

# Carleton University

Department of Systems and Computer Engineering  
SYSC 4001 – Operating Systems (Fall 2025)

## Assignment 2 – Part III: Design and Implementation of an API Simulator (fork/exec)

### Group Members:

- Ibrahim Mustapha (101266157)
- Basil Thotapilly (101313150)

## Overview

This project implements a simplified **API simulator** to reproduce the behavior of the Unix/Linux **fork()** and **exec()** system calls within a simulated operating system. The simulator models key OS concepts — **interrupt handling**, **process control blocks (PCBs)**, and **fixed memory partitions** — using structured trace files, external file tables, and simulated timing.

The system tracks process execution through **execution logs** and **system status snapshots**, which show changes to process state and memory assignment.

The simulation uses:

- **Fixed memory partitions** totaling 100 MB (6 partitions)
- **PCB table** for process metadata
- **External files table** (program1.txt, program2.txt)
- **Trace files** (trace.txt, trace1.txt, trace2.txt) defining process execution

## Simulation Logic

When the simulator runs, it performs the following:

1. **FORK** → triggers an interrupt handler at vector 2, clones the current PCB, and creates a child process.
2. **IF\_CHILD / IF\_PARENT** → determines which instructions the child or parent executes until ENDIF.

3. **EXEC** → triggers an interrupt handler at vector 3, loads a new program (from external\_files.txt) into an available partition, marks it as occupied, and updates the PCB.
4. **CPU / SYSCALL / END\_IO** → standard execution and interrupt operations as defined in Assignment 1.
5. **System snapshots** (system\_status.txt) record the current time, executing process, partition, and state after every FORK and EXEC.

Each simulation event includes timestamps and durations that emulate context switches, kernel mode transitions, and ISR behavior.

## Test Scenarios

### Test 0 – fork + exec (child and parent)

**Trace file:** trace.txt

```
sqlCopy codeFORK, 10
IF_CHILD, 0
EXEC program1, 50
IF_PARENT, 0
EXEC program2, 25
ENDIF, 0
```

#### Results:

- The simulator correctly performs a **FORK** and then executes both programs in sequence (child then parent).
- Execution log (exe.txt) shows context switch, ISR setup, cloning, and PCB updates.
- System snapshot (sysstat.txt) confirms the init process forked successfully, with the child executing program1 and parent later executing program2.

#### Key output (excerpt):

```
arduinoCopy code0, 1, switch to kernel mode
1, 10, context saved
11, 1, find vector 2 in memory position 0x0004
12, 1, load address 0X0695 into the PC
13, 10, cloning the PCB
23, 0, scheduler called
...
36, 50, Program is 15Mb large
86, 225, loading program into memory
```

This matches expected behavior: fork at vector 2 and exec at vector 3, with proper scheduler invocation and memory allocation.

## **Test 1 – fork + CPU (child) + exec (parent)**

**Trace file:** trace1.txt

```
sqlCopy codeFORK, 8
IF_CHILD, 0
CPU, 20
IF_PARENT, 0
EXEC program2, 25
ENDIF, 0
```

**Results:**

- The child process performs a **CPU burst** of 20 ms before termination.
- The parent process then executes program2.
- Execution log (exe1.txt) shows correct kernel transitions and scheduler calls.
- System snapshot (sysstat1.txt) confirms process control consistency — init remains running, and program2 executes afterward.

**Key observation:**

This validates that the simulator correctly switches from forked child computation back to the parent's execution path without preemption.

## **Test 2 – fork + exec (child) + CPU (parent)**

**Trace file:** trace2.txt

```
sqlCopy codeFORK, 12
IF_CHILD, 0
EXEC program1, 40
IF_PARENT, 0
CPU, 30
ENDIF, 0
```

**Results:**

- The child executes program1, while the parent performs a CPU burst after the fork.
- exe2.txt demonstrates correct ISR handling and timing sequence.
- sysstat2.txt shows the init process running and proper handling of the forked process before returning to the parent's execution.

This test confirms that **parent and child isolation** in PCBs and partition mapping is preserved — each follows its trace independently with accurate context saving and restoration.

## Analysis

| Aspect                    | Verified Behavior                                       |
|---------------------------|---|
| <b>FORK ISR</b>           | Vector 2 triggered, PCB cloned, scheduler invoked       |
| <b>EXEC ISR</b>           | Vector 3 triggered, memory loaded based on program size |
| <b>Context Handling</b>   | Correct kernel mode transitions and IRET returns        |
| <b>Memory Management</b>  | Each partition updated via allocate_memory()            |
| <b>PCB State Tracking</b> | System snapshots display accurate process states        |

All tests reproduce expected system behavior. The child process always executes before the parent, adhering to the non-preemptive policy stated in the assignment.

## Conclusions

This simulator effectively models the essential functionality of the `fork()` and `exec()` system calls. The system correctly performs PCB cloning, partition allocation, and execution flow tracking through trace-based simulation.

Each test demonstrates:

- Consistent interrupt handling
- Accurate parent/child process distinction
- Proper snapshot recording and memory updates

The design successfully replicates simplified UNIX-style process management, and all required input/output file structures are complete and verified.

## GitHub Repositories

- **Part II:** [https://github.com/05basil/SYSC4001\\_A2\\_L3\\_P2\\_Group30](https://github.com/05basil/SYSC4001_A2_L3_P2_Group30)
- **Part III:** [https://github.com/Abezinx/SYSC4001\\_A2\\_P3](https://github.com/Abezinx/SYSC4001_A2_P3)