

Computer Operating Systems Assignment 1
Project Report

BLG 312E

CRN: 23148

Beyza Aydeniz - 150200039

April 2, 2023

1 Introduction

Our objective for this task is to utilize the "fork()" system call in order to construct a specific process hierarchy, as illustrated in the figure below, with the aim of enhancing our comprehension of process management within operating systems. This homework assignment consist of two questions.

2 Question 1

Question one requires us to create a tree structure where each left child of a parent has no child, and the depth of right children is determined by the user as a parameter. The depth of left children is fixed at 1. Our task is to develop a code that can construct the specified process tree topology based on the user's input parameters.

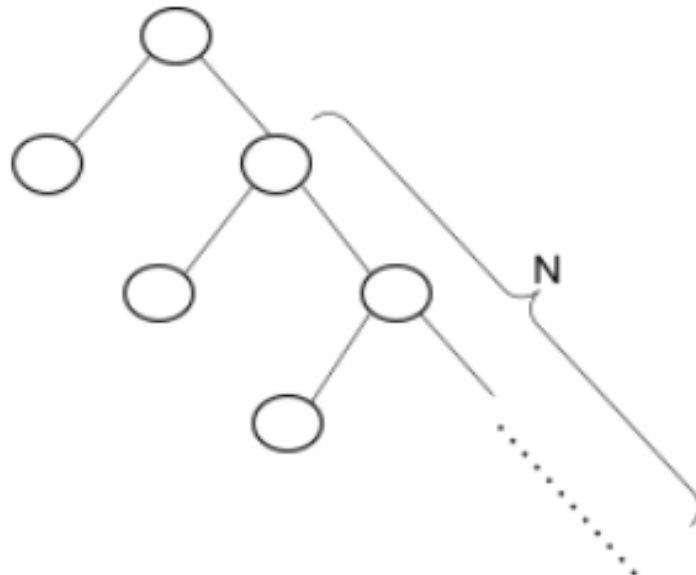


Figure 1: Question 1 Process Hierarchy

For my solution to question 1, I employed a recursive function. Initially, I utilized the "fork()" system call to generate a left child process and display its ID and its parent's ID. Since a left child is not allowed to have any children, after its creation, we returned to the parent process by using the "rc != 0" condition and subsequently utilized the "fork()" function to create a right child process. By restricting "rc != 0", we ensured that the "fork()" operation did not impact the left child process.

Once the right child process is generated, the "rc==0" condition is utilized. At this point, I first display the ID of the root of the tree. Then, I call the recursive function to create a pair of left and right children for every right child process. To prevent an infinite loop, I decrement the depth parameter by 1 each time the recursive function is called, stopping when the depth reaches 0. Even when the depth parameter reaches 0, the left child of the right child process is still generated.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Print the right depth of the tree: 3
Root: 2467
Left child (id: 2505) of 2467
Right child (id: 2506) of 2467
Left child (id: 2507) of 2506
Right child (id: 2508) of 2506
Left child (id: 2509) of 2508
Right child (id: 2510) of 2508
Left child (id: 2511) of 2510
[1] + Done "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-gxmbxri4.duf" 1>"/tmp/Microsoft-MIEngine-Out
-031k5y43.c2f"
test@blg223e:~/hostvolume/OS$

```

Figure 2: Sample question 1 output.

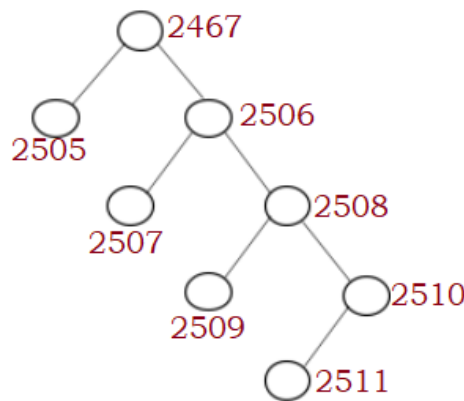


Figure 3: Sample question 1 output tree with N=3.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Print the right depth of the tree: 5
Root: 2588
Left child (id: 2629) of 2588
Right child (id: 2630) of 2588
Left child (id: 2631) of 2630
Right child (id: 2632) of 2630
Left child (id: 2633) of 2632
Right child (id: 2634) of 2632
Left child (id: 2635) of 2634
Right child (id: 2636) of 2634
Left child (id: 2637) of 2636
Right child (id: 2638) of 2636
Left child (id: 2639) of 2638
[1] + Done "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-nhxpqdr0.xpi" 1>"/tmp/Microsoft-MIEngine-Out
-lwdrc0ip.2up"
test@blg223e:~/hostvolume/OS$

```

Figure 4: Sample question 1 output.

The expression of how many of the created processes can be identified as parent processes depending on value N is "N".

3 Question 2

For the second question, I implemented two recursive void functions. The primary difference from the first question is that left children can now have their own left child, and we now require two parameters: the left depth of every right child and its parent, as well as the right depth. I modified the previous code slightly by introducing a temporary variable to store the left depth, and updating the number of recursive function parameter from 1 to 2. Additionally, I added the "left_part" function to generate left children with the required depth. Within the "left_part" function, I employed the "fork()" system call to create a left child and utilized the "rc==0" condition to employ recursion to achieve the desired depth. I also used "wait(NULL)" to ensure that the parent processes wait for their children to terminate before exiting.

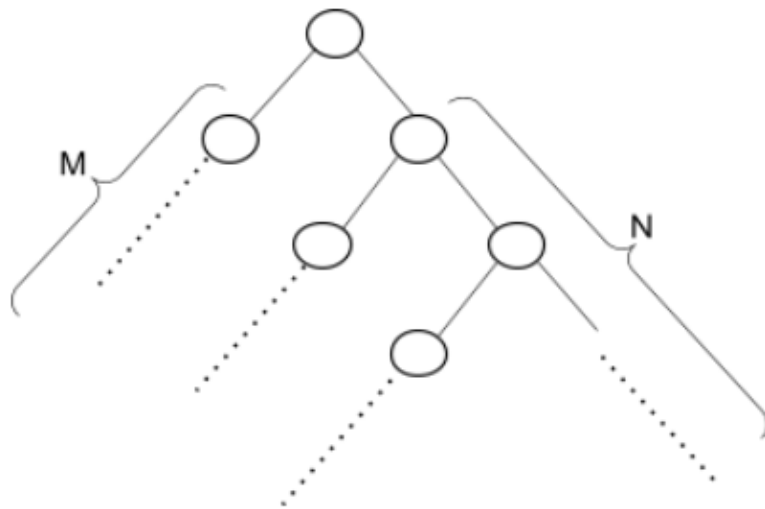


Figure 5: Question 2 Process Hierarchy

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Print the right depth of the tree: 2
Print the left depth of the tree: 3
Left child (id: 2802) of 2764
Left child (id: 2804) of 2802
Left child (id: 2805) of 2804
Right child (id: 2803) of 2764
Left child (id: 2806) of 2803
Right child (id: 2807) of 2803
Left child (id: 2808) of 2806
Left child (id: 2809) of 2807
Left child (id: 2810) of 2808
Left child (id: 2811) of 2809
Left child (id: 2812) of 2811
[1] + Done      "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-kb
wcitj3.ibt" 1>"/tmp/Microsoft-MIEngine-Out-emfrn5xm.3er"
test@blg223e:~/hostvolume/OS$
```

Figure 6: Sample question 2 output.

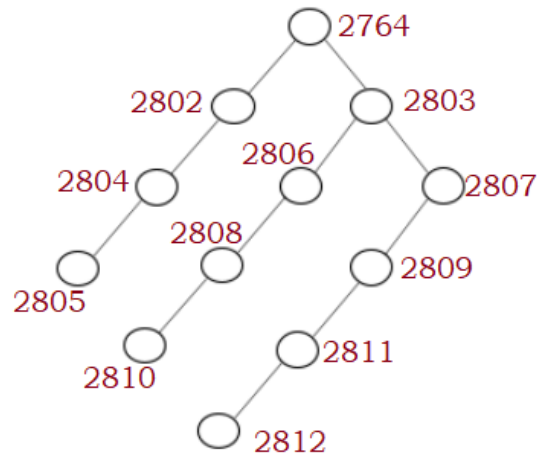


Figure 7: Sample question 2 output tree with N=2 and M=3.

```

Print the right depth of the tree: 3
Print the left depth of the tree: 4
Left child (id: 2906) of 2880
Left child (id: 2908) of 2906
Right child (id: 2907) of 2880
Left child (id: 2909) of 2908
Left child (id: 2910) of 2909
Left child (id: 2911) of 2907
Right child (id: 2912) of 2907
Left child (id: 2913) of 2911
Left child (id: 2914) of 2912
Right child (id: 2915) of 2912
Left child (id: 2916) of 2913
Left child (id: 2917) of 2914
Left child (id: 2918) of 2915
Left child (id: 2919) of 2916
Left child (id: 2920) of 2917
Left child (id: 2921) of 2918
Left child (id: 2922) of 2920
Left child (id: 2923) of 2921
Left child (id: 2924) of 2923
[1] + Done
-hmdcw24.1in
test@big223e:~/hostvolume/OS$

```

Figure 8: Sample question 2 output.

The expression of how many of the created processes can be identified as parent processes depending on N and M value is $(N+1) \times M - 1$.