Homework 1 Report

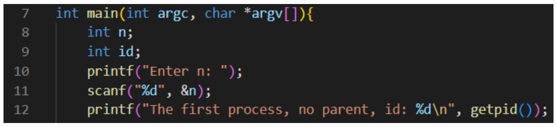
Emre Çamlıca - 150210071,

1 Introduction

In this homework I created the required process hierarchies using the "fork()" and "wait()" system calls and calculated how many processes in the created hierarchies can be identified as parent processes.

2 Question 1

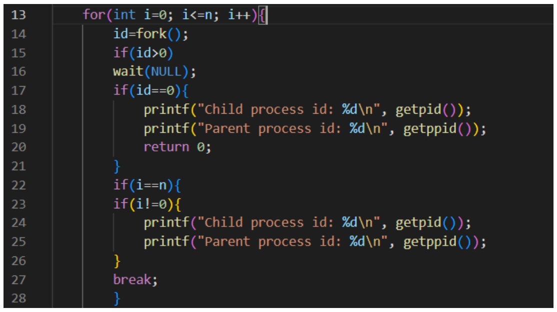
In the solution, I firstly define the variables used to represent the N value and ID's of the processes. Then I ask the user to input N and print the first process beforehand, because it has no parents, as you can see in 1.



```c
int main(int argc, char *argv[]){
    int n;
    int id;
    printf("Enter n: ");
    scanf("%d", &n);
    printf("The first process, no parent, id: %d\n", getpid());
```

Figure 1: Question 1 Initialization

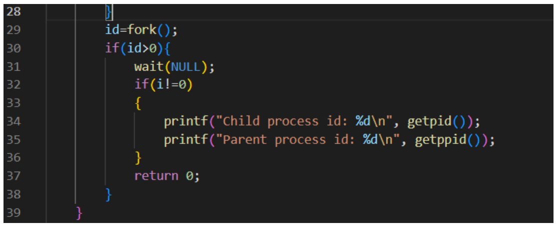
After initializing the variables, I use a for loop to execute the required hierarchy. The for loop iterates from zero to N, including N to make sure that at least one iteration occurs.



```c
    for(int i=0; i<=n; i++){
        id=fork();
        if(id>0)
        wait(NULL);
        if(id==0){
            printf("Child process id: %d\n", getpid());
            printf("Parent process id: %d\n", getppid());
            return 0;
        }
        if(i==n){
        if(i!=0){
            printf("Child process id: %d\n", getpid());
            printf("Parent process id: %d\n", getppid());
        }
        break;
        }
```

Figure 2: Question 1 First Step of the For Loop

2 shows the first step of the for loop. Here, I use the "fork()" system call for the first time to create a child processes to be terminated just after they complete the task of printing, having an id of zero. I used the "wait()" command above for the first time to make sure the parents, which have id's greater than zero, do not terminate before their first, short living children. 3 shows the second step of the for loop. Here I use the "fork()" command for the second time to create the long living child, which will be a parent. Once again I used the "wait()" command the same way to make sure that parent processes do not terminate before their children.



```
        }
        id=fork();
        if(id>0){
            wait(NULL);
            if(i!=0)
            {
                printf("Child process id: %d\n", getpid());
                printf("Parent process id: %d\n", getppid());
            }
            return 0;
        }
    }
```

Figure 3: Question 1 Second Step of the For Loop

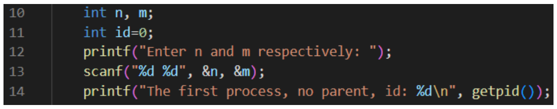
Returning back to 2 the long living children execute the same steps of their parents until the loop reaches the N value. In that case, the long living child no more creates another long living child and terminates, printing its and its parent's id. I put the $(i \neq 0)$ condition to make the code work theoretically for the input $N = 0$, even though we are not required to deal with that case.

2.1 Question 1 Number of Parent Processes

For any N value there are $2(N+1)$ processes and for every parent process, there is exactly one child process which is not a parent. Therefore the processes are divided evenly as those can be identified as parents and those can not. Therefore $2(N + 1)/2 = N + 1$ processes can be identified as parent processes.

3 Question 2

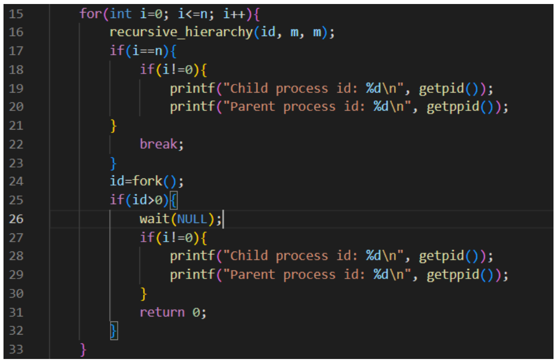
I initialized the question 2 in a similar manner. Except that this time I set $id = 0$ to be sent to my recursive hierarchy function initially as you can observe in 4.



```
int n, m;
int id=0;
printf("Enter n and m respectively: ");
scanf("%d %d", &n, &m);
printf("The first process, no parent, id: %d\n", getpid());
```

Figure 4: Question 2 Initialization

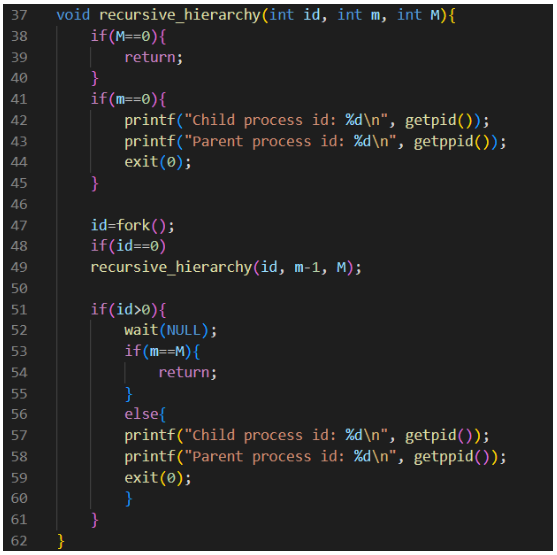
I construct the for loop in a similar manner. This time I use the recursive hierarchy function to create the left sub-trees using the M value for each process in the right sub-tree one by one during every iteration of the for loop. I create the right side of the tree in similar manner with the first question. I use the "fork()" system call for each N value, wait for the child processes to execute and end the for loop in the $(N + 1)$'st iteration. The for loop is shown in 5.



```
for(int i=0; i<=n; i++){
    recursive_hierarchy(id, m, m);
    if(i==n){
        if(i!=0){
            printf("Child process id: %d\n", getpid());
            printf("Parent process id: %d\n", getppid());
    }
        break;
    }
    id=fork();
    if(id>0){
        wait(NULL);
        if(i!=0){
            printf("Child process id: %d\n", getpid());
            printf("Parent process id: %d\n", getppid());
        }
        return 0;
    }
}
```

Figure 5: Question 2 For Loop

To the recursive hierarchy function, shown in 6, I initially send id, which is initially set to zero, and the m value, two times; one to decrement and one to check the end condition. The function first checks for the terminating conditions. When the m value sent to the function is zero, the child process terminates. Even though we only deal with the case $M > 1$, I made the function also work for $M = 0$ value theoretically. After the condition checks, I use the "fork()" call to create a child process, which is to be sent to the same function, with the m value decreased by one. I use the "wait()" command so that the parents wait until their children terminate. The last child sent recursively, having $m = 0$ terminates, triggering the parents' termination until the very first parent sent to the function, which is checked by the condition $(m == M)$. Therefore all but one processes terminate, printing their and their parents' id's.



```c
void recursive_hierarchy(int id, int m, int M){
    if(M==0){
        return;
    }
    if(m==0){
        printf("Child process id: %d\n", getpid());
        printf("Parent process id: %d\n", getppid());
        exit(0);
    }

    id=fork();
    if(id==0)
    recursive_hierarchy(id, m-1, M);

    if(id>0){
        wait(NULL);
        if(m==M){
            return;
        }
        else{
        printf("Child process id: %d\n", getpid());
        printf("Parent process id: %d\n", getppid());
        exit(0);
        }
    }
}
```

Figure 6: The Recursive Hierarchy Function

3.1 Number of Parent Processes

The total number of processes can be calculated as $(M+1)(N+1)$. Considering the fact that $M > 1$, the total number of processes that have no children is equal to $N+1$. Then the total number of parent processes can be found by subtracting this number from the total: $(M+1)(N+1) - N + 1 = M(N+1)$