SYSC4001 Assignment 2: Part 3 Report

Date of submission (Group): 2025-11-07

Colin Calhoun (Student 1) - 101307947
James Noel (Student 2) - 101306496

Repositories:
https://github.com/ColinCalhoun/SYSC4001_A2_P3
https://github.com/ColinCalhoun/SYSC4001_A2_P2

This report summarizes the results of simulations using an API simulator designed to reproduce the behavior of the fork and exec system calls. The simulator uses fixed memory partitions, a PCB table, and external program files to emulate process creation, memory allocation, and program execution. The main goal of these simulations was to observe how process creation and program loading occur and how system calls are represented in the simulator.

Test 1

The init process forks a child. The child executes program1 while the parent executes program2. The child runs to completion first, with the PCB cloned at the fork and memory partition allocated for each program. Execution steps follow expected behavior of fork and exec.

Test 2

The child of init executes program1 and then forks again to create a grandchild process that executes program2. The nested forks are handled correctly, with the child waiting for the grandchild to complete before resuming execution. Memory partition allocation and PCB updates function as expected. One last thing to note is that both the parent and the child execute the line “EXEC, program2” because the previous line was an ENDIF, meaning both processes will run EXEC.

Test 3

The init process forks, and the child executes program1, which includes CPU bursts and a simulated syscall. This scenario demonstrates sequential execution of parent and child, with context saving and memory partition updates visible in the logs.

Test 4

The child of init executes program1, then forks again to create a grandchild. The child and grandchild execute program2 in sequence, while the parent executes its tasks after the IF_PARENT block. Logs show correct handling of multiple concurrent processes and proper partition allocation.

Test 5

The init process executes program2 first, then forks a child that executes program1. The logs confirm that the simulator handles mixed sequences of exec and fork, correctly managing child and parent execution order and updating the PCB and memory partitions.

Discussion

Across all scenarios, the simulator accurately reproduces the key mechanics of fork and exec. Child processes are correctly cloned from parent PCBs, programs are loaded into available memory partitions, and execution follows the priority rules specified in the assignment. The logs demonstrate context saving, memory allocation, and sequential execution, validating that the simulator represents system calls and process scheduling effectively.