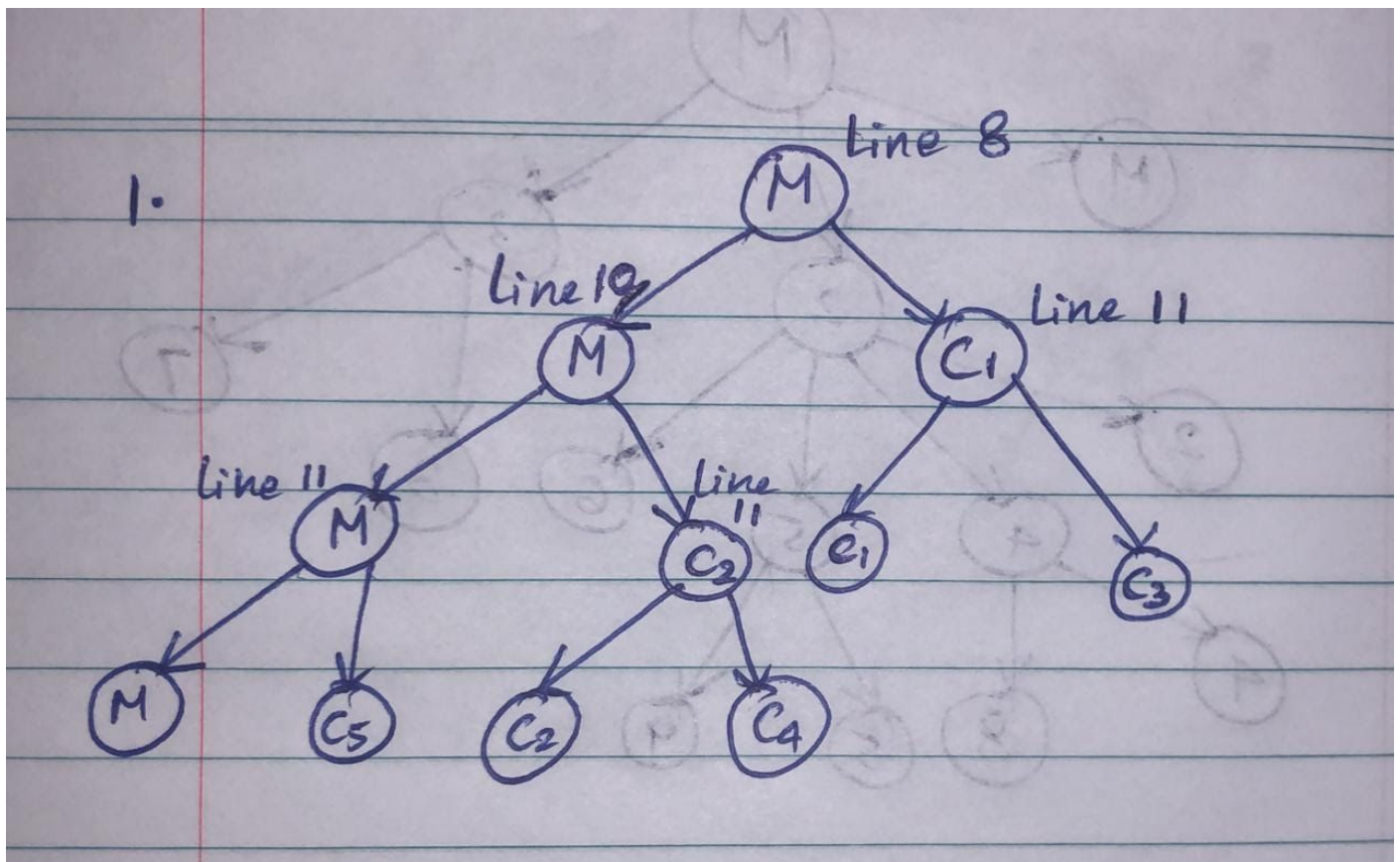


Lab-2 Documentation

CED18I051

Q1.

```
1. #include<stdio.h>
2. #include<sys/types.h>
3. #include<unistd.h>
4.
5. int main()
6. {
7.     pid_t pid;
8.     pid=fork();
9.     if (pid!=0)
10.         fork();
11.     fork();
12.     printf("Count \n");
13.     return 0;
14. }
```



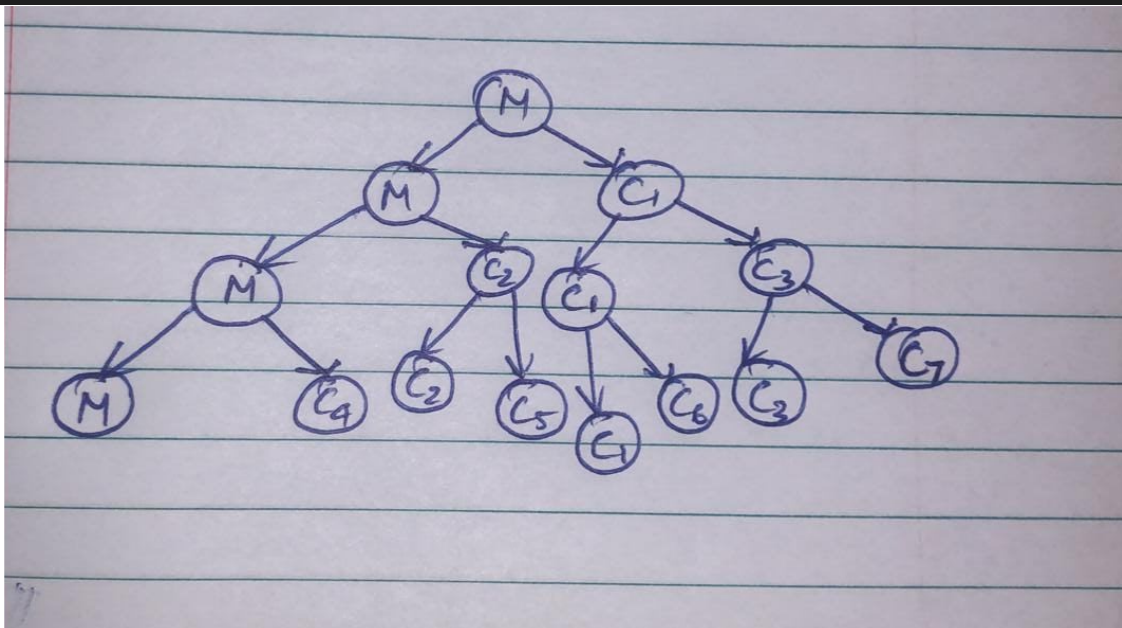
Since the printf statements are not enclosed in any conditions, it will be encountered 6 (number of leafs) times.

Output:

```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./1
Count
Count
Count
Count
Count
Count
Count
subzer0@jarvis:~/Desktop/College/OS/Lab-2$
```

Q2.

```
1. #include<stdio.h>
2. #include<unistd.h>
3.
4. int main()
5. {
6.     printf("OS \n");
7.     fork();
8.     fork();
9.     fork();
10. }
```



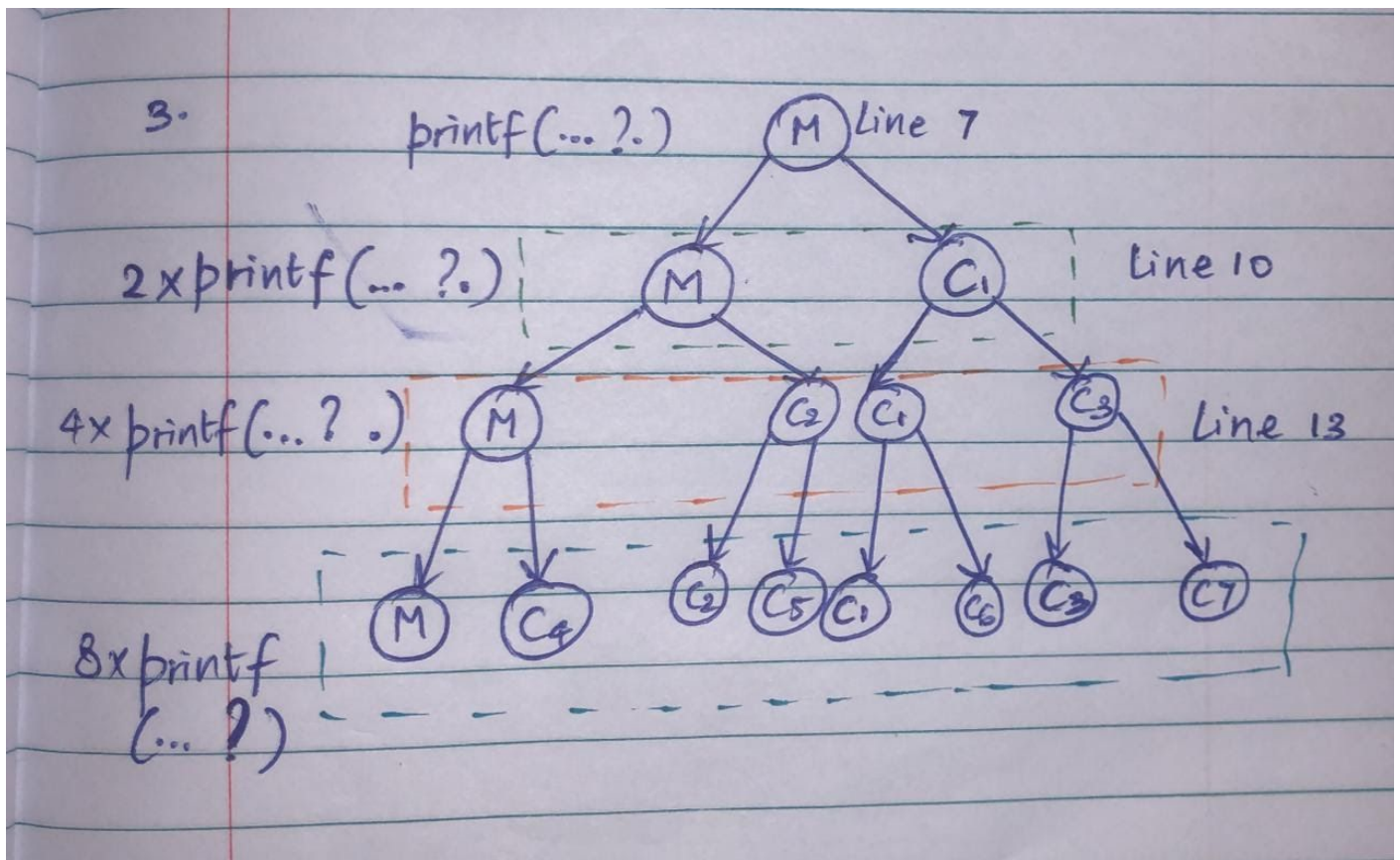
Since the printf statement is before the fork() statements, the printf statement will only be executed on once.

Output:

```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./2
```

Q3.

```
1. #include<stdio.h>
2. #include<unistd.h>
3.
4. int main ()
5. {
6.     printf("This will be printed ?.\n");
7.     fork();
8.
9.     printf("This will be printed ?.\n");
10.    fork();
11.
12.    printf("This will be printed ? .\n");
13.    fork();
14.
15.    printf("This will be printed ?\n");
16.    return 0;
17. }
```



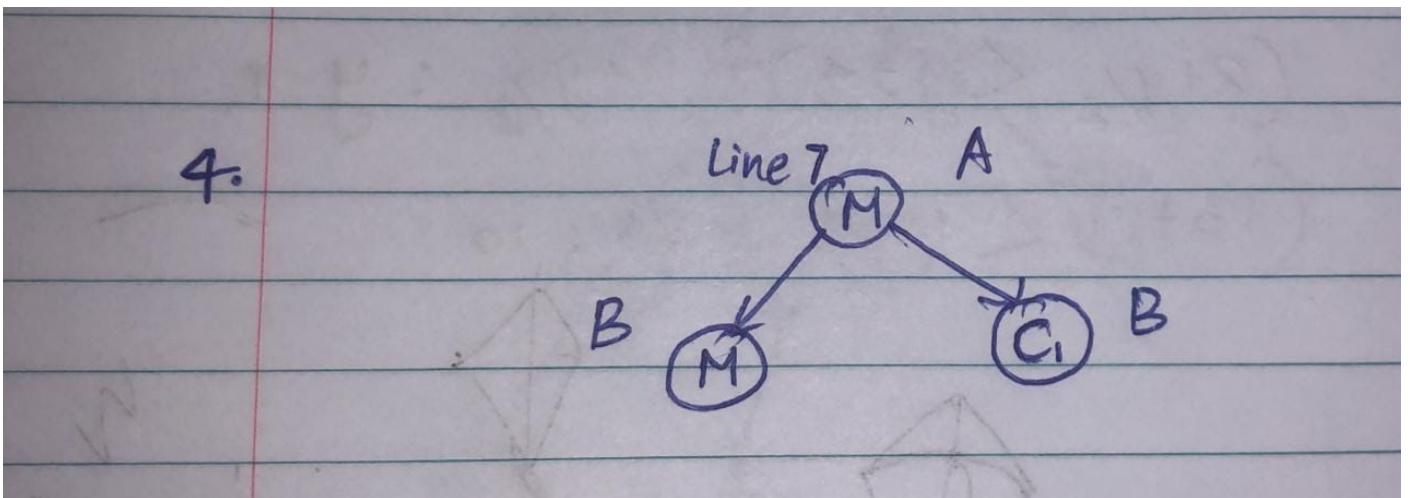
In total there will be $8 + 4 + 2 + 1$ lines of output. The number of each set of lines have been accurately predicted as well. But due to the way processes are allocated, the actual order of each of the statements cannot be predicted.

Output:

```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./3
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ? .
subzer0@jarvis:~/Desktop/College/OS/Lab-2$
```

Q4.

```
1. #include<stdio.h>
2. #include<unistd.h>
3.
4. int main ()
5. {
6.     printf("A \n");
7.     fork();
8.     printf("B\n");
9.     return 0;
10. }
```



Output:

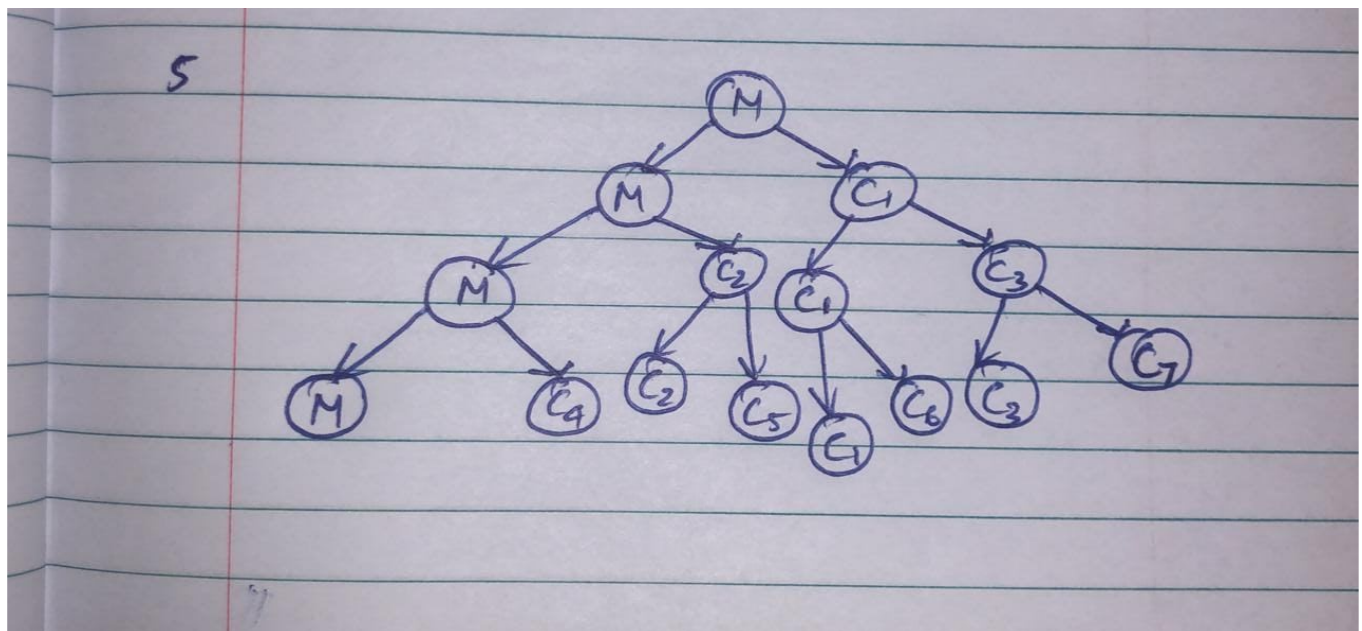
```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./4
A
B
B
subzer0@jarvis:~/Desktop/College/OS/Lab-2$
```

Q5.

```
#include<stdio.h>
#include<unistd.h>

int main(){

    printf("OS ");
    fork();
    fork();
    fork();
}
```



Something to remember is that, “\n” is not just used to get a new line, but also to clear the line buffer. Though the printf statement is placed before any of the forks, what happens is that the line buffer gets copied to all the subsequent images of the program. Hence, all the processes print the line at the end of their execution.

Output:

```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ gcc 5.c -o 5
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./5
OS OS OS OS OS OS OS OS OS subzer0@jarvis:~/Desktop/College/OS/Lab-2$
```

Q6.

```
1. #include<stdio.h>
2. #include<unistd.h>
3.
4. int main () {
5.
6.     printf("A");
7.     fork();
8.     printf("B");
9.     return 0;
10. }
```

As mentioned in the previous, due to the absence of “\n”, the output will be different from what may have been expected.

Output:

```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ gcc 6.c -o 6
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./6
ABABsubzer0@jarvis:~/Desktop/College/OS/Lab-2$
```

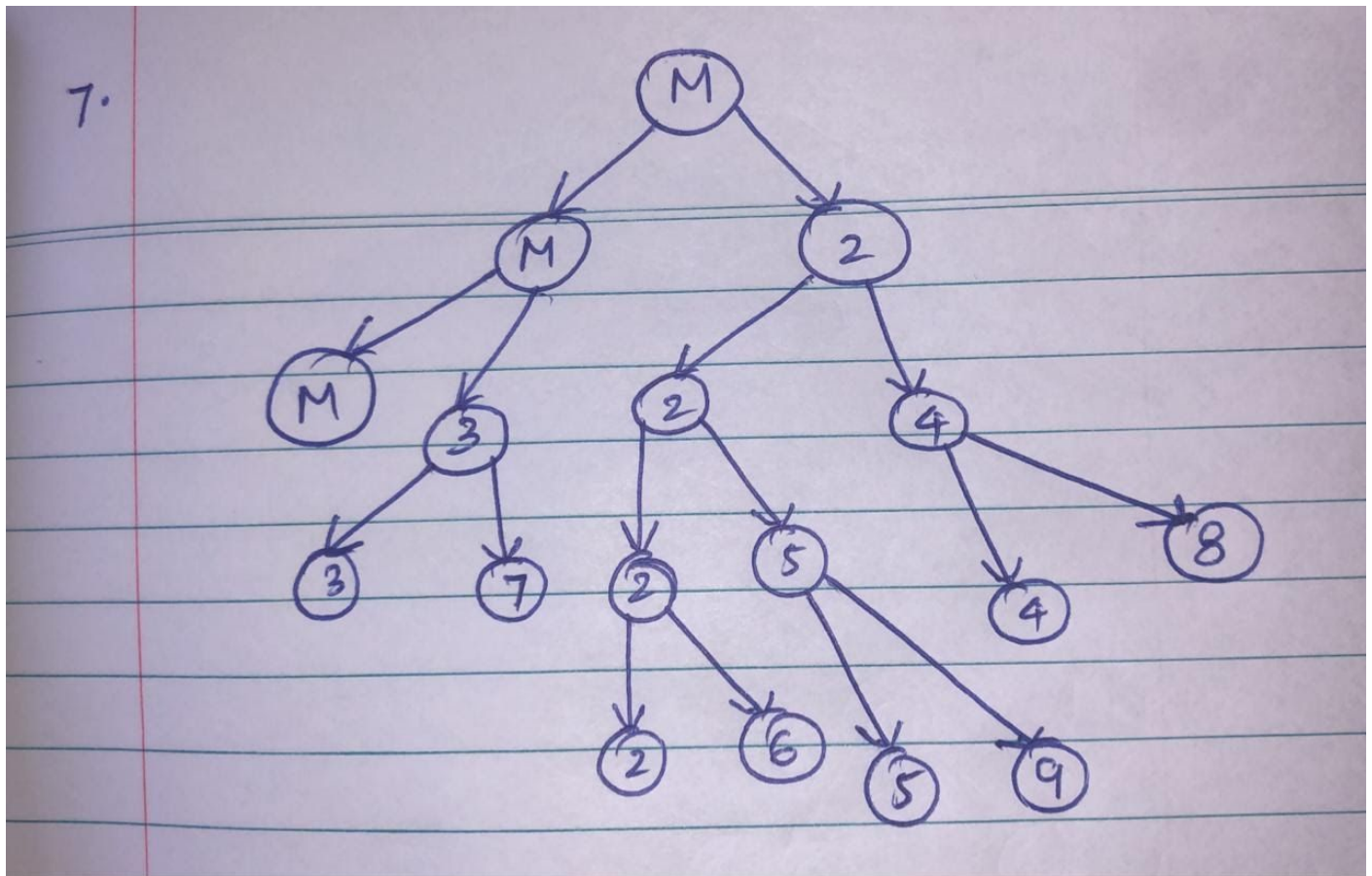
Q7.

Express the following in a process tree setup and also write the C code for the same setup

- 1 forks 2 and 3
- 2 forks 4 5 and 6
- 3 forks 7
- 4 forks 8
- 5 forks 9

I have assumed 1 to be the main.

Tree:



Code:

```

1. #include<stdio.h>
2. #include<unistd.h>
3.
4. int main (){
5.
6.     pid_t _2, _3;
7.
8.     _2 = fork();
9.
10.    if(_2 != 0){
11.
12.        _3 = fork();
13.
14.        if(_3 != 0){           //This block is for 1 (main)
15.            printf("This is 1\n");
16.        }
17.    else{
18.        pid_t _7;
19.        _7 = fork();
20.
21.        if(_7 != 0){           //This block is for 3
22.            printf("This is 3\n");

```

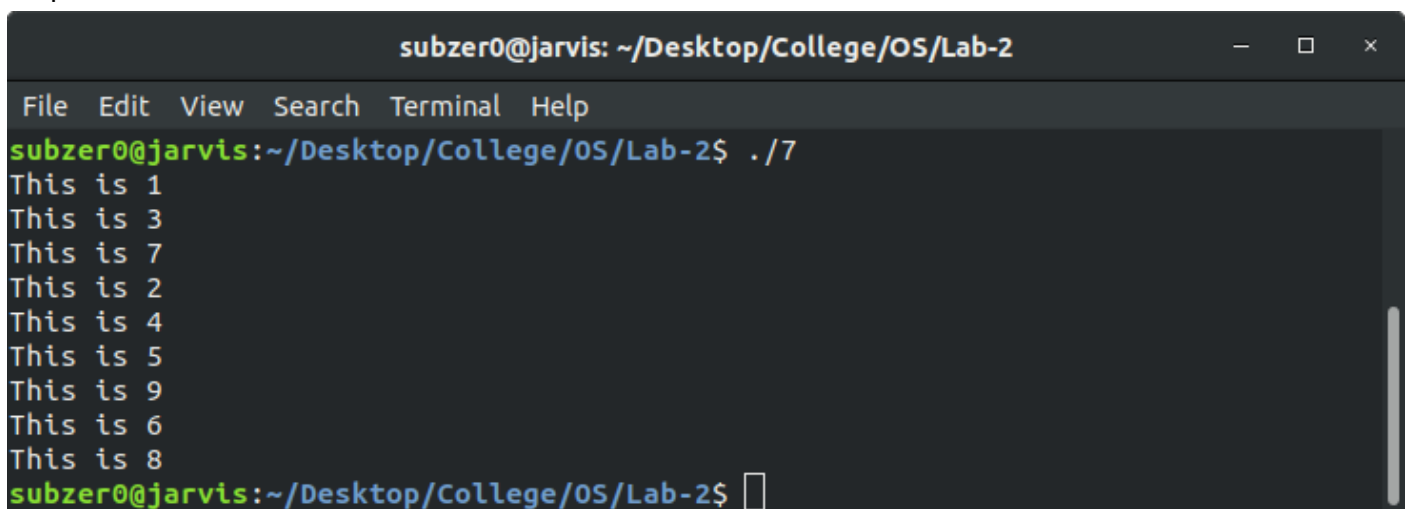
```
23.         }
24.         else{           //This block is for 7
25.             printf("This is 7\n");
26.         }
27.     }
28. }
29. else{
30.     pid_t _4;
31.     _4 = fork();
32.
33.     if(_4 != 0){
34.         pid_t _5;
35.         _5 = fork();
36.
37.         if(_5 != 0){
38.             pid_t _6;
39.             _6 = fork();
40.
41.             if(_6 != 0){           //This block is for 2
42.                 printf("This is 2\n");
43.             }
44.             else{           //This block is for 6
45.                 printf("This is 6\n");
46.             }
47.         }
48.         else{
49.             pid_t _9;
50.             _9 = fork();
51.
52.             if(_9 != 0){           //This block is for 5
53.                 printf("This is 5\n");
54.             }
55.             else{           //This block is for 9
56.                 printf("This is 9\n");
57.             }
58.         }
59.     }
60.     else{
61.
62.         pid_t _8;
63.         _8 = fork();
64.
65.         if(_8 != 0){           //This block is for 4
66.             printf("This is 4\n");
```



```
67.         }
68.         else{           //This block is for 8
69.             printf("This is 8\n");
70.         }
71.     }
72. }
73.
74.     return 0;
75. }
```

What I learnt is that, while building the process tree, it is better to build it as a binary tree rather than building it as a tree. This will help a lot while coding.

Output:



```
subzer0@jarvis: ~/Desktop/College/OS/Lab-2
File Edit View Search Terminal Help
subzer0@jarvis:~/Desktop/College/OS/Lab-2$ ./7
This is 1
This is 3
This is 7
This is 2
This is 4
This is 5
This is 9
This is 6
This is 8
subzer0@jarvis:~/Desktop/College/OS/Lab-2$
```

As it can be observed, each block is run only once as stated by the question.