Report Part C

Chikezilim Afulukwe - 101279214
Abubakr Mohammed - 101287262

After running my implementations for part 2(a) and 2(b), I didn't encounter any deadlock or livelock in any of my runs.

Part 2(a):

Multiple TAs accessed shared memory simultaneously since there were no semaphores. The program never froze or became stuck, despite racing conditions occurring (such as numerous TAs marking the same question or changing the rubric at the same time). Execution continued until the file with student number 9999 was loaded.

Part 2(b): There was no deadlock or livelock produced by the semaphore-based method either. The reason is that there is precisely one semaphore protecting each critical section:

- SEM_QBASE + q for every question;
- SEM_EXAM for loading the exam;
- SEM_RUBRIC for changing/editing the rubric

Since every TA always requests the semaphores in the same order, there is never a circular wait condition. Consequently, the four circumstances necessary for deadlock do not occur at the same time.

Discussion of Execution Order: Depending on process scheduling, the execution order changed with each run. Nonetheless, the following trends held true:
The rubric was only changed by one TA at a time (Part 2(b)).
The following exam was only loaded by TA 0.
No two TAs marked the same question at the same time.
- Each TA recognized and successfully exited the exam with student number 9999 when it was loaded.
In conclusion, neither Part 2(a) nor Part 2(b) runs experienced a deadlock or livelock. In Part 2(b), the semaphore was used to successfully synchronize the processes without creating blocking cycles.