Part 2(c) – Deadlock / Livelock Analysis

Did the program deadlock or livelock?

In all of my runs with different numbers of TAs (2, 3, 4, and 6), **the program did NOT deadlock or livelock**.

All processes were able to:

1. Review the rubric
2. Mark all 5 questions
3. Load the next exam
4. Terminate cleanly when exam **9999** was reached

No TA became stuck waiting for a semaphore forever, and no infinite spinning loops occurred. Execution always progressed.

Deadlock was avoided because:

1. **Rubric access uses only one semaphore (`sem_rubric`)**
   – All TAs wait → one TA enters → exits → next TA enters.
   – No circular wait.
2. **Exam loading is protected by a single semaphore (`sem_examloader`)**
   – Only one TA tries to load the next exam.
   – Other TAs do not hold any other locks while waiting.
3. **Question marking order is controlled by one semaphore (`sem_question`)**
   – A TA briefly locks this semaphore, chooses an unmarked question, unlocks immediately.
   – No TA holds multiple locks at the same time.

Since **no TA ever waits while holding another semaphore**, the circular-wait condition for deadlock never forms.

Livelock would require processes continually *reacting* to each other without making progress.

Here, each TA:

- Takes a semaphore
- Does the work
- Releases it
- Moves on

There is no logic where processes back off, retry, or continually yield, so livelock does not appear.

Observed Execution Order

Execution is **non-deterministic**, but the pattern is consistent:

1. **All TAs read and optionally update the rubric**
   (one TA at a time inside the rubric-critical section)
2. **TAs mark random questions**
   (question selection synchronized so each question is marked once)
3. **One TA loads the next exam**
   (others wait briefly)
4. **Cycle repeats**
   until the exam with student number **9999** is loaded.

The TAs interleave normally, but always make forward progress.