SYSC 4001 Operating Systems Fall 2025

Assignment 2

SYSC4001 L3 – Group 3

Gabriel Wainer

Muhammad Ali – 101291890

Gregory Horvat – 101303925

https://github.com/aliooo36/SYSC4001_A2_P2

https://github.com/aliooo36/SYSC4001_A2_P3

Analysis of Repo Test Cases:

Test 1: FORK and EXEC

In this test, the initial process performs FORK,10, and the logs show the simulator entering kernel mode, saving context, looking up the vector, and cloning the PCB before returning with IRET; as expected, the child is scheduled first. The child then executes EXEC program1,50, and the execution trace confirms the real exec() sequence being simulated: the program is validated, the current partition is freed, a new partition is allocated, and the 10 MB image is loaded with the expected 150 ms cost (15 ms/MB) before the PCB is updated and control returns to user mode. The child then runs the program's body (a 100 ms CPU burst) and terminates. The parent resumes and executes EXEC program2,25, repeating the same steps but with a 15 MB image and thus a 225 ms load, followed by a 250 ms SYSCALL ISR from that program's trace. Across both paths, the system-status snapshots confirm correct PCB fields and fixed-partition allocation after each exec(), and the ordered kernel transitions (vector lookup → loader → PCB update → scheduler → IRET) match OS fork/exec semantics, with logs clearly showing that loader cost dominates runtime while interrupt overhead remains small.

Test 2: Nested FORK and EXEC

In this test, the system begins with a FORK,17 event, after which the simulator logs show a kernel-mode switch, vector lookup, and PCB cloning. The child process then executes EXEC program1,16, which the simulator resolves as a 10 MB executable, freeing its previous partition, allocating a new one, and loading the program with the expected 150 ms cost before updating the PCB and returning to user mode. Once program1 finishes its 53 ms CPU burst, a second FORK,15 occurs, producing another child that executes EXEC program2,33. The logs confirm that program2 (15 MB) incurs a 225 ms load, followed by its own 53 ms CPU burst. Afterward, the parent also performs an identical exec(), showing that both program1 and program2 execute independently with correct partition management and process states—each child running first while the parent waits. The system-status snapshots verify that partitions 3 and 4 were correctly assigned to 15 MB and 10 MB programs respectively, with each exec() freeing and reallocating memory before scheduling. Overall, Test 2 confirms that the simulator accurately models multiple nested fork/exec interactions, maintaining proper PCB updates, partition reuse, and timing behavior consistent with OS semantics.

Test 3: FORK and I/O Driven EXEC

In this scenario, the trace begins with FORK,20, where the simulator enters kernel mode, saves the context, locates vector 2, and clones the PCB before returning with IRET. The child process briefly executes a 10 ms CPU burst, then performs EXEC program1,60. The logs confirm that program1 (10 MB) is loaded after the ISR frees the current partition, allocates a new 10 MB slot, and simulates a 150 ms loader delay (15 ms / MB). Once loaded, the process runs sequential CPU and I/O phases matching the external program trace—50 ms CPU, followed by a 265 ms SYSCALL ISR, another 15 ms CPU, and finally a 265 ms END_IO ISR—each separated by IRET returns to user mode. The system-status snapshots show the expected transitions: the child occupying partition 4 while the parent remains idle, and the PCB reflecting the new executable and active state throughout. This pattern accurately models how an operating system handles forked child execution with subsequent exec(), including I/O interrupts that block and resume the process; the simulator's timing aligns with the expected cumulative CPU and device delays, confirming that interrupt handling and process scheduling function correctly for mixed CPU–I/O workloads

Analysis of Group Defined Test Cases:

Test 1: FORK and Dual EXEC with Device-Driven ISRs

This test begins with FORK,10, where the logs show kernel entry, vector lookup, and PCB cloning; the child is scheduled first while the parent waits. The child executes EXEC program1,25, which the simulator resolves to a 10 MB image from the external file list; it frees the old partition, allocates partition 4, loads the image with the expected 150 ms (15 ms/MB) cost, updates the PCB, and returns to user mode to run program1's body. The execution trace confirms that the SYSCALL code "3" maps to a 300 ms ISR via the device-delay table, matching the observed SYSCALL ISR 300 segment between the two CPU bursts, and the system-status snapshot shows the child correctly occupying partition 4 during this phase. After the child completes, the parent performs its own 15 ms CPU burst and then issues EXEC program2,20; the simulator identifies a 15 MB image, frees memory, allocates partition 3, and simulates the 225 ms loader before running program2's body, where device 5's delay appears as a 211 ms ISR in the logs. Overall, the ordered kernel transitions (vector lookup → free/allocate → loader → occupy → PCB update → scheduler → IRET) and the partition assignments (P4 for 10 MB, P3 for 15 MB) demonstrate correct modeling of fork() and exec(), while the two distinct device codes (3 and 5) produce the expected ISR durations that dominate runtime during I/O phases.

Test 2: FORK with child(program1) and parent(program2) EXEC

The trace issues FORK,15, after which the logs show kernel entry, vector lookup, and PCB cloning; the child is scheduled first and immediately performs EXEC program1,30. The simulator validates program1 as 10 MB, frees the old partition, allocates partition 4, loads the image with the expected 150 ms cost, updates the PCB, and IRET's to user mode; the child then executes program1's body exactly as specified—CPU 60 → SYSCALL(2) → CPU 40 → SYSCALL(4) → CPU 20—which the execution log maps to device ISRs of 150 ms and 250 ms, respectively, with system-status snapshots confirming the partition/state at each milestone. After the child completes, the parent runs a 25 ms CPU segment from the trace and then executes EXEC program2,35: the ISR identifies a 15 MB image, frees memory, allocates partition 3, simulates the 225 ms loader, updates the PCB, and returns to user mode; program2's body then follows its script—CPU 35 → SYSCALL(6) → CPU 45 → SYSCALL(1) → CPU 30—appearing in the log as ISRs of 265 ms and 100 ms. Overall, the ordered kernel transitions, the child-first execution, and the fixed-partition choices (P4 for 10 MB, P3 for 15 MB) align with OS fork/exec semantics; the logs show that interrupt prologues are small while device delays and loader cost dominate runtime, exactly matching each device code's configured delay and the size-based loader rule.

Test 3: Two-stage FORK with 2 Child EXEC(program1 and program2)

The trace begins with a 10 ms CPU burst followed by FORK,20; the logs show kernel entry, vector lookup, and PCB cloning, after which the child runs first and performs EXEC program1,40. The simulator validates program1 as 10 MB, frees the current partition, allocates partition 4, simulates the 150 ms loader, updates the PCB, and IRET's to user mode; the program then executes its body exactly as scripted: CPU 70 → SYSCALL(7)=152 ms → CPU 50 → SYSCALL(3)=300 ms → CPU 40, matching the external file and device delays in the execution trace. The parent later issues a second FORK,18; that child executes EXEC program2,28, where the ISR identifies a 15 MB image, frees memory, allocates partition 3, incurs the 225 ms loader, updates the PCB, and returns to user mode before running CPU 55 → SYSCALL(8)=1000 ms → CPU 35 → SYSCALL(2)=150 ms → CPU 45; finally, the parent completes a trailing CPU 30 segment. System-status snapshots confirm the expected partitioning (P4 for 10 MB, P3 for 15 MB) and process states at each milestone, and the ordered kernel transitions demonstrate correct OS semantics.

Test 4: Child EXEC(porgram1) and Parent EXEC(program2) with mixed device ISRs

The trace issues FORK,12, after which the logs show kernel entry, vector lookup, and PCB cloning; the child is scheduled first and executes EXEC program1,22, which the simulator resolves as a 10 MB image: it frees the old partition, allocates partition 4, simulates the 150 ms loader, updates the PCB, and returns to user mode to run program1's body, matching the external program script and device delays in the execution log and snapshots. The parent then performs a CPU 15 segment and executes EXEC program2,18, identified as 15 MB; the ISR frees memory, allocates partition 3, simulates the 225 ms loader, updates the PCB, and returns to user mode, after which program2 runs as logged. Across both paths, the ordered kernel transitions and fixed-partition choices align with OS fork/exec semantics, while the logs confirm that interrupt prologues are small and loader time + device ISRs dominate runtime; system-status snapshots at key times (post-EXECs) show the expected PCB fields and occupancy, validating correct address-space replacement and scheduling behavior.

Test 5: Child(program3) and Parent(program4) EXEC, mixed devices and different image sizes

The trace starts with a 15 ms CPU burst followed by FORK,18; the logs show kernel entry, vector lookup, and PCB cloning, after which the child runs first and executes EXEC program3,20. The simulator validates program3 as 12 MB from the external file list, frees the current partition, allocates partition 3, and simulates the 180 ms loader before updating the PCB and returning to user mode; program3 then follows its body exactly, with each ISR bracketed by IRET as recorded in the execution log and reflected in the system-status snapshot at time 266 (PID 1 running program3 in partition 3). After the child completes, the parent performs a 25 ms CPU segment and then executes EXEC program4,22: the ISR identifies an 8 MB image, frees memory, allocates partition 5, simulates the 120 ms loader (8×15 ms), updates the PCB, and returns to user mode; program4 then runs, matching the device-delay table and confirming that loader time and device ISRs dominate wall-clock while interrupt prologues stay small. Overall, the ordered kernel transitions, fixed-partition choices, and child-first scheduling accurately model OS fork()/exec() semantics, with logs and snapshots aligning at each milestone.