Assignment 2 Report

SYSC 4001: Operating Systems

Jonathan Gitej (101294584)
Cole Galway(101302762)

Github Part 2 repo:https://github.com/colegalway/SYSC4001_A2_P2

Github Part 3 repo:https://github.com/JonathanGitej/SYSC4001_A2_P3

1. INTRODUCTION

The goal of this assignment is to reproduce the behavior of the fork() and exec() system calls whilst emulating an operating system with a single CPU and fixed memory partitions. Each process is represented by a PCB that stores its essential information such as PID, parent PID, program name, memory partition, and state. The system calls like fork and exec were implemented through the interrupt service routines (ISRs) from assignment 1 that mimic context switching, process duplication, and program replacement.

2. DISCUSSION

Why break?

The reason we break at the end of the EXEC implementation is because it stops the current program's execution loop which allows a new program to start running while the old one is stopped.

Test Cases:

Test 1 (Basic Fork and Exec)

In this case, a fork is performed that creates a child which executes program1, while the parent later executes program2. The simulator correctly cloned the PCB, assigned a new partition to the child, and ensured that the parent waited until the child completed execution.

Test 2 (Nested Fork and Exec)

In the second test case, a fork is called, and the child executes program1 while the parent waits. Afterward, there's a CPU burst. The child completed its execution first, then control returned to the parent to finish its CPU burst.

Test 3 (EXEC Followed by I/O)

In the third test case, a fork occurs which is then followed by the parent executing program1, which includes CPU bursts, SYSCALL, and END_IO operations. This confirmed that CPU and I/O activities are correctly handled after fork and exec events.

Test 4 (4 Program EXEC)

This custom test involved multiple nested forks and execs across four different programs. The trace demonstrated that recursive calls to simulate_trace() correctly handled branching so that each child process completed before its parent resumed.

Test 5 (Repeated Program EXEC)

In the final test, a single program was executed multiple times through consecutive exec calls separated by CPU bursts. Each execution correctly freed the previous partition and reloaded the program, updating the PCB each time.

3. CONCLUSION

In conclusion the fork() and exec() simulations that were added to assignment 1 allowed for an accurate representation on how they work inside an operating system. They were able to manage PCBs, memory allocation and context switching which overall demonstrated how fork() and exec() function. Lastly the test cases provided different scenarios which were all accurately handled by the simulator.