# COMP3520 Operating Systems Internals Assignment 1 – Pthread Programming

## General Instructions

This assignment is about *Pthread* programming. It consists of two parts and you need to tackle both of them.

**This assignment is an individual assignment.** Whilst you are allowed to discuss this assignment with other students, the work that you submit must be substantially your own work. You should not incorporate the works of other students (past or present) into your source code or report.

You will be required to submit your report to *Turnitin* for similarity checking as part of assignment submission. The examiner may use other similarity checking tools in addition to *Turnitin*. Your source code may also be checked.

Submit your source code and report to the appropriate submission inboxes in the COMP3520 LMS website.

## Part A: Parallel Matrix Multiplication

Given two matrices $A$ and $B$, where $A$ has $M$ rows and $K$ columns and $B$ has $K$ rows and $N$ columns, $C = A \times B$ is the matrix multiplication of $A$ and $B$, which gives you a new matrix $C$, where $C$ has $M$ rows and $N$ columns. The entry value of $C$ in $i^{th}$ row and $j^{th}$ column $c_{ij}$ is the inner product of the $i^{th}$ row of $A$ and the $j^{th}$ column of $B$, that is, $c_{ij} = \sum_{k=0}^{K-1} a_{ik} \times b_{kj}$.

For this project, you need to write a *Pthread* program to calculate the matrix multiplication. In the main program the main, or master thread first gets the matrices dimensions $M$, $K$ and $N$ from the user, creates three global matrices $A$, $B$ and $C$, respectively, using dynamic memory allocation, and initialise the matrices with *float* type numbers using the random number generator. The master thread then gets a number from the user on how many threads $t$ ($\leq N$) to be used for the parallel computation, and creates the worker threads to calculate the matrix multiplication. After $t$ threads are created, the master thread waits for all worker threads to complete their tasks, checks the correctness of the result, print out matrix C and terminates.

(To check the correctness the main thread will conduct a sequential computation for the matrix multiplication and compare the result with the one calculated using $t$ threads.)

Consider the whole computation as a single task. You need to divide the single task into a number of smaller tasks. There are various ways to divide the task. For this project, you are asked to divide $N$ columns of matrix $C$ into $t$ blocks and each worker thread will then calculate one block of columns. Each block must contain roughly the same number of columns for load balancing.

## Part B: Terminal Problem

Consider a library containing *m* terminals. Due to high demand, clients must wait in a waiting room for the attendant to serve them. A client must be served by the attendant before occupying a terminal. Each terminal may be occupied by at most one client and each client may not occupy more than one terminal.

The waiting room has a finite capacity of *n* seats. If all seats are occupied when a client arrives, then that client leaves immediately and is never served. Otherwise, the client occupies a seat and waits for the attendant to serve them.

The attendant waits for clients. When there are clients waiting in the waiting room, the attendant randomly selects the next waiting client to serve. However, the attendant will serve a client if and only if a terminal is available. If no terminals are available, the attendant must wait until a terminal is available.

Once the attendant serves a client, the client leaves the waiting room, occupies a terminal for a certain period of time, returns the terminal and then leaves.

When there are no more clients to serve and no clients will arrive in the future, the attendant leaves.

To implement synchronization between the attendant and client threads, you must use ONLY **condition variables** and **mutexes**; **you must NOT use semaphores or busy waiting, or any other types of synchronization mechanisms**.

The main thread first gets, from the user, the total number of seats, the total number of terminals, and also the total number of clients, and clients's arrival rate and terminal usage. It then creates the attendant thread and client threads.

To check whether your program functions properly, **each customer thread must print** the following status messages wherever appropriate where *id* is the identifying number of the thread:

- "Customer %d arrives.\n"
- "Customer %d: oh no! all seats have been taken and I'll leave now!\n"
- "Customer %d: I'm lucky to get a free seat from %d.\n"
- "Customer %d: I'm waiting to be served.\n"
- "Customer %d: I'm getting a terminal now.\n"
- "Customer %d: I'm finished using the terminal and leaving.\n"

**The attendant thread must print** the following status messages wherever appropriate:

- "Attendant: The number of free seats is %d. No customers and I'm waiting. \n"
- "Attendant: The number of free seats now is %d. try to find a free terminal.\n"
- "Attendant: The number of free terminal s is %d. All terminals are occupied. \n"
- "Attendant: The number of free terminals is %d. There are free terminals now. \n"
- "Attendant: Call one customer. The number of free seats is now %d.\n"
- "Attendant: Assign one terminal to the customer. The number of free terminals is now %d.\n"

# Additional Requirements

## Short Report

You need to write a short report regarding your solutions to both "matrix multiplication" and "terminal problem". You must include the following FIVE sections in your report:

- **Basic Concept**
  - The basic concepts involved in solving the problems;
- **Algorithm**
  - The algorithms that you have used (in pseudocode form with explanation);
  - Descriptions and justifications of your use of mutexes and condition variables (if used);
- **Discussion**
  - Testing results to show your algorithm and program working correctly;
  - Descriptions of any limitations of your solution;
  - Potential improvement;
- **Manual**
  - User manual to describe how others can use your programs
- **Acknowledgement**
  - Assistance that you have received in the preparation of your solutions (if any);
  - References (if any)

## Programming Language

Your dispatcher must be implemented in the C language. C++ features are not permitted except those that are part of an official C standard.

## Source Code

The source code needs to be properly commented and appropriately structured to allow another programmer who has a limited working knowledge of C to understand and easily maintain your code.

You need to include a well-structured and properly commented makefile that allows for the compilation of your source code using the make command on the SIT servers, e.g., *ucpu2*.

## Testing and Debugging

The testing and debugging of your source code is **your** responsibility.

It is crucial that you ensure that the source code you submit will compile. No marks will be given if your source code cannot be compiled on the SIT server.

## Code Template

The code templates for this assignment will be given to you. **You must use the templates to implement your solutions.**

# Other Matters

## Academic Dishonesty, Non-serious Attempts and Non-attempts

In order to receive credit for this assignment, you must apply yourself with due diligence and sustained effort to this assignment. Also, you must not engage in academic dishonesty. Penalties for engaging in academic dishonesty, a non-serious attempt or a non-attempt will be severe.