Khadija Lahlou 101315714, Casey Ramanampanoharana 101233513

Gabriel Wainer

SYSC4001A Operating Systems - Report

November 7th 2025 Repos: Part 2 , Part 3

Why doesn't the rest of the trace matter after ENDIF 0?
Since we've already implemented assignment 1 anything after the endif will act how it does in assignment 1. Reaching endif means that both the child and parent have fully executed their own program so in that case it doesn't matter in terms of this simulator logic because we don't really care about the regular syscalls.

The break in the EXEC
We put a break in the exec() right after we do the recursive simulate trace exec() version because when we encounter an exec in the trace file it will be replaced with the exec_trace and be the new current. As it is recursive it'll go through the whole of exec_traces, once the recursion returns = new program has finished execution, to avoid going back to the original trace_file and the old program we use the break so it doesn't continue doing what doesn't apply.

Test 2 - Who's executing the program2?
FORK, 15
IF_CHILD, 0
IF_PARENT, 0
ENDIF, 0 //notice how nothing is happening in the conditionals
EXEC program2, 33 //which process executes this? Why?

Above is the program1 content (this is a nested fork) flow goes like this:

- Parent_init forks → child_init
- child_init → exec() program1
    - Parent_program1 Forks → child_program1
        - exec() program2 //first exec done by child has priority
    - Parent_program1 → exec() program2 //second exec done by parent after child is done

So since the program1 fork as well but the exec is not in the branch then it will follow the order of the child then parent, in short both are doing the process.

Influence of the different components of the simulator
New components to Assignment 1:

- Memory partitions: Simulates the main memory that limits the number of programs we can have as well as how big a program can be, causing allocation errors if there's no space found, in our sim we skip the process if no memory available.
- PCB: Stores the different process information in our sim this is seen through the system status.
- FORK: creates a new process by copying itself.
- EXEC: replaces the old process program with a new one.

Analysis

Test1 - Simple fork() and exec()

```
time:24; current trace: FORK, 10
+-------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   1 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-------------------------------------------------------+
time:247; current trace: EXEC program1, 50
+-------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   1 |    program1 |               4 |   10 | running |
|   0 |        init |               6 |    1 | waiting |
+-------------------------------------------------------+
time:620; current trace: EXEC program2, 25
+-------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-------------------------------------------------------+
|   0 |    program2 |               3 |   15 | running |
+-------------------------------------------------------+
```

Test 1 makes sure that the basic fork() and exec() syscalls are implemented properly. Fork() creates a child process(PID 1) as an exact copy of itself which immediately starts executing, while the parent is in wait as seen in the system status. The child then releases its old memory to get new memory for the new program1, then updates the pcb with the new program info. Once the child has finished executing the parents then does their exec of the program2, erasing the old init program as seen in the system status the end pcb theres only the program2 left.

Test2 - Nested fork() and exec()

```
time:31; current trace: FORK, 17 //fork is called      You, 4 hours ago • final
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   1 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------+
time:220; current trace: EXEC program1, 16 //exec is called by child
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   1 |    program1 |               4 |   10 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------+
```

```
time:249; current trace: FORK, 15
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   2 |    program1 |               3 |   10 | running |
|   0 |        init |               6 |    1 | waiting |
|   1 |    program1 |               4 |   10 | waiting |
+-----------------------------------------------------+
time:530; current trace: EXEC program2, 33 //which process executes this? Why?
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   2 |    program2 |               3 |   15 | running |
|   0 |        init |               6 |    1 | waiting |
|   1 |    program1 |               4 |   10 | waiting |
+-----------------------------------------------------+
time:864; current trace: EXEC program2, 33 //which process executes this? Why?
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   1 |    program2 |               2 |   15 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------+
```

Test 2 makes sure we implemented the recursions properly and helps us find bugs through the nested fork and exec. The nested fork is in the program1, we start with init creates a child → program 1 since its the nested fork it creates its own child, so now init and program1 are waiting for the child to execute first thus program2 is executed there first then in the parent after. The system status does indeed show the correct fork and exec creation (create copy then update it with new program see above test1)

Test 3 - Conditional blocks

```
time:34; current trace: FORK, 20
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   1 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------+
time:277; current trace: EXEC program1, 60
+-----------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------+
|   0 |    program1 |               4 |   10 | running |
+-----------------------------------------------------+
```

Test 3 tests the conditional loop to make sure we implemented the right logic when using the trace files. For this one it is when you fork your child see that the loop is empty for child so does nothing and is not a parent so does nothing and only executes the syscall after the endif. After the parent executes what's in the parent block which is program1.

Test 4 - Memory allocation failure

```
0, 1, switch to kernel mode      You, 5 hours ago • final s
1, 10, context saved
11, 1, find vector 2 in memory position 0x0004
12, 1, load address 0X0695 into the PC
13, 12, cloning to PCB
25, 0, scheduler called
25, 1, IRET
26, 1, switch to kernel mode
27, 10, context saved
37, 1, find vector 3 in memory position 0x0006
38, 1, load address 0X042B into the PC
39, 35, Program is50Mb large
74, 750, loading program into memory
824, 3, marking partition as occupied
827, 6, updating PCB
833, 0, Error memory not allocated for program1
```

For Test 4, we focused on testing how the simulator handles situations where there isn't enough memory. We intentionally made program1 50Mb large, knowing that our biggest memory space was only 40Mb. The exec still tries to find a space for the 50Mb but since there is not it causes an error, we make program1 go back to its original state, the copy of the parent and skip that exec.

Test 5 - Multiple programs in memory

```
time:24; current trace: FORK, 10      You, 5 hours ago • f
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   1 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:667; current trace: EXEC program1, 20
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   1 |    program1 |               1 |   40 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:1431; current trace: FORK, 12
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   2 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:1844; current trace: EXEC program2, 15
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   2 |    program2 |               2 |   25 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:2574; current trace: FORK, 8
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   3 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:2840; current trace: EXEC program3, 18
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   3 |    program3 |               3 |   15 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
```

```
time:5056; current trace: FORK, 14
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   4 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:5251; current trace: EXEC program4, 22
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   4 |    program4 |               4 |   10 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:5769; current trace: FORK, 16
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   5 |        init |               5 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:5937; current trace: EXEC program5, 25
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   5 |    program5 |               5 |    8 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
time:6659; current trace: FORK, 20
+-----------------------------------------------------------+
| PID |program name |partition number | size |   state |
+-----------------------------------------------------------+
|   6 |        init |              -1 |    1 | running |
|   0 |        init |               6 |    1 | waiting |
+-----------------------------------------------------------+
```

For Test 5, again we wanted to test the memory but this time having it filled with other programs and trying to execute one more. We made 6 programs in total systematically filling the available memory. Which cannot fit anywhere, the memory behaves similarly as in Test 4

but this time it's when the parent tries to fork and create a child that the child cannot be created since there is no space, unfortunately there is no state to say unallocated but we set partition to -1 to showcase it being unavailable memory.