

**SYSC4001**  
**Assignment 2 report**  
**Shahd Elsaman**  
**101268283**  
**Nawal Musameh**  
**101360700**

Github link:

[https://github.com/ShahdElsaman/SYSC4001\\_A2\\_P2.git](https://github.com/ShahdElsaman/SYSC4001_A2_P2.git)

[https://github.com/ShahdElsaman/SYSC4001\\_A2\\_P3.git](https://github.com/ShahdElsaman/SYSC4001_A2_P3.git)

The simulator is built on an interrupt handling system and manages set data structures to represent the operating system. It includes the processor control table which is a fixed memory partition table, and an external file list. The simulation reads commands from a trace.txt file and processes them sequentially to simulate the passage of time and changes in system state.

Looking at every test simulation we have:

Test 1: Basic fork/exec:

- This test analyzes fork and exec logic. At time: 24, the system\_status1.txt log shows that init PID 0 has forked, placing itself in the waiting queue and setting the new child PID 1 to running. The child PID 1 then EXECs program1, and the log at time 247. It shows its PCB updated to program1 size 10 in partition 4. After the child finishes its CPU,100 bursts. The parent PID 0 resumes, EXECs program2 log at time: 620 and runs its SYSCALL.

Test 2:

- This is a test of nested process creation. The logs correctly show init PID 0 forking at time 31 the child PID 1 EXECs program1 at time 220 and then PID 1 forking again at time 249 to create a grandchild PID 2. The grandchild PID 2 EXECs program2 at time 530 and runs its CPU, 53 burst . The child PID 1 then resumes, EXECs program2 for itself at time 864, runs its CPU, 53 burst, and finally runs the CPU, 205 from the main trace.

Test 3:

- This test checks the logic for an empty child trace. At time 34, init PID 0 forks, creating child PID 1 . The child's trace block is empty, so it runs the final CPU, 10 and finishes. The parent PID 0 resumes, runs its EXEC program1, 60 command at time: 277 , and executes the trace from program1. After program1 finishes, the parent PID 0 does not execute the CPU, 10 from trace3.txt. This test passes and correctly demonstrates the destructive nature of the exec call which is simulated by the break statement in the EXEC block.

Test 4:

- This test has the child exec program1 and the parent exec program2, which contains a SYSCALL, 2 command. The parent PID 0 EXECs program2 at time 307 .

Test 5:

- The trace forks, the child EXECs program1, and both parent and child are set to run CPU 205 at the end. The program1 contains a FORK and an EXEC. In the system\_status5.txt, we can see that PID 0 forks at time 26 child PID 1 execs at time 259, PID 1 forks at time 290. The grandchild PID 2 execs at time 568 and finally child PID 1 execs again at time 946. The execution5.txt log shows the grandchild PID 2 running CPU, 100, followed by the child PID 1 running its own CPU, 100. The log ends with a single CPU, 205 burst, which belongs to the Parent PID 0.

The break:

For the break in the interrupt.cpp it is placed after the exec instruction exit the loop that processes the trace file. This is important because in OS when a process calls exec it completely replaces its current program with a new one. The original process would stop executing any instructions from its previous trace and the new program will execute by simulate trace. The break is an important brace to make sure that the simulator does not keep processing the original trace after the exec.

