UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Assignment 2

- **Date issued:** 29 September 2023
- **Deadline:** 15 October 2023 (Midnight)
- This assignment consists of a single task. Read **carefully!**

1 Introduction

1.1 Objectives and Outcomes

This assignment aims to test all the concepts that have been learned so far regarding mutual exclusion and locking via practical implementation without assistance.

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

1.2 Submission and Demo Bookings

You are NOT provided with any skeleton code, you will have to implement everything yourself.

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

You will have to demonstrate each task of this practical during the **physical** practical lab session. Booking slots will be made available for the practical demo.

1.3 Mark Allocation

For the assignment, 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 the tutors and answer any questions asked.

The mark allocation is as follows:

Task	**Marks**
Implementation	10
Explanation	10
Total	20

2 Assignment

For this assignment you are tasked with simulating a chat application using a client-server communication setup. A client-server commutation allows multiple clients to communicate with each other by sending a message destined for another client via a central server. The server knows all the clients in the network. You must simulate this communication between clients without the use of socket, rather using only multi-threading.

2.1 Implementation

Each client communications will be handled by 2 threads, a reader and a writer

- A Reader: Checks for any new message, on the server, for the client and logs the message in the clients chat.
- A Writer: Sends new messages, destined for a specific client, to the server and logs the message in the clients chat.
- You will have to create and initialize at-least 4 client objects
  - initialize at-least 10 random messages to send to a randomly selected client
  - Example: { message: [random message], recipient: [client-name]}

A server will be represented by a data-structure, holding messages for clients

- That is, each thread has a fixed location on the server where it receives new messages
- The server does not keep chat history, a new message for a specific client overrides the old one

2.2 Mutual Exclusion

You are expected to enforce mutual exclusion at the following key points:

- Client chat
  - The reader and writer must log messages using mutual exclusion
  - You can use any lock of you choice
- Server
  - Mutual exclusion must be enforced to manage writers writing to the same client
  - You can use any queue based lock of your choice

2.3 Output

The following output must be produced:

- When a thread attempt to send a message:
  (SEND) [Thread-Name]: { sender:[client-name] , recipient:[client-name]}.
- When a thread successfully sent a message:
  (SEND) [Thread-Name]: SUCCESSFUL.
- When a thread reads a new message:
  (RECEIVE) [Thread-Name]: { recipient:[client-name], sender:[client-name] }.

2.4 Note

- You will have to get creative with handling threads, queues and locks.
- A general rule of thumb is that each queue should have their OWN lock when accessed.
- You may add as much code and as many classes as you deem necessary.
- Any locks used in the program will have to be written from scratch. i.e NO using javas pre-built locks.
- You may use any of Javas pre-built data structures.
- Any locks used must be FAIR