Jesse Handa 101264747

Part 2(c) – Deadlock, Livelock and Execution Order

1. Overview and Test Setup

For Assignment 3 Part 2 I implemented a concurrent TA marking system with shared memory. Multiple TA's share a *SharedData* structure which contains:

- *currentStudent* – student # for current exam
- *currentExamIndex* – index of the current exam file
- *rubric[5]* – one character per question
- *questionStatus[5]* – 0 = unmarked, 1 = marking, 2 = done
- *stopFlag* – tells all TAs when to terminate

In Part 2(b) I used semaphores in shared memory:

- *rubric_mutex* – protects rubric updates and *rubric.txt*
- *exam_mutex* – protects *questionStatus[]* and *currentStudent*
- *loader_mutex* – ensures only one TA loads the next exam
- *print_mutex* – avoids interleaved *cout* output

To test I used:
- 3 and 5 TA's
- Exams: *exams/exam_0001.txt* … *exams/exam_0050.txt*
- The termination file *exams/exam_9999.txt* with student number 9999

It stops when *currentStudent* == *9999*.

2. Behaviour of Part 2(a) – No Synchronization

In Part 2(a) there are no semaphores, so each TA:

1. **Reviews and edits the rubric**
    - Sleeps 0.5–1s per question.
    - With 25% probability (rand() % 4 == 0), increments the stored letter.
    - Immediately writes the whole rubric back to *rubric.txt*.
2. **Marks a question**
    - Checks *questionStatus[]* for 0, changes to 1, sleeps for 1–2s, then changes it to 2.
    - Prints which question and which student it marked.
3. **Loads the next exam**
    - If all five questions are at state 2, the TA increments *currentExamIndex* and calls *loadExam()*.
    - *loadExam()* loads *exam_0001*, *exam_0002*, … up to *exam_0050*.
    - When the index reaches 50 it switches to *exam_9999.txt*, which sets *currentStudent* to 9999.

Observed behaviour

- The program always reaches student 9999 eventually and has the TA's exit when *stopFlag* is toggled.
- Race conditions:
    - Two TAs update the same rubric entry at essentially the exact same time, so the rubric letters update unpredictably.
    - Different TAs could choose the same question (since it's just checking questionStatus = 0), so the same question could be graded multiple times in output.
    - More than one TA says the exam is done and tries to advance to the next exam.

Even with these race conditions, everything still works and it does not get stuck:

- Exams keep advancing through the list.
- Question status keeps going from 0 to 1 to 2.
- The final exam (9999) is always reached.

So for Part 2(a) I didn't observe any deadlock or livelock whatsoever: the behavior was inconsistent/unpredictable but the program always finished.

3. Behaviour of Part 2(b) – With Semaphores

In Part 2(b), most of the program is the same, like the functionality, but we added semaphores here to protect all shared data.

Rubric

We wrapped rubric changes with *rubric_mutex*:

- *sem_wait(&rubric_mutex);*
    - TA would read/increments *rubric[i]*, then print using *print_mutex*, and finally call *saveRubric()*.
- *sem_post(&rubric_mutex);*

This was to ensure only one TA can edit the rubric or write to *rubric.txt* at a time, this makes sure rubric updates are consistent.

Question marking

Choosing and grading a question was protected by *exam_mutex*:

- TA would lock *exam_mutex*, find an ungraded question (*questionStatus[i] == 0)*, set its status to 1 and copy *currentStudent*.
- After the marking delay, it locks the *exam_mutex* and sets *questionStatus[i] = 2* (completed).

This results in each question being assigned to only one TA, and the marking results no longer being inconsistent/duplicated.

Exam loading

When a TA thinks an exam is finished:

1. Checks *questionStatus[]* within the *exam_mutex*.
2. All are 2? Then lock the *loader_mutex* and make it the only TA allowed to load the next exam.
3. In the loader section it checks under *exam_mutex* (again), and then increments *currentExamIndex* and calls *loadExam()*.

4. If *currentStudent == 9999 (last student)*, it'll toggle to stop (*stopFlag = 1)* and print that it terminated.

This is to prevent skipping exams, as only one TA can see a finished exam AND go to the next one.
Results
With 3 and 5 TAs:

- All questions are marked once per exam
- Exams are loaded in correct order (1 … 50, then jumps to 9999).
- Rubric output is neat/serialized
- All TAs terminate cleanly after the final exam (9999).

4. Deadlock and Livelock

Since there's no locks in Part 2(a), TAs never wait on each other. They can overwrite each other's data, but they will always keep running. Since no TA can ever be blocked waiting on a semaphore, deadlock should be impossible. Similar to livelock, the exams and questions should always continue progressing and the system always reaches the final exam (9999).

In Part 2(b), four semaphores were used in a way that avoids deadlock and livelock:

- TAs shouldn't have have more than one lock in an order that causes issues:
    - For rubric: *rubric_mutex* then *print_mutex*.
    - For exams: sometimes *exam_mutex* alone, or *loader_mutex* then *exam_mutex*.

- Nowhere in the code should one TA lock *exam_mutex* and then wait for *loader_mutex* while another holds *loader_mutex* and waits for *exam_mutex*.

- Every *sem_wait* should have a *sem_post* soon after, and all of these critical sections should be short.

There should be no circular waits, so that no deadlocks cannot occur.

Livelock also doesn't happen. Since in each loop, a TA will either update the rubric, mark a question, or load the next exam. Exams always advance toward final exam (9999), and once its is loaded, *stopFlag* forces all TAs to exit. There is no endless switching between states.

5. Execution Order (Summary)

The process interactions between TA actions depends on the OS scheduler and random sleeps. With more TAs, there is more interactions, but:

- **Part 2(a)**: output is messy. Rubric messages change unpredictably, questions are "marked" by multiple TAs, and many TAs can attempt to go to the next exam at basically the exact same time.
- **Part 2(b)**: semaphores make everything cleaner. Rubric changes are shown one at a time, each question gets marked once, only one TA will load every new exam, and it always finishes when it reaches the final student (9999).

In both versions I didn't find any deadlock or livelock, but to be sure, only the synchronized version truly guarantees the functionality is correct.