OPERATING SYSTEMS - COM301P

LAB ASSIGNMENT - 2

FORKING PRACTICE SET

VIKNESH RAJARAMON

COE18B060

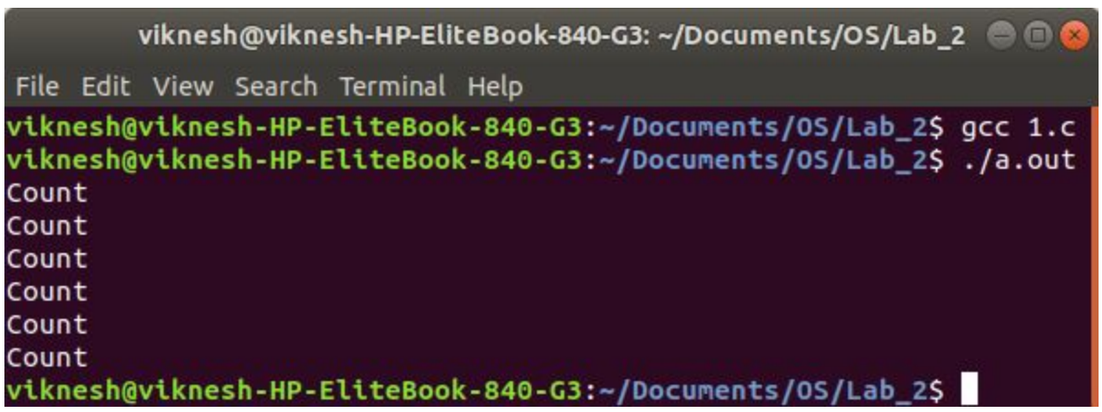
PRACTICE QUESTIONS

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;
}
```

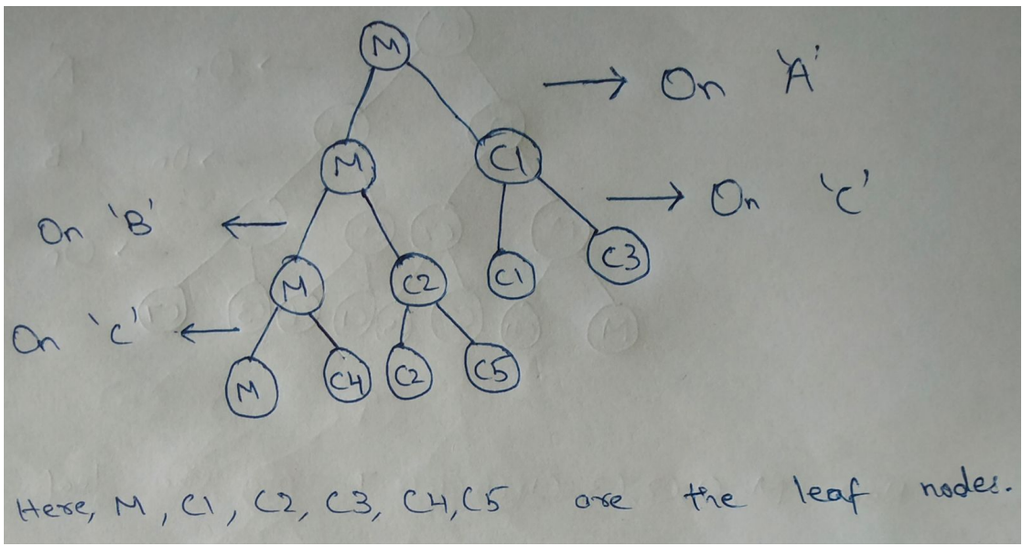
MANUAL TRACE:

Count
Count
Count
Count
Count
Count

OUTPUT:



UNDERSTANDING:



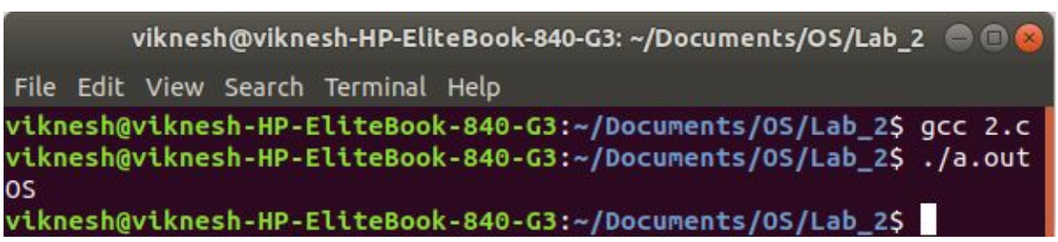
For every leaf node, we get one **"Count"** message. Since there are 6 leaf nodes, we have 6 **"Count"** messages which validates the output we have received.

2)
```
int main()
{
printf("OS\n");
fork();      //A
fork();      //B
fork();      //C
}
```

MANUAL TRACE:

OS

OUTPUT:



UNDERSTANDING:

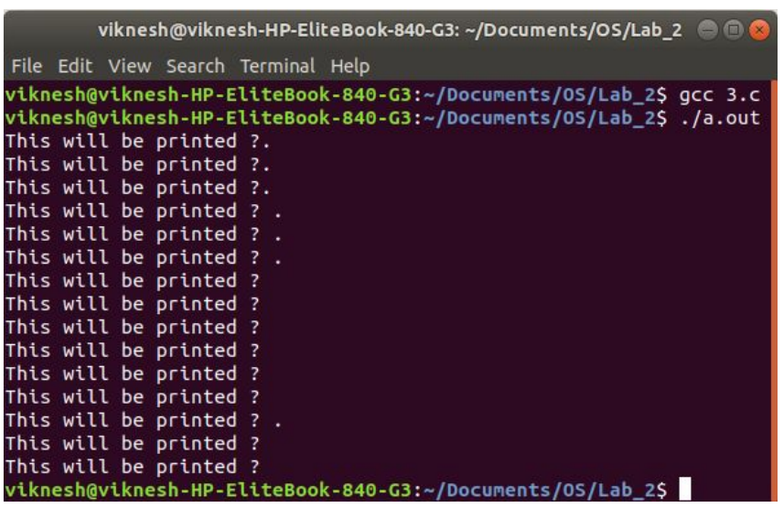
Since all the forking takes place after the **"printf"** statement, only the parent process has the **"printf"** statement. Therefore, the **"OS"** message is printed only once. Also, there is a **"\n"** at the end of the **"printf"** statement. So the buffer is emptied before forking.Since the buffer is empty before forking, all the child processes have an empty buffer and so print nothing.

3)
```
int main()
{
printf("This will be printed ?.\n");       //printf 1
fork();       //A
printf("This will be printed ?.\n");       //printf 2
fork();       //B
printf("This will be printed ? .\n");       //printf 3
fork();       //C
printf("This will be printed ?\n");       //printf 4
return 0;
}
```

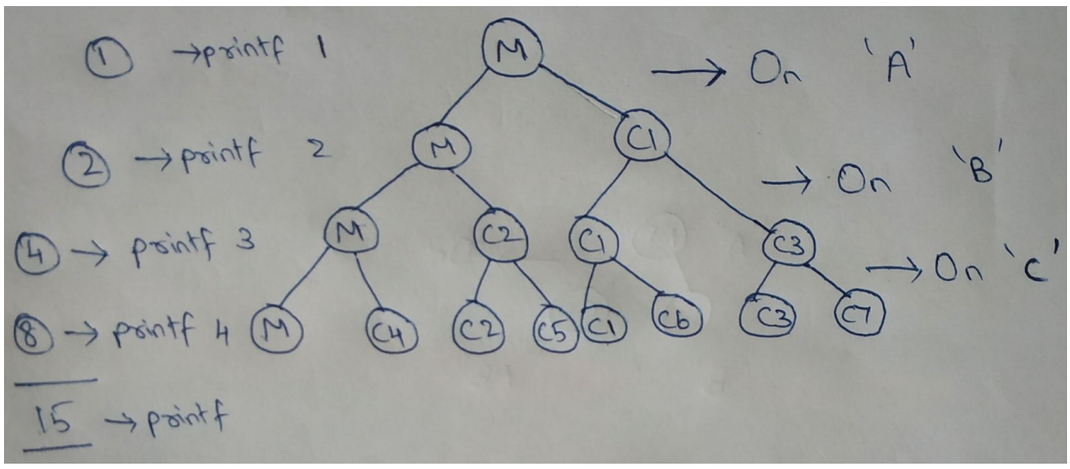
MANUAL TRACE:

This will be printed ?. //printf 1
This will be printed ?. //printf 2
This will be printed ?. //printf 2
This will be printed ? . //printf 3
This will be printed ? . //printf 3
This will be printed ? . //printf 3
This will be printed ? . //printf 3
This will be printed ? //printf 4
This will be printed ? //printf 4
This will be printed ? //printf 4
This will be printed ? //printf 4
This will be printed ? //printf 4
This will be printed ? //printf 4
This will be printed ? //printf 4
This will be printed ? //printf 4

OUTPUT:



UNDERSTANDING:



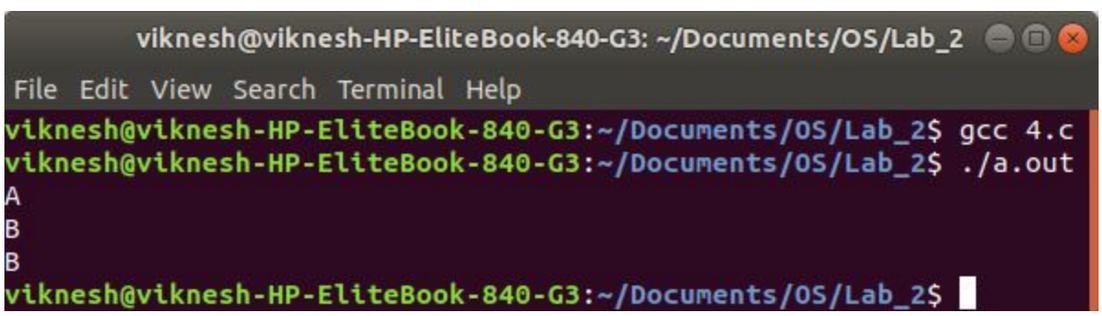
If we look at the process tree, **"printf 1"** is only with the parent process. Therefore, it is printed only once. Similarly after **"fork A"**, the parent process creates a child process **"C1"** after which **"printf 2"** statement is printed. Therefore, 2 **"printf 2"** statements are printed. Similarly, after **"fork B"**, 4 **"printf 3"** statements are printed and after "fork C", 8 **"printf 4"** statements will be printed. The order of the statements is left to the kernel, but the total number of distinct statements remains the same in all cases.

4)
```
int main()
{
printf("A \n");
fork();       //A
printf("B\n");
return 0;
}
```

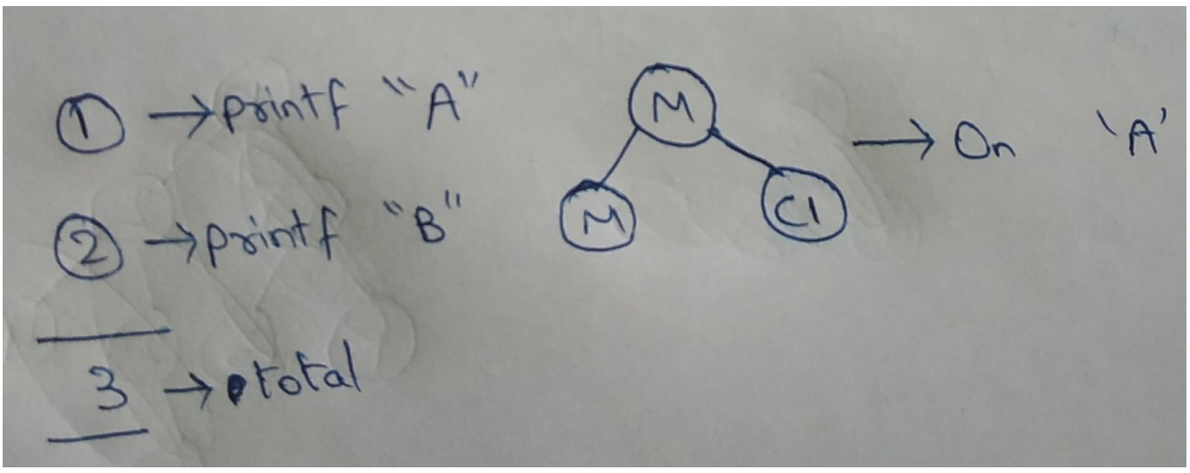
MANUAL TRACE:

A
B
B

OUTPUT:



UNDERSTANDING:



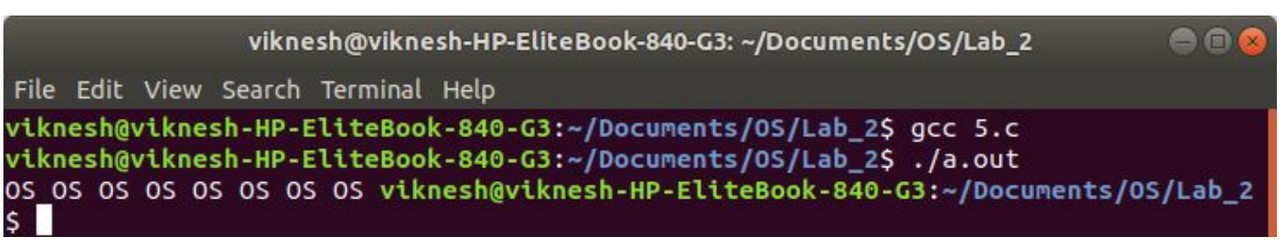
First **"A"** is printed. Before **"fork A"** occurs, the buffer is emptied. Therefore, the **"A"** message is printed only once. After "fork A", the parent process creates a child process. Therefore, 2 processes exist having the **"B"** message. So the **"B"** message is printed 2 times.

5)
```
int main()
{
printf("OS ");
fork();      //A
fork();      //B
fork();      //C
}
```

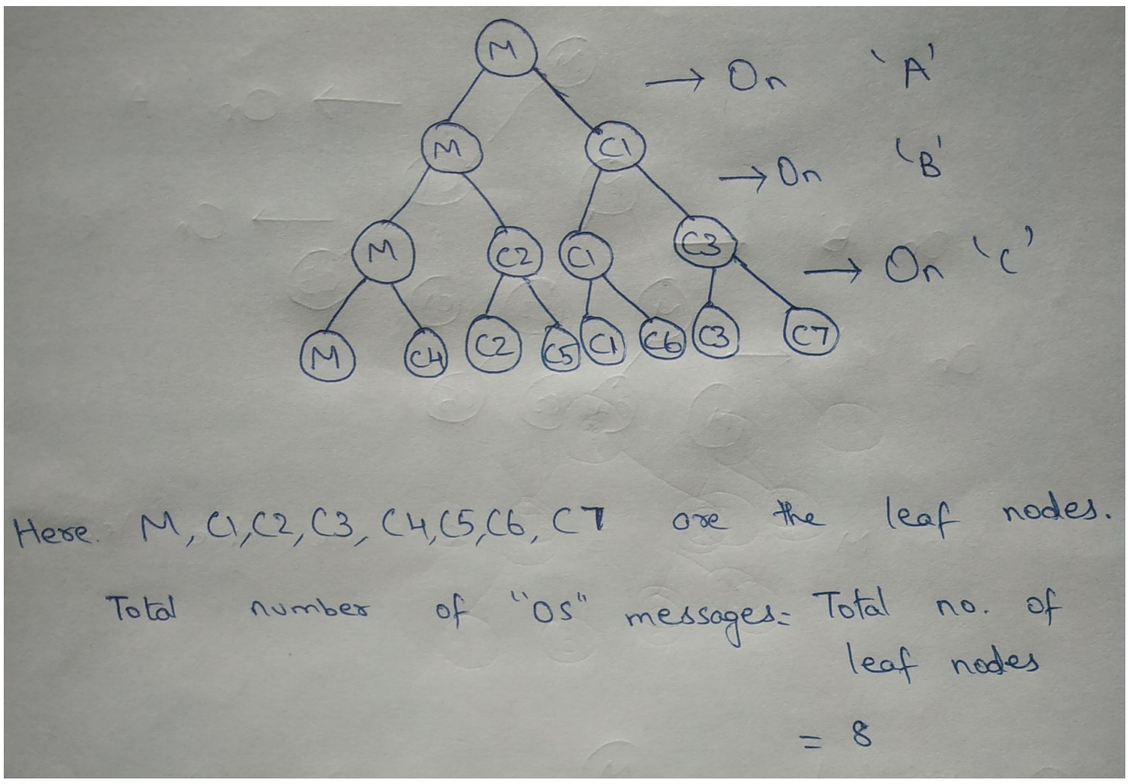
MANUAL TRACE:

OS OS OS OS OS OS OS OS

OUTPUT:



UNDERSTANDING:



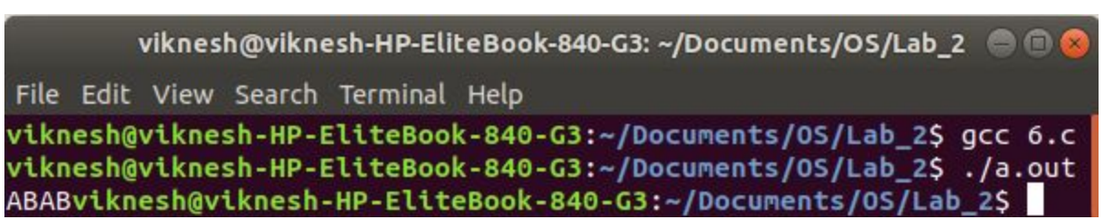
In this question, there is no **"\n"** character in the printf statement. Since the buffer is flushed either when there is a **"\n"** character or the process terminates, the buffer is not empty when **"fork A"**, **"fork B"** and **"fork C"** occur. Since there are 8 processes running after **"fork C"**, we get 8 **"OS "** messages flushed to the STDOUT after the 8 processes terminate.

6)
```
int main()
{
printf("A");
fork();       //X
printf("B");
return 0;
}
```

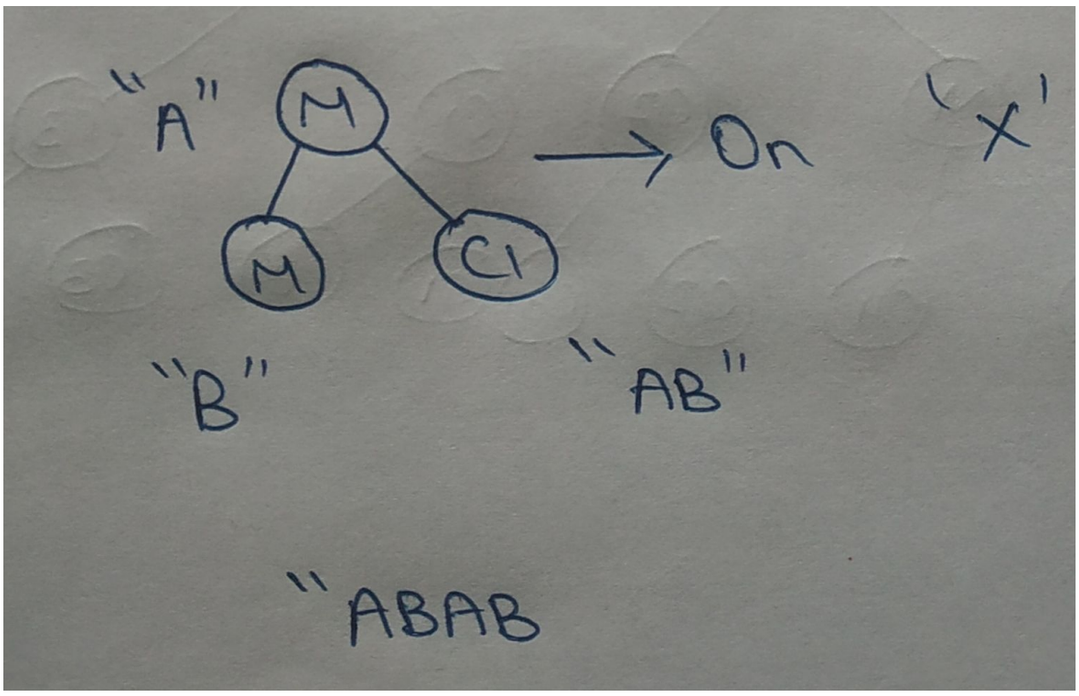
MANUAL TRACE:

ABAB

OUTPUT:



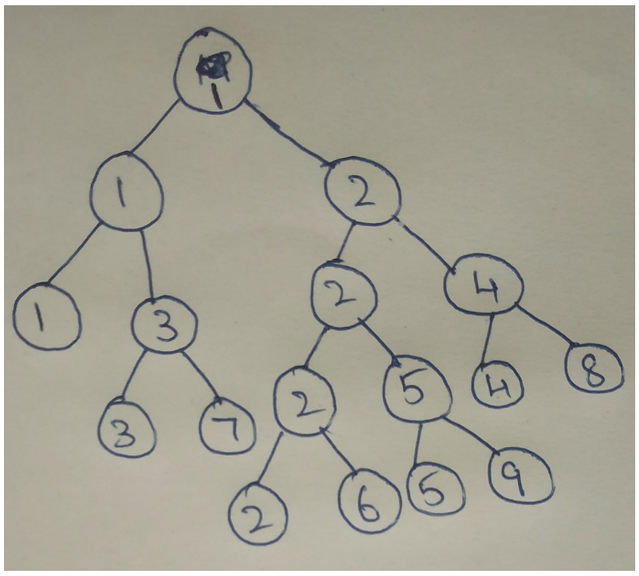
UNDERSTANDING:



In this question, there is no **"\n"** character in the printf statement. Since the buffer is flushed either when there is a **"\n"** character or the process terminates, the buffer is not empty when **"fork X"** occurs. Since there are 2 processes running after **"fork X"**, we get 2 **"AB"** messages flushed to the STDOUT after the 2 processes terminate.

7) Express the following in a process tree setup and also write the C code for the same setup
   a) 1 forks 2 and 3
   b) 2 forks 4 5 and 6
   c) 3 forks 7
   d) 4 forks 8
   e) 5 forks 9

PROCESS TREE:

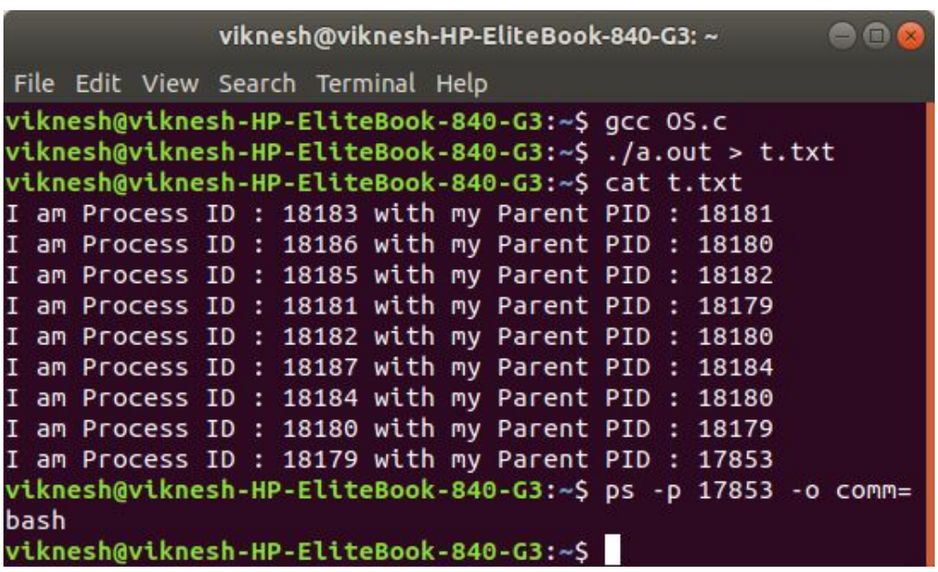


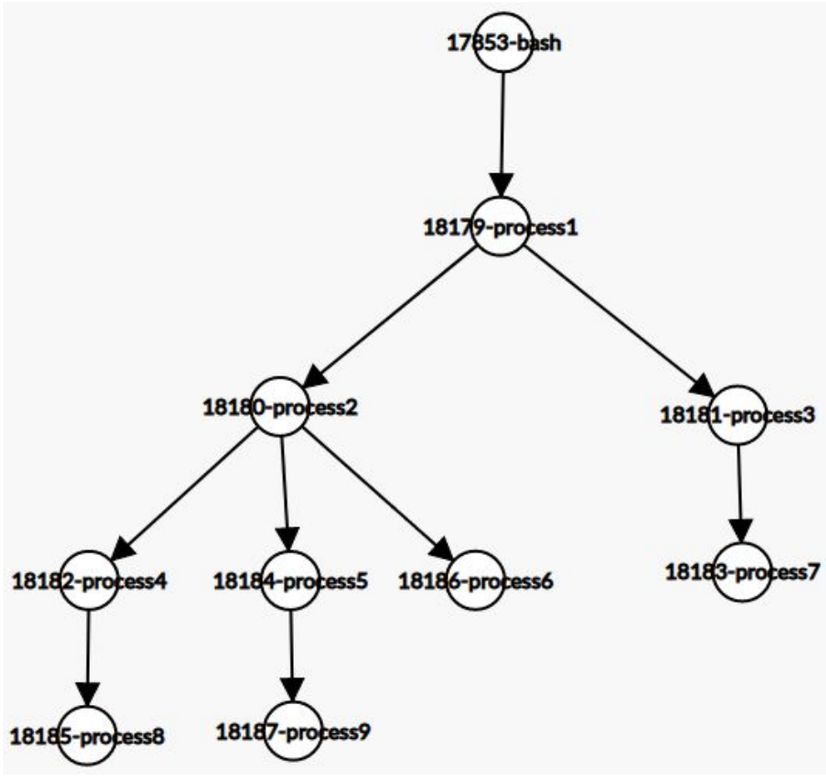
C CODE:

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

int main()
{
    pid_t pid;
    int x=0;
    pid=fork();  //1 forks 2
    if(pid>0)
    {
            pid=fork();  //1 forks 3
            if(pid==0)
            {
                    pid=fork();  //3 forks 7
            }
    }
    else if(pid==0)
    {
            pid=fork();  //2 forks 4
            if(pid>0)
            {
                    pid=fork();  //2 forks 5
                    pid=fork();  //2 forks 6 and 5 forks 9
            }
            else if(pid==0)
            {
                    pid=fork();  //4 forks 8
            }
    }
    while((x=wait(NULL))!=-1);
    printf("I am Process ID : %d with my Parent PID : %d\n",getpid(),getppid());
    return 0;
}
```

OUTPUT:





In the above output, if we look at the parent process ID of **18179**, we can see that the parent process is the bash. Therefore, our main() has **PID 18179**. If we look at the processes that have their parent PID as **18179**, we have two child processes - **18180** and **18181** (for process 2 and 3 respectively). For process ID **18180** (process 2), we have 3 child processes - **18182**, **18184**, **18186** (for process 4, 5 and 6 respectively). For process ID **18181** (process 3), we have 1 child process - **18183** (for process 7 respectively). For process ID **18182** (process 4), we have 1 child process - **18185** (for process 8 respectively). For process ID **18184** (process 5), we have 1 child process - **18187** (for process 9 respectively).