OPERATING SYSTEMS PRACTICE (ASSIGNMENT 2)

Name – Mridul Harish
Roll number – CED18I034

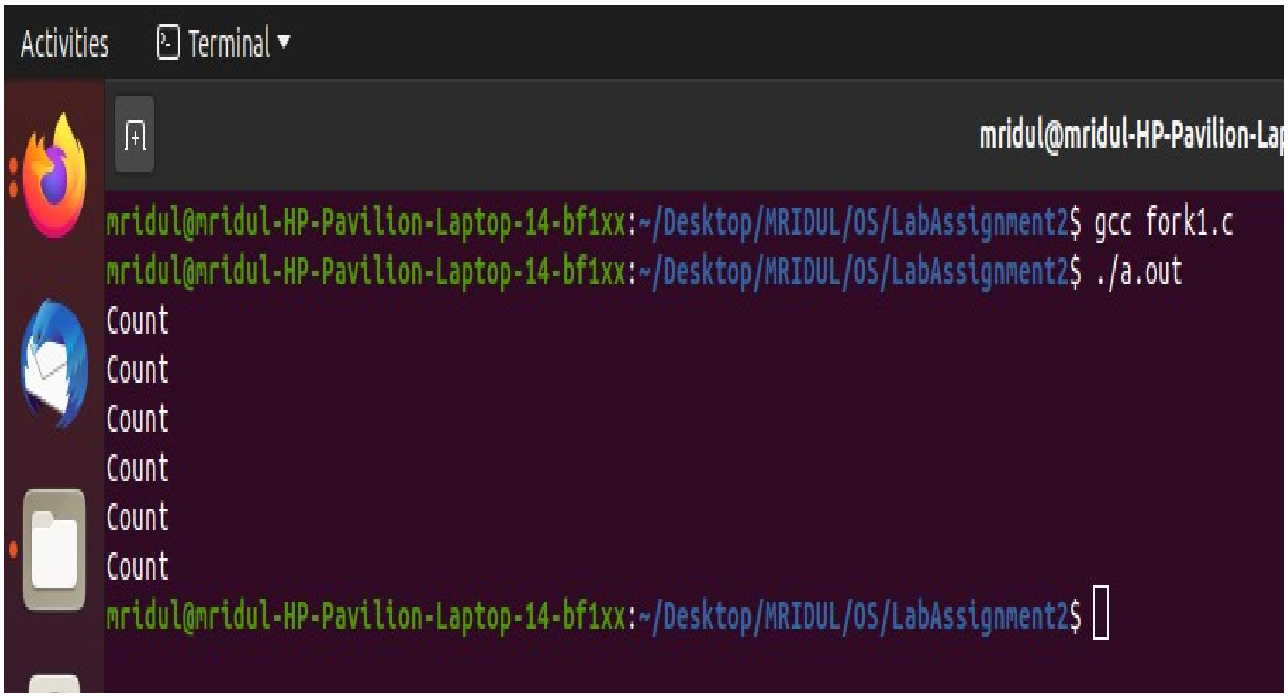
Q1 - Code :-

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
	pid_t pid;
	pid=fork(); //A

	if (pid!=0)
		fork(); //B
	fork(); //C
	printf("Count \n");

	return 0;
}
```

Explanation :-

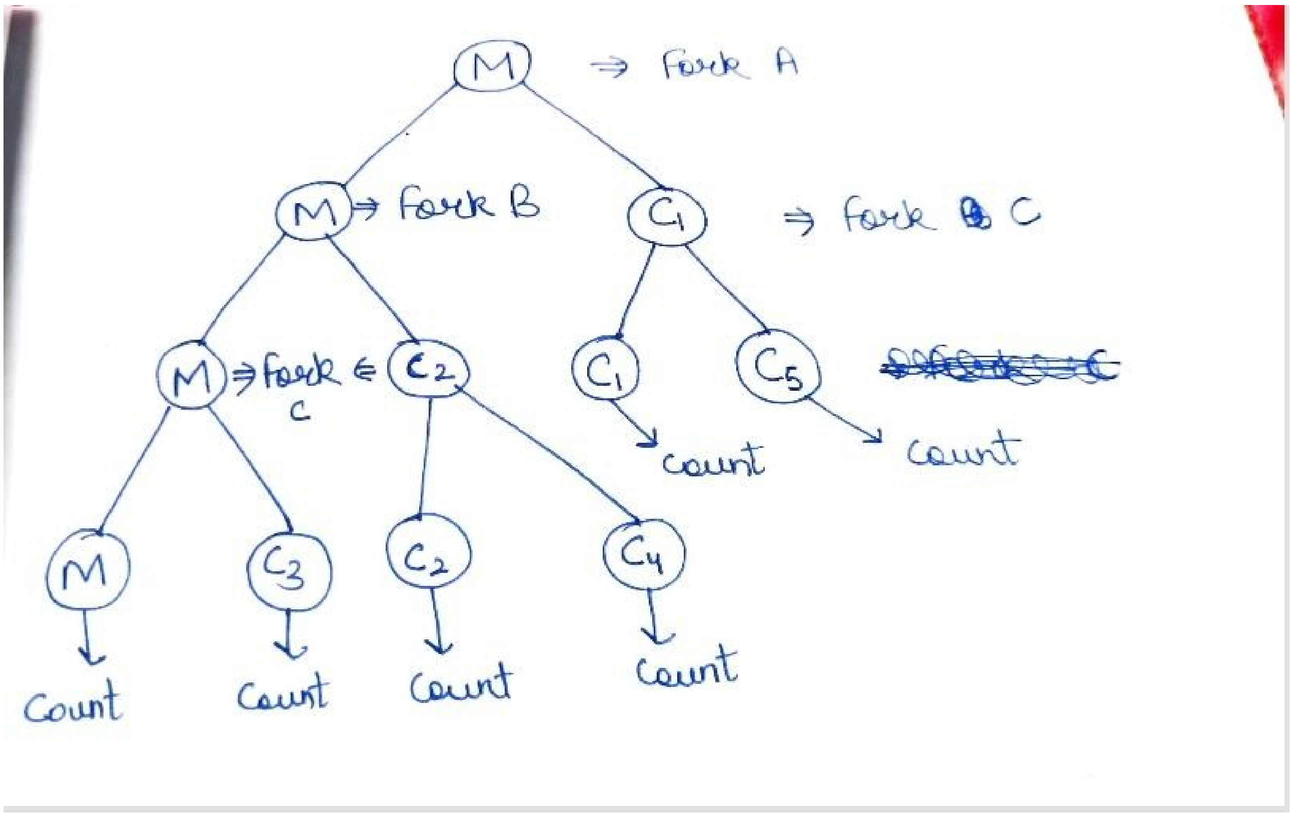
The fork ‘A’ results in 2 processes M and C1. Assuming that the control is transferred to the parent block first, fork ‘B’ is executed as value of pid for parent block is greater than 2 thus the if block is executed resulting in M and C2 followed by fork ‘C’ resulting in M and C3 and C2 and C1. Then the control is transferred to the child block and as pid = 0 for the child image, the if block is not executed and fork ‘C’ is executed resulting in C1 and C5. Then all the processes at the leaf nodes print “Count” i .e. 6 times.

<u>Output</u> :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork1.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
Count
Count
Count
Count
Count
Count
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ 
```

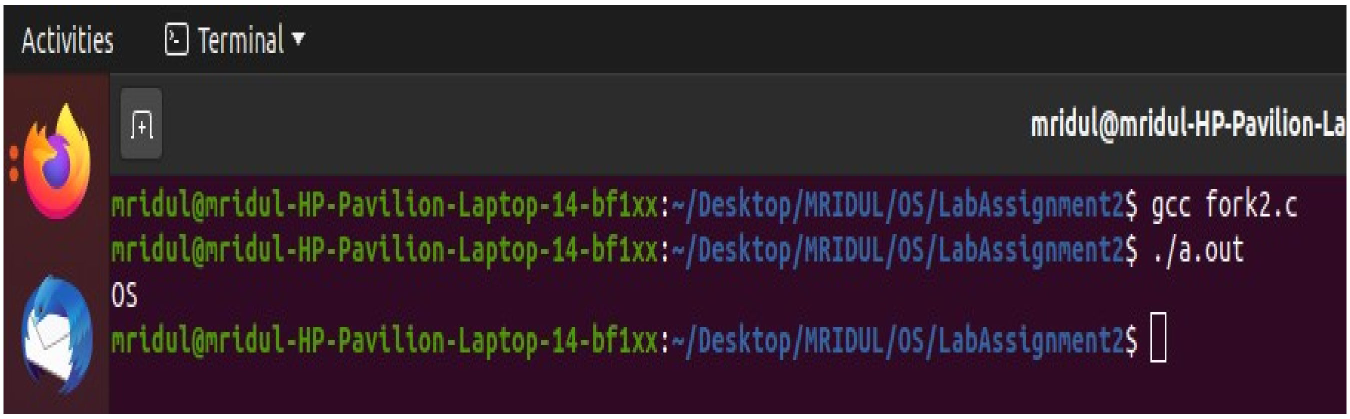
<u>Trace</u> :-



Q2 – Code :-

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
	printf("OS\n");

	fork(); //A
	fork(); //B
	fork(); //C

	return 0;
}
```

Output :-

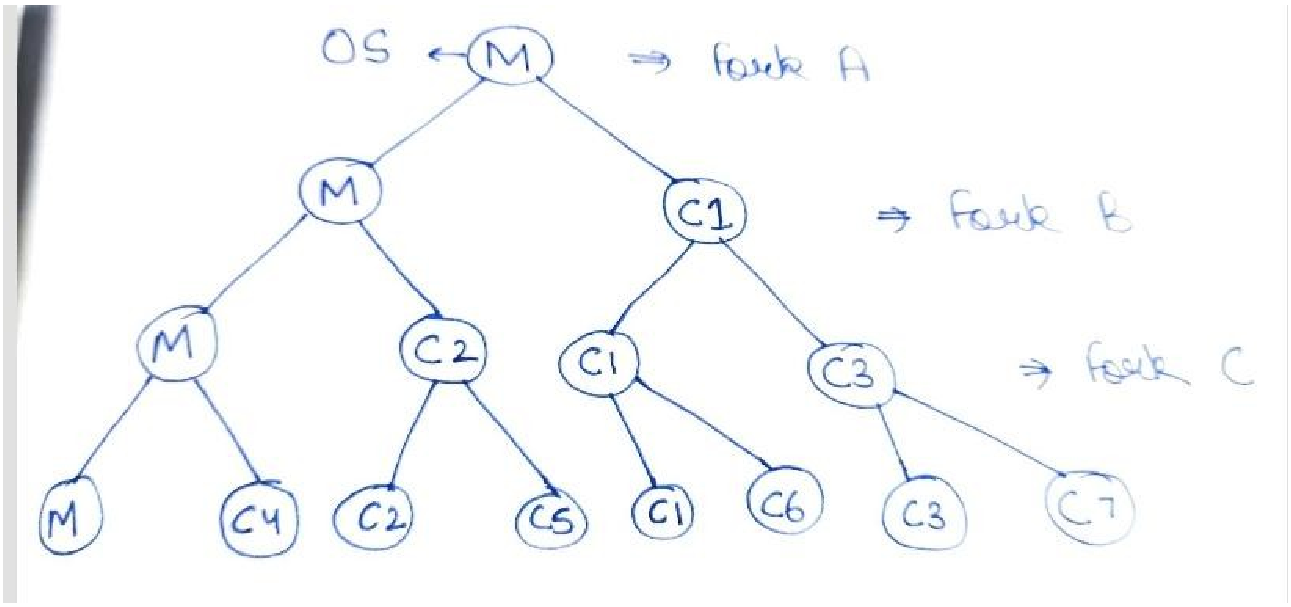


```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork2.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
OS
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$
```

Explanation :-

As soon as the code is run and ./a.out created and executed, printf gets executed and prints “OS”. Then fork ‘A’ is called creating M and C1 followed by fork ‘B’ creating M and C2 and C1 and C3 followed by fork ‘C’ creating M and C4 and C2 and C5 and C1 and C5 and C3 and C7 and thus the program terminates.

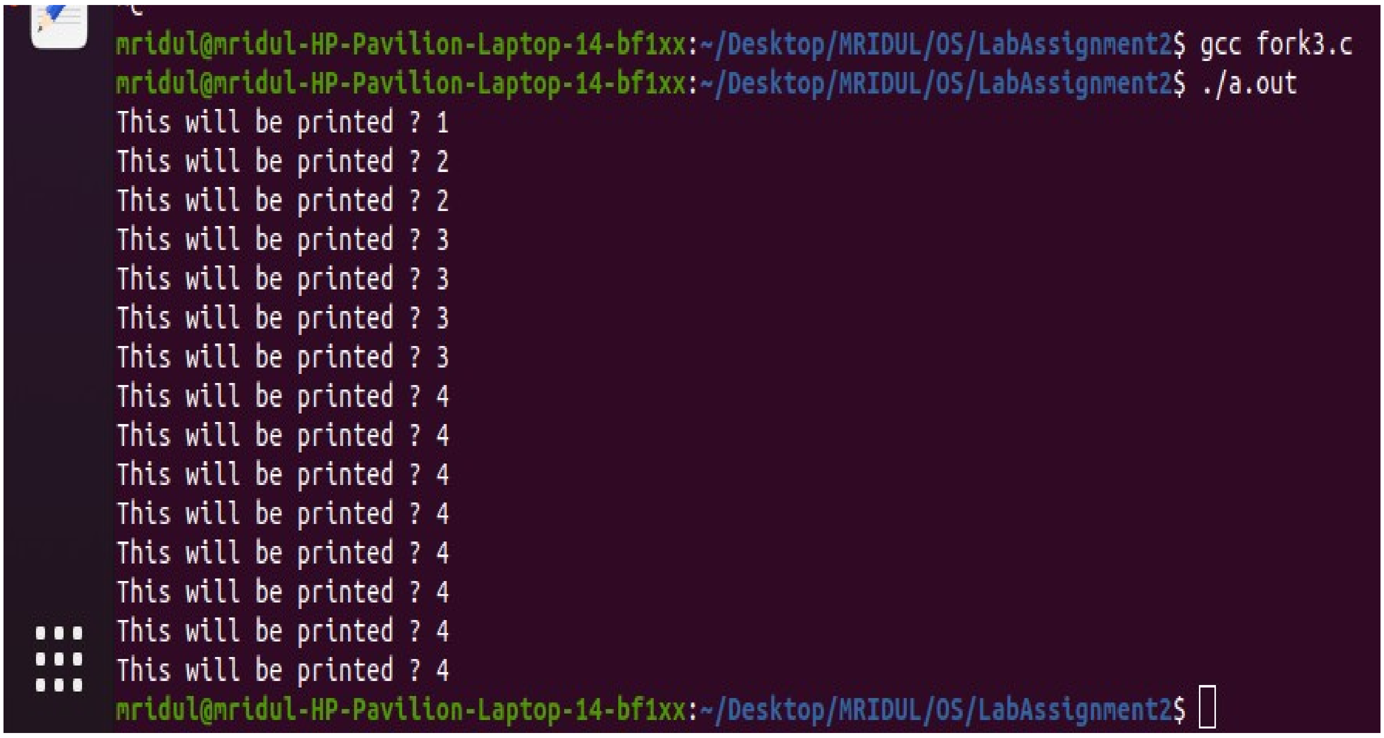
<u>Trace</u> :-



Q3 – <u>Code</u> :-

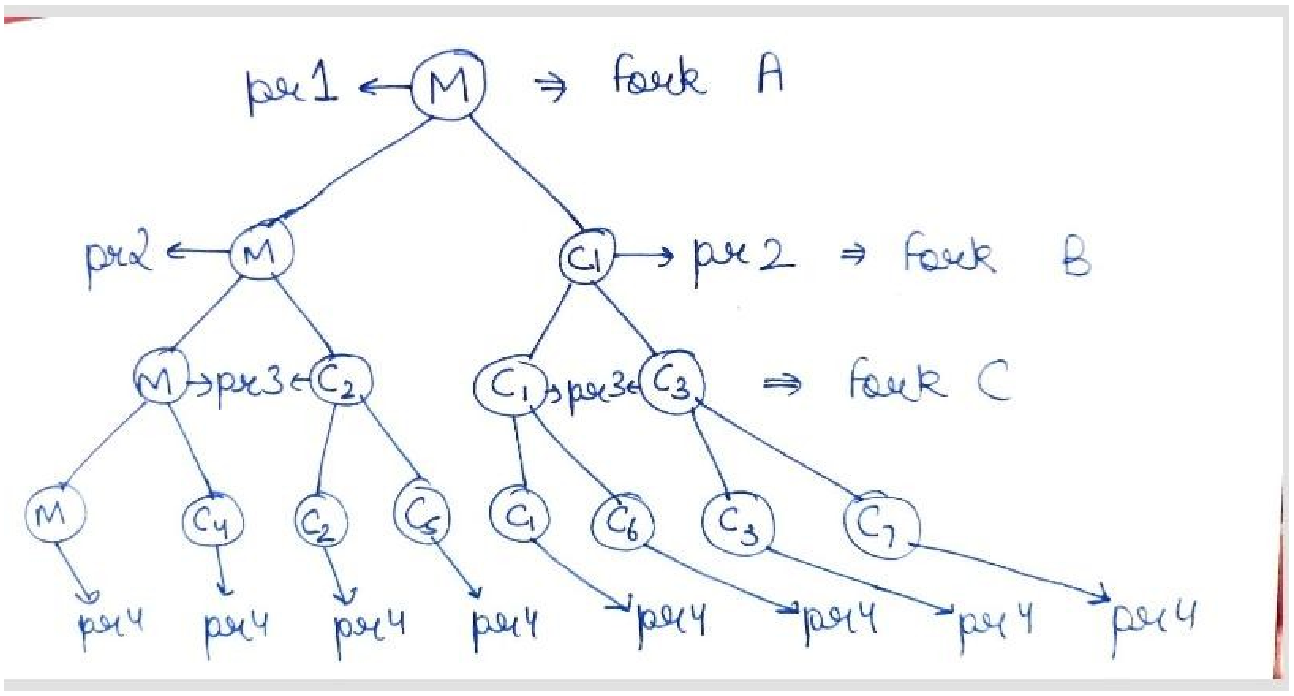
```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main ()
{
        printf("This will be printed ? 1\n"); //pr1
        fork(); //A

        printf("This will be printed ? 2\n"); //pr2
        fork(); //B

        printf("This will be printed ? 3\n"); //pr3
        fork(); //C

        printf("This will be printed ? 4\n"); //pr4

        return 0;
}
```

<u>Output</u> :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork3.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
This will be printed ? 1
This will be printed ? 2
This will be printed ? 2
This will be printed ? 3
This will be printed ? 3
This will be printed ? 3
This will be printed ? 3
This will be printed ? 4
This will be printed ? 4
This will be printed ? 4
This will be printed ? 4
This will be printed ? 4
This will be printed ? 4
This will be printed ? 4
This will be printed ? 4
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$
```

<u>Trace</u> :-



<u>Explanation</u> :-

As soon as the program is run and a.out is executed, printf gets

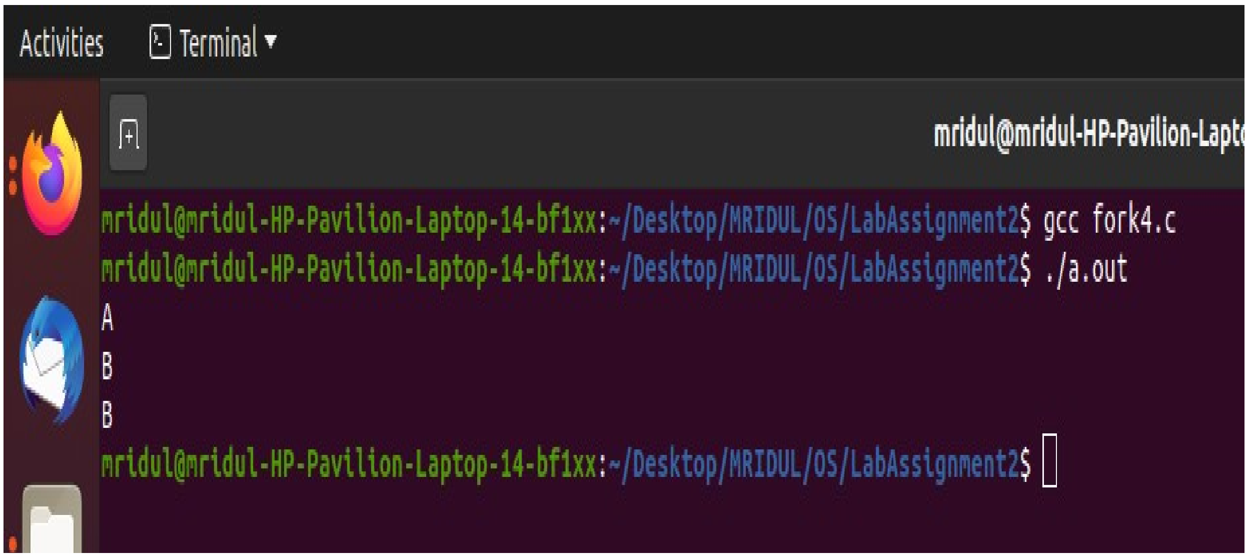
executed printing “pr1”. Then fork ‘A’ get executed creating M and C1 and they print “pr2” followed by fork ‘B’ getting executed creating 4 new processes which all print “pr3” followed by fork ‘C’ getting executed creating 8 new processes which all print “pr4”.

Note – This is the ideal output we are expecting from the binary tree but we do not get this output always, output is random and dependent on the kernel.

Q4 – <u>Code</u> :-

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main ()
{
        printf("A\n");
        fork(); //A

        printf("B\n");

        return 0;
}
```

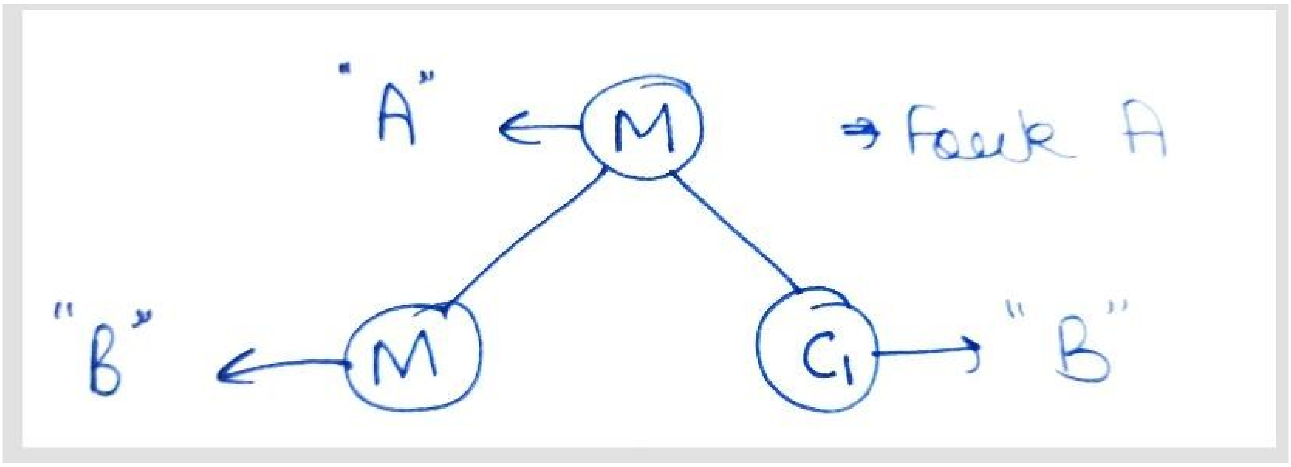
<u>Output</u> :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork4.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
A
B
B
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$
```

}

<u>Trace</u> :-



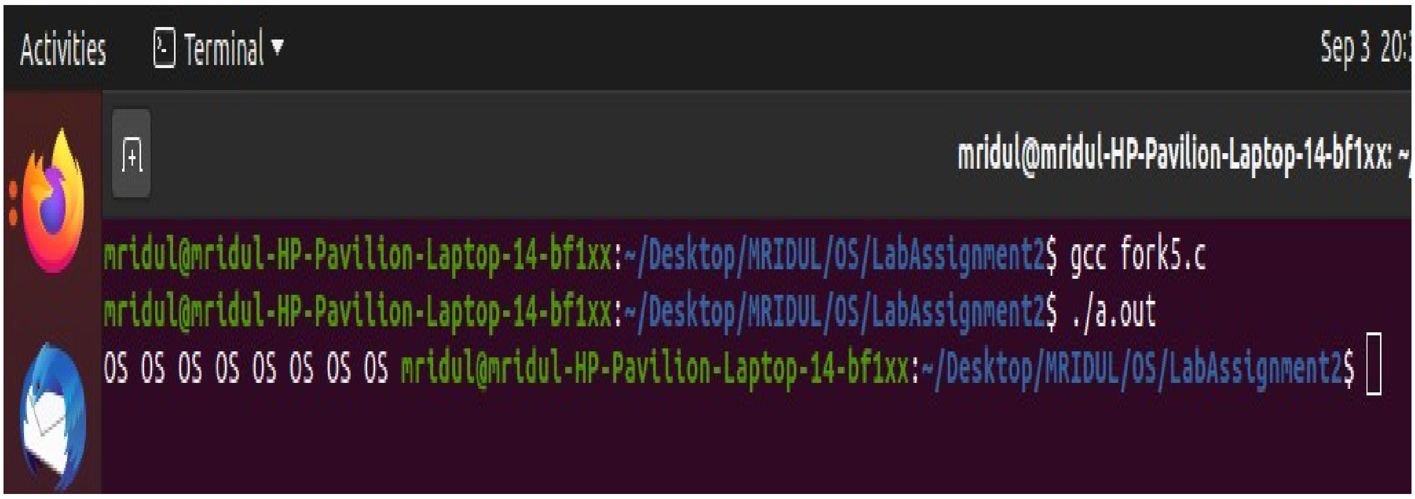
<u>Explanation</u> :-

As soon as the program is run and a.out is created and executed, printf gets executed first printing “A”. Then fork ‘A’ is executed creating M and C1 and as there is no further forking, both the processes print “B”.

Q5 – <u>Code</u> :-

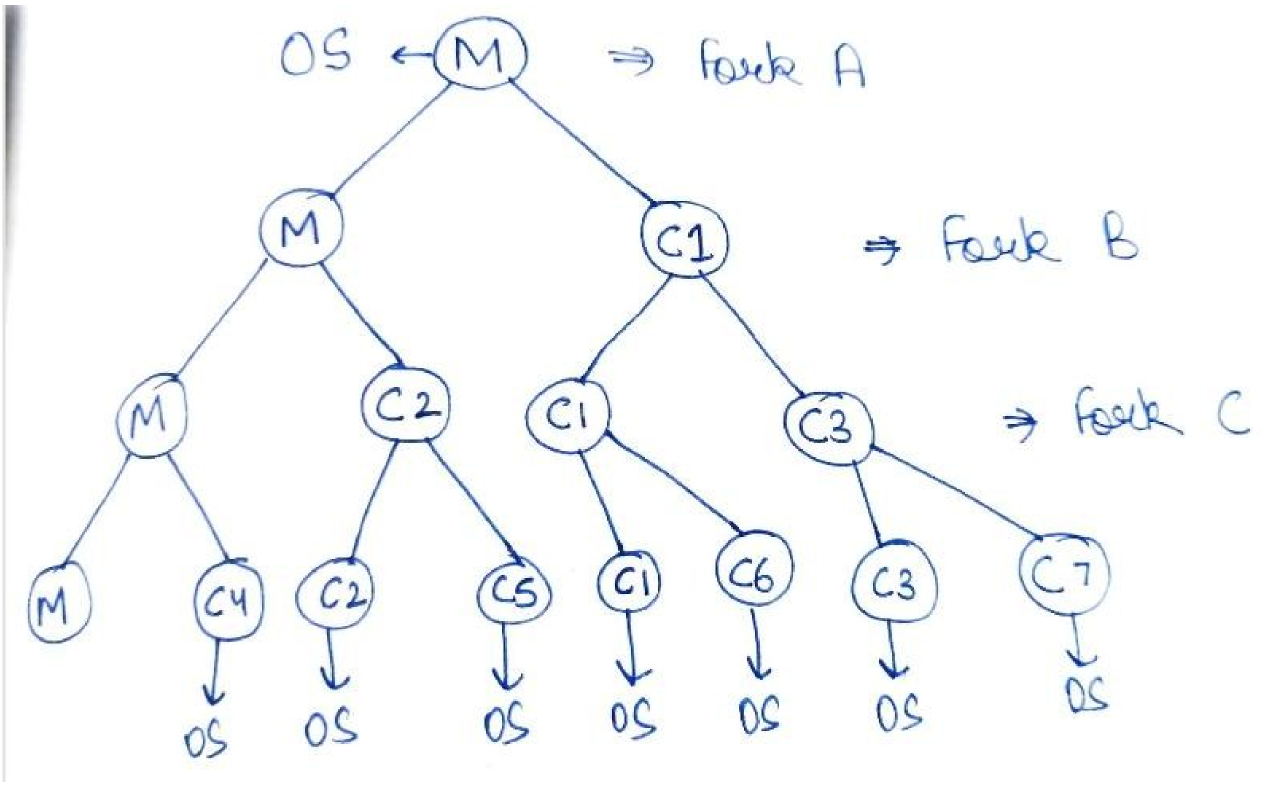
```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
        printf("OS ");

        fork(); //A
        fork(); //B
        fork(); //C

        return 0;
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork5.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
OS OS OS OS OS OS OS OS mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$
```

Trace :-



Explanation :-

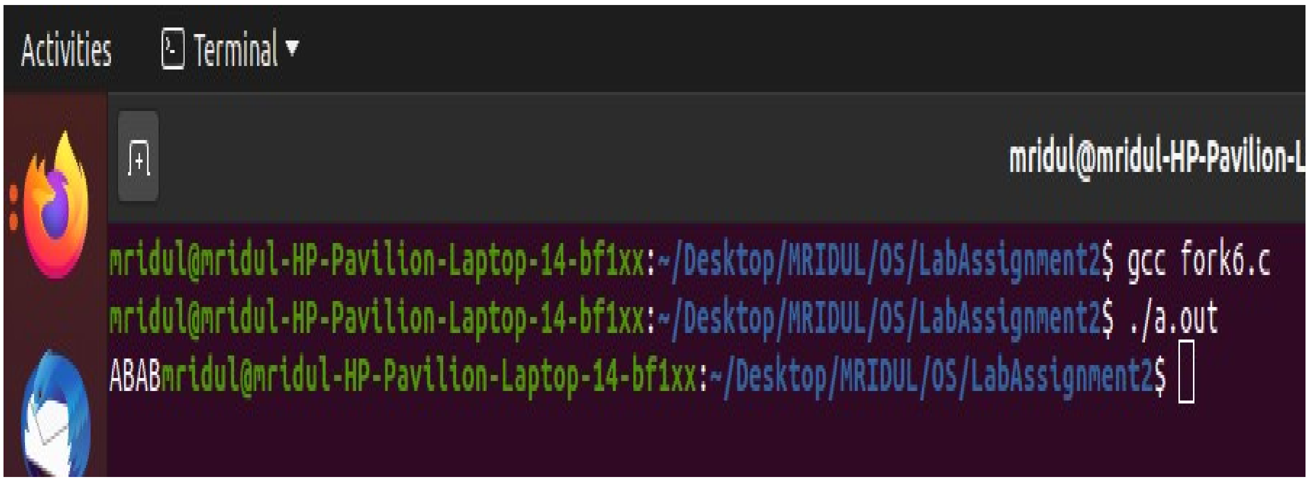
The trace binary tree of the above program is similar to the one in Q2. But the only difference is that "OS" is getting printed 8 times instead of just 1 time as in Q2. This is because the removal of the "\n" from the printf statement. When outputting to standard output using the C library's `printf()` function, the output is usually buffered. The

buffer is not flushed until you output a newline, call `fflush(stdout)` or exit the program. In the above question when we fork the processes, the child processes inherits every part of the parent process, including the unflushed output buffer. This effectively copies the unflushed buffer to each child process and "OS" is printed 7 more times.

Q6 – <u>Code</u> :-

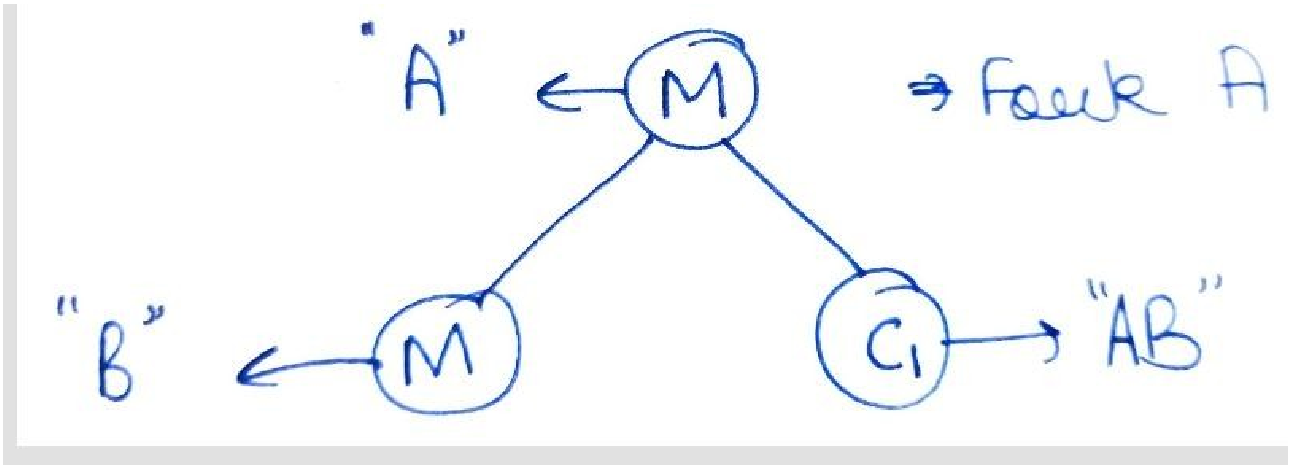
```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
        printf("A");
        fork(); //A

        printf("B");

        return 0;
}
```

<u>Output</u> :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork6.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
ABABmridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ 
```

Trace :-



Explanation :-

The trace binary tree of the above program is similar to the one in Q4. But the only difference is that "ABAB" is getting printed instead of "ABB" as in Q4. This is because the removal of the "\n" from the printf statement. When outputting to standard output using the C library's `printf()` function, the output is usually buffered. The buffer is not flushed until you output a newline, call `fflush(stdout)` or exit the program. Since both processes are identical, they now both have an output buffer that contains A. The execution continues and prints B into the buffers in each process. Both processes then exit, causing a flush of the output buffer, which prints AB twice, once for each process.

Q7 – 1 forks 2, 3

2 forks 4, 5 and 6

3 forks 7

4 forks 8

5 forks 9

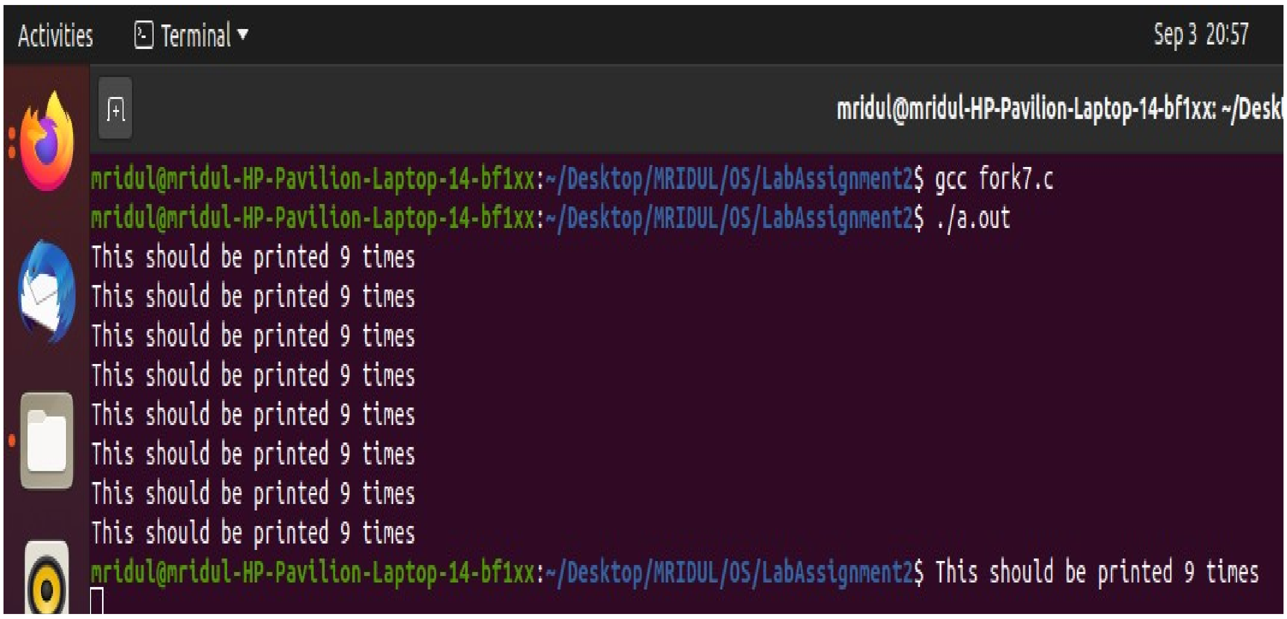
<u>Code</u> :-

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
        int pid1; int pid2; int pid3;

        pid1 = fork(); //creates 1 and 2
        if(pid1 < 0)
        {
                fprintf(stderr, "Failed fork call\n");
        }
        else if(pid1 == 0)
        {
                pid2 = fork(); //creates 2 and 4
                if(pid2 < 0)
                {
                        fprintf(stderr, "Failed fork call\n");
                }
                else if(pid2 == 0)
                {
                        fork(); //creates 4 and 8
                }
                else if(pid2 > 0)
                {
                        fork(); //creates 2 and 5
                        fork(); //creates 2 and 6, creates 5 and 9
                }
        }
        else if(pid1 > 0)
        {
                pid3 = fork(); //creates 1 and 3
                if(pid3 < 0)
                {
                        fprintf(stderr, "Failed fork call\n");
                }
                else if(pid3 == 0)
                {
                        fork(); //creates 3 and 7
                }
        }
printf("This should be printed 9 times\n");
return 0;
}
```

Explanation :-

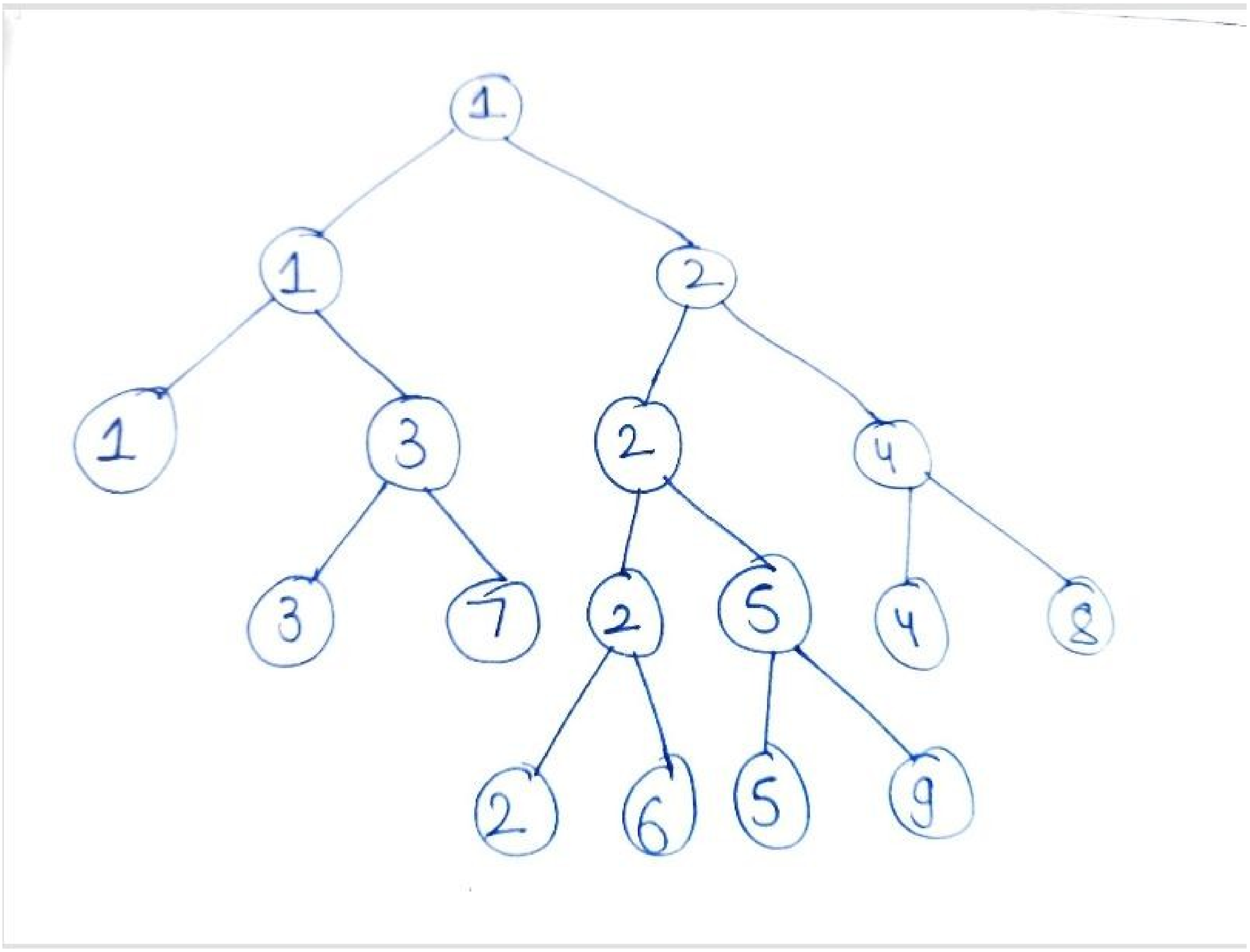
The binary tree is the given trace for the statements given above. As there are 9 leaf nodes, a printf statement at the the end of the code should be printed 9 times as every process will end with that printf only. So as to cross check whether the code written is correct or not, we put a printf statement at the end and it did print 9 times so the give n code is corresponding for the above statements and binary tree.

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ gcc fork7.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ ./a.out
This should be printed 9 times
This should be printed 9 times
This should be printed 9 times
This should be printed 9 times
This should be printed 9 times
This should be printed 9 times
This should be printed 9 times
This should be printed 9 times
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment2$ This should be printed 9 times
```

Trace :-



1
1
2
1
3
2
4
3
7
2
5
4
8
2
6
5
9