

DRONACHARYA

College of Engineering

PRACTICAL FILE

Computer Science & Engineering

OPERATING SYSTEM LAB

NAME: PIYUSH GARG

BRANCH: CSE-1

SEM: 4th

ROLL NO: 22109



Certificate

Certified that this Practical entitled “ Operating System Lab ” submitted by PIYUSH GARG Roll No. 22109, students of Computer Science & Engineering Department, **Dronacharya College of Engineering, Gurgaon** in the partial fulfillment of the requirement for the award of Bachelors of Technology (Computer Science & Engineering) Degree of **MDU, Rohtak**, is a record of student’s own study carried under my supervision & guidance.

Signature
Mr. Naveen

Signature
Dr. Ashima Mehta
HOD(CSE)

INDEX

| Sr. No. | Exp. No. | Practical Name | Sign |
|---------|----------|---|------|
| 1. | 1 A | To study about the basics of UNIX | |
| 2. | 1 B | To study of Basic UNIX Commands | |
| 3. | 1 C | To study of various UNIX editors such as vi, ed, ex and EMACS. | |
| 4. | 2 | To write C Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir | |
| 5. | 3 | C programs to simulate UNIX commands like cp, ls, grep. | |
| 6. | 4 | To write simple shell programs by using conditional, branching and looping statements. | |
| 7. | 5 A | To write a C program for implementation of Priority scheduling algorithms. | |
| 8. | 5 B | To write a C program for implementation of FCFS | |
| 9. | 6 | To write a C-program to implement the producer – consumer problem using semaphores | |
| 10. | 7 | To write a c program to implement IPC using shared memory. | |
| 11. | 8 | To write a C program to implement banker's algorithm for deadlock avoidance. | |
| 12. | 9 | To write a C program to implement algorithm for deadlock detection. | |
| 13. | 10 | To write a C program to implement Threading and Synchronization Applications. | |

| | |
|------------------|--------------------------------|
| Ex.No:1.a | BASICS OF UNIX COMMANDS |
| | INTRODUCTION TO UNIX |

AIM:

To study about the basics of UNIX

UNIX:

It is a multi-user operating system. Developed at AT & T Bell Industries, USA in 1969.

Ken Thomson along with Dennis Ritchie developed it from MULTICS (Multiplexed Information and Computing Service) OS.

By 1980, UNIX had been completely rewritten using C language.

LINUX:

It is similar to UNIX, which is created by Linus Torualds. All UNIX commands works in Linux. Linux is a open source software. The main feature of Linux is coexisting with other OS such as windows and UNIX.

STRUCTURE OF A UNIX SYSTEM:

It consists of three parts.

- a)UNIX kernel
- b) Shells
- c) Tools and Applications

UNIX KERNEL:

Kernel is the core of the UNIX OS. It controls all tasks, schedule all Processes and carries out all the functions of OS.

Decides when one programs tops and another starts.

SHELL:

Shell is the command interpreter in the UNIX OS. It accepts command from the user and analyses and interprets them

| | |
|------------------|--------------------------------|
| Ex.No:1.b | BASICS OF UNIX COMMANDS |
| | BASIC UNIX COMMANDS |

AIM:

To study of Basic UNIX Commands.

CONTENT:

Note: Syn->Syntax

a) date

–used to check the date and time

Syn:\$date

| Format | Purpose | Example | Result |
|--------|-------------------------------------|-----------|--------|
| +%m | To display only month | \$date+%m | 06 |
| +%h | To display month name | \$date+%h | June |
| +%d | To display day of month | \$date+%d | 01 |
| +%y | To display last two digits of years | \$date+%y | 09 |
| +%H | To display hours | \$date+%H | 10 |
| +%M | To display minutes | \$date+%M | 45 |
| +%S | To display seconds | \$date+%S | 55 |

b) cal

–used to display the calendar

Syn:\$cal 2 2009

c) echo

–used to print the message on the screen.

Syn:\$echo “text”

d) ls

–used to list the files. Your files are kept in a directory.

Syn:\$ls-ls

All files (include files with prefix)

ls-l Lodash (provide file statistics)

ls-t Order by creation time

ls-u Sort by access time (or show when last accessed together with-l)

ls-s Order by size

ls-r Reverse order

ls-f Mark directories with /, executable with *, symbolic links with @, local sockets with =, named pipes(FIFOs)with

ls-s Show file size

ls-h “Human Readable”, show file size in Kilo Bytes & Mega Bytes (h can be used together with -l or

ls[a-m]*List all the files whose name begin with alphabets From „a“ to „m“ls[a]*List all the files whose name begins with „a“ or „A“
Eg:\$ls>my list Output of „ls“ command is stored to disk file named „my list“

e)lp

–used to take printouts

Syn:\$lp filename

f)man

–used to provide manual help on every UNIX commands.

Syn:\$man unix command

\$man cat

g)who & whoami

–it displays data about all users who have logged into the system currently. The next command displays about current user only.

Syn:\$who\$whoami

h) uptime

–tells you how long the computer has been running since its last reboot or power-off.

Syn:\$uptime

i)uname

–it displays the system information such as hardware platform, system name and processor, OS type.

Syn:\$uname–a

j) hostname

–displays and set system host name

Syn:\$ hostname

k) bc

–stands for “best calculator”

| | | | |
|---------------------|----------|----------|-----------|
| \$bc | \$ bc | \$ bc | \$ bc |
| 10/2*3 | scale =1 | ibase=2 | sqrt(196) |
| 15 | 2.25+1 | obase=16 | 14 quit |
| | 3.35 | 11010011 | |
| | quit | 89275 | |
| | | 1010 | |
| | | Ā | |
| | | Quit | |
| \$bc | \$ bc-l | | |
| for(i=1;i<3;i=i+1)I | scale=2 | | |
| 1 | s(3.14) | | |
| 2 | 0 | | |
| 3 quit | | | |

FILE MANIPULATION COMMANDS

a) **cat**—this create, view and concatenate files.

Creation:

Syn:\$cat>filename

Viewing:

Syn:\$cat filename

Add text to an existing file:

Syn:\$cat>>filename

Concatenate:

Syn:\$catfile1file2>file3

\$catfile1file2>>file3 (no over writing of file3)

b) **grep**—used to search a particular word or pattern related to that word from the file.

Syn:\$grep search word filename

Eg:\$grep anu student

c) **rm**—deletes a file from the file system

Syn:\$rm filename

d) **touch**—used to create a blank file.

Syn:\$touch file names

e) **cp**—copies the files or directories

Syn:\$cpsource file destination file

Eg:\$cp student stud

f) **mv**—to rename the file or directory

syn:\$mv old file new file

Eg:\$mv -i student student-list(-i prompt when overwrite)

g) **cut**—it cuts or pickup a given number of character or fields of the file.

Syn:\$cut<option><filename>

Eg: \$cut -c filename

\$cut-c1-10emp

\$cut-f 3,6emp

\$ cut -f 3-6 emp

-c cutting columns

-f cutting fields

h) **head**—displays10 lines from the head(top)of a given file

Syn:\$head filename

Eg:\$head student

To display the top two lines:

Syn:\$head-2student

i) **tail**—displays last 10 lines of the file

Syn:\$tail filename

Eg:\$tail student

To display the bottom two lines;

Syn:\$ tail -2 student

j) **chmod**—used to change the permissions of a file or directory.

Syn:\$Schmod category operation permission file

Where, Category—is the user type

Operation—is used to assign or remove permission

Permission—is the type of permission

File—are used to assign or remove permission all

Examples:

\$chmodu-wx student

Removes write and execute permission for users

\$ch modu+rw,g+rwstudent

Assigns read and write permission for users and groups

\$chmodg=rwx student

Assigns absolute permission for groups of all read, write and execute permissions

k) **wc**—it counts the number of lines, words, character in a specified file(s)

with the options as -l,-w,-c

| Category | Operation | Permission |
|-----------|--------------------|------------|
| u— users | +assign | r— read |
| g—group | -remove | w— write |
| o— others | =assign absolutely | x—execute |

Syn: \$wc -l filename

\$wc -w filename

\$wc -c filename

OUTPUT :-

```
piyush@LAPTOP-VIO25I13: ~
piyush@LAPTOP-VIO25I13:~$ date
Sun Jul 18 13:10:24 IST 2021
piyush@LAPTOP-VIO25I13:~$ cal
      July 2021
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 29 30 31

piyush@LAPTOP-VIO25I13:~$ echo "Operating System Lab"
Operating System Lab
piyush@LAPTOP-VIO25I13:~$ ls
a.out test1.c
piyush@LAPTOP-VIO25I13:~$ ls -l
total 24
-rwxr-xr-x 1 piyush piyush 17048 Jul 17 17:38 a.out
-rw-r--r-- 1 piyush piyush   377 Jul 17 17:38 test1.c
piyush@LAPTOP-VIO25I13:~$ whoami
piyush
piyush@LAPTOP-VIO25I13:~$ uptime
 13:10:51 up   6:58,  0 users,  load average: 0.00, 0.00, 0.00
piyush@LAPTOP-VIO25I13:~$ uname -a
Linux LAPTOP-VIO25I13 5.4.72-microsoft-standard-WSL2 #1 SMP Wed Oct 28 23:40:43 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
piyush@LAPTOP-VIO25I13:~$ hostname
LAPTOP-VIO25I13
piyush@LAPTOP-VIO25I13:~$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
10/2*3
15
sqrt(225)
15
quit
```

```
piyush@LAPTOP-VIO25I13: ~
piyush@LAPTOP-VIO25I13:~$ ls
a.out test1.c
piyush@LAPTOP-VIO25I13:~$ touch file.txt
piyush@LAPTOP-VIO25I13:~$ cat > file1.txt
Hello World!
^C
piyush@LAPTOP-VIO25I13:~$ cat file1.txt
Hello World!
piyush@LAPTOP-VIO25I13:~$ cp file1.txt file2.txt
piyush@LAPTOP-VIO25I13:~$ cat file2.txt
Hello World!
piyush@LAPTOP-VIO25I13:~$ grep World file2.txt
Hello World!
piyush@LAPTOP-VIO25I13:~$ ls
a.out file.txt file1.txt file2.txt test1.c
piyush@LAPTOP-VIO25I13:~$ wc -w file1.txt
 2 file1.txt
piyush@LAPTOP-VIO25I13:~$ rm file.txt
piyush@LAPTOP-VIO25I13:~$ ls
a.out file1.txt file2.txt test1.c
piyush@LAPTOP-VIO25I13:~$ rm file1.txt
piyush@LAPTOP-VIO25I13:~$ ls
a.out file2.txt test1.c
piyush@LAPTOP-VIO25I13:~$
```

| | |
|-----------|-------------------------|
| Ex.No:1.c | BASICS OF UNIX COMMANDS |
| | UNIX EDITORS |

AIM:

To study of various UNIX editors such as vi, ed, ex and EMACS.

CONCEPT:

Editor is a program that allows user to see a portions a file on the screen and modify characters and lines by simply typing at the current position. UNIX supports variety of Editors. They are:

ed ex vi
EMACS

Vi- vi stands for “visual”. vi is the most important and powerful editor.vi is a full screen editor that allows user to view and edit entire document at the same time. vi editor was written in the University of California, at Berkley by Bill Joy, who is one of the co-founder of Sun Microsystems.

Features of vi:

It is easy to learn and has more powerful features.

It works with great speed and is case sensitive. Vi has powerful undo functions and has 3 modes:

1. Command mode
2. Insert mode
3. Escape or ex mode

In command mode, no text is displayed on the screen.

In Insert mode, it permits user to edit insert or replace text.

In escape mode, it displays commands at command line.

Moving the cursor with the help of h, l, k, j, I, etc

EMACS Editor

Motion Commands:

M-> Move to end of file

M-< Move to beginning of file

C-v Move forward a screen M -v Move backward a screen C -n Move to next line

C-p Move to previous line

C-a Move to the beginning of the line

C-e Move to the end of the line

C-f Move forward a character

C-b Move backward a character

M-f Move forward a word

M-b Move backward a word

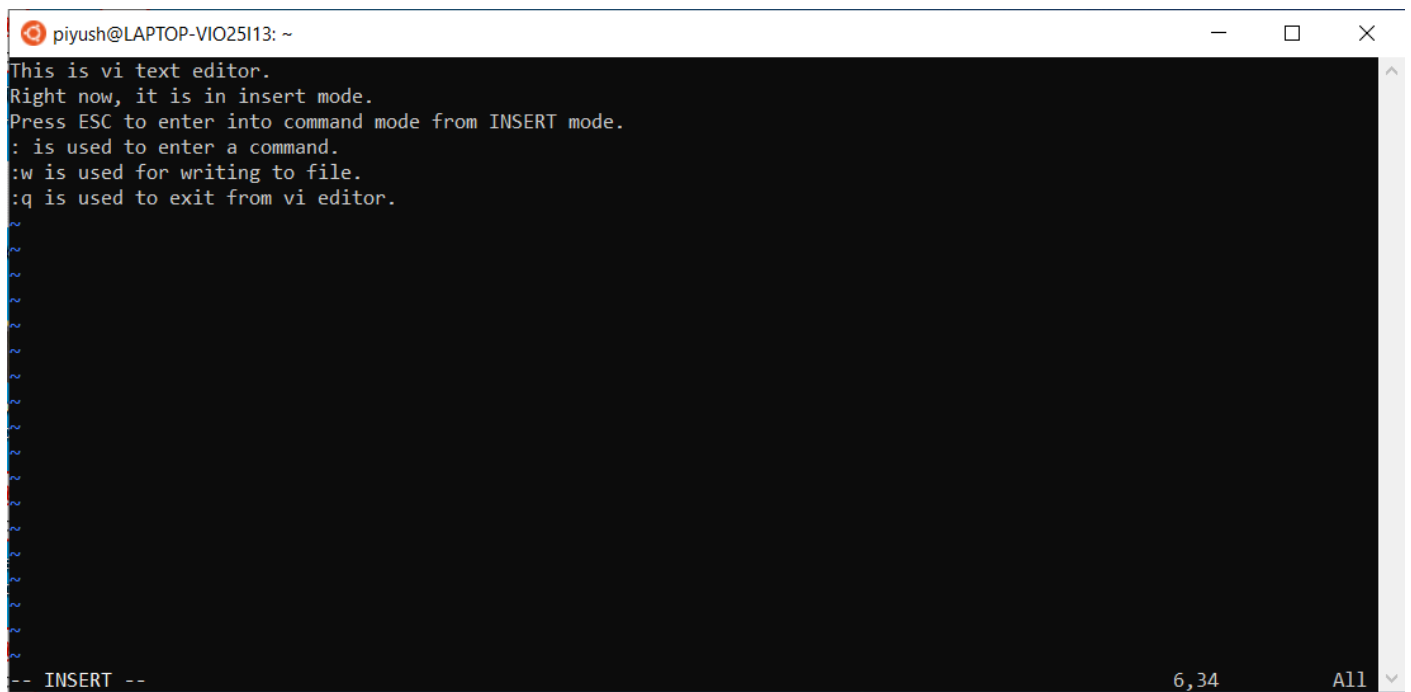
Deletion Commands:

| | |
|---------|---|
| DEL | delete the previous character C -d |
| | delete the current character M -DEL |
| | delete the previous word |
| M-d | delete the next word |
| C-x DEL | deletes the previous sentence |
| M-k | delete the rest of the current sentence |
| C-k | deletes the rest of the current line |
| C-xu | undo the lasted it change |

Search and Replace in EMACS:

| | |
|---|---|
| y | Change the occurrence of the pattern |
| n | Don't change the occurrence, but look for the other q Don't change. Leave query |
| | replace completely |
| ! | Change this occurrence and all others in the file |

OUTPUT :-



The screenshot shows a terminal window titled "piyush@LAPTOP-VIO25I13: ~". The terminal content displays the initial instructions for the vi text editor:

```
This is vi text editor.  
Right now, it is in insert mode.  
Press ESC to enter into command mode from INSERT mode.  
: is used to enter a command.  
:w is used for writing to file.  
:q is used to exit from vi editor.
```

The bottom status bar of the terminal shows "-- INSERT --" on the left, "6,34" in the center, and "All" on the right.

Ex.No:2

**Programs using the following system calls of UNIX operating system
fork, exec, getpid, exit, wait, close, stat, opendir, readdir**

AIM:

To write C Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

1. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (OPENDIR, READDIR, CLOSEDIR)


ALGORITHM:

STEP 1: Start the program.
STEP 2: Create struct dirent.
STEP 3: declare the variable buff and pointer dptr.
STEP 4: Get the directory name.
STEP 5: Open the directory.
STEP 6: Read the contents in directory and print it.
STEP 7: Close the directory.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
    char buff[100];
    DIR *dirp;
    printf("\n\n ENTER DIRECTORY NAME : ");
    scanf("%s", buff);
    if((dirp=opendir(buff))!=NULL)
    {
        printf("The given directory does not exist");
        exit(1);
    }
    while(dptr=readdir(dirp))
    {
        printf("%s\n",dptr->d_name);
    }
    closedir(dirp);
    return 0;
}
```

OUTPUT:

 piyush@LAPTOP-VIO25I13: ~/practicals

```
piyush@LAPTOP-VIO25I13:~/practicals$ cat p2.c
```

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
    char buff[100];
    DIR *dirp;
    printf("\n\n ENTER DIRECTORY NAME : ");
    scanf("%s", buff);
    if((dirp=opendir(buff))==NULL)
    {
        printf("The given directory does not exist");
        exit(1);
    }
    while(dptr=readdir(dirp))
    {
        printf("%s\n",dptr->d_name);
    }
    closedir(dirp);
    return 0;
}
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w p2.c
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out
```

```
ENTER DIRECTORY NAME : .
```

```
p2.c
```

```
a.out
```

```
..
```

```
.
```

```
piyush@LAPTOP-VIO25I13:~/practicals$
```

2. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEM (fork, getpid, exit)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2.

STEP 3: Call fork() system call to create process.

STEP 4: If pid==-1, exit.

STEP 5: If pid!=-1 , get the process id using getpid().

STEP 6: Print the process id.

STEP 7: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int pid,pid1,pid2;
    pid=fork();
    if(pid==-1)
    {
        printf("ERROR IN PROCESS CREATION \n");
        exit(1);
    }
    if(pid!=0)
    {
        pid1=getpid();
        printf("\n the parent process ID is %d\n", pid1);
    }
    else
    {
        pid2=getpid();
        printf("\n the child process ID is %d\n", pid2);
    }
}
```

```
}  
}
```

OUTPUT:

piyush@LAPTOP-VIO25I13: ~/practicals

```
piyush@LAPTOP-VIO25I13:~/practicals$ cat p2_1.c
```

```
#include<stdio.h>  
#include<unistd.h>  
int main()  
{  
    int pid,pid1,pid2;  
    pid=fork();  
    if(pid==-1)  
    {  
        printf("ERROR IN PROCESS CREATION \n");  
        exit(1);  
    }  
    if(pid!=0)  
    {  
        pid1=getpid();  
        printf("\n the parent process ID is %d\n", pid1);  
    }  
    else  
    {  
        pid2=getpid();  
        printf("\n the child process ID is %d\n", pid2);  
    }  
}
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w p2_1.c
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out
```

```
the parent process ID is 535
```

```
the child process ID is 536
```

```
piyush@LAPTOP-VIO25I13:~/practicals$
```

| | |
|----------------|--|
| Ex.No:3 | C programs to simulate UNIX commands like cp, ls, grep. |
|----------------|--|

AIM:

To write C programs to simulate UNIX commands like cp, ls, grep.

1.Program for simulation of cp unix commands**ALGORITHM:**

STEP1: Start the program

STEP 2: Declare the variables ch, *fp, sc=0

STEP3: Open the file in read mode

STEP 4: Get the character

STEP 5: If ch== " " then increment sc value by one

STEP 6: Print no of spaces

STEP 7: Close the file

PROGRAM:

```
#include<fcntl.h>
```

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

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

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

```
{
```

```
    FILE *fp; char ch; int sc=0;
```

```
    fp=fopen(argv[1], "r");
```

```
    if(fp==NULL)
```

```
        printf("unable to open a file", argv[1]);
```

```
    else
```

```
    {
```

```
        while(!feof(fp))
```

```
        {
```

```
            ch=fgetc(fp); if(ch==' ') sc++;
```

```
        }
```

```
        printf("no of spaces %d", sc);
```

```
        printf("\n");
```


```
        fclose(fp);
```

```
    }
```

```
    return 0;
```

```
}
```


OUTPUT:

 piyush@LAPTOP-VIO25I13: ~/practicals

```
piyush@LAPTOP-VIO25I13:~/practicals$ cat p3.c
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
int main(int argc,char *argv[])
{
    FILE *fp; char ch; int sc=0;

    fp=fopen(argv[1],"r");
    if(fp==NULL)
        printf("unable to open a file",argv[1]);
    else
    {
        while(!feof(fp))
        {
            ch=fgetc(fp); if(ch==' ') sc++;
        }
        printf("no of spaces %d",sc);
        printf("\n");
        fclose(fp);
    }
    return 0;
}
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w p3.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out p3.c
no of spaces 25
piyush@LAPTOP-VIO25I13:~/practicals$
```

2.PROGRAM FOR SIMULATION OF LS UNIX COMMANDS

ALGORTIHM:

STEP1 : Start the program

STEP2 : Open the directory with directory object dp

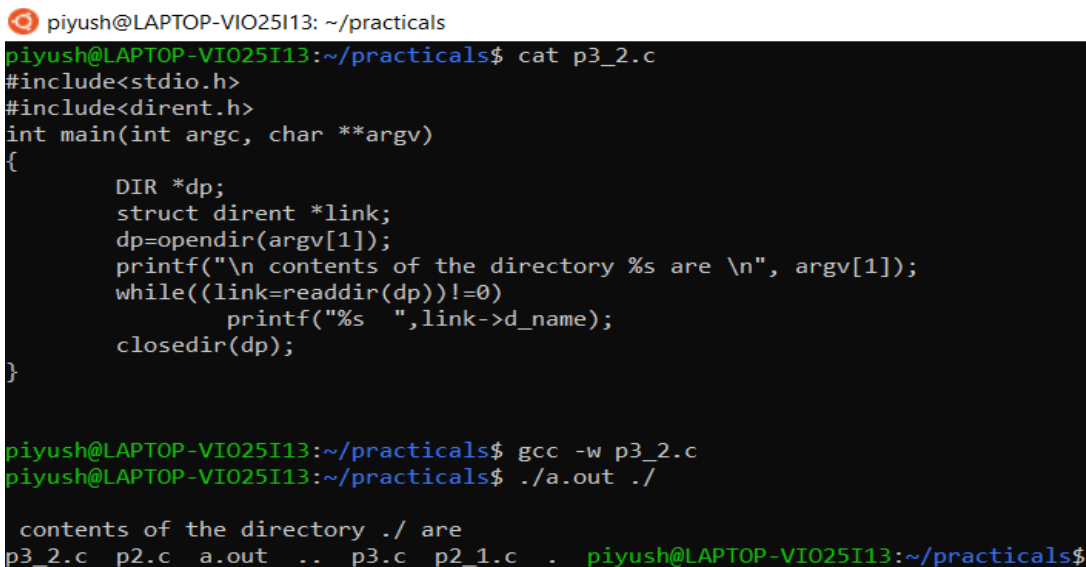
STEP3 : Read the directory content and print it.

STEP4: Close the directory.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
int main(int argc, char **argv)
{
    DIR *dp;
    struct dirent *link;
    dp=opendir(argv[1]);
    printf("\n contents of the directory %s are \n", argv[1]);
    while((link=readdir(dp))!=0)
        printf("%s ",link->d_name);
    closedir(dp);
}
```

OUTPUT:



```
piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ cat p3_2.c
#include<stdio.h>
#include<dirent.h>
int main(int argc, char **argv)
{
    DIR *dp;
    struct dirent *link;
    dp=opendir(argv[1]);
    printf("\n contents of the directory %s are \n", argv[1]);
    while((link=readdir(dp))!=0)
        printf("%s ",link->d_name);
    closedir(dp);
}

piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w p3_2.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out ./

contents of the directory ./ are
p3_2.c p2.c a.out .. p3.c p2_1.c . piyush@LAPTOP-VIO25I13:~/practicals$
```

3. PROGRAM FOR SIMULATION OF GREP UNIX COMMANDS

ALGORITHM

STEP1: Start the program

STEP2: Declare the variables fline[max], count=0, occurrences=0 and pointers *fp, *newline.

STEP 3: Open the file in read mode.

STEP 4: In while loop check fgets(fline,max,fp)!=NULL

STEP 5: Increment count value.

STEP 6: Check newline=strchr(fline, „\n“)

STEP 7: print the count,fline value and increment the occurrence value.

STEP 8: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define max 1024
void usage()
{
    printf(“usage:\t. /a.out filename word \n “);
}
int main(int argc, char *argv[])
{
    FILE *fp;
    char fline[max];
    char *newline;
    int count=0;
    int occurrences=0;
    if(argc!=3)
    {
        usage();
        exit(1);
    }
    if(!(fp=fopen(argv[1],”r”)))
    {
        printf(“grep: couldnot open file : %s \n”,argv[1]);
        exit(1);
    }
```

```

while(fgets(fline,max,fp)!=NULL)
{
    count++;
    if(newline=strchr(fline, "\n"))
        *newline= "\0";
    if(strstr(fline,argv[2])!=NULL)
    {
        printf("%s: %d %s \n", argv[1],count, fline);
        occurrences++;
    }
}
return 0;
}

```

OUTPUT

piyush@LAPTOP-VIO25I13: ~/practicals

```

piyush@LAPTOP-VIO25I13:~/practicals$ cat p3_3.c
#include<stdio.h>
#include<string.h>
#define max 1024
void usage()
{
    printf("usage:\t. /a.out filename word \n ");
}
int main(int argc, char *argv[])
{
    FILE *fp;
    char fline[max];
    char *newline;
    int count=0;
    int occurrences=0;
    if(argc!=3)
    {
        usage();
        exit(1);
    }
    if(!(fp=fopen(argv[1],"r")))
    {
        printf("grep: couldnot open file : %s \n",argv[1]);
        exit(1);
    }
    while(fgets(fline,max,fp)!=NULL)
    {
        count++;
        if(newline=strchr(fline, "\n"))
            *newline="\0";
        if(strstr(fline,argv[2])!=NULL)
        {
            printf("%s: %d %s \n", argv[1],count, fline);
            occurrences++;
        }
    }
    return 0;
}

piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w p3_3.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out p3_3.c exit
p3_3.c: 18          exit(1);

p3_3.c: 23          exit(1);

piyush@LAPTOP-VIO25I13:~/practicals$

```

Ex.No:4

SIMPLE SHELL PROGRAMS

AIM:

To write simple shell programs by using conditional, branching and looping statements.

1. Write a Shell program to check the given number is even or odd

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of n.

STEP 3: Calculate $r = \text{expr } \$n \% 2$.

STEP 4: If the value of r equals 0 then print the number is even

STEP 5: If the value of r not equal to 0 then print the number is odd.

PROGRAM:

```
echo "Enter the Number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is Even number"
else
echo "$n is Odd number"
fi
```

OUTPUT

```
piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ cat p4_1.sh
echo "Enter the Number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is Even number"
else
echo "$n is Odd number"
fi

piyush@LAPTOP-VIO25I13:~/practicals$ chmod 755 p4_1.sh
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_1.sh
Enter the Number
20
20 is Even number
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_1.sh
Enter the Number
19
19 is Odd number
piyush@LAPTOP-VIO25I13:~/practicals$
```

2. Write a Shell program to check the given year is leap year or not

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of year.

STEP 3: Calculate $b = \text{expr } \$y \% 4$.

STEP 4: If the value of b equals 0 then print the year is a leap year.

STEP 5: If the value of r not equal to 0 then print the year is not a leap year.

PROGRAM:

```
echo "Enter the year"
read y
b=`expr $y % 4`
if [ $b -eq 0 ]
then
echo "$y is a leap year"
else
echo "$y is not a leap year"
fi
```

OUTPUT

```
piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ cat p4_2.sh
echo "Enter the year : "
read y
b=`expr $y % 4`
if [ $b -eq 0 ]
then
echo "$y is a leap year"
else
echo "$y is not a leap year"
fi

piyush@LAPTOP-VIO25I13:~/practicals$ chmod 755 p4_2.sh
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_2.sh
Enter the year :
2020
2020 is a leap year
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_2.sh
Enter the year :
2015
2015 is not a leap year
piyush@LAPTOP-VIO25I13:~/practicals$
```

3. Write a Shell program to find the factorial of a number

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of n.

STEP 3: Calculate $i = n - 1$.

STEP 4: If the value of i is greater than 1 then calculate „ $n = n * i$ “ and „ $i = i - 1$ “

STEP 5: Print the factorial of the given number.

PROGRAM:

```
echo "Enter a Number"
read n
i=`expr $n - 1`
p=1
while [ $i -ge 1 ]
do
n=`expr $n \* $i`
i=`expr $i - 1`
done
echo "The Factorial of the given Number is $n"
```

OUTPUT

```
piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ cat p4_3.sh
echo "Enter a Number : "
read n
i=`expr $n - 1`
p=1
while [ $i -ge 1 ]
do
n=`expr $n \* $i`
i=`expr $i - 1`
done
echo "The Factorial of the given Number is $n"

piyush@LAPTOP-VIO25I13:~/practicals$ chmod 755 p4_3.sh
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_3.sh
Enter a Number :
7
The Factorial of the given Number is 5040
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_3.sh
Enter a Number :
5
The Factorial of the given Number is 120
piyush@LAPTOP-VIO25I13:~/practicals$
```

4. Write a Shell program to swap the two integers

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of a,b.

STEP 3: Calculate the swapping of two values by using a temporary variable temp.

STEP 4: Print the value of a and b.

PROGRAM:

```
echo "Enter Two Numbers"
```

```
read a b
```

```
temp=$a
```

```
a=$b
```

```
b=$temp
```

```
echo "after swapping"
```

```
echo $a $b
```

OUTPUT

```
piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ cat p4_4.sh
echo "Enter Two Numbers"
read a b
temp=$a
a=$b
b=$temp
echo "after swapping"
echo $a $b

piyush@LAPTOP-VIO25I13:~/practicals$ chmod 755 p4_4.sh
piyush@LAPTOP-VIO25I13:~/practicals$ ./p4_4.sh
Enter Two Numbers
5 10
after swapping
10 5
piyush@LAPTOP-VIO25I13:~/practicals$
```


| | |
|----------|---------------------------|
| Ex.No:5a | CPU SCHEDULING ALGORITHMS |
| | PRIORITY |

AIM:

To write a C program for implementation of Priority scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

Step 3: Get the value of „n“ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority.

Step 5: Assign wtime as zero .

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.

Step 8: Stop the program.

PROGRAM:

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

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

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

```
typedef struct
```

```
{
```

```
    int pno;
```

```
    int pri;
```

```
    int btime;
```

```
    int wtime;
```

```
}sp;
```

```
int main()
```

```
{
```

```
    int i,j,n;
```

```
    int tbm=0, totwtime=0, totttime=0;
```

```
    sp *p,t;
```

```
    printf("\n PRIORITY SCHEDULING.\n");
```

```
    printf("\n enter the no of process.    \n");
```

```
    scanf("%d",&n);
```

```
    p=(sp*)malloc(sizeof(sp));
```

```
    printf("enter the burst time and priority:\n");
```

```
    for(i=0;i<n;i++)
```

```
    {
```

```
        printf("process%d:",i+1); scanf("%d%d",&p[i].btime,&p[i].pri);
```

```
        p[i].pno=i+1;
```

```

        p[i].wtime=0;
    }
    for(i=0;i<n-1;i++)
    for(j=i+1;j<n;j++)
    {
        if(p[i].pri>p[j].pri)
        {
            t=p[i];
            p[i]=p[j];
            p[j]=t;
        }
    }
    printf("\n process\tbursttime\twaiting time\tturnaround time\n");
    for(i=0;i<n;i++)
    {
        totwtime+=p[i].wtime=tbm;
        tbm+=p[i].btime;
        printf("\n%d\t%d",p[i].pno,p[i].btime);
        printf("\t%d\t%d",p[i].wtime,p[i].wtime+p[i].btime);
    }
    totttime=tbm+totwtime;
    printf("\n total waiting time:%d",totwtime);
    printf("\n average waiting time:%f",(float)totwtime/n);
    printf("\n total turnaround time:%d",totttime);
    printf("\n avg turnaround time:%f",(float)totttime/n);
    return 0;
}

```

OUTPUT:

```

piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w 5a.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out

PRIORITY SCHEDULING.

enter the no of process.
6
enter the burst time and priority:
process1:4 4
process2:5 5
process3:1 7
process4:2 2
process5:3 1
process6:6 6

process      bursttime      waiting time      turnaround time
5            3            0            3
4            2            3            5
1            4            5            9
2            5            9            14
6            6            14            20
3            1            20            21
total waiting time:51
average waiting time:8.500000
total turnaround time:72
avg turnaround time:12.000000piyush@LAPTOP-VIO25I13:~/practicals$

```

| | |
|-----------|---------------------------|
| Ex.No:5.b | CPU SCHEDULING ALGORITHMS |
| | FCFS |

AIM:

To write a C program for implementation of FCFS

ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

Step 3: Get the value of „n“ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.

Step 6: Print total wait time and total turnaround time.

Step 7: Stop the program.

PROGRAM:


```
#include<stdio.h>
#include<stdlib.h>
struct fcfs
{
int pid;
int btime;
int wtime;
int ttime;
}
p[10];
int main()
{
int i,n;
int towtwtime=0,totttime=0;
printf("\n fcfs scheduling...\n");
printf("enter the no of process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i].pid=1;
printf("\n burst time of the process");
scanf("%d",&p[i].btime);
}
```

```

p[0].wtime=0;
p[0].ttime=p[0].btime;
totttime+=p[0].ttime;
for(i=1;i<n;i++)
{
p[i].wtime=p[i-1].wtime+p[i-1].btime;
p[i].ttime=p[i].wtime+p[i].btime;
totttime+=p[i].ttime;
towtwtime+=p[i].wtime;
}
printf("\n total waiting time :%d", towtwtime );
printf("\n average waiting time
:%f", (float)towtwtime/n); printf("\n total turn around
time :%d",totttime);
printf("\n average turn around time: :%f", (float)totttime/n);
}

```

OUTPUT:

 piyush@LAPTOP-VIO25I13: ~/practicals

```
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w 5b.c
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out
```

```
fcfs scheduling...
```

```
enter the no of process 5
```

```
burst time of the process 3
```

```
burst time of the process 1
```

```
burst time of the process 2
```

```
burst time of the process 5
```

```
burst time of the process 4
```

```
total waiting time :24
```

```
average waiting time :4.800000
```

```
total turn around time :39
```

```
average turn around time: :7.800000piyush@LAPTOP-VIO25I13:~/practicals$
```

Ex.No:6

PRODUCER CONSUMER PROBLEM USING SEMAPHORES

AIM:

To write a C-program to implement the producer – consumer problem using semaphores.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Initialize the buffer size and get maximum item you want to produce.

Step 4: Get the option, which you want to do either producer, consumer or exit from the operation.

Step 5: If you select the producer, check the buffer size if it is full the producer should not produce the item or otherwise produce the item and increase the value buffer size.

Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not consume the item or otherwise consume the item and decrease the value of buffer size.

Step 7: If you select exit come out of the program.

Step 8: Stop the program.

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("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1) {
printf("\nENTER YOUR CHOICE\n");
scanf("%d",&n);
switch(n)
{ case 1:
if((mutex==1)&&(empty!=0))
producer();
else
printf("BUFFER IS FULL");
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("\nproducer produces the item%d",x);
mutex=signal(mutex); }
void consumer() {
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes
item%d",x); x--;
mutex=signal(mutex); }
```

OUTPUT:-

 piyush@LAPTOP-VIO25I13: ~/practicals

```
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w 6.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out
```

```
1.PRODUCER
2.CONSUMER
3.EXIT
```

ENTER YOUR CHOICE

2

BUFFER IS EMPTY

ENTER YOUR CHOICE

1

producer produces the item1

ENTER YOUR CHOICE

1

producer produces the item2

ENTER YOUR CHOICE

2

consumer consumes item2

ENTER YOUR CHOICE

1

producer produces the item2

ENTER YOUR CHOICE

1

producer produces the item3

ENTER YOUR CHOICE

1

BUFFER IS FULL

ENTER YOUR CHOICE

2

consumer consumes item3

ENTER YOUR CHOICE

3

```
piyush@LAPTOP-VIO25I13:~/practicals$
```

| | |
|----------------|--------------------------------|
| Ex.No:7 | IPC USING SHARED MEMORY |
|----------------|--------------------------------|

AIM:

To write a c program to implement IPC using shared memory.

ALGORITHM:

- Step 1: Start the process
- Step 2: Declare the segment size
- Step 3: Create the shared memory
- Step 4: Read the data from the shared memory
- Step 5: Write the data to the shared memory
- Step 6: Edit the data
- Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#define SEGSIZE 100
int main(int argc, char *argv[ ])
{
    int shmid,cntr;
    key_t key;
    char *segptr;
    char buff[]="DCE ..... ";
    key=ftok(".", 's');
    if((shmid=shmget(key, SEGSIZE, IPC_CREAT | IPC_EXCL | 0666))== -1)
    {
        if((shmid=shmget(key,SEGSIZE,0))== -1)
        {
            perror("shmget");
            exit(1);
        }
    }
    else
    {
```




```

printf("Creating a new shared memory seg \n");
printf("SHMID:%d",shmld);
}
system("ipcs -m");
if((segptr=(char*)shmat(shmid,0,0))==(char*)-1)
{
perror("shmat");
exit(1);
}
printf("Writing data to shared memory...\n");
strcpy(segptr,buff);
printf("DONE\n");
printf("Reading data from shared
memory...\n"); printf("DATA:-%s\n",segptr);
printf("DONE\n");
printf("Removing shared memory Segment...\n");
if(shmctl(shmid,IPC_RMID,0)== -1)
printf("Can't Remove Shared memory Segment...\n");
else
printf("Removed Successfully");
}

```

OUTPUT:

 piyush@LAPTOP-VIO25I13: ~/practicals

```
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w 7.c
```

```
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out
```

```
Creating a new shared memory seg
```

```
----- Message Queues -----
```

| key | msqid | owner | perms | used-bytes | messages |
|-----|-------|-------|-------|------------|----------|
|-----|-------|-------|-------|------------|----------|

```
----- Shared Memory Segments -----
```

| key | shmid | owner | perms | bytes | nattch | status |
|------------|-------|--------|-------|-------|--------|--------|
| 0x73100654 | 2 | piyush | 666 | 100 | 0 | |
| 0x73100654 | 2 | piyush | 666 | 100 | 0 | |
| 0x73100654 | 2 | piyush | 666 | 100 | 0 | |

```
----- Semaphore Arrays -----
```

| key | semid | owner | perms | nsems |
|-----|-------|-------|-------|-------|
|-----|-------|-------|-------|-------|

```
SHMID:2Writing data to shared memory...
```

```
DONE
```

```
Reading data from shared memory...
```

```
DATA:-DCE.....
```

```
DONE
```

```
Removing shared memory Segment...
```

```
Removed Successfullypiyush@LAPTOP-VIO25I13:~/practicals$
```

Ex.No:8

BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

AIM:

To write a C program to implement banker's algorithm for deadlock avoidance.

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: Compare each and every process using the banker's algorithm.

Step-5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process

Step-6: produce the result of state of process

Step-7: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
    int i,j;
    printf("***** Baner's Algo *****\n");
    input();
    show();
    cal();
    getch();
    return 0;
}
void input()
{
    int i,j;
    printf("Enter the no of Processes\t");
```

```

scanf("%d",&n);
printf("Enter the no of resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}}
void show()
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
for(j=0;j<r;j++)
{
printf("%d\t ",alloc[i][j]);
}
printf("\t");
for(j=0;j<r;j++)
{
printf("%d\t ",max[i][j]);
}
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
printf("%d\t ",avail[j]);
}}
}
void cal()

```

```

{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int safe[100];
int i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
}}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("P%d->",i);
if(finish[i]==1)
{
i=n;
}}}}}}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;


```

```

}
else
{printf("P%d->",i);
}}
if(c1==n)
{printf("\n The system is in safe state");
}
Else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}}

```

OUTPUT:

 piyush@LAPTOP-VIO25I13: ~/practicals

```

piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w 8.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out
***** Baner's Algo *****
Enter the no of Processes          5
Enter the no of resources instances      3
Enter the Max Matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available Resources
3 3 2
Process  Allocation      Max      Available
P1       0 1 0   7 5 3   3 3 2
P2       2 0 0   3 2 2
P3       3 0 2   9 0 2
P4       2 1 1   2 2 2
P5       0 0 2   4 3 3
P0->P1->P2->P3->P4->
The system is in safe statepiyush@LAPTOP-VIO25I13:~/practicals$

```

Ex.No:9

ALGORITHM FOR DEADLOCK DETECTION

AIM:

To write a C program to implement algorithm for deadlock detection.

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: Compare each and every process using the banker's algorithm.

Step-5: If the process is in safe state then it is not a deadlock process otherwise it is a deadlock process

Step-6: produce the result of state of process

Step-7: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
    int i,j;
    printf("***** Deadlock Detection Algo *****\n");
    input();
    show();
    cal();
    getch();
    return 0;
}
void input()
{int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
```

```

printf("Enter the no of resource instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{ for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
} }
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{ for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
} }
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
} }
void show()
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
for(j=0;j<r;j++)
{
printf("%d ",alloc[i][j]);
}
printf("\t");
for(j=0;j<r;j++)
{ printf("%d ",max[i][j]);
}
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
printf("%d ",avail[j]);
} } }
void cal()
{ int finish[100],temp,need[100][100],flag=1,k,c1=0;
int dead[100];
int safe[100];
int i,j;

```

```

for(i=0;i<n;i++)
{ finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{ for(j=0;j<r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
} }
while(flag)
{ flag=0;
for(i=0;i<n;i++)
{ int c=0;
for(j=0;j<r;j++)
{ if((finish[i]==0)&&(need[i][j]<=avail[j]))
{ c++;
if(c==r)
{
for(k=0;k<r;k++)
{ avail[k]+=alloc[i][k];
finish[i]=1;
flag=1;
} //printf("\nP%d",i);
if(finish[i]==1)
{ i=n;
} } } } }
j=0;
flag=0;
for(i=0;i<n;i++)
{
if(finish[i]==0)
{ dead[j]=i;
j++;
flag=1;
} }
if(flag==1)
{
printf("\n\nSystem is in Deadlock and the Deadlock process are\n");
for(i=0;i<n;i++)
{ printf("P%d\t",dead[i]);
} }
else
{
printf("\nNo Deadlock Occur"); } }

```


OUTPUT:

```
piyush@LAPTOP-VI025I13:~/practicals$ ./a.out
***** Deadlock Detection Algo *****
Enter the no of Processes      5
Enter the no of resource instances    3
Enter the Max Matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available Resources
3 3 2
Process  Allocation      Max      Available
P1       0 1 0  7 5 3    3 3 2
P2       2 0 0  3 2 2
P3       3 0 2  9 0 2
P4       2 1 1  2 2 2
P5       0 0 2  4 3 3
No Deadlock Occurpiyush@LAPTOP-VI025I13:~/practicals$
```

AIM:

To write a c program to implement Threading and Synchronization Applications.

ALGORITHM:

- Step 1: Start the process
- Step 2: Declare process thread, thread-id.
- Step 3: Read the process thread and thread state.
- Step 4: Check the process thread equals to thread-id by using if condition.
- Step 5: Check the error state of the thread.
- Step 6: Display the completed thread process.
- Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
void* doSomething(void *arg)
{
    unsigned long i = 0;
    pthread_t id = pthread_self();

    if(pthread_equal(id,tid[0]))
    {
        printf("\n First thread processing\n");
    }
    else
    {
        printf("\n Second thread processing\n");
    }

    for(i=0; i<(0xFFFFFFFF);i++);
    return NULL;
}
int main(void)
{
    int i = 0;
```

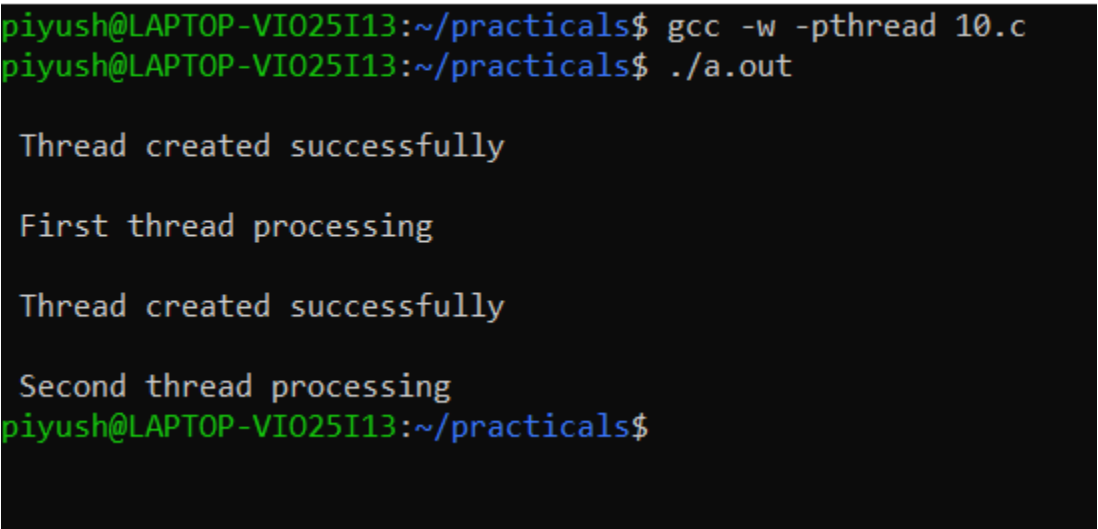
```
int err;

while(i < 2)
{
    err = pthread_create(&(tid[i]), NULL, &doSomething, NULL);
    if (err != 0)
        printf("\ncan't create thread :[%s]", strerror(err));
    else
        printf("\n Thread created successfully\n");

    i++;
}

sleep(5);
return 0;
}
```

OUTPUT:



```
piyush@LAPTOP-VIO25I13: ~/practicals
piyush@LAPTOP-VIO25I13:~/practicals$ gcc -w -pthread 10.c
piyush@LAPTOP-VIO25I13:~/practicals$ ./a.out

Thread created successfully

First thread processing

Thread created successfully

Second thread processing
piyush@LAPTOP-VIO25I13:~/practicals$
```