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 spesific 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.

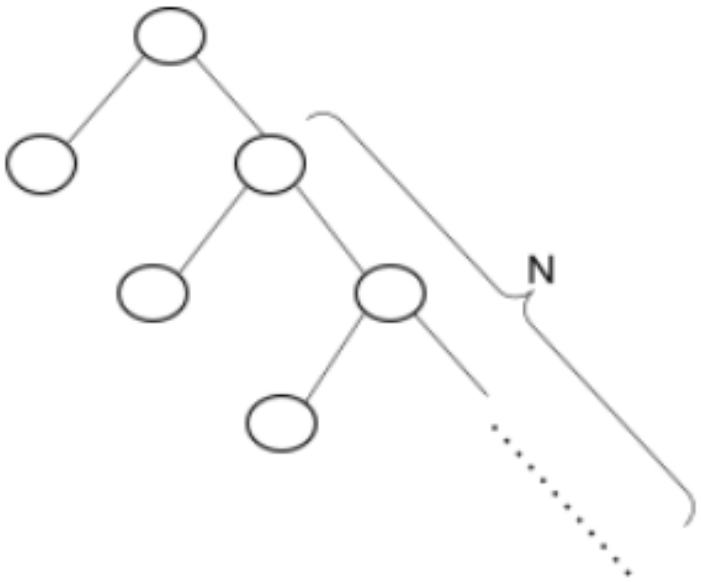

Figure 1: Question 1 Process Hierarchy

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 "rcc ¿ 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.

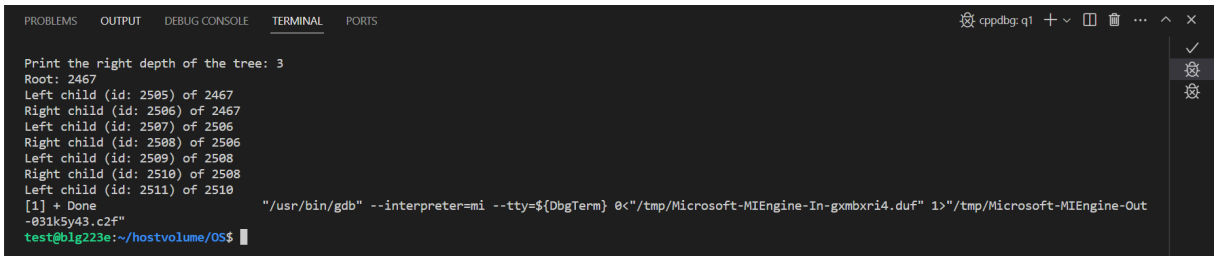

Figure 2: Sample question 1 output.

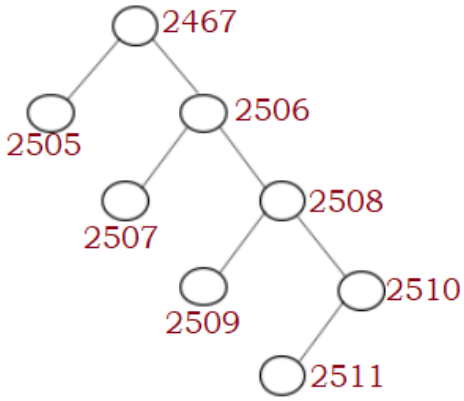

Figure 3: Sample question 1 output tree with N=3.

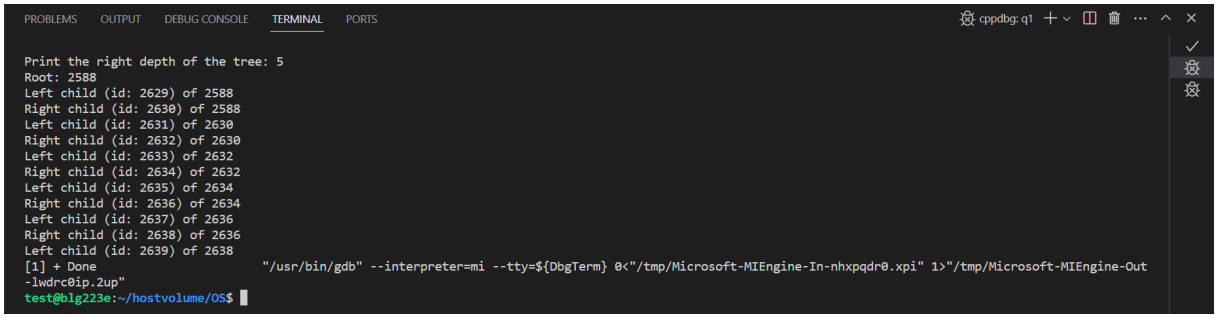

Figure 4: Sample question 1 output.

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.

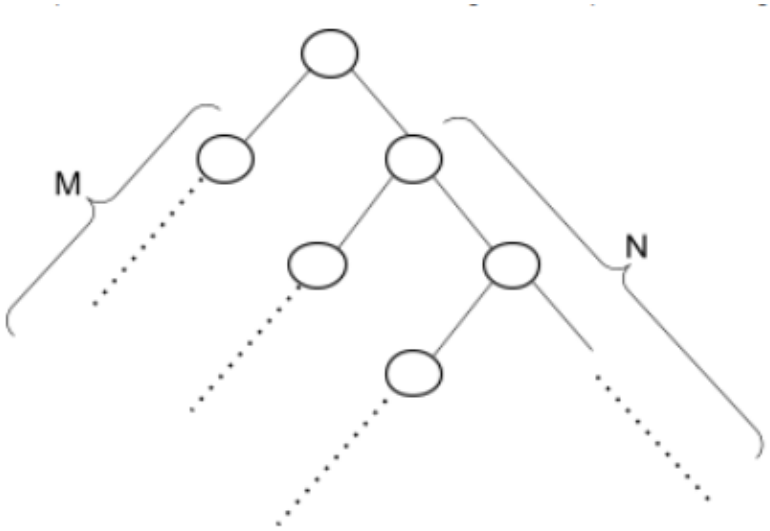

Figure 5: Question 2 Process Hierarchy

Figure 6: Sample question 2 output.

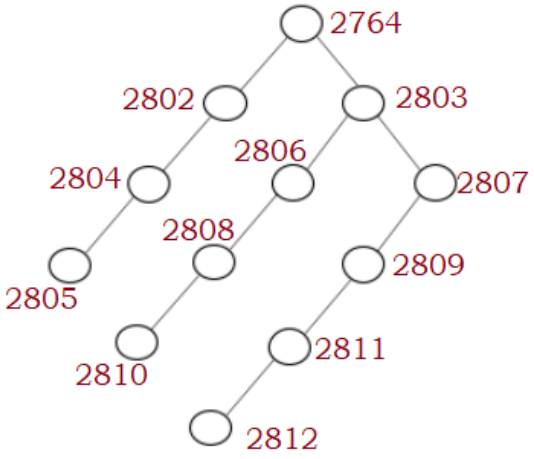

Figure 7: Sample question 2 output tree with N=2 and M=3.

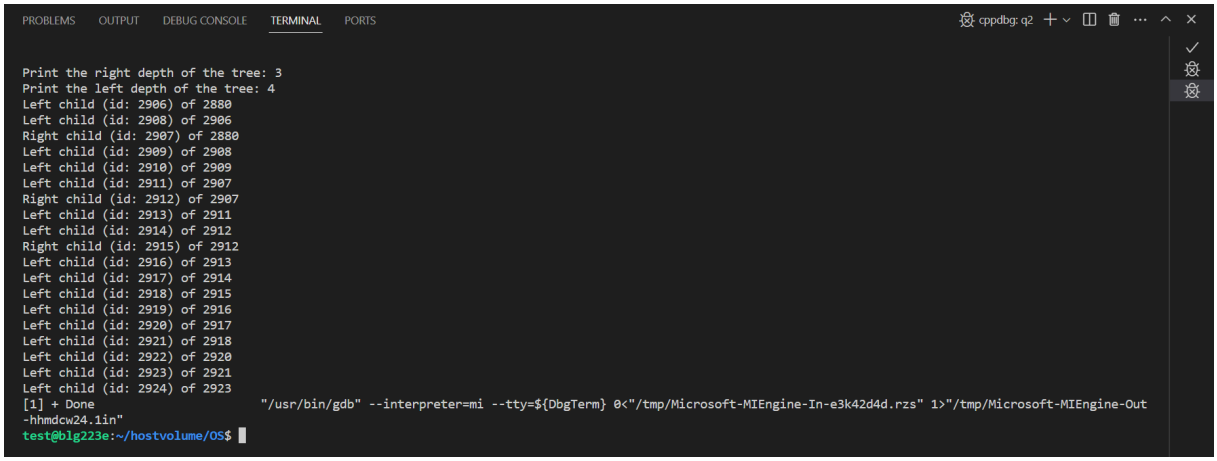

Figure 8: Sample question 2 output.

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