Name: Reg#

Instructions:

Please complete each task independently without using external assistance, including ChatGPT. This is essential for building a comprehensive understanding of the material and honing your problem-solving skills.

Objective:

The objective of this lab session is to provide practical experience in the creation of processes through the utilization of the fork() system call.

Task 1:

Write a C program that creates a binary tree of processes using fork(). Ensure that each process creates exactly two child processes until a certain depth of 5 is reached. Print the process tree structure.

Output:

Task 2:

Write a C program that creates multiple child processes using fork(). Ensure that some of the child processes become zombies while others don't.

Output:

Task 3:

Write a C program that creates multiple child processes using fork(). The parent process terminates before the child process.

Output:

Task 4:

Create a program that uses fork() to spawn a child process. Both parent and child processes should open the same file. Implement a scenario where one process writes to the file, and the other reads from it simultaneously.

Output:

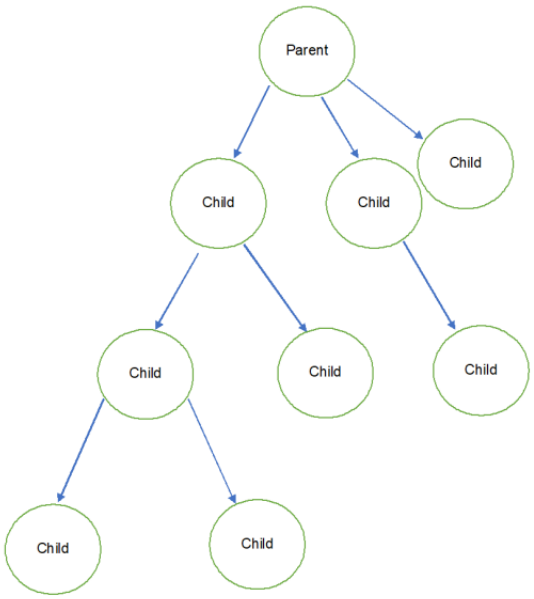
Task 5:

Write a C program in which the parent process employs fork() to create multiple child processes. One of these child processes should utilize the execl() system call to replace its image with a new program. The replacement program is required to take two arguments from the command line, calculate their product, and print the result.

Output:

Task 6:

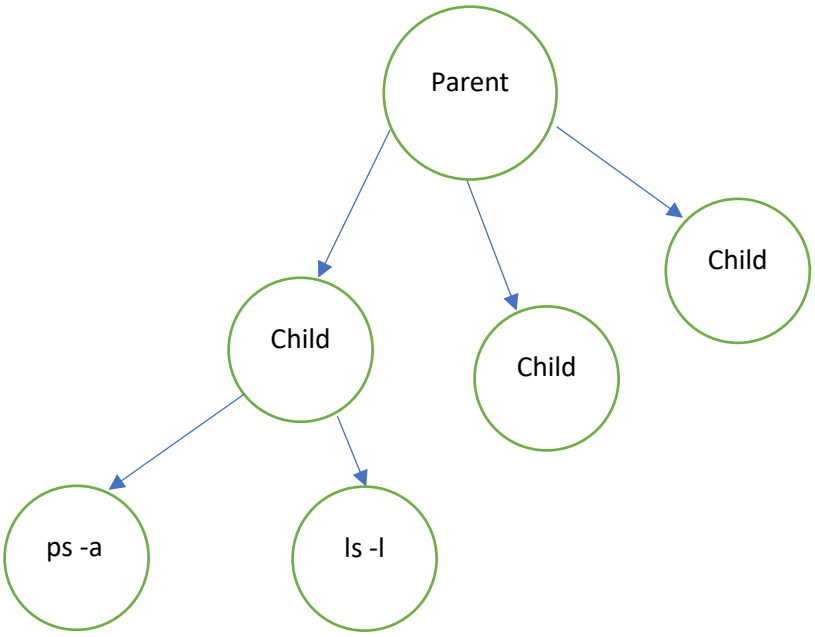
Write a C Program using fork( ) system call to simulate the following scenario:




```
Grandparent process 251
Child process A 255 with parent 251
Child process B 256 with parent 251
Grandchild 1 process 257 with parent 256
Grandchild 2 process 258 with parent 256
Child process C 259 with parent 251
Grandchild 3 process 260 with parent 259
Grandchild 4 process 261 with parent 259
Grandchild 5 process 262 with parent 259
```

Task 7:

Create a C program using the fork() system call to create a parent process and several child processes. The parent process should display its PID, while each child process should display its own PID and the PID of its parent. Demonstrate a structured hierarchy in the output where each child process is clearly identified by its relationship to the parent process, as shown in the sample output. Replace the two grandchild's binaries with binary of ls -l and ps -a respectively.




```
Parent Process    Process ID: 779, Parent Process ID: 778
Child C           Process ID: 783, Parent Process ID: 779
Child B           Process ID: 784, Parent Process ID: 779
Child A           Process ID: 785, Parent Process ID: 779
 Child D          Process ID: 786, Parent Process ID: 785
    PID TTY          TIME CMD
    778 pts/1    00:00:00 tinit
    785 pts/3    00:00:00 a.out
    786 pts/3    00:00:00 ps
 Child E          Process ID: 787, Parent Process ID: 785
total 20
-rwxr-xr-x 1 runner41 runner41 16216 Nov 12 06:52 a.out
-rw-r--r-- 1 runner41 runner41  1237 Nov 12 06:52 main.c
```

Task 8:

Write a C program that implements the following command: ps -aux | grep "bash" > file.txt

Task 9:

Write a C program that uses pipes for communication between parent and child processes. The program receives an array as command-line arguments (in parent process), passes it to the child process, which sorts the array using bubble sort. The child returns the sorted array to the parent, who then prompts the user to input a number and checks its presence in the sorted array. Algorithm for Bubble sort is given below:

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

Task 10:

Write a C program that implements the following command: df -h | awk '{print $2, $5}' > output.txt