SYSC4001 Assignment 2 Group 10

Student 1: James Gohl 101299043
Student 2: Magdi Hajjaj 101299841
Date: 07-11-2025

Part 3 – Design and Implementation of an API Simulator: fork/exec

For part 3 of assignment 2, the objective was to design and implement a simulator that models how a computer handles the fork/exec function as well as interrupts.The goal of this simulator is to be used to analyze the influence of the different components of the fork / exec functionality of a computer. For this assignment the simulator was built on the template code in CPP and various tests were simulated.

The CPP simulator works by taking input from a trace file which contains a pseudo outline of a program. The trace file includes fork and exec system calls. It also includes if_child, if_parent and end if statements that simulate how a program would check which process it is after fork and exec were called. Each exec statement in the trace file also has a program name associated with it. This program name directs the simulator to external files that contain the program that exec is going to run. These programs follow the same structure as the interrupt simulator from assignment 1, as well as the structure of the input trace. The information from the original input trace is then processed through the simulator into an output file. This output contains the steps the CPU takes to run the program in the input and the associated time it takes. There is also a system status output file that contains snapshots of what a PCB table would look like at various points during the execution of the program contained in the trace.

After the simulation was created, various test cases were performed, and the results were analyzed to determine what the system was doing in order to give a better understanding of the fork and exec system calls.

Test 1: Test 1 was a simple trace that had a parent process fork a child. That child then called exec and started running another program which was just a basic CPU burst. When that child terminated, its parent called exec and began running its own program that was just a basic system call. This test just demonstrated the basics of fork and exec but did not go too in depth and did not contain any nested process creations after the original fork and exec.

Test 2: In test 2, it begins with a parent that forks itself into a child process. This child then calls exec and begins executing the program from the exec. Inside this program there are more forks and execs. This creation of various different processes can easily be demonstrated by looking at the system status output which contains the PCB table. Essentially this test demonstrated how a newly created process can create a new process and so forth. By chaining together forks and execs you can create many processes and the PCB table will grow and shrink as the programs run.

Test 3: In test 3 it began with a parent process that forks itself into a child process. This child process did not call exec and just continued with the program doing a simple cpu burst. When the child finished, the parent called exec and began running a program that did not contain any forks or execs. The output for this test was fairly simple as there was only one fork and one exec. This is demonstrated in the small output of the PCB table.

Test 4 and 5: Tests 4 and 5 are fairly similar to the other test performed as they just gave a basic output to help one understand the flow of programs that call fork and exec. There was nothing too complicated like the other tests.

In conclusion, this simulator helps aid in the understanding of what the fork and exec system calls are and how they work. The most helpful part of the simulator was building it. Building it helped teach about when fork and exec do things such as create new processes, modify the PCB, and free and release memory. As well, the various tests performed, although not super in depth, also helped with understanding the flow of how programs with process creation work. It was a challenge to design the simulator, but the struggle helped teach many import parts about how operating systems go about process creation.

Github Repositories links:

SYSC4001_A2_P2: **https://github.com/MagdiHajjaj/SYSC4001_A2_P2**
SYSC4001_A2_P3: **https://github.com/JamesGohl/SYSC4001_A2_P3**