Lab #1 - Simulating Fair-share Process Scheduling

COEN 346

WINTER 2025

March 11th, 2025

Authors:
Mik Driver (40244456)
Victor Depaz (40242703)

Concordia University

1. HIGH LEVEL DESCRIPTION OF CODE

Overview

The purpose of this program is to simulate a fair-share process scheduler. It distributes CPU time fairly among processes owned by multiple users using a time quantum-based approach. The program tracks and logs the state changes of processes (such as start, resume, pause, and finish) to an output file (output.txt) in a thread-safe manner using the EventLogger class.
Programming language: C++

Key Libraries:

- <iostream>: Used for console input and output.
- <vector>: Provides a dynamic array container to store users and processes.
- <thread>: Enables multithreading capabilities for simulating concurrent process execution.
- <chrono>: Used for time-based operations, such as sleeping for a specified duration.
- <algorithm>: Provides utility functions like std::min for efficient computations.
- <mutex>: Ensures thread-safe access to shared resources, such as the log file.

Key Functions & Components

EventLogger Class:

This class logs process events to a file in a thread-safe manner. It opens and manages the log file, ensuring synchronization using a mutex, and writes events immediately via flush().

Process Class:

Represents a process in the system. It manages attributes such as process ID, state, remaining time, and thread control flags (e.g., should_run, terminate_thread). It also handles the lifecycle of process threads (start, stop, run) and simulates execution.

User Class:

Represents a user managing multiple processes. It supports adding processes, starting/stopping threads, checking if all processes are finished, and providing a list of ready processes for the scheduler.

Scheduler Class:

The core of the program, the Scheduler, manages process execution, distributing CPU time fairly among users and their processes. It uses a time quantum approach to allocate CPU time and logs state changes of processes using the EventLogger.

Key Attributes:

- time_quantum: The total time quantum allocated for each scheduling cycle.
- current_time: Tracks the current simulation time.
- logger: An object of the EventLogger class for logging process events.
- users: A std::vector of User objects managed by the scheduler.
- cv: A std::condition_variable for synchronizing threads.

Key Methods:

- Scheduler(int quantum, const std::string& log_filename) (Constructor): Initializes the scheduler with the given time quantum and log file name.
- add_user(std::unique_ptr<User> user): Adds a user to the scheduler's list of users.
- run_scheduler(): Starts all process threads and runs the scheduling loop until all processes are finished. Stops all process threads after scheduling is complete.
- all_processes_finished(): Checks if all processes for all users have finished execution.
- distribute_quantum(): Distributes the time quantum among users and their processes in a round-robin fashion. Logs process events (e.g., started, resumed, paused, finished) using the EventLogger.

Main Function:

The main function initializes the scheduler, creates users and processes, and simulates process execution. It reads process details from input.txt and logs events to output.txt.

Flow:

1. Reads input and initializes the scheduler.
2. Adds users and processes to the scheduler.
3. Runs the scheduler to simulate process execution.

Key Features:

- Uses file I/O to read input and write output.
- Demonstrates the interaction between Scheduler, User, and Process classes.

2. CONCLUSION

The program implements a fair-share process scheduler that ensures fair distribution of CPU time among multiple users and their processes. The Scheduler class allocates time quantum for each process, handles process state transitions, and logs these changes thread-safely using mutex and condition_variable to synchronize thread operations. The Scheduler uses the Process, User and EventLogger classes to ensure correct process execution simulation. The drawbacks compared to round robin technique among all the processes independently of the users is that given a large number of processes, the processes will get a very small amount of cpu execution time. There is also a large amount of context-switching in this case. Advantages are that Users are not left to starve. Also, given a small number of processes, the fair share scheduler executes faster.

3. CONTRIBUTIONS OF MEMBERS

Name	Contribution
Mik Driver	Designed and implemented the Scheduler, User, Process, and EventLogger classes.
Victor Depaz	Aided in design of Scheduler, User, Process, and EventLogger classes. Contributed to process state management, fair-share algorithm, testing, and debugging the scheduling logic.