Assignment 2 Part 2: Group Report
SYSC 4001 A, L4

Justin Kim
101298662

Dorian Bansoodeb
101309988

Friday, November 7th, 2025

Github Repository Link Part 2: https://github.com/kimmerr9/SYSC4001_A2_P2
Github Repository Link Part 3: https://github.com/Dorianbansoodeb/SYSC4001_A2_P3

Objective:

The objective of this assignment was to design and implement an interrupt process management sim in C++. It builds off assignment 1, which could already handle how an OS handles system calls and I/O completion, but now adds on FORK and EXEC functionality. We can mimic all these interrupt types by reading and executing a trace file. The goal of this simulator is to see how PCBs are cloned and updated, and how the process states change over time through the generated outputs.

Methodology:

The simulator was implemented using the repository for assignment 2, and using the interrupts.cpp file. Recursion was used to simulate the nested process executions.

Each line in the trace file now has an activity including CPU, SYSCALL, END_IO, FORK or EXEC.

When a FORK interrupt occurs, the PCB of the current process is cloned and the child process is created in its own partition in memory. The simulator recursively executes the child's trace while the parent process waits for completion.

When an EXEC interrupt occurs, the current process' image is replaced by a new program (found in external_files.txt). The program's size determines the loader time. The execution continues recursively using the external program's trace.

For each interrupt, the system logs transitions such as switching to kernel mode, saving context, loading interrupts and invoking ISRs. All outputs are logged in execution.txt, similarly to Assignment 1. system_status.txt keeps track of the waiting and running states of the processes.

There was a break statement in Part 3 of the code at the end of the EXEC block. This is there to stop the current trace after a successful EXEC. It is needed as in OS, exec() replaces the image process, so the old code and instructions no longer exist. Without the break, the sim would incorrectly continue reading the old trace.

Test Case	Description	Expected Behaviour	Observed output
1 - Mandatory	- init forks, child runs program1 (10 MB) - Parent later runs program2 (15 MB)	- t = 24: FORK creates child (PID 1 running, PID 0 waiting) - t = 247: Child executes program1 in partition 4	Matches

	- program1: CPU 100 ms, program2: SYSCALL 4	- t = 620: Parent executes program2 in partition 3	
2- Mandatory	- init forks a child,runs program1 - program1 forks again, both run - program2 Parent init resumes with CPU burst	- 3 processes appear (init, program1, program2) - Each EXEC replaces the process image - Child runs program2 first, then parent - init finishes with final CPU burst	Matches
3 - Mandatory	- init forks, child exits, parent runs program1 (60 MB) - program1: CPU 50, SYSCALL 6 ,CPU 15 ,END_IO 6	- t = 33: Child finishes CPU burst, parent executes program1 - program1 runs CPU, I/O, and resumes after END_IO - Execution alternates between compute and I/O until completion	Matches
4 - Fork + Exec with I/O	- init forks (12 ms) - ChildEXEC program2 (I/O operations) - Parent,CPU 30 during child's wait	- t = 12, child runs program2, issues SYSCALL 4 - Parent executes CPU while child waits - END_IO wakes child; both finish execution	Valid (line by line check)
5 - Double fork	- init forks twice - 1st child, program1 (CPU 60) - 2nd child → program2 (CPU 30) - Parent → CPU 40	- Three PCBs active: parent + 2 children - Each child executes its program in a partition - Parent runs after children complete CPU bursts	Valid (checked logic)