Part 2: Report

James Bian, Yuvraj Bains
101311339, 101311131
Carleton University
Dec 1 2025

In part.a, the processes sometimes interfered with each other (e.g., two TAs occasionally tried to mark the same question or update the rubric at roughly the same time), but the system always continued to make progress and eventually reached the exam with student number 9999, after which all processes terminated normally. In part b, the added synchronization prevented race conditions on the shared data, and the system also always progressed to the final exam without any process getting stuck. Each TA is an independent process created with fork(). They all share memory, but the CPU determines which process runs at what time. The parent calls fork() n times to create n TA processes, which begin executing their rubric review. In part a, Two or more may modify the same rubric character almost simultaneously, which means rubric.txt may be rewritten by multiple processes all on top of each other. In part b, however, rubric review is protected by the mutex; only one TA can edit at a time, creating a strict order for access. When marking, all TAs scan question_state[] without protection, so two TAs may choose the same question before either sets it to 2. This leads to a lot of overlap between TA work. In Part B, the question_mutex enforces orderly assignment. TAs will find the first available question, locking the mutex until marking is done. A random sleep is done for marking, before the question is set to 2, and it is locked again. This ensures that no 2 TA's mark concurrently. After the questions are marked, or all question states are 2, index is incremented and question state is reset. If student 9999 is detected, all TA's complete the loop they are currently on before terminating, and all_done is set. In Part b of the assignment, semaphores were added to control access to shared memory during rubric correction, question selection, and exam loading. This solution also addressed the three critical section problems. Mutual Exclusion was introduced in part b, with the addition of the rubric mutex, only one TA can mark or review the rubric, which prevented TAs writing over each other. The exclusive lock allows only one process in the critical section, and ensures that no shared resources are updated simultaneously. Progress prevention is fixed with sem_wait and sem_post, which allows another TA to immediately continue on with work when finished using a critical section. No TA is allowed to sleep while using a critical section. Lastly, Bounded Waiting is solved with a FIFO. TAs must wait on the rubric mutex, question mutex, and exam mutex fairly without cutting one another, so that no processes are left waiting indefinitely. Every TA attempting to enter a protected region eventually succeeds after a bounded number of other entries.