SYSC4001 - Assignment 3 - Part 2c Report

Amit Kunkulol - 101260631
Kushal Poudel - 101298706
Repository: https://github.com/amit46/SYSC4001_A3_P2

In Part 2a, the implementation used shared memory without any synchronization. All TAs read and modified the shared rubric array and exam_state.question_done array concurrently. This led to clear race conditions:

- Multiple TAs sometimes corrected the same rubric entry at almost the same time, so characters could jump multiple times unexpectedly.
- When marking questions, multiple TAs could both choose that a question was still not done and print that they had marked that question. This means the same question for the same student could be graded multiple times.
- The loader logic for the next exam depended on a shared state. Without protection, there was a risk of TAs reading or acting on partially updated exam state.

The output from Part 2a runs show these race conditions. you can see overlapping corrections to the rubric and repeated messages about marking the same question for the same student.

In part 2b, we used semaphores for synchronization (implemented using POSIX). By introducing three semaphores, only one TA at a time can be inside the critical section that updates the rubric or the question-done array.

- We no longer have two TAs writing to the same rubric entry at a time.
- Questions can now be claimed, avoiding the double-marking issue.
- We can ensure that only one TA (TA 0) updates which exam is currently loaded. The other TAs never see the partially updated exam state.
    - The other TAs either work on the current student, or cleanly exit once the student ID #9999 is seen.

Order of Execution:

- sem_rubric protects shm->rubric[] and save_rubric().
- sem_questions protects exam_state.question_done[] and the “are all questions done?” check.
- sem_loader protects load_exam() and updates to current_exam_index.
- Each TA process follows this pattern:
    - For each rubric entry:
        - sleep 0.5-1s
        - sem_wait(sem_rubric)
        - read/possibly update rubric
            - write rubric file
        - sem_post(sem_rubric)
- For question marking:

        - loop until all questions are done
        - check if all questions are done
            - if not, claim a random unmarked question
            - sem_post(sem_questions)
            - sleep 1-2s to simulate marking
- TA 0 only (After finishing an exam):
    - sem_wait(sem_loader)
    - load_exam() with the next index
    - sem_post(sem_loader)

Deadlocks and Livelocks:

In all runs with multiple TAs, no deadlock or livelock occurred. This is because each critical section uses only one semaphore, and semaphores are acquired and released in a simple pattern. Also, a TA never holds one semaphore and tries to acquire another inside that same region, avoiding a circular wait.

Discussion of Design:

Mutual Exclusion
Mutual exclusion means that at most one process can be executing inside a critical section that accesses a shared resource at any given time. The three semaphores used ensure this, because there is only one semaphore per critical section, and a TA never tries to acquire another semaphore while holding one.

Progress
Progress means that if no process is in the critical section and one wants to enter, they must be allowed. Our semaphores are initialized to 1, and always use a simple execution pattern. Additionally, no semaphores are held for a long time. This results in no TA ever having to wait a long time to enter a critical section, since each TA in one, will eventually exit theirs.

Bounded Waiting
Bounded waiting limits process starvation by making sure if a process is waiting, there is a limit on how many times other processes can enter the critical section before this process is granted access. In our design, the total work per exam is bounded (there are only 5 rubric entries and 5 questions). After these are handled, TAs move on to the next exam or terminate.
Also, the time a TA spends in a critical section is very short. The rubric update and question-claim operations are constant-time operations that immediately call sem_post() when done.