With the addition of semaphores, our program does not have any livelocking or deadlocking.

TA_Constructor:

The execution of the program starts with TA_Constructor being called with a positive integer argument. TA_Constructor then creates all the shared memory structures along with the set of semaphores. It also attaches and assigns values to some of the shared memory structures and initializes and sets the semaphore set. It then enters a loop to create the requested number of TAs using the fork/execute method. As arguments it passes the ID of the TA to differentiate them, as well as all the shared memory keys and the semaphore ID. It then enters a wait loop until all its children have completed and finishes by cleaning up the semaphore set.

TA_Body:

Using the keys given by TA_Constructor, TA_Body attaches all the shared memory structures as well as saving the TA id and semaphore id to local variables. TA_Body then contains some initialization code. Using a semaphore, only one TA enters this code block. This code block opens the first student file and saves the student number both to a local variable and to a shared memory space for the other TAs. It also opens the rubric and saves its contents to another shared memory space. This code block is never run again by any TA during the remainder of the execution. The TAs now enter main loop phase. This begins with checking the rubric. Whenever a TA decides to update the rubric, it first makes the edit to the version of the rubric that is stored in the shared memory space and then saves that to the text file. A semaphore protects both shared memory space and the file editing. Afterwards, the TAs begin to correct the questions. The programs use an array of 5 integers to keep track of which questions have been graded, are being graded or are untouched. This array is saved to a shared memory space. A 0 signifies that the question is open, a 1 signifies that it is in the process of being graded and 2 means that it has been graded. This array is protected by a semaphore. Once the array has no more 0s the TAs move to waiting loop. This loop checks to see if all the questions are fully graded (array all 2s). If there are still 1s, the TAs just continue in the waiting loop. If the array is full of 2s, a single TA announces the student has been fully graded and opens a new student file, sharing the student number through the shared memory space as well as saving it locally. It also resets the question array. When the other TAs determine that the array has been reset, they update their local version of the student number using version saved in the shared memory space and return to the beginning of the main loop. A semaphore is used to protect the reading and editing of the question array during this phase. At the end of each main loop, each TA checks if they have collectively graded all the students. If they have, they break out of the main loop and exit.

Critical Sections:

1. Checking for initialization block:

As counter is in a shared memory space and initialized to 0 by TA_Constructor it is used to check for this. A TA that is about to check the branch logic locks the relevant semaphore. It then checks the branch.

If counter is 0, it enters the branch, sets counter to 1 and proceeds with the initialization steps mentioned earlier. It then unlocks the relevant semaphore.

If it doesn’t take the branch (counter > 0) it immediately unlocks the relevant semaphore and moves on.

2. Rubric edit:

Using rand() % 100 a rubric edit event a 1/100 chance of happening. If this does occur, the TA locks the relevant semaphore. It then edits the version of the rubric that exists in shared memory. It the uses this version to update the actual text file. It the unlocks the relevant semaphore.

3. Question array:

A five-integer array is used to keep track of the completion progress of the exam questions. When a TA arrives at this section it does the following. It loops five times, first locking the relevant semaphore. It then checks if the element of the array is 0.

If it is it sets it to 1 and unlocks the relevant semaphore and proceeds to grade the question and then continue the loop.

If it isn’t it unlocks the relevant semaphore and continues the loop.

4. Completion check (minor loop):

The same array is used to check if all questions have been fully graded. A TA locks the relevant semaphore and then sums all elements of the array. It then uses conditionals to decide what to do.

If the sum is 10, all questions have been answered. The TA resets the array. It then increments the counter, announces that the student has been graded and the opens the next student file. Afterwards, it saves the new student number to the shared memory space and then unlocks the relevant semaphore. Finally, it saves the number to its own local variable, exits the minor loop and continues the main one.

If the sum is between 5 and 9 (inclusive), some questions are still in progress. It unlocks the relevant semaphore and continues minor loop.

If the sum is less than 5, the array has been reset, and the TA must move on to the new exam. It updates the local version of the student number with the shared memory version and unlocks the relevant semaphore. It then exits the minor loop to continue the main one.