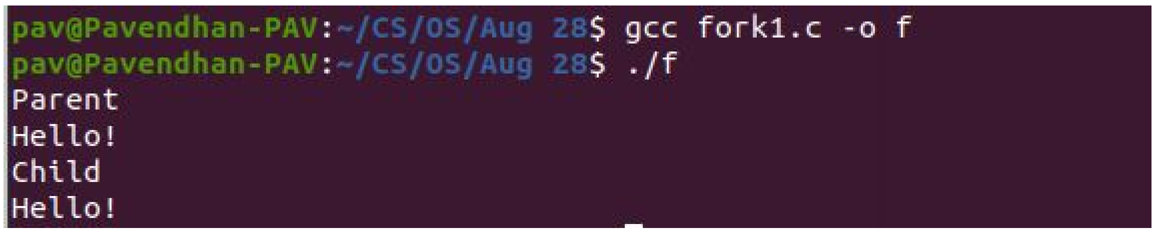
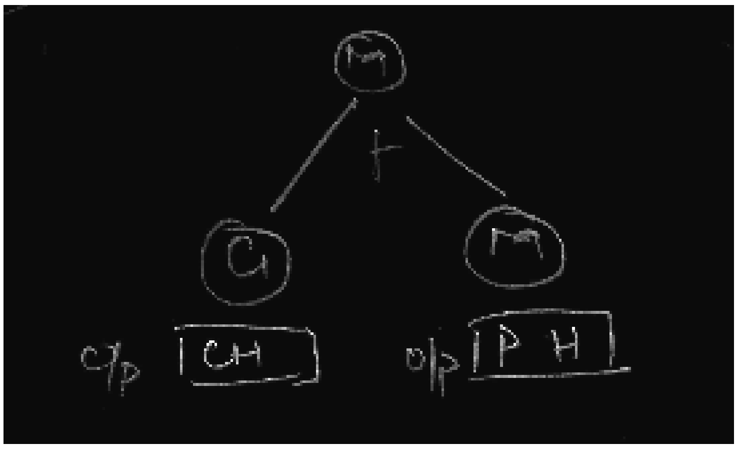
FORK ASSIGNMENT

1)

```
#include <stdio.h>
#include <unistd.h>
int main()
{
   int pid;
   pid = fork(); //A
   if (pid < 0)
      printf("Failed\n");
   else if (pid == 0)
      printf("Child\n");
   else
      printf("Parent\n");
   printf("Hello!\n");
   return 0;
}
```



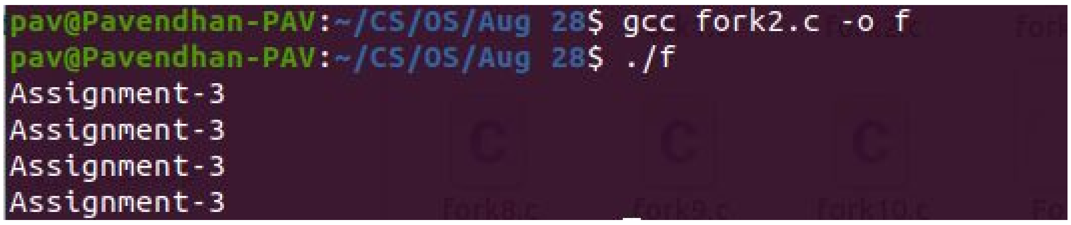
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork1.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
Parent
Hello!
Child
Hello!
```



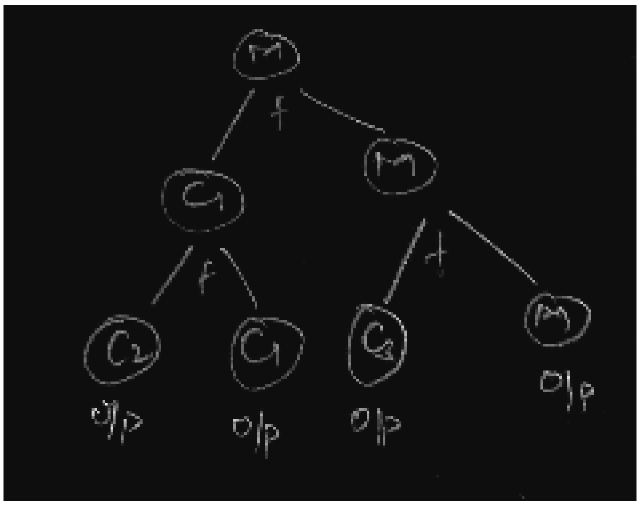
At **A**, when the fork is invoked, in memory there are 2 copies of the program - parent and child respectively with corresponding PIDs greater than 0 and 0 respectively. This can be understood better through binary tree and the order in which the parent child are chosen is upto the kernel, mostly where the parent is chosen.

2)

```
#include<stdio.h>
#include <unistd.h>
int main()
{
   fork(); //A
   fork(); //B
   printf("Assignment-3\n");
}
```



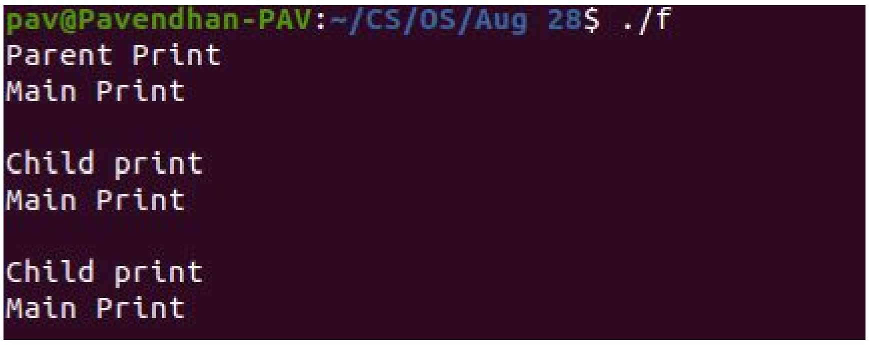
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork2.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
Assignment-3
Assignment-3
Assignment-3
Assignment-3
```



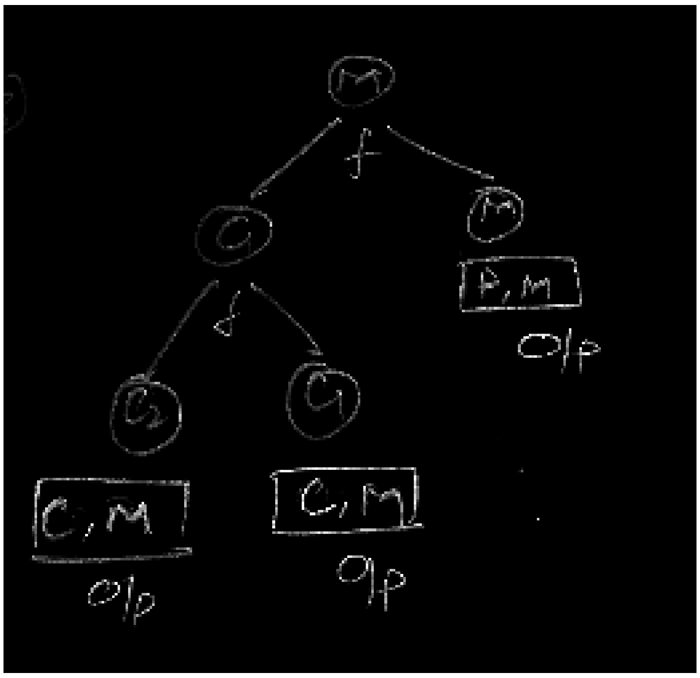
When fork A is invoked there are now 2 copies of the program where each contains its own copy of the fork B, hence there are 2 invoke activities of fork B, giving rise to overall 4 processes in the memory.

3)

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

int main()
{
   int pid;
   pid=fork(); //A
   if (pid<0) fprintf(stderr,"Failed fork \n");
   else if (pid==0)
   {
      fork(); //B
      printf("Child print \n");
   }
   else if (pid>0)
      printf("Parent Print \n");
   printf("Main Print \n\n");
   return 0;
}
```



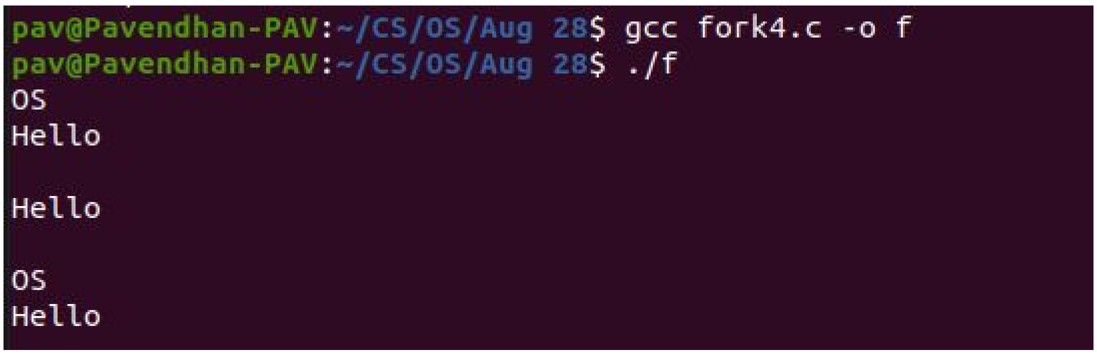
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
Parent Print
Main Print

Child print
Main Print

Child print
Main Print
```



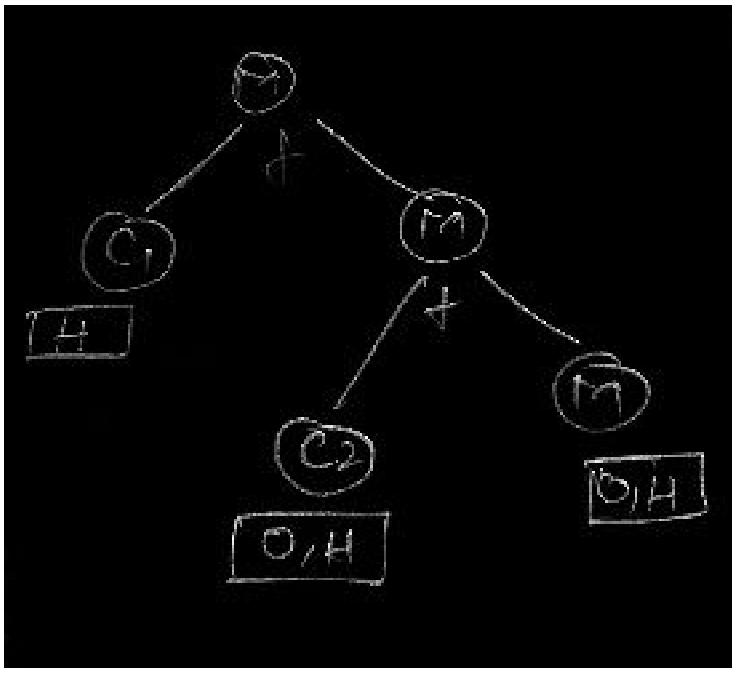
After the first fork, here based on pid another fork is invoked and another copy goes to the else part. When the second/child fork is invoked, in turn another 2 copies are present in memory and prints the corresponding output. (inside the else if loop)

4)

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

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

   if (pid>0) {
      fork(); //B
      printf("OS\n");
   }

   printf("Hello \n\n");
   return 0;
}
```



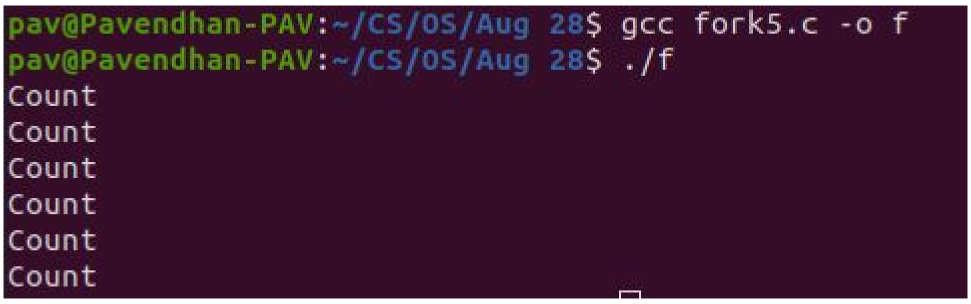
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork4.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
OS
Hello

Hello

OS
Hello
```



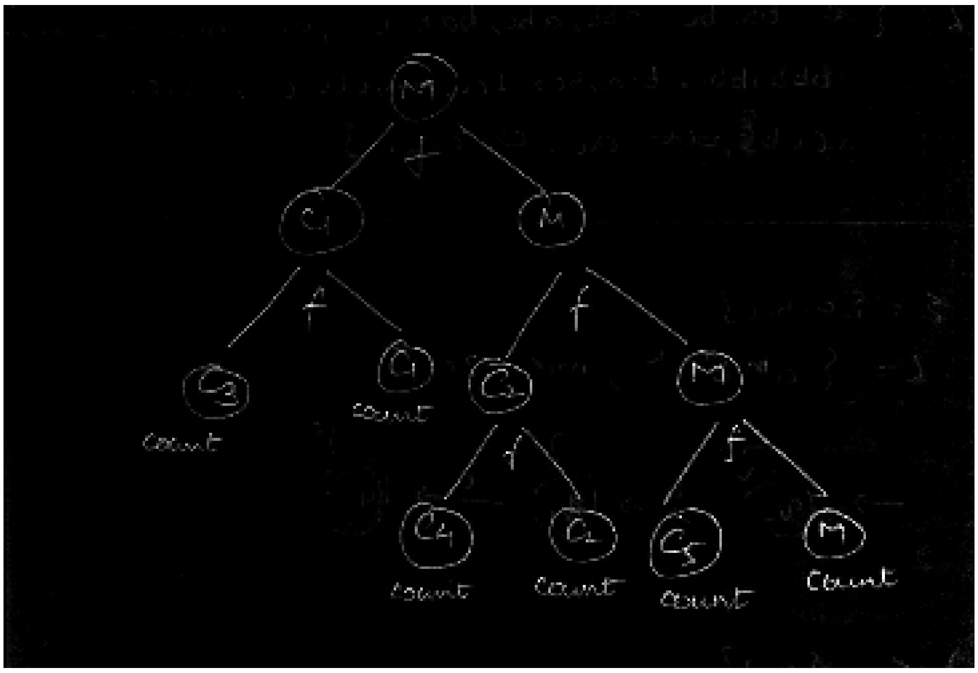
This is the same as the previous program where child fork is instantiated when pid>0 instead of being equal to 0. In the same way there are totally 3 outputs.

5)

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
pid_t pid;
pid=fork();

if (pid!=0)
fork();

fork();
printf("Count \n");

return 0;
}
```



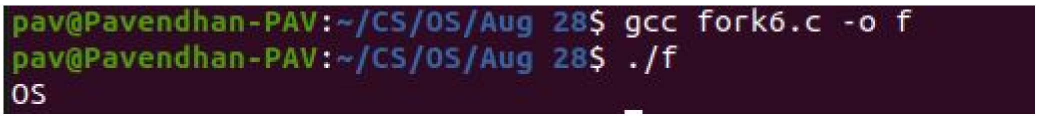
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork5.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
Count
Count
Count
Count
Count
Count
```



Here, the second fork is invoked only when pid is not 0, and the third fork is invoked in every copy of the program. The order of execution is in the hands of the scheduler and cannot be judged beforehand.

6)

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

int main()
{
        printf("OS \n");
        fork();
        fork();
        fork();
}
```

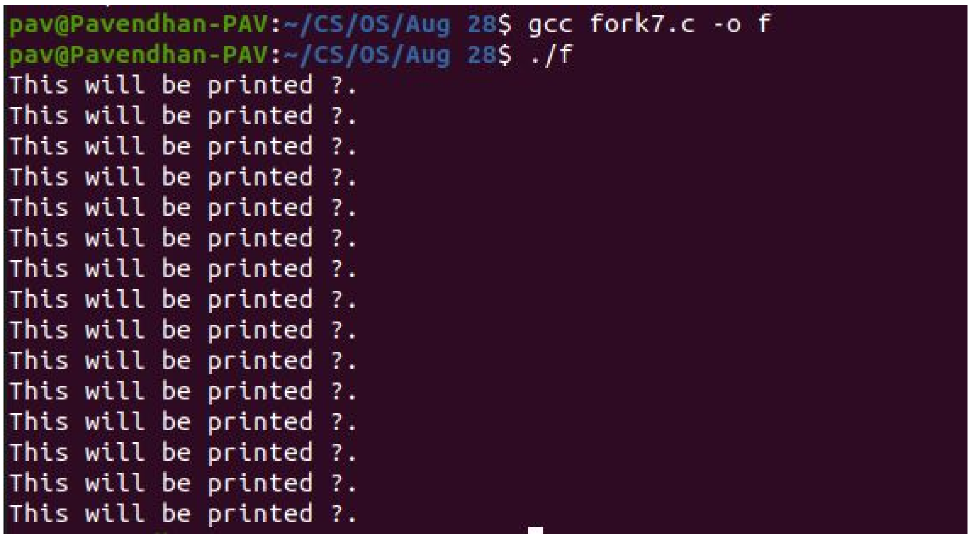


```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork6.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
OS
```

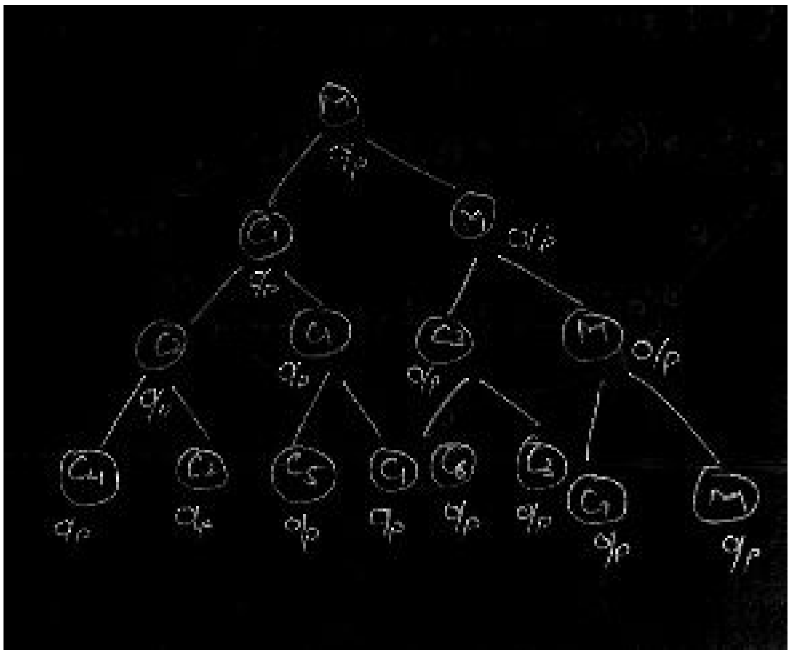
Since, the print statement flushes the buffer due to presence of “\n” and there is no statement after the fork calls so the “OS” output is only due to the parent process. Also, there are 3 fork calls so by the mathematical formula (2^n -1), there are 7 child processes and 8 processes in total including the parent.

7)

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

int main ()
{
printf("This will be printed ?.\n");
fork();
printf("This will be printed ?.\n");
fork();
printf("This will be printed ?.\n");
fork();
printf("This will be printed ?.\n");
return 0;
}
```



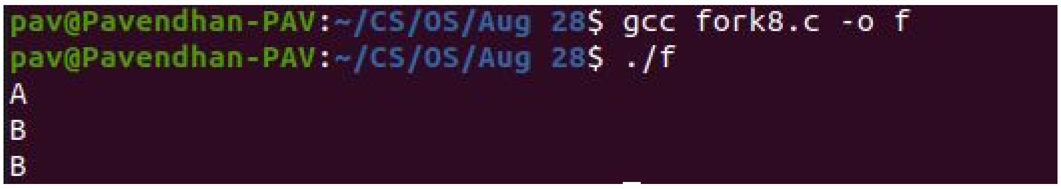
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork7.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ?.
```



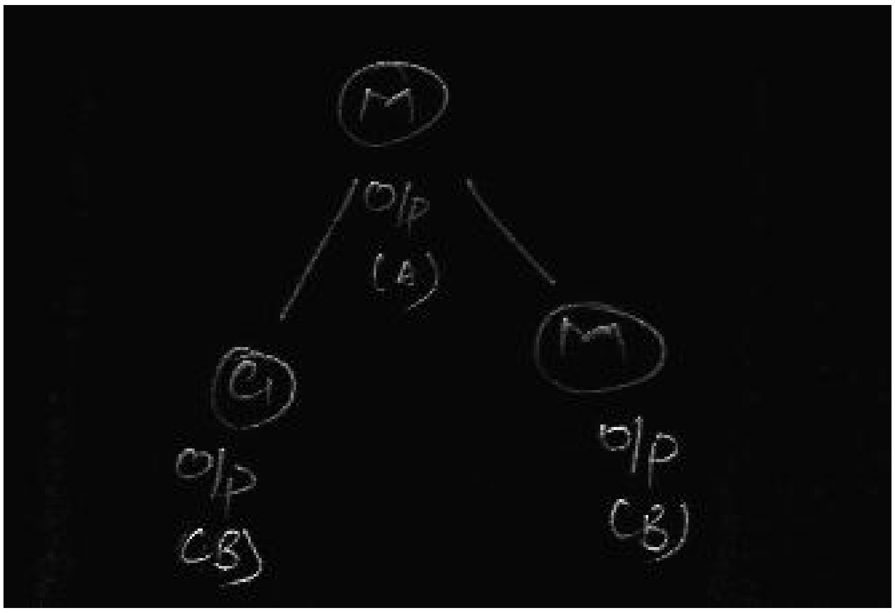
The reasoning can be well justified using the fact that after each fork the process divides into 2. Hence every printf statement is executed by all child processes and the parent process at each stage of fork. This leads to a total of 15 outputs due to recursive child programs and consecutive outputs as shown in the trace above.

8)

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

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



```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork8.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
A
B
B
```

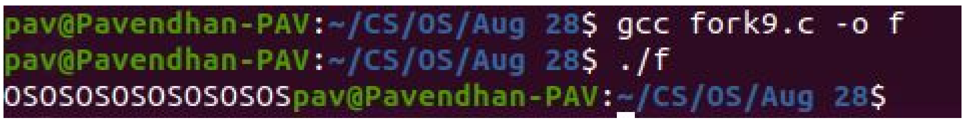


The first output “A” was due to the parent process execution, and the scheduler completes the parent process by printing “B” and finally the child process begins with a newline at the terminal. The scheduler prioritizes the parent process before the child. After parent execution the context is switched.

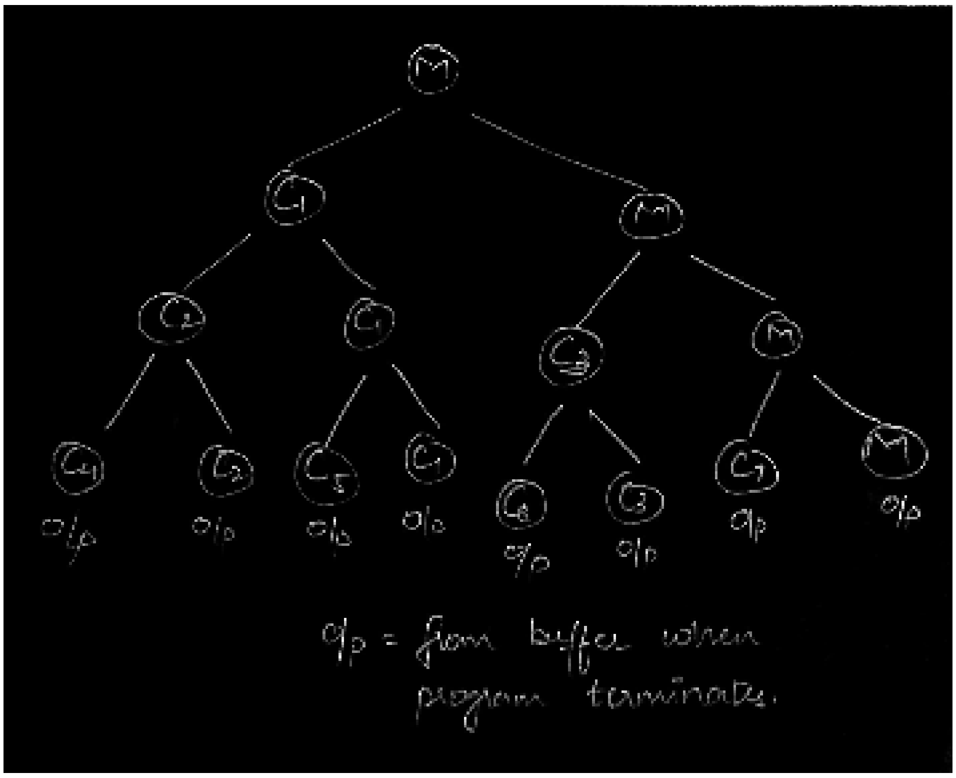
9)

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
```

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



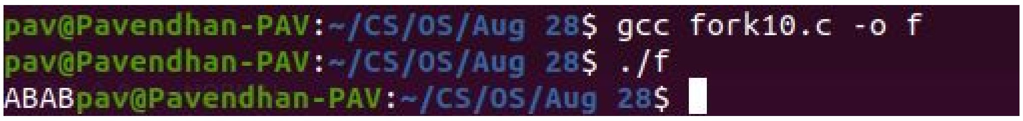
```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork9.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
OSOSOSOSOSOSOSOSpav@Pavendhan-PAV:~/CS/OS/Aug 28$
```



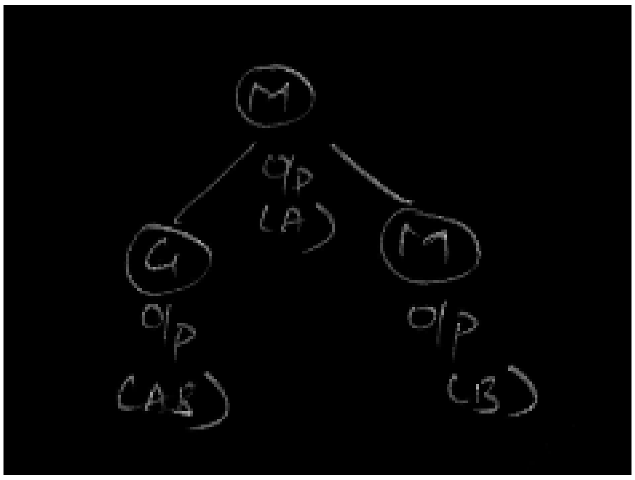
printf() buffers output until a newline is encountered. So the \n-less version stuffs “OS” into the output buffer, forks. Since the 8 processes are identical except their pids, they now have an output buffer that contains “OS”. The execution continues and prints B into the buffers in each process. Both processes then exit, causing a flush of the output buffer, which prints “OS” eight times, once for each process.

10)

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main ()
{
printf("A");
fork();
printf("B");
return 0;
}
```



```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork10.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
ABABpav@Pavendhan-PAV:~/CS/OS/Aug 28$ 
```



Similar to the previous example. Printf() buffers output until a newline is encountered. So the \n-less version stuffs A into the output buffer, forks. Since both processes are identical (except pids), 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.

11)

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

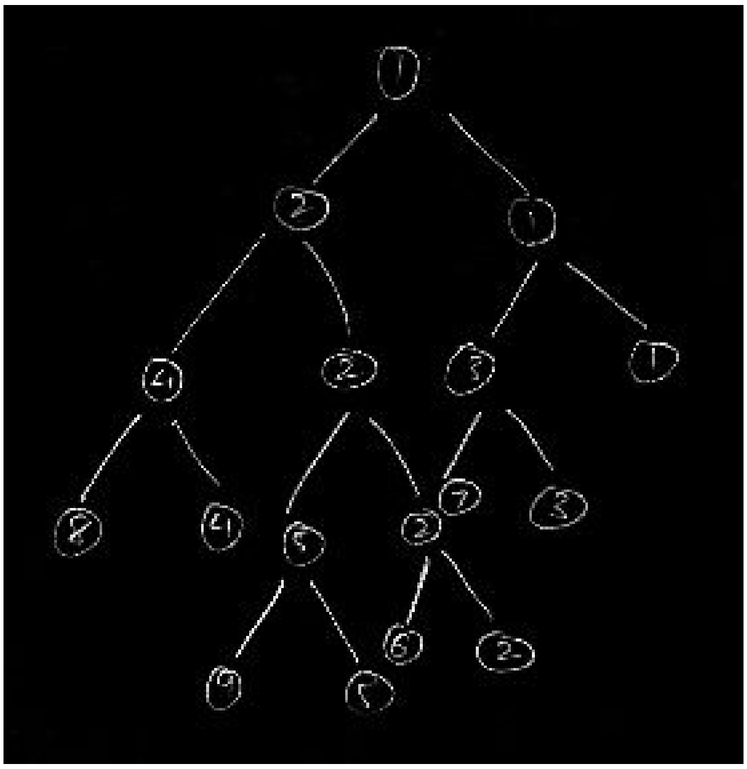
1 forks 2 and 3

2 forks 4 5 and 6

3 forks 7

4 forks 8

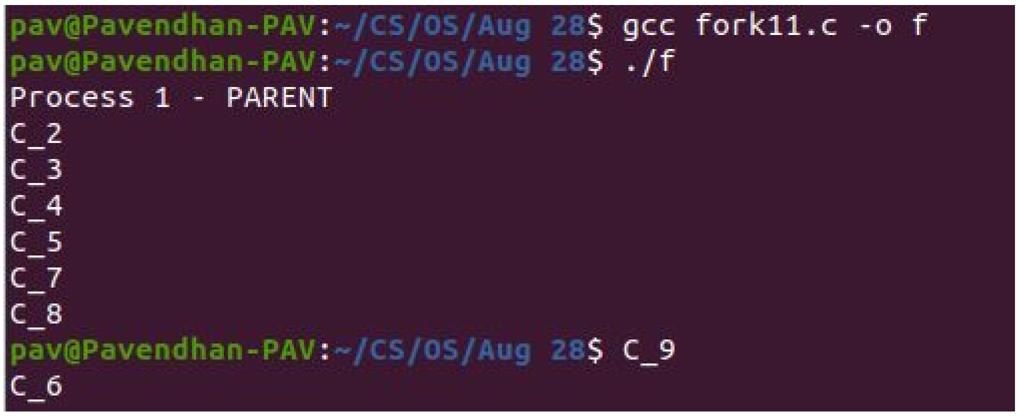
5 forks 9



Multi fork setup is used, implementation of selection statements are use of pids to fork multiple times. Traced the forking tree of the setup shown in the figure above. Here specific children fork corresponding children, this is done with help of pids and if loops.

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

int main ()
{
        printf("Process 1 - PARENT\n");
        pid_t C_2, C_3;
        C_2 = fork(); //-----------------------------------1 forks 2

        if (C_2 == 0)
        {
                //CHILD 2
                printf("C_2\n");
                pid_t C_4, C_5, C_6;
                C_4=fork(); //----------------------------------2 forks 4

                if(C_4==0)
                {
                        //CHILD 4
                        printf("C_4\n");
                        pid_t C_8 =fork(); //----------------------4 forks 8

                        if(C_8==0)
                                printf("C_8\n"); //child 8 code
                }

                else
                {
                        C_5=fork(); //-----------------------------2 forks 5
                        if(C_5==0)
                        {
                                //CHILD 5
                                printf("C_5\n");
                                pid_t C_9=fork(); //--------------------5 forks 9

                                if(C_9==0)
                                        printf("C_9\n"); //CHILD 9
                        }

                        else
                        {
```

```
                    C_6=fork(); //--------------------------2 forks 6
                    if(C_6==0)
                        printf("C_6\n"); //CHILD 6
                }
            }
        }

        else
        {
            C_3 = fork(); //--------------------------------1 forks 3
            if (C_3 == 0)
            {
                //CHILD 3
                printf("C_3\n");
                pid_t C_7;

                C_7=fork(); //------------------------------3 forks 7
                if(C_7==0)
                    printf("C_7\n"); //CHILD 7
            }
        }

        return 0;
}
```



```
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ gcc fork11.c -o f
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ ./f
Process 1 - PARENT
C_2
C_3
C_4
C_5
C_7
C_8
pav@Pavendhan-PAV:~/CS/OS/Aug 28$ C_9
C_6
```