***Problem based assignment on file API’s***

**1.Write a C program that takes one or more file/directory names as command line input and reports the following information**

**on the file:**

**o File type**

**o Number of links**

**o Time of last access**

**o Read, write, and execute permission**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

void getFileCreationTime(char \*path) {

    struct stat attr;

    stat(path, &attr);

    printf("Last modified time: %s", ctime(&attr.st\_mtime));

}

void printFileProperties(struct stat stats)

{

    printf("\nFile access: ");

    if (stats.st\_mode & R\_OK)

        printf("read ");

    if (stats.st\_mode & W\_OK)

        printf("write ");

    if (stats.st\_mode & X\_OK)

        printf("execute");

     printf("\n");

  }

int main(int argc, char \*argv[]){

        const char \*filename;

    int st,i;

  struct stat stats;

  if (stat(argv[1], &stats) == 0)

    {

for (i = 1; i < argc; ++i) {

if (stat(argv[i], &stats) == -1) {

      perror(argv[i]);

}

else {

        filename = argv[i];

     st = stat(filename,&stats);

     if( st==-1 )

     {

         fprintf(stderr,"File error!\n");

         exit(1);

     }

     if( S\_ISREG(stats.st\_mode) )

         printf(" '%s' is regular file\n",filename);

     else

         printf(" '%s' is not a regular file\n",filename);

printf("%s has %ld hard links\n", argv[i], stats.st\_nlink);

printFileProperties(stats);

getFileCreationTime(argv[1]);

}

     }

    }

    else

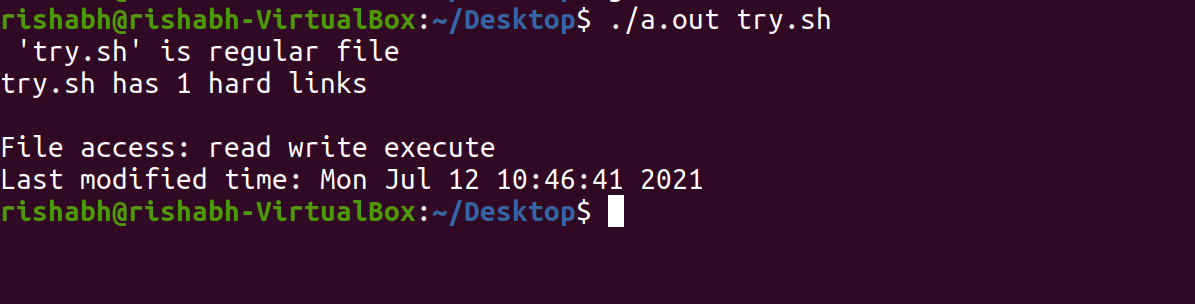
    {

        printf("Unable to get file properties.\n");

      //  printf("Please check whether '%s' file exists.\n", path);

    }

}



**2.  Write a program in C that illustrates how to execute two commands concurrently with a command pipe.**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

int main()

 {

int pfd[2],p;

 pipe(pfd);

p=fork();

 if(p==0)

 {

close(pfd[0]);

 close(1);

 dup(pfd[1]);

 execlp("ls","ls","-l",(char\*)0);

 }

 else

 {

 close(pfd[1]);

close(0);

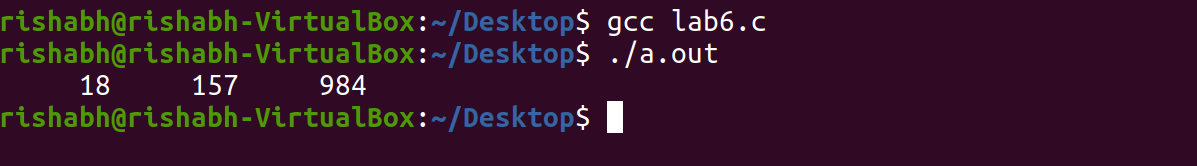
dup(pfd[0]);

 execlp("wc","wc",(char\*)0);

 }

return 0;

 }



**3. Write a C Program that makes a copy of a file using standard I/O and system calls**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<error.h>

int main(int argc,char \* argv[])

{

int fd = open(argv[1],O\_RDONLY);

if(fd==-1){

perror("can't open file");

return 0;

}

int fd1 = creat("copy.txt",0777);

int r;

char buf[1024];

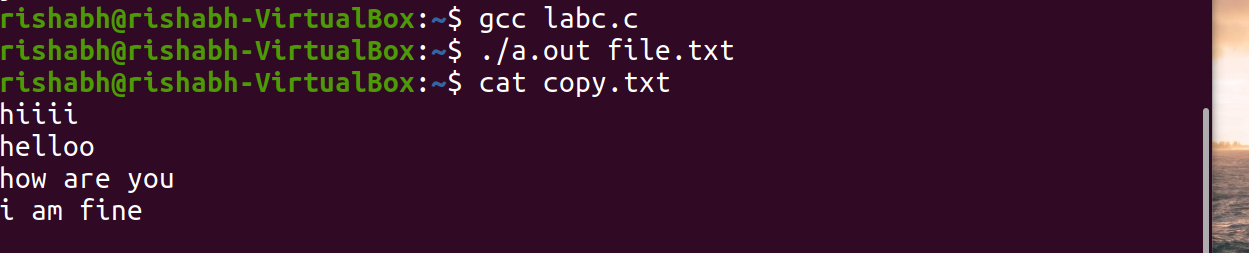
while((r = read(fd,buf,sizeof(buf)))>0)

write(fd1,buf,r);

close(fd);

close(fd1);

}

****

**4. Implement in C the following Unix commands using system calls**

**A)Cat**

**#**include<sys/types.h>

#include<sys/stat.h>

#include<stdio.h>

#include<fcntl.h>

Int main( int argc,char \*argv[] )

{

int fd,i;

char buf[2];

fd=open(argv[1],O\_RDONLY,0777);

if(fd==-argc)

{

printf("file open error");

}

else

{

while((i=read(fd,buf,1))>0)

{

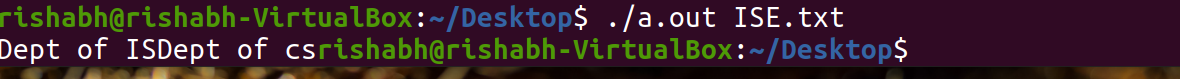
printf("%c",buf[0]);

}

close(fd);

}

}



**B)mv**

#include<sys/types.h>

#include<sys/stat.h>

#include<stdio.h>

#include<fcntl.h>

Int main( int argc,char \*argv[] )

{

int i,fd1,fd2;

char \*file1,\*file2,buf[2];

file1=argv[1];

file2=argv[2];

printf("file1=%s file2=%s",file1,file2);

fd1=open(file1,O\_RDONLY,0777);

fd2=creat(file2,0777);

while(i=read(fd1,buf,1)>0)

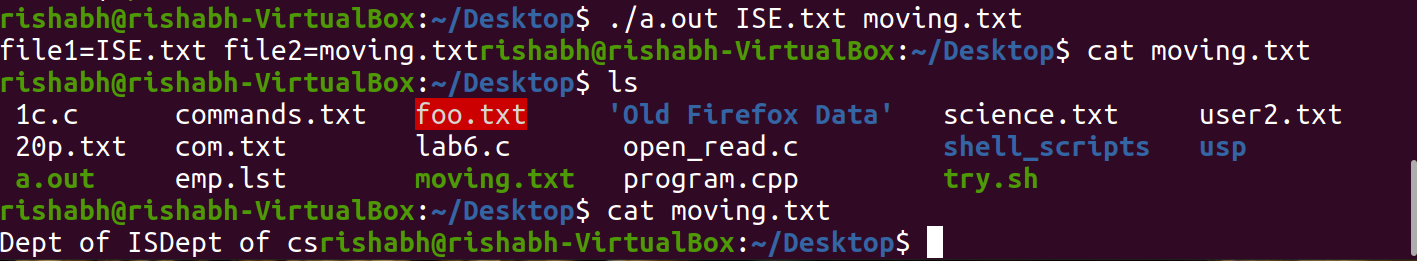
write(fd2,buf,1);

remove(file1);

close(fd1);

close(fd2);

}



**c)cp**

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

 {

   int f1, f2;

   char buff[50];

   long int n;

 if(((f1 = open(argv[1], O\_RDONLY)) == -1 || ((f2=open(argv[2], O\_CREAT |

 O\_WRONLY | O\_TRUNC, 0700))== 1))) {

       perror("problem in file");

       exit(1);

    }

 while((n=read(f1, buff, 50))>0)

     if(write(f2, buff, n)!=n) {

           perror("problem in writing");

           exit(3);

        }

     if(n==-1) {

           perror("problem in reading");

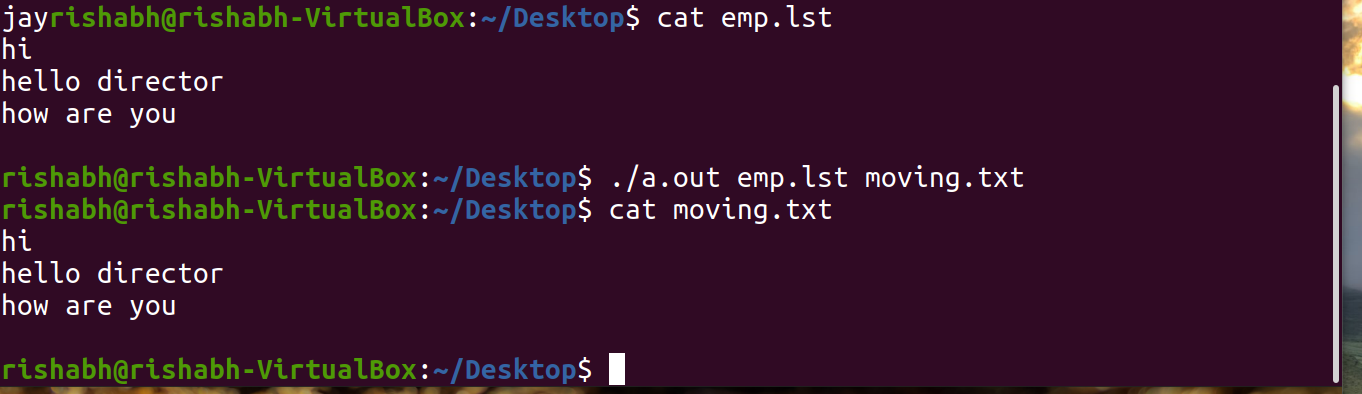
           exit(2);

        }

       close(f2);

       exit(0);

 }



**d)ln**

#include<stdio.h>

#include <stdlib.h>

#include <string.h>

#include<fcntl.h>

#include<unistd.h>

#include<sys/stat.h>

int main(int argc,char\* argv[])

{

int i;

struct stat s;

if (argc<3)

{

 perror("ERROR:Too Few Arguments");

 exit(1);

}

if(argc>4)

{

 perror("ERROR:Too Many Arguments");

 exit(1);

}

if(argc==3)

 i=0;

else

 i=1;

if(i && !(strcmp(argv[1],"-s")==0)) {

 perror("ERROR:Invalid Syntax");

 exit(1);

}

if(access(argv[i+1],F\_OK))

{

  perror("ERROR:File name not Found");

  exit(1);

}

if(!access(argv[i+2],F\_OK))

{

  perror("ERROR:File Name already exist");

  exit(1);

}

if(stat(argv[i+1],&s)<0)

{

  perror("ERROR:Unable to retrieve stat information");

  exit(1);

}

if(!S\_ISREG(s.st\_mode))

{

  perror("ERROR:Not a Regular File");

  exit(1); }

if(argc==3)

 if(link(argv[i+1],argv[i+2])<0)

 {

  perror("ERROR:Unable to create the Link");

  exit(1);

 }

if(argc==4)

 if(symlink(argv[i+1],argv[i+2])<0)

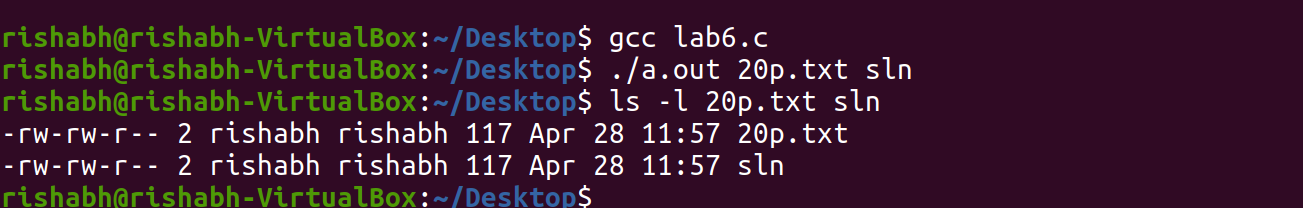
 {

  perror("ERROR:Unable to create the Link");

  exit(1);

 }

}



**5. Write a C program to list for every file in a directory, its inode number and file name.**

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

main(int argc, char \*argv[])

{

char d[50];

if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"-i ");

strcat(d,argv[1]);

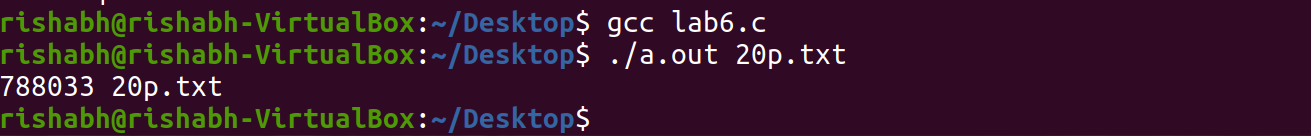
system(d);

}

else

printf("\nInvalid No. of inputs");

}



**6. Write a C Program that demonstrates redirection of standard output to a file .EX:ls>f1.**

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

int main(int argc, char \*argv[])

{

char d[50];

if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"> ");

strcat(d,argv[1]);

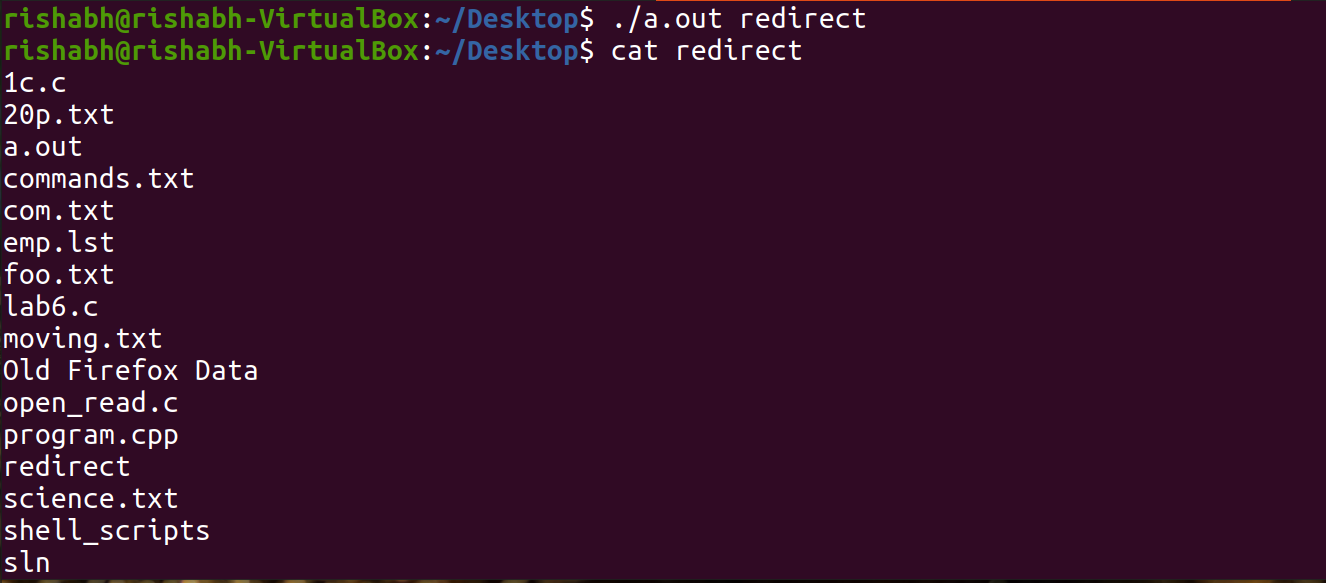
system(d);

}

else

printf("\nInvalid No. of inputs");

}



**7. Write a C program to create a child process and allow the parent to display “parent” and the child to display “child” on the screen**

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

int pid;

switch(pid = fork())

{

case -1:

printf("fork error");

sleep(10);

exit(0);

break;

case 0:

printf("I AM CHILD PROCESS....!\n");

exit(0);

break;

default:

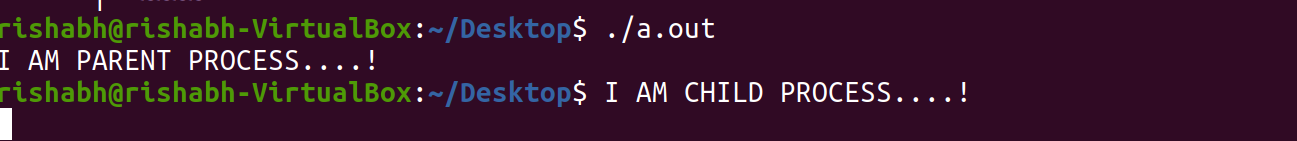
printf("I AM PARENT PROCESS....!\n");

exit(0);

break;

}

}



**8.Write a C program to create a Zombie process.**

#include<stdio.h>

#include<stdlib.h>

int main(){

if(fork()>0){

printf("\*\*\*\*\*CREATION OF ZOMBIE\*\*\*\*\*\n");

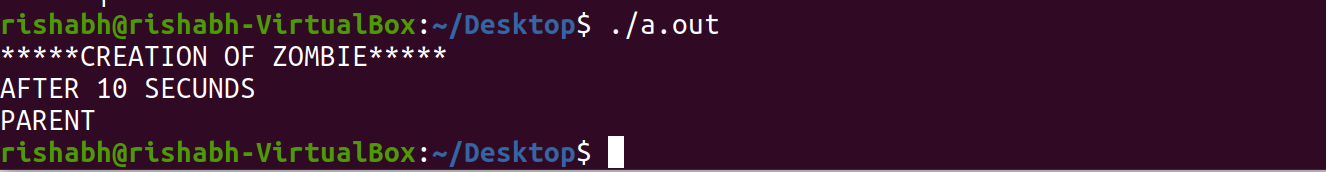
sleep(10);

printf("AFTER 10 SECUNDS\n");

printf("PARENT\n");

}

}



**9.Write a C program that illustrates how an orphan is created.**

#include<stdio.h>

#include<stdlib.h>

int main(){

int pid;

pid=fork();

if(pid==0){

printf("IAM CHILD PROCESS,MY PROCESS ID IS: %d",getpid());

printf("\n THE CHILDS'S PARENT PROCESSS ID IS %d\n",getppid());

sleep(10);

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*AFTER 10 SECUNDS\*\*\*\*\*\*\*\*\*\*");

printf("\n IAM CHILD PROCESS,MY PROCESS ID IS: %d",getpid());

printf("\n THE CHILDS'S PARENT PROCESS ID IS :%d\n",getppid());

}

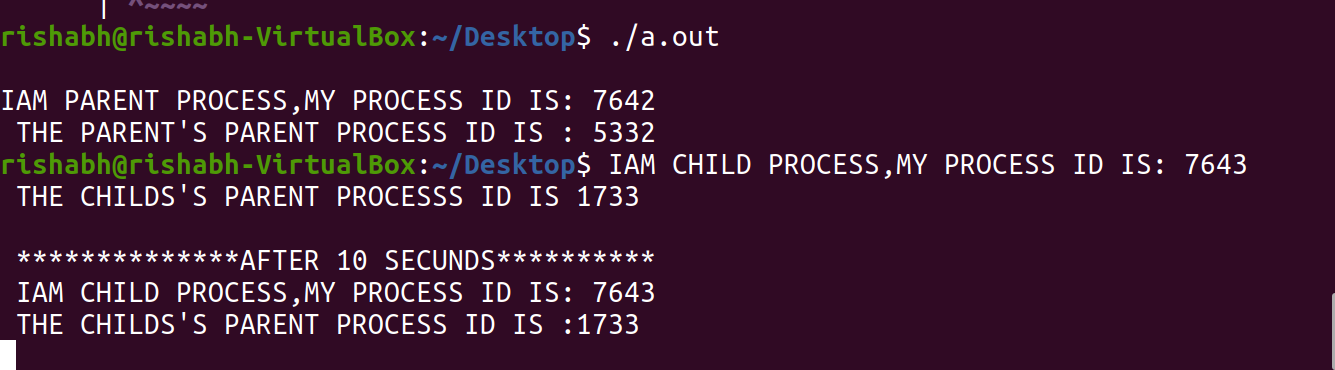
else{

printf("\nIAM PARENT PROCESS,MY PROCESS ID IS: %d",getpid());

printf("\n THE PARENT'S PARENT PROCESS ID IS : %d\n",getppid());

}

}



**11. Write a C program that illustrates communication between two unrelated processes using a named pipe.**

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<unistd.h>

int main()

{

int pfds[2];

char buf[30];

if(pipe(pfds)==-1)

{

perror("pipe");

exit(1);

}

printf("writing to file descriptor #%d\n", pfds[1]);

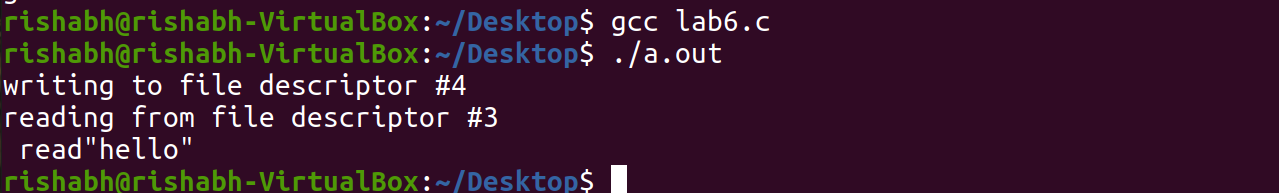
write(pfds[1],"test",5);

printf("reading from file descriptor #%d\n ", pfds[0]);

read(pfds[0],buf,5);

printf("read\"%s\"\n" ,buf);

}



**12. Write a C program that illustrates suspending and resuming processes using signals.**

#include<stdio.h>

#include <signal.h>

#include <stdlib.h>

void sig\_alarm(int signo)

{

printf("from sigalarm function\n");

}

int main()

{

int n;

if(signal(SIGALRM,sig\_alarm)==SIG\_ERR)

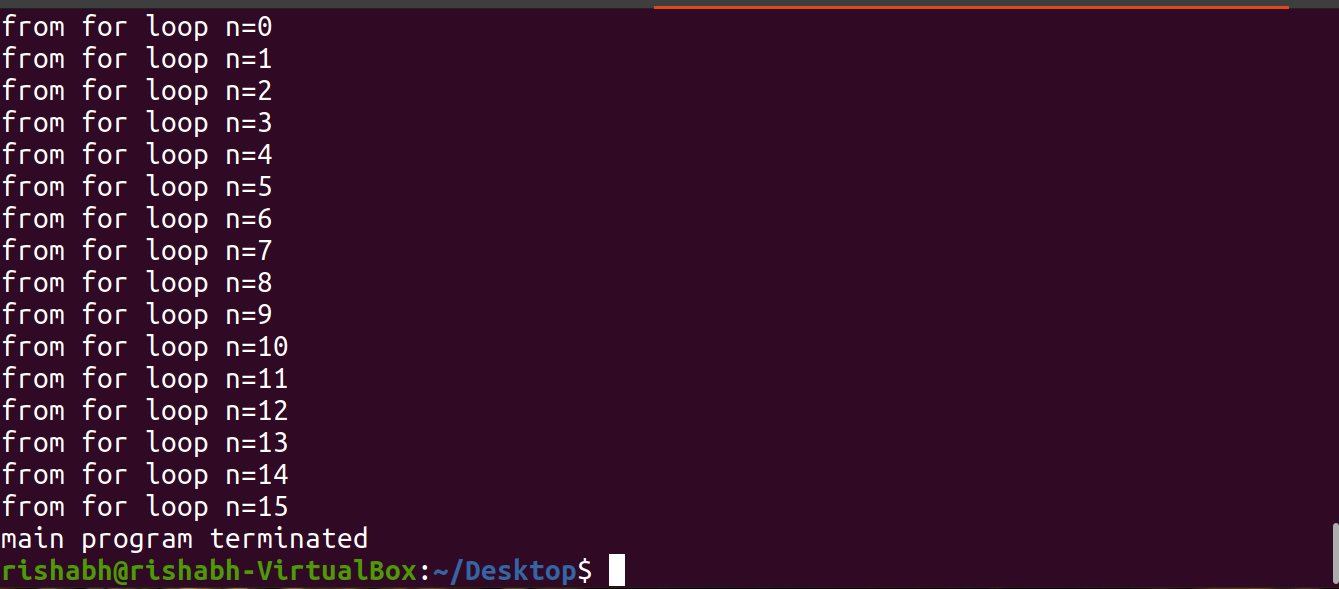
printf("Signal error");

alarm(5);

for(n=0;n<=15;n++)

printf("from for loop n=%d \n",n);

printf("main program terminated\n");



**13. Write a separate program using signal system call to catch the following signals.**

**o   SIGSEGV**

**o   SIGINT**

**o   SIGFPE**

**o   SIGALRM (use alarm system call)**

**o   SIGALRM (use setitimer system call)**

**o   SIGVTALRM (use setitimer system call)**

**o   SIGPROF (use setitimer system call)**

#include<iostream>

#include <stdio.h>

#include <signal.h>

using namespace std;

int main()

{

sigset\_t sigset;

sigemptyset(&sigset);

sigaddset(&sigset, SIGINT);

sigaddset(&sigset, SIGSEGV);

sigaddset(&sigset, SIGFPE);

sigaddset(&sigset, SIGALRM);

sigaddset(&sigset, SIGVTALRM);

sigaddset(&sigset, SIGPROF);

if (sigprocmask(SIG\_SETMASK, &sigset, 0) == -1)

perror("sigprocmask");

//initialize set

if (sigprocmask(0, NULL, &sigset) < 0)

perror("sigprocmask error");

if (sigismember(&sigset, SIGINT))  cout<<"SIGINT \n";

if (sigismember(&sigset, SIGSEGV))  cout<<"SIGSEGV \n";

if (sigismember(&sigset, SIGFPE))  cout<<"SIGFPE \n";

if (sigismember(&sigset, SIGVTALRM))  cout<<"SIGVTALRM \n";

if (sigismember(&sigset, SIGALRM))  cout<<"SIGALRM \n";

if (sigismember(&sigset, SIGPROF))  cout<<"SIGPROF \n";

}

