

SYSC4001 Assignment 2 - Group

Part 3

Omar Elhogaraty - 101302440 [Student 1]

Ahmad El-Jabi - 101303269 [Student 2]

November 7, 2025

Part 3)

Github Repository: https://github.com/omario05/SYSC4001_A2_P3

Introduction:

This project is implementing a C++ interrupt-driven simulator that models how an OS handles CPU bursts, SYSCALL, END_IO, FORK, and EXEC events. Each activity advances simulated time and triggers system calls, ISRs, or process-control events. The simulator prints out two log files: execution.txt, and system_status.txt.

Testing:

For the first trace file given in the assignment, the behaviour we observed is that 1. the parent forks, then child (PID 1) runs first, and parent (PID 0) waits. After that, the child executes program1, loading it and running the CPU for 100 ms. Lastly, the control returns to the parent, which executes program2 and triggers a SYSCALL ISR. Our FORK block allocates the child PCB and recurses into the child trace, the EXEC block computes load time, updates the PCB, prints system status, and recurses into the new program's trace. The parent resumes after the child returns.

For the second trace file given in the assignment, the behaviour we observed is that 1. init would fork, and PID 1 runs. Then, PID 1 executes program1, which reassigns the partition. Next, program1 would fork PID 2, and now we have three processes in the CPU: PID 2 is running, while PID 0 and PID 1 are waiting. Then PID 2 executes program2, then PID 1 executes program2, and finally PID 0 resumes CPU. This recursion between the parent and the child traces produces proper process creation, waiting, accurate partition tracking.

Conclusion:

The simulator we created makes models of multi-level forking and execution operations, ISR handling, and process scheduling. The recursive design, memory allocation tracking, and timed logging of this system has produced the expected output for all the required and custom tests we have created.