

PSNA COLLEGE OF ENGINEERING AND TECHNOLOGY

KOTHANDARAMAN NAGAR, DINDIGUL-624 622

Phone: 0451-2554032, 2554349 Web Link: www.psnacet.edu.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CS8461 - OPERATING SYSTEMS LABORATORY LAB MANUAL

PSNA
COLLEGE OF ENGINEERING AND TECHNOLOGY
DINDIGUL – 624 622.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CS8461-OPERATING SYSTEMS LABORATORY
LAB MANUAL
(REGULATION – 2017)
FOR
IV SEMESTER B.E-CSE

LIST OF EXPERIMENTS

CYCLE-I

1. Basics of UNIX commands.
2. Shell Programming.
3. Implement the following CPU scheduling algorithms
 - a) FCFS
 - b) SJFS
 - c) Priority
 - d) Round Robin
4. Implement Shared memory and IPC
5. Implement Bankers Algorithm for Dead Lock Avoidance
6. Implement an Algorithm for Dead Lock Detection

CYCLE-II

7. Implement all file allocation strategies
 - a) Sequential
 - b) Indexed
 - c) Linked
8. Implement Semaphores
9. Implement all File Organization Techniques
 - a) Single level directory
 - b) Two level
 - c) Hierarchical
 - d) DAG
10. Implement all page replacement algorithms
 - a) FIFO
 - b) LRU
 - c) LFU
11. Implement Paging Technique of memory management.
12. Implement Threading & Synchronization Applications

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

Standalone desktops with C / C++ / Java / Equivalent compiler 30 Nos.

(or)

Server with C / C++ / Java / Equivalent compiler supporting 30 terminals.

STUDY OF UNIX COMMANDS

Ex No:1

Date:

AIM:

To study and execute the UNIX Commands.

Introduction to UNIX:

UNIX was developed by AT&T Bell laboratories by Ken Thompson and Dennis Ritchie with the intension of creating a portable operating system. It was finally written in C and now runs on all machines from notebooks to mainframes. The term UNIX is derived from UNICS (UNiplexed Information Computing System).It is an interactive, multiuser and multitasking operating system and it is suitable for programming and system research projects. It supports individual user, small groups or entire departments on a wide range. It manages memory and disk space.

UNIX COMMANDS

1. Date Command:

This command is used to display the current data and time.

Syntax:

\$date

2. Calender Command:

This command is used to display the calendar of the year or the particular month of calendar year.

Syntax:

- a. \$cal <year>
- b. \$cal <month> <year>

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

3. Echo Command:

This command is used to print the arguments on the screen.

Syntax:

\$echo <text>

4. Banner Command:

It is used to display the text in '#' symbol .It displays the text in the form of a banner.

Syntax :

\$banner <arguments>

5.'who' Command:

It is used to display who are the users connected to our computer currently.

Syntax:

\$who

6.'who am i' Command:

Display the details of the current working directory.

Syntax:

\$who am i

7.'tty' Command:

It will display the terminal name.

Syntax:

\$tty

8.'CLEAR' Command:

It is used to clear the screen.

Syntax:

\$clear

9.'MAN' Command:

It helps us to know about the particular command and its options & working. It is like 'help' command in windows.

Syntax:

\$man <command name>

10. LIST Command:

It is used to list all the contents in the current working directory.

Syntax:

\$ ls – options <arguments>

If the command does not contain any argument means it is working in the Current directory.

Options:

a- used to list all the files including the hidden files.

c- list all the files columnwise.

d- list all the directories.

m- list the files separated by commas.

p- list files include '/' to all the directories.

r- list the files in reverse alphabetical order.

f- list the files based on the list modification date.

x-list in column wise sorted order.

DIRECTORY RELATED COMMANDS:**1. Present Working Directory Command:**

To print the complete path of the current working directory.

Syntax:

\$pwd

2. MKDIR Command:

To create or make a new directory in a current directory.

Syntax:

\$mkdir <directory name>

3. CD Command:

To change or move the directory to the mentioned directory .

Syntax :

\$cd <directory name>.

4. RMDIR Command:

To remove a directory in the current directory & not the current directory itself.

Syntax:

\$rmdir <directory name>

FILE RELATED COMMANDS:**1. CREATE A FILE:**

To create a new file in the current directory we use CAT command.

Syntax:

\$cat > filename.

2. DISPLAY A FILE:

To display the content of file mentioned we use CAT command without '>' operator.

Syntax:

\$cat filename.

3. COPYING CONTENTS:

To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.

Syntax :

\$ cat <source filename> >> <destination filename>

4. SORTING A FILE:

To sort the contents in alphabetical order in reverse order.

Syntax:

\$sort <filename >

Option:

\$ sort -r <filename>

5. COPYING CONTENTS FROM ONE FILE TO ANOTHER:

To copy the contents from source to destination file so that both contents are same.

Syntax:

\$cp <source filename> <destination filename>

\$cp <source filename path > <destination filename path>

6. MOVE Command:

To completely move the contents from source file to destination file and to remove the source file.

Syntax:

\$ mv <source filename> <destination filename>

7. REMOVE Command:

To permanently remove the file we use this command.

Syntax:

\$rm <filename>

8. WORD Command:

To list the content count of no of lines, words, characters.

Syntax:

\$wc<filename>

Options:

-c – to display no of characters.

-l – to display only the lines.

-w – to display the no of words.

9. LINE PRINTER:

To print the line through the printer, we use lp command.

Syntax:

\$lp <filename>

10. PAGE Command:

This command is used to display the contents of the file page wise & next page can be viewed by pressing the enter key.

Syntax:

\$pg <filename>

11. FILTERS AND PIPES

HEAD: It is used to display the top ten lines of file.

Syntax: \$head<filename>

TAIL: This command is used to display the last ten lines of file.

Syntax: \$tail<filename>

PAGE: This command shows the page by page a screenfull of information is displayed after which the page command displays a prompt and passes for the user to strike the enter key to continue scrolling.

Syntax: \$ls -a\p

MORE: It also displays the file page by page .To continue scrolling with more command, press the space bar key.

Syntax: \$more<filename>

GREP: This command is used to search and print the specified patterns from the file.

Syntax: \$grep [option] pattern <filename>

SORT: This command is used to sort the datas in some order.

Syntax: \$sort<filename>

PIPE: It is a mechanism by which the output of one command can be channeled into the input of another command.

Syntax: \$who | wc-l

TR: The tr filter is used to translate one set of characters from the standard inputs to another.

Syntax: \$tr "[a-z]" "[A-Z]"

COMMUNICATION THROUGH UNIX COMMANDS

Command: **MESG**

Description: The message command is used to give permission to other users to send message to your terminal.

Syntax: \$mesg y

Command: **WRITE**

Description: This command is used to communicate with other users, who are logged in at the same time.

Syntax: \$write <user name>

Command: **WALL**

Description: This command sends message to all users those who are logged in using the unix server.

Syntax: \$wall <message>

Command: **MAIL**

Description: It refers to textual information, that can be transferred from one user to another

Syntax: \$mail <user name>

Command: **REPLY**

Description: It is used to send reply to specified user.

Syntax: \$reply<user name>

vi EDITOR COMMANDS

The vi editor is a visual editor used to create and edit text, files, documents and programs. It displays the content of files on the screen and allows a user to add, delete or change part of text. There are three modes available in the vi editor, they are

1. Command mode
2. Input (or) insert mode.

The vi editor is invoked by giving the following commands in UNIX prompt.

Syntax: \$vi <filename> (or)
 \$vi

This command would open a display screen with 25 lines and with tilt (~) symbol at the start of each line. The first syntax would save the file in the filename mentioned and for the next the filename must be mentioned at the end.

Options :

1. vi +n <filename> - this would point at the nth line (cursor pos).
2. vi -n <filename> - This command is to make the file to read only to change from one mode to another press escape key.

INSERTING AND REPLACING COMMANDS:

To move editor from command mode to edit mode, you have to press the <ESC> key. For inserting and replacing the following commands are used.

1. ESC a Command:

This command is used to move the edit mode and start to append after the current character.

Syntax : <ESC> a

2. ESC A COMMAND :

This command is also used to append the file , but this command append at the end of current line.

Syntax: <ESC> A

3. ESC i Command:

This command is used to insert the text before the current cursor position.

Syntax: <ESC> i

4. ESC I Command:

This command is used to insert at the beginning of the current line.

Syntax : <ESC> I

5. ESC o Command:

This command is insert a blank line below the current line & allow insertion of contents.

Syntax: <ESC> o

6. ESC O Command:

This command is used to insert a blank line above & allow insertion of contents.

Syntax : <ESC> O

7. ESC r Command :

This command is to replace the particular character with the given characters.

Syntax: <ESC> rx Where x is the new character.

8. ESC R Command:

This command is used to replace the particular text with a given text.

Syntax: <ESC> R text

9. <ESC> s Command:

This command replaces a single character with a group of character.

Syntax: <ESC> s

10.<ESC> S Command :

This command is used to replace a current line with group of characters.

Syntax : <ESC> S

CURSOR MOVEMENT IN vi :

1.<ESC> h :

This command is used to move to the previous character typed. It is used to move to left of the text . It can also used to move character by character (or) a number of characters.

Syntax : <ESC> h – to move one character to left.

<ESC> nh – to move 'n' character to left.

2.<ESC> l :

This command is used to move to the right of the cursor (ie) to the next character. It can also be used to move the cursor for a number of character.

Syntax : <ESC> l – single character to right.

<ESC> nl - 'n' characters to right.

3.<ESC> j :

This command is used to move down a single line or a number of lines.

Syntax :

<ESC> j – single down movement.

<ESC> nj – 'n' times down movement.

4.<ESC> k :

This command is used to move up a single line or a number of lines.

Syntax :

<ESC> k – single line above.
<ESC> nk – ‘n’ lines above.

5. ENTER (OR) N ENTER :

This command will move the cursor to the starting of next lines or a group of lines mentioned.

Syntax :

<ESC> enter <ESC> n enter.

6. <ESC> + Command :

This command is used to move to the beginning of the next line.

Syntax :

<ESC> + <ESC> n+

7. <ESC> - Command :

This command is used to move to the beginning of the previous line.

Syntax :

<ESC> - <ESC> n-

8. <ESC> 0 :

This command will bring the cursor to the beginning of the same current line.

Syntax :

<ESC> 0

9. <ESC> \$:

This command will bring the cursor to the end of the current line.

Syntax :

<ESC> \$

10. <ESC> ^ :

This command is used to move to first character of first lines.

Syntax :

<ESC> ^

11. <ESC> b Command :

This command is used to move back to the previous word (or) a number of words.

Syntax :

<ESC> b <ESC> nb

12. <ESC> e Command :

This command is used to move towards and replace the cursor at last character of the word (or) no of words.

Syntax :

<ESC> e <ESC> ne

13.<ESC> w Command :

This command is used to move forward by a single word or a group of words.

Syntax :

<ESC> w <ESC> nw

DELETING THE TEXT FROM vi :

1.<ESC> x Command :

To delete a character to right of current cursor positions , this command is used.

Syntax :

<ESC> x <ESC> nx

2.<ESC> X Command :

To delete a character to left of current cursor positions , this command is used.

Syntax :

<ESC> X <ESC> nX

3.<ESC> dw Command :

This command is to delete a single word or number of words to right of current cursor position.

Syntax :

<ESC> dw <ESC> ndw

4.db Command :

This command is to delete a single word to the left of the current cursor position.

Syntax :

<ESC> db <ESC> ndb

5.<ESC> dd Command :

This command is used to delete the current line (or) a number of line below the current line.

Syntax :

<ESC> dd <ESC> ndd

6.<ESC> d\$ Command :

This command is used to delete the text from current cursor position to last character of current line.

Syntax : <ESC> d\$

SAVING AND QUITTING FROM vi :-

1.<ESC> w Command :

To save the given text present in the file.

Syntax : <ESC> : w

2.<ESC> q! Command :

To quit the given text without saving.

Syntax : <ESC> :q!

3.<ESC> wq Command :

This command quits the vi editor after saving the text in the mentioned file.

Syntax : <ESC> :wq

4.<ESC> x Command :

This command is same as 'wq' command it saves and quit.

Syntax : <ESC> :x

5.<ESC> q Command :

This command would quit the window but it would ask for again to save the file.

Syntax : <ESC> : q

CONCLUSION:

The various Unix & Vi Editor commands were studied and executed successfully.

EX.NO:2 PROGRAM TO SIMULATE SYSTEM CALLS IN UNIX OPERATING SYSTEM

Aim:

To execute the program for System calls in unix.

a) Program for system calls(OPENDIR,READDIR,CLOSEDIR)

Algorithm:

1. Read the directory name from the user.
2. Open the directory.
3. If the directory is not found, display an error.
4. Otherwise read the directory.
5. Display the files in the directory.
6. 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);
}
```

b) Program for System calls(Fork,Getpid,Exit)

Algorithm:

1. Assign pid and assign the values using fork() function.
2. If the pid value is negative, display an error.
3. If the pid value is not equal to zero, display the parent process id.
4. Otherwise display the child process id.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
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);
}
}
```

c) Program for System call (LS)

Algorithm:

1. Read a directory name.
2. Open the directory.
3. If the directory is empty, display no files in the directory.
4. Otherwise list all the file names in the directory

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
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);
}
```

EX.NO:3 PROGRAM TO SIMULATE UNIX COMMANDS(GREP,CP).

Aim:

To execute the program to simulate unix commands.

a)Program to simulate grep command.

Algorithm:

1. Read the file name and pattern to be searched.
2. Open the file.
3. If there is no contents in the file ,display the file is empty.
4. Otherwise compare the pattern with each line in the file.
5. If the pattern is same as the starting word of the file ,display the line.
6. Close the file.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
    Char fn[10],pat[10],temp[100];
    File *fp;
    Printf("\nEnter the file name:");
    Scanf("%s",fn);
    Printf("\nEnter the pattern to be searched\n");
    Scanf("%s",pat);
    fp=fopen(fn,"r");
    while(!feof(fp))
    {
        fgets(temp,1000,fp);
        if(strstr(temp,pat))
            printf("%s",temp);
    }
    fclose(fp);
}
```

b)Program to simulate CP command

Algorithm:

1. Read the source file.
2. Open the source file.
3. If the file is empty ,display an error,
4. Otherwise read a new file name.
5. Copy the contents in the source file into new file.
6. Close the two files.

Program:

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

```

void main(){
    FILE *fp1,*fp2;
    char s,st[100],nf[100];
    Printf("\nEnter the source file name:");
    Scanf("%s",s);
    Printf("\nEnter the new file name:");
    Scanf("%s",nf);
    fp1=fopen(s,"r");
    if(fp1!=NULL)
    {
        fp2=fopen(nf,"w");
        while((s=fgetc(fp1))!=EOF)
            fputc(s,fp2);
        printf("\nFile copied");
    }
    else
        printf("\nError");
    fclose(fp1);
    fclose(fp2);
}

```


Ex:no: 2

SHELL PROGRAMMING

Date:

AIM : To write the shell programs for the following

- a) Find the greatest numbers
- b) Sum of 'n' numbers
- c) Swapping of 2 numbers
- d) Checking whether the number is positive or negative
- e) Greatest of 3 numbers
- f) Factorial of a number
- h) Check whether a given number is prime or not.

1. FIND THE GREATEST NUMBERS

PROGRAM:

```
echo "Enter 2 numbers"
read a b
if [ $a -gt $b ]
then
echo "$a is greater"
else
echo "$b is greater"
fi
```

Output:

```
Enter 2 numbers
5 9
9 is greater
```

2. SUM OF N NUMBERS:

PROGRAM:

```
echo "Enter limit"
read n
i=1
sum=0
while [ $i -le $n ]
do
let sum=$sum+$i
let i=$i+1
done
echo "The sum of $n numbers is $sum"
```

Output:

```
Enter limit
5
The sum of 5 numbers is 15
```

3. SWAPPING OF TWO NUMBERS:

PROGRAM:

```
echo "Enter two numbers"
read a b
t=$a
```

```
a=$b
b=$t
echo "a=$a b=$b"
```

Output:

Enter two numbers

2 4

a=4 b=2

4. CHECKING THE NUMBER IS POSITIVE OR NEGATIVE:

PROGRAM:

```
echo "Enter a number"
```

```
read a
```

```
if [ $a -ge 0 ]
```

```
then
```

```
echo "$a is positive"
```

```
else
```

```
echo "$a is negative"
```

```
fi
```

Output:

Enter a number

-8

-8 is negative

Enter a number

4

4 is positive

5. GREATEST OF THREE NUMBERS:

PROGRAM:

```
echo "Enter three numbers"
```

```
read a b c
```

```
if [ $a -gt $b -a $a -gt $c ]
```

```
then
```

```
echo "$a is greatest"
```

```
elif [ $b -gt $a -a $b -gt $c ]
```

```
then
```

```
echo "$b is greatest"
```

```
else
```

```
echo "$c is greatest"
```

```
fi
```

Output:

Enter three numbers

9 1 6

9 is greatest

6. FACTORIAL OF A NUMBER:

PROGRAM:

```
echo "Enter the limit"
```

```
read n
```

```
i=1
```

```
fact=1
```

```
while [ $i -le $n ]
do
let fact=$fact*$i
let i=$i+1
done
echo "Factorial of $n is $fact"
```

Output:

Enter the limit

5

Factorial of 5 is 120

7. CHECK WHETHER A NUMBER IS PRIME OR NOT:

PROGRAM:

```
echo "Enter the number"
read n
x=0
for (( i=2; i<n; i++ ))
do
let s=$n%i
if [ $s -eq 0 ]
then
x=1
break
fi
done
if [ $x -eq 1 ]
then
echo "$n is not a prime number"
else
echo "$n is a prime number"
fi
```

Output:

Enter the number

97

97 is a prime number

Enter the number

66

66 is not a prime number

CONCLUSION:

The above shell programs were written & executed successfully.

Ex.No: 3 **IMPLEMENT THE CPU SCHEDULING ALGORITHMS**

Date: a) FCFS b) SJFS c) Priority d) Round Robin

AIM:

To implement FCFS, SJF, Priority and Round robin scheduling algorithms using C.

PROBLEM DESCRIPTION:

FIRST COME FIRST SERVE SCHEDULING:

With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. The FCFS scheduling algorithm is nonpreemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU either by terminating or by requesting I/O. While considering the performance of the FCFS scheduling algorithm, the higher burst time process makes the lower burst time process to wait for a long time. This effect is known as convoy effect.

SHORTEST JOB FIRST SCHEDULING:

This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. The SJF algorithm may be either preemptive or nonpreemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

PRIORITY SCHEDULING:

A priority is associated with each process and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS order. An SJF algorithm is simply a priority algorithm where the priority(p) is the inverse of the next CPU burst. The larger the CPU burst, the lower the priority and vice versa.

ROUND-ROBIN SCHEDULING:

The round-robin scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as circular queue.

1.FCFS:

ALGORITHM:

- 1:** Get the number of processes and burst time.
- 2:** The process is executed in the order given by the user.
- 3:** Calculate the waiting time and turn around time.
- 4:** Display the gantt chart, avg waiting time and turn around time.

PROGRAM:

```
#include<stdio.h>
void main(int argc,char *argv[])
{
int i,j=0,n,burst[10],wait[10],turn[10];
float w=0,t=0;
printf("Enter the no. of processes");
scanf("%d",&n);
burst[0]=0;
printf("Enter the burst time");
for(i=1;i<=n;i++)
{
scanf("%d",&burst[i]);
}
printf("\n\nGantt chart\n");
printf("\n_____\\n");
for(i=1;i<=n;i++)
printf("\tP%d\\t",i);
printf("\n_____\\n");
for(i=0;i<=n;i++)
{
j=j+burst[i];
wait[i+1]=j;
turn[i]=j;
printf("%d\\t\\t",j);
}
for(i=1;i<=n;i++)
w=w+wait[i];
for(i=0;i<=n;i++)
t=t+turn[i];
w=w/n;
t=t/n;
printf("\nAverage waiting time %0.2f",w);
printf("\nAverage turnaroundtime %0.2f",t);
}
```

SAMPLE INPUT AND OUTPUT:

cc fcfs.c

./a.out

Enter the no. of processes 3

Enter the burst time 3 6 8

Gantt chart

P1	P2	P3
0	3	9
		17

Average waiting time 4.00

Average turn around time 9.67

2.SJF:**ALGORITHM:**

- 1: Get the number of processes and burst time.
- 2: Sort the process based on the burst time in ascending order.
- 3: Calculate the waiting time and turn around time.
- 4: Display the gantt chart,avg waiting time and turn around time.

PROGRAM:

```
#include<stdio.h>
void main(int argc,char *argv[])
{
int b[10],temp,i,j,n,wait[10],burst[10],turn[10];
float w=0,t=0;
printf("Enter the no. of processes");
scanf("%d",&n);
burst[0]=0;
b[0]=0;
printf("Enter the burst time");
for(i=1;i<=n;i++)
{
scanf("%d",&burst[i]);
}
for(i=1;i<=n;i++)
b[i]=burst[i];
for(i=1;i<n;i++)
for(j=i+1;j<=n;j++)
if(b[i]>b[j])
{
temp=b[i];
b[i]=b[j];
b[j]=temp;
}
printf("\nGantt chart");
printf("\n_____\\n");
```

```

for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(b[i]==b[j])
printf("P%d\t",j);
printf("\n_____ \n");
j=0;
for(i=0;i<=n;i++)
{
j=j+b[i];
wait[i+1]=j;
turn[i]=j;
printf("%d\t",j);
}
for(i=1;i<=n;i++)
w=w+wait[i];
for(i=0;i<=n;i++)
t=t+turn[i];
w=w/n;
t=t/n;
printf("\nAverage waiting time is %0.2f",w);
printf("\nAverage turnaroundtime is %0.2f",t);
}

```

SAMPLE INPUT AND OUTPUT:

```

cc sjf.c
./a.out
Enter the no. of processes 3
Enter the burst time 2 1 3
Gantt chart

```

```

P1|  P2|  P3|

```

```

0    1    3    6

```

Average waiting time is 1.33

Average turn around time is 3.33

3) PRIORITY SCHEDULING

ALGORITHM:

- 1:** Get the number of processes, priority and burst time.
- 2:** Sort the process based on the priority in ascending order
- 3:** Calculate the waiting time and turn around time.
- 4:** Display the gantt chart, avg waiting time and turn around time.

PROGRAM:

```

#include<stdio.h>
struct prcs
{

```

```

int pid;
int bt;
int pr;
int tt;
int wt;
};
int main()
{
    int no;
    int k=0,b=0,g=0;
    int i;
    int w=0;
    float n,m;
    struct prcs p[10];
    printf("enter the number of process");
    scanf("%d",&no);
    for(i=1;i<=no;i++)
    {
        printf("enter the process id");
        scanf("%d",&p[i].pid);
        printf(" enter the burst time");
        scanf("%d",&p[i].bt);
        printf("enter the priority");
        scanf("%d",&p[i].pr);
        p[i].wt=0;
        p[i].tt=0;
    }
    for(i=1;i<=no;i++)
    {
        int f=1,j;
        for(j=1;j<=no-i;j++)
        {
            int h=0,l=0,m=0;
            if(p[j].pr>p[j+1].pr && i<no)
            {
                f=0;
                l=p[j].pr;
                p[j].pr=p[j+1].pr;
                p[j+1].pr=l;
                h=p[j].pid;
                p[j].pid=p[j+1].pid;
                p[j+1].pid=h;
                g=p[j].bt;
                p[j].bt=p[j+1].bt;
                p[j+1].bt=g;
            }
            if(f)
                break;
        }
    }
}

```



```

    }
    }
    }
    p[1].wt=0;
    p[1].tt=p[1].bt;
    for(i=2;i<=no;i++)
    {
        p[i].tt=p[i-1].tt+p[i].bt;
        p[i].wt=p[i-1].tt;
        b=p[i].wt+b;
        k=p[i].tt+k;
    }
    k=k+p[1].tt;
    m=(float)b/(float)no;
    n=(float)k/(float)no;
    for (i=1;i<=no;i++)
    {
        printf("the waiting time for p[%d]==%d\n",p[i].pid,p[i].wt);
        printf("the turnarround time for p[%d]==%d\n",p[i].pid,p[i].tt);
    }
    printf("the average waiting time==%f\n",m);
    printf("the average turnarround time==%f\n",n);
    printf("_____ \n");
    for (i=1;i<=no;i++)
    {
        printf("Process%d",p[i].pid);
    }
    printf("\n");
    printf("_____ \n");
    for(i=1;i<=no;i++)
    {
        printf("%d\t",p[i].wt);
    }
    printf("%d\n",p[no].tt); return(0);
}

```

SAMPLE INPUT AND OUTPUT:

```

cc pri.c
./a.out
enter the number of process3
enter the process id1
enter the burst time2
enter the priority3
enter the process id2
enter the burst time2
enter the priority1
enter the process id3
enter the burst time4

```

enter the priority2
 the waiting time for p[2]==0
 the turnarround time for p[2]==2
 the waiting time for p[3]==2
 the turnarround time for p[3]==6
 the waiting time for p[1]==6
 the turnarround time for p[1]==8
 the average waiting time==2.666667
 the average turnarround time==5.333333

Process2Process3Process1

0 2 6 8

4. RR Scheduling

ALGORITHM:

- 1: Initialize all the structure elements
- 2: Receive inputs from the user to fill process id, burst time and arrival time.
- 3: Calculate the waiting time for all the process id.
 - i) The waiting time for first instance of a process is calculated as:
 $a[i].waittime = count + a[i].arrivt$
 - ii) The waiting time for the rest of the instances of the process is calculated as:
 - a) If the time quantum is greater than the remaining burst time then waiting time is calculated as:
 $a[i].waittime = count + tq$
 - b) Else if the time quantum is greater than the remaining burst time then waiting time is calculated as:
 $a[i].waittime = count - remaining\ burst\ time$
- 4: Calculate the average waiting time and average turnaround time
- 5: Print the results of the step 4.

PROGRAM:

```

#include<stdio.h>
void main()
{
  int b[10],i,j=1,n,temp,burst[10],wait[10],turn[10],p[10],a=1,q,tat[10],t1=0;

  float t=0,w=0;
  printf("Enter the no of process & Q");
  scanf("%d%d",&n,&q);
  burst[0]=0;
  b[0]=0;
  tat[0]=0;
  p[0]=0;
  printf("Enter burst time");
  for(i=1;i<=n;i++)
    scanf("%d",&burst[i]);

```

```

for(i=1;i<=n;i++)
b[i]=burst[i];
printf("\n\n\t\t Gantt chart\n");
printf("-----\n");
for(i=1;i<=n;i++)
{
if(b[i]>0)
{
a=1;
printf("P%d\t",i);
if(b[i]>=q)
{
t1=t1+q;
p[j]=t1;
j++;
}
else if(b[i]<q)
{
t1=t1+b[i];
p[j]=t1;
j++;
}
b[i]=b[i]-q;
if(b[i]<=0)
tat[i]=t1;
}
else
a++;
if(a==n+1)
break;
if(i==n)
i=0;
}
printf("\n-----\n");
for(i=0;i<j;i++)
printf("%d\t",p[i]);
for(i=1;i<=n;i++)
{
t=t+tat[i];
w=w+tat[i]-burst[i];
}
w=w/n;
t=t/n;
printf("\nThe average waiting time is %0.2f",w);
printf("\nThe average turn around time is %0.2f",t);
}

```

SAMPLE INPUT AND OUTPUT:

cc rr.c

./a.out

Enter the number of processes 3

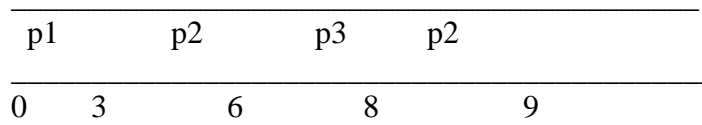
Enter the time quantum 3

Enter the Burst Time P1 3

Enter the Burst Time P2 4

Enter the Burst Time P3 2

GanttChart



The average Waiting Time 3.666667

The average Turnaround Time 6.6666

CONCLUSION:

Thus FCFS, SJFS, Priority and Round Robin scheduling algorithms is implemented using C and executed.

Ex.No:4(i) INTER PROCESS COMMUNICATION USING SHARED MEMORY

AIM:

To write a c program for inter process communication using shared memory.

ALGORITHM:

1. Create a child process using `fork()` command.
2. Create shared memory for parent process `shmget()` system call.
3. The current process writes the content in shared memory using `shmptr` pointer.
4. Attach the same shared memory to the child process.
5. The data in shared memory is read by the child process using `shmptr()` pointer.
6. Detach and rebase the shared memory.

PROGRAM:

```
#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>
int main()
{
int child,shmidx,i;
char *shmpt;
child=fork();
if(!child)
{
shmidx=shmget(2041,32,0666|IPC_CREAT);
shmpt=shmat(shmidx,0,0);
printf("\n PARENT PROCESS WRITING \n");
for(i=0;i<10;i++)
```

```

{
shmptr[i]='a'+i;
putchar(shmptr[i]);
}
printf("\n\n %s",shmptr);
wait(NULL);
}
else
{
shmid=shmget(2041,32,0666);
shmptr=shmat(shmid,0,0);
printf("\n CHILD PROCESS READING\n");
for(i=0;i<10;i++)
putchar(shmptr[i]);
shmdt(NULL);
shmctl(shmid,IPC_RMID,NULL);
}
return 0;
}

```

SAMPLE INPUT AND OUTPUT:

```

[root@local host~]# ./a.out
parent process writing
abcdefghij
child process reading
abcdefghij

```

CONCLUSION:

The above program for inter process communication using shared memory was executed successfully and output was verified.

Ex.No:4(ii)

INTER PROCESS COMMUNICATION USING PIPES

Date:

AIM:

To write a c program for implementation of inter process communication using pipes.

ALGORITHM:

1. Create the child process using fork() command
2. Create the pipe structure using pipe() command.
3. Close the read end of the pipe using close() command.
4. Write the data in the pipe using write() command.
5. Close the writer end of the child process using close() command.

6. Display the string in the pipe using read system call.
7. Stop the process.

PROGRAM:

```
#include<stdio.h>
int main()
{
int fd[2],child;
char a[10];
printf("\n enter the string to store into the pipe");
scanf("%s",a);
pipe(fd);
child=fork();
if(!child)
{
close(fd[0]);
write(fd[1],a,5);
wait(0);
}
else
{
close(fd[1]);
read(fd[0],a,5);
printf("\n the string reterived from the pipe is%s\n",a);
}
return 0;
}
```

SAMPLE INPUT AND OUTPUT:

```
[root@local host~]# ./a.out
Enter The String To Store Into The Pipe
Temple
The String Reterived From The Pipe Is: Temple
```

CONCLUSION:

The above program for inter process communication using pipes was executed successfully and output was verified.

Ex.No: 5 **IMPLEMENT AN ALGORITHM FOR DEAD LOCK**
Date: **DETECTION**

AIM:

To write a C program to implement an Algorithm to detect Deadlock.

PROBLEM DESCRIPTION:

Deadlock detection is the process of actually determining that a deadlock exists and identifying the process and resources involved in the deadlock. The basic idea is to check the allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state,

The omplexity of algorithm is $O(N^2)$ where N is the number of proceeds. Another potential problem is starvation; same process killed repeatedly.

ALGORITHM:

- 1:** Get the no of processes.
- 2:** Get the process numbers.
- 3:** Get the no of resources types and instances of it.
- 4:** Get Max demand of each process of $n \times m$ matrices.
- 5:** Get the $n \times m$ matrices the number of resources of each type currently allocated to each process.
- 6:** Calculate the $n \times m$ of the remaining resource need of each process.
- 7:** Initialize work as available resource and array of finish to false.
- 8:** Check the Needed resource is lesser than the available resource if not display the System not in safe state and if it is lesser than system in safe state.

9: Initialize work as sum of work and allocation, check if array of finish is true go to step 7 again if not go to step 8.

10: Check that request can be immediately granted.

11: If single request is lesser than or equal to available if true means arrive to new state.

12: Print the sequence if it is in safe state or print not in safe state.

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][j];
                        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");
    }
}

```

```

}
else
{
    printf("\nNo Deadlock Occur");
}
}

```

SAMPLE INPUT AND OUTPUT:

```

***** Deadlock Detection Algo *****
Enter the no of Processes      3
Enter the no of resource instances  3
Enter the Max Matrix
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix
3 3 3
2 0 3
1 2 4
Enter the available Resources
1 2 0
Process  Allocation      Max      Available
P1      3 3 3            3 6 8      1 2 0
P2      2 0 3            4 3 3
P3      1 2 4            3 4 4

System is in Deadlock and the Deadlock process are
P0      P1      P2

```

CONCLUSION:

Thus an algorithm is implemented using C to detect deadlock and executed.

Ex.No: 6 **IMPLEMENT BANKERS ALGORITHM FOR DEAD LOCK**
Date: **AVOIDANCE**

AIM:

To write a C program to implement Bankers Algorithm to avoid Deadlock.

PROBLEM DESCRIPTION:

There are possibilities of side effects of preventing deadlocks and low device utilization reduced system throughput. When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system.

When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If yes, the resources are allocated, otherwise the process must wait until some other process releases enough resources.

ALGORITHM:

- 1:** Get the no of processes.
- 2:** Get the process numbers.
- 3:** Get the no of resources types and instances of it.
- 4:** Get Max demand of each process of $n \times m$ matrices.
- 5:** Get the $n \times m$ matrices the number of resources of each type currently allocated to each process.
- 6:** Calculate the $n \times m$ of the remaining resource need of each process.
- 7:** Initialize work as available resource and array of finish to false.
- 8:** Check the Needed resource is lesser than the available resource if not display the System not in safe state and if it is lesser than system in safe state.

9: Initialize work as sum of work and allocation, check if array of finish is true go to step 7 again if not go to step 8.

10: Check that request can be immediately granted.

11: If single request is lesser than or equal to available if true means arrive to new state.

12: Print the sequence if it is in safe state or print not in safe state.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int p,r,m,z,k,buf,a[10],avail[10],max[10][10],alloc[10][10],need[10][10],i,j,top;
void get()
{
printf("\nENTER THE NUMBER OF PROCESSES:");
scanf("%d",&p);
printf("\nENTER THE NUMBER OF RESOURCE TYPES:");
scanf("%d",&r);
for(j=1;j<=r;j++)
{
printf("\nENTER THE NUMBER OF RESOURCES FOR TYPE %d : ",j);
scanf("%d",&avail[j]);
}
for(i=1;i<=p;i++)
{
printf("\nENTER THE MAXIMUM NUMBER OF RESOURCES REQUIRED FOR
PROCESS %d:\n",i);
for(j=1;j<=r;j++)
{
printf("\nFor Resource type %d :",j);
scanf("%d",&max[i][j]);
}
}
printf("\nENTER THE ALLOCATED INSTANCES:\n");
for(i=1;i<=p;i++)
{
printf("\nPROCESS %d:",i);
printf("\n-----");
for(j=1;j<=r;j++)
{
printf("\nResource Type - %d :",j);
scanf("%d",&m);
if(m<=avail[j])
{
alloc[i][j]=m;
avail[j]=avail[j]-m;
}
}
else
```

```

printf("\nALLOCATION EXCEEDS MAXIMUM. U CAN'T ALLOCATE IT.");
}
}
}
void disp1()
{
printf("\nNEEDED RESOURCES:");
printf("\n-----");
for(i=1;i<=p;i++)
{
printf("\nProcess %d:\t",i);
for(j=1;j<=r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
printf(" %d",need[i][j]);
}
} //for i
} //function
void seqnc()
{
top=1;
while(top<=p)
{
for(i=1;i<=p;i++)
{
buf=0;
z=0;
for(j=1;j<=r;j++)
{
z=z+need[i][j];
if(need[i][j]<=avail[j])
buf++;
}
if(buf==r&&z!=0)
{
a[top]=i;
top++;
for(k=1;k<=r;k++)
{
avail[k]=avail[k]+alloc[i][k];
need[i][k]=0;
} //for
} //if(buf==r)
} //for i
} //while
} //function
void disp2()

```

```

{
printf("\nThe Sequence of allocation to the processes:");
for(i=1;i<=p;i++)
printf(" %d",a[i]);
}
void main()
{
clrscr();
get();
disp1();
seqnc();
disp2();
getch();
}

```

SAMPLE INPUT AND OUTPUT:

```

Enter the number of resources for type 1:10
Enter the number of resources for type 2:5
Enter the number of resources for type 3:7
Enter the maximum number of resources required for a process 1:
for resource type 1:7
for resource type 2:5
for resource type 3:3
Enter the maximum number of resources required for a process 2:
for resource type 1:3
for resource type 2:2
for resource type 3:2
Enter the maximum number of resources required for a process 3:
for resource type 1:9
for resource type 2:0
for resource type 3:2
Enter the maximum number of resources required for a process 4:
for resource type 1:2
for resource type 2:2
for resource type 3:2
Enter the maximum number of resources required for a process 5:
for resource type 1:4
for resource type 2:3
for resource type 3:3
Enter the allocated instances:
process1
_____
Resource type 1:0
Resource type 2:1
Resource type 3:0
process2
_____

```


Resource type 1:2
Resource type 2:0
Resource type 3:0
process3

Resource type 1:3
Resource type 2:0
Resource type 3:2
process4

Resource type 1:2
Resource type 2:1
Resource type 3:1
process5

Resource type 1:0
Resource type 2:0
Resource type 3:2

Needed resources

process 1:	743
process 2:	122
process 3:	600
process 4:	011
process 5:	431

The sequence of allocation of process24513

CONCLUSION:

The Banker's Algorithm is implemented using C to avoid Deadlock.

Ex.No: 7 **IMPLEMENT ALL FILE ALLOCATION STRATEGIES**

Date: **a) Sequential b) Indexed c) Linked**

AIM :

To implement the various file allocation techniques.

PROBLEM DESCRIPTION:

a) Sequential file allocation strategy: In this type of strategy, the files are allocated in a sequential manner such that there is a continuity among the various parts or fragments of the file.
b) Indexed file allocation strategy: In this type of strategy, the files are allocated based on the indexes that are created for each fragment of the file such that each and every similar indexed file is maintained by the primary index thereby providing flow to the file fragments.
c) Linked file allocation strategy: In this type of strategy, the files are allocated in a linked list format where each and every fragment is linked to the other file through either addresses or pointers. Thus, the starting location of the file serves for the purpose of extraction of the entire file because every fragment is linked to each other.

a) Sequential File Allocation

Algorithm:

1. Declare the starting block no. and the length of the file.
2. Get the Starting block no. and length of the file from the user.
3. Allocate files sequentially until end of the file.
4. Display the fragments of the file.

Program:

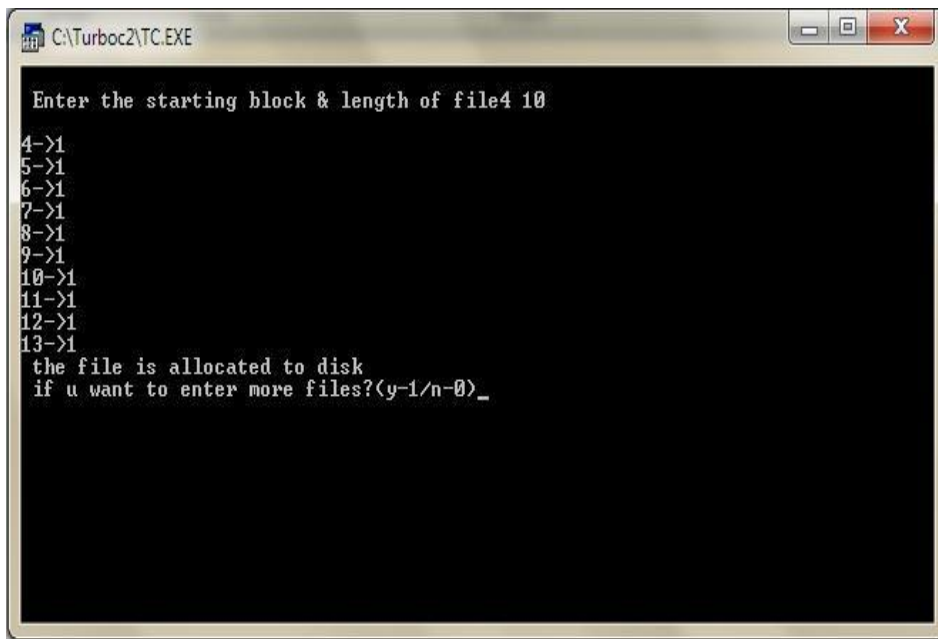
```
#include<stdio.h>
```

```

#include<conio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch();
}

```

SAMPLE INPUT AND OUTPUT:



```
C:\Turboc2\TC.EXE

Enter the starting block & length of file4 10

4->1
5->1
6->1
7->1
8->1
9->1
10->1
11->1
12->1
13->1
the file is allocated to disk
if u want to enter more files?(y-1/n-0)_
```

b) Linked File Allocation

Algorithm:

1. Initialize the AVAIL linked list, where each node consist of starting address, size of the empty block and a link for next available node
2. Initialize the FAT (File Allocation Table) which is implemented as array of pointers.
3. Display the AVAIL List
4. Read File allocation request which consist of File name, No of blocks and its contents
5. Traverse the AVAIL linked list from the starting node
6. Retrieve the required no of blocks from AVAIL List
7. Assign the contents of file to the retrieved blocks
8. Update the FAT by making an entry in FAT
9. Update the AVAIL LIST
10. Display the AVAIL List and FAT table

Program:

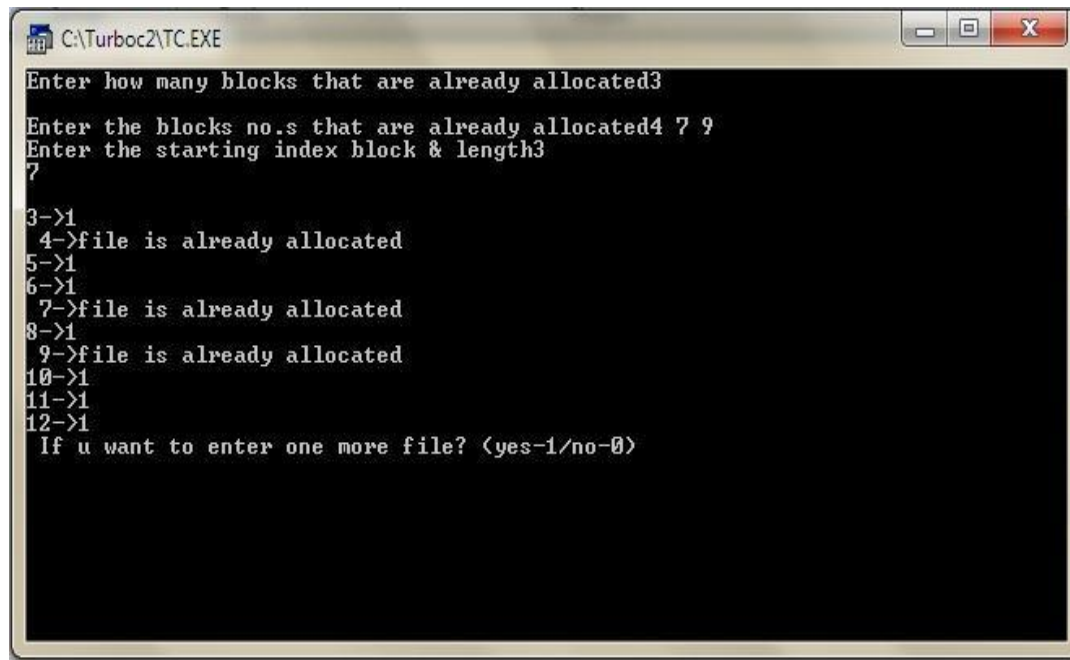
```
#include<stdio.h>
#include<conio.h>
main()
{
int f[50],p,i,j,k,a,st,len,n,c;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks that are already allocated");
scanf("%d",&p);
printf("\nEnter the blocks no.s that are already allocated");
```

```

for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
X:
printf("Enter the starting index block & length");
scanf("%d%d",&st,&len);
k=len;
for(j=st;j<(k+st);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n %d->file is already allocated",j);
k++;
}
}
printf("\n If u want to enter one more file? (yes-1/no-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch( );
}

```

SAMPLE INPUT AND OUTPUT:



```
C:\Turboc2\TC.EXE
Enter how many blocks that are already allocated3
Enter the blocks no.s that are already allocated4 7 9
Enter the starting index block & length3
7
3->1
4->file is already allocated
5->1
6->1
7->file is already allocated
8->1
9->file is already allocated
10->1
11->1
12->1
If u want to enter one more file? <yes-1/no-0>
```

c) Indexed File Allocation

Algorithm:

1. Declare the index block no. and total no.of files in a block .
2. Get the index block no. and total no.of files in a block from the user.
3. Allocate files based on the index block no.
4. Arrange the files based on indexes which are created for each fragment of the file such that each and every similar indexed file is maintained by the primary index to provide the flow to file fragments

Program:

```
#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:
printf("enter index block\t");
scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;

printf("enter no of files on index\t");
```

```

scanf("%d",&n);
}
else
{
printf("Block already allocated\n");
goto x;
}
for(i=0;i<n;i++)
scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("Block already allocated");
goto x;
}
for(j=0;j<n;j++)
f[inde[j]]=1;
printf("\n allocated");
printf("\n file indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf(" Enter 1 to enter more files and 0 to exit\t");
scanf("%d",&c);
if(c==1)
goto x;
else
exit();
getch();
}

```

SAMPLE INPUT AND OUTPUT:



A screenshot of a Turbo C++ IDE window titled "C:\Turboc2\TC.EXE". The window contains a text-based program for file allocation. The program prompts the user to enter an index block (9) and the number of files on the index (3). It then displays "allocated" and "file indexed". The user is prompted to enter file indices: "9->1:1", "9->2:1", and "9->3:1". The program then displays a message: "Enter 1 to enter more files and 0 to exit".

```
enter index block      9
enter no of files on index  3
1 2 3

allocated
file indexed
9->1:1
9->2:1
9->3:1 Enter 1 to enter more files and 0 to exit  _
```

CONCLUSION:

The file allocation strategies are implemented using C and executed.

AIM:

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

PROBLEM DESCRIPTION:

The producer-consumer problem (also known as the bounded-buffer problem) is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e. removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. The solution for the producer is to go to sleep if the buffer is full. The next time the consumer removes an item from the buffer, it wakes up the producer who starts to fill the buffer again. In the same way the consumer goes to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

ALGORITHM:

- 1:** The Semaphore mutex, full & empty are initialized.
- 2:** In the case of producer process
 - a) Produce an item in to temporary variable.
 - b) If there is empty space in the buffer check the mutex value for enter into the critical section.
 - c) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.
- 3:** In the case of consumer process
 - a) It should wait if the buffer is empty
 - b) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
 - c) Signal the mutex value and reduce the empty value by 1. Consume the item.
- 4:** Print the result

PROGRAM:

```
#include<stdio.h>
char buf[20],p[20],cos[20];
int mutex,i,k,c,sz,n;
mutex=0;
void prosig()
{
mutex=mutex+1;
}
```

```

void consig()
{
mutex=mutex-1;
}
int buffer(int mutex)
{
if(mutex==0)
return 1;
else
return 0;
}
void producer(int sz)
{
int c;
c=buffer(mutex);
if(c==1)
{
printf("\nProducer can produce the item and give $ for exit\n");
i=0;
while(i<sz&&(p[i]=getchar())!='$')
{
buf[i]=p[i];
i++;
}
k=i;
prosig();
printf("\nProduction done successfully\n");
}
else if(k<sz)
{
printf("Producer can also produce items");
while((p[k]=getchar())!='$')
{
buf[k]=p[k];
k++;
}
prosig();
printf("\nProduction done successfully\n");
}
else if(k>=sz)
{
printf("\nBuffer is full,can't produce\n");
}
}
void consumer()
{
int c1;

```

```

c1=buffer(mutex);
if(c1==0)
{
printf("\nConsumer can consume item\n");
for(i=0;i<k;i++)
cos[i]=buf[i];
printf("\nConsumed item is:\n");
for(i=0;i<k;i++)
printf("\n%c",cos[i]);
consig();
printf("\nSuccessfully done\n");
}
else
{
printf("\nBuffer is empty,can't consume\n");
}
}
int main()
{
int op,sz;
printf("Enter the buffer size");
scanf("%d",&sz);
do
{
printf("\n1.Producer\t2.Consumer\t3.Exit\n");
printf("\nEnter your choice\n");
scanf("%d",&op);
switch(op)
{
case 1:
producer(sz);
break;
case 2:
consumer();
break;
case 3:
exit(0);
}
}
while(op<=2);
return 0;
}

```

SAMPLE INPUT AND OUTPUT:

```

cc pcp.c
./a.out
Enter the buffer size5

```

```

1.Producer  2.Consumer  3.Exit
Enter your choice
1
Producer can produce the item and give $ for exit
ho$
Production done successfully
1.Producer  2.Consumer  3.Exit
Enter your choice
1
Producer can also produce items ney$
Production done successfully
1.Producer  2.Consumer  3.Exit
Enter your choice
1
Buffer is full,can't produce
1.Producer  2.Consumer  3.Exit
Enter your choice
2
Consumer can consume item
Consumed item is:
h
o
n
e
y
Successfully done
1.Producer  2.Consumer  3.Exit
Enter your choice
3

```

CONCLUSION:

The Producer & consumer Problem using Semaphore is implemented using C and executed.

Ex.No:9 **IMPLEMENTATION OF SINGLE & TWO LEVEL DIRECTORIES**

Date:

AIM:

To implement the single & two level directory.

PROBLEM DESCRIPTION:

a) Single level directory:

In a single-level directory system, all the files are placed in one directory. This is very common on single-user OS's. A single-level directory has significant limitations, however, when the number of files increases or when there is more than one user. Since all files are in the same directory, they must have unique names. If there are two users who call their data file "test", then the unique-name rule is violated. Although file names are generally selected to reflect the content of the file, they are often quite limited in length. Even with a single-user, as the number of files increases; it becomes difficult to remember the names of all the files in order to create only files with unique names.

b) Two level directory:

In the two-level directory system, the system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. There are still problems with two-level directory structure. This structure effectively isolates one user from another. This is an advantage when the users are completely independent, but a disadvantage when the users want to cooperate on some task and access files of other users. Some systems simply do not allow local files to be accessed by other users.

Single level directory:

Algorithm:

Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
clrscr();
printf("enter number of directorios:");
scanf("%d",&master);
```

```

printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("*****\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
getch();
}

```

```

C:\ Turbo C++ IDE
enter number of directories:2
enter names of directories:a
b
enter size of directories:2
3
enter the file names :aa1
aa2
aa3
aa4
aa5

directory      size      filenames
*****
a              2        aa1
              aa2
b              3        aa3
              aa4
              aa5

```

Two level directory

Algorithm:

Program:

```

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

```

```
getch(); }
```

```

C:\ Turbo C++ IDE
enter number of directories:2
enter directory 1 names:a1
enter size of directories:2
enter subdirectory name and size:as1
3
enter file name:n1
enter file name:n2
enter file name:n3
enter subdirectory name and size:as2
2
enter file name:n4
enter file name:n5
enter directory 2 names:b1
enter size of directories:2
enter subdirectory name and size:bs1
2
enter file name:n6
enter file name:n7
enter subdirectory name and size:bs2
3
enter file name:n8
enter file name:n9
enter file name:n10

dirname      size      subdirname    size      files
*****
a1            2        as1           3        n1      n2      n3
              as2           2        n4      n5
b1            2        bs1           2        n6      n7
              bs2           3        n8      n9      n10
  
```

DAG

A **directed acyclic graph (DAG)** is a directed graph with no directed cycles that is formed by a collection of vertices and directed edges with each edge connecting one vertex to another, such that there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again.

Algorithm:

Step 1: Start the Program

Step 2: Obtain the required data through char and int datatypes.

Step 3: Enter the filename, index block. Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

Step 6: DAG is allocated to the unused linked allocation.

Step 7: Stop the execution

Program

```

#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
  
```



```

struct tree_element *link[5];
};
typedef struct tree_element node;
typedef struct
{
char from[20]; char to[20];
}link;
link L[10]; int nofl;
node * root;
void read_links();
void draw_link_lines();
void search(node *root,char *s,int *x,int *y);
void create(node **root,int lev,char *dname,int lx,int rx,int x);
void display(node *root);
void main()
{
int gd=DETECT,gm;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"C:\\TC\\BGI");
display(root);
getch();
closegraph();
}
void read_links()
{
int i;
printf("how many links");
scanf("%d",&nofl);
for(i=0;i<nofl;i++)
{
printf("File/dir:");
fflush(stdin);
gets(L[i].from);
printf("user name:");
fflush(stdin);
gets(L[i].to);
}
}
void draw_link_lines()
{
int i,x1,y1,x2,y2;
for(i=0;i<nofl;i++)
{
search(root,L[i].from,&x1,&y1);

```

```

search(root,L[i].to,&x2,&y2);
setcolor(LIGHTGREEN);
setlinestyle(3,0,1);
line(x1,y1,x2,y2);
setcolor(YELLOW);
setlinestyle(0,0,1);
}
}
void search(node *root,char *s,int *x,int *y)
{
int i;
if(root!=NULL)
{
if(strcmpi(root->name,s)==0)
{
*x=root->x;
*y=root->y;
return;
}
else
{
for(i=0;i<root->nc;i++)
search(root->link[i],s,x,y);
}
}
}
void create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("enter name of dir/file(under %s):",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for dir/ 2 for file:");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("no of sub directories /files (for %s):",(*root)->name);

```

```

scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create( & ( (*root)->link[i] ) , lev+1 , (*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else
(*root)->nc=0;
}
}
/* displays the constructed tree in graphics mode */
void display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root !=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1)
bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name);
for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}
}

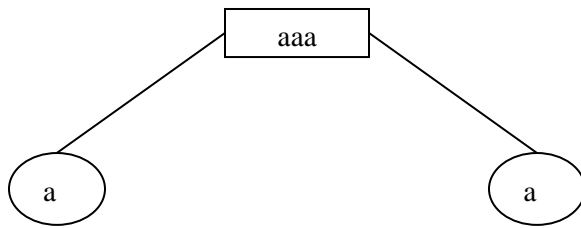
```

OUTPUT

```

enter name of dir/file(under root):aaa
enter 1 for dir/ 2 for file:1
no of sub directories /files (for aaa):2
enter name of dir/file(under aaa):a1
enter 1 for dir/ 2 for file:2
enter name of dir/file(under aaa):a2
enter 1 for dir/ 2 for file:2

```



Hierarchical file organization

A hierarchical **file organization** is a system organized by graded categorization which could contain thousands of files and still allow users to find any individual file quickly

Algorithm:

Step 1: Start the Program

Step 2: Obtain the required data through char and int datatypes.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks.

Step 6: Hierarchical level directory is allocated to the unused linked allocation.

Step 7: Stop the execution

Program

```

#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x, y, ftype, lx, rx, nc, level;
struct tree_element *link[5];
};
typedef struct tree_element node;
void create(node **root,int lev,char *dname,int lx,int rx,int x);
void display(node *root);
void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);
getch();
closegraph();
}
  
```

```

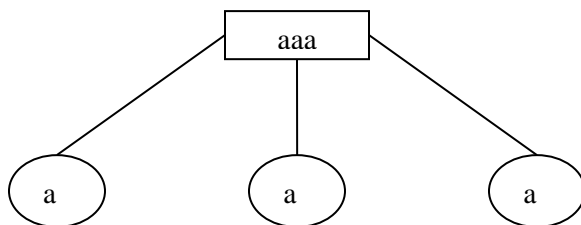
void create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i, gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) : ",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 for file :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)->name);
scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap, lx+gap*i+gap/2);
}
else
(*root)->nc=0;
}
}
void display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root !=NULL)
{
for(i=0;i<root->nc;i++)
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
if(root->ftype==1)
bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else

```

```
fil ellipse(root->x,root->y,20,20);  
outtextxy(root->x,root->y,root->name);  
for(i=0;i<root->nc;i++)  
display(root->link[i]);  
} }
```

SAMPLE INPUT AND OUTPUT

Enter name of dir/file(under root) : aaa
enter 1 for Dir/2 for file :1
No of sub directories/files(for aaa):3
Enter name of dir/file(under aaa) : a1
enter 1 for Dir/2 for file :2
Enter name of dir/file(under aaa) : a2
enter 1 for Dir/2 for file :2
Enter name of dir/file(under aaa) : a3
enter 1 for Dir/2 for file :2



CONCLUSION:

The single & two level directories is implemented & executed successfully.

Ex.No. 10 **IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHMS**
Date: (FIFO, LFU and LRU)

AIM:

To implement the page replacement algorithms.

PROBLEM DESCRIPTION:

When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in.

- FIFO (First in first out): When a page must be replaced, the oldest page is chosen.
- LRU (Least Recently Used): When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- LFU (Least Frequently Used): It refers to the removal of the page that will not be used in the future, for the longest period of time.

FIFO:

Algorithm:

1. Start the algorithm
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. Stop the algorithm.

PROGRAM:

```

#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
for(i=0;i<no;i++)
frame[i]= -1;
j=0;
printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
{
printf("%d\t\t",a[i]);
avail=0;
for(k=0;k<no;k++)
if(frame[k]==a[i])
avail=1;
if (avail==0)
{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf("Page Fault Is %d",count);
return 0;
}

```

SAMPLE INPUT AND OUTPUT:

ENTER THE NUMBER OF PAGES: 20

ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

ENTER THE NUMBER OF FRAMES :3

	ref string	page frames	
7	7	-1	-1
0	7	0	-1
1	7	0	1
2	2	0	1
0			
3	2	3	1

0	2	3	0
4	4	3	0
2	4	2	0
3	4	2	3
0	0	2	3
3			
2			
1	0	1	3
2	0	1	2
0			
1			
7	7	1	2
0	7	0	2
1	7	0	1

Page Fault Is 15

LFU ALGORITHM

Algorithm:

1. Start the algorithm
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. Stop the algorithm

PROGRAM:

```
#include<stdio.h>
int n;
main()
{
int seq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s,pf=0;
int count=1,p=0;
float pfr;
clrscr();
printf("enter max limit of the sequence:");
scanf("%d",&max);
printf("enter the sequence:");
for(i=0;i<max;i++)
scanf("%d",&seq[i]);
printf("enter the no of frames:");
scanf("%d",&n);
fr[0]=seq[0];
```

```

pf++;
printf("%d\t",fr[0]);
i=1;
while(count<n)
{
flag=1;
p++;
for(j=0;j<i;j++)
{
if(seq[i]==seq[j])
flag=0;
}
if(flag!=0)
{
fr[count]=seq[i];
printf("%d\t",fr[count] );
count++;
pf++;
}
i++;
}
printf("\n");
for(i=p;i<max;i++)
{
flag=1;
for(j=0;j<n;j++)
{
if(seq[i]==fr[j])
flag=0;
}
if(flag!=0)
{
for(j=0;j<n;j++)
{
m=fr[j];
for(k=i;k<max;k++)
{
if(seq[k]==m)
{
pos[j]=k;
break;
}
}
else
pos[j]=-1;
}
}
for(k=0;k<n;k++)

```

```

{
if(pos[k]==-1)
flag=0;
}
if(flag!=0)
s=findmax(pos);
if(flag==0)
{
for(k=0;k<n;k++)
{
if(pos[k]==-1)
{
s=k;
break;
}
}
}
pf++;
fr[s]=seq[i];
for(k=0;k<n;k++)
printf("%d\t",fr[k]);
printf("\n");
}
}
pfr=(float)pf/(float)max;
printf("\n theno of page faults are:%d",pf);
printf("\n page fault rate:%f",pfr);
getch();
}
int findmax(int a[])
{
int max,i,k=0;
max=a[0];
for(i=0;i<n;i++)
{
if(max<a[i])
{
max=a[i];
k=i;
}
}
return k;
}

```

SAMPLE INPUT AND OUTPUT:

enter max limit of the sequence:20
enter the sequence:

1
2
3
4
2
5
3
4
2
6
7
8
7
9
7
8
2
5
4
9

enter the no of frames:3

1	2	3
4	2	3
4	5	3
4	5	2
6	5	2
7	5	2
7	8	2
7	8	9
2	8	9
5	8	9
4	8	9

The no of page faults are:13

page fault rate:0.650000

LRU ALGORITHM

Algorithm:

1. Start the algorithm
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
7. Display the number of page faults.
8. Stop the algorithm

PROGRAM:

```
#include<stdio.h>
int f[30],fs;
int cnt[30];
int flag,ps[30];
main()
{
int p,i,pos=0,j,k,max,s,pf=0;
float pfr;
void increment();
int check();
clrscr();
printf("enter no of pages:");
scanf("%d",&p);
printf("enter the pages:");
for(i=0;i<p;i++)
scanf("%d",&ps[i]);
printf("enter the frame size:");
scanf("%d",&fs);
for(k=0;k<fs;k++)
f[k]=-999;
for(i=0,j=0;i<p;i++)
{
flag=0;
if(f[i]==-999)
{
check(j,ps[j]);
if(flag==0)
{
f[j]=ps[i];
pf++;
for(k=0;k<=j;k++)
cnt[k]+=1;
j++;
}
}
else
{
check(fs,ps[i]);
if(flag==0)
{
max=cnt[0],pos=0;
for(k=1;k<fs;k++)
if(max<cnt[k])
max=cnt[k],pos=k;
f[pos]=ps[i],pf++;
}
```

```

increment(ps[i]);
}
}
for(k=0;k<fs;k++)
printf("%4d",f[k]);
printf("\n");
}
pfr=(float)pf/p;
printf("total no of page faults:%d\n",pf);
printf("page fault rare:%f",pfr);
getch();
}
void increment(int ele)
{
int k;
for(k=0;k<fs;k++)
{
if(f[k]==ele)
cnt[k]=1;
else
cnt[k]+=1;
}
}
check(int pos,int ele)
{
int k,s;
for(k=0;k<fs;k++)
if(f[k]==ele)
{
cnt[k]=1;
for(s=0;s<pos;s++)
if(s!=k)
cnt[s]+=1;
flag=1;
}
}

```

SAMPLE INPUT AND OUTPUT:

enter no of pages:20

enter the pages:

1
2
3
4
2
5
3
4

2
6
7
8
7
9
7
8
2
5
4
9

enter the frame size:3

1-999-999

1 2-999

1 2 3

4 2 3

4 2 3

4 2 5

3 2 5

3 4 5

3 4 2

6 4 2

6 7 2

6 7 8

6 7 8

9 7 8

9 7 8

9 7 8

2 7 8

2 5 8

2 5 4

9 5 4

total no of page faults:16

page fault rare:0.800000

CONCLUSION:

The page replacement algorithms were implemented & executed successfully.

Ex.No:11

IMPLEMENT PAGING TECHNIQUE OF MEMORY MANAGEMENT

Date:

AIM:

To implement the concept of paging technique of memory management.

ALGORITHM:

- 1: Read the base address, page size, number of pages and memory unit.
- 2: If the memory limit is less than the base address display the memory limit is less than limit.
- 3: Create the page table with the number of pages and page address.
- 4: Read the page number and displacement value.
- 5: If the page number and displacement value is valid, add the displacement value with the address corresponding to the page number and display the result.
- 6: Display the page is not found or displacement should be less than page size.

PROGRAM:

```
#include<stdio.h>
int i,j,k,ps,np1,op,np,np2,fn,f=0;
char p1[50][50],p2[50][50];
int pgtb(int r,int fn,int np)
{
    for(i=0;i<np;i++)
        printf("\n\t%d\t\t%d",i,r++);
    if(i>np)
        printf("\n\t%d\t\tPage Fault",i);
    }return(r);
}

void frame(int np1,int np2,int fn,int ps,char p1[50][50],char p2[50][50])
{
    for(i=0;i<np1;i++)
    {
        if(i<fn)
        {
            printf("\n-----\n");
            printf("\nFrame No:%d\n",i);
            for(j=0;j<ps;j++)
```



```

printf("\t%c",p1[i][j]);
}}
k=np1;
for(i=0;i<np2;i++)
{
if(k<fn)
{
printf("\n-----\n");
printf("\nFrame No:%d\n",k);
k++;
for(j=0;j<ps;j++)
printf("\t%c",p2[i][j]);
}}}
int main()
{
printf("\nENTER THE PAGE SIZE:");
scanf("%d",&ps);
printf("\nENTER NO OF FRAMES:");
scanf("%d",&fn);
printf("\nENTER NO OF PAGES FOR PROCESS1:");
scanf("%d",&np1);
printf("\nENTER NO OF PAGES FOR PROCESS2:");
scanf("%d",&np2);
if(np1+np2>fn)
printf("\nPage Fault will occur\n");
printf("\nPROCESS1");
printf("\n-----\n");
p1[np1][ps];
p2[np2][ps];
for(i=0;i<np1;i++)
{
printf("Enter CHAR for PAGE%d:",i);
scanf("%s",&p1[i]);
}
printf("\nPROCESS2");
printf("\n-----\n");
for(i=0;i<np2;i++)
{
printf("Enter CHAR for PAGE%d:",i);
scanf("%s",&p2[i]);
}
while(1)
{
printf("\n1.Page Table for PROCESS1\n2.Page Table for PROCESS2\n3.Frame Allotmen
t\n4.Free Frame List\n5.Exit\n");
printf("ENTER YOUR CHOICE:");
scanf("%d",&op);

```

```

switch(op)
{
case 1:
printf("Page Table for PROCESS1");
printf("\nPAGE INDEX\tFRAME INDEX\n");
f=pgtb(f,fn,np1);
break;
case 2:
printf("Page Table for PROCESS2");
printf("\nPAGE INDEX\tFRAME INDEX\n");
f=pgtb(f,fn,np2);
break;
case 3:
frame(np1,np2,fn,ps,p1,p2);
break;
case 4:
if(np1+np2>fn)
printf("Page Fault has occurred");
else if(np1+np2==fn)
printf("\nNo Free Frames");
else
{
printf("Free Frame List");
printf("\n-----\n");
for(i=np1+np2;i<fn;i++)
printf("%dth frame",i);
}
break;
case 5:
return(0);
break;
}
}
}

```

SAMPLE INPUT AND OUTPUT:

ENTER THE PAGE SIZE:

10

ENTER NO OF FRAMES:

20

ENTER NO OF PAGES FOR PROCESS1:

3

ENTER NO OF PAGES FOR PROCESS2:

4

PROCESS1

Enter CHAR for PAGE 0 -- a

Enter CHAR for PAGE1 --b

Enter CHAR for PAGE 2-- c

PROCESS2

Enter CHAR for PAGE0-- d

Enter CHAR for PAGE1-- e

Enter CHAR for PAGE2-- f

Enter CHAR for PAGE3-- g

1. Page Table for PROCESS1

2. Page Table for PROCESS2

3. Frame Allotment

4. Free Frame List\

5. Exit

ENTER YOUR CHOICE: 1

Page Table for PROCESS1

PAGE INDEX	FRAME INDEX
------------	-------------

0	0
---	---

1	1
---	---

2	2
---	---

1. Page Table for PROCESS1

2. Page Table for PROCESS2

3. Frame Allotment

4. Free Frame List\

5. Exit

ENTER YOUR CHOICE: 2

Page Table for PROCESS2

PAGE INDEX	FRAME INDEX
------------	-------------

0	3
---	---

1	4
---	---

2	5
---	---

3	6
---	---

1. Page Table for PROCESS1

2. Page Table for PROCESS2

3. Frame Allotment

4. Free Frame List

5. Exit

ENTER YOUR CHOICE: 3

Frame No 0

a

Frame No: 1

b

Frame No: 2

c

Frame No: 3

d

Frame No: 4

e

Frame No: 5

f

Frame No: 6

g

- 1. Page Table for PROCESS1
2. Page Table for PROCESS2
3. Frame Allotment
4. Free Frame List
5. Exit

ENTER YOUR
CHOICE: 4

Free Frame List

7th frame 8th frame 9th frame 10th frame 11th frame 12th frame 13th frame 14th frame 15th
frame 16th frame 17th frame 18th frame 19th frame

1. Page Table for PROCESS1
2. Page Table for PROCESS2
3. Frame Allotment
4. Free Frame List
5. Exit

ENTER YOUR
CHOICE: 5

CONCLUSION:

The concept of paging is implemented and executed.

EX NO:12 IMPLEMENTATION OF THREADING AND SYNCHRONIZATION

AIM

To write a C program to implement threading and synchronization

ALGORITHM

Step 1: Start the Program

Step 2: Initialize the threads with the pthread_t data type.

Step 3: Create threads using pthread_create function.

Step 4: If the thread identifier is equal to the created pthread_id then the first thread will be processed otherwise the second thread will be processed.

Step 5: After the execution of one thread the cpu is kept idle for 5 units of time using sleep.

Step 6: Stop the program.

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("\n can't create thread :[%s]", strerror(err));
        else
```

```
        printf("\n Thread created successfully\n");
        i++;
    }
    sleep(5);
    return 0;
}
```

OUTPUT

-bash-3.2\$ **gcc threadpgm.c -lpthread**

-bash-3.2\$ **./a.out**

Thread created successfully

First thread processing

Thread created successfully

Second thread processing

CONCLUSION:

Thus the program to implement threading and synchronization executed and verified.

C

CONTENT BEYOND SYLLABUS

IMPLEMENTATION OF MEMORY ALLOCATION

AIM:

To implement first fit, best fit and worst fit using C.

ALGORITHM:

- 1: Get the number of free space available.
- 2: Get the starting address of each free space and how much space available.
- 3: While getting the address and space we have to check whether there is duplication, if there is a duplication then give the error message for duplication & get the address and space once again.
- 4: Get the space for the process.
- 5: If choice is first fit, then search the first memory location from the available list its size should be greater than or equal to the process size and or equal to available space in first fit order.
- 6: If choice is best fit, then sort the available list in ascending order based on the space. and search the first memory location from the available list its size should be greater than or equal to the process size and or equal to available space in best fit order.
- 7: If choice is worst fit, then sort the available list in descending order based on the space. and search the first memory location from the available list its size should be greater than or equal to the process size and or equal to available space in worst fit order.
- 8: If there is no sufficient space output the error message.
- 9: Display the process and its corresponding allocated memory space

Program:

```
#include<stdio.h>
int main()
{
    int n,i,z,s[10],a[10],p,j,ch,k,l,temp,b[10],flag=0;
    char ch1,ch2;
    do
    {
        printf("enter the no. of memory spaces");
        scanf("%d",&n);
        printf("enter the memory size");
        for(i=0;i<n;i++)
            scanf("%d",&a[i]);
        printf("enter the process size to be allocated");
        scanf("%d",&p);
        do
        {
            printf("\n1.BESTFIT\n2.FIRSTFIT\n3.EXIT");
            printf("\nenter your choice");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:
```



```

for(i=0;i<n;i++)
{
b[i]=a[i]-p;
if(b[i]>=0)
flag=1;
}
if(flag==1)
{
for(i=0;i<n-1;i++)
for(l=i;l<n;l++)
if(b[i]>b[l])
{
temp=b[i];
b[i]=b[l];
b[l]=temp;
}
if(b[0]<0)
{
for(i=0;i<n;i++)
while(b[i]<0)
{
i++;
z=b[i];
}
k=p+z;
printf("\nprocess size \t memory size\n");
printf("\n%d\t\t%d",p,k);
break;
}
else
{
k=p+b[0];
printf("\nprocess size \t memory size\n");
printf("\n%d\t\t%d",p,k);
}
}
else
printf("the required memory size is not available..");
break;
case 2:
for(i=0;i<n;i++)
{
b[i]=a[i]-p;
if(b[i]>=0)
flag=1;
}
if(flag==1)

```

```

{
for(i=0;i<n;i++)
if(b[i]>=0)
{
k=p+b[i];
break;
}
printf("process size \t memory size \n");
printf("\n%d\t\t%d",p,k);
}
else
printf("\nthe required memory size is not available...");
break;
case 3:
exit(0);
break;
}
printf("\n DO you want to continue (y/n):");
scanf("%s",&ch1);
}
while(ch1=='y'||ch1=='Y');
printf("\nDO you want to continue(y/n)");
scanf("%s",&ch2);
}
while(ch2=='y'||ch2=='Y');
}

```

SAMPLE INPUT AND OUTPUT:

./a.out

Enter the no of units of memory spaces:4

Enter the memory sizes:17

16

12

13

Enter the number of process size to be allocated: 14

1. BEST FIT

2. FIRST FIT

3. WORST FIT

4. EXIT

ENTER YOUR CHOICE 1

Process size	memorysize
--------------	------------

14

16

Do you want to continue?(Y/N):Y

1. BEST FIT

2. FIRST FIT

3. WORST FIT

```
4. EXIT
ENTER YOUR CHOICE 2
Process size      memory size
14                12
Do you want to continue?(y/n):y
1. BEST FIT
2. FIRST FIT
3. WORST FIT
4. EXIT
ENTER YOUR CHOICE 3
Process size      memory size
14                17
Do you want to continue?(y/n):y
1. BEST FIT
2. FIRST FIT
3. WORST FIT
4. EXIT
ENTER YOUR CHOICE 4
```

CONCLUSION:

The program for implementation of best fit and worst fit was executed successfully and output was verified.