Assignment 2 Part III Report

Students: Yuvraj Bains (101235916) and Jared St.Louis (101236784)

Course: SYSC 4001 - Operating Systems

Assignment: Assignment 2 - Part III

GitHub Repository: https://github.com/JStLouisCode/SYSC4001_Ass2_P3

1. Introduction

This report goes over how our fork/exec system call simulator works and what results we got. It manages process creation (fork), program loading (exec), memory management with fixed partitions, and process scheduling. We ran three required tests and added two of our own to check for reliability.

2. Implementation Overview

2.1 FORK System Call

The FORK call copies the running process to create a child. When it happens, a new PCB gets created with a unique PID, copying the parent's info. The child runs first while the parent waits in the queue. Once the child is done, the parent resumes.

In our code, we extract the child's trace between IF_CHILD and IF_PARENT, make a new PCB, recursively call simulate_trace() for the child, and continue the parent afterward.

2.2 EXEC System Call

EXEC replaces the current process with a new program. We look up the program size, free the old memory, allocate a new partition using best-fit, simulate loading time (15ms/MB), and then execute the new trace. We add a break statement so the old program doesn't keep running after EXEC.

2.3 Memory Management

We used fixed partitions: 40MB, 25MB, 15MB, 10MB, 8MB, and 2MB. The best-fit algorithm chooses the smallest partition that fits. This keeps fragmentation low and uses memory efficiently.

3. Test Scenario Analysis

3.1 Test 1 – Basic Fork and Exec

Init forks → child runs EXEC program1 → parent later runs EXEC program2. The child gets priority, memory reallocation worked fine, total runtime about 886ms.

3.2 Test 2 – Nested Fork

The child itself forks again, which creates a grandchild. Both child and grandchild execute EXEC program2. Everything ran properly, total runtime around 1123ms.

3.3 Test 3 – Conditional with EXEC in Parent

Parent runs EXEC while child just runs a small CPU burst. Parent's process gets replaced with program1_v3. Total runtime about 900ms.

4. Custom Test Scenarios

Custom Test 1 – FORK Mid-Execution

We forked after some CPU work to test behavior after ENDIF. Both parent and child executed correctly.

Custom Test 2 – Multiple Forks in Parent

The parent forks multiple times sequentially. The simulator handled it fine with correct process order and memory usage.

5. Performance Analysis

Test 1 took 886ms total. Program loading used most of the time (~42%). CPU and I/O followed, with minor interrupt overhead. This matched expectations given the 15ms/MB loading time.

6. Key Questions Answered

Why the break after EXEC?

EXEC replaces the current process image. Without the break, the simulator would keep running the old code. In real systems, the old code is gone once exec() runs.

How recursion ties in

Both FORK and EXEC rely on recursion. FORK handles child traces recursively, EXEC runs new program traces recursively, keeping process flow consistent with real OS behavior.

7. Challenges and Fixes

- Parsing IF_CHILD/IF_PARENT blocks: solved using flags.
- Memory management: freed old partitions before reallocating.
- Child priority: ensured the child always ran before the parent resumed.

8. Conclusion

Our simulator captures fork and exec behavior accurately, from process creation and scheduling to memory allocation. All required and custom tests produced correct results.

Appendix: Repo Contents

interrupts.cpp – main simulation code
interrupts.hpp – helper functions
input_files/ – test traces
output_files/ – results
report.docx – this document
GitHub: https://github.com/JStLouisCode/SYSC4001_Ass2_P3