Computer Network

Assignment : 2

List of Topics: Inter-Process Communication(PIPE), Fork()

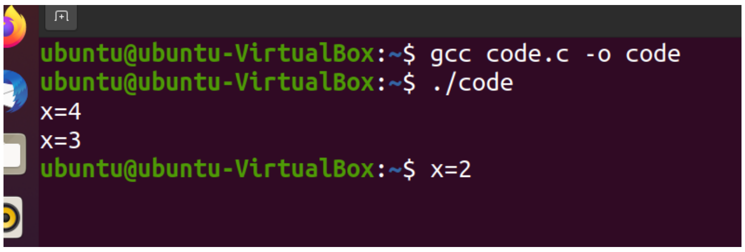
Group No. : 22

Urja Gandhi	202112039
Sharvi Gabani	202112066
Sagar Variya	202112114

1. Enumerate all possible outputs of the following program.

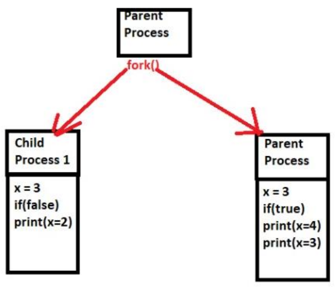
```
int main() {
   int x = 3;
   if (fork() != 0)
      printf("x=%d\n", ++x);
   printf("x=%d\n", --x);
   exit(0);
}
```

Output :



```
ubuntu@ubuntu-VirtualBox:~$ gcc code.c -o code
ubuntu@ubuntu-VirtualBox:~$ ./code
x=4
x=3
ubuntu@ubuntu-VirtualBox:~$ x=2
```

Explanation :



After calling fork(), we get a child process c1.

Fork returns 0, if we are returned to newly created child process or a positive value .i.e. child process id.

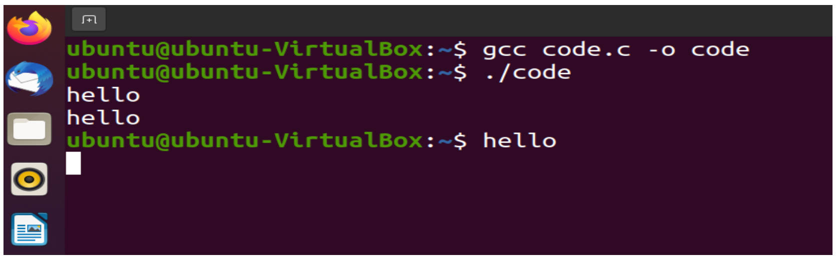
Here, x will be incremented and printed only if we are in parent process. If child process is in execution, the condition won't be true and x will not be incremented. Thus, x (3) is decremented in child process execution.

While in parent process, x is first incremented and then decremented.

2. How many "hello" output lines does this program print?

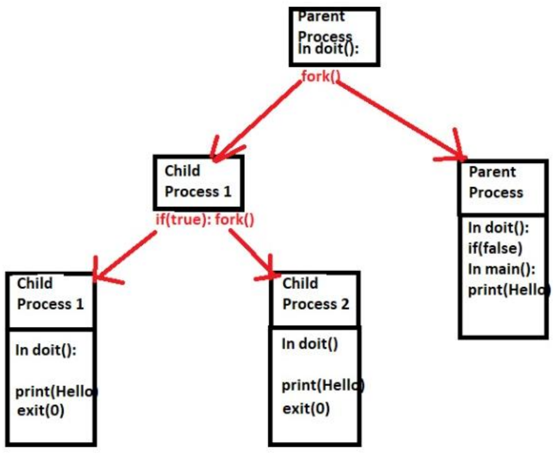
```
void doit() {
    if (fork() == 0) {
       fork();
       printf("hello\n");
       exit(0);
    }
  return;
}
int main() {
    doit();
    printf("hello\n");
    exit(0);
}
```

Output :



```
ubuntu@ubuntu-VirtualBox:~$ gcc code.c -o code
ubuntu@ubuntu-VirtualBox:~$ ./code
hello
hello
ubuntu@ubuntu-VirtualBox:~$ hello
```

Explanation :


Parent Process In doit():
fork()
Child Process 1
if(true): fork()
Parent Process
In doit(): if(false) In main(): print(Hello)
Child Process 1
In doit(): print(Hello) exit(0)
Child Process 2
In doit() print(Hello) exit(0)

Here, parent process executes doit() function.

fork() system call creates a child process c1. If condition will be satisfied at the time of execution of c1.

And again fork() is called creating another child process c2. Both the child processes c1 and c2 will print hello from doit() function and exit the code.

While parent returns to main() function and there it prints hello and exit the code.

1. Using the fork, you need to create 3 children of a parent. And by use of the IPC mechanism, you need to set up communication between them.

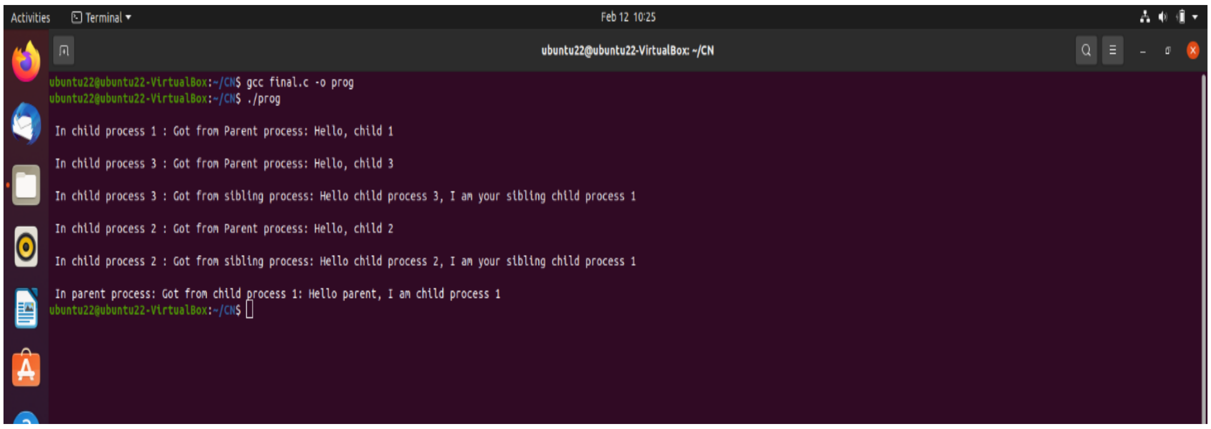
Show your communication set up by the following scenario by sending echo packets:

> **a. Parent to it's all child**
> **b. Any one child to its parent**
> **c. Any one child to it's all siblings**

ques1.c :

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

int main()
{
    int fd[6][2], i;

    // opening 6 pipes
    for(int i=0; i<6; i++)
    {
      /* pipe() returns : 0 on Success
                         -1 on error
         fd[i][0]: fd(file descriptor) for the read end of pipe
         fd[i][1]: fd for the write end of pipe
      */
      if(pipe(fd[i]) < 0)
      {
            printf("\n An error occurred while opening the pipe \n");
            exit(1);
      }
    }

      // creating child process 1
      int id1 = fork();

      //creation of a child process was unsuccessful
      if(id1 < 0)
      {
      printf("\n An error occurred while creating child process 1\n");
        exit(1);
      }


      if(id1 == 0) //In child 1
      {
      //closing all the file descriptors which are not used by child
process 1
      close(fd[0][1]);
      close(fd[1][0]);
      close(fd[1][1]);
      close(fd[2][0]);
      close(fd[2][1]);
      close(fd[3][0]);
      close(fd[4][0]);
      close(fd[5][0]);

      char * recv;
      // reading message from fd[0]
      if(read(fd[0][0], &recv, sizeof(recv)) < 0)
      {
            printf("\n An error occurred while reading from pipe in child
process 1 \n");
            exit(1);
      }
      printf("\n In child process 1 : Got from Parent process: %s \n",
recv);

      //(b) -> Child to parent
      char * send_msg1 = "Hello parent, I am child process 1";
      // writing message to fd[3]
      if(write(fd[3][1], &send_msg1, sizeof(send_msg1)) < 0)
      {
            printf("\n An error occurred while writing in pipe from child
process 1 \n");
            exit(1);
      }

      //(c) -> Child to siblings
      char * send_msg2 = "Hello child process 2, I am your sibling child
process 1";
      if(write(fd[4][1], &send_msg2, sizeof(send_msg2)) < 0)
      {
            printf("\n An error occurred while writing in pipe from child
process 1 \n");
            exit(1);
      }
      char * send_msg3 = "Hello child process 3, I am your sibling child
process 1";
      if(write(fd[5][1], &send_msg3, sizeof(send_msg3)) < 0)
      {
            printf("\n An error occurred while writing in pipe from child
process 1 \n");
            exit(1);
      }

      close(fd[0][0]);
      close(fd[3][1]);
      close(fd[4][1]);
      close(fd[5][1]);
     }
     else //in parent
     {
      int id2 = fork();
      if(id2 < 0)
       {
            printf("\n An error occurred while creating child process 2 
\n");
            exit(1);
      }


      if(id2 == 0) //In child 2
      {
            close(fd[0][0]);
            close(fd[0][1]);
            close(fd[1][1]);
            close(fd[2][0]);
            close(fd[2][1]);
            close(fd[3][0]);
            close(fd[3][1]);
            close(fd[4][1]);
            close(fd[5][0]);
            close(fd[5][1]);

            char * recv1;
            if(read(fd[1][0], &recv1, sizeof(recv1)) < 0)
            {
                  printf("\n An error occurred while reading from pipe in 
child process 2 \n");
                  exit(1);
            }
            printf("\n In child process 2 : Got from Parent process: %s 
\n", recv1);

            char *recv2;
            if(read(fd[4][0], &recv2, sizeof(recv2)) < 0)
            {
                  printf("\n An error occurred while reading from pipe in 
child process 2 \n");
                  exit(1);
            }
            printf("\n In child process 2 : Got from sibling process: %s 
\n", recv2);

            close(fd[1][0]);
            close(fd[4][0]);
      }
      else //again in parent
      {
            int id3 = fork();
            if(id3 < 0)
            {
                  printf("\n An error occurred while creating child 
process 3 \n");
                  exit(1);
            }


            if(id3 == 0) //In child 3
            {
                  close(fd[0][0]);
                  close(fd[0][1]);
                  close(fd[1][0]);
                  close(fd[1][1]);
                  close(fd[2][1]);
                  close(fd[3][0]);
                  close(fd[3][1]);
                  close(fd[4][0]);
                  close(fd[4][1]);
                  close(fd[5][1]);

                  char * recv1;
                  if(read(fd[2][0], &recv1, sizeof(recv1)) < 0)
                  {
                        printf("\n An error occurred while reading from 
pipe in child process 3 \n");
                        exit(1);
                  }
                  printf("\n In child process 3 : Got from Parent process: 
%s \n", recv1);

                  char *recv2;
                  if(read(fd[5][0], &recv2, sizeof(recv2)) < 0)
                  {
                        printf("\n An error occurred while reading from 
pipe in child process 2 \n");
                        exit(1);
                  }
                  printf("\n In child process 3 : Got from sibling 
process: %s \n", recv2);

                  close(fd[2][0]);
                  close(fd[5][0]);
            }
            else{

                  //parent process code
                  close(fd[0][0]);
                  close(fd[1][0]);
                  close(fd[2][0]);
                  close(fd[3][1]);
                  close(fd[4][0]);
                  close(fd[4][1]);
                  close(fd[5][0]);
                  close(fd[5][1]);

                  char * msg1 = "Hello, child 1";

                  if(write(fd[0][1], &msg1, sizeof(msg1)) < 0)
                  {
                        printf("\n An error occurred while writing in pipe
from parent process for child process 1\n");
                        exit(1);
                  }

                  char * msg2 = "Hello, child 2";
                  if(write(fd[1][1], &msg2, sizeof(msg2)) < 0)
                  {
                        printf("\n An error occurred while writing in pipe
from parent process for child process 2\n");
                        exit(1);
                  }

                  char * msg3 = "Hello, child 3";
                  if(write(fd[2][1], &msg3, sizeof(msg3)) < 0)
                  {
                        printf("\n An error occurred while writing in pipe
from parent process for child process 3\n");
                        exit(1);
                  }

                  char * recv_msg;
                  if(read(fd[3][0], &recv_msg, sizeof(recv_msg)) < 0)
                  {
                        printf("\n An error occurred while reading from
pipe in parent process from child process 1\n");
                        exit(1);
                  }

                  printf("\n In parent process: Got from child process 1:
%s \n", recv_msg);

                  close(fd[0][1]);
                  close(fd[1][1]);
                  close(fd[2][1]);
                  close(fd[3][0]);
            }
      }
     }

    return 0;
}
```

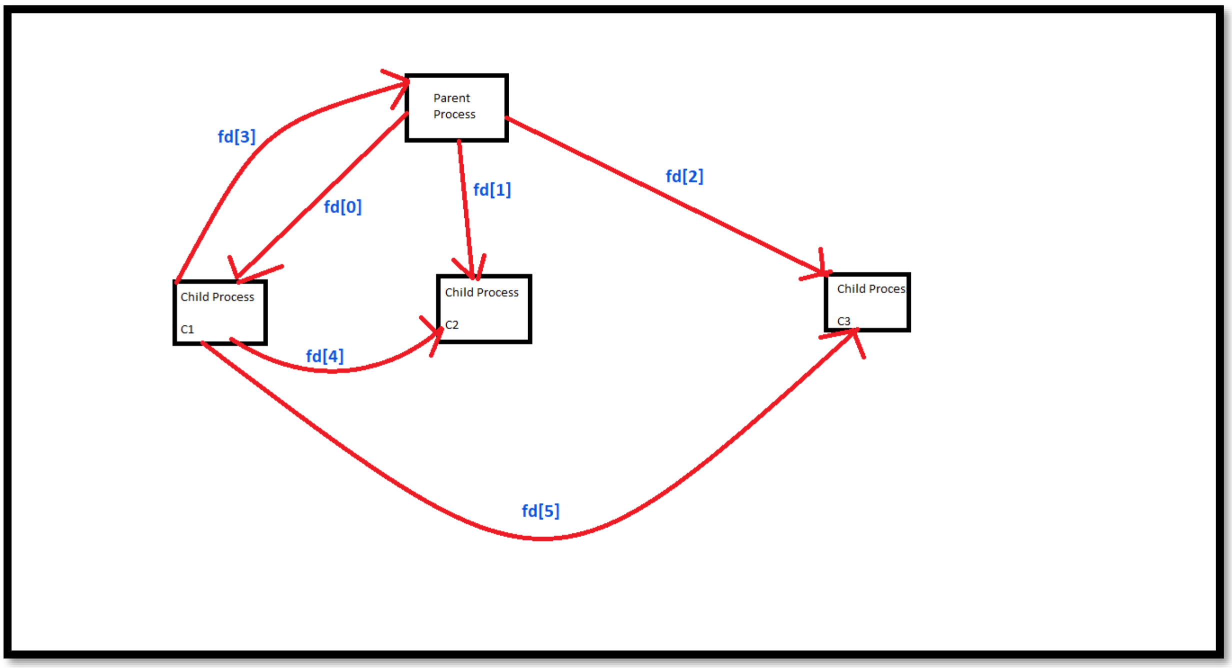
Output :

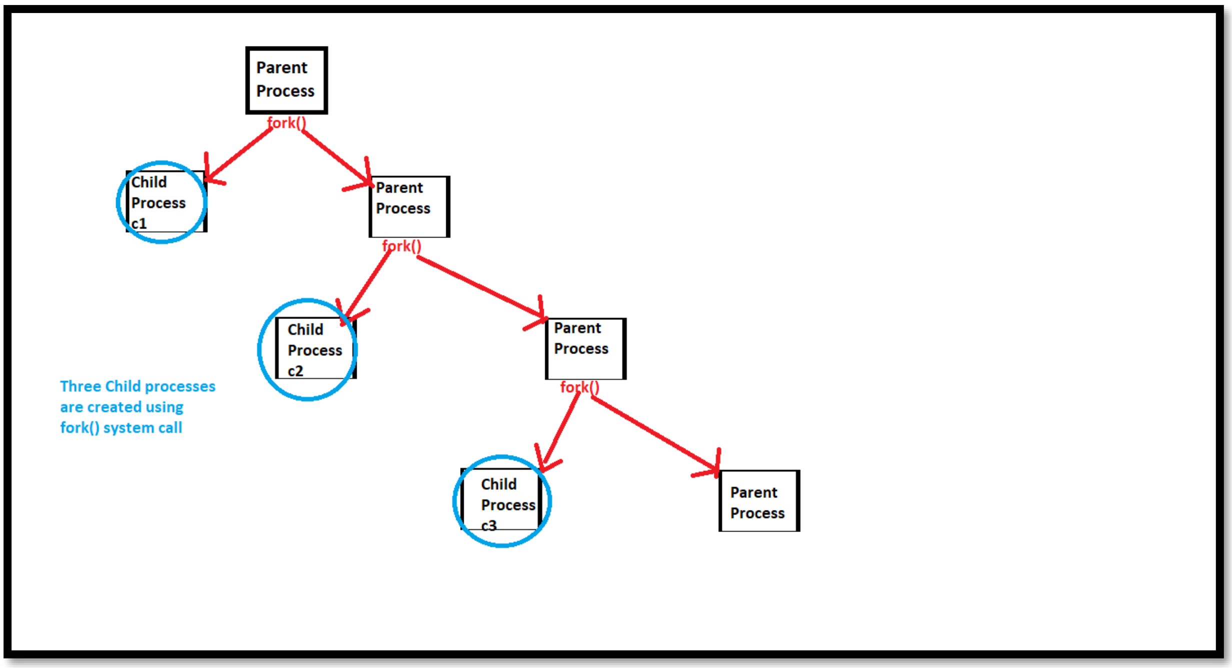


```
ubuntu22@ubuntu22-VirtualBox:~/CN$ gcc final.c -o prog
ubuntu22@ubuntu22-VirtualBox:~/CN$ ./prog

 In child process 1 : Got from Parent process: Hello, child 1

 In child process 3 : Got from Parent process: Hello, child 3

 In child process 3 : Got from sibling process: Hello child process 3, I am your sibling child process 1

 In child process 2 : Got from Parent process: Hello, child 2

 In child process 2 : Got from sibling process: Hello child process 2, I am your sibling child process 1

 In parent process: Got from child process 1: Hello parent, I am child process 1
ubuntu22@ubuntu22-VirtualBox:~/CN$ 
```

Explanation :

Topology Diagram :





Parent Process:

(a) Parent to all its child
[write to]
fd[0][1] -> for child process C1
fd[1][1] -> for child process C2
fd[2][1] -> for child process C3

(b) One of its child to parent
[read from]
fd[3][0] -> from child process C1

(c) Child process to all its sibling process
-

Child Process C1:

(a) Parent to all its child
[read from]
fd[0][0] -> from parent process

(b) One of its child to parent
[write to]
fd[3][1] -> for parent process

(c) Child process to all its sibling process
[write to]
fd[4][1] -> for child process C2
fd[5][1] -> for child process C3

Child Process C2:

(a) Parent to all its child
[read from]
fd[1][0] -> from parent process

(b) One of its child to parent
-

(c) Child process to all its sibling process
[read from]
fd[4][0] -> from child process C1

Child Process C3:

(a) Parent to all its child
[read from]
fd[2][0] -> from parent process

(b) One of its child to parent
-

(c) Child process to all its sibling process
[read from]
fd[5][0] -> from child process C1