SYSC 4001 - Assignment 3 Report PartPartC: Concurrency Analysis

Student 1: Ajay Uppal
ID: 101308579
Student 2: Paul Felfli
ID: 101299663
Date: December 1 2025
Instructor: Professor Gabriel Wainer

Behaviour of Part 2.a (No Synchronization)

In the unsynchronized version, every TA repeatedly performs three phases: checking the rubric, marking questions, and advancing to the next exam. All of these operations are performed directly on a shared SharedRegion structure without any locks. As a result, the TAs interleave completely freely. Despite this, we did not encounter deadlock or livelock. The program always progressed from one student to the next and eventually reached the sentinel exam (9999). Each TA terminated normally once the sentinel value appeared in shared memory.

What we did see were the characteristic effects of race conditions. Since the rubric is updated by multiple TAs without mutual exclusion, the final letter in each rubric line is non-deterministic — some updates overwrite others depending on the exact timing of each process. A similar issue occurs with question marking. Multiple TAs sometimes read the same question index before any of them set its state to “done,” causing two or more TAs to “finish” the same question. The output logs clearly showed several duplicated markings. A more serious race appears when all five questions for an exam are marked: every TA that simultaneously notices the “all done” condition increments the shared exam index. This can cause certain exams to be skipped entirely. Even though this behaviour is incorrect from a logical standpoint, it does not create a situation where the processes wait on each other in a way that would cause a deadlock or livelock. The system always moved forward and ultimately terminated.

Behaviour of Part 2.b (Semaphore-Protected)

The synchronized version introduces a semaphore set with two semaphores: one dedicated to the rubric and one protecting the exam state (student ID, question states, and exam index). Each shared-memory modification is now wrapped in a critical section. This changes the execution behaviour dramatically.

Rubric corrections now occur one at a time. Each TA must acquire semaphore 1 before changing a rubric line and writing the updated version to rubric.txt. As a result, the rubric evolves consistently, and no corrections are lost or overwritten. Marking questions also becomes deterministic. A TA can only claim a question index while holding semaphore 0, which guarantees that each question is marked exactly once. When all five questions are done, a single TA advances to the next exam because the check and increment occur while exclusively holding semaphore 0. This prevents skipping exams and ensures that student files are processed strictly in order.

No deadlock conditions were observed. The program avoids circular waiting because a TA never holds both semaphores at once. Rubric operations only acquire semaphore 1, whereas all exam operations (marking and loading) use only semaphore 0. There is no scenario where a TA blocks while holding one semaphore and simultaneously waiting for the other, so the system cannot form a cycle of dependencies. The critical sections are short and always end with a matching semaphore release. Similarly, there were no signs of livelock. TAs do not endlessly retry actions that other TAs repeatedly undo; once a question is claimed, it stays claimed, and once a student is finished, the system moves on unambiguously.

The logged output for Part 2.b showed an orderly sequence: each question marked once, each exam loaded by exactly one TA, and a clean termination as soon as the sentinel exam was reached. This contrasts sharply with the behaviour of Part 2.a and confirms that the semaphore structure in Part 2.b effectively enforces mutual exclusion and stable progress.

Conclusion

Our testing shows that Part 2.a does not deadlock or livelock, but its execution is highly inconsistent because all shared memory operations are unsynchronized. Duplicate markings, overwritten rubric entries, and occasional skipped exams appeared routinely. By contrast, Part 2.b enforces proper synchronization, resulting in clean, predictable execution. Every question is processed once, exam order is preserved, and all TAs terminate as soon as the sentinel appears. The absence of circular wait or conflicting resource acquisition prevents deadlock, and the forward progress guaranteed by each critical section avoids livelock. Overall, the semaphore-based solution in Part 2.b satisfies the classic mutual-exclusion, progress, and bounded-waiting requirements, while Part 2.a intentionally violates them.