Operating Systems Assignment-1

Aybike Zeynep Taşkın - 150190004

1 Introduction

In this assignment we make use of the fork() system call that we learned to create different process trees.

2 Question 1

2.1 Process Tree

In the first question I created a process tree using fork(). Right sub-tree has depth of N which is a positive value that is a user input value. Left sub-tree is fixed with depth of 1. In my code in main function I take N as user input with scanf() function. Then I do all my implementation in "right-tree" function that takes N as the input. In the function at first I create a fork() process and create one left child as usual if fork is equal to zero. Then I kill the left child because we wont continue from there any further. In the parent process I created an if statement that checks whether N is greater than zero. If it is, I create another fork process and if it is equal to zero; I print the right child, decrement N (so that we can do the implementation correctly for N times), and call the "right-tree" function again with the updated N value. With that way, I implemented a recursive function that creates N right children. In the end, we obtain the required tree.

```
if(N>0){
    pid2 = fork();
    if(pid2 < 0){
        fprintf(stderr, "Fork Failed");
        exit(1);
    }
    //right child
    else if(pid2 == 0){
        printf("Right child %d with parent %d is created.
            N=%d\n", getpid(), getppid(), N);
        N--;
        right_tree(N);
    }
    //parent
```

```
        else{
            wait(NULL);
            exit(0);
        }
    }
```

I created a Makefile for my code so it can be run as the example below and the output of the example tree with N=3 can be seen as follows:

```
vagrant@vagrant-VirtualBox:~/Masaüstü/OS$ make q1
make: 'q1' is up to date.
vagrant@vagrant-VirtualBox:~/Masaüstü/OS$ ./q1
Enter positive integer N value: 3
Left child 3490 with parent 3489 is created. N=3
Right child 3491 with parent 3489 is created. N=3
Left child 3492 with parent 3491 is created. N=2
Right child 3493 with parent 3491 is created. N=2
Left child 3494 with parent 3493 is created. N=1
Right child 3495 with parent 3493 is created. N=1
Left child 3496 with parent 3495 is created. N=0
vagrant@vagrant-VirtualBox:~/Masaüstü/OS$
```

Figure 1: Output of part 1

The process tree of the example with process ID's that was outputted on the console when N=3 can be seen below:

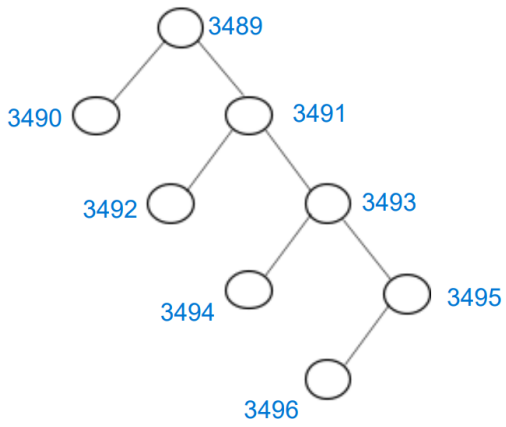

Figure 2: Process tree of the example

2.2 Expression of parent processes

All processes except for left children can be considered as parent processes since they all have children that are derived from them. Also the number of parents can be seen on the console output too by counting the number of parents with unique ID's. We have N right children created and we also have the starting

parent. So the expression for number of parent processes can be derived as: **N+1**

3 Question 2

3.1 Process Tree

In the second question we extend the question and create process tree with left sub-tree of depth M and right sub-tree with length N. My thought process for this question was to create left children for each right node M times with a for loop, and implement N right children recursively as we implemented in the first question. In my code again I take M and N as user inputs in main function. Then I created a function called "left-right-tree". I create a process and if it is equal to zero, I start the creation of the left children. For that, I created a for loop that creates different processes for M-1 times and creates M left children of the parent. The parent process of my function is exactly the same with the first question except for I give decremented N and the same M values to the recursive function. In the end, I create N right children each having M depth sub-trees. While I was writing my code, I encountered with a problem that I later fixed. I created the correct amount of children but right children did not have the correct parents (because somewhere in the code I was unintentionally creating unnecessary processes), and the first left child that came after the right child also did not have the correct parent id. Later I solved this problem by creating the first left child outside of the for loop before creating a new process, then in the for loop I iterated over M-1 times and created new processes for new left children. That solved my problem.

```
//child process (left child)
else if(pid==0){
    printf("Left child %d with parent %d is created\n",
        getpid(), getppid());
    for(int i=1; i<M; i++){
        pid1 = fork();
        if(pid1 == 0){
            printf("Left child %d with parent %d is created\n",
                getpid(), getppid());
            //exit(0);
        }
        //parent
        else{
            wait(NULL);
            exit(0);
        }
    }
    exit(0);
}
```

The output and the process tree for when N=2 and M=3 can be seen below:

```
vagrant@vagrant-VirtualBox:~/Masaüstü/OS$ make q2
make: 'q2' is up to date.
vagrant@vagrant-VirtualBox:~/Masaüstü/OS$ ./q2
Enter positive integer N and M values separated by space: 2 3
Left child 4143 with parent 4142 is created
Left child 4144 with parent 4143 is created
Left child 4145 with parent 4144 is created
Right child 4146 with parent 4142 is created N=2
Left child 4147 with parent 4146 is created
Left child 4148 with parent 4147 is created
Left child 4149 with parent 4148 is created
Right child 4150 with parent 4146 is created N=1
Left child 4151 with parent 4150 is created
Left child 4152 with parent 4151 is created
Left child 4153 with parent 4152 is created
vagrant@vagrant-VirtualBox:~/Masaüstü/OS$
```

Figure 3: Output of part 2

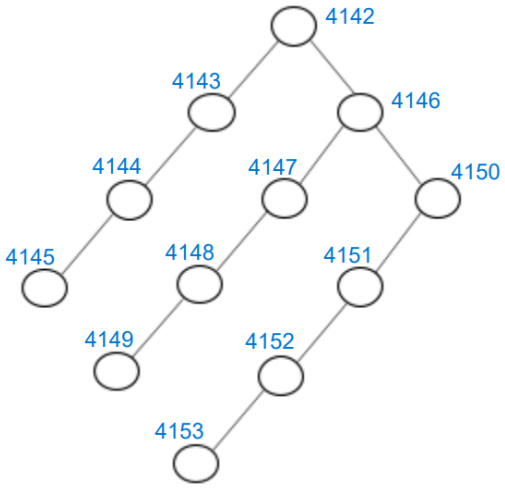

Figure 4: Process tree of the example

3.2 Expression of parent processes

All processes except for the last left children that is being created in each for loop can be considered as parent processes. The number of all processes is (N+1)*(M+1) and the number of all last left children is N+1. So the expression for the number of parent processes is: **(N+1)(M+1) - (N+1) = (N+1)*M**