BLG312E - Computer Operating Systems - HW1

Enis Teper - 504221511

1 Introduction

In this homework it is required from us to implementing fork() function to create tree based with respect to input from the user. Implementing this structure will improve our experience about the usage of the fork() function.

2 Question 1

In question 1 the input will be **N** and actually it represents edge number between parent nodes. In 1 graph is shown for **N=2**. As can be seen from the graph there are 2 edges colored as red which connects 3 parents at total. Because of P2 and P3 processes are child of their previous processes and they have child processes we can call them as both child and parent. However, since we are considering to calculate parent processes we will call them as a parent.

From the assumption of the N is equal to edge number between parents, parent process number can be found as Equation 1.

$$Parent\ Process\ Number = N + 1 \tag{1}$$

Also, total process number can be calculated as given at Equation 2.

$$Total\ Process\ Number = 2(N + 1) \tag{2}$$

Finally, from these equations we can say that N + 1 of 2(N+1) process will be parent process.

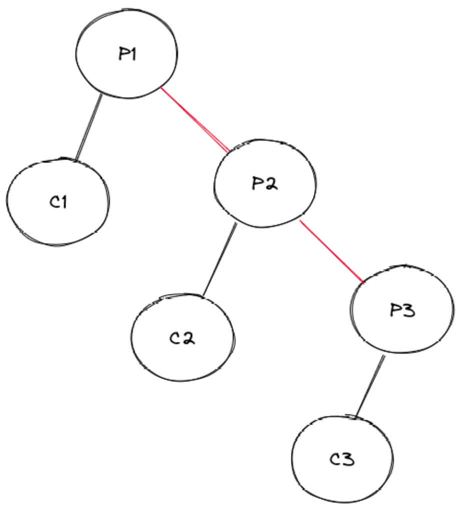
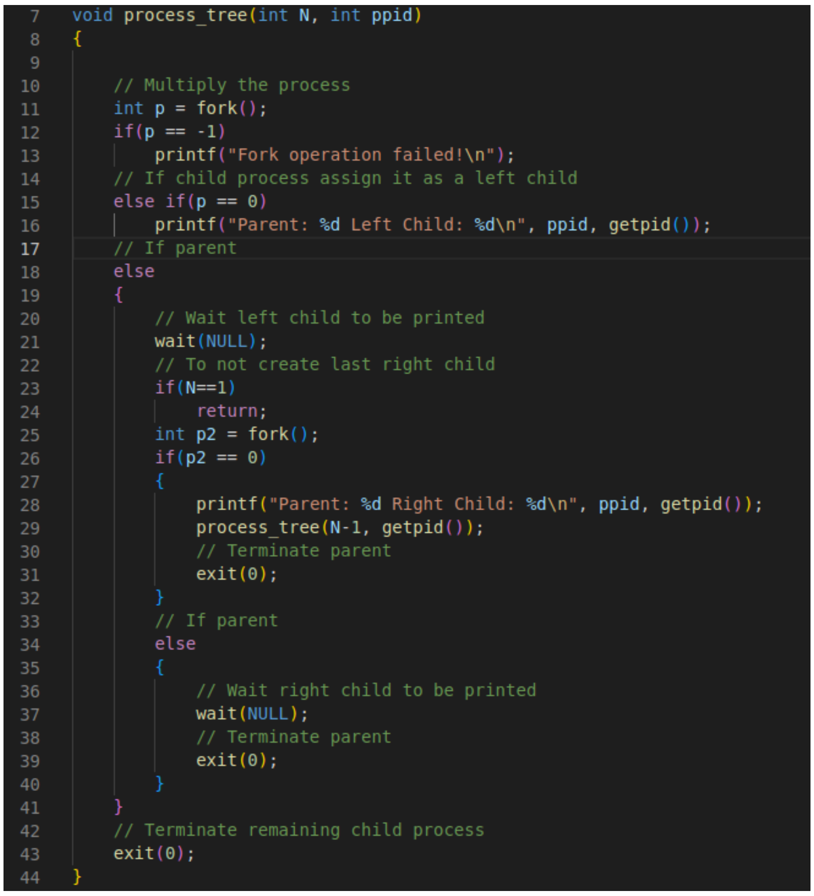


Figure 1: Question 1 Graph

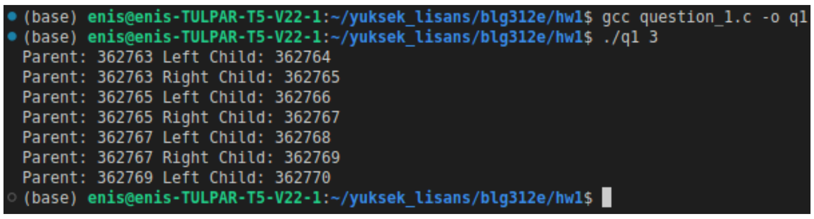
At 5 code solution of this question, **N** number is obtained as an input from the user. After that, PID of the main process is given as parameter to **process_tree** function with **N**. The function recursively creates left child and right child for each parent until **N** equals to 1. When the N becomes 1 it does not create right child but only left child. At the final, each process is exited with **exit(0)** command. The result for the N=3 is shown at 6.



```c
void process_tree(int N, int ppid)
{

    // Multiply the process
    int p = fork();
    if(p == -1)
        printf("Fork operation failed!\n");
    // If child process assign it as a left child
    else if(p == 0)
        printf("Parent: %d Left Child: %d\n", ppid, getpid());
    // If parent
    else
    {
        // Wait left child to be printed
        wait(NULL);
        // To not create last right child
        if(N==1)
            return;
        int p2 = fork();
        if(p2 == 0)
        {
            printf("Parent: %d Right Child: %d\n", ppid, getpid());
            process_tree(N-1, getpid());
            // Terminate parent
            exit(0);
        }
        // If parent
        else
        {
            // Wait right child to be printed
            wait(NULL);
            // Terminate parent
            exit(0);
        }
    }
    // Terminate remaining child process
    exit(0);
}
```

Figure 2: Question 1 Code



```
(base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ gcc question_1.c -o q1
(base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ ./q1 3
Parent: 362763 Left Child: 362764
Parent: 362763 Right Child: 362765
Parent: 362765 Left Child: 362766
Parent: 362765 Right Child: 362767
Parent: 362767 Left Child: 362768
Parent: 362767 Right Child: 362769
Parent: 362769 Left Child: 362770
(base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$
```

Figure 3: Question 1 Result for N=3

3 Question 2

Question 2 is a extended version of Question 1. In this question additionally to **N**, **M** is also required from the user to generate extended tree. Similarly to **N**, **M** value represents edges between left sub processes which is shown at 1.

From the assumption of the **N** is equal to edge number between main parents, and **M** is equal to edge number between sub-parents total parent process number can be found as Equation 3. These equations are valid since we know N and M values are positive integers. For this reason zero or minus inputs are not considered.

$$Parent\ Process\ Number = M(N+1) \tag{3}$$

Also, total process number can be calculated as given at Equation 2.

$$Total\ Process\ Number = (M+1)(N+1) \tag{4}$$

Finally, from these equations we can say that **M(N+1)**of **(M+1)(N+1)** process will be parent process.

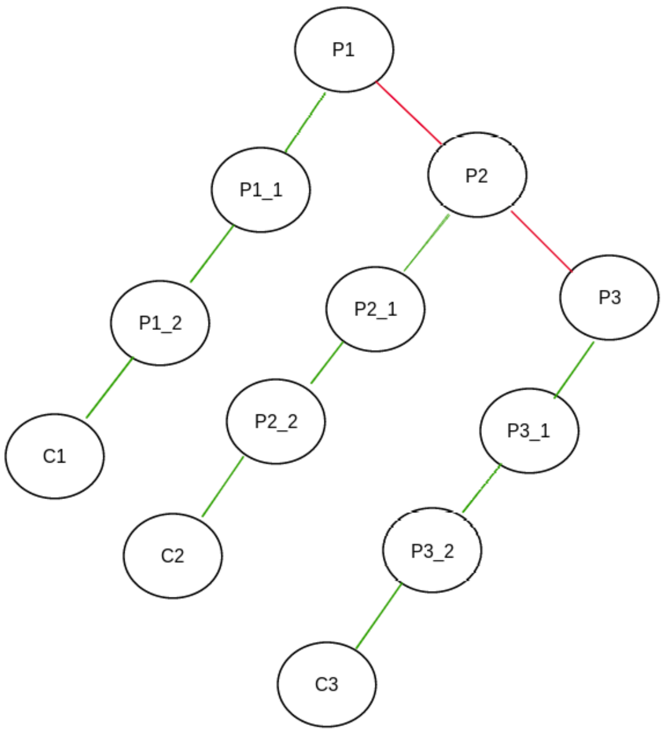
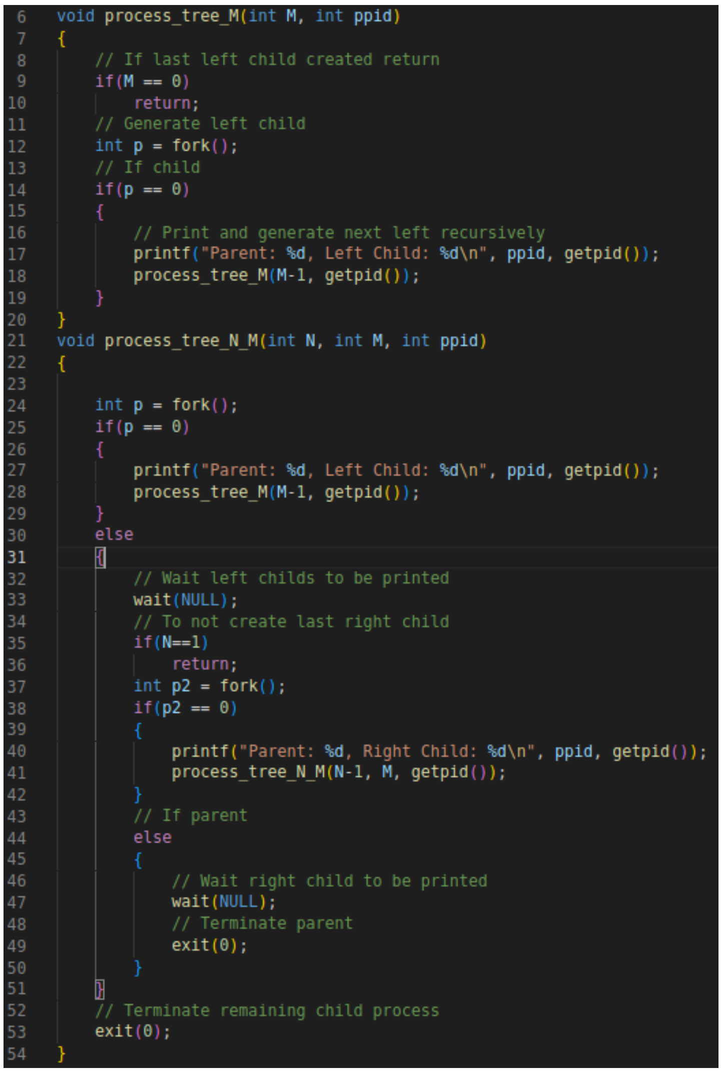


Figure 4: Question 1 Graph

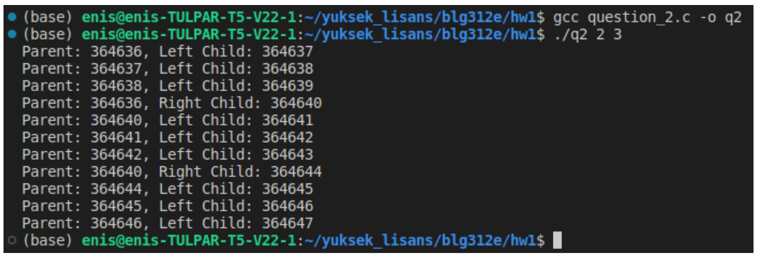
Similar to Question 1 solution, N, M inputs and main process PID values are passed to **process_tree_N_M** function. Then, with respect to **M** value left sub processes are created with the **process_tree_M** function. After creating all left sub process of that level algorithm creates right sub child to continue next level. Thus, the algorithm generates all processes.



```c
void process_tree_M(int M, int ppid)
{
    // If last left child created return
    if(M == 0)
        return;
    // Generate left child
    int p = fork();
    // If child
    if(p == 0)
    {
        // Print and generate next left recursively
        printf("Parent: %d, Left Child: %d\n", ppid, getpid());
        process_tree_M(M-1, getpid());
    }
}
void process_tree_N_M(int N, int M, int ppid)
{

    int p = fork();
    if(p == 0)
    {
        printf("Parent: %d, Left Child: %d\n", ppid, getpid());
        process_tree_M(M-1, getpid());
    }
    else
    {
        // Wait left childs to be printed
        wait(NULL);
        // To not create last right child
        if(N==1)
            return;
        int p2 = fork();
        if(p2 == 0)
        {
            printf("Parent: %d, Right Child: %d\n", ppid, getpid());
            process_tree_N_M(N-1, M, getpid());
        }
        // If parent
        else
        {
            // Wait right child to be printed
            wait(NULL);
            // Terminate parent
            exit(0);
        }
    }
    // Terminate remaining child process
    exit(0);
}
```

Figure 5: Question 1 Code

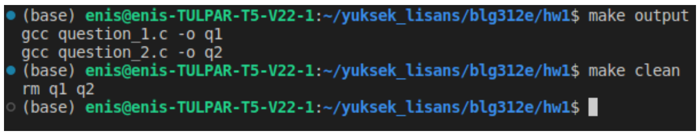


```
● (base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ gcc question_2.c -o q2
● (base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ ./q2 2 3
  Parent: 364636, Left Child: 364637
  Parent: 364637, Left Child: 364638
  Parent: 364638, Left Child: 364639
  Parent: 364636, Right Child: 364640
  Parent: 364640, Left Child: 364641
  Parent: 364641, Left Child: 364642
  Parent: 364642, Left Child: 364643
  Parent: 364640, Right Child: 364644
  Parent: 364644, Left Child: 364645
  Parent: 364645, Left Child: 364646
  Parent: 364646, Left Child: 364647
○ (base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ 
```

Figure 6: Question 1 Result for N=2 and M=3

4 Conclusion

In this homework, tree based parent-child relations are generated with **fork()** function. In order to compile the code makefile can be used for both generating and cleaning output files. Usage of it is shown as below.



```
● (base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ make output
  gcc question_1.c -o q1
  gcc question_2.c -o q2
● (base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ make clean
  rm q1 q2
○ (base) enis@enis-TULPAR-T5-V22-1:~/yuksek_lisans/blg312e/hw1$ █
```

Figure 7: Makefile Usage Example