Assignment #2
Part 3 - API Simulator

Bashar Saadi - 101108918
Kalid Wehbe - 101259994
Github Repo: https://github.com/kalidwehbe/Assignment_2
Nov 7, 2025

Summary:
This simulator implements the fork() and exec() system calls for an operating system interrupted simulation model with memory management using fixed partitions. It is tested in five scenarios: Basic operations, nested processes, I/O handling, Pre-fork CPU burst, and complex hierarchies.

System Call Implementations

1. fork() - Creates a child process (cloned by a parent PCB). The algorithm: switch to kernel mode (1ms), save context (10ms) find vector, load ISR (1ms/ISR), clone PCB, allocate memory, call the scheduler, and IRET (1ms). The child always executes with higher priority to the parent and runs nonpreemptively before the parent is dissipated from the wait queue.
2. exec() - Replaces a currently running process with a new program. After an interrupt, the system searches for external_files.txt to figure out program size then uses first-fit to find a partition slot. Loading is 15ms/MB. After the PCB is updated, we return from the interrupt and a new program beings to execute

This code works using recursion to be able to create a child using fork and then run the child immediately. As soon as the fork operation is complete the child begins to run and then it follows the trace file given. Then it goes through a series of functions. For example, if IF_PARENT comes up on the trace or program files then we ignore it since it's a child that's running. If the call is IF_CHILD we run through the code because the child is the one running. Once the child is done running we go right back to its parent so it can continue running.

Why is the break function so important?
The exec function has a break function to get out of the main loop. To understand this we first have to understand what an exec does. Exec stops the current program being read and starts reading a different program. (kinda like how we were reading from the trace file then started to read from the program1.txt file). If we don’t break the process will go back to the original code it was reading and continue which is not what exec does. Exec changes the program and once it’s done it continues to read off its own program and doesn’t go back to the previous program that breaks allows us to do so.

Assignment #2
Part 3 - API Simulator

Test 1 - Basic fork() and exec()
Demonstrates partition allocation for different sized programs by using child to execute program 1 (10MB) and parent executes program 2 (15MB)

Test 2 - Nested Fork Opr.
Demonstrates that both child processes execute before parents and both child and parent of program 1 can execute program 2. Init forks -> child becomes program1 - >forks creating grandchild (PID 2). Handles 3 generations and the wait queue maintains parent processes.

Test 3 - Fork with I/O Opr.
Demonstrates interrupt handling with process management. Fork is tested with SYSCALL and END_IO operations where we can watch the parent execute program 1 containing I/O operations. Result is I/O handling with device specific delays.

Test 4 - Pre-fork CPU Burst
There is a CPU burst 20ms before a fork(). Result is the activity handled before the process is created. Init executes CPU burst before forking then child executes the program with a SYSCALL.

Test 5 - Multiple Fork and Exec Sequences (Nested fork and exec)
A parent executes program 2 which contains another fork() demonstrating multiple fork and exec operations.

Key Observations

- Fork overhead = context save + PCB copy + Scheduling
- Exec overhead = file search + partition find + loading + PCB update
  - Larger programs have proportionally larger exec times.

Limitations

- Non-preemptive (processes run to completion)
- First-fit memory allocation is bound for fragmentation as programs grow.

Conclusion
The simulator was made to model basic OS processes and memory management. It correctly demonstrates how processes are created and managed, how system calls transition the OS from user to kernel mode, and how fixed partition memory allocations work. The system logs all ISR steps and timing and implements a first-fit algorithm to memory management.