SYSC 4001 A: Operating Systems | Fall 2025

Carleton University

Assignment 3 — Part II: Concurrent TA Processes in Unix

Ozan Kaya • 101322055

Nathaniel Babyak • 101310590

Table of Contents

About

This project demonstrates the challenges and solutions of concurrent programming in a simulated environment where N Teaching Assistant (TA) processes mark student exams. The TAs share access to an **Exam Marking Status** array in shared memory and a persistent **Rubric file** (`rubric.txt`).

- **Part 2.a (Unsynchronized):** The implementation uses shared memory but no synchronization, highlighting **race conditions**, **lost updates**, and **data corruption**.
- **Part 2.b (Synchronized):** The implementation utilizes **System V Semaphores** to enforce **mutual exclusion** over three critical sections, ensuring data integrity and efficient, non-overlapping work distribution.

Structure

The following files and directories are included in the repository:

```
.
├── docs/
│   └── reportPartC.pdf          # Comparative analysis report (Part 2.c)
├── exams/                       # Contains all exam_xxxx.txt files, including the termination file
├── src/
│   ├── part2a_101322055_101310590.cpp # Unsynchronized solution (Races)
│   └── part2b_101322055_101310590.cpp # Synchronized solution (Semaphores)
├── .gitignore
├── Makefile                     # Build and run automation
└── rubric.txt                   # Shared, persistent file (input/output), pre-initialized
```

Getting Started

Prerequisites

- **C++ Compiler:** `g++` is required (C++11 standard or newer).
- **Unix/Linux System V IPC:** Support for System V Shared Memory (`shmget` , `shmat`) and Semaphores (`semget` , `semop`) is mandatory.

Note: The necessary input files (`rubric.txt` and the files within the `exams/` directory) are **already included** in this repository and do not need to be manually created.

Running the Program

1. Compile Project

Use the provided `Makefile` to compile both versions of the program. This command will create the `bin/` directory and place the executables inside.

```
make all
```

2. Execution

The program takes one command-line argument, `X`, which is the number of concurrent TA processes (must be $X \geq 2$).

A. Run Part 2.a (Unsynchronized)

This run demonstrates the race conditions and data corruption.

```
# Example: Run with 4 concurrent TAs
make run_a X=4
```

B. Run Part 2.b (Synchronized)

This run verifies the successful mutual exclusion and data integrity.

```
# Example: Run with 4 concurrent TAs
make run_b X=4
```

3. IPC Cleanup

If the program is terminated prematurely (e.g., using `Ctrl+C`), the shared memory and semaphores might remain in the system. If you encounter errors like "shmget failed," manually clean up the resources:

1. **List Resources:**

   ```
   ipcs -m // List Shared Memory Segments
   ipcs -s // List Semaphores
   ```

2. **Remove Resources:** Use the specific IDs returned by the previous commands.

   ```
   # Replace <SHM_ID> and <SEM_ID> with the specific IDs
   ipcrm -m <SHM_ID>
   ipcrm -s <SEM_ID>
   ```