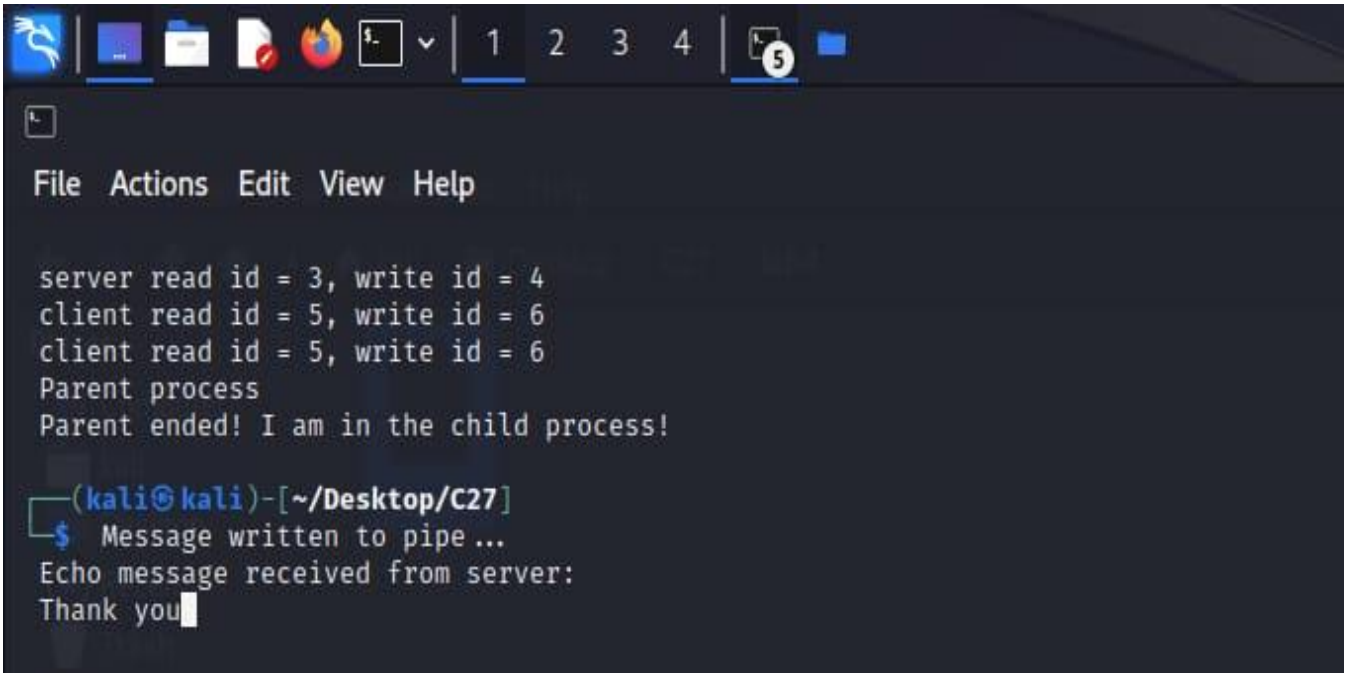
Student Name:	Aditya.Rajesh.Wanwade
Roll No:	C2-27
Practical No:	4
Aim:	Write C programs to simulate Intra & Inter – Process Communication (IPC) techniques: Pipes, Messages Queues, and Shared Memory.

A) ECHOSERVER USING PIPES SOURCE:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include<sys/types.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define msgsize 30

int main()
{
   int ser[2], cli[2], pid;
   char inbuff[msgsize];
   char *msg = "Thank you";

   system("clear"); // Clear the screen

   // Create pipes
   pipe(ser);
   pipe(cli);

   printf("\n server read id = %d, write id = %d", ser[0], ser[1]);
   printf("\n client read id = %d, write id = %d", cli[0], cli[1]);

   pid = fork(); // Fork a child process

   if (pid == 0) // Child process
   {
      printf("\n I am in the child process!");
      close(cli[0]); // Close unused read end of client pipe
      close(ser[1]); // Close unused write end of server pipe

      // Write message to client pipe
      write(cli[1], msg, strlen(msg) + 1); // Include the null terminator

      printf("\n Message written to pipe...");
```

```
        sleep(2); // Wait for a moment

        // Read echo message from server pipe
        read(ser[0], inbuff, msgsize);
        printf("\n Echo message received from server:");
        printf("\n %s", inbuff);
    }
    else // Parent process
    {
        close(cli[1]); // Close unused write end of client pipe
        close(ser[0]); // Close unused read end of server pipe

        printf("\n Parent process");

        // Read message from client pipe
        read(cli[0], inbuff, msgsize);

        // Write the same message back to server pipe
        write(ser[1], inbuff, msgsize);

        printf("\n Parent ended!");
    }

    return 0;
}
```

<u>OUTPUT:</u> (All the test cases are included):

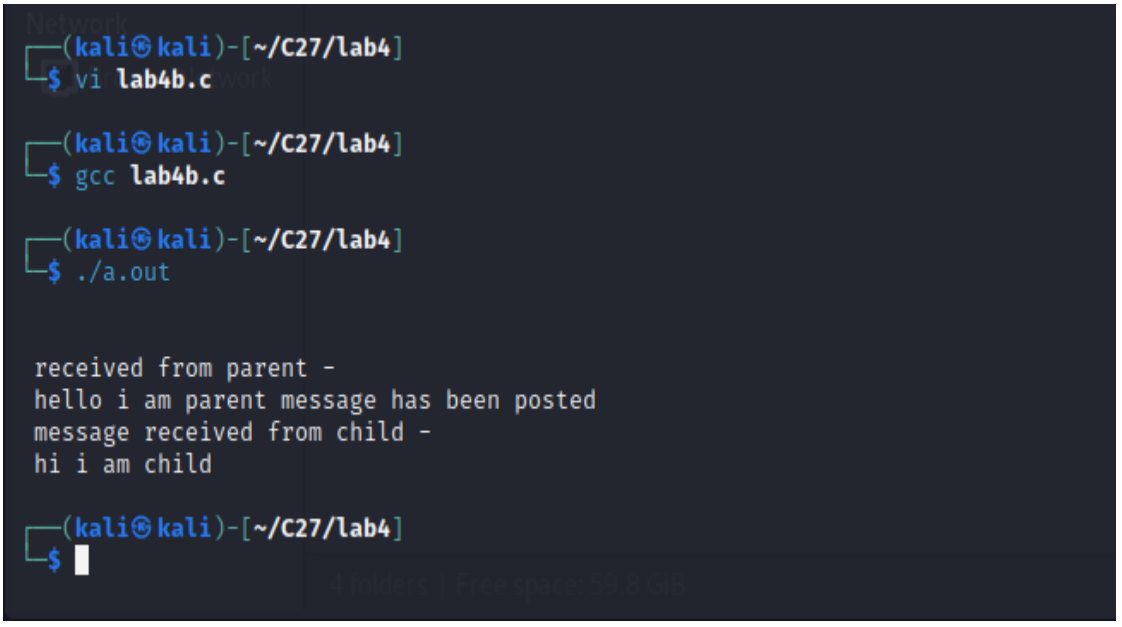


B) ECHO SERVER USING MESSAGES:

```
#include<sys/ipc.h>
#include<stdio.h>
#include<string.h>
#include<sys/msg.h>
#include<stdlib.h>
#include <unistd.h>
struct
{
long mtype;
char mtext[20];
}send,recv;
int main()
{
int qid,pid,len;
qid=msgget((key_t)0X2000,IPC_CREAT|0666);
if(qid==-1)
{
perror("\n message failed");
exit(1);
}
send.mtype=1;
strcpy(send.mtext,"\n hello i am parent");
len=strlen(send.mtext);
pid=fork();
if(pid>0)
{
if(msgsnd(qid,&send,len,0)==-1)
{
perror("\n message sending failed");
exit(1);
}
printf("\n message has been posted");
sleep(2);
if(msgrcv(qid,&recv,100,2,0)==-1)
{
perror("\n msgrcv error:");
exit(1);
}
printf("\n message received from child - %s\n",recv.mtext);
}
else
{
send.mtype=2;
strcpy(send.mtext,"\n hi i am child");
len=strlen(send.mtext);
```

```
if(msgrcv(qid,&recv,100,1,0)==-1)
{
perror("\n child message received failed");
exit(1);
}
if(msgsnd(qid,&send,len,0)==-1)
{
perror("\n child message send failed");
}
printf("\n received from parent - %s",recv.mtext);
}
}
```

<u>OUTPUT:</u> (All the test cases are included):



<u>Result</u>: Thus the program was executed and verified successfully.