EX:NO: 1 **BASIC UNIX COMMANDS**

AIM:

To study and perform the basic command operation in shell programming.

BASIC COMMANDS:

1. who

This command displays the currently logged on username.

Syntax: $who

Output: [jack@unixITSERVER ~]$ who

```
07cs60   pts/0       2009-02-20 14:30 (192.168.1.171)
07cs61   pts/1       2009-02-20 15:03 (192.168.1.252)
08it39   pts/3       2009-02-20 15:43 (192.168.1.47)
08it50   pts/6       2009-02-20 15:43 (192.168.1.51)
08it22   pts/8       2009-02-20 15:43 (192.168.1.30)
08it29   pts/16      2009-02-20 15:44 (192.168.1.40)
08it01   pts/2       2009-02-20 15:44 (192.168.1.77)
```

2. whoami

This command display the system user name.

Syntax: $whoami

Output: [jack @unixITSERVER ~]$ whoami

jack

3. pwd

This command gives the currently logged on username.

Syntax: $pwd

Output: [jack @unixITSERVER ~]$ pwd

/home/ jack

4. mkdir

This command is used to create a new document.

Syntax: $mkdir

Output: [jack @unixITSERVER ~]$ mkdir vcx

5. cd

This command is used to create and change the document name.

Syntax: $cd

Output: [jack @unixITSERVER ~]$ cd vcx

[jack @unixITSERVER vcx]$ cd\

6. cd..

This command is used to exit from the directory.

Syntax: $cd...

Output: [jack @unixITSERVER ~]$ cd…
[jack @unixITSERVER ~]$

7. echo

This command is used to display the message.

Syntax: $echo”message”.

Output: [jack @unixITSERVER ~]$ echo "itdept"
itdept

WORD COUNT COMMAND:

8. wc

This command gives the number of lines, words and character in the given file.

Syntax: $wc file name.

Output: [jack @unixITSERVER ~]$ wc itdept
2 10 49 itdept

9. wc-w

This command gives the number of words in the given file.

Syntax: $wc-w file name.

Output: [jack @unixITSERVER ~]$ wc -w itdept
10 itdept

10. wc-l

This command gives the number of lines in the given file.

Syntax: $wc-l file name.

Output: [jack @unixITSERVER ~]$ wc -l itdept
2 itdept

11. wc-c

This command gives the number of character in the given file.

Syntax: $wc-c file name.

Output: [jack @unixITSERVER ~]$ wc -c itdept
49 itdept

DATE COMMAND:

12. date:

This command is used to give system date and time.

Syntax: $date

Output: [jack @unixITSERVER ~]$ date
Fri Feb 20 16:25:31 IST 2009

13. date+%m:

This command displays the current month in numerals.

Syntax: $date+%m.

Output: [jack @unixITSERVER ~]$ date +%m
02

14. date+%h

This command displays the current hour.

Syntax: $date+%h.

Output: [jack @unixITSERVER ~]$ date +%h
Feb

15. date+%y

This command displays the current year .

Syntax: $date+%y.

Output: [jack @unixITSERVER ~]$ date +%y
09

16. date+%d

This command displays the current date.

Syntax: $date+%d

Output: [jack @unixITSERVER ~]$ date +%d
20

17. date+%h%m%s

This command displays the current hour, month, seconds.

Syntax: $date+%h%m%s

Output: [jack @unixITSERVER ~]$ date +%h%m%s
Feb021235127855

18. tty

This command gives the current terminal in which we are working.

Syntax: $tty

Output: [jack @unixITSERVER ~]$ tty
/dev/pts/25

CAT COMMAND:

19. Cat > Filename:

This command creates the new file.

Syntax: $cat>file name.

Output: [jack @unixITSERVER ~]$ cat itdept
God is great.
He will not leave us at any circumtanes

20. cat Filename

This command is used to display the contents in the file.

Syntax: $cat filename

Output: [jack @unixITSERVER ~]$ cat itdept
God is great.
He will not leave us at any circumtanes

21. cp

This command used to copy the content of one file into the another file.

Syntax: $cp source destination.

Output: [jack @unixITSERVER ~]$ cp itdept valli
[jack @unixITSERVER ~]$ cat valli
God is great.
He will not leave us at any circumtanes.

GREP COMMAND:

22. grep pattern filename

This command is used to search file in one or more form.

Syntax: $grep pattern filename.

Output: [jack @unixITSERVER ~]$ grep g itdept
God is great.

23. sort

This command gives the data in the ascending order.

Syntax: $sort filename.

Output: [jack @unixITSERVER ~]$ sort itdept
God is great.
He will not leave us at any circumtanes.

24. sort-m

This command used merge two files .

Syntax: $sort-m filename1 filename2.

Output: [jack @unixITSERVER ~]$ sort -m itdept valli
God is great.
God is great.
He will not leave us at any circumtanes.
He will not leave us at any circumtanes.

MOVE COMMAND:

25. mv

This command is used to move the content of one file into another.

Syntax: $mv source destination.

Output: [jack @unixITSERVER ~]$ mv itdept valli
[jack @unixITSERVER ~]$ cat valli
God is great.

LIST COMMANDS:

26. ls

This command is used to list the filename in order.

Syntax: $ls

Output: [jack @unixITSERVER ~]$ ls
h valli vcx

27. ls-l

This command is used to list out the files with login the date,etc…

Syntax: $ls-l

Output: [jack @unixITSERVER ~]$ ls -l
total 24
drwxrwxr-x 2 08it05 08it05 4096 2009-02-20 15:58 h
-rw-rw-r-- 1 08it05 08it05 49 2009-02-20 16:18 valli
drwxrwxr-x 2 08it05 08it05 4096 2009-02-20 16:02 vcx

28. ls-n

This command is used to display the file column by column.

Syntax: $ls-n

Output: [jack @unixITSERVER ~]$ ls -n
total 24
drwxrwxr-x 2 715 715 4096 2009-02-20 15:58 h
-rw-rw-r-- 1 715 715 49 2009-02-20 16:18 valli
drwxrwxr-x 2 715 715 4096 2009-02-20 16:02 vcx

29. ls-m

This command is used to display filename which is separated by commas.

Syntax: $ls-m.

Output: [jack @unixITSERVER ~]$ ls -m
h, valli, vcx

30. ls-t

This command is used to display all the files without its details.

Syntax: $ls-t.

Output: [jack @unixITSERVER ~]$ ls -t
Itdept valli

31. ls-d

This command is used display all the directories.

Syntax: $ls-d.

Output: [jack @unixITSERVER ~]$ ls -d
vcx

32. ls-s

This command is used to list out the file with the amount of space occupied by them.

Syntax: $ls-s

Output: [jack @unixITSERVER ~]$ ls -s valli
8 valli

33. ls-v

This command gives the version of the file.

Syntax: $ls-v filename

Output: [jack @unixITSERVER ~]$ ls -v valli
valli

34. ls-i

This command gives the mode of the file.

Syntax: $ls-i filename.

Output: [jack @unixITSERVER ~]$ ls -i valli
8583336 valli

EX:NO:2 **SHELL PROGRAMMING**

a) GREATEST OF THREE NUMBER

AIM:

To find the greatest of three given numbers using shell programming in UNIX operating system.

ALGORITHM:

STEP 1: Start the program.
STEP 2: Enter any three number.
STEP 3: It will check the condition
$a-gt $b –a test $a –gt $ c.
STEP 4: I f a is greater than b and c
Print a is greater.
STEP 5: If b is greater than a and c.
Print b is greater .
STEP 6: Else c is greater.
STEP 7: Stop.

PROGRAM:

```
echo ”enter the first number”;
read a
echo ”enter the second number”;
read b
echo ”enter the third number”;
read c
if test $a – gt $b –a $a – gt $c
then
echo ”a is greater”
else if test $b – gt $c-a $b - gt $c
then
echo ”b is greater”
else
echo ” cis greater”
fi
fi
```

OUTPUT:

enter the first number

4

enter the second number

5

enter the third number

7

c is greater

b) ARITHEMETIC OPERATION

AIM:

To perform arithmetic operations using switch statements in shell programming.

ALGORITHM:

STEP 1: Start
STEP 2: print the arithmetic operations.
STEP 3: Read the two numbers.
STEP 4: Write the four arithmetic operations in correct
number usingcase operation.
STEP 5: print the values relevant options.
STEP 6: Close the case operations.
STEP 7: Stop**.**

PROGRAM:

```
echo ”enter the first number”;
read a
echo “enter the second number”;
read b
echo “1)addition”;
echo “2)substraction”;
echo “3)multiplication”;
echo “4)division”;
echo “enter your choice”;
read ch
case $ch in
1)expr $a + $b;;
2)expr $a - $b;;
3)expr $a \* $b;;
4)expr $a / $b;;
esac
```

OUTPUT:

```
enter the first number
4
enter the second number
3
1)addition
2)substraction
3)multiplication
4)division
enter your choice
1
7
```

c) FACTORIAL OF A GIVEN NUMBER

AIM:

To find the factorial of the given number using shell script in unix operating System.

ALGORITHM:

STEP 1: Read the number from the user to which the factorial is to be determined.
STEP 2: Initialise the value for I and f as one.
STEP 3: Check whether i less than or equal to the given number using while loop.
STEP 4: Multiply i and f and initialize the value of f.
STEP 5: Increment the value of I by one.
STEP 6: Repeat the while loop until the condition is false.
STEP 7: Print the value of f as the factorial of a given number**.**

PROGRAM:

```
echo “enter the number “;
read a
f=1
for((i=1;i<=a;i++))
do
((if=$i \* $f))
done
echo “factorial is $f”
```

OUTPUT:

```
Enter the number
5
Factorial is 120.
```

d) FIBANOCCI SERIES

AIM:

To generate the Fibonacci series of the given number using shell script in UNIX operating system.

ALGORITHM:

STEP 1: Read the number of terms to display from the user.
STEP 2: Assign the value of I as zero and for j as one.
STEP 3: Print the value of f1 and f2.
STEP 4: Assign the value of f as two.
STEP 5: Check whether the given number is less than given number using while loop.
STEP 6: Add the value of f1 and f2 and assign it to f3.
STEP 7: Print the value f3.
STEP 8: Assign the value of f2 to f3.
STEP 9: Increment the value of I by 1and repeat the done.

PROGRAM:

```
#bin/sh
c=1
echo “enter the limit”
read a
i=-1;
j=1;
echo “result is”
while test  $c –le $a
do
((i=$i + $j))
echo “$i”
t=$i
i=$j
j=$t
((c=$c+1))
done
```

OUTPUT:

```
enter  the limit
4
result is
0
1
1
2
```

EX:NO: 3 IMPLEMENTATION OF SYSTEM CALLS

FORK:

AIM

To create a new child process using fork system call.

ALGORITHM

1. Declare a variable *x* to be shared by both child and parent.
2. Create a child process using fork system call.
3. If return value is -1 then

a. Print "Process creation unsuccessfull"

b. Terminate using exit system call.

4. If return value is 0 then

a. Print "Child process"

b. Print process id of the child using getpid system call

c. Print value of *x*

d. Print process id of the parent using getppid system call

5. Otherwise

a. Print "Parent process"

b. Print process id of the parent using getpid system call

c. Print value of *x*

d. Print process id of the shell using getppid system call.

6. Stop

PROGRAM:

```
\#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
main()
{
        int pid;
        printf("\n The parent ID is %d",getpid);
        pid=fork();
        printf("\nThe child ID is %d",pid);
        if(pid<0)
        {
                fprintf(stderr,"Fork failed");
        }
        else if(pid==0)
        {
                execlp("/bin/ls,""ls",NULL);
        }
        else
```

```
        {
                wait(NULL);
                printf("\nChild completed");
                exit(0);
        }
}
```

OUTPUT:

```
aa.c
allo
allo.c
greatest3.sh
sys.c
The parent ID is 138539358
The child ID is 3277
Child completed
```

IMPLEMENTATION OF SYSTEM CALLS STAT, OPENDIR & READDIR

AIM

To display file status using stat system call.

ALGORITHM

1. Get *filename* as command line argument
2. If *filename* does not exist then stop.
3. Call stat system call on the *filename* that returns a structure
4. Display members st_uid, st_gid, st_blksize, st_block, st_size, st_nlink, etc.,
5. Convert time members such as st_atime, st_mtime into time using ctime function
6. Compare st_mode with mode constants such as S_IRUSR, S_IWGRP, S_IXOTH and display file permissions
7. Stop

PROGRAM:

```
#include<dirent.h>
#incude<stdlib.h>
#include<stdio.h>
#include<sys/types>
#include<sys/stat.h>
main()
{
struct  stat  buf;
int    exists;
DIR*d;
struct  dirent  *de;
d=opendir(".");
if(d==NULL)
{
fprintf(stderr,"could  not  open\n");
```

```
exit(1);
}
for(de=read  dir(d);de!=NULL; de=read   dir(d))
{
exists=stat(de->d_name,&buf);
if(exists<0)
{
Fprintf (stderr,"%s  not  found" ,de->d_name);
}
else
{
printf(" %s%ld\n",de->d_name,buf.st_size);
}
}
}
```

OUTPUT:

```
tt.sh 292
aa.c 17
student 0
io. C 269
.zshrc 658
Exp.sh 3444
.dmrc 26
Fox 0
Friend o
Io 4096
Ss 8
```

IMPLEMENTATION OF INPUT/OUTPUT SYSTEM CALLS

AIM

To append content to an existing file.

ALGORITHM

Step:1 Declare a character buffer buf to store 100 bytes
Step 2: Get exisiting filename as command line argument
Step 3:Create a file with the given name using open system call with O_APPEND option.
Step 4: Check the file descriptor
a) If value is negative, then stop.
Step 5: Get input from the console until user types Ctrl+D
a) Read 100 bytes (max.) from console and store onto buf using read system call
b) Write length of buf onto file using write system call.
Step 6: Close the file using close system call
Step 7: Stop

PROGRAM :

```
#include<stdio.h>
#include<sys/types.h>
#include<funtl.h>
main()
{
int fd,s,n,i=0;
char buff[30];
printf("The file in reverse order");
fd=open("camera.c"O_RDWR);
lseek(fd,-1,2);
while(1)
{
n=read(fd,buff,1);
write(1,buff,n);
if(lseek(fd,-2,1)==1)
break;
}
Close(fd);
}
```

OUTPUT:

```
[student @localhost~]$ cat>camera.c
How are you?
[student @localhost~]$cc ioscal.c
[student @localhost~]$./a.out
?uoy era woH
Files is in reverse order
```

EX:NO:4 IMPLEMENTATION OF SIMULATION CONCEPT

a) Ls Command

AIM

To simulate ls command using UNIX system calls.

ALGORITHM

1. Store path of current working directory using getcwd system call

2. Scan directory of the stored path using scandir system call and sort the resultant array of structure

3. Display dname member for all entries if it is not a hidden file.

4. stop

PROGRAM

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

main()
{
struct dirent **namelist;
int n,i;
char pathname[100];

getcwd(pathname);
n = scandir(pathname, &namelist, 0, alphasort);

if(n < 0)
printf("Error\n");
else
for(i=0; i<n; i++)
if(namelist[i]->d_name[0] != '.')
printf("%-20s", namelist[i]->d_name);
}
```

OUTPUT

```
$ gcc list.c -o list
$ ./list
a.out
dirlist.c
ex6c.c
```

b) Grep Command

AIM

To simulate grep command using UNIX system call

ALGORITHM

1. Get filename and search string as command-line argument.
2. Open the file in read-only mode using open system call
3. If file does not exist, then stop
4. Let length of the search string be *n*
5. Read line-by-line until end-of-file

a. Check to find out the occurrence of the search string in a line by examining characters in the range 1–n, 2–n+1, etc
b. If search string exists, then print the line

6. Close the file using close system call
7. Stop

PROGRAM CODING:

```
#include<stdio.h>
main()
{
system("ls");
system("grepstdio5.c");
}
```

OUTPUT:

```
a.out desktop documents download interprocess.c ipc.c music zzz pc.c
pictures public s.c simul.c templates videos.
```

EX:NO:5 a) IMPLEMENTATION OF FIRST COME FIRST SERVE

AIM

To schedule snapshot of processes queued according to FCFS (First Come First Serve) scheduling.

ALGORITHM

1. Define an array of structure process with members pid, btime, wtime & ttime
2. Get length of the ready queue, i.e., number of process (say n)
3. Obtain btime for each process.
4. The wtime for first process is 0.
5. Compute wtime and ttime for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
6. Compute average waiting time awat and average turnaround time atur
7. Display the btime, ttime and wtime for each process
8. Display GANTT chart for the above scheduling
9. Display awat time and atur
10. stop

PROGRAM:

```
#include<stdio.h>
int main()
{
 int i,n,j,p[10],wait[10],turn[10],burst[10];
float avgwaittime=0,avgturntime=0;
printf("\n Enter the number of process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the %d name",i);
scanf("%d",&p[i]);
printf("\n Enter the cpu burst time of p[%d]:"i);
scanf("%d",&burst[i]);
wait[i]=0;
turn[i]=0;
}
wait[0]=0;
for(i=0;i<n;i++)
{
wait[i+1]=wait[i]+burst[i];
printf("\n\n %d \n \n",wait[i+1]);
}
for(i=0;i<n;i++)
```

```
{
avgwaittime=avgwaittime+wait[i];
}
for(i=0;i<n;i++)
{
printf("waiting time for process: %d \n",wait[i]);
}
avgwaittime=(avgwaittime)/n;
printf(\n avgwaiting time is:%f",avgwaittime);
for(i=0;i<n;i++)
{
turn[i]=burst[i]+wait[i];
printf("\n turn around time of process:%d is %d \n",i,turn[i]);
avgturntime=(avgturntime)+turn[i];
}
avgturntime=(avgturntime)/n;
printf("avg turn around time is:%f",avgturntime);
return 0;
}
```

OUTPUT:

```
 Enter the number of process 3
Enter the 0name 1
Enter the cpu burst time of  p[0]:24
Enter the 1name 2
Enter the cpu burst time of  p[1]:3
Enter the 2name 3
Enter the cpu burst time of  p[2]:3
24
27
30
Waiting time for process:0
Waiting time for process:24
Waiting time for process:27
Avgwaiting time is:17.000000
Turn around time of process:0 is 24
Turn around time of process:1 is 27
Turn around time of process:2 is 30
Avg turn around time is : 27.000000
```

B) IMPLEMENTATION OF SHOREST JOB FIRST SCHEDULING ALGORITHM

AIM

To schedule snapshot of processes queued according to SJF (Shortest Job First) scheduling.

ALGORITHM

1. Define an array of structure process with members pid, btime, wtime & ttime
2. Get length of the ready queue, i.e., number of process (say n)
3. Obtain btime for each process.
4. *Sort* the processes according to their *btime* in ascending order
   a. If two process have same *btime*, then FCFS is used to resolve the tie.
5. The *wtime* for first process is 0.
6. Compute wtime and ttime for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
b. . $ttime_i = wtime_i + btime_i$
7. Compute average waiting time awat and average turnaround time atur
8. Display the btime, ttime and wtime for each process
9. Display GANTT chart for the above scheduling
10. Display awat time and atur
11. stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,n,t,j,p[10],wait[10],turn[10],burst[10];
float avgwaittime=0,avgturntime=0;
char*c;
printf(“enter the number of process”);
scanf(“%d”,&n);
for(i=0;i<n;i++)
{
print(“enter the %d name”,i);
scanf(“%d”,&p[i]);
printf(“enter the cpu burst time of p[%d]:”,i);
scanf(“%d”,&burst[i]);
wait[i]=0;
turn[i]=0;
}
for(i=0;i<n;i++)
{
```

```
for(j=i+1;j<n;j++)
{
if(burst[i]>burst[j])
{
t=burst[i];
burst[i]=burst[j];
burst[j]=t;
}
}
}
for(i=0;i<n;i++)
printf("\n the process is sorted order is %d",burst[i]);
wait[0]=0;
for(i=0;i<n;i++)
{
wait[i+1]=wait[i]+burst[i];
printf("\n\n\n the wait time %d \n" wait[i+1]);
}
for(i=0;i<n;i++)
{
printf("waiting time for process:%d\n",wait[i]);
avgwaittime=(avgwaittime)+wait[i];
}
avgwaittime=(avgwaittime)/n;
printf("average waiting time is:%f", avgwaittime);
for(i=0;i<n;i++)
{
turn[i]=burst[i]+wait[i];
printf("\n turn around time of process  %d is %d\n",i,turn[i]);
avgturntime= (avgturntime)+turn[i];
}
avgturntime= (avgturntime)/n;
printf("average turn around time is:%f",avgturntime);
return 0;
}
```

OUTPUT:

```
enter the number of process:3
enter the 0 name  1
enter the cpu burst time of  p[0]:6
enter the 1 name 2
enter the cpu burst time of  p[1]:3
enter the 2 name 3
enter the cpu burst time of  p[2]:5
the process in sorted order is  3
the process in sorted order is  5
the process in sorted order is  6
the wait time  3
```

the wait time 8
the wait time 14
waiting time for process:0
waiting time for process:3
waiting time for process:8
average waiting time is:3.666667
turn around time of process:0 is 3
turn around time of process:1 is 8
turn around time of process:2 is 14
average turn around time is:8.333333

EX:NO:6 a) IMPLEMENTATION OF PRIORITY SCHEDULING ALGORITHM

AIM:

To schedule snapshot of processes queued according to Priority scheduling.

ALORITHM:

1. Define an array of structure process with members pid, btime, wtime & ttime, pri
2. Get length of the ready queue, i.e., number of process (say n)
3. Obtain btime and pri for each process.
4. *Sort* the processes according to their *pri* in ascending order
   a. If two process have same *pri*, then FCFS is used to resolve the tie.
5. The *wtime* for first process is 0.
6. Compute wtime and ttime for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
b. . $ttime_i = wtime_i + btime_i$
7. Compute average waiting time awat and average turnaround time atur
8. Display the btime, ttime and wtime , pri for each process
9. Display GANTT chart for the above scheduling
10. Display awat time and atur
11. stop

PROGRAM:

```
#include<stdio.h>
int main()
{
int s,i,n,t,j,p[10],wait[10],turn[10],burst[10],a[10];
float avgwaittime=0,avgturntime=0;
printf("enter the number of process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
 printf("enter the %d name",i);
scanf("%d",&p[i]);
printf("enter the cpu burst time of p[%d]:",i);
scanf("%d",&burst[i]);
printf("enter priority for process:");
scanf("%d",&a[i]);
wait[i]=0;
turn[i]=0;
}
printf("\n the process according to the priority is");
for(i=0;i<n;i++)
{
```

```
for(j=i+1;j<n;j++)
{
if(a[i]>a[j])
{
t=a[i];
a[i]=a[j];
a[j]=t;
s=burst[i];
burst[i]=burst[j];
burst[j]=s;
}
}
printf("\n \t %d \t %d \t %d",p[i],a[i],burst[i]);
}
wait[0]=0;
for(i=0;i<n;i++)
{
wait[i+1]=wait[i]+burst[i];
printf("\n\n\n the wait time%d\n",wait[i+1]);
}
for(i=0;i<n;i++)
{
avgwaittime= avgwaittime+ wait[i];
}
for(i=0;i<n;i++)
{
printf("\n waiting time for process %d is %d \n",i,wait[i]);
}
avgwaittime=( avgwaittime)/n;
printf("average waiting time is:%f",avgwaittime);
for(i=0;i<n;i++)
{
turn[i]=burst[i]+wait[i];
printf("\n turn around time of process:%d is %d \n",i,turn[i]);
avgturntime=(avgturntime)+turn[i];
}
avgturntime=(avgturntime)/n;
printf("avg turn around time is:%f",avgturntime);
return 0;
}
```

OUTPUT:

```
enter the number of process  3
enter the 0 name 1
enter the cpu burst time of p[0]:3
enter the priority for process:2
enter the 1 name 2
```

```
enter the cpu burst time of p[1]:2
enter the priority for process:4
enter the 2 name 3
enter the cpu burst time of p[2]:5
enter the priority for process:1
the process according to the priority is
   1    1    5
   2    2    3
   3    4    2
the wait time 5
the wait time 8
the wait time 10
waiting time for process 0 is 0
waiting time for process 1 is 5
waiting time for process 2 is 8
average waiting time is :4.333333
turn around time of process 0 is 5
turn around time of process 1 is 8
turn around time of process 2 is 10
avg turn around time is:7.666667
```

b)IMPLEMENTATION OF ROUND ROBIN SCHEDULING ALGORITHM

AIM:

To schedule snapshot of processes queued according to Round robin scheduling.

ALORITHM:

1. Get length of the ready queue, i.e., number of process (say n)
2. Obtain btime for each process.
3. Get the time slice per round,say TS
4. Determine the number of rounds for each process
5. The wait time for first process is 0
6. If $B_i > TS$ then process takes more than one round. Therefore turnaround and waiting time should include the time spent for other remaining processes in the same round
7. Calculate *average* waiting time and turn around time
8. Display the GANTT chart that includes
   a. order in which the processes were processed in progression of rounds
   b. Turnaround time T_i for each process in progression of rounds.
9.Display the *burst* time, *turnaround* time and *wait* time for each process (in order of rounds they were processed).
10. Display *average* wait time and turnaround time
11. Stop

PROGRAM CODING:

```
#include<stdio.h>
#define FINISHED 0
#define FINISHED 1
#define RUNNING 2
struct  pro
{
char id[6];
int burst_time;
int wait_time;
int turn_time;
int state;
}
process[10];
int no;
int  main()
{
int I,j,sum,add=0,time=0,time_slice=1,curr_proc,prev_time=0,n;
int fin=0;
```

```
printf("\n Enter the number of process:");
scanf("%d",&no);
for(i=1;i<=no;i++)
{
printf("\n Enter the process id:");
scanf("%s",&processor[i].id);
printf("\n enter the cpu burst time:");
scanf("%d",&process[i].burst_time);
process[i].wait_time=0;
process[i].turn_time=process[i].burst_time;
process[i].state=WAITING;
}
curr_proc=1;
process[curr_proc].state=RUNNING;
while(1)
{
process[curr_proc].burst_time--;
time++;
for(j=1;j<=no;j++)
{
if(process[j].state==WAITING)
{
process[j].wait_time++;
process[j].turn_time++;
}
}
if(time_slice==5||process[curr_proc].burst_time==0)
{
if(process[curr_proc].burst_time==0)
{
process[curr_proc].state=FINISHED;
fin++;
}
else
process[curr_proc].state=WAITING;
printf("\n %s-%d to %d ",process[curr_proc].id,prev_time,time);
if(fin==0)
break;
curr_proc=curr_proc%no+1;
while(process[curr_proc].state=FINISHED)
{
curr_proc=curr_proc%no+1;
}
process[curr_proc].state=RUNNING;
prev_time=time;
time_slice=0;
}
```

```
time_slice++;
}
printf("\n process\t waiting time\t turn around time\n");
sum=0;
for(i=0;i<=no;i++)
{
sum+=process[i].wait_time;
add+=process[i].turn_time;
printf("\n %s\t %d\t\t%d",process[i].id,process[i].wait_time,process[i].turn_time);
printf("\n average waiting time %d\n",sum/no);
printf("\n average turn around time %d\n",add/no);
return 0;
}
```

OUTPUT:

```
 [student@localhost~]$vi rr.c
[student@localhost~]$cc rr.c
[student@localhost~]$./a.out
Enter the number of process:3
Enter the process id:1
Enter the cpu burst time:10
Enter the process id:2
Enter the cpu burst time:12
Enter the process id:3
Enter the cpu burst time:3
1-0 to 5
2-5 to 10
3-10 to 13
1-13 to 18
2-18 to 23
2-23 to 25
```

Process	Waiting time	Turn around time
1	8	18
2	13	25
3	10	13

```
Average waiting time:10
Average turn around time:18
```

EX:NO:7 IMPLEMENTATION OF INTERPROCESS COMMUNICATION

a)Shared Memory

AIM

To demonstrate communication between process using shared memory.

ALGORITHM

Server:

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

Client:

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

PROGRAM

SERVER

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

#define shmsize 27
```

```
main()
{
char c;
int shmid;
key_t key = 2013;
char *shm, *s;
if ((shmid = shmget(key, shmsize, IPC_CREAT|0666)) < 0)
{
perror("shmget");
exit(1);
}
printf("Shared memory id : %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) onto shared memory\n");
for (c = 'a'; c <= 'z'; c++)
*s++ = c;
*s = '\0';
while (*shm != '*');
printf("Client finished reading\n");
if(shmdt(shm) != 0)
fprintf(stderr, "Could not close memory segment.\n");

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

CLIENT

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

#define shmsize 27

        main()
{
int shmid;
key_t key = 2013;
char *shm, *s;
if ((shmid = shmget(key, shmsize, 0666)) < 0)
{
printf("Server not started\n");
exit(1);
}
else
printf("Accessing shared memory id : %d\n",shmid);

if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
{
```

```
perror("shmat");
exit(1);
}
printf("Shared memory contents:\n");
for (s = shm; *s != '\0'; s++)
putchar(*s);
putchar('\n');

*shm = '*';
}
```

OUTPUT

SERVER
$ ./shms
Shared memory id : 196611
Writing (a-z) onto shared memory
Client finished reading

CLIENT
$ ./shmc
Accessing shared memory id : 196611
Shared memory contents:
abcdefghijklmnopqrstuvwxyz

B) PIPES

AIM

To generate 25 fibonacci numbers and determine prime amongst them using pipe.

ALGORITHM

1. Declare a array to store fibonacci numbers
2. Decalre a array *pfd* with two elements for pipe descriptors.
3. Create pipe on *pfd* using pipe function call.
   a. If return value is -1 then stop
4. Using fork system call, create a child process.
5. Let the child process generate 25 fibonacci numbers and store them in a array.
6. Write the array onto pipe using write system call.
7. Block the parent till child completes using wait system call.
8. Store fibonacci nos. written by child from the pipe in an array using read system call
9. Inspect each element of the fibonacci array and check whether they are prime

a. If prime then print the fibonacci term.

10. Stop

PROGRAM CODE:

```
#include<stdio.h>
#include<sys\type.h>
#include<unistd.h>
int main()
{
int fd[2],n,m,pipid,pid;
pipid=pipe(fd);
int odd sum=0,even sum=0;
if(pid!=0)
{
printf("\n pipe is not created");
}
printf("\npipe is created");
pid=fork();
if(pid!=0)
{
close(fd[0]);
while(1)
{
printf("\nenter the no"):
scanf("%d",&n);
write(fd[i],&n,sizeof(n));
if(n==99)
```

```
break;
}
close(fd[i]);
}
if(pid==0)
{
close(fd[i]);
while(10
{
read(fd[0],&m,sizeof(m));
if(m!=-99)
{
if(m%2!=0)
{
oddsum=oddsum+m;
}
else
evensum=evensum+m;
}
else
{
printf("evensum=%d\n",evensum);
printf("oddsum=%d\n",oddsum);
break;
}
}
close(fd[0]);
}
}
```

OUTPUT:

enter the no:45

enter the no:10

enter the no:5

enter the no:20

enter the no:-99

pipe is created

evensum=30

oddsum=50

C)Message Queue

AIM

To exchange message between server and client using message queue.

ALGORITHM

Server:

1. Decalre 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 newline 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. Decalre 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 newline 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:

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

```
struct mesgq
{
long type;
char text[200];
} mq;
main()
{
int msqid, len;
key_t key = 2013;
if((msqid = msgget(key, 0644|IPC_CREAT)) == -1)
{
perror("msgget");
exit(1);
}
printf("Enter text, ^D 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("From Client: \"%s\"\n", mq.text);
}
msgctl(msqid, IPC_RMID, NULL);
}
```

CLIENT:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesgq
{
long type;
char text[200];
} mq;
main()
{
int msqid, len;
key_t key = 2013;
if ((msqid = msgget(key, 0644)) == -1)
{
printf("Server not active\n");
exit(1);
}
printf("Client ready :\n");
while (msgrcv(msqid, &mq, sizeof(mq.text), 0, 0) != -1)
{
printf("From Server: \"%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("Server Disconnected\n");
}
```

OUTPUT:

SERVER

```
$ ./srvmsg
Enter text, ^D to quit:
hi
From Client: "hello"
Where r u?
From Client: "I'm where i am"
bye
From Client: "ok"
^D
```

CLIENT

```
$ ./climsg
Client ready:
From Server: "hi"
hello
From Server: "Where r u?"
I'm where i am
From Server: "bye"
ok
Server Disconnected
```

EX:NO: 8 PROGRAM FOR PRODUCER CONSUMER PROBLEM

AIM

To synchronize producer and consumer processes using semaphore

ALGORITHM

1. Create a shared memory segment *BUFSIZE* of size 1 and attach it.
2. Obtain semaphore id for variables *empty*, *mutex* and *full* using semget function.
3. Create semaphore for *empty*, *mutex* and *full* as follows:
   a. Declare *semun,* a union of specific commands.
   b. The initial values are: 1 for mutex, N for empty and 0 for full
   c. Use semctl function with SETVAL command
4. Create a child process using fork system call.
   a. Make the parent process to be the *producer*
   b. Make the child process to the *consumer*
5. The *producer* produces 5 items as follows:
   a. Call *wait* operation on semaphores *empty* and *mutex* using semop function.
   b. Gain access to buffer and produce data for consumption
   c. Call *signal* operation on semaphores *mutex* and *full* using semop function.
6. The *consumer* consumes 5 items as follows:
   a. Call *wait* operation on semaphores *full* and *mutex* using semop function.
   b. Gain access to buffer and consume the available data.
   c. Call *signal* operation on semaphores *mutex* and *empty* using semop function.
7. Remove shared memory from the system using shmctl with IPC_RMID argument
8. Stop

Program code:

```
#include<stdio.h>
#include<stdlib.h>
static int next=1,nextc=0,nextp=1;
int buffer[50],in=0,out=0,cut=0;
void producer();
void consumer();
int main()
{
 int nos,numcount;
system("clear");
printf("\n shared memory concept using producer consumer algorithm");
printf("\n enter the number of values to be shared(1-100):");
scanf("%d",&nos);
for(numcount=0;(in<nos)&&(numcount<nos);numcount++,cut++)
```

```
{
producer();
consumer();
}
printf("\n shared memory process completed");
return(0);
}
void producer()
{
printf("item %d is produced \n",nextp);
if((in+1)%5==out)
printf("buffer is full \n");
else
{
buffer[in]=nextp;
in=(in+1)%5;
nextp++;
}
}
void consumer()
{
 if(in==out)
{
 printf("\n buffer is empty\n");
}
nextc=buffer[out];
out=(out+1)%5;
printf("item is consumed \n",next);
}
```

OUTPUT:

```
Shared memory concept using producer consumer algorithm
Enter the number of values to be shared(1-100):3
Item 1 is produced
Item is consumed
Item 2 is produced
Item is consumed
Item 3 is produced
Item is consumed
Shared memory process completed
```

EX:NO:9 IMPLEMENTATION OF MEMORY MANAGEMENT

a) FIRST FIT

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 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 size;
int actual;
} h[10];
main()
{
int i, np, nh, j;
printf("Enter the number of Holes : ");
scanf("%d", &nh);
for(i=0; i<nh; i++)
{
printf("Enter size for hole H%d : ",i);
scanf("%d", &h[i].size);
h[i].actual = h[i].size;
}
printf("\nEnter number of process : " );
scanf("%d",&np);
for(i=0;i<np;i++)
{
printf("enter the size of process P%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\tFirst fit\n");
printf("\nProcess\tPSize\tHole");
for(i=0; i<np; i++)
{
if(p[i].flag != 1)
printf("\nP%d\t%d\tNot allocated", 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:

```
Enter the number of Holes : 5
Enter size for hole H0 : 100
Enter size for hole H1: 500
Enter size for hole H2 : 200
Enter size for hole H3 : 300
Enter size for hole H4 : 600
Enter number of process: 4
Enter the size of process P0:212
Enter the size of process P1:417
Enter the size of process P2:112
Enter the size of process P3:426
Process         psize           hole
P0              212             H1
P1              417             H4
P2              112             H1
P3              446             Not Allocated
Hole            Actual          Available
H0              100             100
H1              500             176
H2              200             200
H3              300             300
H4              600             183
```

b)Best Fit

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(struct hole[], int);
printf("Enter the number of Holes : ");
scanf("%d", &nh);
for(i=0; i<nh; i++)
{
```

```
printf("Enter size for hole H%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 process P%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\tBest fit\n");
printf("\nProcess\tPSize\tHole");
for(i=0; i<np; i++)
{
if(p[i].flag != 1)
printf("\nP%d\t%d\tNot allocated", 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(struct hole bh[], int n)
{
struct hole temp;
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:

Enter the number of Holes : 5
Enter size for hole H0 : 100
Enter size for hole H1: 500
Enter size for hole H2 : 200
Enter size for hole H3 : 300
Enter size for hole H4 : 600
Enter number of process: 4

Enter the size of process P0:212

Enter the size of process P1:417

Enter the size of process P2:112

Enter the size of process P3:426

Process	psize	hole
P0	212	H3
P1	417	H1
P2	112	H2
P3	446	H4

Hole	Actual	Available
H1	500	83
H3	300	88
H2	200	88
H0	100	100
H4	600	174

EX:NO:10 IMPLEMENTATION OF CONTIGUOUS FILE ALLOCATION TECHNIQUE

AIM

To implement file allocation on free disk space in a contiguous manner

ALGORITHM

1. Assume no. of blocks in the disk as 20 and all are free.
2. Display the status of disk blocks before allocation.
3. For each file to be allocated:
   a. Get the *filename*, *start* address and file *length*
   b. If *start* + *length* > 20, then goto step 2.
   c. Check to see whether any block in the range (start, start + length-1) is allocated. If so, then go to step 2.
   d. Allocate blocks to the file contiguously from start block to start + length – 1.
4. Display directory entries.
5. Display status of disk blocks after allocation
6. Stop

PROGRAM

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

int num=0, length[10], start[10];
char fid[20][4], a[20][4];
void directory()
{
int i;
printf("\nFile Start Length\n");
for(i=0; i<num; i++)
printf("%-4s %3d %6d\n",fid[i],start[i],length[i]);
}
void display()
{
int i;
for(i=0; i<20; i++)
printf("%4d",i);
printf("\n");
for(i=0; i<20; i++)
printf("%4s", a[i]);
}
main()
{
int i,n,k,temp,st,nb,ch,flag;
char id[4];
for(i=0; i<20; i++)
strcpy(a[i], "");
printf("Disk space before allocation:\n");
```

```
display();
do
{
printf("\nEnter File name (max 3 char) : ");
scanf("%s", id);
printf("Enter start block : ");
scanf("%d", &st);
printf("Enter no. of blocks : ");
scanf("%d", &nb);
strcpy(fid[num], id);
length[num] = nb;
flag = 0;
if((st+nb) > 20)
{
printf("Requirement exceeds range\n");
continue;
}
for(i=st; i<(st+nb); i++)
if(strcmp(a[i], "") != 0)
flag = 1;
if(flag == 1)
{
printf("Contiguous allocation not possible.\n");
continue;
}
start[num] = st;
for(i=st; i<(st+nb); i++)
strcpy(a[i], id);;
printf("Allocation done\n");
num++;
printf("\nAny more allocation (1. yes / 2. no)? : ");
scanf("%d", &ch);
} while (ch == 1);
printf("\n\t\t\tContiguous Allocation\n");
printf("Directory:");
directory();
printf("\nDisk space after allocation:\n");
display();
printf("\n");
}
```

OUTPUT:

```
$ ./a.out
Disk space before allocation:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Enter File name (max 3 char) : ls
Enter start block : 3
Enter no. of blocks : 4
Allocation done
Any more allocation (1. yes / 2. no)? : 1

Enter File name (max 3 char) : cp
Enter start block : 14
Enter no. of blocks : 3
Allocation done

Any more allocation (1. yes / 2. no)? : 1
```

Enter File name (max 3 char) : tr
Enter start block : 18
Enter no. of blocks : 3
Requirement exceeds range

Enter File name (max 3 char) : tr
Enter start block : 10
Enter no. of blocks : 3
Allocation done

Any more allocation (1. yes / 2. no)? : 1

Enter File name (max 3 char) : mv
Enter start block : 0
Enter no. of blocks : 2
Allocation done

Any more allocation (1. yes / 2. no)? : 1

Enter File name (max 3 char) : ps
Enter start block : 12
Enter no. of blocks : 3
Contiguous allocation not possible.

Enter File name (max 3 char) : ps
Enter start block : 7
Enter no. of blocks : 3
Allocation done

Any more allocation (1. yes / 2. no)? : 2

Contiguous Allocation
Directory:
File Start Length
ls 3 4
cp 14 3
tr 10 3
mv 0 2
ps 7 3
Disk space after allocation:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Mv mv ls ls ls ls ps psps tr tr tr cp cp cp

EX:NO:11 PROGRAM FOR DEADLOCK PREVENTION

AIM

To write a c program to implement deadlock detection.

ALGORITHM

1. Start the program
2. Get the number of process and type of resource
3. For each process get the number of instances of allocated resources type and maximum instances each resources type
4. Get the available instances of each resources type
5. Calculate the need matrix for every process
6. Add up the allocated resource of a process to available resources if need
7. The sequence of process which satisfies the above condition gives the result
8. Stop the process

PROGRAM:

```
#include<stdlib.h>
void main()
{
int cl[10][10],al[10][10],av[10],I,j,k,m,n,ne[10][10],flag=0;
printf("\n enter the matrix:");
scanf("%d%d",&m,&n);
printf("\n enter the claim matrix:")
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
scanf("%d",&cl[i][j]);
}
printf("\n enter allocated matrix");
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
scanf("%d",&al[i][j]);
}
}
printf("\n the need matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
ne[i][j]=cl[i][j]-al[i][j];
printf("\t%d",ne[i][j]);
}
```

```
printf(“\n”);
}
printf(“\n enter available matrix”);
for(i=0;i<n;i++)
scanf(“%d”,&av[i]);
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
printf(“\t%d”,cl[i][j]);
}
printf(“\n”);
}
printf(“\n allocated matrix:\n”);
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
printf(“\t%d”,al[i][j]);
}
printf(“\n”);
}
printf(“available matrix: \n”);
for(i=0;i<n;i++)
{
printf(“%d\t”,av[i]);
}
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
if(av[j]>=ne[i][j])
flag=1;
else
flag=0;
}
}
if(flag==0)
printf(“unsafe state”);
else
printf(“safe state”);
}
```

OUTPUT:

```
enter the matrix: 4      3
enter the claim matrix
3  2  2  6  1  3  3  1  4  4  2  2
enter the allocated matrix
1     0  0  5  1  1  2  1  1  0  0  2
the need matrix
2  2  2
1  0  2
1  0  3
4  2  0
enter the available matrix : 1  1  2
claim matrix
3  2  2
6  1  3
3  1  4
4  2  2
allocated matrix
1  0  0
5  1  1
2  1  1
0  0  2
available matrix
1  1  2
safe state
```

EX:NO:12 IMPLEMENT REMOTE PROCEDURE CALL

AIM

To write a Java program for implementing factorial of a given number using Remote Procedure Call.

ALGORITHM

INTERFACE

Step 1: Declare server's remote interface for Factorial by extending *Remote* interface.

IMPLEMENTATION

Step 1: Start the program.

Step 2: Define a class FactImpl by implementing the interface method.

Step 3: Define the interface method to find factorial of the number.

Step 4: Return the factorial value.

Step 5: Stop the program.

SERVER

Step 1: Start the program.

Step 2: Create an Interface object.

Step 3: Register the object with the RMI registry on the server machine using *rebind* Method.

Step 4: Stop the program.

CLIENT

Step 1: Start the program.

Step 2: Look for service in the server using lookup ().

Step 3: Get the value for which factorial has to be found from the user as command line input.

Step 4: Call the remote factorial ().

Step 5: Display the factorial value.

Step 6: Stop the program.

PROGRAM

// remote method interface--FactIntf.java

```
import java.rmi.*;
public interface FactIntf extends Remote
{
	long factorial(int n)throws RemoteException;
}
```

//Remote behaviour implementation--FactImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class FactImpl extends UnicastRemoteObject implements FactIntf
{
	public FactImpl() throws RemoteException { }
	public long factorial(int n)throws RemoteException
	{
		long f = 1;
		for(int i=n; i>0; i--)
			f *= i;
		return f;
	}
}
```

//Server that registers the service--FactServer.java

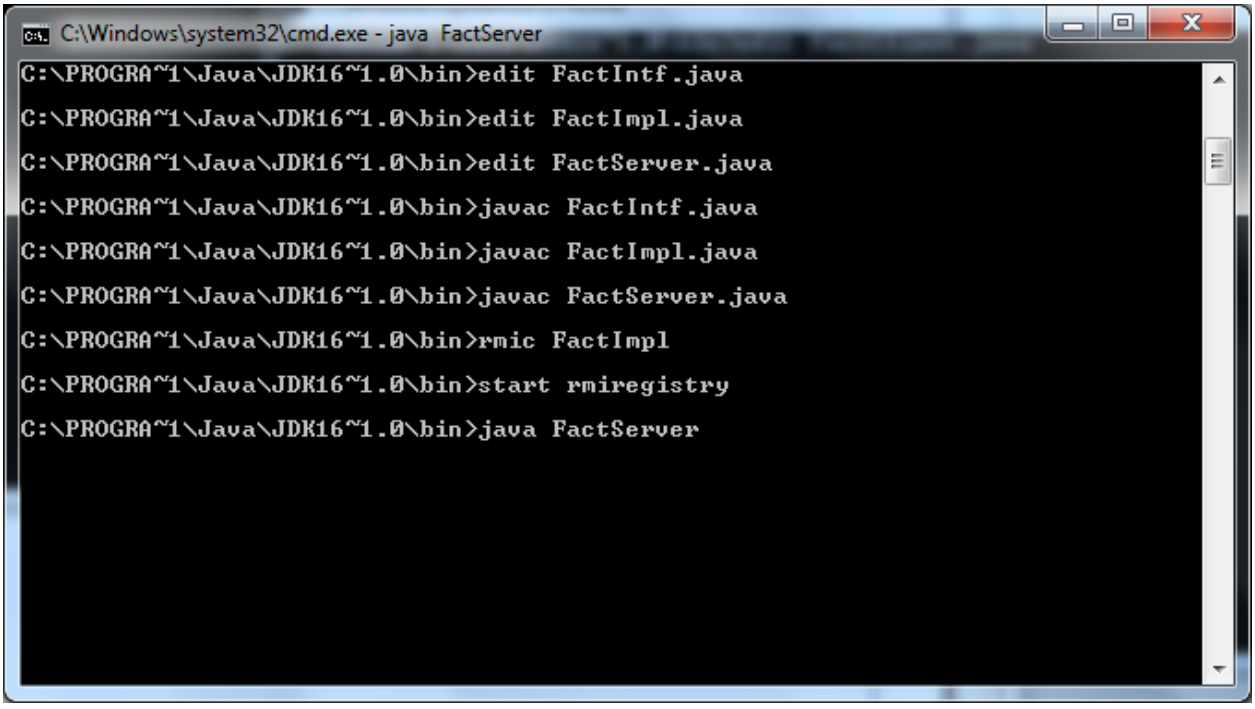
```
import java.rmi.*;
public class FactServer
{
	public static void main(String arg[])
	{
		try
		{
			FactIntf Fa = new FactImpl(); Naming.rebind("FactService", Fa);
		}
		catch(Exception e)
		{
			System.out.println(e.getMessage());
		}
	}
}
```

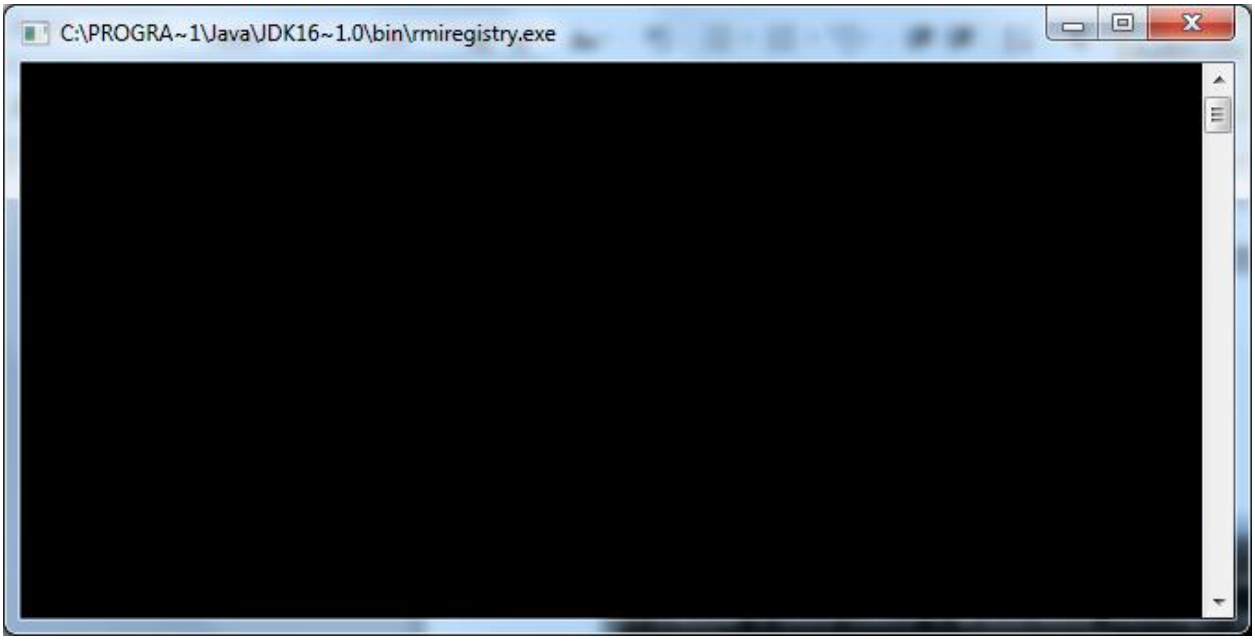
// Client that invokes remote host methods--FactClient.java

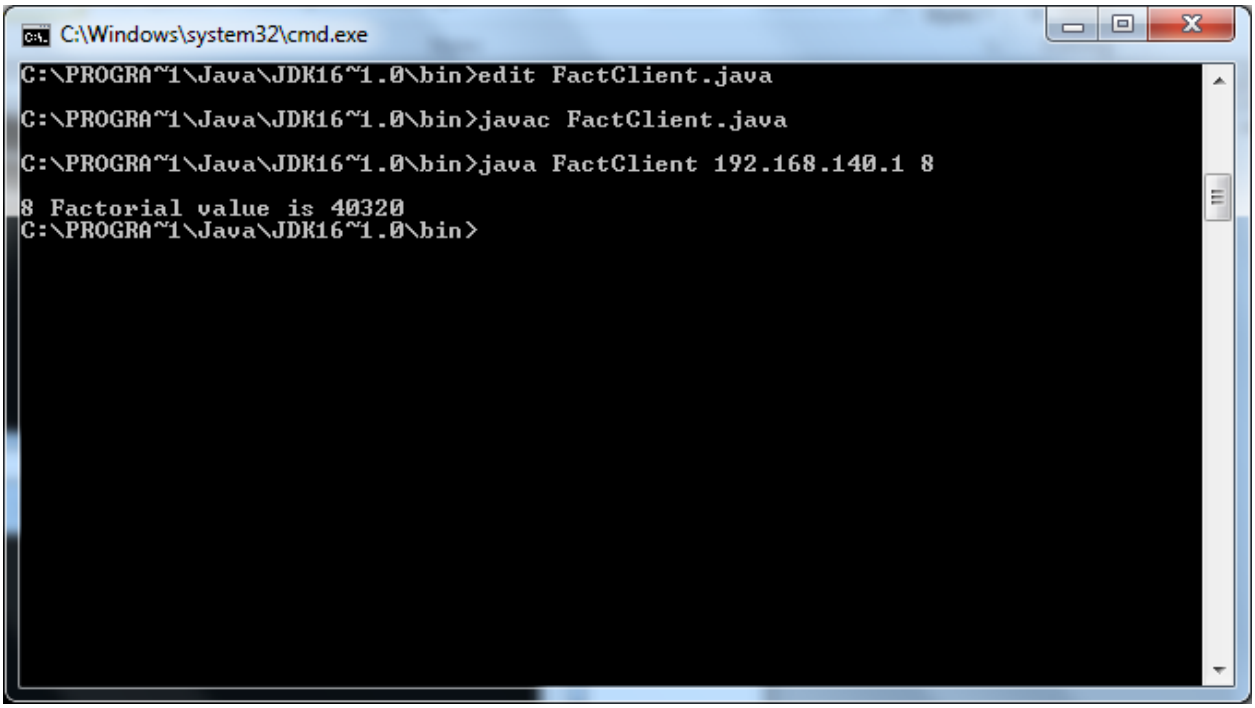
```
import java.rmi.*;
import java.net.*;
public class FactClient
{
	public static void main(String args[])
	{
		try
		{
			FactIntf Fa = (FactIntf) Naming.lookup("rmi://" + args[0] +
		"/FactService");
			if (args.length != 2)
```

```
                {
                        System.out.println("Usage: java FactClient <remoteip>
                <number>");
                        System.exit(-1);
                }
                int n = Integer.parseInt(args[1]);
                long factval = Fa.factorial(n);
                System.out.print("\n" + n + " Factorial value is " + factval);
        }
        catch(Exception e)
        {
                System.out.println(e.getMessage());
        }
    }
}
```

OUTPUT




C:\PROGRA~1\Java\JDK16~1.0\bin\rmiregistry.exe


C:\Windows\system32\cmd.exe
C:\PROGRA~1\Java\JDK16~1.0\bin>edit FactClient.java
C:\PROGRA~1\Java\JDK16~1.0\bin>javac FactClient.java
C:\PROGRA~1\Java\JDK16~1.0\bin>java FactClient 192.168.140.1 8
8 Factorial value is 40320
C:\PROGRA~1\Java\JDK16~1.0\bin>

CONTENT BEYOND SYLLABUS

EX:NO:13 IMPLEMENTATION OF BANKERS ALGORITHM

AIM:

To simulate Bankers algorithm for deadlock detection using C program.

ALGORITHM:

Step1: Start the program

Step 2: Get the nos of process and type of resource

Step 3:For each process,get the nos of instances of allocated resource type and max instances of each resource type

Step 4: Get the available instances of each resource type.

Step5: Calculate the need matrix for every process where need=max-allocation

Step 6:Add up the allocated resource of a process to available if need[i] available

Step 7:Stop the program.

PROGRAM:

```
#include<stdlib.h>
void main()
{
int cl[10][10],al[10][10],av[10],I,j,k,m,n,ne[10][10],flag=0;
printf("\n enter the matrix:");
scanf("%d%d",&m,&n);
printf("\n enter the claim matrix:")
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
scanf("%d",&cl[i][j]);
}
printf("\n enter allocated matrix");
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
scanf("%d",&al[i][j]);
```

```
}
}
printf(“\n the need matrix:\n”);
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
ne[i][j]=cl[i][j]-al[i][j];
printf(“\t%d”,ne[i][j]);
}
printf(“\n”);
}
printf(“\n enter available matrix”);
for(i=0;i<n;i++)
scanf(“%d”,&av[i]);
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
printf(“\t%d”,cl[i][j]);
}
printf(“\n”);
}
printf(“\n allocated matrix:\n”);
for(i=0;i<m;i++)
{
for(j=0;i<n;j++)
{
printf(“\t%d”,al[i][j]);
}
printf(“\n”);
}
printf(“available matrix: \n”);
for(i=0;i<n;i++)
{
printf(“%d\t”,av[i]);
}
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
if(av[j]>=ne[i][j])
flag=1;
else
flag=0;
}
}
if(flag==0)
printf(“unsafe state”);
```

```
else
printf("safe state");
}
```

OUTPUT:

```
enter the matrix: 4      3
enter the claim matrix
3  2  2  6  1  3  3  1  4  4  2  2
enter the allocated matrix
1     0  0  5  1  1  2  1  1  0  0  2
the need matrix
2  2  2
1  0  2
1  0  3
4  2  0
enter the available matrix : 1  1  2
claim matrix
3  2  2
6  1  3
3  1  4
4  2  2
allocated matrix
1  0  0
5  1  1
2  1  1
0  0  2
available matrix
1  1  2
safe state
```