SYSC 4001 Assignment 2
Part III: Design and Implementation of an API Simulator: fork/exec

Stefan Martincevic, 101295641
Sadman Sajid, 101303949

07/11/2025

Part II Repository: https://github.com/sadman-sajid1/SYSC4001_A2_P2
Part III Repository: https://github.com/stefan-martincevic/SYSC4001_A2_P3

FORK() and EXCE() by Parent and Child Test

In trace 1 when FORK() is called by init the simulator switches to kernel mode and enters the FORK ISR. A new child PCB is created by cloning the parent's PCB. The child process is given a new PID and is placed in the ready queue, while the parent becomes waiting. The child gains higher priority over the parent. The child process calls the EXEC() system call and the simulator looks up program1 in external_files.txt. Its size is found at 10 MB and the simulator searches for a free partition in the memory table that can accommodate 10 MB, seen in Figure 1. The loader simulates reading the program from disk into memory, which takes 150 ms (15 ms x size: 10 MB). The partitions status is shown to switch from "free" to "program1" and the child PCB is updated (Program name = program1, Partition = 4, Size = 10 MB, State = running). A scheduler call entry appears at the end of the ISR and the child executes the contents of program1. The previous can be seen in Figure 2. Once the child terminates, Figure 3 shows the parent continuing its execution by triggering the EXEC() system call for program2. Similarly, the simulator finds program2 which is 15 MB and chooses partition 3, shown in Figure 1. The load time is 225 ms (15 MB x 15 ms). Then Partition 3 becomes occupied, and the PCB of the parent is updated (program = program2, Partition = 3, Size =15 MB).

```
time: 240; current trace: EXEC, 50+----------------------
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   1 |    program1 |               4 |   10 | running |
|   0 |        init |               6 |    1 | waiting |
+-------------------------------------------------------+
time: 663; current trace: EXEC, 25+----------------------
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   0 |    program2 |               3 |   15 | running |
+-------------------------------------------------------+
```

Figure 1: System Status of Trace 1

```
37, 50, Program is 10 Mb large
87, 150, loading program into memory
237, 3, marking partition as occupied
240, 6, updating PCB
246, 0, scheduler called
246, 1, IRET
247, 100, CPU Burst
```

Figure 2: Execution of program 1 in Trace 1

```
360, 75, Program is 15 Mb large
435, 225, loading program into memory
660, 3, marking partition as occupied
663, 6, updating PCB
669, 0, scheduler called
669, 1, IRET
```

Figure 3: Execution of program2 in Trace 1

Nested FORK() Inside EXEC() Test

Trace 2 is used to test and analyse nested FORK() calls in the simulation. When init executes fork a child process is created with a cloned PCB of the parent. The child loads program1 into its memory image, allocates partition, runs the loader and calls the EXCE() system call to begin executing the contents of program3. The contents of program3 fork the child process to create a grandchild

process, allocate it to a partition, run the loader. Since program2's EXEC() system call occurs after the ENDIF line, both the child (PID 1) and grandchild (PID 2) processes execute program 4 shown in Figure 4.

```
time: 599; current trace: EXEC, 33+----------------------
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   2 |    program4 |               3 |   15 | running |
time: 975; current trace: EXEC, 33+----------------------
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   1 |    program4 |               2 |   15 | running |
```

Figure 4: System Status of Trace 2

FORK() Without Child EXEC() Test

Running Trace 3 creates a fork, creating a child process. Since the child does not call EXEC(), the memory image of the child is a clone of the parent. First the child runs the CPU burst and when it terminates the parent continue its process and runs the CPU burst afterwards. As the parent has a EXEC() system call, after the IF_PARENT line, the memory image of the parent is changed to program 5 and is executed. This test demonstrates the creation of a child process that continues executing the program that is loaded in the parents memory. This is confirmed looking at the system status in Figure 5 where both child (PID 1) and parent (PID 0) run the init program, and only the parent executes program 5.

```
time: 34; current trace: FORK, 20+-----------------------
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   1 |        init |               6 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-------------------------------------------------------+
time: 260; current trace: EXEC, 60+----------------------
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   0 |    program5 |               4 |   10 | running |
+-------------------------------------------------------+
```

Figure 5: System Status of Trace 3

Program Size Larger than Partition Test

In Trace 4, program6, program7, program8 are executed and are of size 10 MB, 15 MB, and 50 MB respectively. The largest partition in this OS is 40 MB. program6 is assigned to partition 4 and program7 is assigned to partition 3. When observing the system status for Trace 4, program 8 was not assigned to any partition as it was too large, thus the program could not be executed.

PID Reuse Test

Trace 5 is used to test whether the OS allows old PIDs to be safely reused after a process is finished. First in this trace a child is forked and performs EXEC() with program9 loaded in its memory image with a PID of 1. Once this process terminates, the parent process forks again to create a new child with the same PID of 1. This child performs EXEC() with program10. The results of the previous steps are shown in Figures 6 and 7. This would validate that resources are cleaned up and the process table can be reused, meaning that once a process terminates, a new process with the same PID can be created and execute a different program.

PID	program name	partition number	size	state
1	program9	4	10	running

Figure 1: PID 1 Executes program9

PID	program name	partition number	size	state
1	program10	3	15	running

Figure 2: PID 1 Executes program10