EX NO: 1A	BASICS UNIX COMMANDS
DATE:	

AIM:

To implement the Basic unix commands.

COMMANDS:

1.1 GENERAL PURPOSE COMMANDS:

1. **THE DATE COMMAND:**

The **date** command can also be used with following format.

Format	Purpose	Example
+ %m	To display only month	$ date + %m
+ %h	To display month name	$ date + %h
+ %d	To display day of month	$ date + %d
+ %y	To display last two digits of the year	$ date + %y
+ %H	To display Hours	$ date + %H
+ %M	To display Minutes	$ date + %M
+ %S	To display Seconds	$ date + %S

```
[exam@fosslab ~]$ date
Sat Feb 14 11:48:18 IST 2015
```

2. THE echo COMMAND:

```
[exam@fosslab ~]$ echo learning unix is intresting
learning unix is intresting
```

3. THE Who COMMAND:

```
[exam@fosslab ~]$ who
exam pts/0        2015-02-14 11:48 (192.168.8.5)
exam20 pts/0      2015-02-14 11:48 (192.168.8.6)
```

4. THE Who am i COMMAND:

```
[exam@fosslab ~]$ who am i
exam   pts/0      2015-02-14 11:48 (192.168.8.5)
```

5. THE UNIX CALENDER:

Cal:

```
[exam@fosslab ~]$ cal 2 2015
   February 2015
Su  Mo  Tu  We  Th  Fr  Sa
 1   2   3   4   5   6   7
 8   9  10  11  12  13  14
15  16  17  18  19  20  21
22  23  24  25  26  27  28
```

6. THE Finger COMMAND:

```
[exam@fosslab ~]$ finger exam25
Login: exam                     Name:
Directory: /home/exam           Shell: /bin/bash
On since Sat Feb 14 11:48 (IST) on pts/0 from 192.168.8.5
No mail. No Plan.
```

7. THE id COMMAND:

```
[exam@fosslab ~]$ id
uid=662(exam) gid=662(exam) groups=662(exam)
```

8. THE tty COMMAND:

```
[exam@fosslab ~]$ tty
/dev/pts/0
```

9.VIEW THE CONTENT:

```
[exam@fosslab ~]$ cat test
welcome to operating system. it is an interesting subject.
```

10. CLEARING THE SCREEN

```
[exam@fosslab student ~]$ tput clear
```

1.2 DIRECTORY COMMANDS

1.CREATE A DIRECTORY:

```
[exam@fosslab ~]$ mkdir student
[exam@fosslab ~]$ cd student
[exam@fosslab student]$
```

2 CURRENT WORKING DIRECTORY:

```
[exam@fosslab ~]$ pwd
/home/exam
```

3.REMOVING A DIRECTORY:

[exam@fosslab student]$ rmdir student
[exam@fosslab ~]$

4.LISTING THE FILES AND DIRECTORIES:

ls:

[exam@fosslab student~]$ ls
a.out data program public_html share stud25 student test test1

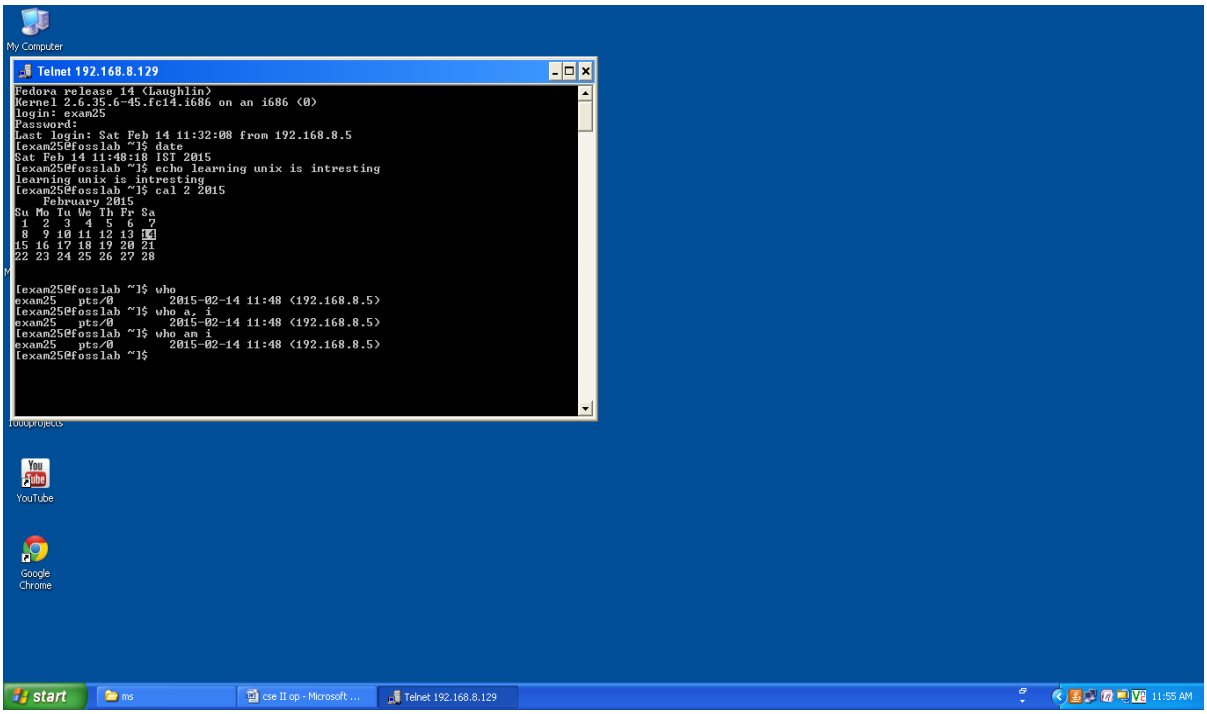
5.CHANGING THE WORKING DIRECTORY:

Cd:

[exam@fosslab ~]$ pwd
/home/exam/
[exam@fosslab ~]$ mkdir student
[exam@fosslab ~]$ cd student
[exam@fosslab student]$ pwd
/home/exam/student/

6. THE PATH

[exam@fosslab student~]$ echo $PATH
/usr/lib/qt-3.3/bin:/usr/lib/mpich2/bin:/usr/lib/ccache:/usr/local/bin:/bin:/usr
/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/lib/alliance/bin:/usr/libexec/sdcc:/op
t/ns2/bin:/opt/ns2/tcl8.4.14/unix:/opt/ns2/tk8.4.14/unix:/opt/ns2/ns-2.34/:/opt/
ns2/nam-1.14/:/home/exam25/bin



7.CHANGE THE PASSWORD:

[exam@fosslab ~]$ passwd
(current) UNIX PASSWORD: ********
New Password: ********
Re-enter Password: ********
$

1.3 FILE HANDLING COMMANDS

1. THE CAT COMMAND:

[exam@fosslab ~]$ cat>test
welcome to operating system. it is an interesting subject.

2. COPYING THE FILE:

Cp:

[exam@fosslab student~]$ cat test
welcome to operating system. it is an interesting subject.
[exam@fosslab student ~]$ cat test1
the basic unix commands are cat, pwd, mkdir, rmdir, cd, path,clear,cp,rm,
mv,ls,wc.
[exam@fosslab student~]$ cp test test1
[exam@fosslab student ~]$cat test1
welcome to operating system. it is an interesting subject.

3. REMOVING A FILE:

Rm:

[exam@fosslab student ~]$ rm test1
[exam@fosslab student ~]$ cat test1
Cat :test1: No such file or directory

4.MOVING A FILE:

[exam@fosslab student~]$ cat >test1
the basic unix commands are cat,pwd,clear,cp,mv,rm,mv,test..^C

mv:

[exam@fosslab student ~]$ mv test test1
[exam@fosslab student ~]$ cat test 1
the basic unix commands are cat, pwd, mkdir, rmdir, cd, path,clear,cp,rm,
mv,ls,wc.

5. DIRECTING OUTPUT TO A FILE:

[exam@fosslab student~]$ ls>test
[exam@fosslab student ~]$ cat test
a.out
data
mylist

program
public_html
share
stud25
student
test
test1
[exam@fosslab ~]$

6. COUNTING NUMBER OF WORDS IN A FILE:

wc:

[exam@fosslab ~]$ wc test
10 10 70 test

7. THE FILE COMMAND

[exam@fosslab ~]$ file test
test: ASCII Pascal program text
[exam@fosslab ~]$ cat test
a.out
data
mylist
program
public_html
share
stud25
student
test

8. CHANGING THE FILE PERMISSION:

Chmod:

[exam@fosslab ~]$ chmod u-wx test
[exam@fosslab ~]$ cat > test
-bash: test: Permission denied

1.4 FILTER COMMANDS

1.SORTING THE CONTENTS: (sort)

[exam@fosslab ~]$ sort test1
a.out
data
mylist
program
public_html
share
stud25

student
test
test1

2. THE uniq COMMAND:

[exam@fosslab ~]$ cat > dept.lst
01 accounts 3977
01 accounts 3977
02 admin 1707
03 marketing 39
03 marketing 39
04 personel 77
05 production 1739
06 sales 1008^C
[exam@fosslab ~]$ uniq dept.lst
01 accounts 3977
02 admin 1707
03 marketing 39
04 personel 77
05 production 1739

3. ADDING LINE NUMBERS:

nl

[exam@fosslab ~]$ nl test1
1 a.out
2 data
3 mylist
4 program
5 public_html

4. SELECTING FIELDS FROM A LINE:

cut

[exam@fosslab ~]$ cat >std
Aswini
Bharathi
Charu
Deepa^C
[exam@fosslab ~]$ cut -c1 std
A
B
C
D

5. THE more COMMAND:

[exam@fosslab ~]$ more test1
a.out
data
mylist
program
public_html
stud25
student
test
test1

6. PASTING FILES:

[exam@fosslab ~]$ paste std
Aswini
Bharathi
Charu
Deepa

7. COMPARING FILES:

cmp

[exam@fosslab ~]$ cmp test1 std
test1 std differ: byte 1, line 1

8. THE mesg COMMAND:

mesg

[exam@fosslab ~]$ mesg exam25
Usage: mesg [y|n]
[exam@fosslab ~]$ y
Message from exam25@fosslab.linuxpert.in on pts/0 at 12:34 ...
hiiiii ... study well for your exams
hiiiii ... study well for your exams

9. THE write COMMAND:

write

[exam@fosslab ~]$ write exam25
Message from exam25@fosslab.linuxpert.in on pts/0 at 12:30 ...
Hiiii
Hiiii

10. SENDING MESSAGE TO ALL THE USERS:

wall:

[exam@fosslab ~]$ wall os laboratory lab records
Broadcast message from exam25@fosslab.linuxpert.in (pts/0) (Wed Jan 21 13:21:os laboratory lab records

11. SENDING MAIL TO USERS:

mail:

[exam@fosslab ~]$ mail user2

Subject: about operating systems

THE OPERATING SYSTEMS BOOk is a "practice of some materials to gain knowledge"

EOT

12.THE reply COMMAND:

reply

reply exam25

Thanks For Giving A Mail

13. NO LOGGING OUT:

[exam@fosslab ~]$ std.lst test1.lst &

[1] 4663

14. THE nohup COMMAND:

[exam@fosslab ~]$ nohup test1.lst &

[1] 4685

15. Execution of a Job With Low Priority:

nice

[exam@fosslab ~]$ nice wc -l std &

[2] 4721

[1] Exit 127 nohup test1.lst

[exam@fosslab ~]$ 4 std

16. THE at COMMAND:

[exam@fosslab ~]$ at 12.54pm

at> today at evening 4pm

at> 21.30 tue next at 9.20

at> 2pm apr3 next 3rd

17. THE sleep COMMAND:

[exam@fosslab ~]$ sleep 1

-

18. KILLING PROCESSES WITH SIGNALS

kill

[exam@fosslab ~]$ kill 4921

RESULT:

Thus the basic UNIX commands were executed successfully.

EX NO: 1B	SYSTEM CALLS OF UNIX OPERATING SYSTEM
DATE:	

I. CLOSE()

AIM:

To write the program to implement the system calls close().

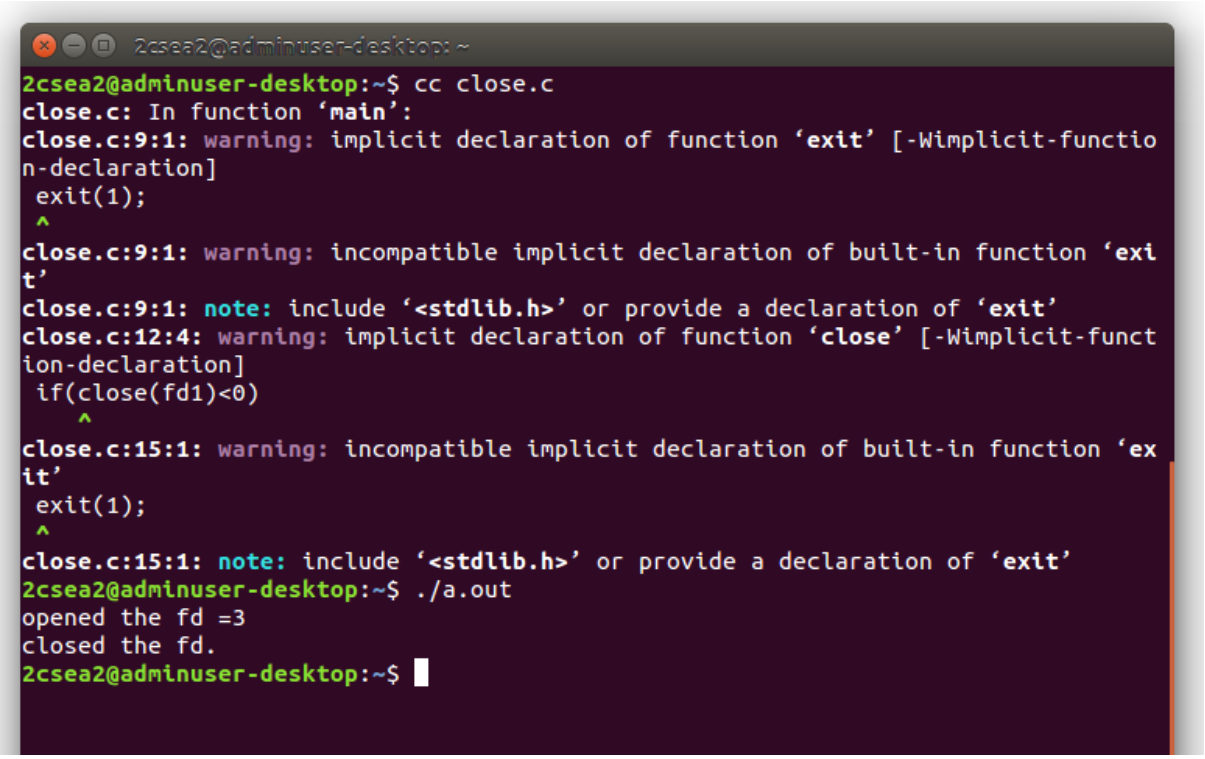
ALGORITHM:

Step 1: Start
Step 2: In the main function pass the arguments.
Step 3: Create structure as stat buff and the variables as integer.
Step 4: Use the for loop initialization.
Step 5: Stop.

PROGRAM:

```
#include<stdio.h>

#include<fcntl.h>

int main()

{

int fd1 = open("foo.txt",O_RDONLY);

if(fd1<0)

{

perror("c1");

exit(1);

}

printf("opened the fd =%d\n",fd1);

if(close(fd1)<0)

{

perror("c1");

exit(1);

}

printf("closed the fd.\n");

}
```

OUTPUT:



RESULT:

Thus the program was executed and verified successfully.

II. GETPID()

AIM:

To write the program to implement the system calls getpid()

ALGORITHIM:

Step 1: Start
Step 2: Get the process id integer value by using the system call getpid()
Step 3: It returns the process id of the calling process.
Step 4: After getting the pid value it prints the process id number an exists.
Step 5: Then compile the program either with the gcc or cc command.
Step 6: Run the program.
Step 7: Stop

PROGRAM:

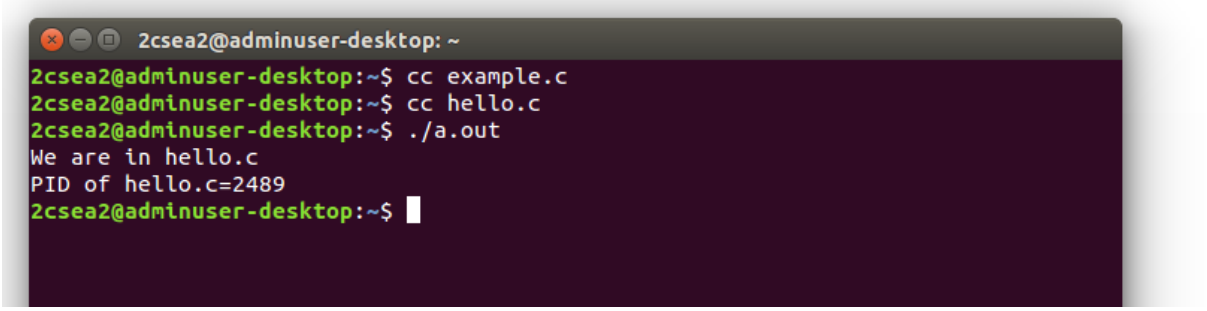
Example.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
printf("PID of example.c=%d\n",getpid());
char *args[]={"hello", "c","programming",NULL};
execv("./hello",args);
printf("BACK TO EXAMPLE.C");
}
```

Hello.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main (int argc, char *argv[])
{
printf("We are in hello.c\n");
printf("PID of hello.c=%d\n",getpid());
return 0;
}
```

OUTPUT:



RESULT:

Thus the program for getpid() system call has been executed and verified successfully.

III. FORK()

AIM:

To write the program to create a Child Process using system call fork().

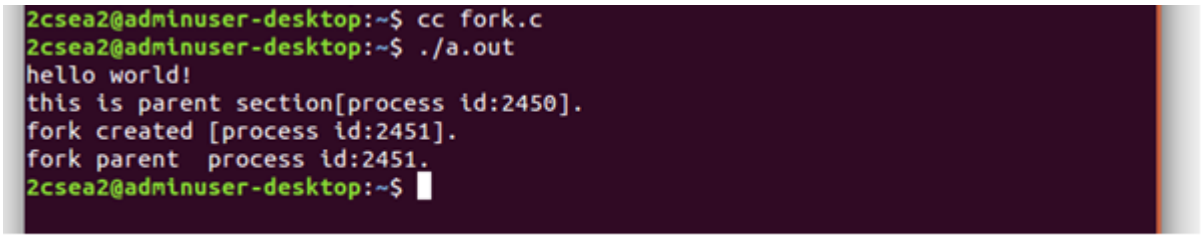
ALGORITHM:

Step 1: Declare the variable pid.
Step 2: Get the pid value using system call fork().
Step 3: If pid value is less than zero then print as “Fork failed”.
Step 4: Else if pid value is equal to zero include the new process in the system‟s file using execlp system call.
Step 5: Else if pid is greater than zero then it is the parent process and it waits till the child completes using the system call wait()
Step 6: Then print “Child complete”.

PROGRAM:

```
#include <stdio.h>
#include<unistd.h>
int main()
{
int id;
printf("hello world!\n");
id=fork();
if (id>0)
{
printf("this is parent section[process id:%d].\n",getpid());
}
else if(id==0)
{
printf("fork created [process id:%d].\n",getpid());
printf("fork parent  process id:%d.\n",getpid());
}
else
{
printf("fork created failed!!\n");
}
return 0;
}
```

OUTPUT:



RESULT:

Thus the program for fork() system call has been executed and verified successfully.

IV. OPEN()

AIM:

To write the program to implement the system call open( ).

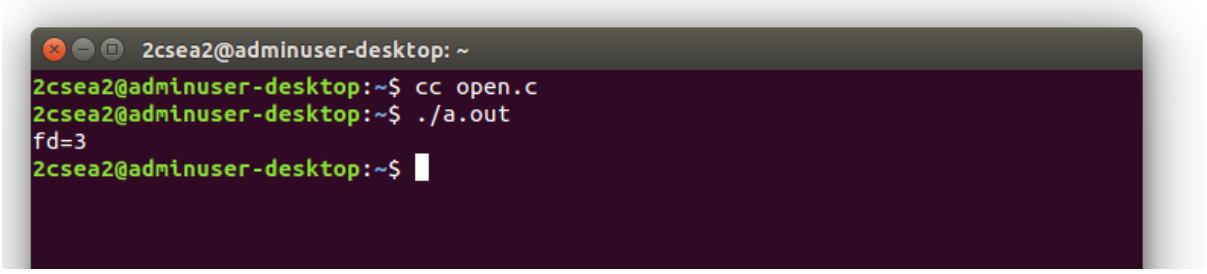
ALGORITHM:

Step 1 : Declare the structure elements.
Step 2 : Create a temporary file named temp1.
Step 3 : Open the file named “test” in a write mode.
Step 4 : Enter the strings for the file.
Step 5 : Write those strings in the file named “test”.
Step 6 : Create a temporary file named temp2.
Step 7 : Open the file named “test” in a read mode.
Step 8 : Read those strings present in the file “test” and save it in temp2.
Step 9 : Print the strings which are read.

PROGRAM:

```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
int fd=open("foo.txt",O_RDONLY|O_CREAT);
printf("fd=%d\n",fd);
if(fd ==-1)
{
printf("Eror no.=%d\n",errno);
printf("Program");
}
return 0;
}
```

OUTPUT:



RESULT:

Thus the program for open() system call has been executed and verified successfully.

V. READ()

AIM:

To write the program to implement the system call read( ).

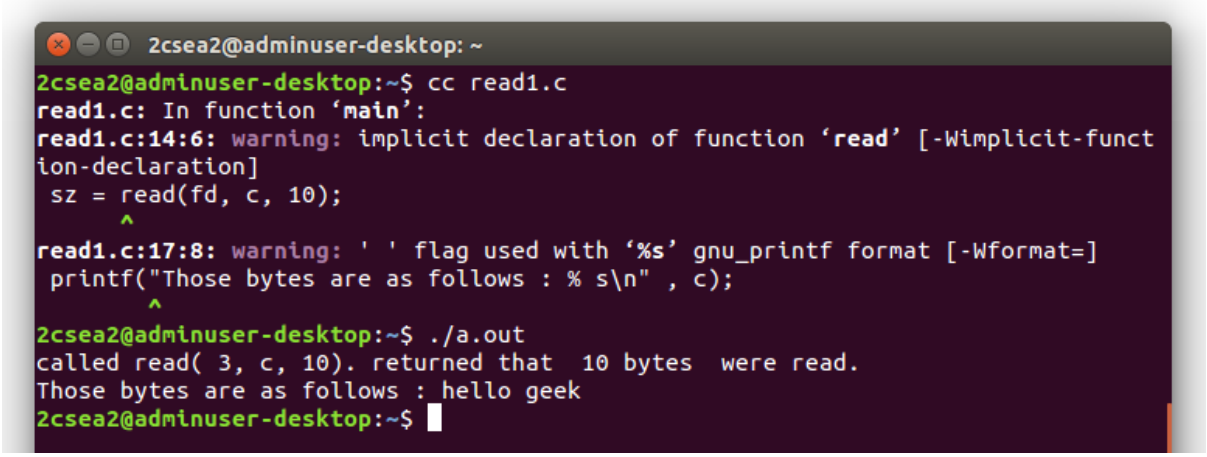
ALGORITHM:

Step 1 : Declare the structure elements.
Step 2 : Create a temporary file named temp1.
Step 3 : Open the file named “test” in a write mode.
Step 4 : Enter the strings for the file.
Step 5 : Write those strings in the file named “test”.
Step 6 : Create a temporary file named temp2.
Step 7 : Open the file named “test” in a read mode.
Step 8 : Read those strings present in the file “test” and save it in temp2.
Step 9 : Print the strings which are read.

PROGRAM:

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
int main()
{
int fd, sz;
char *c = (char *) calloc(100,sizeof(char));
fd = open("foo.txt", O_RDONLY);
if(fd < 0)
{
perror("r1");
exit(1);
}
sz = read(fd, c, 10);
printf("called read(% d, c, 10). returned that" " % d bytes  were read.\n" , fd, sz);
c[sz] = '\0';
printf("Those bytes are as follows : % s\n" , c);
}
```

OUTPUT:



RESULT:

Thus the program for read() system call has been executed and verified successfully.

VI. WRITE ()

AIM:

To write the program to implement the system call write( ).

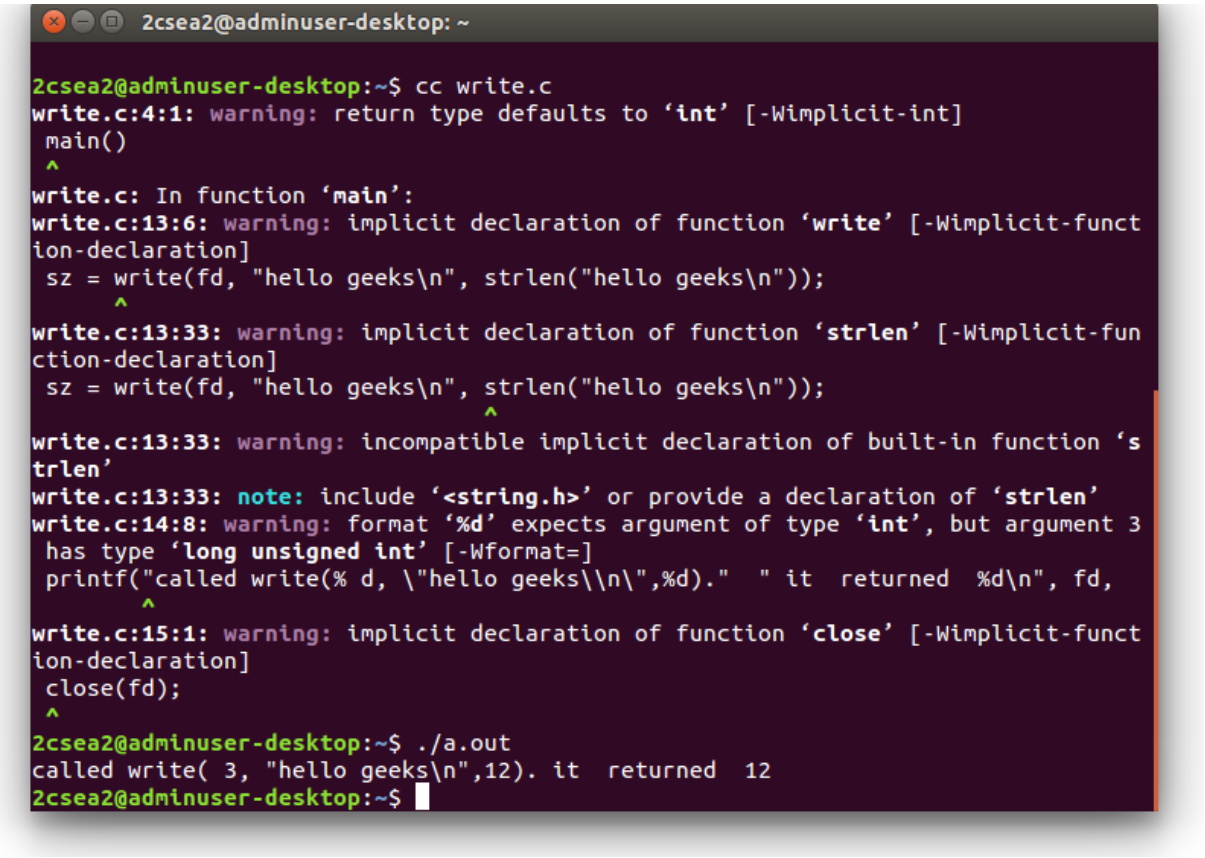
ALGORITHM:

Step 1 : Declare the structure elements.
Step 2 : Create a temporary file named temp1.
Step 3 : Open the file named “test” in a write mode.
Step 4 : Enter the strings for the file.
Step 5 : Write those strings in the file named “test”.
Step 6 : Create a temporary file named temp2.
Step 7 : Open the file named “test” in a read mode.
Step 8 : Read those strings present in the file “test” and save it in temp2.
Step 9 : Print the strings which are read.

PROGRAM:

```
#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

main()

{

int  sz;

int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

if(fd < 0)

{

perror("r1");

exit(1);

}

sz = write(fd, "hello geeks\n", strlen("hello geeks\n"));

printf("called write(% d, \"hello geeks\\n\",%d)."  " it  returned  %d\n", fd, strlen("hello geeks\n"), sz);

close(fd);

}
```

OUTPUT:



RESULT:

Thus the program for write() system call has been executed and verified successfully.

EX NO: 2	SIMULATION OF UNIX COMMANDS (GREP, CP, LS)
DATE:	

A. SIMULATION OF GREP UNIX COMMAND

AIM:

To write a c program to simulate grep unix command.

ALGORITHM:

Step 1: Include necessary header files.

Step 2: Make necessary declarations.

Step 3: Read the file name from the user and open the file in the read only mode.

Step 4: Read the pattern from the user.

Step 5: Read a line of string from the file and search the pattern in that line.

Step 6: If pattern is available, print the line.

Step 7: Repeat the step 4 to 6 till the end of the file.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include<string.h>
#include <fcntl.h>

void match_pattern(char *argv[])
{
   int fd,r,j=0;
   char temp,line[100];
   if((fd=open(argv[2],O_RDONLY)) != -1)
   {
      while((r=read(fd,&temp,sizeof(char)))!= 0)
      {
         if(temp!='\n')
         {
            line[j]=temp;
            j++;
         }
         else
         {
            if(strstr(line,argv[1])!=NULL)
               printf("%s\n",line);
            memset(line,0,sizeof(line));
            j=0;
         }

      }
   }
}

main(int argc,char *argv[])
{
   struct stat stt;
   if(argc==3)
   {
      if(stat(argv[2],&stt)==0)
         match_pattern(argv);
      else
      {
         perror("stat()");
exit(1);
      }
   }
}
```

test.txt
he is using ubuntu...

OUTPUT:

RESULT:

Thus the program for simulating grep unix command has been executed and verified successfully.

B. SIMULATION OF CP UNIX COMMAND

AIM:

To write a c program to simulate cp unix command.

ALGORITHM:

Step 1: Include necessary header files for manipulating directory.

Step 2: Declare and initialize required objects.

Step 3: Read matrix rows and columns.

Step 4:Read the matrix elements form the user.

Step 5: Compute sum of diagonal matrix.

Step 6: Stop the program.

PROGRAM:

```
#include<stdio.h>
void get_matrix(int m[20][20],int n)
{
int i,j;
for(i=0;i<n;i++)
 for(j=0;j<n;j++)
{
scanf("%d",&m[i][j]);
}
}
void print_matrix(int m[20][20],int n)
{
int i,j;
for(i=0;i<n;i++)
{
printf("\n\t");
for(j=0;j<n;j++)
{
printf("%d\t",m[i][j]);
}
}
printf("\n");
}
int main()
{
int m[20][20],n,i,trace=0;
printf("\n enter the no.of.R&c for the square matrix (n*n):");
scanf("%d",&n);
printf("\n enter the elements for matrix (%d*%d)\n",n,n);
get_matrix(m,n);
printf("\n matrix read:");
print_matrix(m,n);
for(i=0;i<n;i++)
{
trace = trace +m[i][i];
}
printf("\n sum of diagonal elements(trace) of the  matrix :%d\n",trace);
return 0;
}
```

OUTPUT:

```
2csea2@adminuser-desktop:~$ cc test2.c
2csea2@adminuser-desktop:~$ ./a.out

 enter the no.of.R&c for the square matrix (n*n):3

 enter the elements for matrix (3*3)
1 2 3
4 5 6
7 8 9

 matrix read:
        1       2       3
        4       5       6
        7       8       9

 sum of diagonal elements(trace) of the  matrix :15
2csea2@adminuser-desktop:~$
```

RESULT:

Thus the program for simulating cp unix command has been executed and verified successfully.

C. SIMULATION OF LS UNIX COMMAND

AIM:

To write a c program to simulate ls unix command.

ALGORITHM:

Step 1: Include necessary header files for manipulating directory.

Step 2: Declare and initialize required objects.

Step 3: Read the directory name form the user.

Step 4: Open the directory using opendir() system call and report error if the directory is not available.

Step 5: Read the entry available in the directory.

Step 6: Display the directory entry ie., name of the file or sub directory.

Step 7: Repeat the step 6 and 7 until all the entries were read.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>

int main(void)
{
struct dirent *de;
DIR *dr = opendir(".");
if(dr==NULL)
{
printf("could not open current directory");
return 0;
}
while ((de=readdir(dr))!=NULL)
printf("%s\t",de->d_name);
closedir(dr);
return 0;
}
```

OUTPUT:

```
2csea2@adminuser-desktop:~$ cc  ls1.c
2csea2@adminuser-desktop:~$ ./a.out
.xsession-errors        examples.desktop        ls1.c   .cache  .sjf.c.swp      semaphore.c     Documents       Videos  Pictures        Public
.profile        a.out   .viminfo        .ICEauthority   .gconf  .local  .rr.c.swp       .bashrc ls.c    ..      .Xauthority     .       .bash_
logout  student .factorial.sh.swp       fcfs.c  Desktop .xsession-errors.old    .config .bash_history   factorial.sh    Downloads       Music.
dmrc    .semaphore.c.swo        Templates       fcfs.c~ .job.c.swp      .shortest.c.swp 2csea2@adminuser-desktop:~$
```

RESULT:

Thus the program for simulating ls unix command has been executed and verified successfully.

EX NO: 3	SHELL PROGRAMMING
DATE:	

A. AREA OF THE CIRCLE

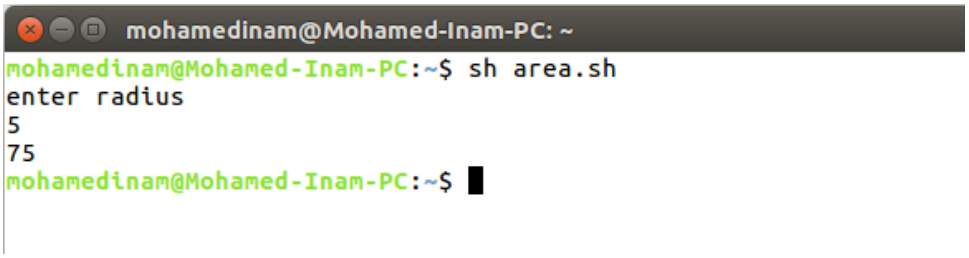
AIM:

To write a shell program to find the area of circle.

SHELL SCRIPT:

```
echo "enter radius"
read r
val=`expr 3 \* $r \* $r`
echo "$val"
```

OUTPUT:



RESULT:

Thus the shell program to find the area of circle is written and executed successfully

EX NO: 3B	SHELL PROGRAMMING – BIGGEST OF TWO NUMBERS
DATE:	

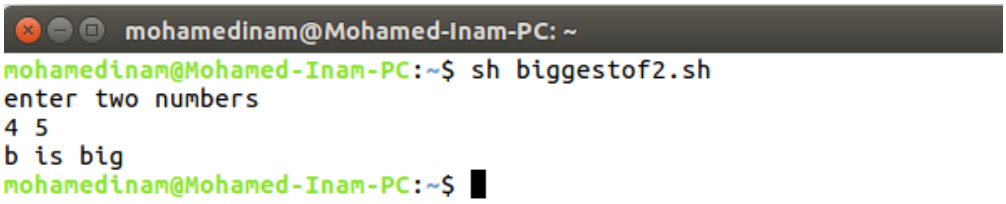
AIM:

To write a shell program to find the biggest of two numbers

SHELL SCRIPT:

```
echo "enter two numbers"
read a b
if [ $a -gt $b ]
then
echo "a is big"
else
echo "b is big"
fi
```

OUTPUT:



RESULT:

Thus the shell program to find the biggest of two numbers is written and executed successfully

EX NO: 3C	SHELL PROGRAMMING – BIGGEST OF THREE NUMBERS
DATE:	

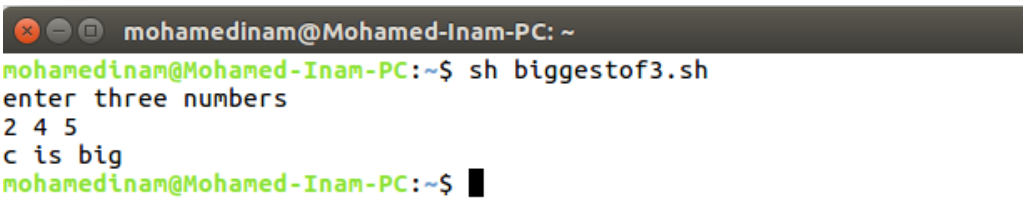
AIM:

To write a shell program to find the biggest of three numbers

SHELL SCRIPT:

```
echo "enter three numbers"
read a b c
if [ $a -gt $b ]&&a [ $b -gt $c ]
then
echo "a is big"
elif [ $b -gt $c ]
then
echo "b is big"
else
echo "c is big"
fi
```

OUTPUT:



RESULT:

Thus the shell program to find the biggest of three numbers is written and executed successfully.

EX NO: 3D	SHELL PROGRAMMING – NATURAL NUMBERS UPTO 'N'
DATE:	

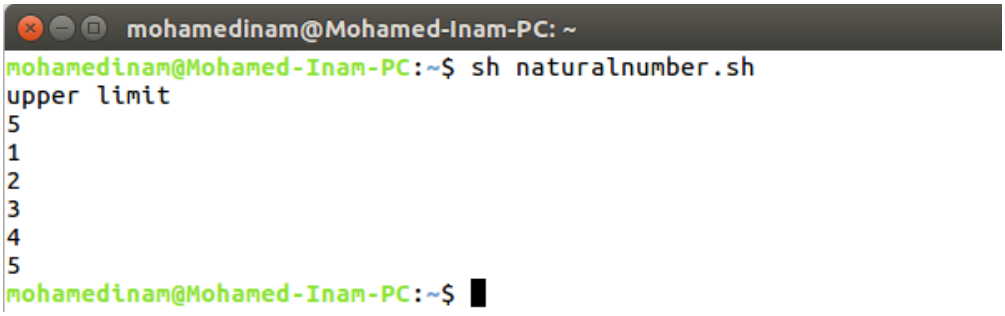
AIM:

To write a shell program to find n natural numbers

SHELL SCRIPT:

```
echo "upper limit"
read n
$i=0
while [ $i -lt $n ]
do
echo $i
i=`expr $i+1`
done
```

OUTPUT:



RESULT:

Thus shell program to find n natural numbers is written and executed successful

EX NO: 3E	SHELL PROGRAMMING – NATURAL NUMBERS USING FOR LOOP
DATE:	

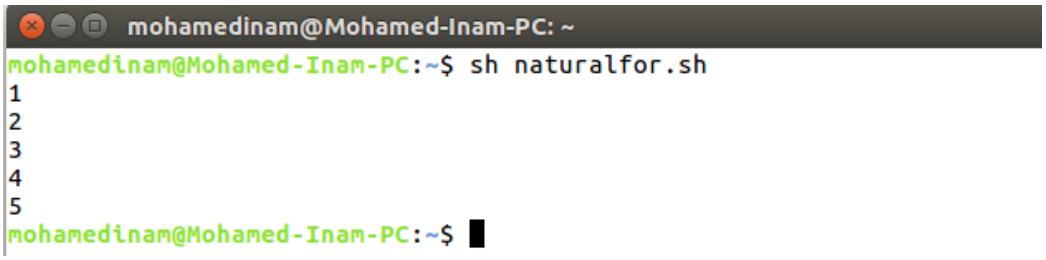
AIM:

To write a shell program to find n natural numbers using for loop.

SHELL SCRIPT:

```
for i in 1 2 3 4 5
do
echo $i
done
```

OUTPUT:



RESULT:

Thus the shell program to find n natural numbers using for loop is written and executed successfully

EX NO: 3F	SHELL PROGRAMMING – ARITHMETIC OPERATIONS
DATE:	

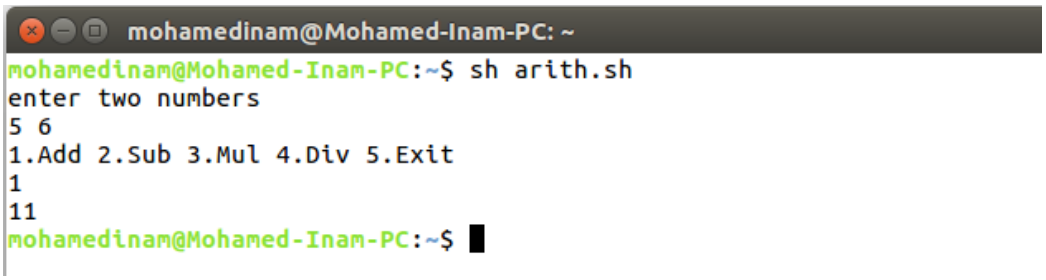
AIM:

To write a shell program to perform arithmetic operations on two numbers.

SHELL SCRIPT:

```
echo "enter two numbers"
read a b
echo "1.Add 2.Sub 3.Mul 4.Div 5.Exit"
read op
case $op in
1) c=`expr $a + $b` ;;
2) c= `expr $a - $b` ;;
3) c= `expr $a \ * $b` ;;
4) c= `expr $a / $b` ;;
5) exit
esac
echo $c
```

OUTPUT:



RESULT:

Thus the shell program to perform arithmetic operations on two numbers is written and executed successfully.

EX NO: 3G	SHELL PROGRAMMING – GROSS SALARY OF AN EMPLOYEE
DATE:	

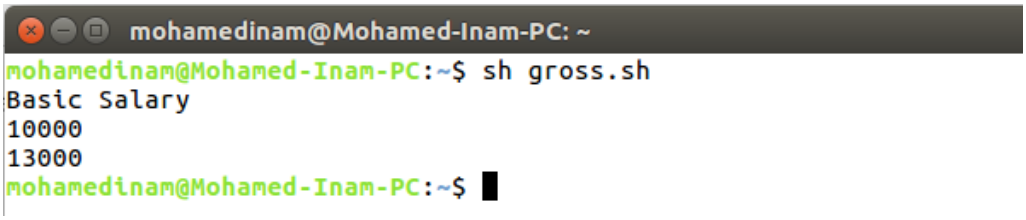
AIM:

To write a shell program to calculate gross salary of an employee

SHELL SCRIPT:

```
echo "Basic Salary"
read basic
da=`expr $basic \* 10 / 100`
hra=`expr $basic \* 20 / 100`
gross=`expr $basic + $da + $hra`
echo "$gross"
```

OUTPUT:



RESULT:

Thus the shell program to calculate gross salary of an employee is written and executed successfully.

EX NO: 3H	SHELL PROGRAMMING – FACTORIAL OF A NUMBER
DATE:	

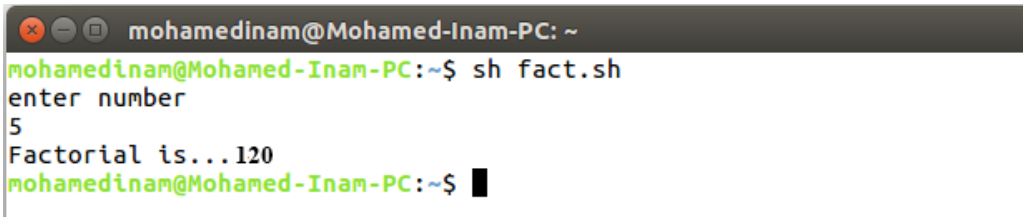
AIM:

To write a shell program to find factorial of a number.

SHELL SCRIPT:

```
echo "enter number"
read n
i=1
f=1
 while [ $i -lt $n ]
do
 f=`expr $f \* $i`
 i=`expr $i + 1`
done
echo "Factorial is...$f"
```

OUTPUT:



RESULT:

Thus the shell program to find factorial of a number is written and executed successfully.

EX NO: 4	CPU SCHEDULING ALGORITHMS
DATE:	

A. FIRST COME FIRST SERVE

AIM:

To write the program to implement CPU & scheduling algorithm for first come first serve scheduling.

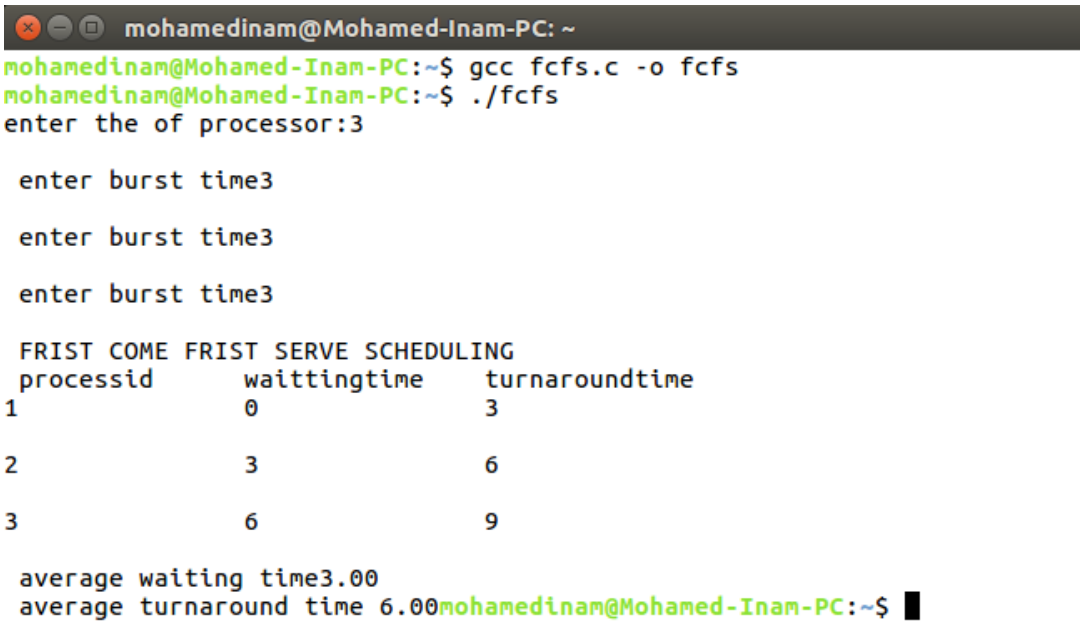
ALGORITHM:

1. Start the program.
2. Get the number of processes and their burst time.
3. Initialize the waiting time for process 1 and 0.
4. Process for(i=2;i<=n;i++),wt.p[i]=p[i-1]+bt.p[i-1].
5. The waiting time of all the processes is summed then average value time is calculated.
6. The waiting time of each process and average times are displayed
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,n,sum,wt,tat,twt,ttat;
int t[10];
float awt,atat;
printf("enter the of processer:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n enter burst time");
scanf("\n %d",&t[i]);
}
printf("\n FIRST COME FRIST SERVE SCHEDULING");
printf("\n processid\twaittingtime\tturnaroundtime\n");
printf("1\t\t0\t\t%d\n",t[0]);
sum=0;
twt=0;
ttat=t[0];
for(i=1;i<n;i++)
{
sum+=t[i-1];
wt=sum;
tat=sum+t[i];
twt=twt+wt;
ttat=ttat+tat;
printf("\n%d\t\t%d\t\t%d",i+1,wt,tat);
printf("\n");
}
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\n average waiting time%4.2f",awt);
printf("\n average turnaround time %4.2f",atat);
}
```

OUTPUT:



RESULT:

Thus the FCFS process scheduling program was executed and verified successfully.

EX NO: 4B	SHORTEST JOB FIRST – SCHEDULING ALGORITHM
DATE:	

AIM:

To write a program to implement cpu scheduling algorithm for shortest job first scheduling.

ALGORITHM:

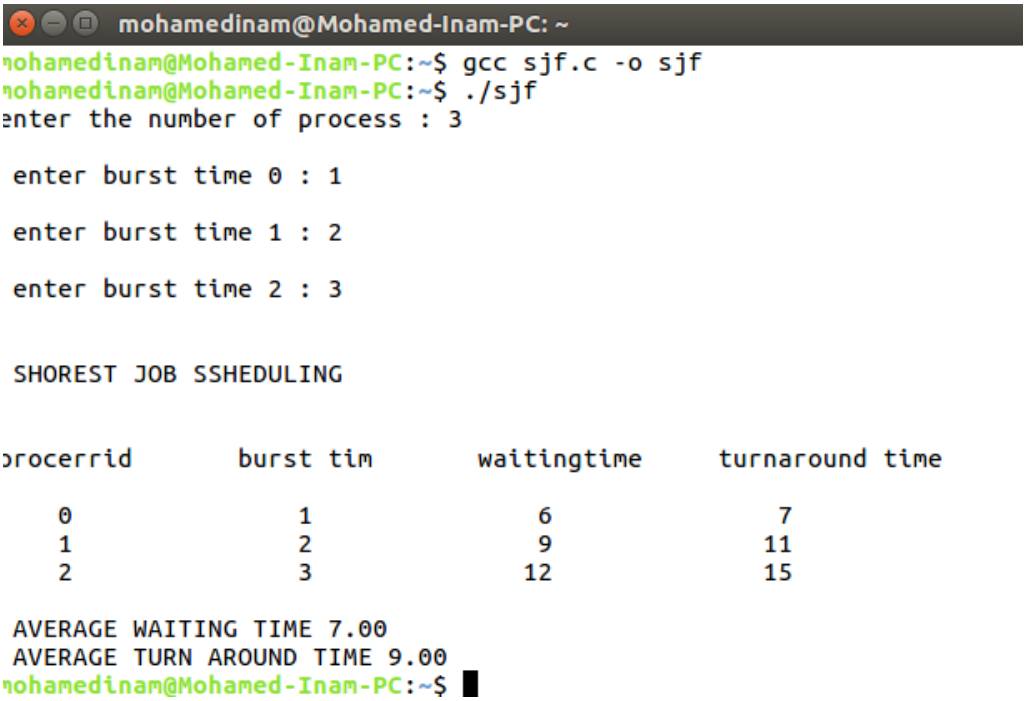
1. Start the program.
2. Get the number of processes and their burst time
3. Initialize the waiting time for process 1 as 0.
4. The processes are stored according to the burst time.
5. The waiting time for the processes are calculated a follows:
   for(i=2;i<=n;i++).wt.p[i]=p[i=1]+bt.p[i-1].
6. The waiting time of all the processes summed and then the average time is calculate
7. The waiting time of each processes and average time are displayed.
8. Stop the program.

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,j,k,n,sum,wt[10],tt[10],twt,ttat;
int t[10],p[10];
float awt,atat;
printf("enter the number of processer");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n enter burst time %d",i);
scanf("\n%d",&t[i]);
}
for(i=0;i<n;i++)
p[i]=i;
for(i=0;i<n;i++)
{
for(k=0;k<i+1;k++)
{
if(t[i]>t[k])
{
int temp;
temp=t[i];
t[i]=t[k];
t[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}}
printf("\n\n SHOREST JOB SSHEDULING\n\n");
printf("\nprocerrid\tburst tim\twaitingtime\tturnaround time\n\n");
wt[0]=0;
for(i=0;i<n;i++)
{
sum=0;
for(k=0;k<i;k++)
{
wt[i]=sum+t[k];
sum=wt[i];
}
}
for(i=0;i<n;i++)
{
```

```
tt[i]=t[i]+wt[i];
}
for(i=0;i<n;i++)
{
wt[i]=sum+t[k];
sum=wt[i];
}
}
for(i=0;i<n;i++)
{
tt[i]=t[i]+wt[i];
}
for(i=0;i<n;i++)
{
printf("%5d\t\t%5d\t\t%5d\t\t%5d\n",p[i],t[i],wt[i],tt[i]);
}
twt=0;
ttat=t[0];
for(i=1;i<n;i++)
{
twt=twt+wt[i];
ttat=ttat+tt[i];
}
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\n AVERAGE WAITING TIME %4.2f",awt);
printf("\n AVERAGE TURN AROUND TIME %4.2f",atat);
}
}
```

OUTPUT:



RESULT:

Thus the SJF program was executed and verified successfully

EX NO: 4C	PRIORITY – SCHEDULING ALGORITHM
DATE:	

AIM:

To write a 'C' program to perform priority scheduling.

ALGORITHM:

1. Start the program.
2. Read burst time, waiting time, turn the around time and priority.
3. Initialize the waiting time for process 1 and 0.
4. Based up on the priority process are arranged
5. The waiting time of all the processes is summed and then the average waiting time
6. The waiting time of each process and average waiting time are displayed based on the priority.
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,j,n,tat[10],wt[10],bt[10],pid[10],pr[10],t,twt=0,ttat=0;
float awt,atat;
printf("\n----------PRIORITY SCHEDULING----------\n");
printf("Enter the number of process: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
pid[i]=i;
printf("Enter the Burst time of Pid %d: ",i);
scanf("%d",&bt[i]);
printf("Enter the Priority of Pid %d: ",i);
scanf("%d",&pr[i]);
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(pr[i]>pr[j])
{
t=pr[i];
pr[i]=pr[j];
pr[j]=t;
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
t=pid[i];
pid[i]=pid[j];
pid[j]=t;
}
}
tat[0]=bt[0];
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
}
printf("\n ---------------------------------- \n");
printf("Pid \t Priority \t Burst time \t WaitingTime \tTurnAroundTime \n");
printf("\n ---------------------------------- \n");
for(i=0;i<n;i++)
{
```

```
printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d",pid[i],pr[i],bt[i],bt[i],wt[i],ta
t[i]);
}
for(i=0;i<n;i++)
{
ttat=ttat+tat[i];
twt=twt+wt[i];
}
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\n\n Avg.Waiting Time: %f\n Avg.Turn Around Time: %f\n",awt,atat);
}
```

OUTPUT:

```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ gcc priority.c -o priority
mohamedinam@Mohamed-Inam-PC:~$ ./priority

-----------PRIORITY SCHEDULING-----------
Enter the number of process: 4
Enter the Burst time of Pid 0: 2
Enter the Priority of Pid 0: 3
Enter the Burst time of Pid 1: 6
Enter the Priority of Pid 1: 2
Enter the Burst time of Pid 2: 4
Enter the Priority of Pid 2: 1
Enter the Burst time of Pid 3: 5
Enter the Priority of Pid 3: 7

 -------------------------------------
Pid      Priority        Burst time      WaitingTime     TurnAroundTime

 -------------------------------------

 2               1               4               0               4
 1               2               6               4               10
 0               3               2               10              12
 3               7               5               12              17

 Avg.Waiting Time: 6.500000
 Avg.Turn Around Time: 10.750000
mohamedinam@Mohamed-Inam-PC:~$
```

RESULT:

Thus the priority scheduling program was executed and verified successfully

EX NO: 4D	ROUND ROBIN – SCHEDULING ALGORITHM
DATE:	

AIM:

To write a program to implement CPU scheduling for Round Robin Scheduling.

ALGORITHM:

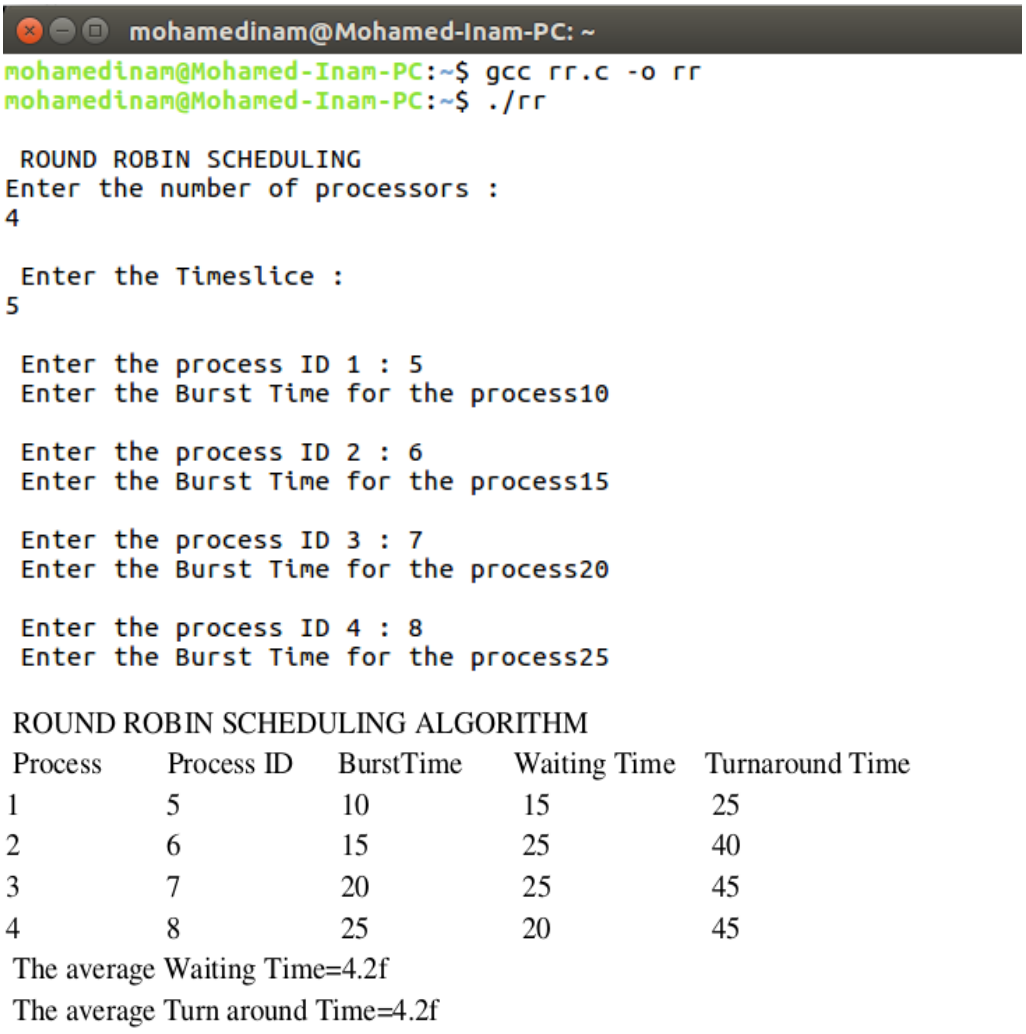
1. Get the number of process and their burst time.
2. Initialize the array for Round Robin circular queue as '0'.
3. The burst time of each process is divided and the quotients are stored on the round Robin array.
4. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.
5. The waiting time for each process and average times are displayed.
6. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.

PROGRAM:

```
#include<stdio.h>
void main()
{
int ts,pid[10],need[10],wt[10],tat[10],i,j,n,n1;
int bt[10],flag[10],ttat=0,twt=0;
float awt,atat;
printf("\n ROUND ROBIN SCHEDULING \n");
printf("Enter the number of processors \n");
scanf("%d",&n);
n1=n;
printf("\n Enter the Timeslice \n");
scanf("%d",&ts);
for(i=1;i<=n;i++)
{
printf("\n Enter the process ID %d",i);
scanf("%d",&pid[i]);
printf(" Enter the Burst Time for the process");
scanf("%d",&bt[i]);
need[i]=bt[i];
}
for(i=1;i<=n;i++)
{
flag[i]=1;
wt[i]=0;
}
while(n!=0)
{
for(i=1;i<=n;i++)
{
if(need[i]>=ts)
{
for(j=1;j<=n;j++)
{
if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
wt[j]+=ts;
}
need[i]=0;
n--;
flag[i]=0;
}
}
twt=0;
ttat=0;
for(i=1;i<=n1;i++)
{
tat[i]=wt[i]+bt[i];
twt=twt+wt[i];
ttat=ttat+tat[i];
}}
awt=(float)twt/n1;
atat=(float)ttat/n1;
```

```
printf("\n\n ROUND ROBIN SCHEDULING ALGORITHM \n\n");
printf("\n Process \t Process ID \t BurstTime \t Waiting Time \t Turnaround Time \n");
for(i=1;i<=n1;i++)
{
printf("\n %5d \t %5d \t\t %5d \t\t %5d \t\t %5d \n",i,pid[i],bt[i],wt[i],tat[i]);
}
printf("\n The average Waiting Time=%4.2f",awt);
printf("\n The average Turn around Time=%4.2f",atat);
}
```

OUTPUT:



RESULT:

Thus the Round Robin scheduling program was executed and verified successfully.

EX NO: 5	SEMAPHORES
DATE:	

AIM:

To write a C program to implement the Producer & consumer Problem (Semaphore).

ALGORITHM:

1: The Semaphore mutex, full & empty are initialized.

2: In the case of producer process

i) Produce an item in to temporary variable.

ii) If there is empty space in the buffer check the mutex value for enter into the critical section.

iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.

3: In the case of consumer process

i) It should wait if the buffer is empty

ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer

iii) Signal the mutex value and reduce the empty value by 1.

iv) Consume the item.

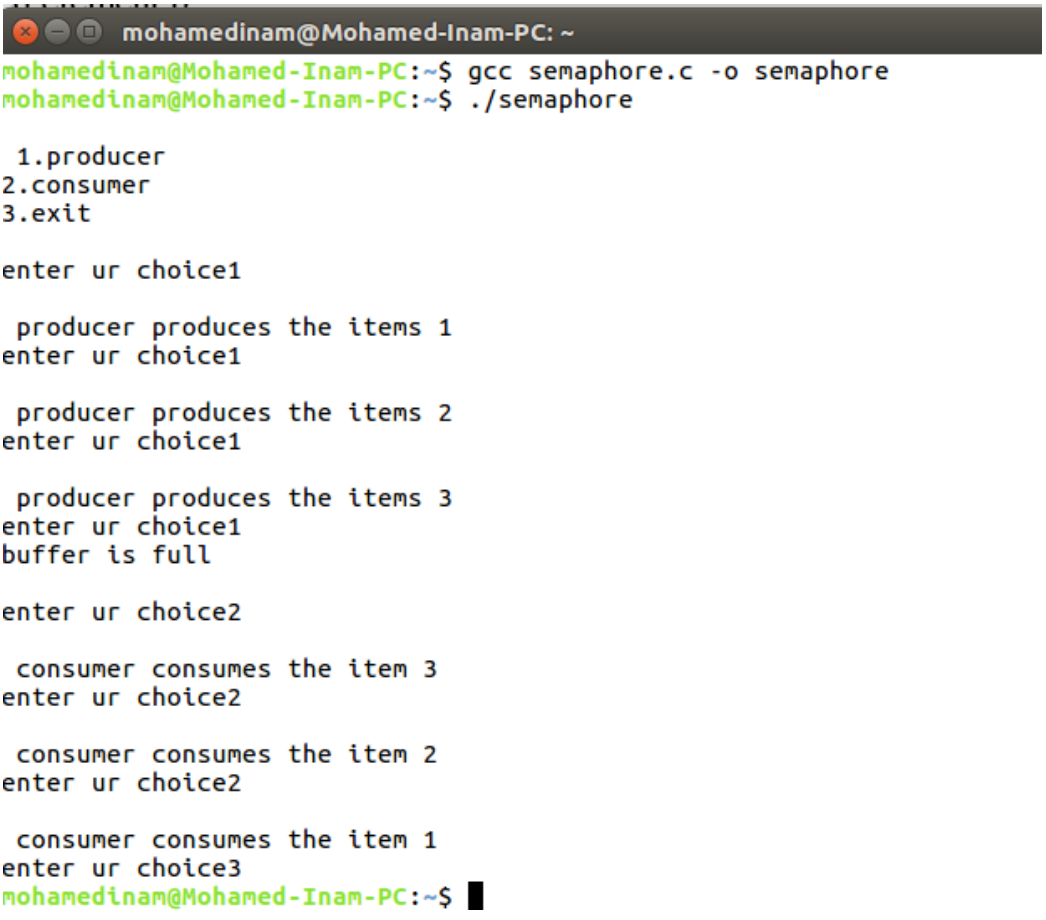
4: Print the result

PROGRAM

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n 1.producer\n2.consumer\n3.exit\n");
while(1) {
printf(" \nenter ur choice");
scanf("%d",&n);
switch(n)
{
case 1:if((mutex==1)&&(empty!=0))
producer();
else
printf("buffer is full\n");
break;
case 2:if((mutex==1)&&(full!=0))
consumer();
else
printf("buffer is empty");
break;
case 3:exit(0);
break;
}
}
}
int wait(int s)
{
        return(--s);
}
int signal(int s)
{
        return (++s);
 }
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
```

```
 x++;
printf("\n producer produces the items %d",x);
 mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes the item %d",x);
x--;
mutex=signal(mutex);
}
```

OUTPUT:



RESULT:

Thus the program for Producer & consumer Problem using Semaphore was executed successfully

EX NO: 6A	SHARED MEMORY
DATE:	

AIM

To demonstrate communication between process using shared memory.

ALGORITHM:

Server

1.Initialize size of sharedmemory *shmsize* to 27.
2.Initialize *key* to 2013(somerandomvalue).
3.Create a sharedmemory segment using shmget with *key* & IPC_CREAT as parameter.
a.If shared memory identifier *shmid*is-1,thenstop.
4. Display *shmid*.
5.Attach server process to the sharedmemory using shmmat with *shmid* as parameter.
a.If pointer to the sharedmemory is not obtained,then stop.
6.Clear contents of the shared region us in gmemsetfunction.
7.Write a–z onto the sharedmemory.
8.Wait till client reads the sharedmemory contents
9.Detach process from the sharedmemory using shmdt systemcall.
10.Remove sharedmemory from the system using shmctl with IPC_RMID argument
11.Stop

Client

1.Initialize sizeof sharedmemory *shmsize* to 27.
2.Initialize *key* to 2013(samevalueasinserver).
3.Obtain access to the same sharedmemory segment using same *key*.
a.If obtained then display the *shmid* else print"Servernotstarted"
4. Attach client process to the sharedmemory using shmmat with *shmid* as parameter.
a.If pointer to the sharedmemory is not obtained,then stop.
5.Read contents of sharedmemory and print it.
6.After reading, modify the first character of sharedmemory to '*'
7.Stop

PROGRAM:

//SERVER//

```
/*Sharedmemoryserver-shms.c*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/un.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#defineshmsize27

main()
{
    charc;
    intshmid;
    key_tkey=        2013;
    char*shm,*s;

    if((shmid=shmget(key,shmsize,IPC_CREAT|0666))<0)
    {
        perror("shmget");
        exit(1);
    }
    printf("Sharedmemoryid:%d\n",shmid);

    if((shm=shmat(shmid,NULL,0))==(char*)-1)
    {
        perror("shmat");
        exit(1);
    }

    memset(shm,0,shmsize);
    s=shm;
    printf("Writing(a-z)ontosharedmemory\n");
    for(c='a';c<='z';c++)
        *s++=c;
    *s='\0';

    while(*shm!='*');
    printf("Clientfinishedreading\n");

    if(shmdt(shm)!=0)
        fprintf(stderr,"Couldnotclosememorysegment.\n");

    shmctl(shmid,IPC_RMID,0);
}
```

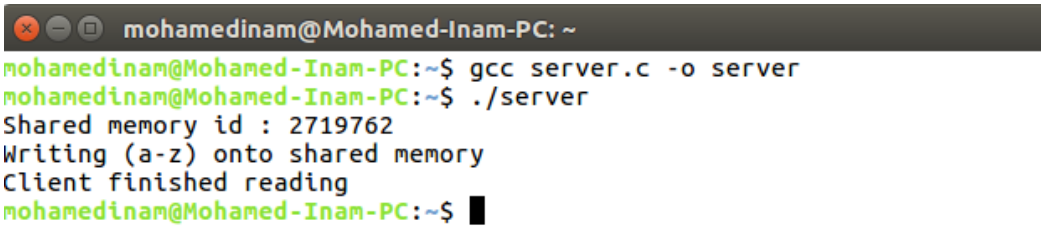
//CLIENT//

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

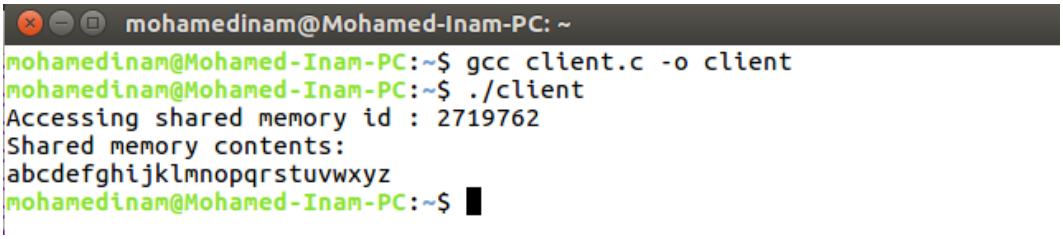
```
#include<sys/ipc.h>
#include<sys/shm.h>

#defineshmsize27

main()
{
    int shmid;
    key_tkey=2013;
    char*shm,*s;

    if((shmid=shmget(key,shmsize,0666))<0)
    {
        printf("Servernotstarted\n");
        exit(1);
    }
    else
        printf("Accessingsharedmemoryid:%d\n",shmid);

    if((shm=shmat(shmid,NULL,0))==(char*)-1)
    {
        perror("shmat");
        exit(1);
    }

    printf("Sharedmemorycontents:\n");
    for(s=shm;*s!='\0';s++)
        putchar(*s);
    putchar('\n');

    *shm='*';
}
```

OUTPUT:

//SERVER



//CLIENT



RESULT:

Thuscontentswrittenontosharedmemorybytheserverprocessisreadbytheclient process.

EX NO: 6B	IPC – CHAT MESSAGING
DATE:	

AIM

To exchange message between server and client using message queue.

ALGORITHM

Server

1. Declare a structure *mesgq* with *type* and *text* fields.
2. Initialize *key* to 2013 (some random value).
3. Create a message queue using msgget with *key* & IPC_CREAT as parameter.
   a. If message queue cannot be created then stop.
4. Initialize the message *type* member of *mesgq* to 1.
5. Do the following until user types Ctrl+D
   a. Get message from the user and store it in *text* member.
   b. Delete the new line character in *text* member.
   c. Place message on the queue using msgsend for the client to read.
   d. Retrieve the response message from the client using msgrcv function
   e. Display the *text* contents.
6. Remove message queue from the system using msgctl with IPC_RMID as parameter.
7. Stop

Client

1. Declare a structure *mesgq* with *type* and *text* fields.
2. Initialize *key* to 2013(same value as in server).
3. Open the message queue using msgget with *key* as parameter.
   a. If message queue cannot be opened then stop.
4. Do while the message queue exists
   a. Retrieve the response message from the server using msgrcv function
   b. Display the *text* contents.
   c. Get message from the user and store it in *text* member.
   d. Delete the new line character in *text* member.
   e. Place message on the queue using msgsend for the server to read.
5. Print "Server Disconnected".
6. Stop.

PROGRAM

Server

```
/*Serverchatprocess-srvmsg.c*/
#include<stdio.h>
 #include<stdlib.h>
 #include<string.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
structmesgq
 {
      longtype;
      chartext[200];
 }mq;

 main()
 {
      int msqid,len;
      key_tkey=2013;

      if((msqid=msgget(key,0644|IPC_CREAT))==-1)
      {
           perror("msgget");
           exit(1);
      }

      printf("Enter text to quit:\n");
      mq.type=1;

      while(fgets(mq.text,sizeof(mq.text),stdin)!=NULL)
      {
           len=strlen(mq.text);
         if(mq.text[len-1]=='\n') mq.text[len-1]='\0';
                msgsnd(msqid,&mq,len+1,0);

           msgrcv(msqid,&mq,sizeof(mq.text),0,0);
           printf("FromClient:\"%s\"\n",mq.text);
      }
      msgctl(msqid,IPC_RMID,NULL);
 }
```

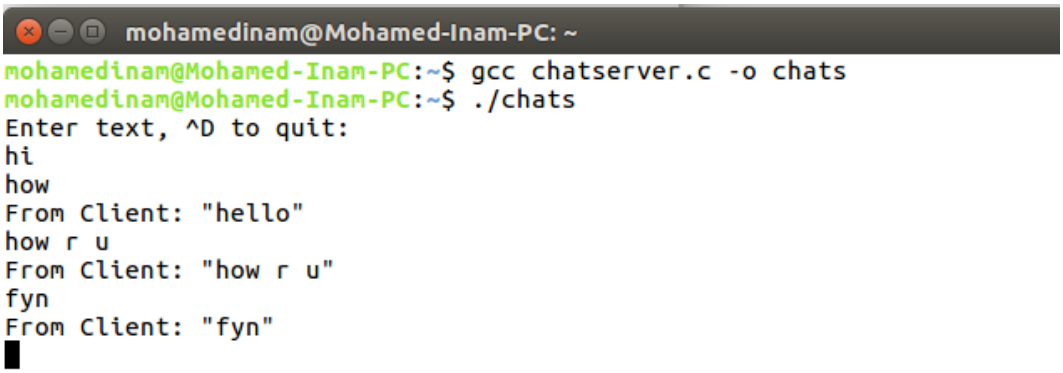
Client

```
 /*Clientchatprocess-climsg.c*/

 #include<stdio.h>
 #include<stdlib.h>
 #include<string.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
```

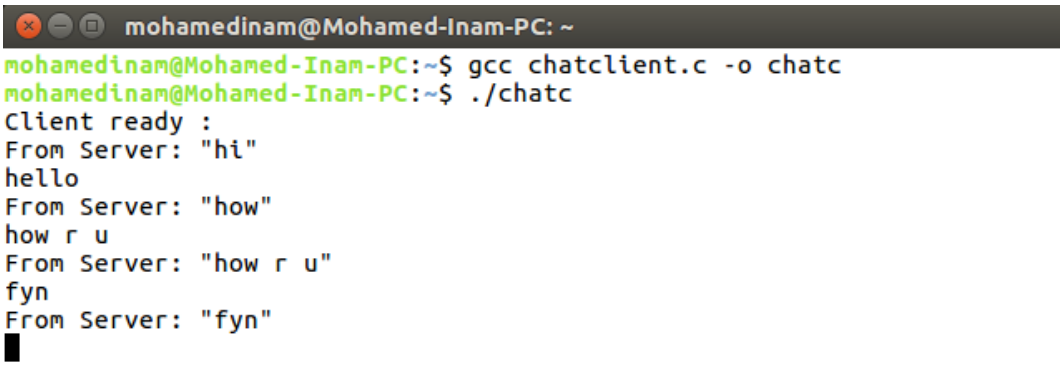
```
structmesgq
{
    longtype;
    chartext[200];
}mq;

main()
{
    int msqid,len;
    key_tkey=2013;

    if((msqid=msgget(key,0644))==-1)
    {
        printf("Servernotactive\n");
        exit(1);
    }

    printf("Clientready:\n");
    while(msgrcv(msqid,&mq,sizeof(mq.text),0,0)!=-1)
    {
        printf("FromServer:\"%s\"\n",mq.text);

        fgets(mq.text,sizeof(mq.text),stdin);
        len=strlen(mq.text);
      if(mq.text[len-1]=='\n') mq.text[len-1]='\0';
            msgsnd(msqid,&mq,len+1,0);
    }
    printf("ServerDisconnected\n");
}
```

OUTPUT:

//CHAT SERVER



//CHAT CLIENT



RESULT

Thus chat session between client and server was done using message queue.

EX NO: 7A	DEADLOCK AVOIDANCE – BANKER'S ALGORITHM
DATE:	

AIM:

To implement deadlock avoidance by using Banker's Algorithm.

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type Rj is available.
- Max: If max[i, j]=k, Pi may request at most k instances resource Rj.
- Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj
- Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,

Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   - Finish[i] =False
   - Need<=Work

   If no such I exists go to 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to 2. Otherwise raise an error condition.
2. if Request<=Available go to 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
   Available=Available-Request I;
   Allocation I =Allocation+Request I;
   Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. Or not we allow the request.
10. Stop the program.

PROGRAM:

/* BANKER'S ALGORITHM */

```
#include<stdio.h>

struct da
{
   int max[10],a1[10],need[10],before[10],after[10];
}p[10];
 void main()
{        int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
          clrscr();
          printf("\n ENTER THE NO. OF PROCESSES:");
           scanf("%d",&n);
          printf("\n ENTER THE NO. OF RESOURCES:");
           scanf("%d",&r);
 for(i=0;i<n;i++)
 {
          printf("PROCESS %d \n",i+1);
   for(j=0;j<r;j++)
   {
          printf("MAXIMUM VALUE FOR RESOURCE %d:",j+1);
          scanf("%d",&p[i].max[j]);
   }
  for(j=0;j<r;j++)
   {
          printf("ALLOCATED FROM RESOURCE %d:",j+1);
           scanf("%d",&p[i].a1[j]);
           p[i].need[j]=p[i].max[j]-p[i].a1[j];
   }
 }
          for(i=0;i<r;i++)
          {
                   printf("ENTER TOTAL VALUE OF RESOURCE %d:",i+1);
                   scanf("%d",&tot[i]);
          }
          for(i=0;i<r;i++)
          {
                   for(j=0;j<n;j++)
                            temp=temp+p[j].a1[i];
                            av[i]=tot[i]-temp;
                             temp=0;
          }

printf("\n\t RESOURCES  ALLOCATED   NEEDED   TOTAL  AVAIL");
```

```
  for(i=0;i<n;i++)
  {
          printf("\n P%d \t",i+1);
   for(j=0;j<r;j++)
           printf("%d",p[i].max[j]);
           printf("\t");
   for(j=0;j<r;j++)
          printf("%d",p[i].a1[j]);
          printf("\t");
   for(j=0;j<r;j++)
          printf("%d",p[i].need[j]);
          printf("\t");
   for(j=0;j<r;j++)
   {
            if(i==0)
            printf("%d",tot[j]);
   }
printf("    ");
   for(j=0;j<r;j++)
   {
      if(i==0)
      printf("%d",av[j]);
   }
 }
printf("\n\n\t AVAIL  BEFORE\T AVAIL AFTER ");
  for(l=0;l<n;l++)
  {
   for(i=0;i<n;i++)
   {
           for(j=0;j<r;j++)
           {
            if(p[i].need[j] >av[j])
            cn++;
            if(p[i].max[j]==0)
            cz++;
           }
if(cn==0 && cz!=r)
   {
     for(j=0;j<r;j++)
     {
          p[i].before[j]=av[j]-p[i].need[j];
          p[i].after[j]=p[i].before[j]+p[i].max[j];
          av[j]=p[i].after[j];
          p[i].max[j]=0;
     }
```

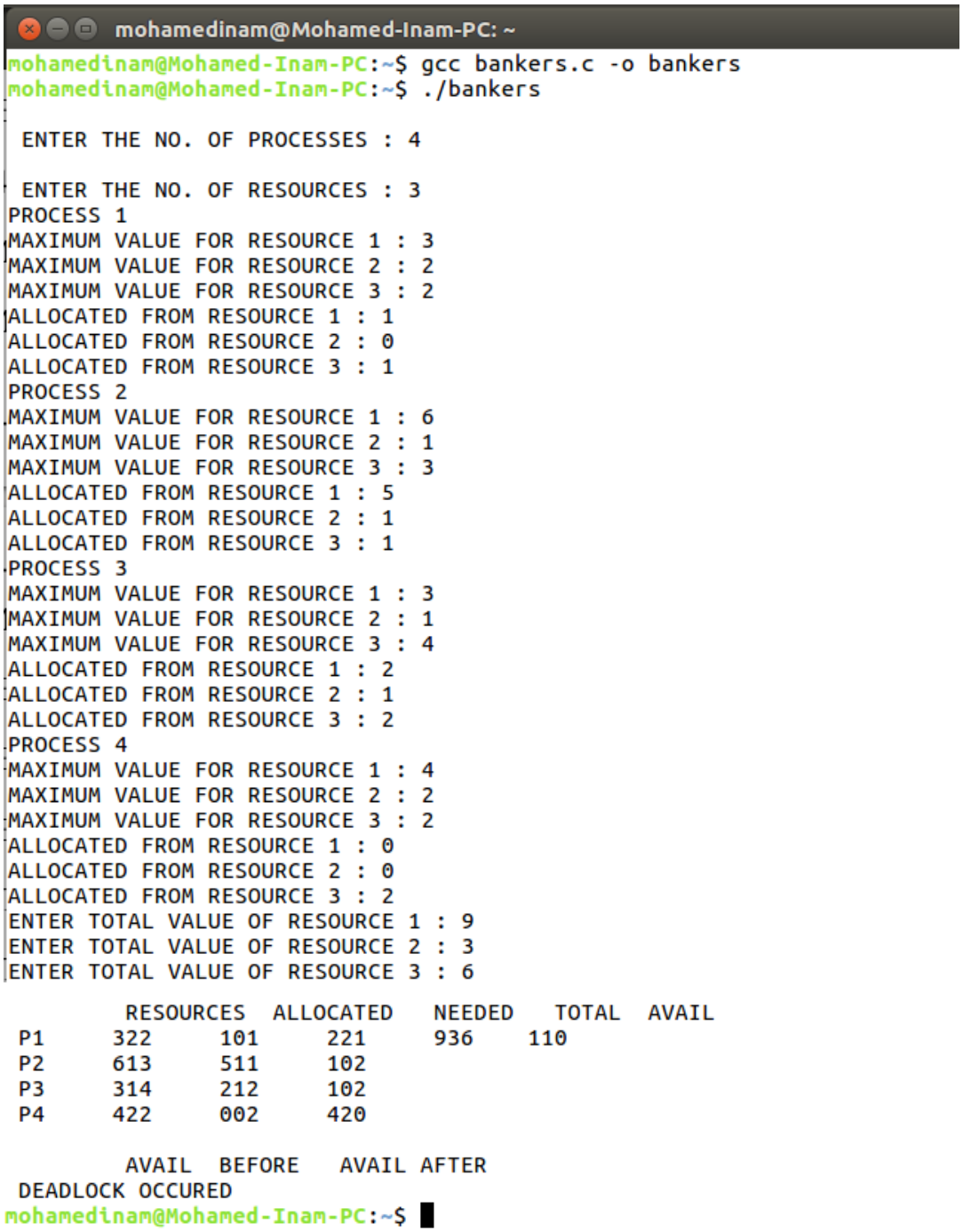
```
printf("\n P %d \t",i+1);
      for(j=0;j<r;j++)
           printf("%d",p[i].before[j]);
          printf("\t");
      for(j=0;j<r;j++)
           printf("%d",p[i].after[j]);
           cn=0;
           cz=0;
           c++;
           break;
   }
   else
   {
     cn=0;cz=0;
   }
  }
}
 if(c==n)
          printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");
    else
          printf("\n DEADLOCK OCCURED");

}
```

OUTPUT:

//RUN: NO deadlock

```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ gcc bankers.c -o bankers
mohamedinam@Mohamed-Inam-PC:~$ ./bankers

 ENTER THE NO. OF PROCESSES : 4

 ENTER THE NO. OF RESOURCES : 3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1 : 3
MAXIMUM VALUE FOR RESOURCE 2 : 2
MAXIMUM VALUE FOR RESOURCE 3 : 2
ALLOCATED FROM RESOURCE 1 : 1
ALLOCATED FROM RESOURCE 2 : 0
ALLOCATED FROM RESOURCE 3 : 0
PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1 : 6
MAXIMUM VALUE FOR RESOURCE 2 : 1
MAXIMUM VALUE FOR RESOURCE 3 : 3
ALLOCATED FROM RESOURCE 1 : 5
ALLOCATED FROM RESOURCE 2 : 1
ALLOCATED FROM RESOURCE 3 : 1
PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1 : 3
MAXIMUM VALUE FOR RESOURCE 2 : 1
MAXIMUM VALUE FOR RESOURCE 3 : 4
ALLOCATED FROM RESOURCE 1 : 2
ALLOCATED FROM RESOURCE 2 : 1
ALLOCATED FROM RESOURCE 3 : 1
PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1 : 4
MAXIMUM VALUE FOR RESOURCE 2 : 2
MAXIMUM VALUE FOR RESOURCE 3 : 2
ALLOCATED FROM RESOURCE 1 : 0
ALLOCATED FROM RESOURCE 2 : 0
ALLOCATED FROM RESOURCE 3 : 2
ENTER TOTAL VALUE OF RESOURCE 1 : 9
ENTER TOTAL VALUE OF RESOURCE 2 : 3
ENTER TOTAL VALUE OF RESOURCE 3 : 6

         RESOURCES  ALLOCATED   NEEDED   TOTAL  AVAIL
 P1      322      100      222      936    112
 P2      613      511      102
 P3      314      211      103
 P4      422      002      420

         AVAIL  BEFORE   AVAIL AFTER
 P 2     010      623
 P 1     401      723
 P 3     620      934
 P 4     514      936
 THE ABOVE SEQUENCE IS A SAFE SEQUENCE
mohamedinam@Mohamed-Inam-PC:~$
```

//RUN2: Deadlock occurs



RESULT

Thus the banker's algorithm is implemented successfully for Deadlock avoidance & Dead Lock Prevention.

EX NO: 7B	DEADLOCK DETECTION – BANKER'S ALGORITHM
DATE:	

AIM

To implement deadlock detection by using Banker's Algorithm.

ALGORITHM:

1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vector**W** to equal the Available vector.
3. Find an index*i* such that process*i* is currently unmarked and the*i* th row of**Q** is less than or equal to **W** . That is,*Q ik … Wk*, for 1 *… k … m* . If no such row is found, terminate the algorithm.
4. If such a row is found, mark process*i* and add the corresponding row of the allocation matrix to **W** . That is, set*Wk = Wk* + A*ik*, for 1 *… k … m* . Return to step 3.

PROGRAM:

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];
printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
printf("\nEnter the request matrix:");

for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];
}
}
//marking processes with zero allocation
for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
```

```
      else
        break;
    }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail

for(j=0;j<nr;j++)
    w[j]=avail[j];

//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
   for(j=0;j<nr;j++)
    {
      if(request[i][j]<=w[j])
        canbeprocessed=1;
      else
         {
         canbeprocessed=0;
         break;
          }
     }
if(canbeprocessed)
{
mark[i]=1;

for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}

//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;
if(deadlock)
printf("\n Deadlock detected");
else
```

```
printf("\n No Deadlock possible");
}
```

OUTPUT:

```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detected
--------------------------------
```

RESULT:

Thus the banker's algorithm is implemented successfully for Deadlock detection.

EX NO: 8A	IMPLEMENTATION OF THREADING AND SYNCHRONIZATION APPLICATIONS
DATE:	

AIM:

To implement threading and synchronization techniques using c language.

ALGORITHM:

Step 1: Start the program.
Step 2: Identify the thread by an id called ThreadId.
Step 3: Represent the thread id by the type pthread_t.
Step 4: Include the header file "#include<pthread.h>" to access the thread functions.
Step 5: This function is used to create a thread pthread_create.
Step 6: If the thread is created successfully, return value will be zero, Otherwise pthread_create will return an error number of type integer.
Step 7: Stop the program.

PROGRAM:

```
#include <stdio.h>
#include <pthread.h>

/*thread function definition*/
void* threadFunction(void* args)
{
   while(1)
   {
      printf("I am threadFunction.\n");
   }
}
int main()
{
   /*creating thread id*/
   pthread_t id;
   int ret;

   /*creating thread*/
   ret=pthread_create(&id,NULL,&threadFunction,NULL);
   if(ret==0){
      printf("Thread created successfully.\n");
   }
   else{
      printf("Thread not created.\n");
      return 0; /*return from main*/
   }

   while(1)
   {
      printf("I am main function.\n");
   }

   return 0;
}
```

OUTPUT:

```
Thread created successfully.
I am threadFunction.
I am threadFunction.
I am threadFunction.
I am threadFunction.
…
…
I am threadFunction.
I am main function.
I am main function.
I am main function.
I am main function.
…
…
I am main function.
I am threadFunction.

… and so on.
```

RESULT:

Thus to implement threading and synchronization techniques using c language has been executed and verified successfully.

EX NO: 8B	MEMORY ALLOCATION METHODS FOR FIXED PARTITION
DATE:	

I. FIRST FIT ALLOCATION

AIM:

To allocate memory requirements for processes using first fit allocation.

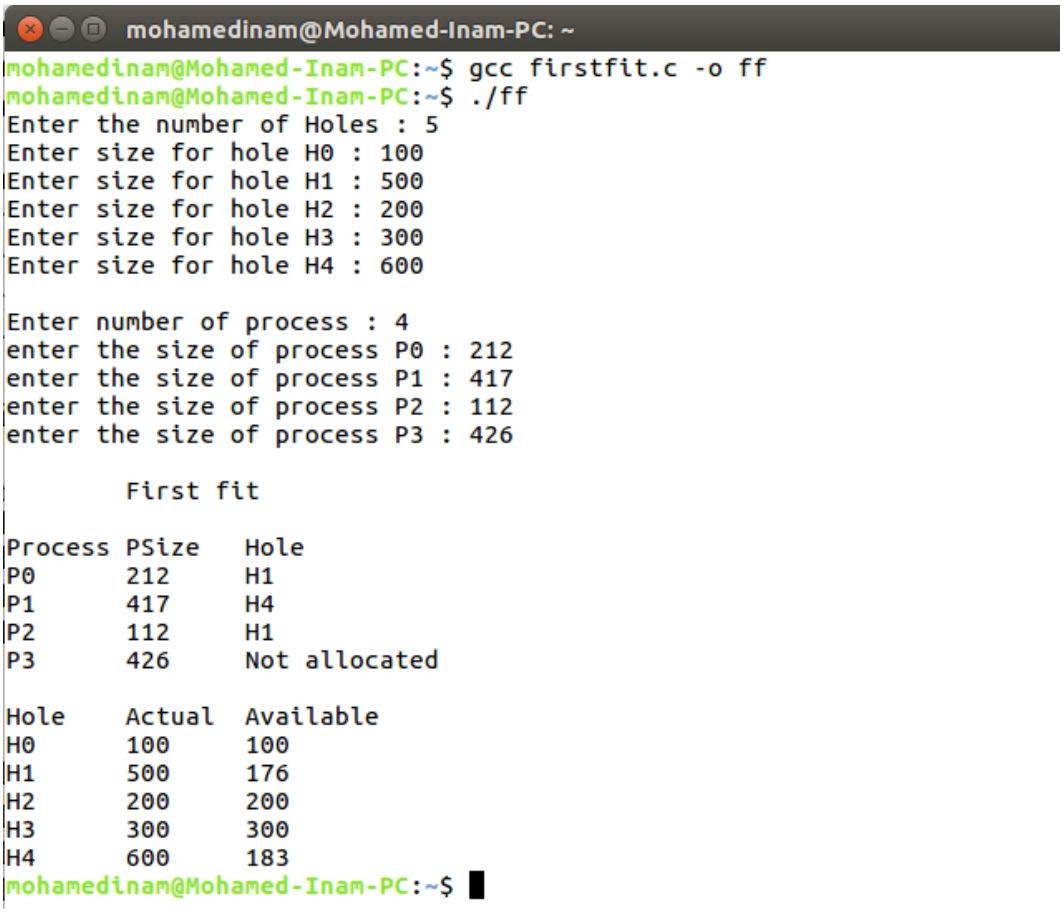
ALGORITHM:

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
   a. If hole size>process size then
   i. Mark process as allocated to that hole.
   ii. Decrement hole size by process size.
   b. Otherwise check the next from the set of holes
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop the program.

PROGRAM

```
#include<stdio.h>
struct process
{
int size; int flag; int holeid;
}p[10];
struct hole
{
int size;
int actual;
}h[10];
main()
{
int i,np,nh,j;
printf("EnterthenumberofHoles:");
scanf("%d",&nh);
for(i=0;i<nh;i++){
printf("EntersizeforholeH%d:",i);
scanf("%d",&h[i].size);
h[i].actual=h[i].size;
}
printf("\nEnternumberofprocess:");
scanf("%d",&np);
for(i=0;i<np;i++){
printf("enterthesizeofprocessP%d:",i);
scanf("%d",&p[i].size);
p[i].flag=0;
}
for(i=0;i<np;i++){
for(j=0;j<nh;j++){
if(p[i].flag!=1){
if(p[i].size<=h[j].size){
p[i].flag=1; p[i].holeid=j; h[j].size-=p[i].size;
}}}}

printf("\n\tFirstfit\n");
printf("\nProcess\tPSize\tHole");
for(i=0;i<np;i++){
if(p[i].flag!=1)
printf("\nP%d\t%d\tNotallocated",i,p[i].size);
else
printf("\nP%d\t%d\tH%d",i,p[i].size,p[i].holeid);
}
printf("\n\nHole\tActual\tAvailable");
for(i=0;i<nh;i++)
printf("\nH%d\t%d\t%d",i,h[i].actual,h[i].size);
printf("\n");
}
```

OUTPUT:



RESULT:

Thus processes were allocated memory using first fit method.

II. WORST FIT ALLOCATION

AIM:

To allocate memory requirements for processes using worst fit allocation.

ALGORITHM:

Step 1: Start the program.
Step 2: Input memory blocks and processes with sizes.
Step 3: Initialize all memory blocks as free.
Step 4: Start by picking each process and find the
maximum block size that can be assigned to
current process i.e., find max(bockSize[1],
blockSize[2],.....blockSize[n]) >
processSize[current], if found then assign
it to the current process.
Step 5: If not then leave that process and keep checking
the further processes.
Step 6: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

RESULT

Thus processes were allocated memory using worst fit method.

III. BEST FIT ALLOCATION

AIM:

To allocate memory requirements for processes using best fit allocation.

ALGORITHM:

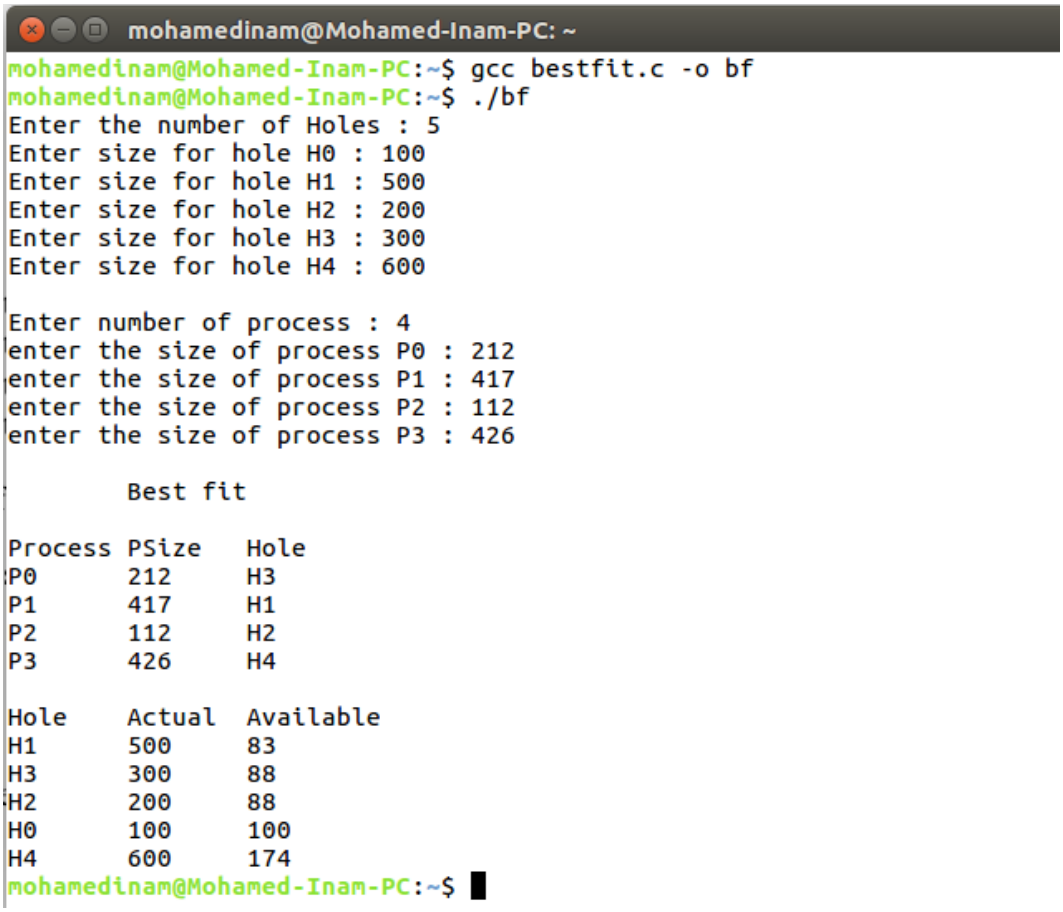
1. Declare structures hole *and* process *to* hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
   a. Sort the holes according to their sizes in ascending order
   b. If hole size>process size then
      i. Mark process as allocated to that hole.
      ii. Decrement hole size by process size.
   c. Otherwise check the next from the set of sorted hole.
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

PROGRAM:

```
#include<stdio.h>

struct process
{
int size;
int flag;
int holeid;
}p[10];

struct hole
{
int hid;
int size;
int actual;
}h[10];

main()
{
int i,np,nh,j;
void bsort(structhole[],int);
printf("Enter the number of Holes:");
scanf("%d",&nh);
for(i=0;i<nh;i++)
{
printf("Enter size for holeH%d:",i);
scanf("%d",&h[i].size);
h[i].actual=h[i].size;
h[i].hid=i;
}
printf("\nEnter number of process:");
scanf("%d",&np);
for(i=0;i<np;i++)
{
printf("enter the size of processP%d:",i);
scanf("%d",&p[i].size);
p[i].flag=0;
}

for(i=0;i<np;i++)
{
bsort(h,nh);
for(j=0;j<nh;j++)
{
if(p[i].flag!=1)
{
if(p[i].size<=h[j].size)
{
p[i].flag=1;
p[i].holeid=h[j].hid;
h[j].size-=p[i].size;
}
}
}
```

```
}

printf("\n\tBestfit\n");
printf("\nProcess\tPSize\tHole");
for(i=0;i<np;i++)
{
if(p[i].flag!=1)
printf("\nP%d\t%d\tNotallocated",i,p[i].size);
else
printf("\nP%d\t%d\tH%d",i,p[i].size,p[i].holeid);
}

printf("\n\nHole\tActual\tAvailable");
for(i=0;i<nh;i++)
printf("\nH%d\t%d\t%d",h[i].hid,h[i].actual, h[i].size);
printf("\n");
}

Void bsort(structholebh[],intn)
{
struct holetemp;
int i,j;
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(bh[i].size>bh[j].size)
{
temp=bh[i];
bh[i]=bh[j];
bh[j]=temp;
}
}
}
}
```

OUTPUT:



RESULT:

Thus processes were allocated memory using best fit method.

EX NO: 9	PAGING TECHNIQUE OF MEMORY MANAGEMENT
DATE:	

AIM:

To implement the Memory management policy- Paging.

ALGORITHM:

Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into fixed - sized blocks.
Step 3: Frames – Physical memory is broken into fixed – sized blocks.
Step 4: Calculate the physical address using the following
Physical address = (Frame number * Frame size) + offset
Step 5: Display the physical address.
Step 6: Stop the process.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
struct pstruct
{
    int fno;
    int pbit;
}ptable[10];

int pmsize,lmsize,psize,frame,page,ftable[20],frameno;

void info()
{
    printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
    printf("\n\nEnter the Size of Physical memory: ");
    scanf("%d",&pmsize);
    printf("\n\nEnter the size of Logical memory: ");
    scanf("%d",&lmsize);
    printf("\n\nEnter the partition size: ");
    scanf("%d",&psize);
    frame = (int) pmsize/psize;
    page = (int) lmsize/psize;
    printf("\nThe physical memory is divided into %d no.of frames\n",frame);
    printf("\nThe Logical memory is divided into %d no.of pages",page);
}
void assign()
{
    int i;
    for (i=0;i<page;i++)
    {
    ptable[i].fno = -1;
    ptable[i].pbit= -1;
    }

    for(i=0; i<frame;i++)
            ftable[i] = 32555;
    for (i=0;i<page;i++)
    {
    printf("\n\nEnter the Frame number where page %d must be placed: ",i);
            scanf("%d",&frameno);
            ftable[frameno] = i;
            if(ptable[i].pbit == -1)
            {
                    ptable[i].fno = frameno;
                    ptable[i].pbit = 1;
```

```
                }
        }
        getch();
        printf("\n\nPAGE TABLE\n\n");
        printf("PageAddress  FrameNo. PresenceBit\n\n");
        for (i=0;i<page;i++)
                printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptable[i].pbit);
        printf("\n\n\n\tFRAME TABLE\n\n");
        printf("FrameAddress   PageNo\n\n");
        for(i=0;i<frame;i++)
                printf("%d\t\t%d\n",i,ftable[i]);
}

void cphyaddr()
{
        int laddr,paddr,disp,phyaddr,baddr;
        getch();
        printf("\n\n\n\tProcess to create the Physical Address\n\n");
        printf("\nEnter the Base Address: ");
        scanf("%d",&baddr);
        printf("\nEnter theLogical Address: ");
        scanf("%d",&laddr);
        paddr = laddr / psize;
        disp = laddr % psize;
         if(ptable[paddr].pbit == 1 )
                phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
        printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
        clrscr();
        info();
        assign();
        cphyaddr();
        getch();
}
```

OUTPUT:

```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ clear

mohamedinam@Mohamed-Inam-PC:~$ gcc paging.c -o paging
mohamedinam@Mohamed-Inam-PC:~$ ./paging

MEMORY MANAGEMENT USING PAGING
Enter the Size of Physical memory: 16
Enter the size of Logical memory: 8
Enter the partition size: 2

The physical memory is divided into 8 no.of frames
The Logical memory is divided into 4 no.of pages
Enter the Frame number where page 0 must be placed: 5
Enter the Frame number where page 1 must be placed: 6
Enter the Frame number where page 2 must be placed: 7
Enter the Frame number where page 3 must be placed: 2

PAGE TABLE
PageAddress  FrameNo. PresenceBit
0               5               1
1               6               1
2               7               1
3               2               1

        FRAME TABLE
FrameAddress   PageNo

0               32555
1               32555
2               3
3               32555
4               32555
5               0
6               1
7               2
        Process to create the Physical Address
Enter the Base Address: 1000

Enter theLogical Address: 3

The Physical Address where the instruction present: 1013mohamedinam@Mohamed-Inam-PC:~
$
```

RESULT:

Thus the Memory management policy- Paging isimplemented successfully.

EX NO: 10	PAGE REPLACEMENT ALGORITHMS
DATE:	

A. FIFO PAGE REPLACEMENT ALGORITHM

AIM:

To implement page replacement algorithms FIFO (First In First Out).

ALGORITHM:

1: Create a queue to hold all pages in memory

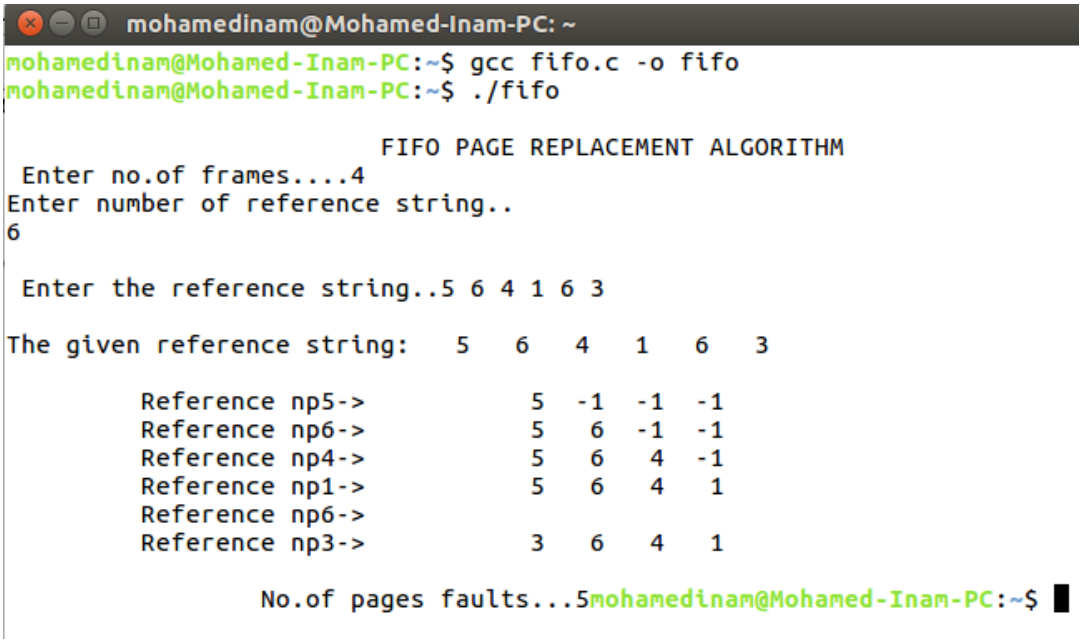
2: When the page is required replace the page at the head of the queue

3: Now the new page is inserted at the tail of the queue

PROGRAM:

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
printf("\n \t\t\t FIFO PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of frames....");
 scanf("%d",&nof);
printf("Enter number of reference string..\n");
 scanf("%d",&nor);
printf("\n Enter the reference string..");
 for(i=0;i<nor;i++)
 scanf("%d",&ref[i]);
printf("\nThe given reference string:");
 for(i=0;i<nor;i++)
 printf("%4d",ref[i]);
 for(i=1;i<=nof;i++)
          frm[i]=-1;
 printf("\n");
 for(i=0;i<nor;i++)
 {
  flag=0;
          printf("\n\t Reference np%d->\t",ref[i]);
  for(j=0;j<nof;j++)
  {
          if(frm[j]==ref[i])
          {
                   flag=1;
                    break;
          }}
    if(flag==0)
    {
           pf++;
          victim++;
           victim=victim%nof;
          frm[victim]=ref[i];
          for(j=0;j<nof;j++)
                   printf("%4d",frm[j]);
  } }
printf("\n\n\t\t No.of pages faults...%d",pf);
}
```

OUTPUT:



RESULT:

Thus the FIFO page replacement algorithm is implemented successfully.

EX NO: 10B	LRU PAGE REPLACEMENT ALGORITHM
DATE:	

AIM:

To implement page replacement algorithm LRU (Least Recently Used LRU (Lease Recently Used). Here we select the page that has not been used for the longest period of time.

ALGORITHM:

1: Create a queue to hold all pages in memory

2: When the page is required replace the page at the head of the queue

3: Now the new page is inserted at the tail of the queue

4: Create a stack

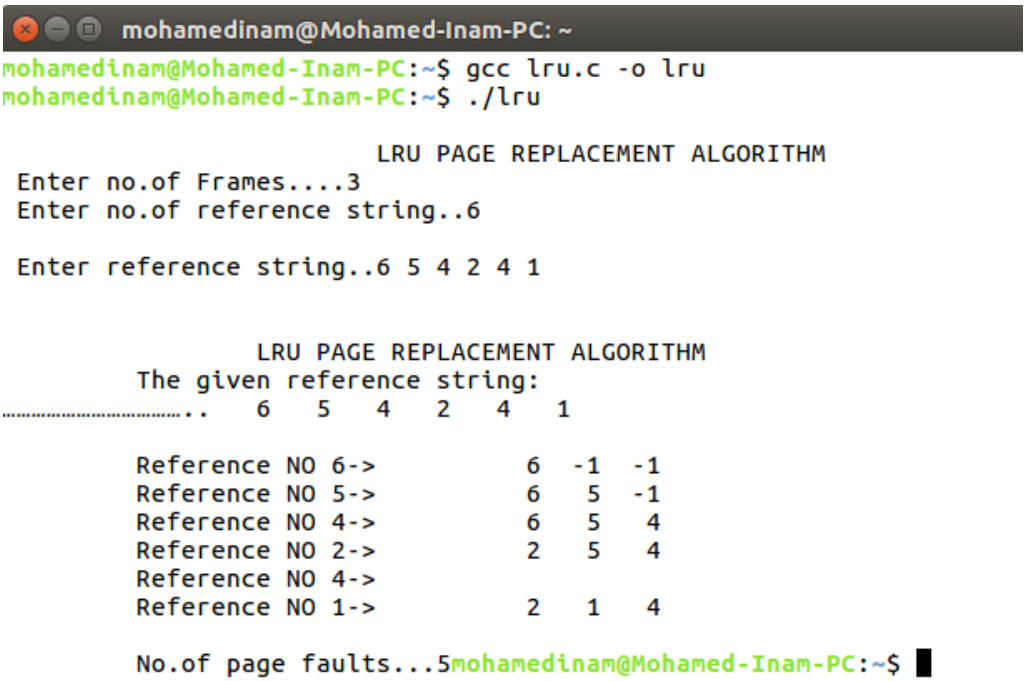
5: When the page fault occurs replace page present at the bottom of the stack

PROGRAM:

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();
void main()
{
clrscr();
printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of Frames....");
scanf("%d",&nof);
printf(" Enter no.of reference string..");
scanf("%d",&nor);

printf("\n Enter reference string..");
for(i=0;i<nor;i++)
        scanf("%d",&ref[i]);

printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
printf("\n\t The given reference string:");
printf("\n……..");

 for(i=0;i<nor;i++)
printf("%4d",ref[i]);

 for(i=1;i<=nof;i++)
 {
frm[i]=-1;
 lrucal[i]=0;
 }

 for(i=0;i<10;i++)
recent[i]=0;
printf("\n");

  for(i=0;i<nor;i++)
  {
flag=0;

printf("\n\t Reference NO %d->\t",ref[i]);
   for(j=0;j<nof;j++)
   {
     if(frm[j]==ref[i])
     {
        flag=1;
        break;
     }
   }
   if(flag==0)
   {
        count++;
     if(count<=nof)
```

```
            victim++;
        else
            victim=lruvictim();
            pf++;
            frm[victim]=ref[i];

        for(j=0;j<nof;j++)
            printf("%4d",frm[j]);
      }
             recent[ref[i]]=i;
   }
            printf("\n\n\t No.of page faults...%d",pf);
            getch();
}

int lruvictim()
{
  int i,j,temp1,temp2;

            for(i=0;i<nof;i++)
            {
                    temp1=frm[i];
                     lrucal[i]=recent[temp1];
            }
  temp2=lrucal[0];
  for(j=1;j<nof;j++)
  {
            if(temp2>lrucal[j])
            temp2=lrucal[j];
  }
  for(i=0;i<nof;i++)
            if(ref[temp2]==frm[i])

  return i;
  return 0;
}
```

OUTPUT:



RESULT:

Thus the LRU page replacement algorithm is implemented successfully.

EX NO: 10C	OPTIMAL (LFU) PAGE REPLACEMENT ALGORITHM
DATE:	

AIM:

To implement page replacement algorithms Optimal (The page which is not used for longest time)

ALGORITHM:

Optimal algorithm:

Here we select the page that will not be used for the longest period of time.

OPTIMAL:

1: Create an array

2: When the page fault occurs replace page that will not be used for the longest period of time.

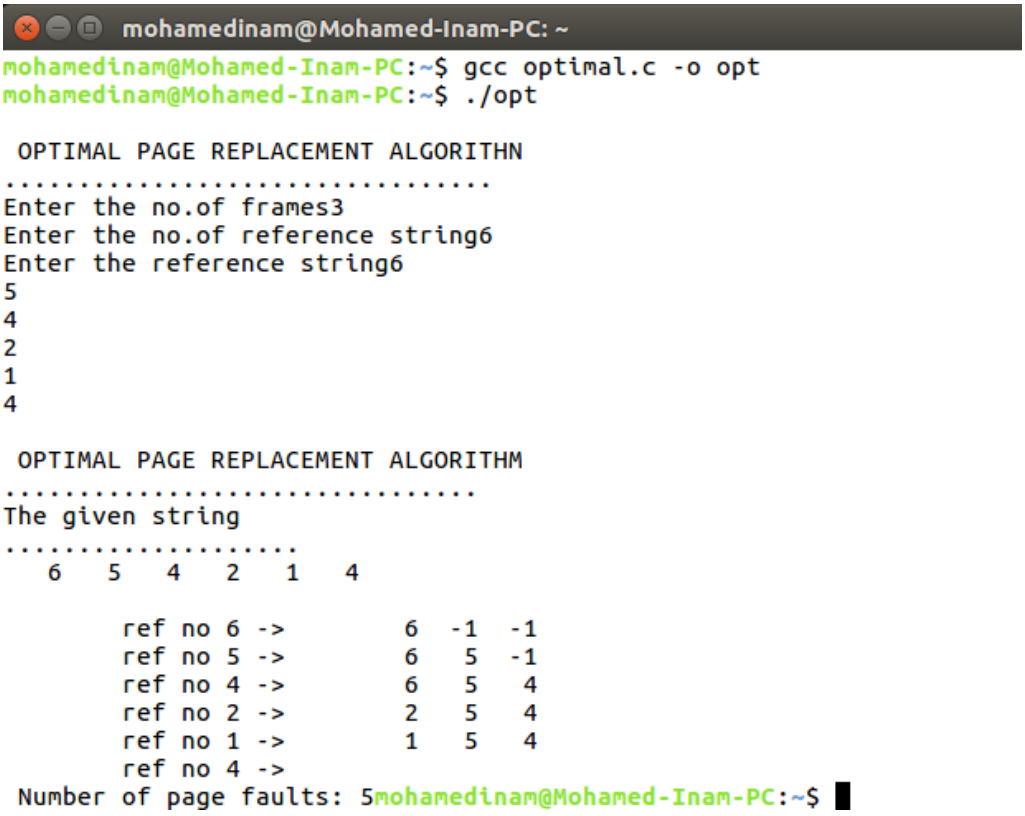
PROGRAM:

/*OPTIMAL (LFU) page replacement algorithm*/

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n.....");
printf("\nEnter the no.of frames");
  scanf("%d",&nof);
printf("Enter the no.of reference string");
  scanf("%d",&nor);
printf("Enter the reference string");
  for(i=0;i<nor;i++)
     scanf("%d",&ref[i]);
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
  printf("\n.....");
printf("\nThe given string");
  printf("\n.....\n");
  for(i=0;i<nor;i++)
     printf("%4d",ref[i]);
  for(i=0;i<nof;i++)
  {
     frm[i]=-1;
     optcal[i]=0;
  }
  for(i=0;i<10;i++)
        recent[i]=0;
        printf("\n");
  for(i=0;i<nor;i++)
  {
    flag=0;
printf("\n\tref no %d ->\t",ref[i]);

    for(j=0;j<nof;j++)
    {
        if(frm[j]==ref[i])
        {
          flag=1;
```

```
            break;
          }
     }
     if(flag==0) {
          count++;
          if(count<=nof)
             victim++;
          else
             victim=optvictim(i);
                  pf++;
                  frm[victim]=ref[i];

          for(j=0;j<nof;j++)
            printf("%4d",frm[j]);
     }
   }
printf("\n Number of page faults: %d",pf);
}
int optvictim(int index)
{
  int i,j,temp,notfound;

  for(i=0;i<nof;i++) {
    notfound=1;

    for(j=index;j<nor;j++)
         if(frm[i]==ref[j])
         {
            notfound=0;
            optcal[i]=j;
            break;
         }
    if(notfound==1)
            return i;
  }
  temp=optcal[0];
  for(i=1;i<nof;i++)
     if(temp<optcal[i])
            temp=optcal[i];
  for(i=0;i<nof;i++)
     if(frm[temp]==frm[i])
            return i;
 return 0;
}
```

OUTPUT:



RESULT:

Thus the LFU page replacement algorithm is implemented successfully.

EX NO: 11	VARIOUS FILE ORGANIZATION TECHNIQUES
DATE:	

A. IMPLEMENTATION OF SINGLE LEVEL DIRECTORY

AIM:

To write a C program to implement Single - Level directory structure in C

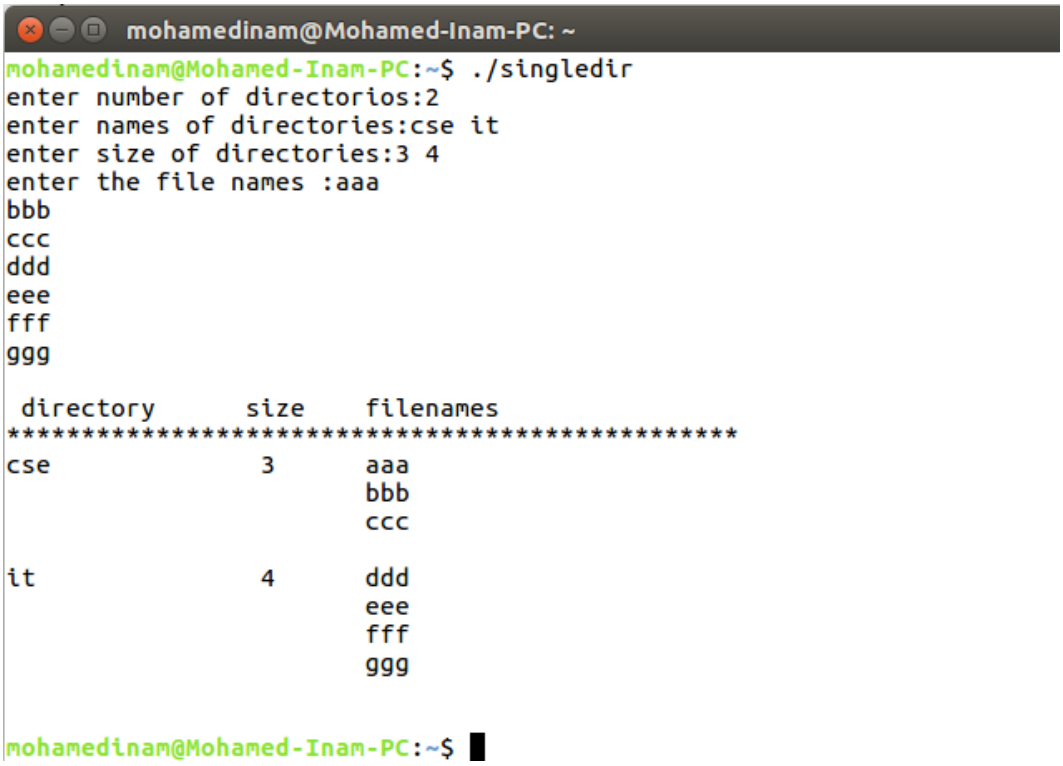
ALGORITHM:

1. Start
2. Declare the number, names and size of the directories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories.
5. Stop

PROGRAM:

```
#include<stdio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("**************************************************\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
}
```

OUTPUT:



RESULT:

Thus the Single level directory structure is written and executed successfully.

EX NO: 11B	IMPLEMENTATION OF TWO-LEVEL DIRECTORY
DATE:	

AIM:

To write a C program to implement Two-level directory structure in C.

ALGORITHM:

1. Start
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories.
5. Stop.

PROGRAM:

```
#include<stdio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
clrscr();
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}}}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****************************************************\n");
for(i=0;i<n;i++){
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");}
printf("\n");    }
    }
```

OUTPUT:

```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ ./twolvldir
enter number of directories:2
enter directory 1 names:cse
enter size of directories:3
enter subdirectory name and size:os 3
enter file name:aaa
enter file name:bbb
enter file name:ccc
enter subdirectory name and size:cn 2
enter file name:xyz
enter file name:mih
enter subdirectory name and size:pqt 2
enter file name:anh
enter file name:kkk
enter directory 2 names:ece
enter size of directories:2
enter subdirectory name and size:ct 2
enter file name:aav
enter file name:vcs
enter subdirectory name and size:dc 4
enter file name:afs
enter file name:ged
enter file name:eee
enter file name:ttt

dirname         size    subdirname      size    files
*****************************************************
cse             3       os              3       aaa     bbb     ccc
                        cn              2       xyz     mih
                        pqt             2       anh     kkk
                ece             2       ct              2       aav     vcs
                        dc              4       afs     ged     eee     ttt
                mohamedinam@Mohamed-Inam-PC:~$
```

RESULT:

Thus a C program to implement Two-level directory structure is written and executed successfully.

EX NO: 12	FILE ALLOCATION STRATEGIES
DATE:	

A. SEQUENTIAL FILE ALLOCATION

AIM:

To Write a C Program to implement Sequential File Allocation method.

ALGORITHM:

1: Start the program.
2: Get the number of memory partition and their sizes.
3: Get the number of processes and values of block size for each process.
4: First fit algorithm searches all entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.
5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates if.
6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.
7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.
8: Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
clrscr();
printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
        printf("Enter no. of blocks occupied by file%d",i+1);
        scanf("%d",&b[i]);
        printf("Enter the starting block of file%d",i+1);
        scanf("%d",&sb[i]);
        t[i]=sb[i];
        for(j=0;j<b[i];j++)
                c[i][j]=sb[i]++;
        }
printf("Filename\tStart block\tlength\n");
 for(i=0;i<n;i++)
        printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
printf("Enter file name:");
 scanf("%d",&x);
printf("File name is:%d",x);
printf("length is:%d",b[x-1]);
printf("blocks occupied:");
 for(i=0;i<b[x-1];i++)
        printf("%4d",c[x-1][i]);
}
```

OUTPUT:

```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ gcc seq.c -o seq
mohamedinam@Mohamed-Inam-PC:~$ ./seq
Enter no.of files:2
Enter no. of blocks occupied by file 1 : 4
Enter the starting block of file 1 : 2
Enter no. of blocks occupied by file 2 : 10
Enter the starting block of file 2 : 5
Filename        Start block     length
1         2     4
2         5     10

Enter file name : inam

File name is: 12803
length is: 10
blocks occupied :    2   5   1   5   0   0   0   0   0   0
```

RESULT:

Thus the SEQUENTIAL file allocation method is implemented successfully.

EX NO: 12B	FILE ALLOCATION STRATEGIES – INDEXED FILE ALLOCATION
DATE:	

AIM:

To Write a C Program to implement Indexed File Allocation method.

ALGORITHM:

1: Start.

2: Let n be the size of the buffer

3: check if there are any producer

4. If yes check whether the buffer is full

5. If no the producer item is stored in the buffer

6: If the buffer is full the producer has to wait

7: Check there is any consumer.If yes check whether the buffer is empty

8: If no the consumer consumes them from the buffer

9: If the buffer is empty, the consumer has to wait.

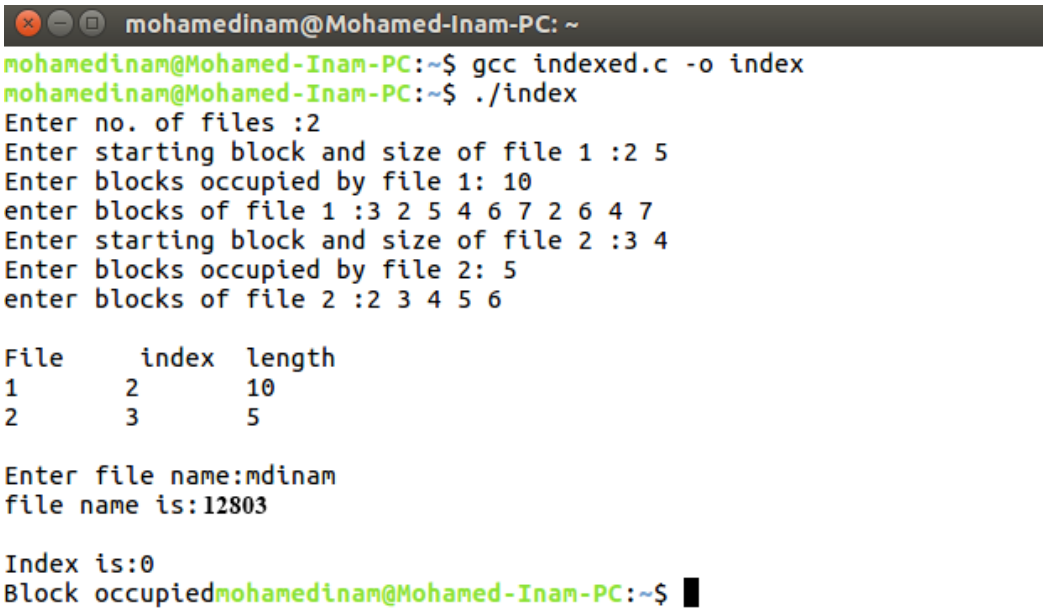
10: Repeat checking for the producer and consumer till required

11: Terminate the process.

PROGRAM:

```
#include<stdio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
clrscr();
printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {      printf("Enter starting block and size of file%d:",i+1);
        scanf("%d%d",&sb[i],&s[i]);
        printf("Enter blocks occupied by file%d:",i+1);
        scanf("%d",&m[i]);
        printf("enter blocks of file%d:",i+1);
        for(j=0;j<m[i];j++)
                scanf("%d",&b[i][j]);
 }
printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++)
 {
        printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 }
printf("\nEnter file name:");
 scanf("%d",&x);
printf("file name is:%d\n",x);
 i=x-1;
printf("Index is:%d",sb[i]);
printf("Block occupied are:");
 for(j=0;j<m[i];j++)
printf("%3d",b[i][j]);
}
```

OUTPUT:



RESULT:

Thus the indexed file allocation method is implemented successfully

EX NO: 12C	FILE ALLOCATION STRATEGIES – LINKED FILE ALLOCATION
DATE:	

AIM:

To Write a C Program to implement Linked File Allocation method.

ALGORITHM:

1: Create a queue to hold all pages in memory

2: When the page is required replace the page at the head of the queue

3: Now the new page is inserted at the tail of the queue

4: Create a stack

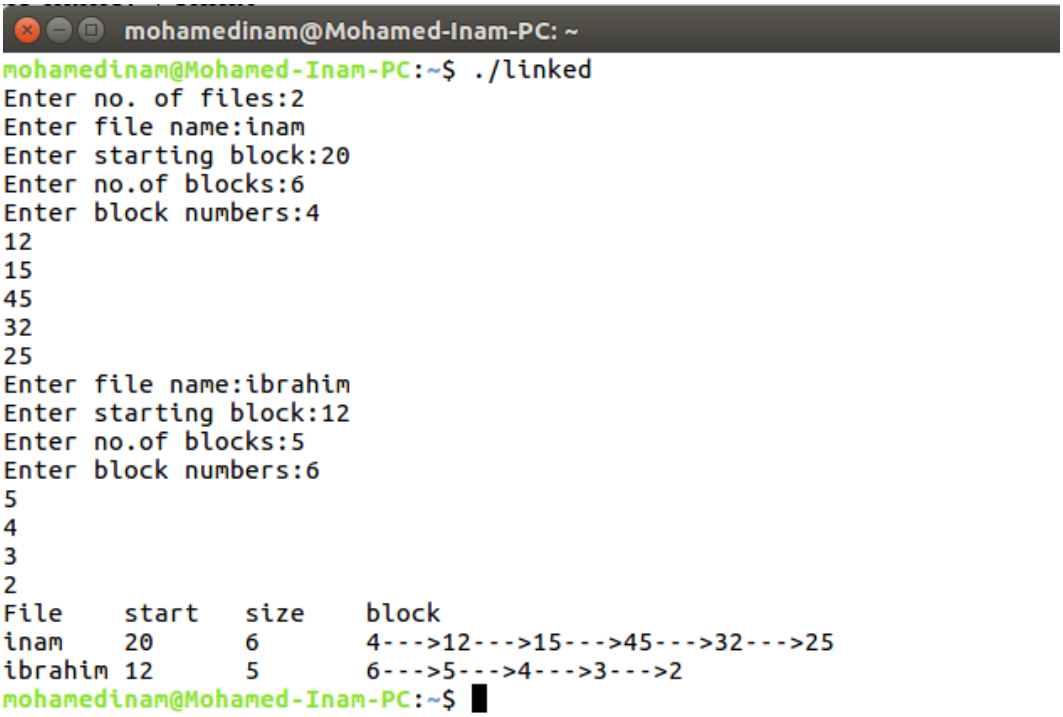
5: When the page fault occurs replace page present at the bottom of the stack

6: Stop the allocation.

PROGRAM:

```
#include<stdio.h>
struct file
{
        char fname[10];
        int start,size,block[10];
}f[10];
main()
{
        int i,j,n;
       clrscr();
       printf("Enter no. of files:");
       scanf("%d",&n);
 for(i=0;i<n;i++)
 {
       printf("Enter file name:");
       scanf("%s",&f[i].fname);
       printf("Enter starting block:");
        scanf("%d",&f[i].start);
        f[i].block[0]=f[i].start;
       printf("Enter no.of blocks:");
        scanf("%d",&f[i].size);
       printf("Enter block numbers:");
        for(j=1;j<=f[i].size;j++)
       {
              scanf("%d",&f[i].block[j]);
       }
 }

 printf("File\tstart\tsize\tblock\n");
 for(i=0;i<n;i++)
 {
       printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
       for(j=1;j<=f[i].size-1;j++)
              printf("%d--->",f[i].block[j]);
              printf("%d",f[i].block[j]);
              printf("\n");
 }
}
```

OUTPUT:



RESULT:

Thus the linked file allocation method is implemented successfully.