SYSC4001 Assignment 3 Part 2c

Zachary Gallant 101272210
Jared St. Louis 101236784

Deadlock

Deadlock happens when two processes each hold a resource and wait forever for the other to release theirs. In part2b_deadlock.c, I purposely created this situation:

- TA 1 grabs rubric_sem then waits for questions_sem.
- TA 2 grabs questions_sem then waits for rubric_sem.

Both TAs end up stuck waiting for each other. This confirms a real deadlock scenario.

In our actual synchronized program (part2b.c), this never happens because I keep the locking order consistent and only hold one semaphore at a time, which breaks the circular wait condition.

Livelock

Livelock is different: the processes are not stuck waiting. They are actively running, but never make progress. In part2b_livelock.c, both TAs keep releasing and retrying their locks at the same time, trying to be "polite," and they just bounce back and forth.

The output keeps printing attempts, but no real work happens, which confirms a livelock.

Our real part2b.c avoids this by not backing off unnecessarily and by letting only one TA handle exam transitions at a time using exam_sem. That guarantees progress.

Comparison

Case	**What Happens**	**What was seen in my demos**
Deadlock	Both TAs wait forever	Program freezes
Livelock	Both TAs keep running but make no progress	Repeated attempts, no work done
Part2b.c	Avoids both	Runs cleanly from first exam to student 9999

Execution Order of the Real Program (part2b.c)

The synchronized version runs in a predictable and clean order:

1. All TAs start, load the rubric from shared memory, and print what they are doing.

2. A TA enters the rubric section one at a time (protected by rubric_sem).

3. TAs pick and mark questions, but only one TA modifies questions_marked[] at a time (protected by questions_sem).

4. When all questions for an exam are done, exactly one TA loads the next exam because exam_sem prevents multiple TAs from doing it.

5. This repeats until the exam with student 9999 is reached.

6. Once that happens, the finished flag is set and all TAs exit in an orderly way.

This shows that the synchronization in part2b.c guarantees forward progress and avoids both deadlock and livelock.

Why Our Final Part2b Code Works

- Only one lock is held at a time
- Locking order is always consistent
- Only one TA loads the next exam (protected by exam_sem)
- Critical sections are small
- Conditions are checked inside the locks to avoid bouncing behaviour

This makes sure the TAs always make progress and don't block each other.