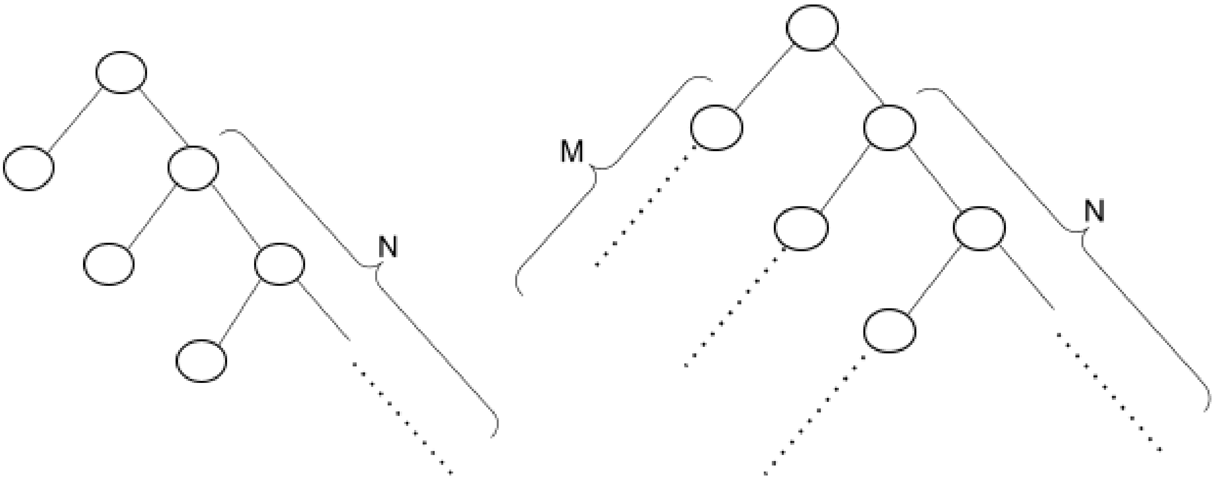
BLG 312E Computer Operating Systems Homework 1 Report

Fatih Baskın, 150210710

1st of April, 2023

1 Introduction

In this homework, I was asked to implement some code to generate a parametric process tree. For the first question, there is a parameter N which is the length of the right sub-tree and every right node has a left child node. Then, I was asked to improve on this code to add parameter M, the number of left child nodes. Program asks for parameters when it is running, from the terminal. Source code of those programs are q1.c and q2.c.



(a) First parametric tree (question 1)

(b) Second parametric tree (question 3)

Figure 1: These are the parametric trees that have been asked.

2 Question 1

In this question, I was asked to implement a tree in which there is a parameter N, the number of right child nodes, and each right node (including root) has a left child node. Each child process must print its process id (pid), parent process id (ppid), and terminate.

Each node is a process and child nodes (processes) are made using `fork()` system call. For preventing zombie processes, the parent process must wait for the child process to end using `wait()` system call. One downside of this approach is if the depth of the tree is high, there would be a lot of parent processes waiting for child processes to end, occupying memory.

The program behavior explained above can be easily generated using **preorder-traversal-like** process generation. Code is made using recursive `left_process()` and `right_process(int)` functions. Since these functions are recursive, child processes must terminate immediately when they are done, they should not pop back program stack and continue on previous functions. Therefore, `exit(EXIT_SUCCESS)` system call is used to terminate the child process (current process) on spot. Program name: q1, program flow:

1. Root process prints its details.

2. Root process calls `left_process()` function.
    (a) Inside the `left_process()` function, `fork()` system call is called, this returns a negative value if the process cannot be created, returns child process ID if the process can be created (this means the current process is the parent process), and returns 0 if the current process is the child process.
    (b) If `fork()` return value is negative, print error message.
    (c) If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
    (d) If `fork()` return value is zero, print current **pid** and **ppid**. Then terminates using `exit(EXIT_SUCCESS)` system call.
3. After the left sub-tree is handled, the program checks N value, if $N == 0$ then there is no need to proceed further, program terminates using `return 0` system call. Otherwise, the program proceeds further.
4. Program creates a new process using `fork()` system call.
5. If `fork()` return value is negative, print error message.
6. If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
7. If `fork()` return value is zero, this is a child process and process calls `right_process(N)` function to generate the right sub-tree. Then terminates using `exit(EXIT_SUCCESS)` system call.
    (a) First, the right process writes its details: pid and ppid.
    (b) Then, process calls `left_process()` to generate left sub-tree. The program flow of this function is explained in item number 2. Jump to 2.
    (c) After the left sub-tree is handled, decrement N, if $N == 0$, there is no need to proceed further, terminate this process using `exit(EXIT_SUCCESS)` system call. Otherwise, proceed further.
    (d) Process creates a new process using `fork()` system call.
    (e) If `fork()` return value is negative, print error message.
    (f) If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
    (g) If `fork()` return value is zero, this is a child process and process calls `right_process(N)` function to generate the right sub-tree. This will generate recursive function calls until N is equal to 0. Remember to check item number 7.c. Then terminate this process (child process) using `exit(EXIT_SUCCESS)` system call.
    (h) Terminate this process (parent process) using `exit(EXIT_SUCCESS)` system call.
8. Program terminates using `return 0`.

Using this, program flow, a process writes itself, waits for the left processes to end, waits for the right processes to end then finally terminates. Root-left-right, this is the logic of the preorder traversal.

3 Question 1 - Answer

There is a root process, there are $N (N \geq 0)$ right sub-processes. Each right sub-process and the root process have one left process. So those singe left-processes are the leaves. They don't have any sub-processes and other processes have a child so they can be designated as a parent process. There are $2 \times (N + 1)$ processes ($N + 1$ coming from root + N right processes) and $N + 1$ processes are leaves, so $N + 1$ processes can be designated as parent processes. If the root process is not considered a created process, N processes can be considered as a created parent process.

4 Question 2

In this question, I was asked to extend the program so that the left processes can be in M depth. The body of the `left_process(int)` is more like the previous question's `right_process` because forking is done in the parent process now. Also `right_process(int, int)` now accepts two parameters, N and M. Also, the body of the main program and `right_process(int, int)` is adjusted to incorporate the new left process generation schedule. But the logic of the program is still the same, preorder traversal. Program name: q2, program flow:

1. Root process prints its details.

2. Program checks the value of the M. If $M == 0$ then the program skips the left sub-tree generation, jumping to item number 7.
3. Program creates a new process using `fork()` system call.
4. If `fork()` return value is negative, print error message.
5. If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
6. If `fork()` return value is zero, this is a child process and process calls `left_process(M)` function to generate the left sub-tree. Then terminates using `exit(EXIT_SUCCESS)` system call.
   (a) First, the left process prints its details: pid and ppid.
   (b) Then, process decrements the M, if $M == 0$ then process terminates here using `exit(EXIT_SUCCESS)`. Otherwise, the process proceeds further.
   (c) Program creates a new process using `fork()` system call.
   (d) If `fork()` return value is negative, print error message.
   (e) If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
   (f) If `fork()` return value is zero, this is a child process and process calls `left_process(M)` function to generate the left sub-tree. This will generate recursive function calls until M is equal to 0. call of Remember, M was decremented in item number 6.b. Then terminate this process (child process) using `exit(EXIT_SUCCESS)` system call.
   (g) Terminate this process (parent process) using `exit(EXIT_SUCCESS)` system call.
7. After the left sub-tree is handled, the program checks N value, if $N == 0$ then there is no need to proceed further, program terminates using `return 0` system call. Otherwise, the program proceeds further.
8. Program creates a new process using `fork()` system call.
9. If `fork()` return value is negative, print error message.
10. If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
11. If `fork()` return value is zero, this is a child process and process calls `right_process(N, M)` function to generate the right sub-tree. Then terminates using `exit(EXIT_SUCCESS)` system call.
    (a) First, the right process writes its details: pid and ppid.
    (b) Process checks the value of the M. If $M == 0$ then the program skips the left sub-tree generation, jumping to item number 11.g.
    (c) Process creates a new process using `fork()` system call.
    (d) If `fork()` return value is negative, print error message.
    (e) If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
    (f) If `fork()` return value is zero, this is a child process and process calls `left_process(M)` function to generate the left sub-tree. Details of this function were described in item number 6. Then terminate this process (child process) using `exit(EXIT_SUCCESS)` system call.
    (g) After the left sub-tree is handled, decrement N, if $N == 0$, there is no need to proceed further, terminate this process using `exit(EXIT_SUCCESS)` system call. Otherwise, proceed further.
    (h) Process creates a new process using `fork()` system call.
    (i) If `fork()` return value is negative, print error message.
    (j) If `fork()` return value is positive, use `wait()` system call to wait for child process to terminate.
    (k) If `fork()` return value is zero, this is a child process and process calls `right_process(N, M)` function to generate the right sub-tree. This will generate recursive function calls until N is equal to 0. Remember to check item number 11.g. Then terminate this process (child process) using `exit(EXIT_SUCCESS)` system call.
    (l) Terminate this process (parent process) using `exit(EXIT_SUCCESS)` system call.
12. Program terminates using `return 0`.

5 Question 2 - Answer

There is a root process, there are $N (N \geq 0)$ right sub-processes. Each right sub-process and the root process have $M (M \geq 0)$ left processes. So the end processes of those left processes are the leaves. They don't have any sub-processes and other processes have a child so they can be designated as a parent process. There are $(M+1) \times (N+1)$ processes ($N+1$ coming from root + N right processes, $M+1$ are coming from M left processes and the parent process) and $N+1$ processes are leaves, so $M \times (N+1)$ processes can be designated as parent processes if $M > 0$. If the root process is not considered as a created process, then there are $M \times (N+1) - 1$ created parent processes.

If $M == 0$ there are no left processes, there are $N+1$ processes in total, and the last right process is the only leaf so there are N parent processes if $M == 0$. If the root process is not considered as a created parent process then there are $N - 1$ created parent processes if $M == 0$.

6 Example Outputs

The output of question 1:

```
fatih@fatih-linux:~/Desktop/OS_HWs/os_hw1$ make q1
gcc q1.c -o q1
fatih@fatih-linux:~/Desktop/OS_HWs/os_hw1$ ./q1
Please provide a n value which is greater than or equal to zero: 3
This is the root process, pid:24829.
This is a left child process, pid:24840, ppid:24829.
---
This is a right child process, n:3, pid:24841, ppid:24829.
This is a left child process, pid:24842, ppid:24841.
---
This is a right child process, n:2, pid:24843, ppid:24841.
This is a left child process, pid:24844, ppid:24843.
---
This is a right child process, n:1, pid:24845, ppid:24843.
This is a left child process, pid:24846, ppid:24845.
fatih@fatih-linux:~/Desktop/OS_HWs/os_hw1$ ./q1
Please provide a n value which is greater than or equal to zero: -1
Please provide a n value which is greater than or equal to zero: 0
This is the root process, pid:24865.
This is a left child process, pid:24884, ppid:24865.
```

The output of question 3:

```
fatih@fatih-linux:~/Desktop/OS_HWs/os_hw1$ make q2
gcc q2.c -o q2
fatih@fatih-linux:~/Desktop/OS_HWs/os_hw1$ ./q2
Please provide a n value which is greater than or equal to zero: 3
Please provide a m value which is greater than or equal to zero: 2
This is the root process, pid:25272.
This is a left child process, m:2, pid:25291, ppid:25272.
This is a left child process, m:1, pid:25292, ppid:25291.
---
This is a right child process, n:3, pid:25293, ppid:25272.
This is a left child process, m:2, pid:25294, ppid:25293.
This is a left child process, m:1, pid:25295, ppid:25294.
---
This is a right child process, n:2, pid:25296, ppid:25293.
This is a left child process, m:2, pid:25297, ppid:25296.
This is a left child process, m:1, pid:25298, ppid:25297.
---
This is a right child process, n:1, pid:25299, ppid:25296.
This is a left child process, m:2, pid:25300, ppid:25299.
This is a left child process, m:1, pid:25301, ppid:25300.
fatih@fatih-linux:~/Desktop/OS_HWs/os_hw1$ ./q2
Please provide a n value which is greater than or equal to zero: -1
Please provide a n value which is greater than or equal to zero: 0
Please provide a m value which is greater than or equal to zero: -1
Please provide a m value which is greater than or equal to zero: 0
This is the root process, pid:25314.
```