SYSC 4001

Assignment 3: Part 2

Saim Hashmi (101241041)
Abdullah Salman (101282570)
Link to Part 2: click here

Design Discussion: Critical Section Solution

This discussion explains how the design of the concurrent exam marking system addresses the three requirements of the critical section problem: mutual exclusion, progress and waiting.

Mutual Exclusion

In the system, TAs and the parent process share several resources such as:

- The rubric for 5 questions
- The per question state array ("not started", "being marked" and "done")
- The exam index and termination flag
- Also the logging output

To prevent multiple processes from updating these resources simultaneously , we used semaphores which is used in Part 2.b:

- To protect rubric and the flag that indicates whether the rubric needs writing to the file.
- Ensures that only one TA marks a given question at a time and updates the exam completion status.
- It is also to prevent output from multiple TAs from interleaving and maintains readable logs.
- Also to use parent process to signal all TAs when a new exam is loaded.

Implementation:

- Whenever a TA or the parent needs to modify shared data, it first calls sem_wait() on the corresponding semaphore performs a brief operation (for e.g. claim a question, update rubric or write a log line) and then calls sem_post() to release the semaphore.
- This guarantees that no two processes can modify the same resource simultaneously, achieving mutual exclusion.

Progress:

Progress ensures that if one or more processes want to enter a critical section and it is free some process will be able to enter without indefinite delay.

Critical section in this system are deliberately kept short for example:

- Reading or updating a single rubric entry.
- Claiming a question from the question state array
- Printing a single log line

Longer operations such as marking a question (simulated with a delay) are performed outside the critical section.

The effect due to this is no TA holds a semaphore while performing long work, so other TAs can still access shared resources promptly. The parent periodically checks exam completion and signals TAs without spinning or blocking. As a result all processes make progress are n critical sections.

Waiting:

Waiting requires that every process requesting access to a critical section will be allowed entry after a finite number of other processes have entered. This ensures that each semaphore protects a single shared structure (rubric, question and log). Critical sections are short and determined and each TA releases the semaphore immediately after the operation. There is no locking preventing starvation.

The effect due to this is each TA gets a chance to access the rubric or mark a question within a predictable amount of time. Even with 2,3, or 4 TAs no process can hold a resource, and all TAs eventually complete their tasks.

Observations Across different number of TAs:

- **2 TAs:** Fewer race conditions, execution order is simpler, but the critical section still arranged in serial order to access.
- **3 TAs:** Slightly more interleaving, semaphores prevent conflicts and marking order is non determined but consistent.
- **4 TAs:** More potential for concurrent attempts to access resources but semaphores still enforce safe execution.

In all cases no deadlock or livelock occurred, and the system reliably completed all exams including the last exam for student no. 9999.