SYSC 4001: Assignment 3
Part 2c Report
Noor ElShanawany - 101302793
Lina Elayed - 101297993

**Part 2.c: Deadlock, Livelock, and Execution Order Analysis**

1. **Overview and Setup**

This report analyzes the behavior of the concurrent marking system created in Part 2 of the assignment. The goal was to observe whether the system experiences deadlock or livelock and to discuss how the processes execute.

All runs used:

- 3 Teaching Assistants (TAs)
- Exam files from exam_0001.txt to exam_0020.txt
- A termination exam exam_9999.txt
- Each exam file contains one student number
- The system stops when the student number 9999 is loaded

Shared memory stores:

- The current student number
- The current exam index
- A rubric of 5 characters: one per question
- An array tracking question status: unmarked, marking, or done
- A stop flag used to terminate all processes

In Part 2.b, semaphores were added to control access to shared memory.

2. **Behavior of Part 2.a (No Semaphores)**

In Part 2.a, the TAs run at the same time without any synchronization. Each TA:

1. Reviews the rubric and may randomly correct an entry.
2. Selects an unmarked question and marks it.
3. After all questions are marked, one TA loads the next exam.
4. When the student number is 9999, the system stops.

Observations:

During multiple runs:

- The program always reached the termination exam and stopped correctly.
- Race conditions were visible.
- Multiple TAs sometimes marked the same question because there was no protection.
- Rubric values were sometimes updated by different TAs at the same time.

Even with these issues, no deadlock or livelock was observed. The program always continued running and eventually terminated.

Possible Risk:

Even though no deadlock was seen, the design is unsafe because:

- Two or more TAs can change shared data at the same time.
- Exam loading is not protected.
- Question marking is not protected.

This shows why synchronization is needed.

### 3. Behavior of Part 2.b (With Semaphores)

In Part 2.b, semaphores were added to protect all important sections.

The following semaphores were used:

- rubric_mutex: protects rubric changes and file writing
- exam_mutex: protects question status and current student
- loader_mutex: ensures only one TA loads the next exam
- print_mutex:  prevents mixed console output

Observations:

With semaphores:

- Each question is marked only once per student.
- Only one TA loads the next exam.
- Rubric updates happen one at a time.
- Output is clean and readable.
- The program always stops correctly at student 9999.
- All TAs exit cleanly.

No deadlock or livelock occurred in any tested run.

**4. Deadlock and Livelock Analysis**

Part 2.a:

- No deadlock was observed.
- No livelock was observed.
- However, race conditions were common.
- The program is unsafe and could fail if expanded.

Part 2.b:

- No deadlock occurred.
- No livelock occurred.
- Semaphores ensure that:
    - Only one TA modifies the rubric at a time.
    - Only one TA marks a question at a time.
    - Only one TA loads the next exam.
- There is no situation where TAs wait on each other forever.

**5. Execution Order Discussion**

The execution order is different each time the program runs, which is expected in concurrent systems.

- TAs mark questions in different orders.
- Different TAs load different exams depending on timing.
- Rubric corrections happen at random times.
- Regardless, all students always have 5 marked questions.
- The program always moves forward without getting stuck.

In Part 2.b, semaphores control all shared data access, so even though execution order is still unpredictable, the results are always correct.

**6. Conclusion**

- In Part 2.a, the program runs without synchronization. It finishes correctly but contains race conditions and unsafe shared memory access.
- In Part 2.b, semaphores properly control all shared memory access.
- No deadlocks or livelocks were observed in either part.
- The semaphore solution is safe and consistent.