SYSC4001 - Assignment 2

Abubakr Mohammed(101287262)
Kezi Afulukwe(101279214)

The API simulator I built is a tool to simulate and study simple operating-system process-management behavior, specifically the outcomes of executing "fork" and "exec" system calls, memory allocations, control of parent-child processes, and CPU use, all entirely within the discrete simulation rather than using an operating system. The simulator processes an input trace file (a sequence of timed commands) while keeping a process table (PCB), memory partitions, and process priority (the child runs first after the fork).

The idea behind the simulator is to provide a clear step by step perspective of the reaction of an operating system to a series of process events when all the processes are predictable. It captures all the significant changes: each time a process produces a child it does so, each time it executes on the CPU, each time it loads a program, and how memory and states change as a result of each significant event. This allows us to understand the way processes are scheduled, the way memory is shared, and the precise order of execution of parents and children as dictated by the rules.

Description of the Five Test Cases

Test 1: Sequential CPU

This simple trace only has CPU bursts. The log shows the system time progressing for each CPU event. The process table remains stable, showing a single running process, with only the current time and finished workloads changing.

- **What is being tested:** Baseline logging and time advancement, showing correct understanding of basic CPU event processing and logging.

Test 2: Fork then Exec

This classic fork-exec scenario checks correct parent/child relationship and prioritization. A process forks, the child runs to completion (possibly executing a program with exec), and only then does the parent resume. The main points under examination are:

- Child partition allocation
- Correct parent waiting during the child's lifetime
  Accurate system log of transitions

Test 3: More Complex Sequence

Suppose this trace includes a fork, additional CPU work in both parent and child, and perhaps both child and parent perform exec. The logs will show:

- Child's actions taking priority and completed fully before the parent's resume.

- Multiple changes in PCB table: process states, running versus waiting.

- Partition allocation and deallocation as exec events are processed.

Analysis: This demonstrates not just single-use of FORK-EXEC but the simulator's ability to handle more prolonged branching, correct resumption of the parent, and proper ordering of resource management and status logging.

Test 4: Custom Forks then Diverged Execs

Here, after a fork, the child process immediately executes program2, and after it completes, the parent resumes and executes program1. The logs show the child is always running after the fork, gets the next available partition, and its exec logs all the memory load accounting and transitions. The parent is properly blocked until the child ends; after exec, the statuses again reflect the changes in partitions and states. Logging shows, step by step, how the scheduler is called after each significant event, always maintaining process table consistency. The trace confirms correct child-priority enforcement and exec event isolation for both parent and child, with precise partition tracking.

Test 5: Minimal Forks and Dual Execs

Test 5 further stresses the system with a fork where both child and parent perform short CPU bursts followed by execs to different programs ('program2' for child, 'program1' for parent). The trace validates that at no point do the processes ever use the same partition concurrently; memory is always freed before reuse, and the scheduling model never violates the child-first rule. It confirms system resilience to branching and simultaneous memory use.

Overall Goal of the Experiments

By running these five diverse simulation scenarios, the aim is to ensure precision of the basic OS process management logic as specified. Specifically, these cover:

- Ensure that fork and exec rules are correct.
- Processing of difficult cases in any partition allocation and memory.
- Status of log system in a way that changes are indicated in the right order.
- Also, the system should not be allowed to go into an undefined or unlawful state, even as traces become more complex and develop branches.