OPERATING SYSTEMS PRACTICE (COM301P)

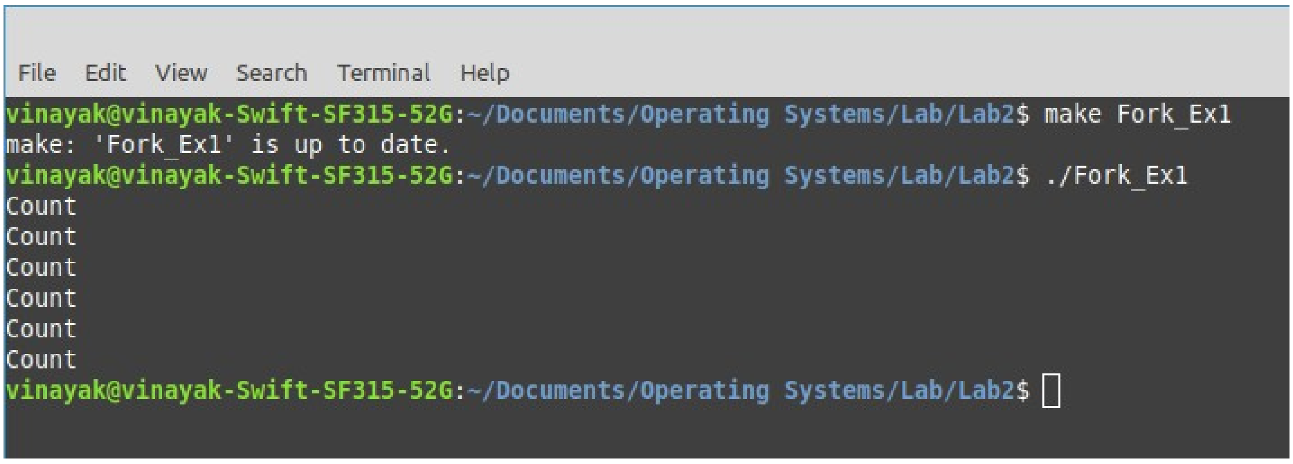
Name: Vinayak Sethi **Roll No:** COE18B061

Forking Assignment

Program 1:

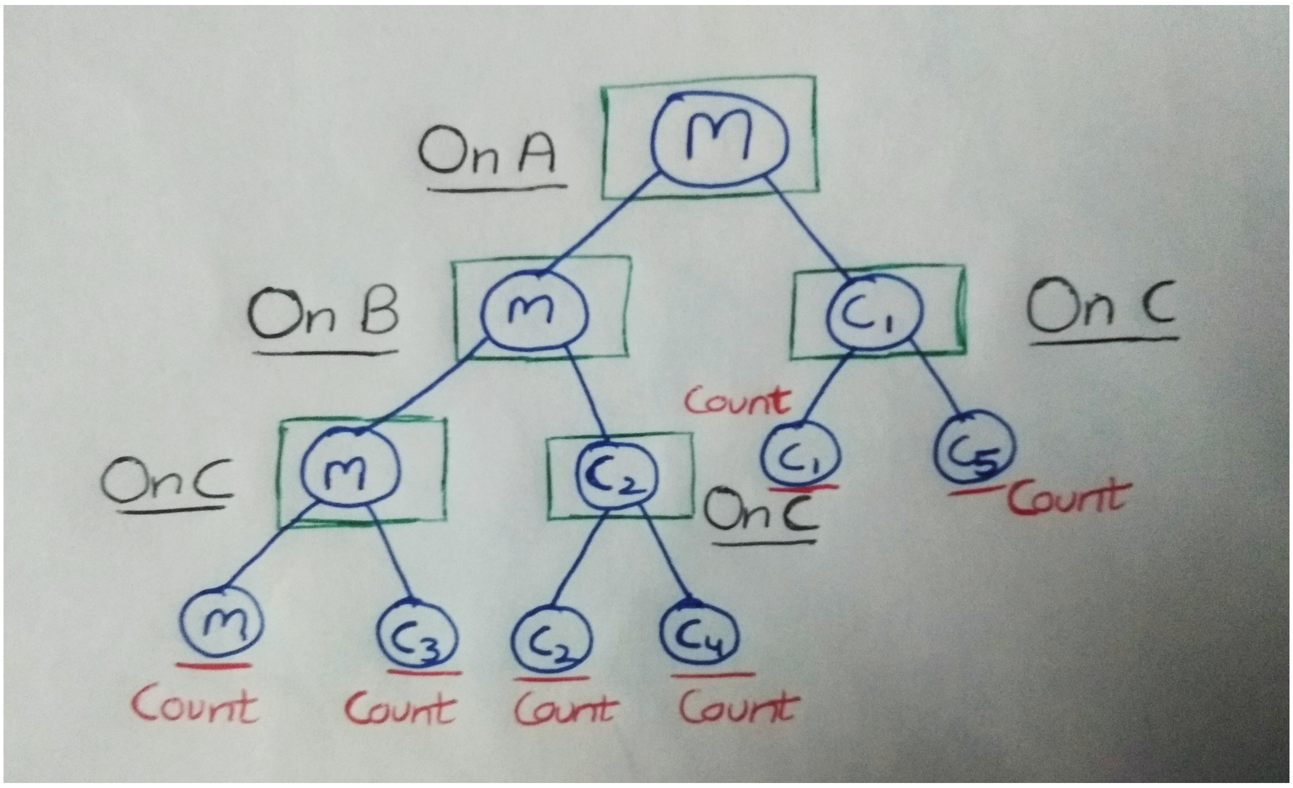
```
//Forking Example 1
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
        pid_t pid;
        pid = fork(); //A
        if (pid != 0)
        fork(); //B
        fork(); //C
        printf("Count \n");
        return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex1
make: 'Fork_Ex1' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex1
Count
Count
Count
Count
Count
Count
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$
```

Process Tree:



Reasoning:

In the process tree **M** is main, **C1** is child-1, **C2** is child-2, **C3** is child-3, and so on.
As we know on forking we get 2 processes parent and child process and on **successful fork** return value we get is **0**, so pid for child process is 0.
So for parent block (pid != 0) we have fork point which creates 2 new processes and then on final fork which is part of both parent and child process, we get 2 new process for each leaf point of B.

So, in total from point B, there are 2 forks() for parent and 1 fork() for child.

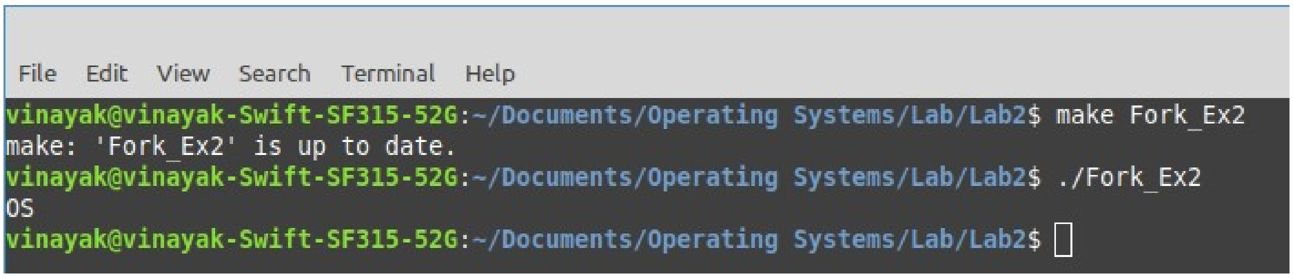
Hence in total parent breaks into 2^(2) +1 = 4 processes and child breaks into 2^(1) + 1 = 2 processes.

Hence total no of times we get **Count** printed is 4 + 2 = **6 times**.

Program 2:

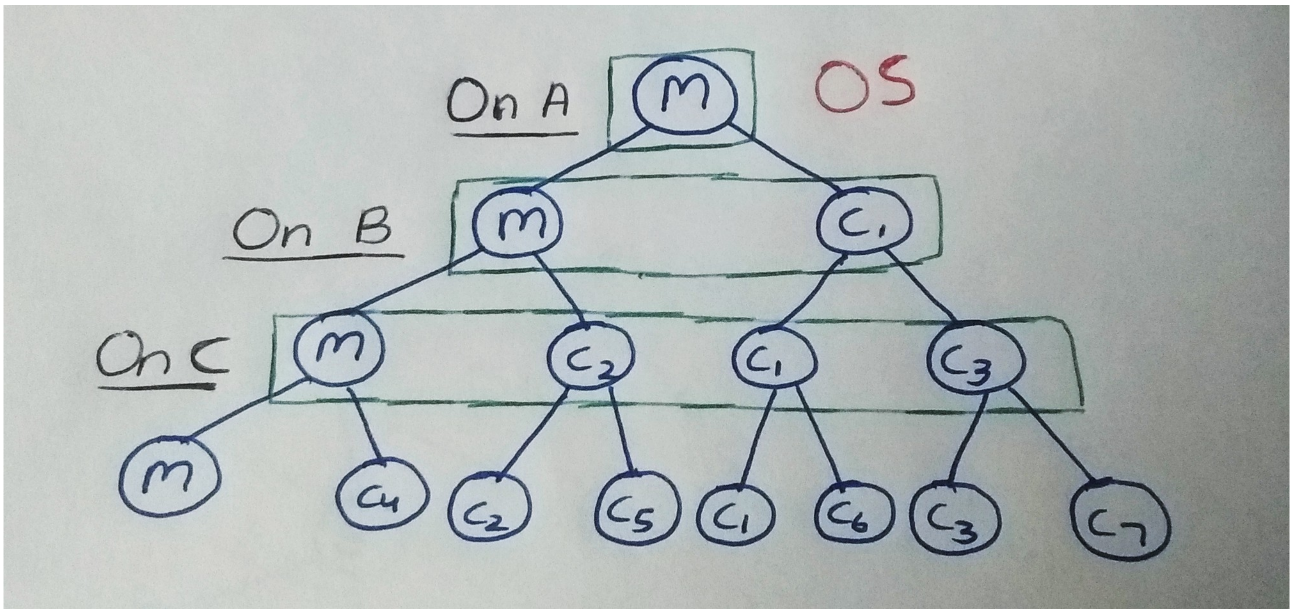
```
//Forking Example 2
#include<stdio.h>
#include<unistd.h>

int main()
{
        printf("OS \n");
        fork(); //A
        fork(); //B
        fork(); //C
        return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex2
make: 'Fork_Ex2' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex2
OS
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$
```

Process Tree:



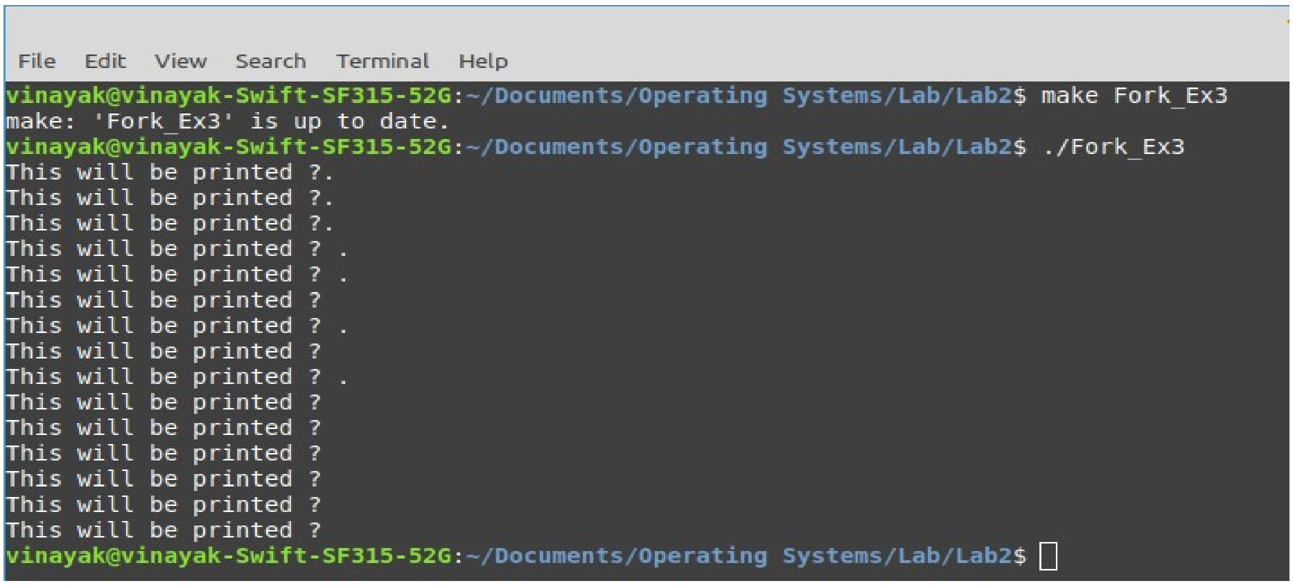
Reasoning:

In the process tree **M** is main, **C1** is child-1, **C2** is child-2, **C3** is child-3, and so on.
Since “OS\n” is before the forking point so it will be printed only once, but we have 3 fork() calls.
So, in total we get 2^(3) = **8** processes where **1** is the **parent** process and other(C1, C2, ..., C7) **7** processes are **child** processes.

Program 3:

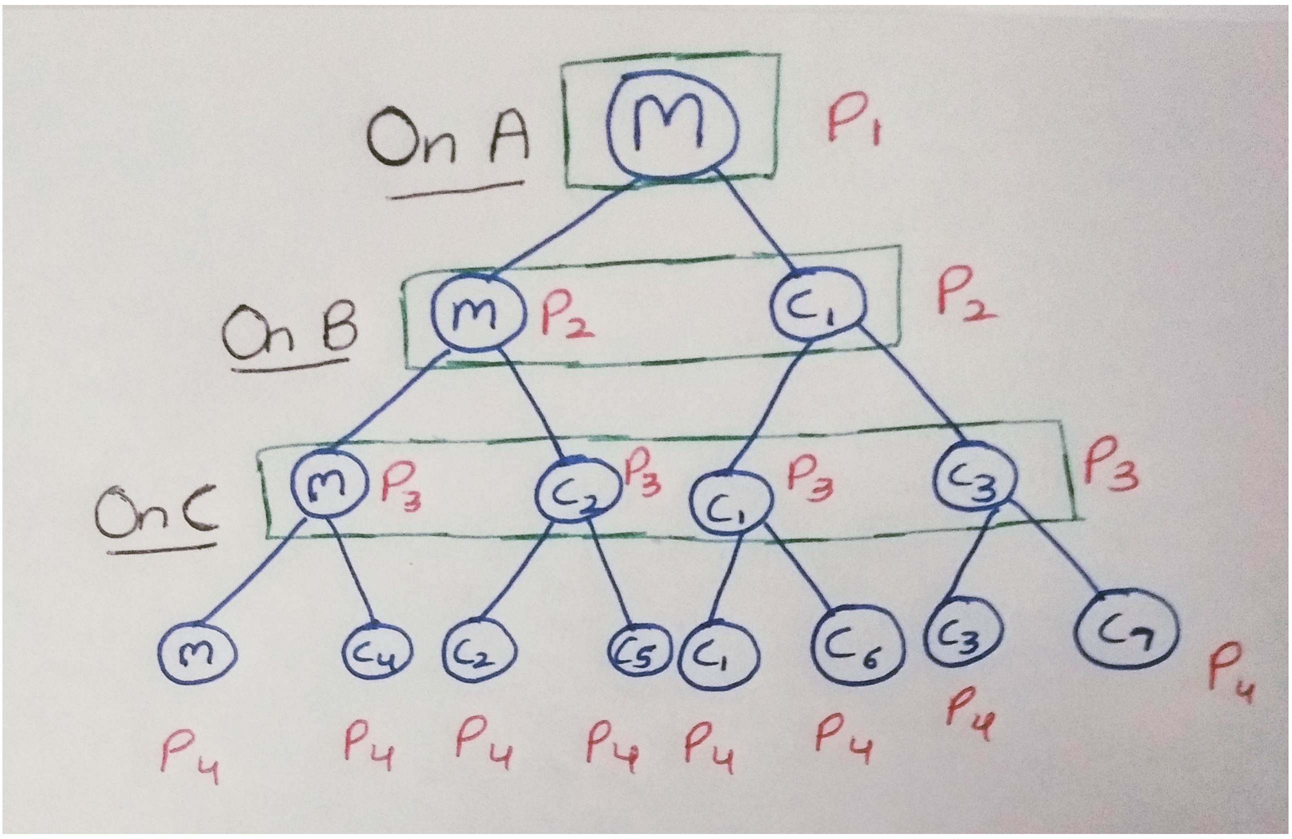
```
//Forking Example 3
#include<stdio.h>
#include<unistd.h>

int main()
{
        printf("This will be printed ?.\n"); //P1
        fork(); //A
        printf("This will be printed ?.\n"); //P2
        fork(); //B
        printf("This will be printed ? .\n"); //P3
        fork(); //C
        printf("This will be printed ?\n"); //P4
        return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex3
make: 'Fork_Ex3' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex3
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 ?
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ 
```

Process Tree:



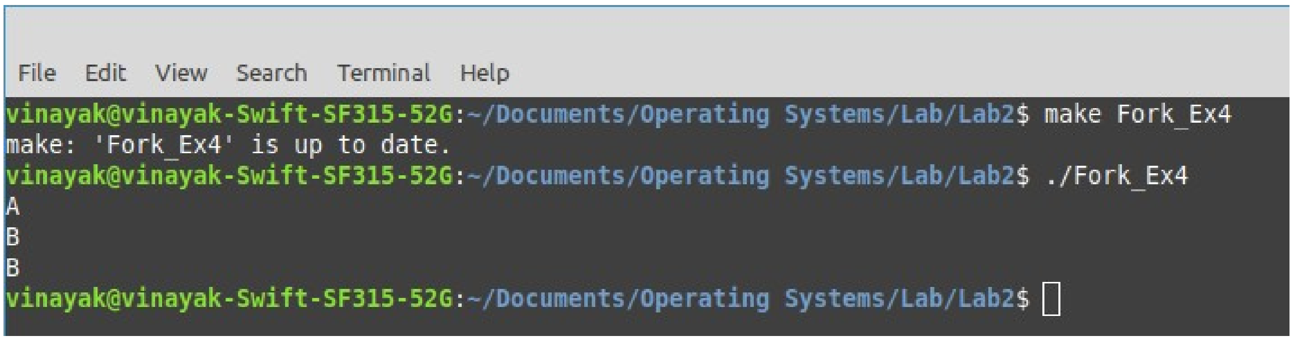
Reasoning:

In the process tree **M** is main, **C1** is child-1, **C2** is child-2, **C3** is child-3, and so on.
Here **P1, P2, P3, P4** represents 1st printf statement, 2nd printf statement ... 4th printf statement respectively.
As there are 3 fork() calls in the program so total we have 2^(3) = 8 processes at the last level.
And each fork() point is coupled with a printf statement so whenever a new process is created, the message will be dispalyed. So in total 2^(3) – 1 = 7 times the printf statement is printed before the last fork point, which is equal to non-leaf nodes in the process tree and after 3rd fork there is printf statement which means last printf **(P4)** will be printed 8 times.
So in total 7 + 8 = **15** times printf is executed, but the order of execution is totally dependent on the kernel.

Program 4:

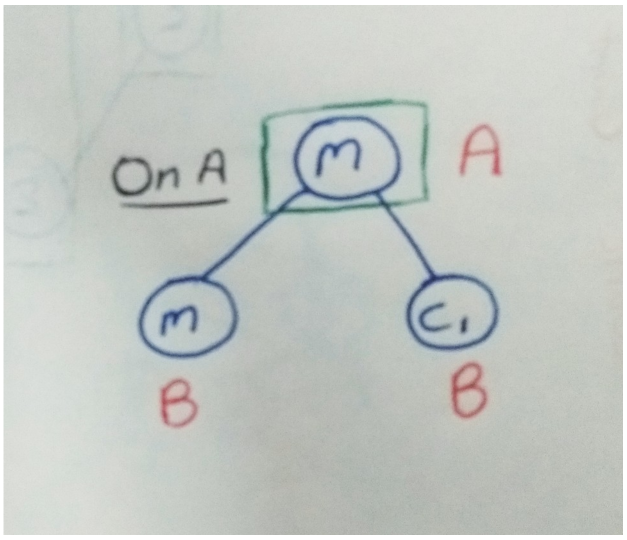
```
//Forking Example 4
#include<stdio.h>
#include<unistd.h>

int main()
{
        printf("A \n");
        fork(); //A
        printf("B\n");
        return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex4
make: 'Fork_Ex4' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex4
A
B
B
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ 
```

Process Tree:



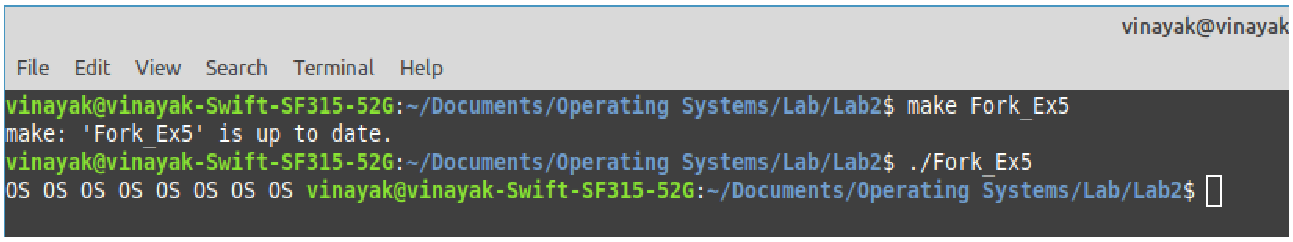
Reasoning:

Initially **A** will be printed and on successful fork() we get 2 processes and after that **B** will be printed **twice** ,once for each process.

Program 5:

```
//Forking Example 5
#include<stdio.h>
#include<unistd.h>

int main()
{
        printf("OS ");
        fork(); //A
        fork(); //B
        fork(); //C
        return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex5
make: 'Fork_Ex5' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex5
OS OS OS OS OS OS OS OS vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$
```

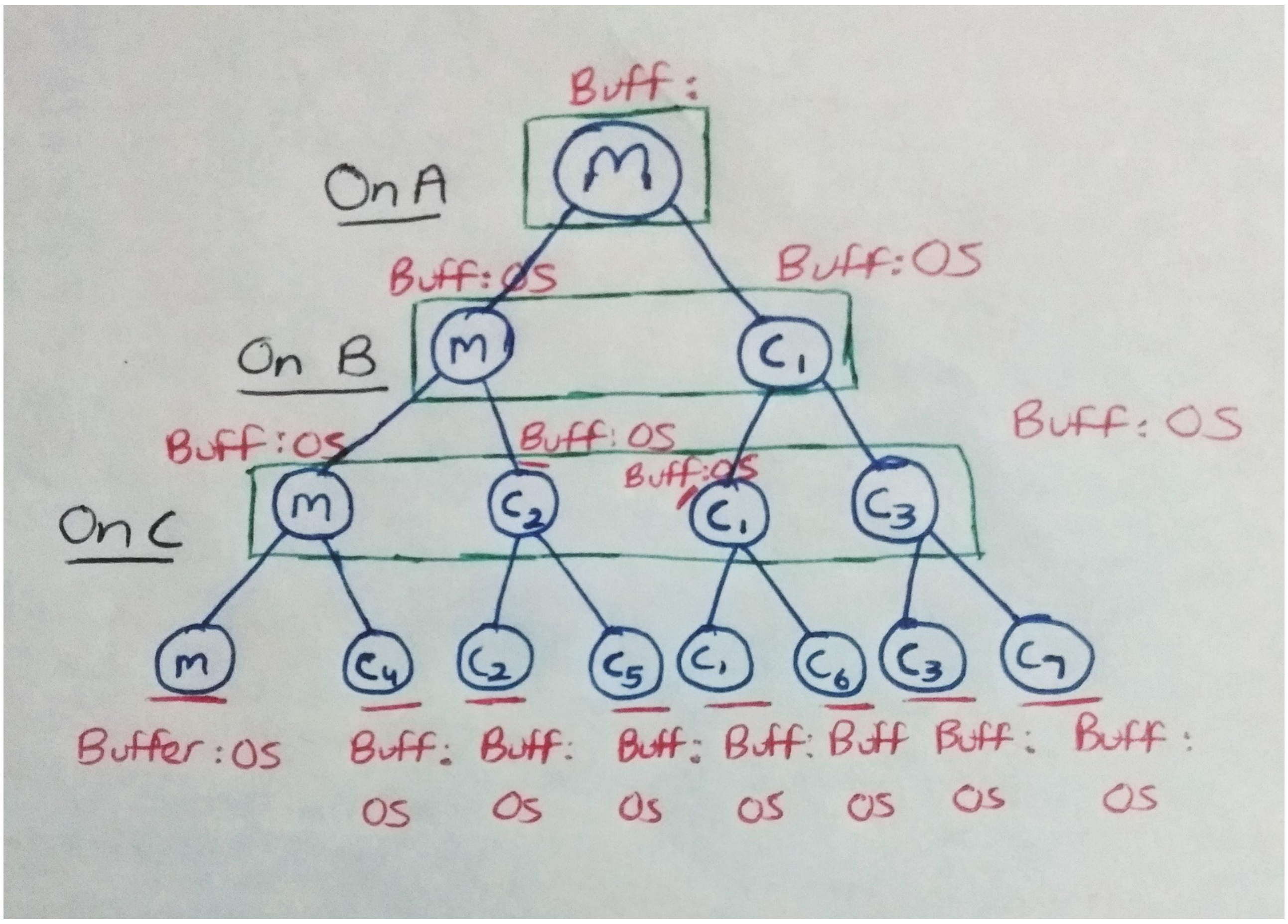
Reasoning:

Initially printf in main stores “OS “ in output buffer.
And there is no “\n” at the end of printf, which means the output is not flushed out.
After printf statement we have fork() call and child inherit the same as parent post forking point., which means all of main()’s child processes will also inherit the same output buffer. Hence, the message “OS ” is present in the output buffers of every child process.

When the process terminates, the buffers are flushed. So we have total of 2^(3) = **8 processes** (including the original process), and the unflushed buffer will be flushed at the termination of each individual process.

Hence OS will be printed 8 times.

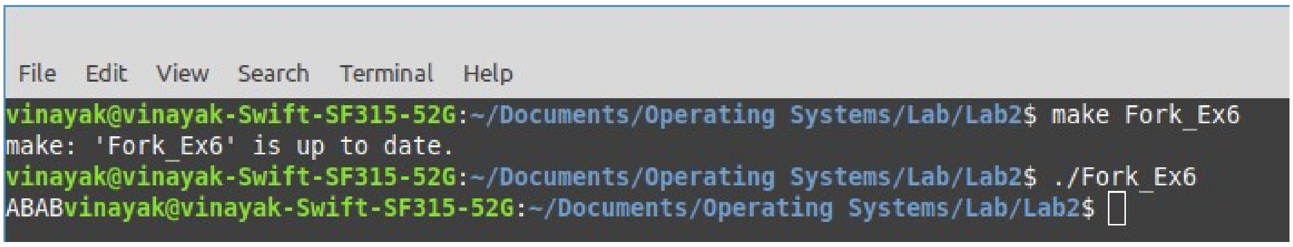
Process Tree:



Program 6:

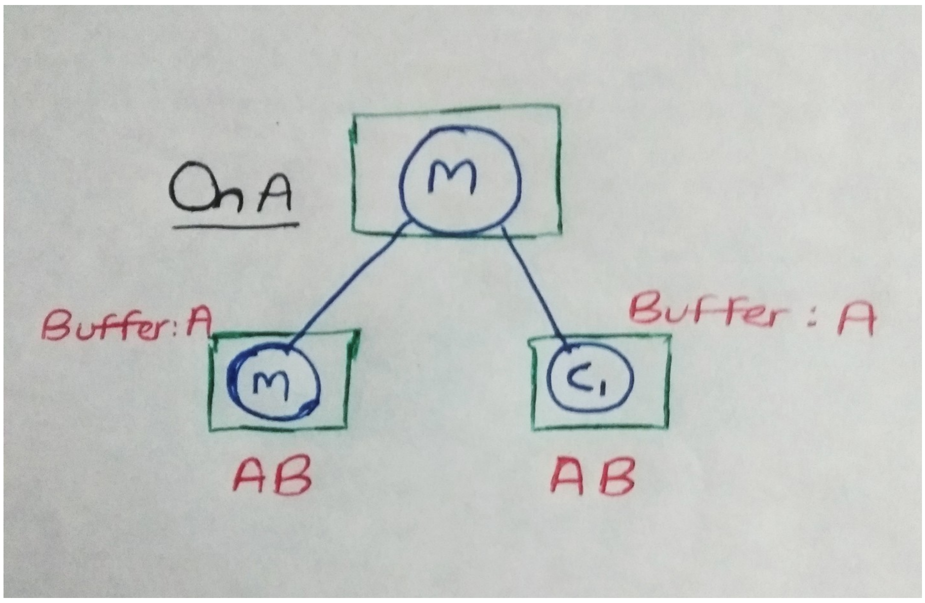
```
//Forking Example 6
#include<stdio.h>
#include<unistd.h>

int main()
{
	printf("A");
	fork(); //A
	printf("B");
	return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex6
make: 'Fork_Ex6' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex6
ABABvinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$
```

Process Tree:



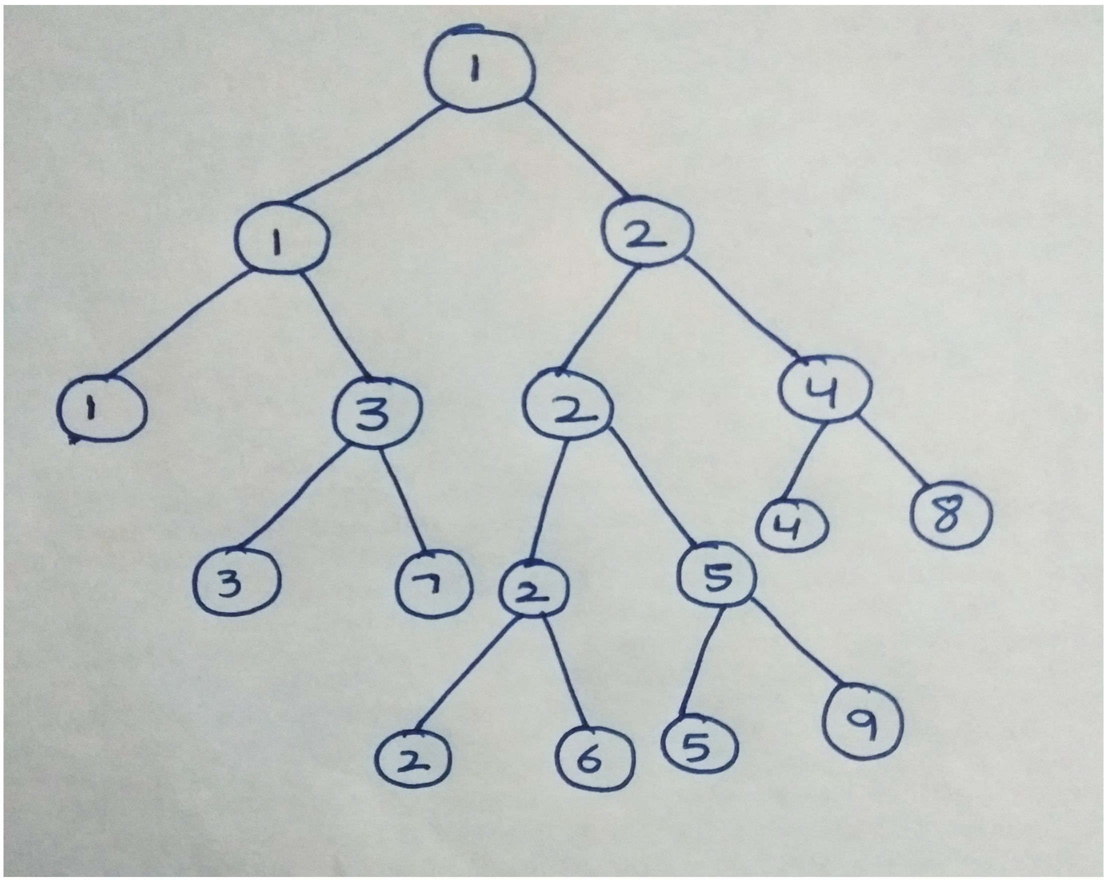
Reasoning:

Since printf statement is not ended with “\n” , which means output buffer is not flushed, Hence, the child processes inherit the same output buffer which already has “A” written on it.

After fork() point we have printf(“B”); which means it will be in both the output buffers of 2 processes after forking point. Then the buffer will be flushed as process terminates.
So. we get the output as **“ABAB”**.

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

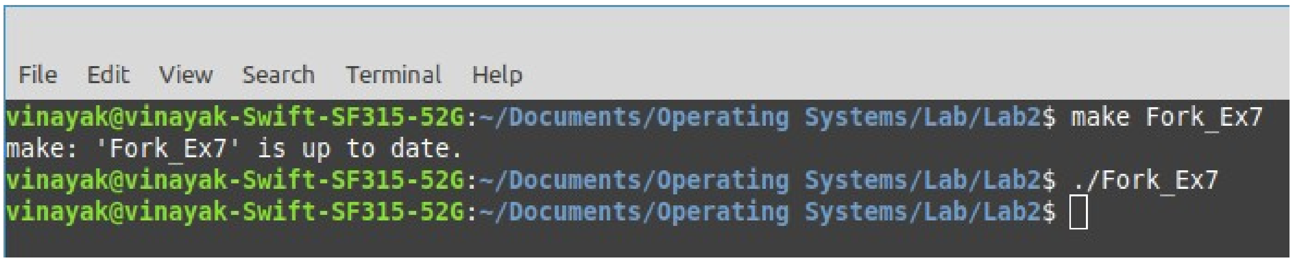
Process Tree:


$$\begin{tikzcd}
 & & & 1 \arrow[dl] \arrow[dr] & & & \\
 & & 1 \arrow[dl] \arrow[d] & & 2 \arrow[d] \arrow[dr] & & \\
 & 1 & 3 \arrow[dl] \arrow[d] & & 2 \arrow[dl] \arrow[d] & 4 \arrow[d] \arrow[dr] & \\
 & 3 & 7 & 2 \arrow[dl] \arrow[d] & 5 \arrow[d] \arrow[dr] & 4 & 8 \\
 & & 2 & 6 & 5 & 9 &
\end{tikzcd}$$

Code:

```
//Forking Example 7
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
	pid_t pid;
	pid = fork(); //process 1 and 2

	if(pid > 0) //for parent 1
	{
		fork(); //process 1 and 3
		if(pid == 0) //process 3
		{
			fork(); //process 3 and 7
		}
	}

	else if (pid == 0) //for child 2
	{
		fork(); //process 2 and 4
		if (pid == 0) //for child 4
		{
			fork(); //process 4 and 8
		}
		else if (pid > 0) //for parent 2
		{
			fork(); // process 2 and 5
			fork(); // process 2 create 2 and 6, process 5 creates 5 and 9
		}
	}

	return 0;
}
```

Output:



```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ make Fork_Ex7
make: 'Fork_Ex7' is up to date.
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$ ./Fork_Ex7
vinayak@vinayak-Swift-SF315-52G:~/Documents/Operating Systems/Lab/Lab2$
```