Operating Systems Simulation Project Report

CSEN 602 – Milestone 2
German University in Cairo

Team Member: [Omar Ibrahim, Youssef Hamed, Abdelrahman Mohamed, Amr Eladawy, Mohamed, Anas Galal, Omar Hany]
Date: [8/5/2025]

Abstract

This report summarizes the design, implementation, and testing of an Operating Systems simulation developed in C with a GTK3 front-end. The simulator implements process control, memory management, synchronization, and multiple scheduling algorithms (FCFS, Round Robin, and MLFQ).

Introduction

The goal of this project is to build a user-friendly OS simulator that demonstrates core operating system concepts: process lifecycle management, scheduling policies, memory operations, and inter-process synchronization. A graphical interface allows step-by-step execution, parameter tuning, and real-time visualization of process state and memory usage.

Objectives

- **Simulate process execution** by parsing and interpreting instruction streams.
- **Manage memory** in a simulated heap, including variable allocation and access.
- **Implement synchronization** primitives (mutexes) to control access to shared resources.
- **Support multiple scheduling algorithms** (FCFS, Round Robin with configurable quantum, and Multi-Level Feedback Queue).
- **Provide a GTK3 GUI** for control, logging, and visualization.

System Architecture

The simulator is organized into modular components:W

- `memory`** module**: Manages a byte-array heap and variable lookup by name.
- `pcb`** and `process` modules**: Define and manipulate Process Control Blocks (PCBs) tracking process IDs, state (NEW, READY, RUNNING, BLOCKED, EXIT), and memory bounds.
- `interpreter`** module**: Reads program files, decodes instructions (assignment, I/O, synchronization calls), and executes them against the memory and PCB layers.
- `mutex`** module**: Provides `semWait`/`semSignal` operations to implement mutual exclusion for named semaphores (mutexes).
- `scheduler`** module**: Implements FCFS, Round Robin, and MLFQ policies. The API `initScheduler(algo, quantum)` sets the active policy; `schedule()` and `onProcessUnblocked()` handle queue management and preemption.
- `gui`** module**: A GTK3 application presenting controls (scheduler selection, quantum spinner, step/run buttons), log output, memory view, and process state table.

Implementation Details

1 Memory Management

The `memory.c` file defines a contiguous simulated memory array. Variables are inserted with `storeVariable(name, value)`, and looked up by `loadVariable(name)`. Bounds checking and initialization ensure safe access.

2 Process Control Block

Each process is represented by a `PCB` struct holding its ID, program counter index, state enum, and memory offset ranges. PCBs are allocated dynamically on process creation and tracked in the scheduler's ready and blocked queues.

3 Instruction Interpreter

`interpreter.c` tokenizes each line of the input program. It supports assignments (e.g. `a = b + 1`), print statements, blocking reads, and mutex operations. The interpreter returns control to the scheduler after each instruction or on I/O waits.

4 Scheduling Policies

- **FCFS** enqueues processes in arrival order and runs them to completion.

- **Round Robin (RR)** uses a time-quantum; the scheduler preempts processes after `quantum` steps and requeues them.

- **MLFQ** maintains multiple queues of decreasing priority; each higher-priority queue uses shorter quanta. Processes demote on expiry and promote on I/O blocking to prevent starvation.

The GUI exposes a combo-box and spin-button to switch policies at runtime via `on_scheduler_changed()`.

5 Synchronization

The `mutex` module implements named semaphores with counters and blocking queues. `semWait(name)` decrements the semaphore or blocks the calling process, while `semSignal(name)` wakes one blocked process, integrating with the scheduler via `onProcessUnblocked()`.

6 GUI and Integration

A single GTK3 window shows:

- **Control bar**: Scheduler selection, quantum value, step/run/stop buttons.

- **Execution log**: Live text output of debug messages and instruction traces.

- **Process table**: Lists PCBs with PID, state, and PC.

- **Memory view**: Shows current variables and values.

Testing and Validation

Three sample programs (Program_1, Program_2, Program_3) were used to validate:

1. **Basic arithmetic and I/O**, ensuring correct interpreter operation and GUI prompts.

2. **Mutex contention**, verifying blocking and unblocking semantics.

3. **Scheduling behavior**, confirming RR preemption and MLFQ priority movement.

Conclusion

This project successfully demonstrates a miniature OS environment, complete with memory, processes, synchronization, and scheduling, all managed within a GTK3 graphical interface. The modular design permits easy extension (e.g., additional algorithms) and offers clear visualization for teaching core OS concepts.
