Operating System COM301P

Programming Assignment Lab - 2

By :

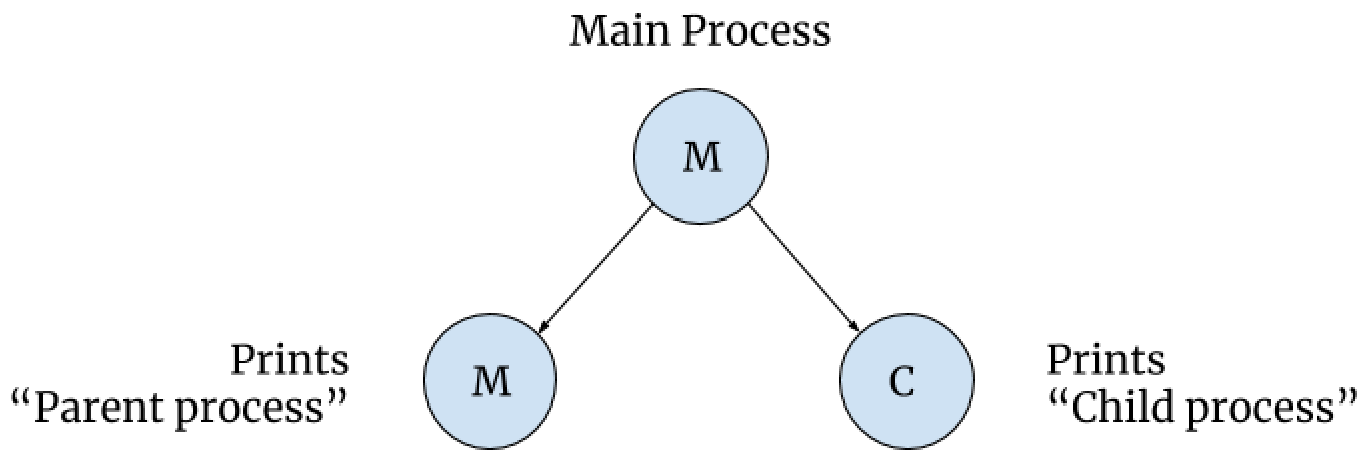
Sai Kaushik S
CED18I044

C code:

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

int main(){
   int pid = fork();
   if(pid < 0) printf("Fork failed\n");
   else if(pid == 0) printf("Child process\n");
   else printf("Parent process\n");
   return 0;
}
```

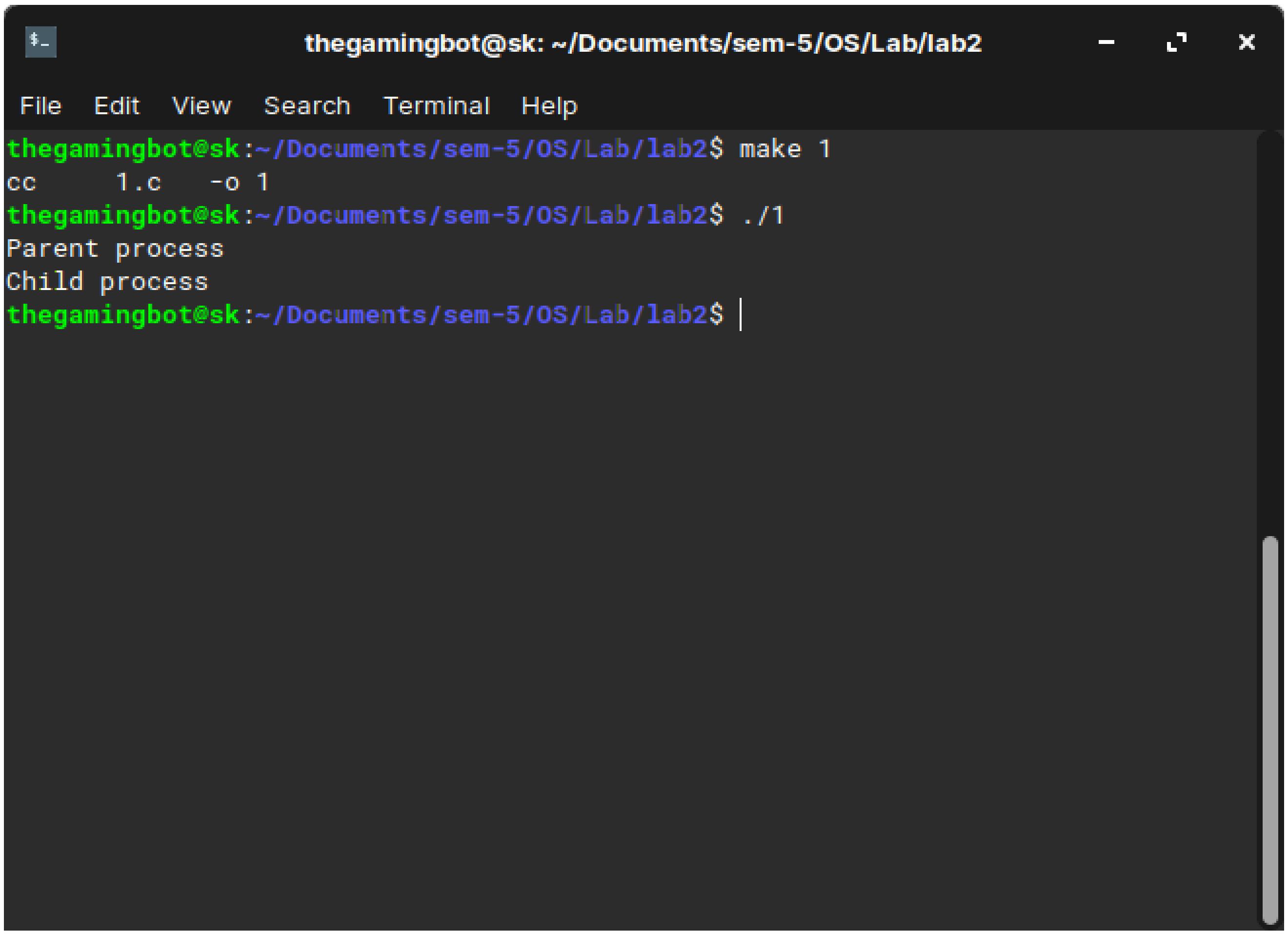
Binary Tree:



Explanation:

When the main process(M) is forked, it creates a new child process(C), with pid = 0. The main process(M) has a pid > 0 as it is a parent. So, "Parent process" is printed by the main process(M). Now, the child process(C) executes the same program starting from the instruction after the fork call. Since a child process(C) has a pid = 0, it prints "Child process" and exits.

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

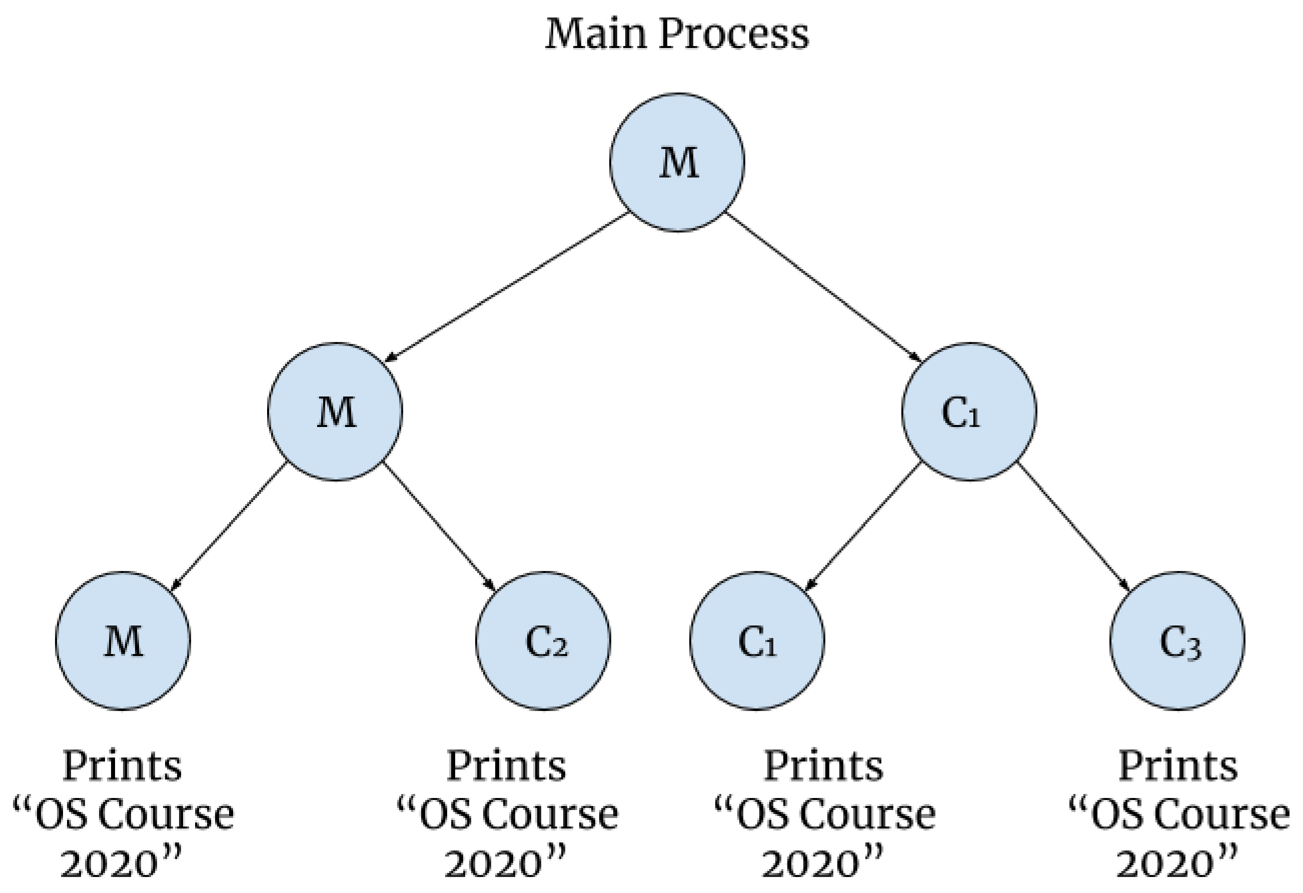
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 1
cc     1.c   -o 1
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./1
Parent process
Child process
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ 
```

C code:

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

int main(){
   fork();
   fork();
   printf("OS Course 2020\n");
   return 0;
}
```

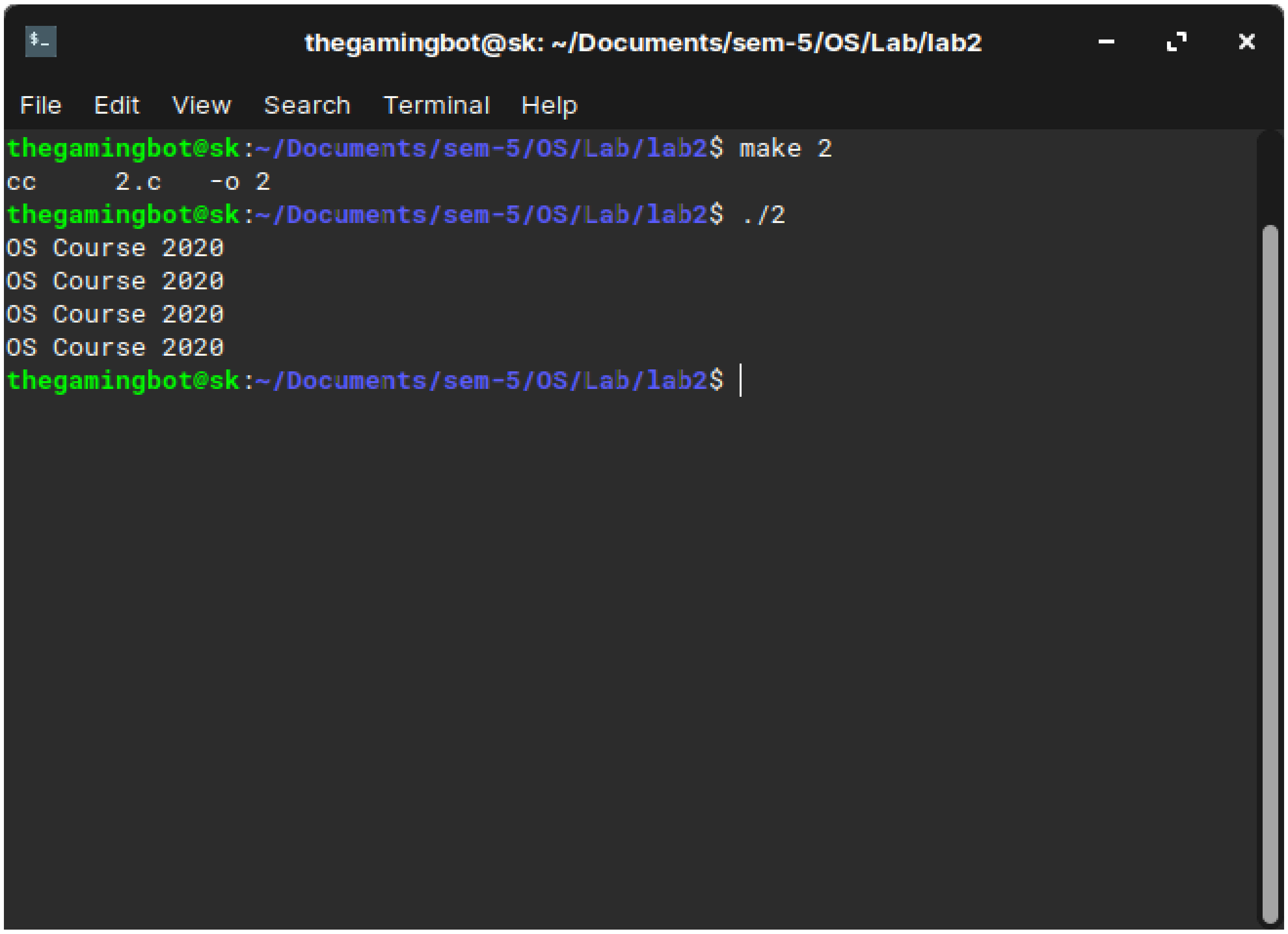
Binary Tree:



Explanation:

When the main process(M) is forked, it creates a new child process(C_1). The main process(M) continues and gets forked, creating another child(C_2). The child process(C_1) is forked and it gives another child process(C_3). There is a print statement at the end of the program and there are 4 processes created. So, "OS Course 2020" is printed 4 times.

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

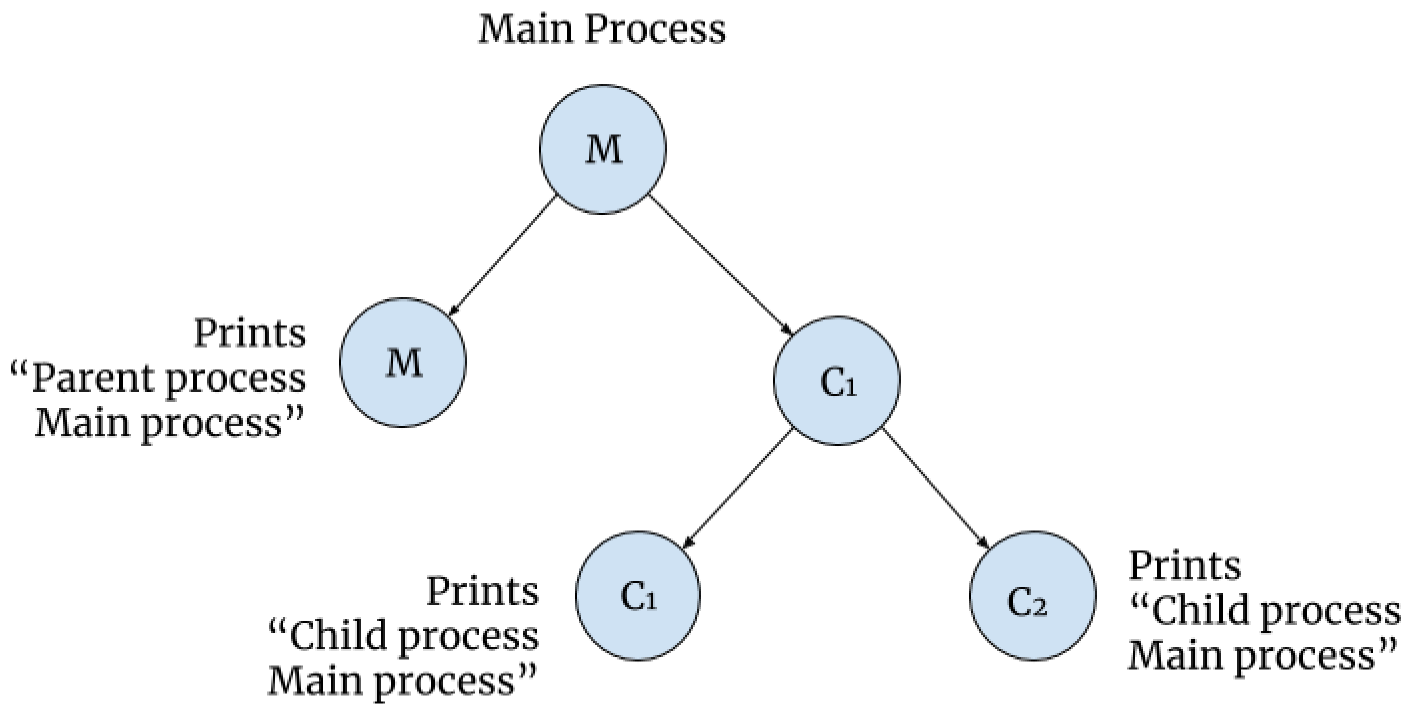
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 2
cc     2.c   -o 2
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./2
OS Course 2020
OS Course 2020
OS Course 2020
OS Course 2020
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$
```

C code:

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

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

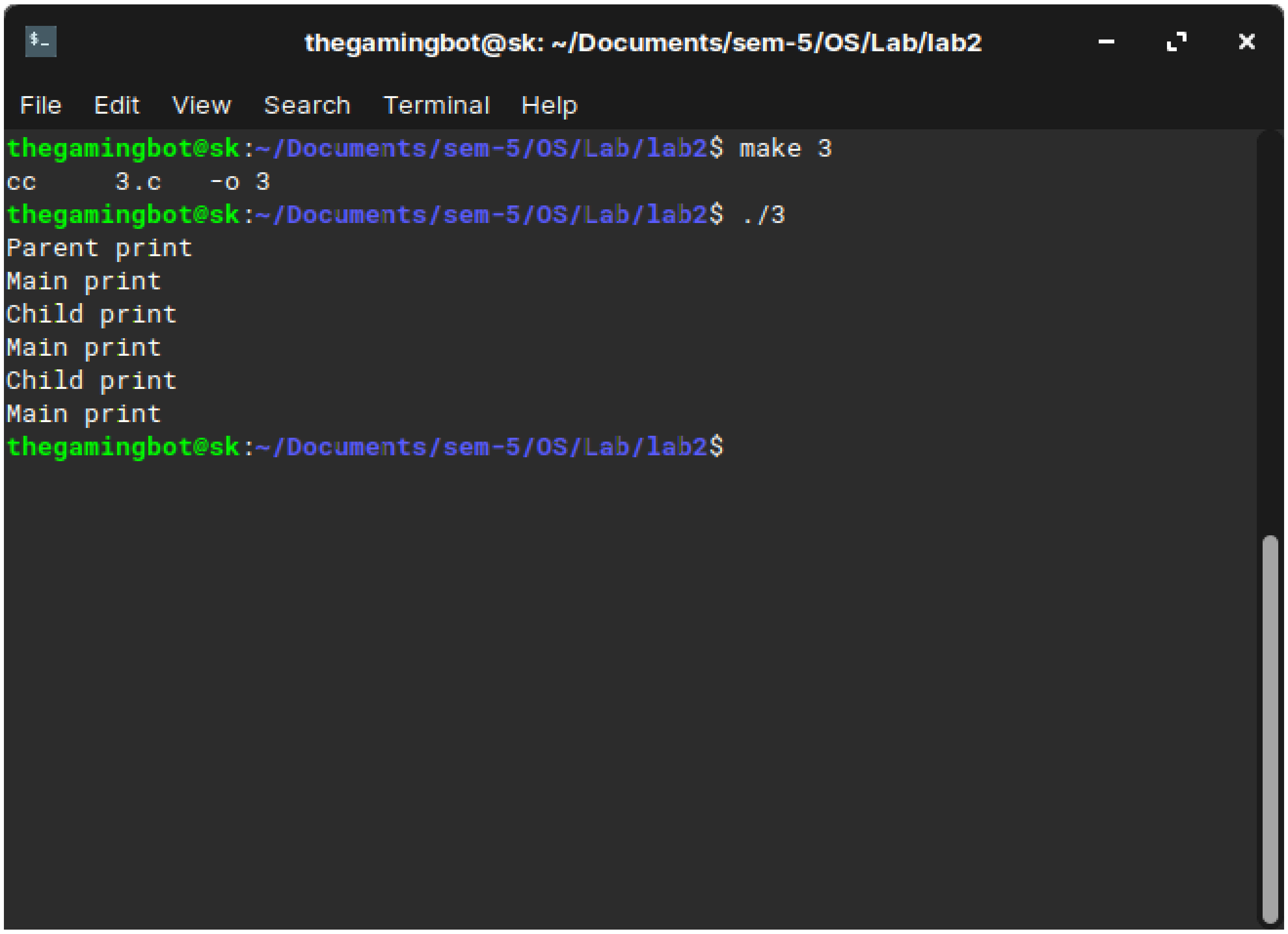
Binary Tree:


Main Process
M
M
Prints “Parent process Main process”
C1
C1
Prints “Child process Main process”
C2
Prints “Child process Main process”

Explanation:

When the main process(M) is forked, it creates a new child process(C_1), with pid = 0. The main process(M) has a pid > 0 as it is a parent. So, “Parent print” is printed by the main process(M) and the “Main print”. Now, the child process(C_1) executes the same program starting from the instruction after the fork call. Since the child process(C_1) has a pid = 0, it prints forks and creates a new child process(C_2) and prints “Child print” and “Main print”. The child process(C_2) prints “Child print” and “Main print”. The program then terminates.

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

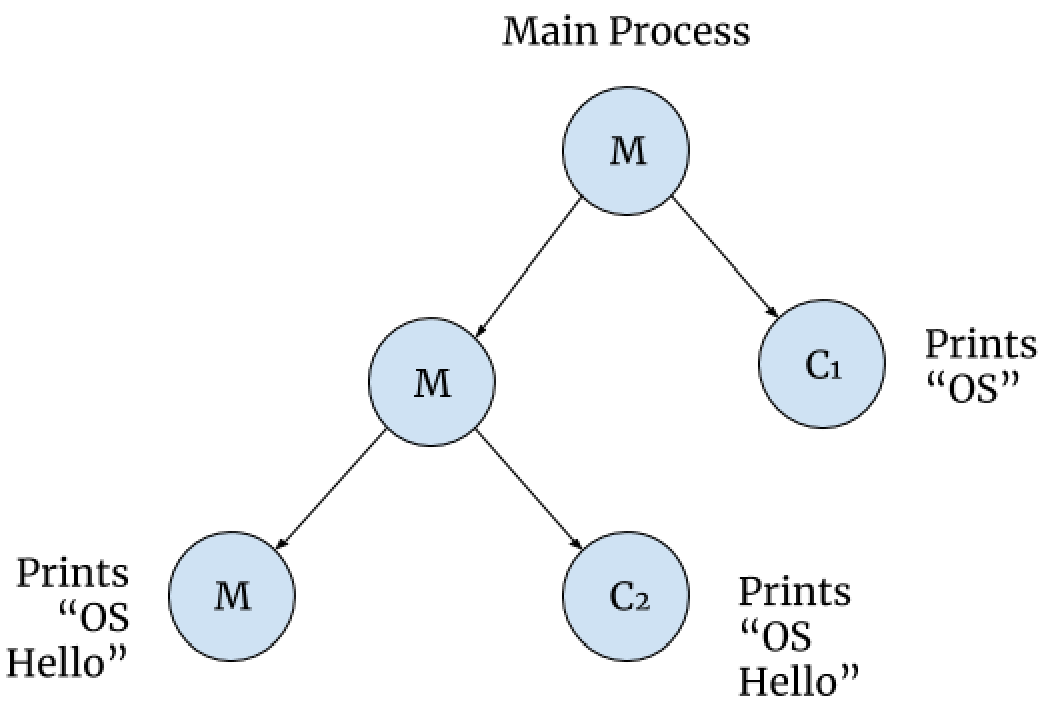
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 3
cc     3.c   -o 3
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./3
Parent print
Main print
Child print
Main print
Child print
Main print
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$
```

C code:

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

int main(){
   int pid = fork();
   if(pid > 0){
       fork();
       printf("OS\n");
   }
   printf("Hello\n");
   return 0;
}
```

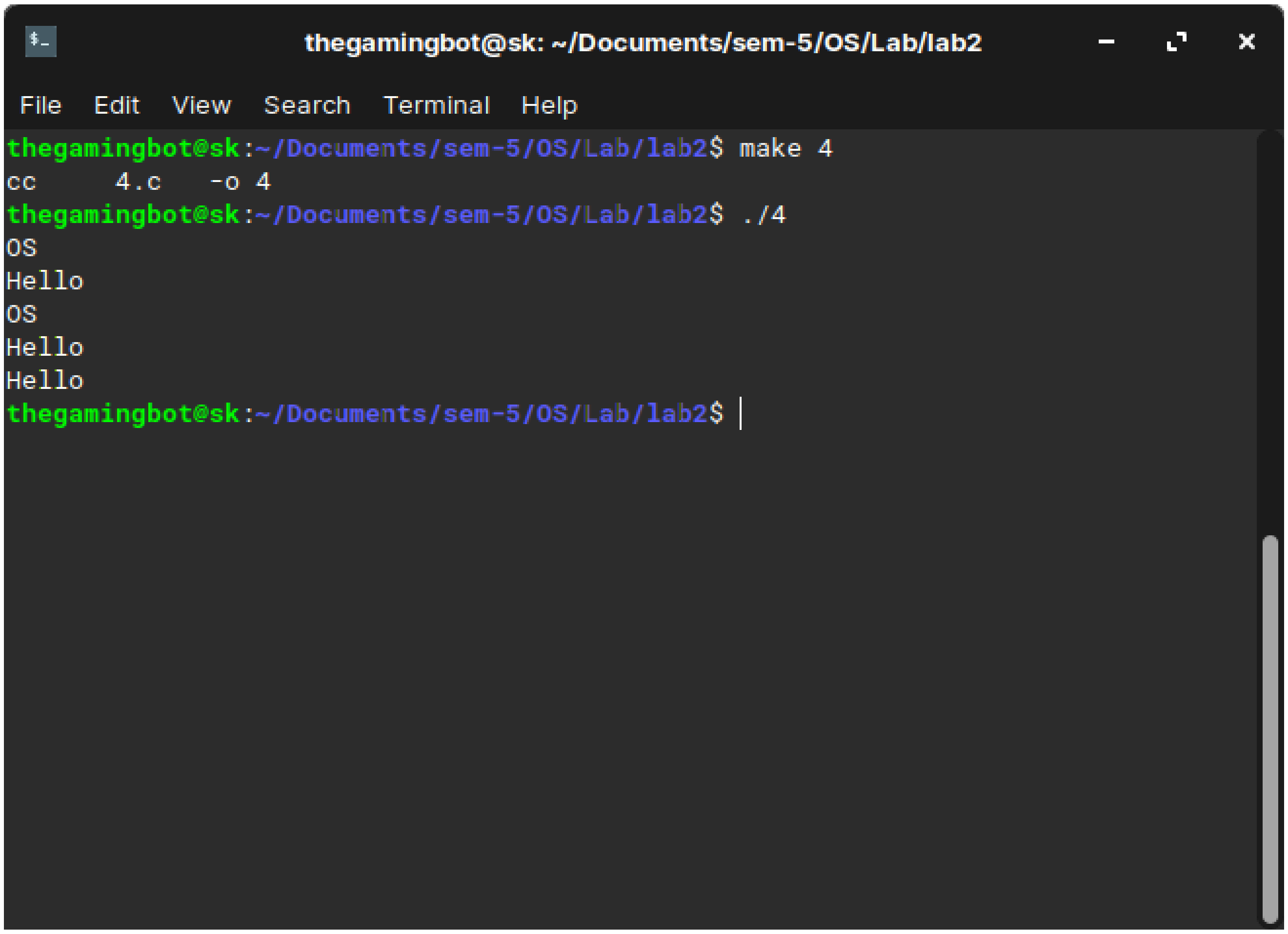
Binary Tree:



Explanation:

When the main process(M) is forked, it creates a new child process(C_1), with pid = 0. The main process(M) has a pid > 0 as it is a parent. So, the main process(M) is forked again creating another child process (C_2) and prints "OS " and "Hello". The child process(C_2) prints "OS" and "Hello". The child process(C_1) prints "Hello". The program terminates.

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

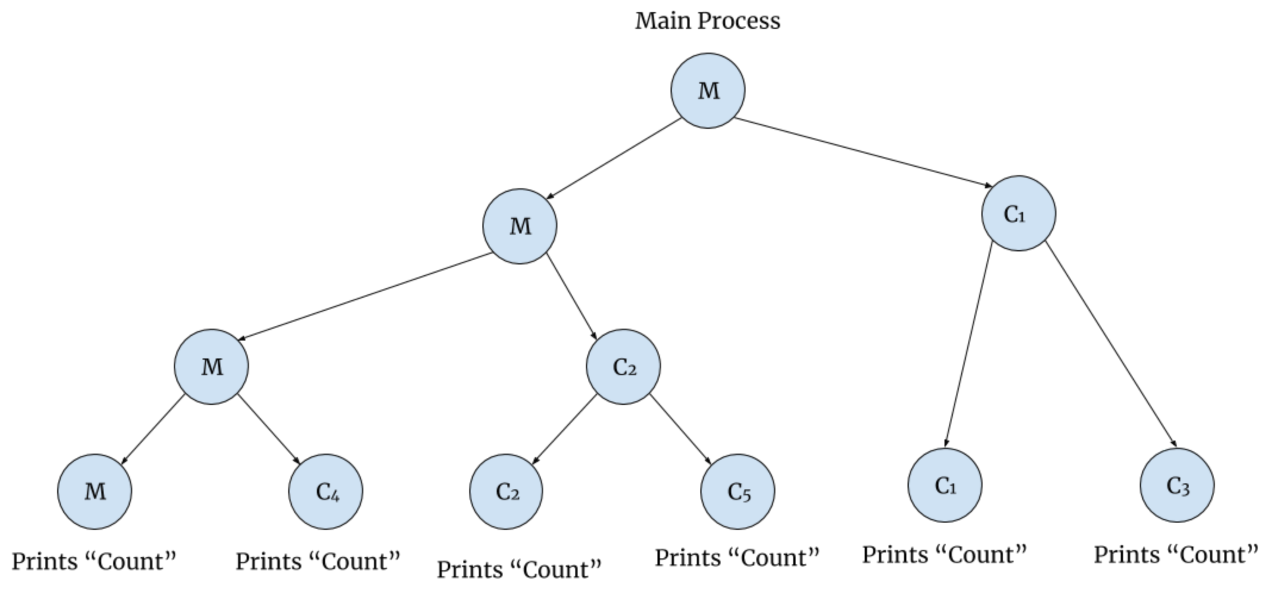
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 4
cc     4.c   -o 4
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./4
OS
Hello
OS
Hello
Hello
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ 
```

C code:

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

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

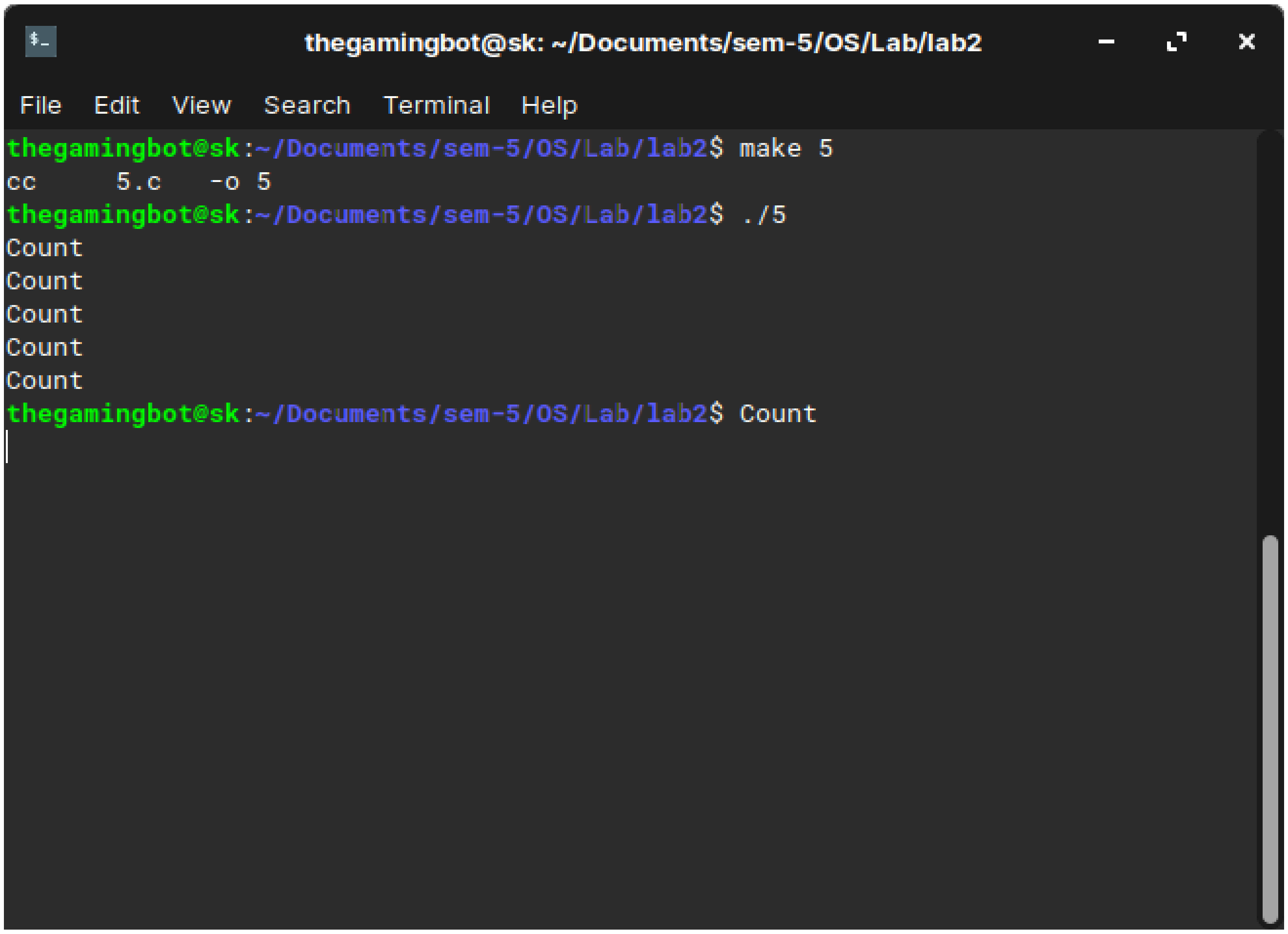
Binary Tree:



Explanation:

When the main process(M) is forked, it creates a new child process(C_1). The main process(M) is a parent, so it continues and gets forked, creating another child(C_2). The main process(M) forked again, creating a new child process(C_4). The child process(C_2) gets forked, creating a new child process(C_5). The child process(C_1) has pid=0, so it does not enter the if block. The child process(C_1) gets forked, giving another child process(C_3).Since there are 6 total processes created and there is a print statement at the end of the program, the print statement is displayed 6 times.

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

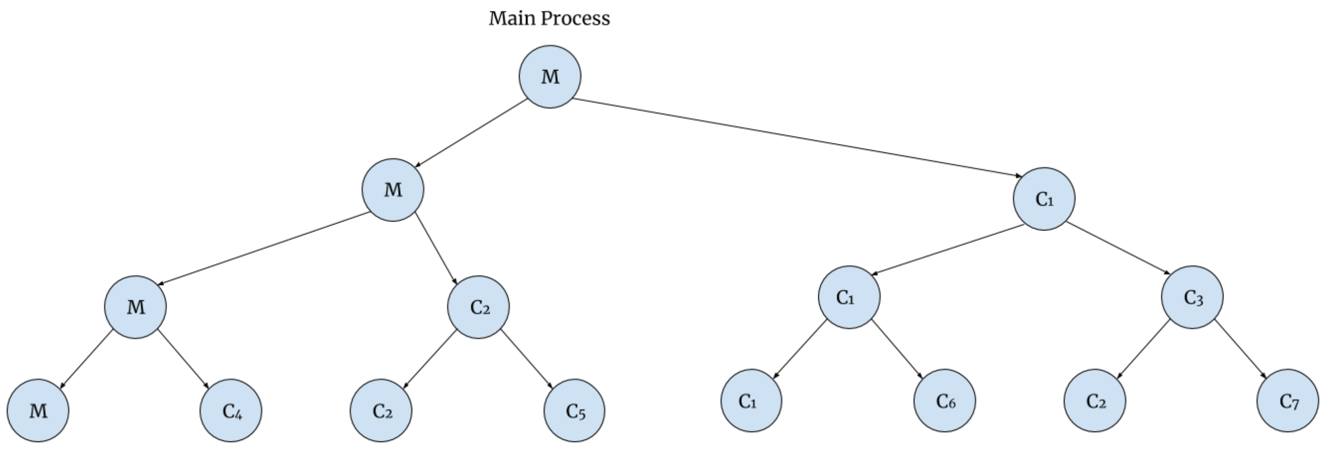
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 5
cc     5.c   -o 5
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./5
Count
Count
Count
Count
Count
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ Count
```

C code:

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

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

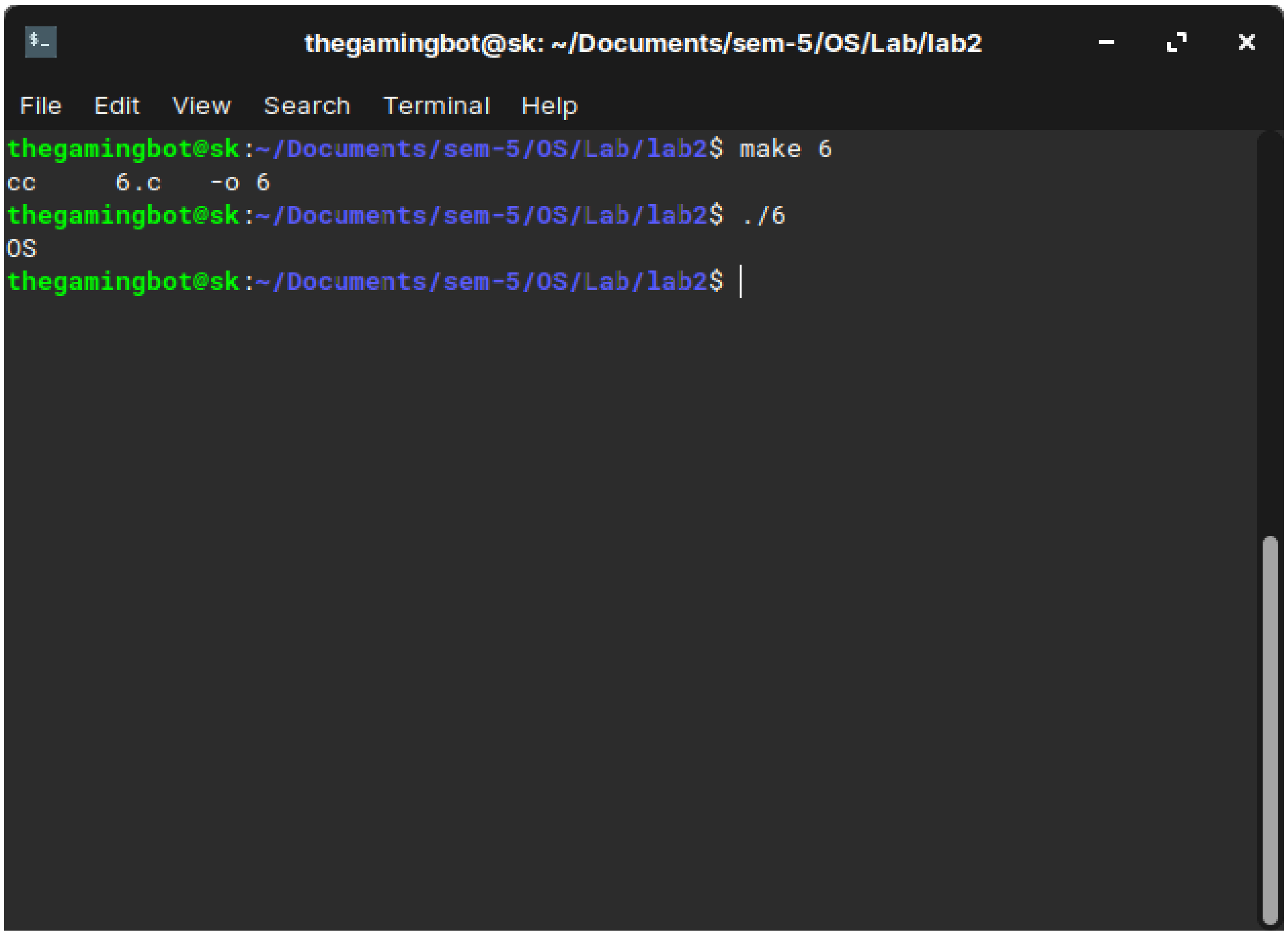
Binary Tree:



Explanation:

The print statement is displayed. When the main process(M) is forked, it creates a new child process(C_1). The main continues and gets forked creating another child process(C_2), and gets forked once more creating yet another child process(C_4). The child process(C_2) is forked, giving another child process(C_5). The child process(C_1) is forked giving a child process(C_3), and it is forked again giving yet another child process(C_6). The child process(C_3) is forked to give a child process(C_7).

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

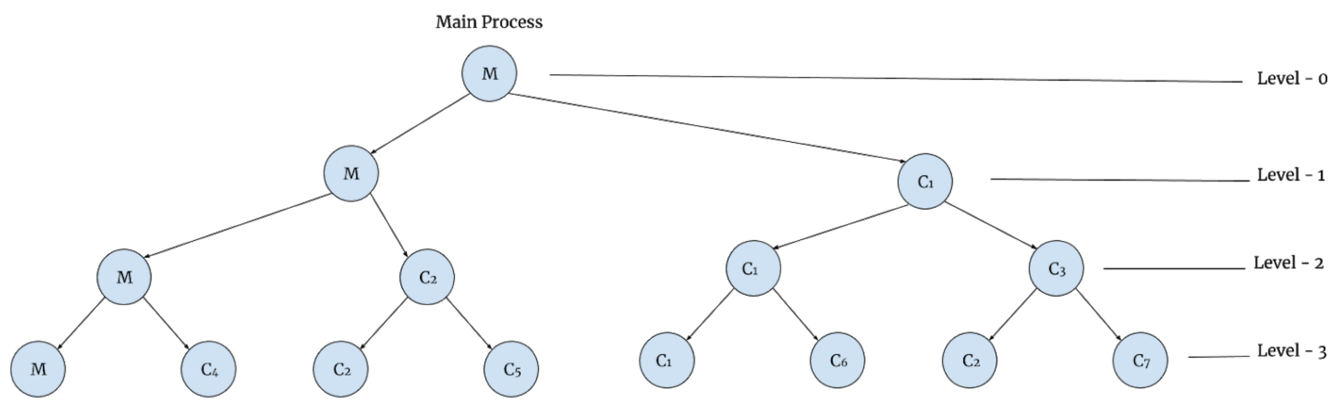
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 6
cc     6.c   -o 6
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./6
OS
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ 
```

C code:

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

Binary Tree:



Level - 0 prints “This will be printed ?.”

Level - 1 prints “This will be printed ?.”

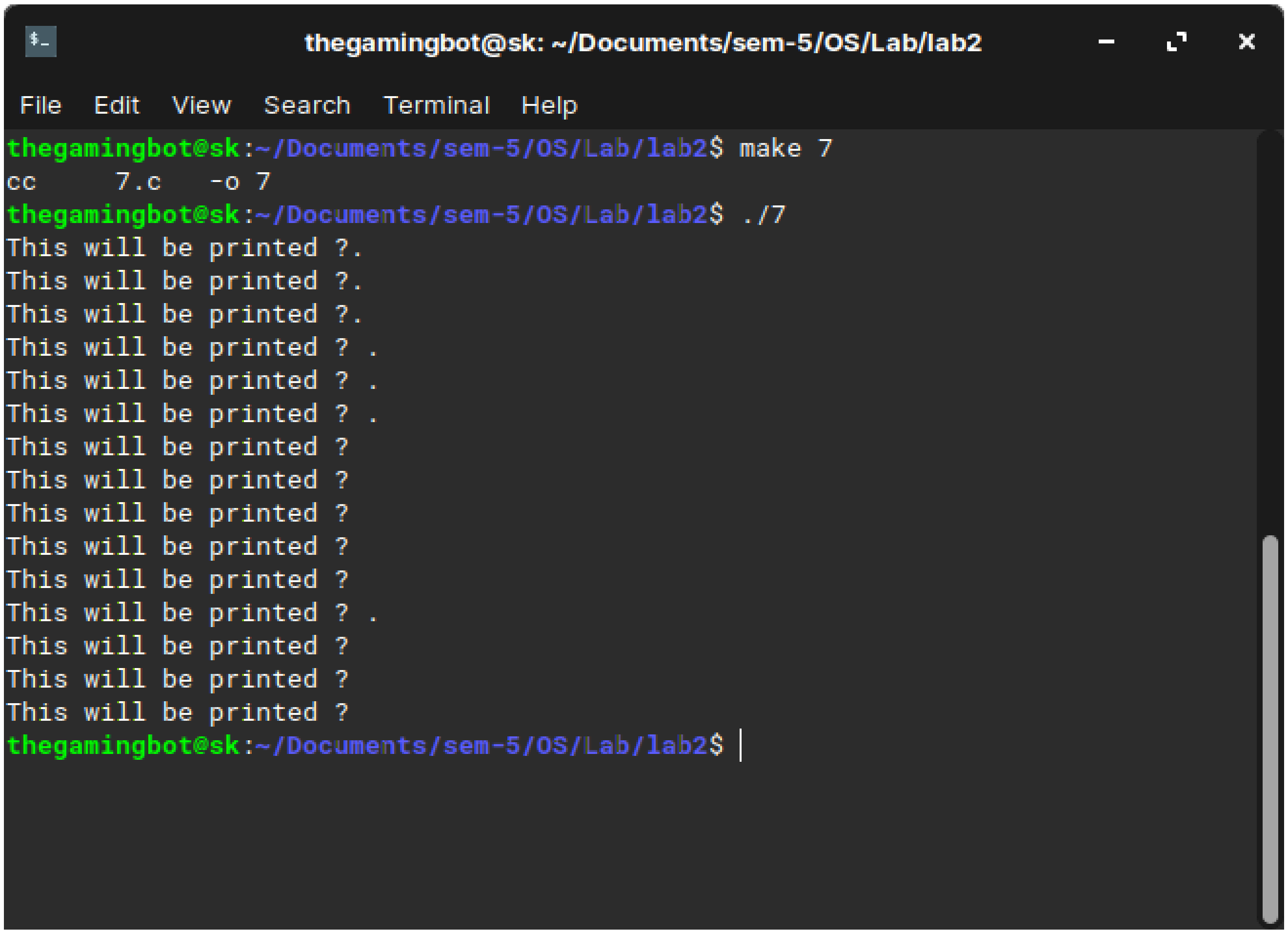
Level - 2 prints “This will be printed ? .”

Level - 3 prints “This will be printed ?”

Explanation:

When the main process(M) is forked, it creates a new child process(C_1). The main continues and gets forked creating another child process(C_2), and gets forked once more creating yet another child process(C_4). The child process(C_2) is forked, giving another child process(C_5). The child process(C_1) is forked giving a child process(C_3), and it is forked again giving yet another child process(C_6). The child process(C_3) is forked to give a child process(C_7). At each level, there are different print statements. So, after the fork at nth level, (n+1)th print statement is printed. The output order varies on which child process gets executed first, the kernel, the h/w resources available for the process.

Output:



thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

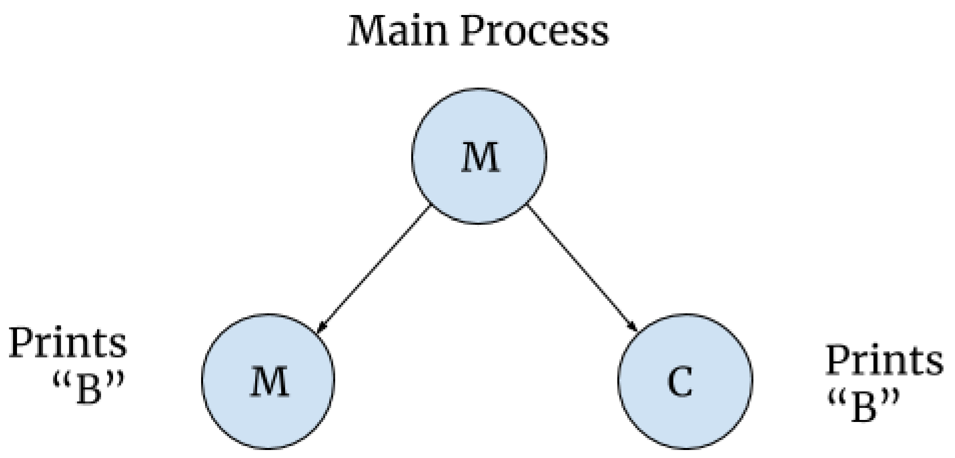
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 7
cc     7.c   -o 7
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./7
This will be printed ?.
This will be printed ?.
This will be printed ?.
This will be printed ? .
This will be printed ? .
This will be printed ? .
This will be printed ?
This 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 ?
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ 
```

C code:

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

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

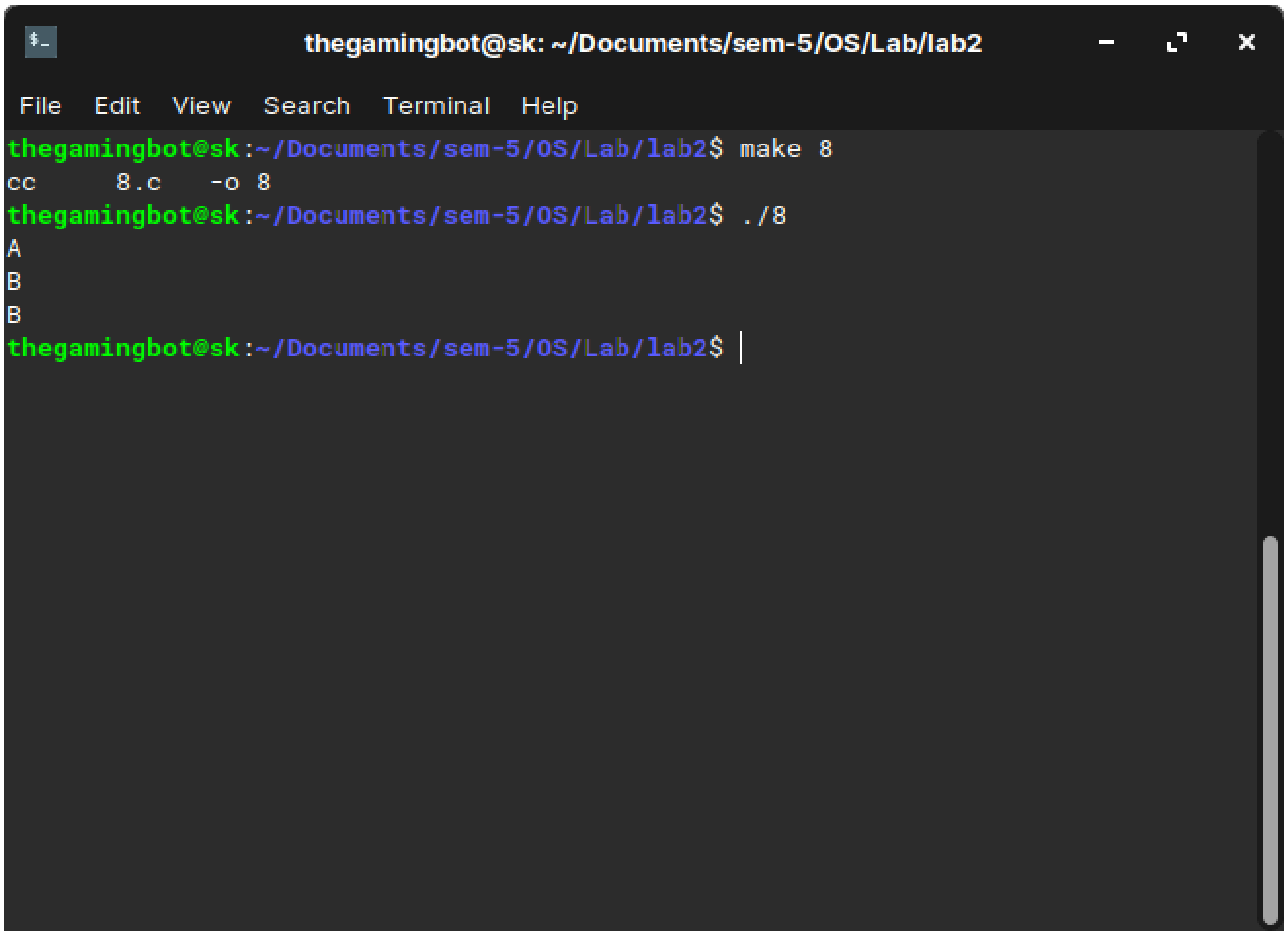
Binary Tree:



Explanation:

"A" is printed. When the main process(M) is forked, it creates a new child process(C_1). The main process(M) continues and prints "B". The child process(C_1) continues from the next line printing "B". The program terminates.

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab2
File Edit View Search Terminal Help

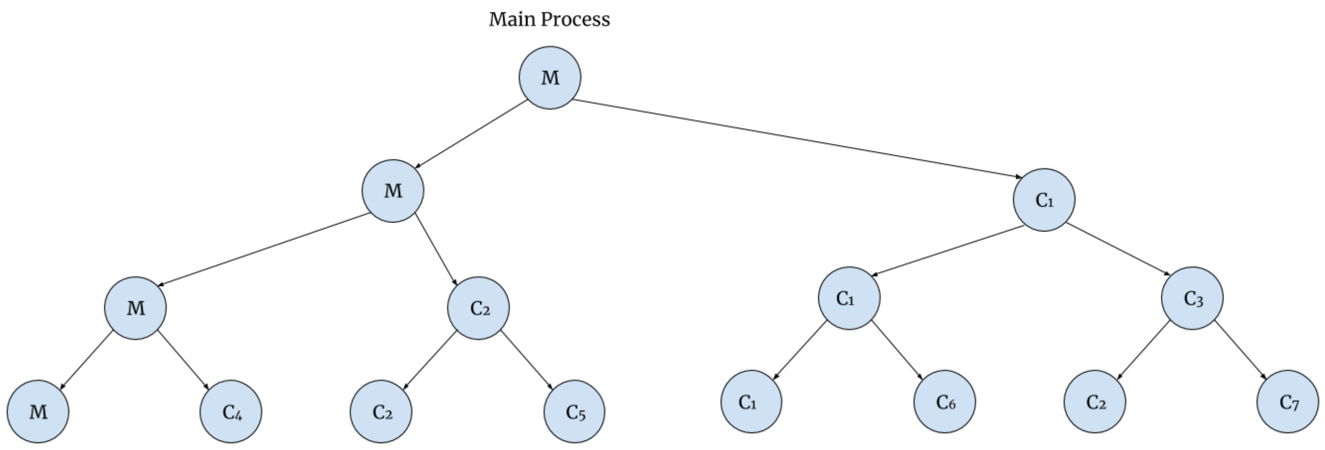
```
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ make 8
cc     8.c    -o 8
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$ ./8
A
B
B
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab2$
```

C code:

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

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

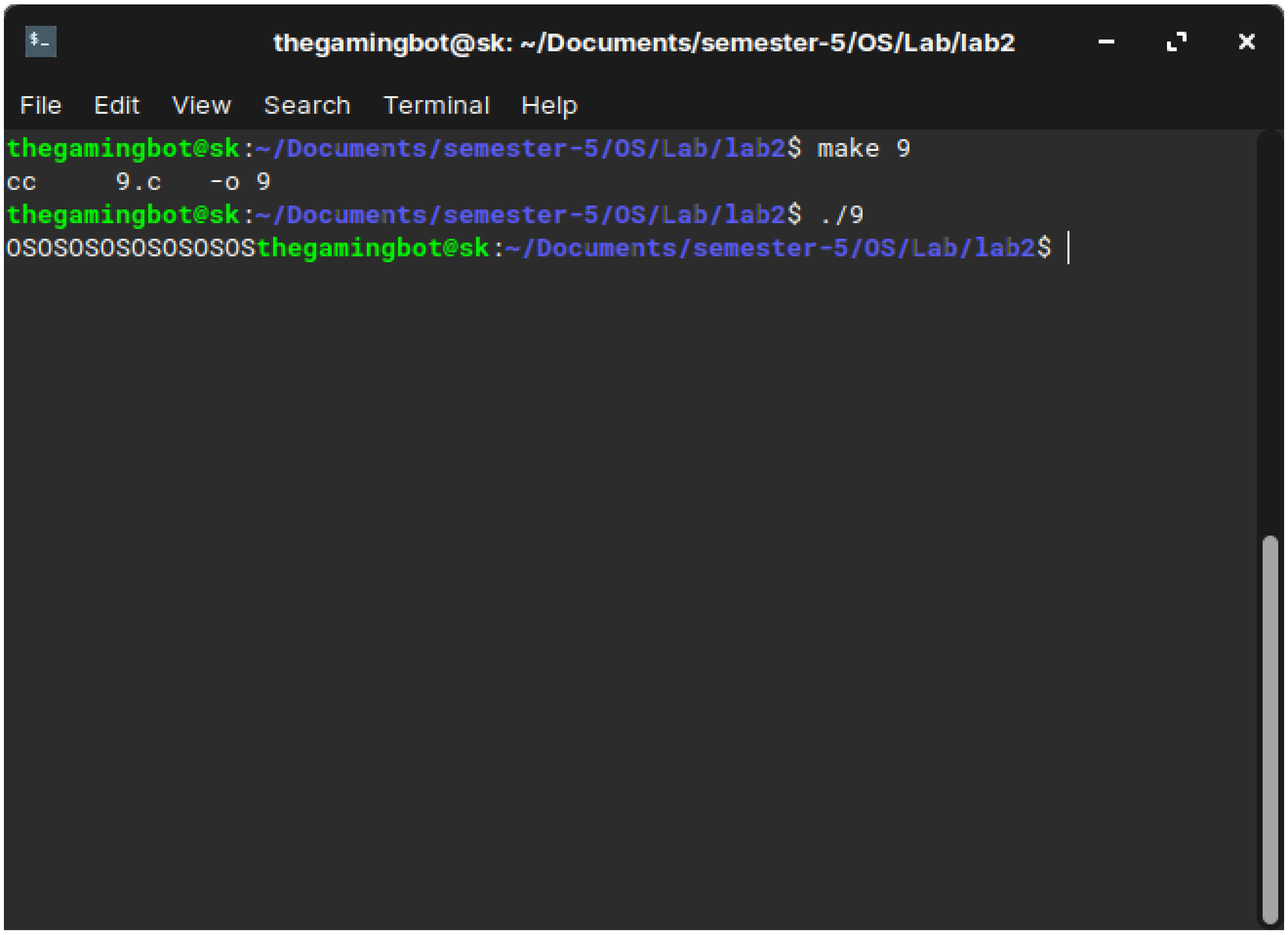
Binary Tree:



Explanation:

“OS” is printed. Since there is no newline character, the print string stored as a buffer is not cleared. When the main process(M) is forked, it creates a new child process(C1). The main continues and gets forked creating another child process(C2), and gets forked once more creating yet another child process(C4). The child process(C2) is forked, giving another child process(C5). The child process(C1) is forked giving a child process(C3), and it is forked again giving yet another child process(C6). The child process(C3) is forked to give a child process(C7). All the children inherit everything the parent has, including the buffers, variables. So the buffer is dumped to the output on termination. I observed that sometimes the program terminated before all the processes dumped their buffers. For the above program, the program sometimes printed OS 5 times befores the program terminated, and the rest 3 were printed after the path was printed.

Output:


thegamingbot@sk: ~/Documents/semester-5/OS/Lab/lab2
File Edit View Search Terminal Help

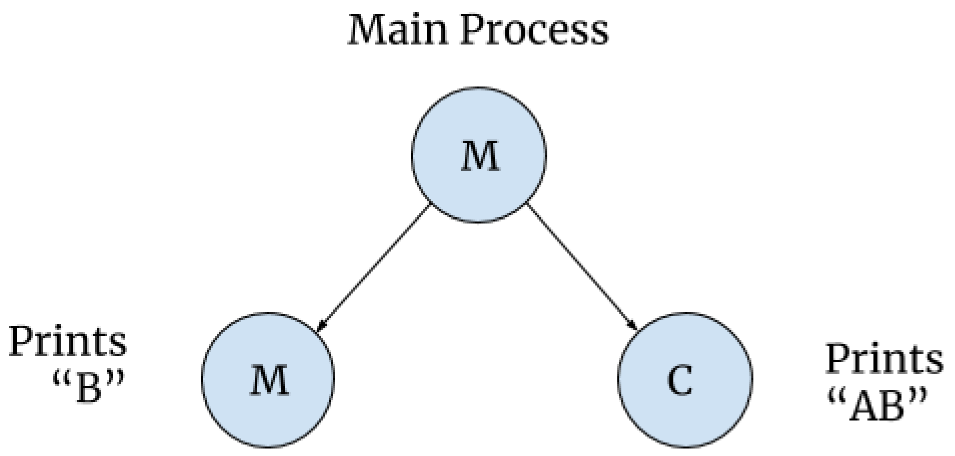
```
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ make 9
cc     9.c   -o 9
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ ./9
OSOSOSOSOSOSOSOSthegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ 
```

C code:

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

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

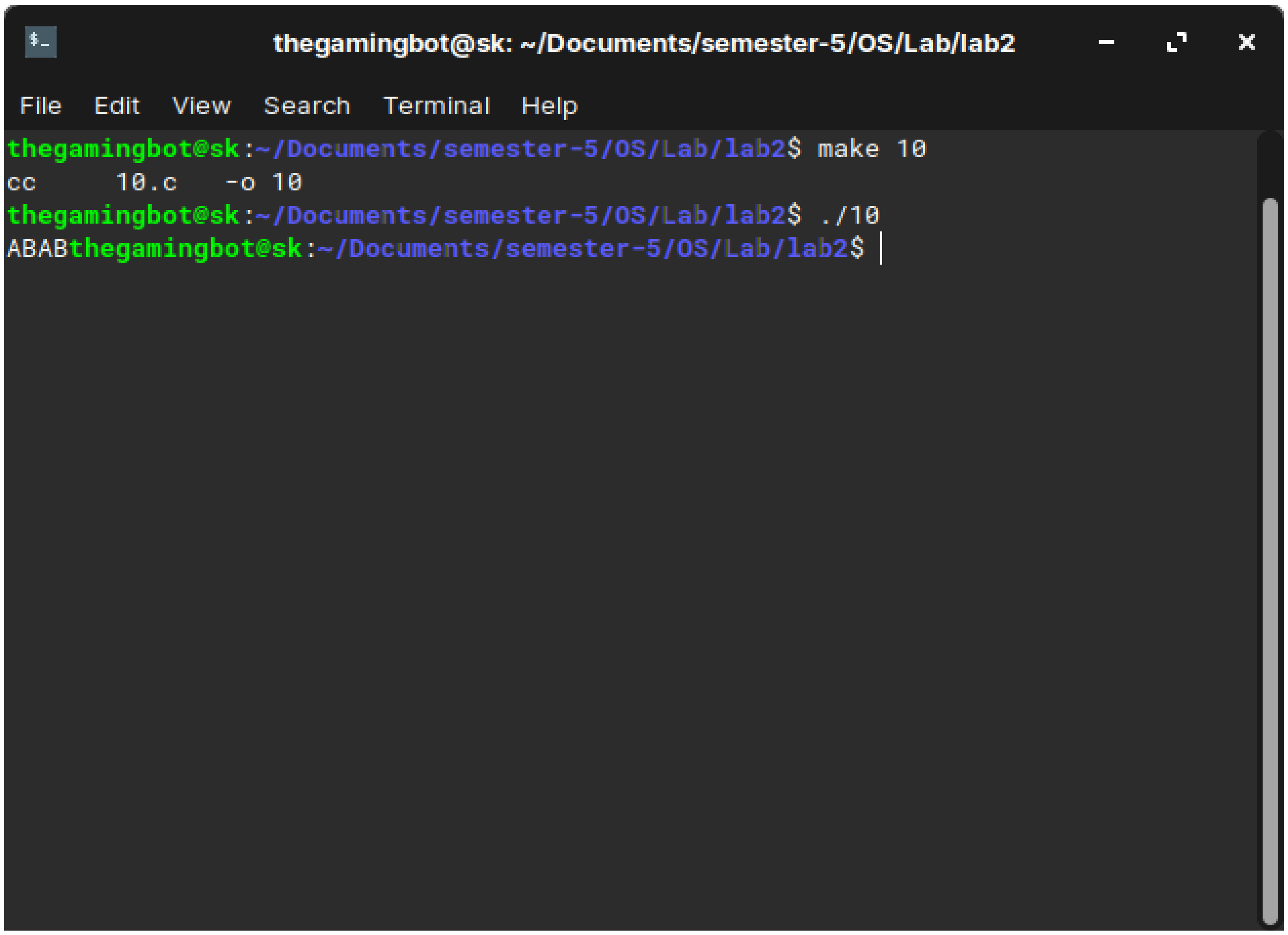
Binary Tree:



Explanation:

"A" is printed. Since there is no new line character, the output string that is buffered, is not cleared. When the main process(M) is forked, it creates a new child process(C). The main process(M) prints "B". Now the child process, inherits all the buffered content of the parent. So it dumps the buffered content and prints "B".

Output:


thegamingbot@sk: ~/Documents/semester-5/OS/Lab/lab2
File Edit View Search Terminal Help

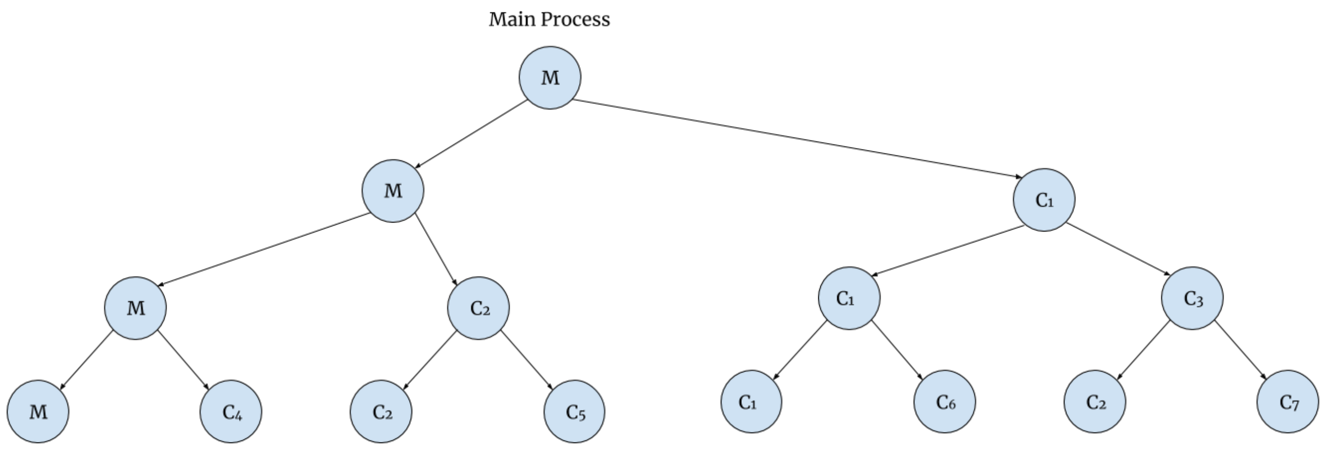
```
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ make 10
cc     10.c   -o 10
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ ./10
ABABthegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ 
```

C code:

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

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

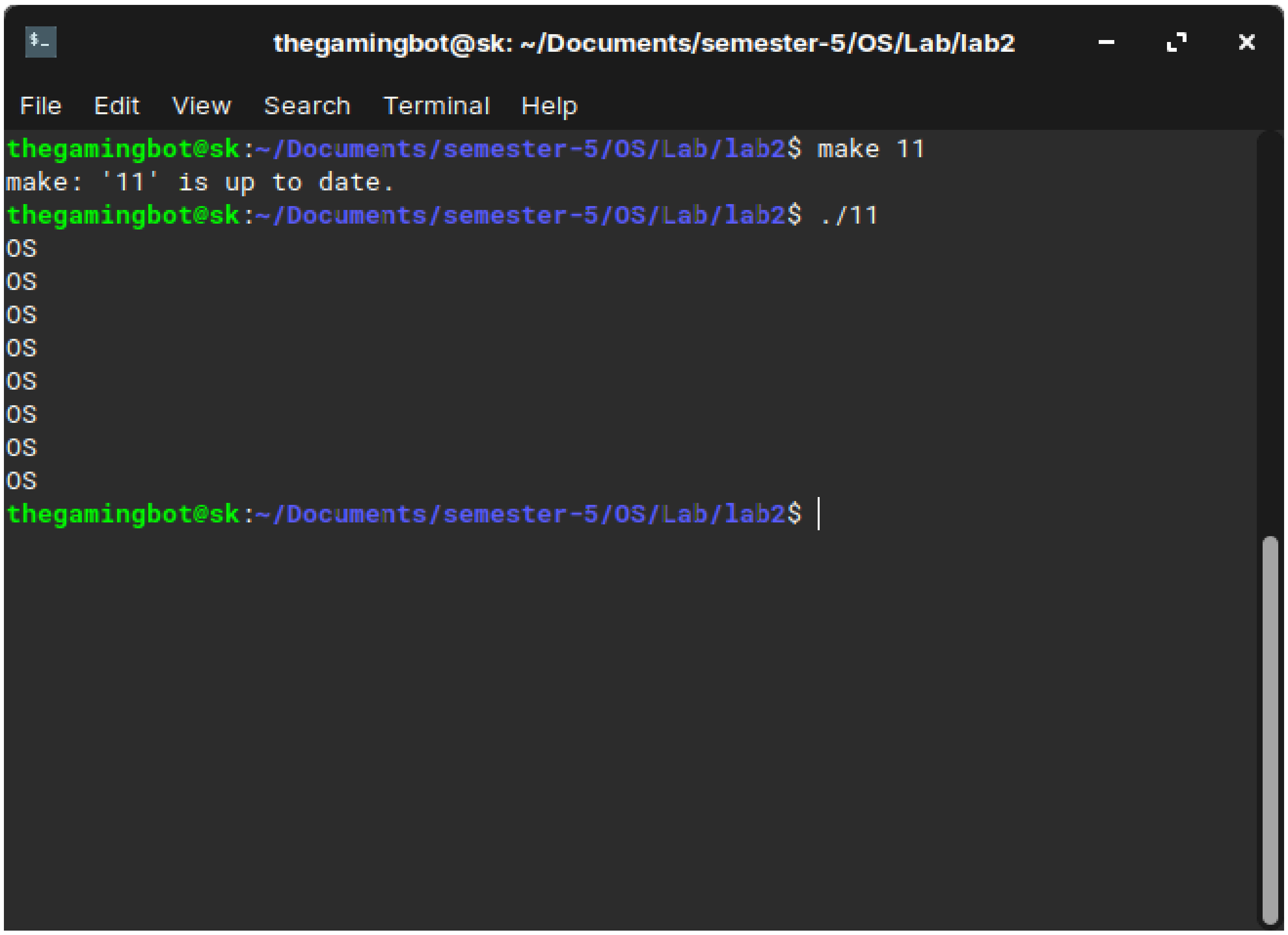
Binary Tree:



Explanation:

When the main process(M) is forked, it creates a new child process(C_1). The main continues and gets forked creating another child process(C_2), and gets forked once more creating yet another child process(C_4). The child process(C_2) is forked, giving another child process(C_5). The child process(C_1) is forked giving a child process(C_3), and it is forked again giving yet another child process(C_6). The child process(C_3) is forked to give a child process(C_7). Since there is a print statement at the end of main(), and there are 8 processes in total created. So "OS" is dumped to the display 8 times.

Output:



thegamingbot@sk: ~/Documents/semester-5/OS/Lab/lab2
File Edit View Search Terminal Help

```
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ make 11
make: '11' is up to date.
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ ./11
OS
OS
OS
OS
OS
OS
OS
OS
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ 
```

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

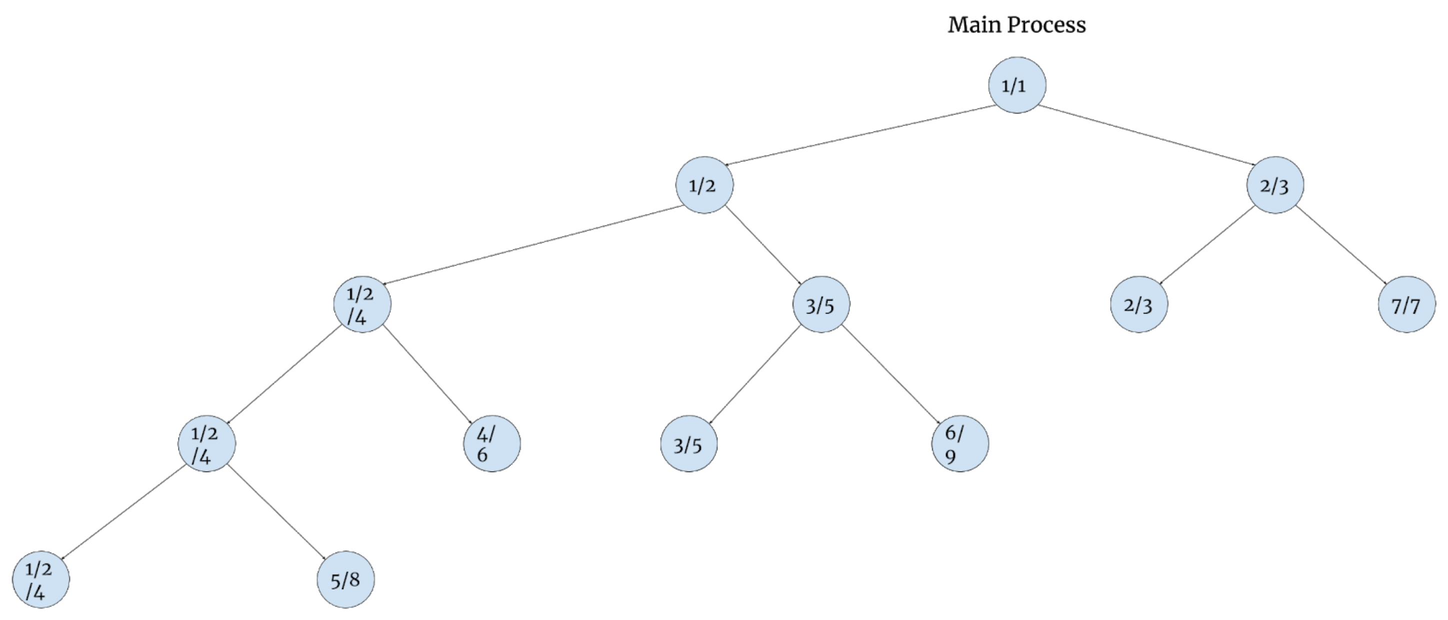
- **1 forks 2 and 3**
- **2 forks 4 5 and 6**
- **3 forks 7**
- **4 forks 8**
- **5 forks 9**

C code:

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

int main(){
   int x = fork();          // Creates process 2
   if(x > 0){
       int y = fork();      // Creates process 3
       if(y > 0){
           int z = fork();  // Creates process 4
           if(z > 0)
               fork();      // Creates process 5
       }
       else if(y == 0){
           fork();          // Creates process 6
       }
   }
   else if(x == 0){
       fork();              // Creates process 7
   }
   printf("Hello\n");
}
```

Binary Tree:



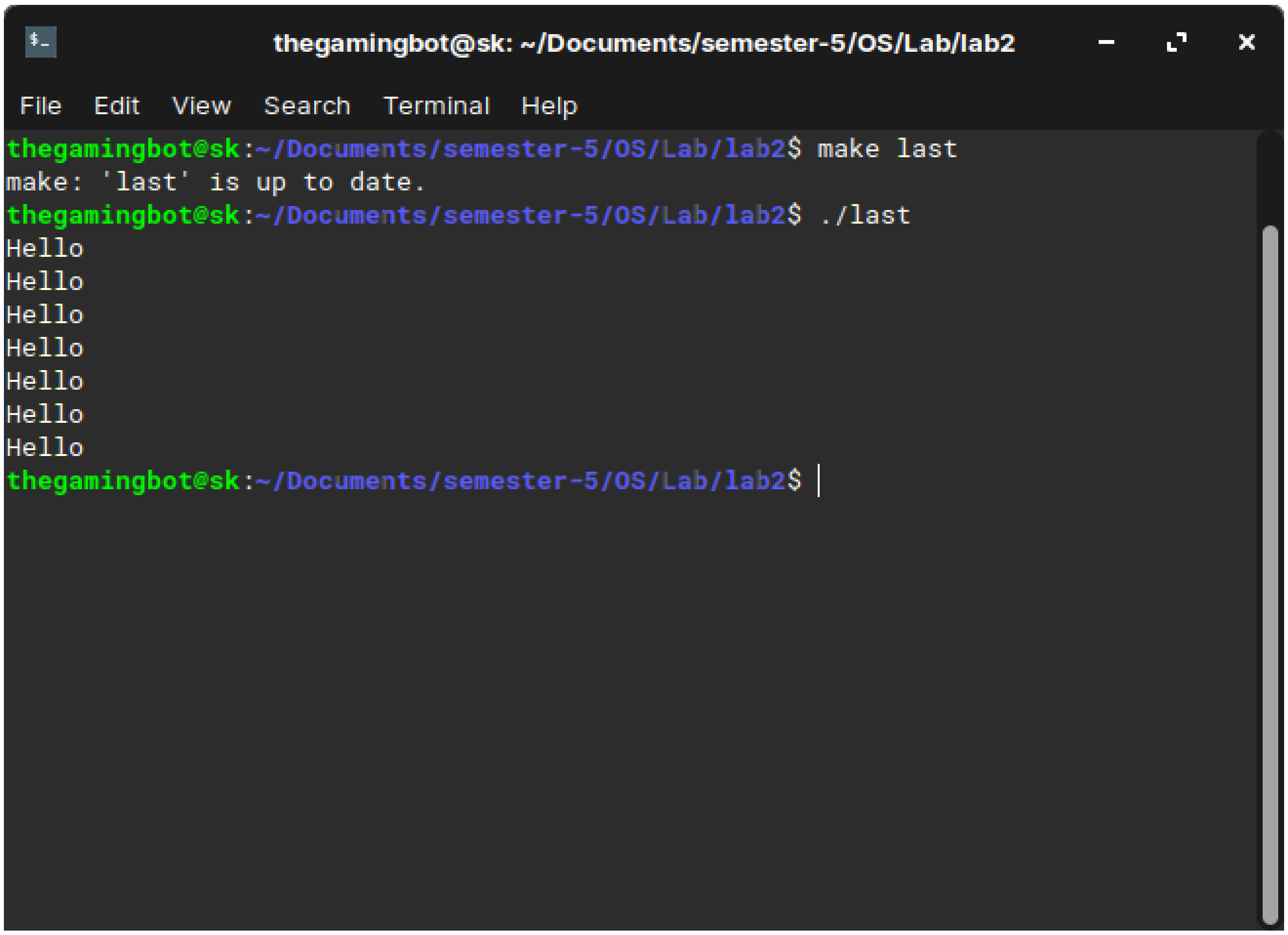
Explanation:

My naming convention:
1 == 1
2 == 1
3 == 2
4 == 1
5 == 3
6 == 4
7 == 7
8 == 5
9 == 6

1 forks 2 and 3 => 1 forks 1 and 2
2 forks 4, 5 and 6 => 2 forks 2, 5 and 6 => 1 forks 1, 3 and 4
3 forks 7 => 2 forks 2 and 7
4 forks 8 => 2 forks 2 and 8 => 1 forks 1 and 5
5 forks 9 => 3 forks 3 and 6

There is a print statement at the end of main, and 7 processes are created throughout the program execution. So "Hello" is printed 7 times.

Output:


thegamingbot@sk: ~/Documents/semester-5/OS/Lab/lab2
File Edit View Search Terminal Help

```
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ make last
make: 'last' is up to date.
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ ./last
Hello
Hello
Hello
Hello
Hello
Hello
Hello
thegamingbot@sk:~/Documents/semester-5/OS/Lab/lab2$ 
```