**CSC3002F: Operating Systems Assignment 2**

**Social Distancing Shop Simulation**

Process Synchronization



**By:** Zukiswa Lobola (LBLZUK002)

**Lecturer:** Michelle Kuttel

**Date:**  22 June 2020

# INTRODUCTION

In an operating system, to enable multithreaded programming, multiple threads (processes) have to execute concurrently. These threads work with resources that must be shared between them and often need to coordinate with each other to gain access to sections of memory without interference. Threads coordinate access to shared resources by means of process synchronization.

Amidst the world-wide occurring Covid-19 pandemic, this has given rise to societal constraints, such as restriction on the amount of people allowed in a shop at a time, that mimic that of a multithreaded operating system. Thus, a java simulation model of a shop (shared resource) with customers (threads) obeying the social distancing regulations has been created as a mechanism to display various concepts of process synchronization through the use of semaphores.

# THREADS RUNNING IN THE PROGRAM

In our social distancing shop simulation, many customers visit the shop – however only a restricted an amount are allowed in the shop at a given time. Since there are many customers who are unpredictable and wish to access the various parts of the shop – these are represented as threads.

thread = threaded panel to display shop

thread = counter thread for updating counters

thread = to inspect the shop to make sure customers are obeying social distancing regulations

# CLASSES SHARED AMONGST THREADS

# SYNCHRONIZATION MECHANISMS ADDED TO DESIGNATED CLASSES

* Binary Semaphores were added to this class as a lock to protect the *critical sections* of the code as the semaphore object named . Critical sections refers to the segments of the code where several threads need mutually exclusive access to shared resources/data in that region. Mutually exclusive access is necessary to avoid data inconsistencies where any alteration to these resources affects the outcome of this resource for the other threads. These locks were used in the , and methods to protect the sections which update the class counters that all threads have access to. When a customer arrives, enters and leaves the shop, it affects the and class variables. Only one thread should update these class counters at a time to maintain data consistency and accuracy throughout the program.
* Counting Semaphores were added to this class to act as a buffered barrier which controls the number of threads that have access to certain sections of the code at a time. Every time a customer enters the shop the counter for increases, and the space available for more customers decreases until it is full and another customer cannot enter unless another customer leaves the shop. This problem is known as the producer-consumer problem, where customers entering are the producers who add data () into the buffer until it becomes full and can only be removed from the buffer by consumers who remove data () represented by customers leaving the shop. The restriction on the maximum amount of customers inside the shop at a given time acts as the buffer.

* Binary semaphore – mutual exclusion

* Binary semaphore – mutual exclusion

# LIVENESS IN THE CODE

# PROTECTION AGAINST DEADLOCK