Assignment 3 - SYSC 4001 PART II (Concurrency Report)

Aryan Kumar Singh (101299776)
Srivathsan Murali (101287534)

Part A – Unsynchronised Marking System (Race Conditions)

Overview

Part A implements a marking system using shared memory without a synchronization mechanism in place. Multiple TAs correct concurrently and the access shared between the data structures such as questions_marked, current_exam_index, and current_student_id stay without protection.

This causes race conditions to demonstrate the consequences of uncontrolled concurrent access. (Race conditions are the fault in a system where the result is based on unpredictability of the multiple processes or threads that are running based on the shared data structures)

Observed Behaviour

When running Part A with multiple TAs (e.g., 2 or more), the termination behavior was realized as whenever student ID 9999 appeared in exam 20 or earlier, no matter its placement all TAs would soon cease marking. However, part A was riddled with race conditions with the following issues being consistently observed:

- Duplicate marking of questions: Two TAs often marked the same question for the same student.
- Inconsistent completion logs: Messages such as "Completed marking all questions" appeared even when other questions were still unmarked.
- Non-sequential exam processing: Exams were sometimes skipped or loaded multiple times.

This behaviour clearly demonstrates that without synchronization, order of process execution is unpredictable and dependent solely on the OS scheduler.

Timing Observations

Part A displayed only think-time delays (rubric analysis) and did not reliably reflect actual marking duration per question. This further emphasizes the unsuitability of the system for performance measurement since:

- Marking logs overlapped
- Completion times were inconsistent
- No guaranteed order of execution existed

Conclusion (Part A)

Part A successfully illustrates:

- The dangers of race conditions
- Data inconsistency in parallel shared memory systems
- The necessity of synchronization primitives such as semaphores

This uncontrolled environment is unsuitable for accurate grading or performance analysis.

Part B – Synchronised Marking System (Semaphore Controlled)

Overview

Part B introduces semaphores to control access to critical sections and prevent race conditions through coordinated access. An array of three semaphores were used:

- SEM_SHARED – Protects shared exam data
- SEM_QUESTIONS – Controls access to marking logic
- SEM_RUBRIC – Ensures exclusive rubric modification

Observed Behaviour

The critical section problem is the challenge in concurrent programming of managing access to shared resources by multiple processes to prevent data inconsistencies or race conditions. Our design solves the challenge through incorporating three requirements: mutual exclusion, progress, and bounded waiting.

Our solution ensures mutual exclusion through a tiered semaphore architecture tailored to each shared resource. For rubric modifications, a binary semaphore guarantees exclusive access, preventing concurrent writes that could corrupt assessment criteria. Question marking employs an array of semaphores (one per question), allowing parallel marking across different questions while serializing access to each specific question. Exam loading is protected by an additional binary semaphore, ensuring only one Teaching Assistant transitions between exams. This structured approach isolates critical sections effectively, maintaining data integrity across all concurrent operations. (NOTE: While we did testing with random sequences of exams being assigned. When asking the professor on whether the code should load exams sequentially or randomly, he stated it does not matter.)

To satisfy the progress requirement, our solution employs blocking semaphore operations with sem_wait() and sem_signal() (equivalent to classic P() and V() operations) to manage critical sections. While this ensures mutual exclusion, the progress requirement is partially addressed through structured access patterns rather than non-blocking mechanisms. Teaching Assistants follow a sequential workflow—first checking the rubric, then marking questions—which naturally distributes access opportunities. However, potential progress issues exist: TAs may encounter

indefinite blocking if semaphore release fails, and the continuous loop structure with fixed delays introduces periodic rather than event-driven progression. Resource release is enforced through disciplined sem_signal() calls in all code paths, preventing abandoned locks but not eliminating potential deadlock scenarios from circular wait conditions.

Our solution addresses bounded waiting through a sequential examination approach rather than randomized selection. Each Teaching Assistant systematically checks all questions in order during rubric review (lines 1-5), ensuring every question receives consideration. For question marking, TAs employ a linear search algorithm that scans questions sequentially from first to last, guaranteeing each unmarked question will eventually be discovered. While this deterministic approach prevents indefinite starvation, it introduces potential fairness issues: early questions may receive more frequent access than later ones. The implementation includes bounded retry loops with fixed iteration limits (RUBRIC_SIZE attempts) and a termination upon seeing a student id of 9999 to prevent infinite search loops.

The synchronization framework implements a three-tier semaphore system managing distinct resource types. Binary semaphores control singular resources: SEM_RUBRIC serializes rubric file modifications, while SEM_SHARED regulates access to exam state transitions and loading operations. A unified SEM_QUESTIONS semaphore provides broad protection for question-related activities, though this creates potential contention points where rubric checking blocks question marking unnecessarily. Shared memory management follows proper lifecycle protocols with shmget(), shmat(), and shmdt() calls, and the system includes comprehensive cleanup routines that remove both shared memory segments (IPC_RMID) and semaphore sets upon completion.

Under synchronization the following improvements were evident:

- Each question was marked exactly once
- Exams were processed sequentially
- No overlapping writes or data corruption occurred
- Student IDs remained consistent throughout marking
- Correct handling of termination condition (student ID = 9999)

Timing Results

In Part B, marking duration was accurately observable through structured logs. Each TA completed marking cycles that corresponded to controlled sleep intervals, making the process measurable and repeatable.

Example observation:

- Average marking time per question: ~1.5 seconds
- Think-time averaged between 0.5–1.0 seconds
- Total exam duration remained consistent across runs

Comparison between Part A and B

Feature	Part A	Part B
Race Conditions	Present	Eliminated
Marking Accuracy	Unreliable	Deterministic
Timing Precision	Poor	Consistent
Data Safety	Compromised	Guaranteed
System Reliability	Low	High

Conclusion (Part B)

Part B demonstrates how synchronization restores system correctness, predictability, and fairness. The use of semaphores ensures critical sections are respected and that shared memory integrity is preserved.

This version is suitable for real-world concurrent systems where correctness and timing accuracy are mandatory.

Final Summary

This assignment clearly demonstrates the central role of synchronization in operating systems. While Part A shows the chaotic consequences of concurrent access, Part B provides a robust and controlled solution using semaphores. The contrast reinforces theoretical operating system concepts with practical implementation evidence.