Part 2(c) – Deadlock/Livelock Analysis

After running my implementations for Part 2(a) and Part 2(b), I did not encounter any deadlock or livelock in any of my runs.

Part 2(a):
Since there were no semaphores, multiple TAs accessed shared memory at the same time. Although race conditions occurred (e.g., multiple TAs marking the same question or modifying the rubric at the same time), the program never froze or became stuck. Execution continued until the exam file with student number 9999 was loaded.

Part 2(b):
The semaphore-based solution also did not produce any deadlock or livelock. The reason is that each critical section is protected by exactly one semaphore:
- SEM_RUBRIC for rubric editing
- SEM_EXAM for exam loading
- SEM_QBASE + q for each question

No circular wait condition ever occurs, because each TA always requests the semaphores in the same order. Therefore, the four conditions required for deadlock do not simultaneously occur.

Execution Order Discussion:
The execution order varied on each run depending on process scheduling. However, the following patterns were consistent:
- Only one TA edited the rubric at a time (Part 2(b)).
- Only TA 0 loaded the next exam.
- No two TAs marked the same question simultaneously.
- When the exam with student number 9999 was loaded, each TA detected it and exited correctly.

Conclusion:
There was no deadlock or livelock in any of the runs for Part 2(a) or Part 2(b). The semaphore usage in Part 2(b) successfully synchronized the processes without causing blocking cycles.