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

AIM:

To study about the basics of UNIX Commands

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.

By1980, 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 LINUXSYSTEM:

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

No:1.b	BASICS OF UNIX COMMANDS
	BASIC UNIX COMMANDS

AIM:

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

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	O1
+%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:$lsls–s

All files (include files with prefix)

ls–l Lodetai (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:whowhoami

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
10/2*3
15
```

```
$ bc
scale =1
2.25+1
3.35
quit
```

```
$ bc
ibase=2
obase=16
11010011
89275
1010
Ā
Quit
```

```
$ bc
sqrt(196)
14 quit
```

```
$bc
for(i=1;i<3;i=i+1)I
1
2
3 quit
```

```
$ bc-l
scale=2
s(3.14)
0
```

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:$ch mod 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 g–group o– others	+assign -remove =assign absolutely	r– read w– write x-execute

Syn: $wc –l filename
$wc –w filename
$wc–c filename

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 is 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.

Itworksgreatspeedandiscasesensitive.vihaspowerfulundofunctionsandhas3modes:

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

RESULT:

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 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: Ifpid!=-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>
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:

RESULT:

.

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>
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);
 }
}
```

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> 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:

2. 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.
STEP4: 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++;
}
}
}
```

OUTPUT

RESULT:

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=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

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=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

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

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

RESULT:

Ex.No:5	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  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\t%d",p[i].pno,p[i].btime);
printf("\t\t%d\t\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);
}
```

OUTPUT:

Ex.No:5.b	CPU SCHEDULING ALGORITHMS
	ROUND ROBIN SCHEDULING

AIM:

To write a C program for implementation of Round Robin 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 and read the time quantum.
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<stdlib.h>
struct rr
{
int pno,btime,sbtime,wtime,lst;
}p[10];
int main()
{
int pp=-1,ts,flag,count,ptm=0,i,n,twt=0,totttime=0;
printf("\n round robin scheduling..... ");
printf("enter no of processes:");
scanf("%d",&n);
printf("enter the time slice:");
scanf("%d",&ts); printf("enter
the burst time");
for(i=0;i<n;i++)
{
printf("\n process%d\t",i+1);
scanf("%d",&p[i].btime);
p[i].wtime=p[i].lst=0;
p[i].pno=i+1;
p[i].sbtime=p[i].btime;
}
```

```
printf("scheduling ...\n");
do
{
flag=0;
for(i=0;i<n;i++)
{
count=p[i].btime;
if(count>0)
{
flag=-1;
count=(count>=ts)?ts:count;
printf("\n process %d",p[i].pno);
printf("from%d",ptm);
ptm+=count;
printf("to%d",ptm);
p[i].btime-=count;
if(pp!=i)
{
pp=i;
p[i].wtime+=ptm-p[i].lst-count;
p[i].lst=ptm;
}
}
```

OUTPUT:

Ex.No:5.c	CPU SCHEDULING ALGORITHMS
	FCFS

AIM:

To write a C program for implementation of FCFS and SJF 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 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 timeand turnaround time.
Step 5: Calculate total wait time and total turnaround time by dividing by total number ofprocess.
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[i].ttime;
for(i=0;i<n;i++)
{
p[i].wtime=p[i-1].wtime+p[i-1].btim
p[i].ttime=p[i].wtime+p[i].btime;
totttime+=p[i].ttime;
towtwtime+=p[i].wtime;
}
for(i=0;i<n;i++)
{{
printf("\n waiting time for process”);
printf("\n turn around time for process”);
printf("\n");
}}
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n);
printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```

OUTPUT:

Ex.No:5.d	CPU SCHEDULING ALGORITHMS
	SJF SCHEDULING

AIM:

To write a C program for implementation of SJF 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 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>
typedef struct
{
int  pid;
int btime;
int wtime;
}
sp;
int main()
{
int i,j,n,tbm=0,towtwtime=0,totttime
sp*p,t;
printf("\n sjf schaduling ..\n");
printf("enter the no of processor");
scanf("%d",&n);
p=(sp*)malloc(sizeof(sp));
printf("\n enter the burst time");
for(i=0;i<n;i++)
{
printf("\n process %d\t",i+1);
scanf("%d",&p[i].btime);
p[i].pid=i+1;
p[i].wtime=0;
```

```
}
for(i=0;i<n;i++)
for(j=j+1,j<n;j++)
{
if(p[i].btime>p[j].btime)
{
t=p[i];
p[i]=p[j];
p[j]=t;
}}
printf("\n process scheduling\n");
printf("\n process \tburst time \t w
for(i=0;i<n;i++)
{
towtwtime+=p[i].wtime=tbm;
tbm+=p[i].btime;
printf("\n%d\t\t%d",p[i].pid,p[i].bt
printf("\t\t%d\t\t%d",p[i].wtime,p[i
}
totttime=tbm+towtwtime;
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n);
printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```

OUTPUT:

RESULT:

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:-

RESULT:

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[]="poooda..... ";
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",shmid);
}
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:

RESULT:

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 ",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 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:

RESULT:

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 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("********** 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");
for(i=0;i<n;i++)
{printf("P%d\t",dead[i]);
}}
else
{
printf("\nNo Deadlock Occur"); }}
```

OUTPUT:

RESULT:

Ex.No:10	THREADING & SYNCHRONIZATION APPLICATIONS

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;
}
```

SAMPLE OUTPUT:

RESULT:

Ex.No:11.a	MEMORY ALLOCATION METHODS FOR FIXED PARTITION
	FIRST FIT

AIM:

To write a C program for implementation memory allocation methods for fixed partition using first fit.

ALGORITHM:

Step 1:Define the max as 25.
Step 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. Step 3: Get the number of blocks,files,size of the blocks using for loop.
Step 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
Step 5: Check highest<temp,if so assign ff[i]=j,highest=temp
Step 6: Assign frag[i]=highest, bf[ff[i]]=1,highest=0
Step 7: Repeat step 4 to step 6.
Step 8: Print file no,size,block no,size and fragment.
Step 9: Stop the program.

PROGRAM:

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

```
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

OUTPUT:

Ex.No:11.b	MEMORY ALLOCATION METHODS FOR FIXED PARTITION
	WORST FIT

AIM:

To write a C program for implementation of FCFS and SJF scheduling algorithms.

ALGORITHM:

Step 1:Define the max as 25.
Step 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. Step 3: Get the number of blocks,files,size of the blocks using for loop.
Step 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
Step 5: Check temp>=0,if so assign ff[i]=j break the for loop.
Step 6: Assign frag[i]=temp,bf[ff[i]]=1;
Step 7: Repeat step 4 to step 6.
Step 8: Print file no,size,block no,size and fragment.
Step 9: Stop the program.

PROGRAM:

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

```
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

OUTPUT:

Ex.No:11.c	MEMORY ALLOCATION METHODS FOR FIXED PARTITION
	BEST FIT

AIM:

To write a C program for implementation of FCFS and SJF scheduling algorithms.

ALGORITHM:

Step 1:Define the max as 25.
Step 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. Step 3: Get the number of blocks,files,size of the blocks using for loop.
Step 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
Step 5: Check lowest>temp,if so assign ff[i]=j,highest=temp
Step 6: Assign frag[i]=lowest, bf[ff[i]]=1,lowest=10000
Step 7: Repeat step 4 to step 6.
Step 8: Print file no,size,block no,size and fragment.
Step 9: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
```

```
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

OUTPUT:

RESULT:

Ex.No:12	PAGING TECHNIQUE OF MEMORY MANAGEMENT

AIM:

To write a c program to implement Paging technique for memory management.

ALGORITHM:

Step 1: Start the process
Step 2: Declare page number, page table, frame number and process size.
Step 3: Read the process size, total number of pages
Step 4: Read the relative address
Step 5: Calculate the physical address
Step 6: Display the address
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];
int counter;
pthread_mutex_t lock;
void* doSomeThing(void *arg)
{
pthread_mutex_lock(&lock);
 unsigned long i = 0;
counter += 1;
printf("\n Job %d started\n", counter);
for(i=0; i<(0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
pthread_mutex_unlock(&lock);
return NULL;
}
int main(void)
{
int i = 0;
int err;
if (pthread_mutex_init(&lock, NULL) != 0)
{ printf("\n mutex init failed\n");
return 1;
```

```
}
while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err != 0)
printf("\ncan't create thread :[%s]", strerror(err));
i++;
}
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_mutex_destroy(&lock);
return 0;}
```

OUTPUT:

RESULT:

Ex.No:13.a	PAGE REPLACEMENT ALGORITHMS
	FIFO

AIM:

To write a C program for implementation of FIFO page replacement algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the necessary variables.
Step 3: Enter the number of frames.
Step 4: Enter the reference string end with zero.
Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen.
Step 6: When a page is brought into memory, it is inserted at the tail of the queue.
Step 7: Initially all the three frames are empty.
Step 8: The page fault range increases as the no of allocated frames also increases.
Step 9: Print the total number of page faults.
Step 10: Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int i=0,j=0,k=0,i1=0,m,n,rs[30],flag=1,p[30];
system("clear");
printf("FIFO page replacement algorithm ...\\n");
printf("enter the no. of frames:");
scanf("%d",&n);
printf("enter the reference string:");
while(1)
{
scanf("%d",&rs[i]);
if(rs[i]==0)
break;
i++;
}
m=i;
for(j=0;j<n;j++)
p[j]=0;
for(i=0;i<m;i++)
```

```
{
flag=1;
for(j=0;j<n;j++)
if(p[j]==rs[i])
{
printf("data already in page... \n");
flag=0;
break;
}
if(flag==1)
{
p[i1]=rs[i];
i1++;
k++;
if(i1==n)
i1=0;
for(j=0;j<n;j++)
{
printf("\n page %d:%d",j+1,p[j]);
if(p[j]==rs[i])
printf("*");
}
printf("\n\n");
}
}
printf("total no page faults=%d",k);
}
```

OUTPUT:

Ex.No:13.b	PAGE REPLACEMENT ALGORITHMS
	LRU

AIM:

To write a c program to implement LRU page replacement algorithm.

ALGORITHM:

Step 1: Start the process
Step 2: Declare the size
Step 3: Get the number of pages to be inserted
Step 4: Get the value
Step 5: Declare counter and stack
Step 6: Select the least recently used page by counter value
Step 7: Stack them according the selection.
Step 8: Display the values
Step 9: Stop the process

ROGRAM:

```
#include<stdio.h>
main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
{
c1=0;
for(j=0;j<f;j++)
{
if(p[i]!=q[j])
c1++;
```

```
}
if(c1==f)
{c++;
if(k<f)
{q[k]=p[i];
k++;
for(j=0;j<k;j++)
printf("\t%d",q[j]);
printf("\n");
}
else
{for(r=0;r<f;r++)
{c2[r]=0;
for(j=i-1;j<n;j--)
{if(q[r]!=p[j])
c2[r]++;
else
break;
}}
for(r=0;r<f;r++)
b[r]=c2[r];
for(r=0;r<f;r++)
{
for(j=r;j<f;j++)
{
if(b[r]<b[j])
{
t=b[r];
b[r]=b[j];
b[j]=t;
}}}
for(r=0;r<f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
printf("\t%d",q[r]);
}
printf("\n");
}}}
printf("\nThe no of page faults is %d",c);
}
```

OUTPUT:

Ex.No:13.c	PAGE REPLACEMENT ALGORITHMS
	LFU

AIM:

To write C program to implement LFU page replacement algorithm.

ALGORITHM:

Step 1: Start the process
Step 2: Declare the size
Step 3: Get the number of pages to be inserted
Step 4: Get the value
Step 5: Declare counter and stack
Step 6: Select the least frequently used page by counter value
Step 7: Stack them according the selection.
Step 8: Display the values
Step 9: Stop the process

PROGRAM:

```
#include<stdio.h>
int main()
{
int f,p;
int pages[50],frame[10],hit=0,count[50],time[50];
int  i,j,page,flag,least,minTime,temp;
printf("Enter no of frames : ");
scanf("%d",&f);
printf("Enter no of pages : ");
scanf("%d",&p);
for(i=0;i<f;i++)
{
frame[i]=-1;
}
for(i=0;i<50;i++)
{
count[i]=0;
}
printf("Enter page no : \n");
for(i=0;i<p;i++)
{
```

```
scanf("%d",&pages[i]);
}
printf("\n");
for(i=0;i<p;i++)
{
count[pages[i]]++;
time[pages[i]]=i;
flag=1;
least=frame[0];
for(j=0;j<f;j++)
{
if(frame[j]==-1 || frame[j]==pages[i])
{
if(frame[j]!=-1)
{
hit++;
}
flag=0;
frame[j]=pages[i];
break;
}
if(count[least]>count[frame[j]])
{
least=frame[j];
}
}
if(flag)
{
minTime=50;
for(j=0;j<f;j++)
{
if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
{
temp=j;
minTime=time[frame[j]];
}
}
count[frame[temp]]=0;
frame[temp]=pages[i];
}
for(j=0;j<f;j++)
{
printf("%d ",frame[j]);
}
```

```
printf("\n");
}
printf("Page hit = %d",hit);
return 0;
}
```

OUTPUT:

RESULT:

Ex.No:14.a	FILE ORGANIZATION TECHNIQUE
	SINGLE LEVEL DIRECTORY

AIM:

To write C program to organize the file using single level directory.

ALGORITHM:

Step-1: Start the program.
Step-2: Declare the count, file name, graphical interface.
Step-3: Read the number of files
Step-4: Read the file name
Step-5: Declare the root directory
Step-6: Using the file eclipse function define the files in a single level
Step-7: Display the files
Step-8: Stop the program

FLOWCHART:

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm,count,i,j,mid,cir_x;
char fname[10][20];
clrscr();
initgraph(&gd,&gm,"c:/tc/bgi");
cleardevice();
setbkcolor(GREEN);
printf("enter number of files");
scanf("&d",&count);
if(i<count)
// for(i=0;i<count;i++)
{
cleardevice();
setbkcolor(GREEN);
printf("enter %d file name:",i+1);
scanf("%s",fname[i]);
setfillstyle(1,MAGENTA);
```

```
mid=640/count;
cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);
settextjustify(1,1);
outtextxy(320,125,"root directory");
setcolor(BLUE);
i++;
for(j=0;j<=i;j++,cir_x+=mid)
{
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[i]);
}}
getch();
}
```

Ex.No:14.b	FILE ORGANIZATION TECHNIQUE
	TWO LEVEL DIRECTORY

AIM:

To write C program to organize the file using two level directory.

ALGORITHM:

Step-1: Start the program.
Step-2: Declare the count, file name, graphical interface.
Step-3: Read the number of files
Step-4: Read the file name
Step-5: Declare the root directory
Step-6: Using the file eclipse function define the files in a single level
Step-7: Display the files
Step-8: Stop the program

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 main()
{int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"null",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\tc\bgi");
display(root);
getch();
closegraph();
}
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 the name of dir file name %s",dname);
fflush(stdin);
gets((*root)->name);
if(lev==0 || lev==1)
(*root)-> ftype=1;
else
(*root)->ftype=2;
(*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)
{if(lev==0 || lev==1)
{if((*root)->level==0)
printf("how many users");
else
printf(" how many files");
printf("(for %s):",(*root)->name);
scanf("%d",&(*root)->nc);
}else
(*root)->nc=0;
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;
}}
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:

RESULT:

Ex.No:15.a	FILE ALLOCATION STRATEGIES
	SEQUENTIAL

AIM:

To write a C program for sequential file for processing the student information.

ALGORITHM:

Step-1: Start the program.
Step-2: Get the number of records user want to store in the system.
Step-3: Using Standard Library function open the file to write the data into the file.
Step-4: Store the entered information in the system.
Step-5: Using do..While statement and switch case to create the options such as
1-DISPLAY, 2.SEARCH, 3.EXIT.
Step-6: Close the file using fclose() function.
Step-7: Process it and display the result.
Step-8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
typedef struct
{int sno;
char name[25];int m1,m2,m3;
}STD;
void display(FILE *);
int search(FILE *);
void main()
{int i,n,sno_key,opn;
FILE *fp;
clrscr();
printf("How many records ?");
scanf("%d",&n);
fp=fopen("stud.dat","w");
for(i=0;i<n;i++)
{printf("Enter the student information : %d(sno,Name,M1,M2,M3):",i+1);
scanf("%d%s%d%d%d,&s.sno,s.name,&s.m1,&s.m2,&s.m3);
fwrite(&s,sizeof(s),1,fp);
}
fclose(fp);
fp=fopen("stdu.dat","r");
do
```

```
{printf("1-DISPLAY\n2.SEARCH\n 3.EXIT\n YOUR OPTION: ");
scanf("%d",&open);
switch(opn)
{
case 1:
printf("\n Student Records in the file \n");
display(fp);
break;
case  2:
printf("Read sno of the student to be searched :");
scanf("%d",&sno_key);
if(search(fp,sno_key)){
printf("success!! Record found in the file\n");
printf("%d\t%s\t%d\t%d\t%d\n", s.sno,s.name,s.m1,s.m2,s.m3);
}
else
printf("Failure!! Record %d not found\n",sno_key);
break;
case  3:
printf("Exit !! press key");
getch();
break;
default:
printf("Invalid option!!! Try again!!\n");
break;
}
}while(opn!=3);
fclose(fp);
}
Void display(FILE *fp)
{rewind(fp);
while(fread(&s,sizeof(s),1,fp))
printf("%d\t%s\t%d\t%d\t%d\n",s.sno,s.name,s.m1,s.m2,s.m3);
}
int search(FILE  *fp,int sno_key)
{rewind(fp);
while(fread(&s,sizeof(s),1,fp))
If(s.sno==sno_key)
return 1;
return 0;
}
```

OUTPUT:

Ex.No:15.b	FILE ALLOCATION STRATEGIES
	LINKED

AIM:

To write a C program for random access file for processing the employee details.

ALGORITHM:

Step-1: Start the program.
Step-2: Get the number of records user want to store in the system.
Step-3: Using Standard Library function open the file to write the data into the file.
Step-4: Store the entered information in the system.
Step-5: Using do..While statement and switch case to create the options such as
1-DISPLAY, 2.SEARCH, 3.EXIT.
Step-6: Close the file using fclose() function.
Step-7: Process it and display the result.
Step-8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct record
{
char empname[20];
int age;
float salary;
};
typedef struct record person
FILE *people;
void main()
{
person employee;
int I,n;
FILE *fp;
printf("How many records:");
scanf("%d",&n);
fp=fopen("PEOPLE.txt","w");
for(i=0;i<n;i++)
{
printf("Enter the employee information :%d(EmpName, Age,Salary):",i+1);
scanf("%s%d%f",employee.empname,&employee.age,& employee.salary);
```

```
fwrite(,&employee.sizeof(employee,1,people);
}
fclose(fp);
int rec,result;
people=fopen("PEOPLE.txt","r");
printf("Which record do you want to read from file?);
scanf("%d",&rec);
while(rec>=0)
{
fseek(people,rec*sizeof(employee),SEEK_SET);
result=fread(&em[ployee,sizeof(employee),1,people)
if(result==1)
{
printf("\n RECORD %d\n",rec);
printf("Given name:%s\n", employee.empname);
printf("Age:%d years\n",employee.age);
printf("Current salary:$ %8.2f\n\n",employee.salary);
}
else
printf( "\n RECORD %d not found !\n\n",rec);
printf("Which record do you want(0to3)"):
scanf("%d"<&rec):
}
fclose(people):
getch():
}
```

OUTPUT:

Ex.No:15.c	FILE ALLOCATION STRATEGIES
	INDEXED

AIM:

To write a C program for random access file for processing the employee details.

ALGORITHM:

Step-1: Start the program.
Step-2: Get the number of records user want to store in the system.
Step-3: Using Standard Library function open the file to write the data into the file.
Step-4: Store the entered information in the system.
Step-5: Using do..While statement and switch case to create the options such as
1-DISPLAY, 2.SEARCH, 3.EXIT.
Step-6: Close the file using fclose() function.
Step-7: Process it and display the result.
Step-8: Stop the program.

PROGRAM:

```
#include
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();
}
```

OUTPUT:

RESULT: