person is first in the queue, i.e. their turn to submit the ballot, the thread will sleep for a randomly selected amount of time between 200 and 1000 milliseconds before exiting.

2.1.1 Output

The following output is expected:

- When a person ENTERS the voting station, the following will need to be output: [Thread-Name][Person-Number] entered the voting station.
- When a person has CAST a vote, the following will need to be output: [Thread-Name][Person-Number] cast a vote.

- When a person EXITS the voting station, the following will need to be output:

  QUEUE: {[Thread-Name]:[Person-Number]} -> ....
  Example: QUEUE: {Thread-1:Person 1} -> {Thread-2:Person 1}
  Note: This is the queue in the voting station

Note that you will have to be creative in printing the queue. Hint: consider using the node to store the information.

2.2 Task 2 - Time-out Lock

The following needs to be completed:

- The MCS Queue Lock from the previous task needs to be replaced by a Time-out Lock
- Implement your Time-out Lock inside a **Timeout** class.
- Change the simulation you have created to make use of the Time-out instead of the MCS Queue.
- The output remains the same as Task 1.

Note: Select a timeout based on the acceptable time a person should wait/spend in the voting station.