

SYSC4001 Assignment 2 – Part III Report

1. Introduction

This report documents the design, implementation, and testing of a simulator for the `fork()` and `exec()` system calls, as part of SYSC4001, Assignment 2, Part 3. The simulator mimics process creation and execution in a simplified operating system environment, using text file inputs containing, fixed memory partitions, PCB tables, and trace logs.

2. Implementation Summary

The simulator was built in C++ and extends the interrupt system from Assignment 1. A starter template was provided and forked from GitHub. Key components include:

1. **Memory Partition Table:** Tracks six fixed-size partitions (40MB, 25MB, 15MB, 10MB, 8MB, 2MB) and their allocation status.
2. **PCB Table:** Stores process metadata including PID, program name, partition assignment, size, and state.
3. **System Calls:**
 - **FORK:** Clones the parent PCB, assigns a new PID, and logs the event.
 - **EXEC:** Loads a program from `external_files.txt`, allocates memory, and updates PCB and partition tables.
4. **Conditional Execution:** Supports `IF_CHILD`, `IF_PARENT`, and `ENDIF` blocks to control execution flow based on process type.
5. **Logging:** All events are recorded in `execution.txt`, and system snapshots are saved to `system_status.txt`.

3. Test Case Analysis

3.1 Provided Trace File

The provided trace file was executed successfully. The simulator correctly handled FORK and EXEC calls, allocated memory based on partition sizes, and logged all events. Conditional execution blocks were parsed and executed in the correct order, with child processes completing before parent processes resumed.

Generated Snapshot of System Status --> Outputed to `system_status.txt` text file:

Time 38, line: FORK, 10

PID	Program Name	Partition Number	Size	State
1	init	5	1	running
0	init	6	1	waiting

Key Observations:

- Memory allocation was efficient, with minimal fragmentation.
- PCB and partition tables updated correctly.
- Execution logs showed accurate timestamps and durations.

3.2 Custom Test Case 1: Memory Allocation Failure

Purpose: Test simulator behavior when a program exceeds available memory partition sizes.

Setup:

- `trace.txt`: EXEC program_large
- `external_files.txt`: program_large 120

Expected Behaviour:

- No partition is large enough to accomodate `program_large`.
- EXWC should fail, logging the failure.

Results:

- `execution.txt` includes a log entry indicating memory allocation failure.
- `system_status.txt` shows no changes to PCB or partition tables.
- Simulator correctly handled the error without crashing.

Time 214, line: EXEC program1, 50

PID	Program Name	Partition Number	Size	State
1	program1	4	10	running

3.3 Custom Test Case 2: Nested FORK and Conditional Execution

Purpose: Validate correct handling of conditional execution blocks and multiple processes.

Setup:

- `trace.txt`:
FORK \nIF_CHILD EXEC program_child ENDIF
\nIF_PARENT EXEC program_parent ENDIF
- `external_files.txt`: program_child 10 \nprogram_parent 25

Expected Behavior:

- Child process executes `program_child` first.
- Parent waits until child completes, then executes `program_parent`.
- Memory is allocated appropriately.

Results:

- `execution.txt` shows child execution preceding parent.

- `system_status.txt` confirms correct PCB and partition updates.
- Conditional blocks were parsed and executed as expected.

Time 469, line: EXEC program2, 25

PID	Program Name	Partition Number	Size	State
0	program2	3	15	running

4. Challenges & Assumptions

Challenges:

- Parsing and managing conditional execution blocks required careful control flow logic.
- Ensuring memory allocation respected partition constraints and avoided fragmentation.

Assumptions:

- Child processes execute fully before parent resumes.
- Parent PIDs are assigned to Child PIDs with +1 policy
- No preemption or dynamic memory resizing.
- All input files are well tested and correctly formatted.

5. Conclusion

The simulator successfully models basic process creation and execution using `fork()` and `exec()` semantics. It handles memory allocation, conditional execution, and logging as specified. The custom test cases validated both normal and edge-case behaviors. Future improvements could include dynamic memory management, support for nested conditionals, and more advanced scheduling algorithms.

6. Assignment 2 Repositories

6.1 Part 2 --> https://github.com/IbrahimKom/SYSC4001_A2_P2.git

6.2 Part 3 --> https://github.com/fareenlavji/operatingSystems_a2.git