Goal

the point of this assignment was to extend the interrupts simulator from a1 so it can handle fork() and exec(). the simulator now keeps track of pcbs, partitions, interrupt vectors and timing for each system call. basically it should show how a process is created, runs, calls syscalls, and loads new programs.

General Behaviour

every interrupt starts with the normal prelude: switch to kernel mode, context saved (10ms), find vector, load address. fork clones the pcb, assigns the child a best-fit partition based on the parent program size, and suspends the parent. exec loads a new program: bookkeeping (from the trace), loader = 15ms × program size, marking partition 3ms, update pcb 6ms, scheduler called, then iret. we only dump system_status.txt after fork or exec.anything after ENDIF runs twice (child first, parent later). the scheduler only shows up when we switch back to parent.

test 4: tail + device isr

trace:

```
CPU,10
FORK,12
IF_CHILD,0
EXEC progC,30
IF_PARENT,0
ENDIF,0
EXEC progP,25
CPU,20
```

external files: progC(10mb), progP(15mb)

results:

1. after fork the child takes partition 5 (8mb) and parent waits in 6 (2mb).
2. child exec progC -> loader 150ms, mark 3, update pcb 6. snapshot shows progC in partition 4, 10mb exact fit.
3. inside progC.txt it did a syscall(4) and end_io(4), each 250ms, so total time jumps a lot.
4. child runs the tail exec progP, loader 225ms -> partition 3 (15mb).
5. then scheduler swaps to parent and parent runs the same exec progP → partition 2 (25mb, 10mb free).
6. no weird scheduler line before the child tail so flow is correct.

takeaways: best-fit picked 4->10mb, 3 -> 15mb, 2 -> 25mb. device_table values clearly stretch runtime. all snapshots matched expected partitions and sizes.

test 5: fork-after-exec

trace:

```
EXEC base10,25
FORK,15
IF_CHILD,0
CPU,12
IF_PARENT,0
CPU,8
ENDIF,0
EXEC util15,20
```

external files: base10(10mb), util15(15mb)

results:

1. exec base10(25) put the first program into partition 4 (10mb).
2. fork cloned that image -> child got best-fit partition 3 (15mb) labelled base10, parent waiting.
3. child did cpu 12, parent did cpu 8.
4. after endif, both ran exec util15 (child first). loader 225ms, mark+pcb updates, scheduler, iret, same as before.
5. in base10.txt, syscall(6) and end_io(6) each took 265ms which matches the device table.
6. final snapshots show proper partitions (2,3,4 used) and correct sizes.

the simulator behaves like a simplified os: handles fork, exec, interrupt flow and memory management correctly. logs clearly show context switches, syscall durations, and memory updates. both tests prove the timing and control flow are right. i think it’s working fine now.