Part C: Deadlock, Livelock and Execution Order

For this part I ran the concurrent TA programs from Part 2(a) and Part 2(b) several times with different numbers of TAs. I watched the output on the terminal and checked the rubric and exam files when needed to see what was happening.

Deadlock and livelock

In all of my tests I did not see a deadlock or a livelock.

- The program from **Part 2(a)** has no semaphores at all. The TAs can update the rubric and mark questions at the same time, which causes race conditions, but they still keep making progress. Exams are loaded one after another and eventually the program reaches the student number 9999, so all TAs exit.

- The program from Part 2(b) uses three semaphores:

    - one for updating the rubric,

    - one for marking questions,

    - one for loading the next exam.

  In this version the TAs sometimes have to wait on a semaphore, but every wait is followed by a post and there is no situation where a TA holds one semaphore and waits forever on another one. Because of that there is no circular wait and I did not see any deadlock. The program always finishes after the 9999 exam is reached. I also did not see livelock. The TAs do real work each time through the loop. They do not just keep changing state without making progress. Rubric lines keep changing, questions get marked, and exams move forward.

Execution order of the processes

Since there is no deadlock, this is how the processes execute.

In **Part 2(a)**, where there are no semaphores, the order is very chaotic:

- All TAs print "reviewing rubric for student X" close together.

- Different TAs update the same rubric line almost at the same time, so the letter jumps quickly (for example from A to D or E).

- Multiple TAs may try to mark questions in a way that overlaps, so the print statements for marking are mixed together.

- Any TA who finishes marking and sees that all questions are marked can load the next exam. There is no control, so several TAs may try to load the exam nearly at once, but the program still moves forward.

In **Part 2(b)**, the semaphores make the execution order more structured:

- When a TA wants to change the rubric, it first waits on sem_rubric. This means only one TA updates and writes the rubric at a time. Other TAs have to wait their turn, so rubric changes appear in a cleaner sequence.

- When TAs pick a question to mark, they use sem_mark around the marked[] array. This prevents two TAs from picking the same question. The printout shows each question number being marked once for each exam.

- When all questions are marked, the TAs use sem_exam to control loading the next exam. Only one TA opens the next file and updates currentStudent and examNumber, then resets marked[]. The other TAs see the new student and continue from there.

The exact order of prints still changes between runs, because the operating system scheduler decides which process runs first, and the delays use random values. However, in every run I checked:

- Questions eventually all get marked for each exam

- Exams advance to the next file,

- The program finishes when student 9999 is reached,

- There is no point where all TAs are stuck forever.

So my conclusion for Part C is that, for the implementations I used in Part 2(a) and Part 2(b), no deadlock or livelock occurred, and the execution order is controlled by the semaphores in Part 2(b) so that progress is always made.