Part3: Report

Brodie Macleod 101302779

Sahil Todeti - 101259541 (Student 2)

https://github.com/auriza-jpg/SYSC4001_A2_P3.git

https://github.com/auriza-jpg/SYSC4001_A2_P2.git

This assignment section involved extending the existing Interrupts-API simulator from the previous assignment to include FORK and EXEC system calls.

Analyze results (From Test8). This Test Case contains many of the features contained in smaller ones

Init (Trace.txt)	Program 2	Program 3	CPU Bursts Programs

FORK, 10 IF_CHILD, 0 EXEC program1, 20 CPU, 8 EXEC program2, 15 IF_PARENT, 0 EXEC program3, 25 CPU, 20 ENDIF, 0 CPU, 12	CPU, 10 FORK, 8 IF_CHILD, 0 EXEC program4, 18 IF_PARENT, 0 CPU, 6 ENDIF, 0 CPU, 5	CPU, 10 FORK, 9 IF_CHILD, 0 EXEC program5, 16 IF_PARENT, 0 CPU, 7 ENDIF, 0 CPU, 8	Program 1: CPU, 12 Program 4: CPU, 14 Program 5: CPU, 11

Notes and explanations reference the timestamp of a given line.

<table>
<tr><th>Execution.txt (For test_8) + More Notes</th><th>Notes, Explanations and System Status</th></tr>
<tr><td>0, 1, switch to kernel mode
1, 10, context saved
11, 1, find vector 2 in memory position 0x0004
12, 1, load address 0X0695 into the PC
13, 10, Cloning the PCB</td><td>Calling Fork (init)
” SYSCALL” (Save context, vector to address, run ISR)

- ISR copies the information needed from the PCB of the parent process to the child process. In the simulation, a new PCB block initialized with fields from the parent block. As the Partitions are fixed, we need to place it in an empty partition. The new PID is acquired by incrementing the largest current PID</td></tr>
<tr><td>23,0, Scheduler called
23, 1, IRET
**************************************</td><td>time: 24; current trace: FORK, init, 10 (Result of calling fork)
+--+
| PID |program name |partition number | size | state |
+--+
| 1 | init | 5 | 1 | running |
| 0 | init | 6 | 1 | waiting |
+--+
Note: Incremented PID and new partition. Same size and same name (copied from parent). Default interrupt return. Child init is running while the parent is waiting.

Here, the child process is interacting with conditionals. To assign IF_CHILD and post-ENDIF instructions to the child program, the simulator uses tags to build a child_trace from the parents, ignoring IF_PARENT to ENDIF. The child's trace is built as [EXEC], skips adding [CPU8, EXEC] and breaks to begin executing the trace. This involves a recursive call to parse_trace () with [EXEC] and the wait queue + parent PCB</td></tr>
<tr><td>24, 1, switch to kernel mode
25, 10, context saved
35, 1, find vector 3 in memory position 0x0006
36, 1, load address 0X042B into the PC
37, 20, Program is 5 Mb large
57, 75, loading program into memory
132, 4, marking partition as occupied
136, 4, updating PCB
140, 0, scheduler called
140, 1, IRET</td><td>Calling Exec program1, 20

Like fork, the system call interface loads the EXEC ISR, taking the program name/Address as a parameter. The simulator queries the external_files table to obtain the new program’s size, and loads it into memory taking 15 * size ms
The old memory partition is freed to represent the process of freeing heap, stack and pointers from the PCB
A new partition is allocated based on the program size, representing assigning a new stack, heap etc to the new program.
The PCB is updated to contain the new program's name

****
Return to user mode as the new program is being executed. The simulator searches for the new program's trace (simulating adding the program to the process's memory). Another recursive call to parse_trace with the current wait queue and the program's loaded trace.</td></tr>
</table>

<table>
<tr>
<td></td>
<td>time: 365; current trace: EXEC, program3, 25
\--+
| PID |program name |partition number | size | state |
+--+
| 0 | program3 | 4 | 9 | running |
+--+</td>
</tr>
<tr>
<td>177, 1, switch to kernel mode
178, 10, context saved
188, 1, find vector 3 in memory position 0x0006
189, 1, load address 0X042B into the PC
190, 25, Program is 9 Mb large
215, 135, loading program into memory
350, 7, marking partition as occupied
357, 7, updating PCB
364, 0, scheduler called
364, 1, IRET

365, 10, CPU Burst
375, 1, switch to kernel mode
376, 10, context saved
386, 1, find vector 2 in memory position 0x0004
387, 1, load address 0X0695 into the PC
388, 9, Cloning the pcb
397,0, Scheduler called
397, 1, IRET
398, 1, switch to kernel mode
399, 10, context saved
409, 1, find vector 3 in memory position 0x0006
410, 1, load address 0X042B into the PC
411, 16, Program is 4 Mb large
427, 60, loading program into memory
487, 8, marking partition as occupied
495, 8, updating PCB
503, 0, scheduler called
503, 1, IRET
504, 11, CPU Burst
913, 7, CPU Burst
920, 8, CPU Burst</td>
<td>Calling EXEC for PID 0 (root parent), loading program 3 into the parent's PCB. It is visible that the further IF_CHILD CPU and EXEC instructions were not executed as the child process terminated. This is also visible in the system_status update when exec finishes, PID 0, now containing program 3, is the only active PCB in the table.
time: 365; current trace: EXEC, program3, 25 (State after exec)
+--+
| PID |program name |partition number | size | state |
+--+
| 0 | program3 | 4 | 9 | running |
+--+

365 begins executing program 3’s (PID 0) 10ms of CPU burst
375 Fork is then called, creating a child (PID 1)

time: 398; current trace: FORK, program3, 9 (State after Fork)
+--+
| PID |program name |partition number | size | state |
+--+
| 1 | program3 | 3 | 9 | running |
| 0 | program3 | 4 | 9 | waiting |
+--+
398 EXEC on PID 1, conditional instruction from the program3’s IF_CHILD trace

time: 504; current trace: EXEC, program5, 16
+--+
| PID |program name |partition number | size | state | (State after EXEC)
+--+
| 1 | program5 | 3 | 4 | running |
| 0 | program3 | 4 | 9 | waiting |
+--+

504 executing program 5’s 11ms of CPU Burst, and returning control to parent (PID 0), within the IF_PARENT conditional (913, 7, CPU 7), and then exiting the IF_DEF to terminate after executing 920, 8, CPU burst.</td>
</tr>
</table>

There were certain required test cases which must be run, along with questions in the form of comments. The first was given by **TestCase1**

Init PCB block

```
FORK, 10 //fork is called by init
IF_CHILD, 0 EXEC program1, 50 //child executes program1
```

```
IF_PARENT, 0
EXEC program2, 25 //parent executes program2
ENDIF, 0 //rest of the trace doesn't really matter (why?)
```

Contents of program1:

```
CPU, 100
```

Contents of program2:

```
SYSCALL, 4
```

OUTPUT FROM OutputFiles/_test_1_system_status.txt

```
time: 24; current trace: FORK, init, 10                 NOTES:
+------------------------------------------------------+
| PID |program name |partition number | size |   state |
+------------------------------------------------------+
|   1 |        init |               5 |    1 | running |   init 5 (child) and init 6 (parent)
|   0 |        init |               6 |    1 | waiting |
+------------------------------------------------------+
time: 246; current trace: EXEC, program1, 50             **if child block*
+------------------------------------------------------+
| PID |program name |partition number | size |   state |
+------------------------------------------------------+   child runs to completion (program1)
|   1 |    program1 |               4 |   10 | running |
|   0 |        init |               6 |    1 | waiting |
+------------------------------------------------------+
time: 648; current trace: EXEC, program2, 25                   **if_parent block**
+------------------------------------------------------+
| PID |program name |partition number | size |   state |
+------------------------------------------------------+    parent is running to completion
|   0 |    program2 |               3 |   15 | running |
+------------------------------------------------------+
```

The rest of the trace after ENDIF would not matter, as all PCBs that could potentially reach past the conditionals have replaced their memory with a new program. This is demonstrated in test_case9

```
… (same as test_case1)
ENDIF
CPU 55
```

With execution.txt output never reaching the CPU call:

```
…
661, 250, SYSCALL ISR_Activities
911, 1, IRET    //Note: IRET is associated with program2's SYSCALL.
```

Other individual testing, with various programs, to see how variables affected the simulation. This can be viewed in test cases 4 to 8. These were created to test performance variations given variable context switch overheads, as well as IO versus CPU-bound processes, and slower versus faster loading speeds. **Slower load-per-MB time** makes EXEC-heavy traces significantly longer, especially when large programs (20 MB) are repeatedly loaded. **IO-bound tests** are most affected by context switch overhead, increasing or decreasing significantly. The most drastic differences in performance time were seen in tests that involved forking and executing several large file-sized IO-heavy programs, and increasing or decreasing the loading speed, as well as the context switch overhead. Context switching also heavily affected exec or fork-heavy tasks, as many SYSCALLs are needed. Overall, across all test cases, modifying the load speed saw the most dramatic performance increase.