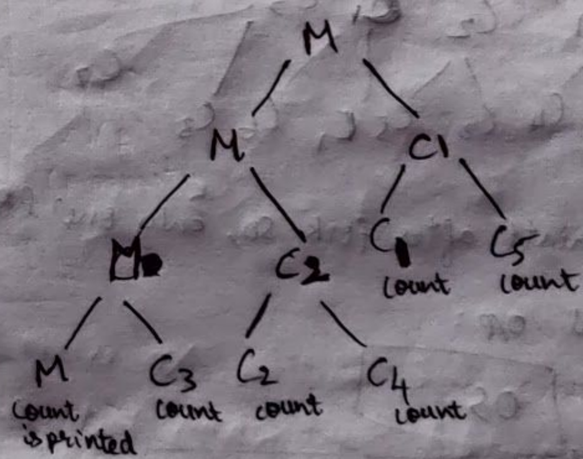
COM301P - OS - Assignment - 2

G.S.Santhosh Raghul
COE18B045

II Fork practice set

Prg 1

```
int main()
{
pid_t pid;
pid = fork();          --> M < M, C1
if(pid != 0)
    fork();            --> inside parent block so M < M, C2
fork();                --> happens in M, C2, C1 --> M < M, C3 , C2 < C2, C4 , C1 < C1, C5
printf("Count \n");
return 0;
}
```

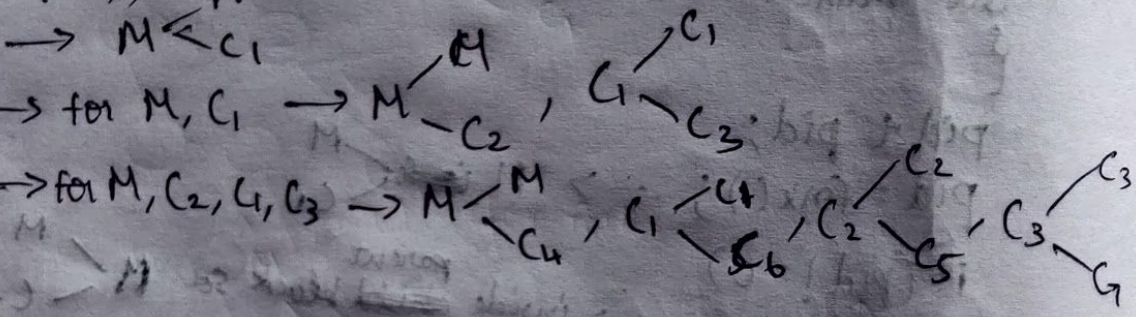


final op:

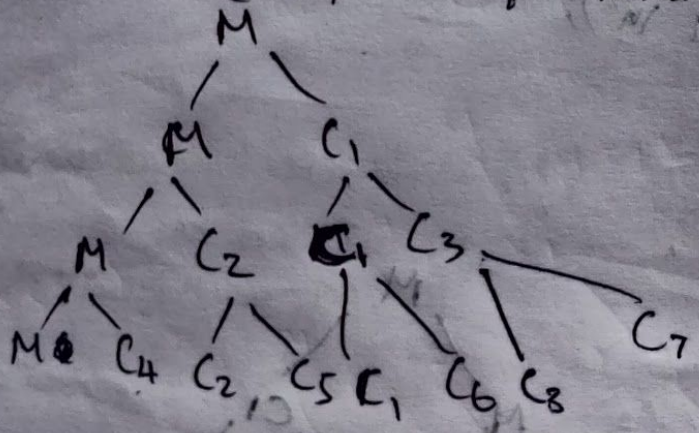
Count.
Count
Count
Count
Count
Count

Prg 2

```
int main()
{
printf("OS\n");
fork(); → M ← C1
fork(); → for M, C1 → M
fork(); → for M, C2, C1, C3 → M
}
```



(OS is printed before forking)



no printf after fork so, only one line o/p

final o/p:

OS

Prg-3

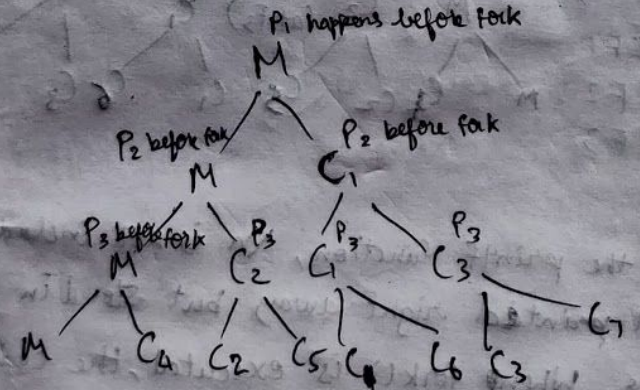
```
int main()
{
printf("This will be printed ?. \n");
fork();
printf(" This will be printed ?. \n");
fork();
printf("This will be printed? . \n");
fork();
printf("This will be printed ? \n");
return 0;
}
```

fork(); → M, C1

fork(); → for M, C1 → M, C2 ; C1, C3

fork(); → for M, C2, C1, C3 → M, C4 ; C2, C5 ; C1, C6 ; C3, C7

Let the 4 printf statements be P_1, P_2, P_3, P_4



P_4 happens for M, C_1 to C_7.

assuming child executes before parent, printf statements are executed in the foll order.

printf calling process	P_1	P_2	P_3	P_4	P_4	P_3	P_4	P_4	P_2	P_3	P_4	P_4	P_3	P_4	P_4
	M	C_1	C_3	C_7	C_3	C_1	C_6	C_1	M	C_2	C_5	C_2	M	C_4	M

~~Final o/p:~~

~~This will be~~

1st printf is executed once

2nd printf is executed twice

3rd printf is executed four times

4th printf is executed eight times

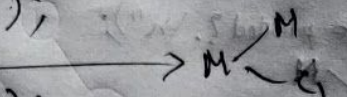
(Total = 15)

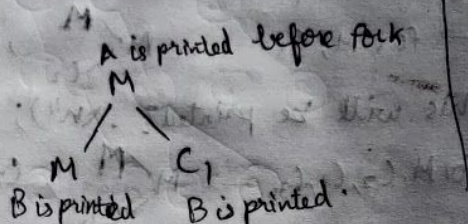
Prog-3 final O/P: (15 lines)

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.

Prg 4

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



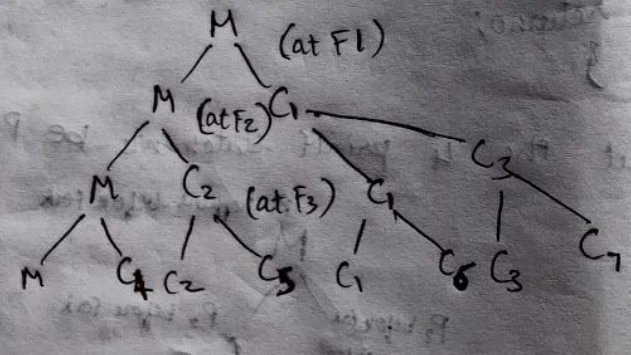


Final o/p:

A
B
B

Prg 5

```
int main()
{
printf("OS");
fork(); // F1
fork(); // F2
fork(); // F3
}
```



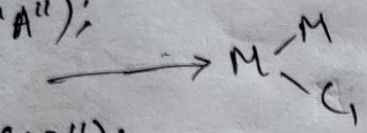
here, in the printf function, \n is not included. So, OS is not printed right away but stored in the output buffer. When fork() is executed, the child processes inherit everything from the parent including the output buffer. So, after all 3 forks are over, there are 8 processes in the memory. The content of the output buffer is flushed onto stdout when there is \n or when the process terminates. Since there are no statements after the 3rd fork, each process terminates and the buffer content of all the 8 processes are flushed on the stdout one by one, resulting in a final output of:

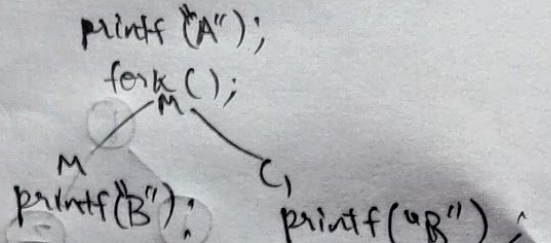
OS OS OS OS OS OS OS OS

(OS 8 times)

Prg-6

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

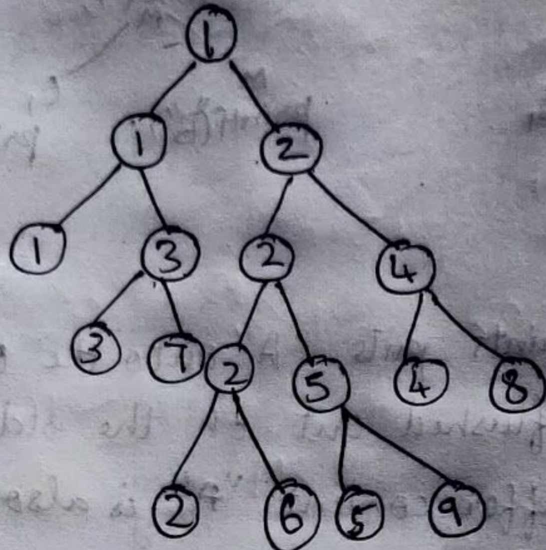




Again, the 1st printf puts "A" onto the output buffer. Before this could be flushed out on the stdout, forking happens and this buffer content "A" is also inherited by the child C_1. ~~How~~ Now there are 2 processes with "A" in the buffer. ~~Next w~~ Lets assume child is executed first. printf("B"); puts "B" onto the buffer and the buffer content is now AB. since the next line is return 0;, the buffer content is flushed out on the screen. Now, the printf("B"); statement is executed for the parent and the buffer content becomes AB. Again on encountering return 0;, the buffer content is flushed out to ~~the~~ give a final output of :

ABAB

Tree from the given question is:



pseudo code:

let blocks be represented by indentation.

```
fork();          1 -> 1, 2
| parent block:
|   | fork();    1 -> 1, 3
|   | child block:
|   |     | fork();   3 -> 3, 7
| child block:
|   | fork();    2 -> 2, 4
|   | parent block:
|   |     | fork();   2 -> 2, 5
|   |     | fork();   — 2 -> 2, 6  and  5 -> 5, 9
|   | child block:
|   |     | fork();   4 -> 4, 8
```

C code for the above pseudo code

```
int main()
{
    int x=0;
    pid_t pid1,pid2;
    pid1=fork(); // 1 forks 2 here
    if(pid1==-1)
        printf("fork failed\n");
    else if(pid1>0) // parent block
    {
        pid2=fork(); // 1 forks 3 here
        if(pid2==-1)
            printf("fork failed");
        else if(pid2==0) // inner child block
            fork(); // 3 forks 7 here
    }
    else //child block
    {
        pid2=fork(); // 2 forks 4 here
        if(pid2==-1)
            printf("fork failed");
        else if(pid2>0) // inner parent block
        {
            fork(); // 2 forks 5 here
            fork(); // 2 forks 6 and 5 forks 9 here
/* the above fork can also be split as 1 fork() in parent block and another
fork() in child block */
        }
        else // inner child block
        {
            fork(); // 4 forks 8 here
        }

    }
    while((x=wait(NULL))!=-1);
// to make sure all children terminate before the parent to get correct pid, ppid
    printf("This is process %d with parent %d\n",getpid(),getppid());
    return 0;
}
```

Output of the code

```
santi@edith:~/OS/L2$ ./prg7
This is process 77157 with parent 77154
This is process 77158 with parent 77155
This is process 77159 with parent 77153
This is process 77160 with parent 77156
This is process 77154 with parent 77152
This is process 77155 with parent 77153
This is process 77156 with parent 77153
This is process 77153 with parent 77152
This is process 77152 with parent 77100
santi@edith:~/OS/L2$ ps -A | grep 77100
  77100 pts/0    00:00:00 bash
santi@edith:~/OS/L2$
```

Tree formed using the output and comparison with the given tree