UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Practical 2

- **Date issued:** 03 August 2022
- **Deadline:** 10 August 2023 (Midnight)
- This practical consists of 2 task. Read each task **carefully!**

1 Introduction

1.1 Objectives and Outcomes

This practical aims to further explore locking via implementation of new locks when 3 or more threads are involved.

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

1.2 Submission and Demo Bookings

You are provided with some skeleton code to aid in the assignment, consisting of the following Java classes: **Main, Mythread, SharedResources, Filter, Bakery**

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 for **locking** apart from those already provided.
- 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

You are developing an algorithm to solve a computational problem and you decided to incorporate multi-threading to speed up the processing time. Before writing the algorithm you decide to test multi-threading operations with regard to the shared resources i.e. the critical section. Complete Task 1 and 2 below.

2.1 Task 1 - Filter Lock

For this task you will need to implement the above mentioned test using the **FilterLock** as highlighted in the textbook. A FilterLock will need to be implemented inside the Filter class. i.e. A **lock()** and **unlock()** method. Each thread:

- access the shared resource at least twice and sleep for a randomly selected amount of time between 200 and 1000 milliseconds in the critical section each time.
- for every attempt level i, in FilterLock, print out the following:
  {Thread-Name}: level[{me}] = {i} , victim[{i}] = {me}
  Example: Thread-1: level[1] = 0 , victim[0] = 1
- print out the following before leaving the critical section:
  {Thread-Name}: ———————————— DONE

2.2 Task 2 - Bakery Lock

For this task you will need to modify your previous implementation to make use of a BakeryLock.

- Implement your BakeryLock inside the **Bakery** class.
- Change the test you have created to make use of the BakeryLock instead of the FilterLock.

Note! Unlike FilterLock, BakeryLock lock uses different data structures i.e. a flag and a label. Print out the following for every attempt to acquire the lock i.e. every call to lock:

- {Thread-Name}: flag[{i}] = true , label[{i}] = {max + 1}