SYSC4001 – Operating Systems

Assignment 3 – Part 2: Concurrent Exam Marking System

Gabrielle Farah (101296153)

Jumana Mahmoud(101295239)

2025-12-01

1. Introduction

The goal of this assignment is to implement a concurrent exam marking system where multiple Teaching Assistants (TAs) simultaneously mark exams. The system uses shared memory to store exam data and semaphores to ensure safe concurrent access.

Two versions of the system are implemented:

- Part 2A: Concurrent marking without synchronization (potential race conditions).
- Part 2B: Concurrent marking with synchronization using System V semaphores to prevent conflicts in shared resources.

The system simulates marking of 20 exams, each with 5 questions, while occasionally allowing TAs to update the rubric.

2. System Design

2.1 Shared Memory Structure

Shared memory is used to store global data accessible by all TAs:

Field	Type	Description
rubric[NUM_QUESTIONS][100]	char array	Marks and rules for each question

current_exam	int	Tracks the exam currently being marked
questions_marked[NUM_QUESTIONS]	int array	Tracks which questions have been marked
exam_content[100]	char array	Stores student ID or exam content

This memory is allocated using shmget and attached by each process using shmat. After all TAs finish, the memory is cleaned up with shmdt and shmctl.

2.2 Synchronization (Part 2B)

To ensure safe concurrent access:

- Semaphores used:

Semaphore	Index	Purpose
RUBRIC_SEM	0	Protects rubric modification
EXAM_LOAD_SEM	1	Controls loading and access to current exam
QUESTION_SEM[i]	2+i	Protects each question from simultaneous marking

- Operations:
    - semaphore_wait(semid, sem_num) → Wait (P operation)
    - semaphore_signal(semid, sem_num) → Signal (V operation)

This prevents race conditions where:

1. Multiple TAs modify the rubric simultaneously.
2. Multiple TAs mark the same question concurrently.

3. Exam loading overlaps between TAs.

2.3 Process Management

- Each TA is represented by a separate process created using fork().
- Child processes execute ta_process(), which:
    1. Reads the current exam.
    2. Reviews and optionally updates the rubric (20% chance per question).
    3. Marks each question with a simulated delay (1–2 seconds).
    4. Loads the next exam when all questions are marked (synchronized with EXAM_LOAD_SEM).
- The parent process waits for all TAs to finish using wait(NULL).

2.4 Rubric Modification

TAs occasionally correct the rubric:

1. Find the comma in the rubric string.
2. Increment the character after the comma (simulating a score adjustment).
3. Save the updated rubric to rubric.txt while holding RUBRIC_SEM.

2.5 Termination Condition

- Exams with student ID "9999" signal TAs to stop.
- current_exam is set to MAX_EXAMS + 1 to ensure all TAs terminate.

3. Implementation Details

3.1 Compilation

```
gcc -o marking_system_partA marking_system_partA.c
gcc -o marking_system_partB marking_system_partB.c
```

3.2 Execution

```
./marking_system_partA 4   # Part 2A: no semaphores
./marking_system_partB 4   # Part 2B: with semaphores
```

3.3 File Structure

- rubric.txt → Contains 5 question rubrics.
- exam_01.txt … exam_20.txt → Each exam contains student ID and answers.
- Each TA marks different questions simultaneously without conflict.
- Rubric updates are logged with semaphore protection.

6. Conclusion

- Part 2A demonstrates concurrent marking without synchronization, which may lead to race conditions.
- Part 2B ensures safe concurrent access using System V semaphores, avoiding conflicts in:
    - Rubric modification
    - Question marking
    - Exam loading
- Shared memory efficiently stores exam data accessible by all TAs.
- The system simulates realistic marking delays and allows multiple TAs to work concurrently.