Part 2.c

Both programs from Part 2.a and Part 2.b were executed multiple times using different numbers of TA processes (2, 3, and 4 TAs). The goal was to observe whether any deadlock or livelock conditions occurred, and to analyze the execution order of the processes.

Part2a - No Semaphores

Deadlock / Livelock Observation

During all test runs of Part 2a, no deadlock or livelock occurred.

Why Deadlock Did Not Occur:

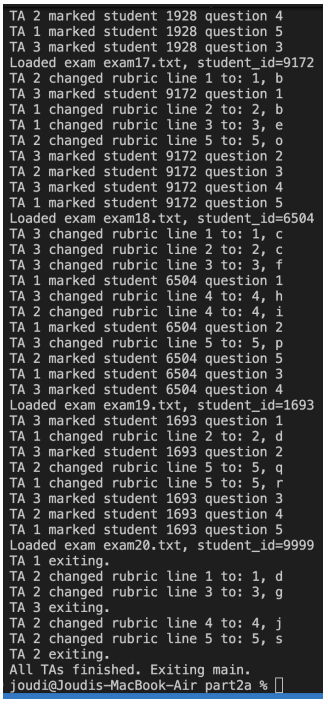
Part 2.a contains race conditions, but it does not enter a deadlock because:

1. **No TA waits for another TA:**
    a. All operations on shared data (rubric updates, question selection) are done without blocking.
2. **At least one TA always finishes all 5 questions.**
    a. The TA that increments questions_done last loads the next exam.
3. **The loop only stops when student ID = 9999.**
    a. Therefore every TA keeps moving forward even if the work distribution is uneven.

Why Livelock Did Not Occur:

Although rubric updates and question marking happen in unpredictable order, the program always progresses because:

1. No TA re-attempts failed actions.
2. Random delays ensure no TA dominates or "starves" the others.
3. Work is always eventually completed.



Part2b - With Semaphores

Deadlock / Livelock Observation

No deadlock or livelock was observed in any run of Part 2b either.

Why Deadlock Did Not Occur:

Part 2.b uses three semaphores:

1. sem_rubric → protects rubric updates
2. sem_question → protects question selection and marking
3. sem_exam_load → protects exam loading

Deadlock is avoided because:

1. **Semaphores are never nested**
    a. TAs never wait while holding another semaphore, so no blocking chains occur.
2. **All semaphore-protected regions are short**
    a. Each TA releases the semaphore quickly, allowing the system to progress.
3. **All TAs follow the same order of operations**
    a. No TA blocks indefinitely on a resource that another TA is waiting for.

Why Livelock Did Not Occur:

1. TAs do not repeatedly retry failed actions.
2. Random sleep intervals ensure progress.
3. One TA always becomes the loader when questions_done == 5.

Execution Order (Since No Deadlock Occurred)

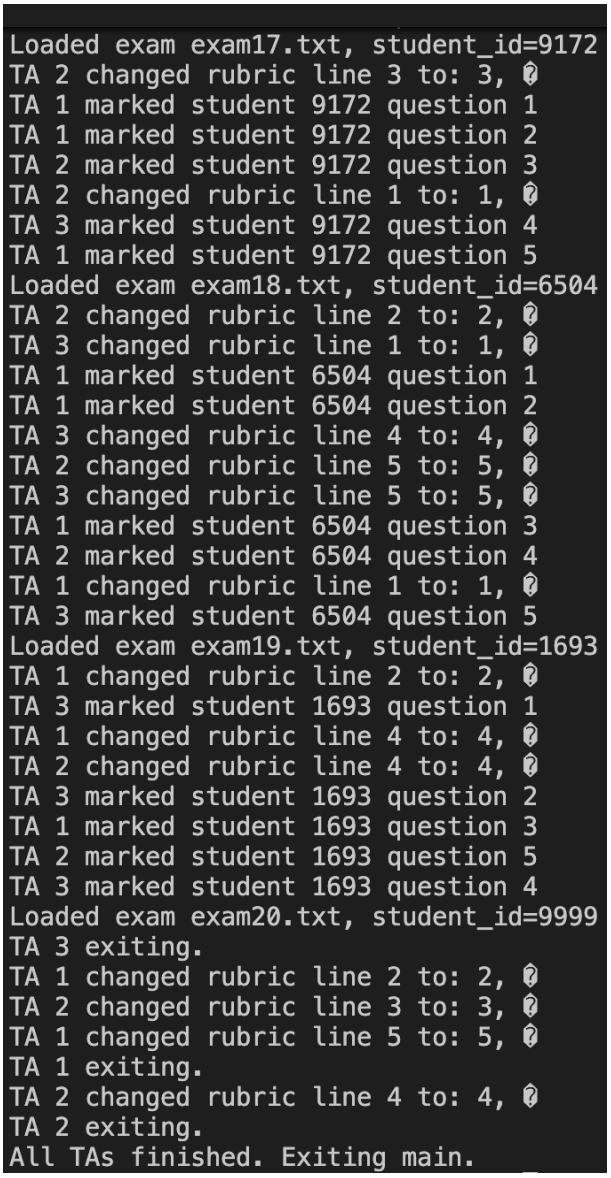
Across both versions, execution order varies between runs due to:

- OS process scheduling
- Independent random delays
- Separate review and marking times

However, the high-level pattern is consistent:

1. All TAs review the rubric (possibly making updates).
2. TAs compete to select and mark questions.
3. The TA that marks the final question loads the next exam.
4. The process repeats until the exam containing student ID 9999 is reached.
5. All TAs exit.

The exact order of rubric modifications and question marking changes in every run, but forward progress always occurs, and all TAs eventually terminate correctly.



Critical Section Requirements

1. **Mutual Exclusion**
    a. **Part 2a (No Semaphores)**
        i. *Mutual exclusion is not guaranteed.*
        ii. *Multiple TAs can simultaneously write to the rubric file and shared rubric memory.*
        iii. *This results in visible race conditions (overlapping writes, corrupted ASCII characters).*
        iv. *Question selection is also unsafe: two TAs may pick the same question.*
    b. **Part 2b (With Semaphores)**
        - Mutual exclusion is enforced using three semaphores:
        i. *sem_rubric: ensures only one TA updates and writes rubric changes.*
        ii. *sem_question: ensures only one TA selects or marks a question at a time.*
        iii. *sem_exam_load: ensures only one TA loads a new exam into shared memory.*

2. **Progress**

   If no TA is in the critical section, one waiting TA must be able to enter it without indefinite delay.

   a. **Part 2a**
      i. *Progress is generally achieved because no TA waits on locks.*
      ii. *However, there is no controlled entry, so updates can interleave unpredictably.*

   b. **Part 2b**
      i. *Semaphores ensure TAs take turns fairly.*
      ii. *No TA blocks forever: as soon as one leaves a critical section, another can proceed.*
      iii. *Since semaphores are not nested, no waiting occurs.*

3. **Bounded Waiting**

   After a TA requests entry to the critical section, there is a limit on how long it must wait.

   a. **Part 2a**
      i. *Because TAs do not wait for access, there is no guarantee of fairness; some TAs may mark many questions while others do very few.*

   b. **Part 2b**
      i. *Semaphores enforce FIFO access at the OS level.*
      ii. *No TA is skipped indefinitely because:*
         1. *Each semaphore is binary (0 or 1).*
         2. *A TA that waits is unblocked as soon as the previous TA posts the semaphore.*