# SYSC4001
# Assignment 3 - Part 2

Report

Eric Cui – 101237617

Aydan Eng – 101295820

Part 2 GitHub

**Introduction**

This report briefly discusses about part 2 section of the assignment 3. The objective is to simulate the effects of semaphores in programs where multiple processes read and write on shared memory. The simulation create two similar program, where one uses semaphores and the other does not, and compare the outcomes of the process.

**Order of operation**

The order at which the program was executed is as follows:

1. The initialize.cpp script setup the shared memory, initialize the variables, load students and rubric, and clone the number of TA requested by the arg input via fork() exec().
2. Each ta assign themselves an id.
3. TA begin marking rubric, waiting between 0.5-1.0 seconds to check the rubric with 50% chance of changing the rubric to the next letter.
    a. In the semaphore version, the TA takes a semaphore before changing rubric and release the semaphore after. This ensures that only one ta grades at a time.
4. When TA finishes, they wait until other TAs are finished as well before proceeding.
5. The TA then start grading students. They go through the questions one by one and check if it has been graded. If it is not graded, the TA take 1.0-2.0 seconds to grade a question.
    a. In the semaphore version, the TA takes a semaphore before grading a question and release the semaphore after. This ensures that no question is graded by multiple TA.
6. When a TA finishes grading all questions of a student's test, they wait until other TAs are finished as well before collectively move onto the next student.
7. The TA repeat step 5-6 until they finished 20 students. Then they end the program.

Due to the order of operation, the program did not experience deadlock/livelock, and therefore part 3 is skipped.

**Analysis**

The non-semaphore version of the program have demonstrated the problem with multi-threading. In the program, TAs constantly mark the same questions at the same time, resulting in many questions being graded multiple times. This occurrence significantly

increases as more TAs are added to the program, as well as number of times each repeat. In the semaphore version, the use of semaphore has effectively prevented the concurrent issue, ensuring that each question are only graded once.

**Conclusion**

This simulation demonstrates the problem with multi-threading and how semaphore addresses this problem effectively.