CHENNAI INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University Chennai), Sarathy Nagar, Kundrathur, Chennai - 600069.

Department of Computer Science and Engineering

Vision of the Department

To Excel in the emerging areas of Computer Science and Engineering by imparting knowledge, relevant practices and inculcating human values to transform the students as potential resources to contribute innovatively through advanced computing in real time situations.

Mission of the Department

DM1. To provide strong fundamentals and technical skills for Computer Science applications through effective teaching learning methodologies.

DM2. To transform lives of the students by nurturing ethical values, creativity and novelty to become Entrepreneurs and establish start-ups.

DM3. To habituate the students to focus on sustainable solutions to improve the quality of life and the welfare of the society.

DM4. To enhance the fabric of research in computing through collaborative linkages with industry and academia.

DM5. To inculcate learning of the emerging technologies to pursue higher studies leading to lifelong learning.

CS8461 OPERATING SYSTEMS LABORATORY

COURSE OBJECTIVES AND OUTCOMES

OBJECTIVES:

To learn Unix Commands and shell programming.
To implement various CPU Scheduling Algorithms.
To implement process creation and Inter Process Communication.
To implement Deadlock Avoidance and Deadlock Detection Algorithms.
To implement Page Replacement Algorithms.
To implement File Organization and File Allocation Strategies.

List of Experiments:

1. Basics of UNIX commands.
2. Write programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.
3. Write C programs to simulate UNIX commands like cp, ls, grep, etc.
4. Shell Programming.
5. Write C Programs to implement various CPU
Scheduling algorithms.
6. Implementation of Semaphores
7.Implementation of Shared memory and IPC
8.Bankers Algorithm for Dead Lock Avoidance.
9.Implementation of Dead Lock Detection Algorithm
10.Write a C program to implement Threading & Synchronization Applications
11.Implementation of the following Memory Allocation Methods for fixed partition
    a) First Fit   b) Worst Fit   c) Best Fit
12.Implementation of Paging Technique of memory management
13.Implementation of the following Page Replacement Algorithms
    a) FIFO   b) LRU   c) LFU
14. Implementation of the various File Organization Techniques
15.Implementation of the following File Allocation Strategies
    a) Sequential   b) Indexed   c) Linked

OUTCOMES:

At the end of the course, the student should be able to

- Compare the performance of various CPU Scheduling Algorithms
- Implement Deadlock avoidance and Detection Algorithms
- Implement Semaphores
- Create processes and implement IPC
- Analyze the performance of the various Page Replacement Algorithms
- Implement File Organization and File Allocation Strategies

System Requirements
Standalone desktops server with C / C++ / Java / Equivalent complier 30 Nos.

INDEX

CONTENT BEYOND THE SYLLABUS

EXP. NUMBER	DATE OF EXPERIMENT	TITLE	DATE OF COMPLETION	MARKS	SIGNATURE OF THE FACULTY
16		Virtualization using Virtual box			
17		Kernel Configuration			

Ex. No: 1	BASICS OF UNIX COMMANDS

AIM:

To study and execute the commands in Unix.

COMMANDS:

1. Date Command:

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

Syntax :

$date

$date +%ch

Options : -

a = Abbreviated weekday.
A = Full weekday.
b = Abbreviated month.
B = Full month.
c = Current day and time.
C = Display the century as a decimal number.
d = Day of the month.
D = Day in „mm/dd/yy‟ format
h = Abbreviated month day.
H = Display the hour.
L = Day of the year.
m = Month of the year.
M = Minute.
P = Display AM or PM
S = Seconds
T = HH:MM:SS format
u = Week of the year.
y = Display the year in 2 digit.
Y = Display the full year.
Z = Time zone .
To change the format:

Syntax:

$date "+%H-%M-%S‟

2. Calendar 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>

Multi line echo command:

To have the output in the same line, the following commands can be used.

Syntax: $echo <text\>text

To have the output in different line, the following command can be used.

Syntax: $echo “text
>line2
>line3”

4.’who’ Command:

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

Syntax: $who –option’s

Options: -

H–Display the output with headers.

b–Display the last booting date or time or when the system was lastely rebooted.

5.’who am i’ Command:

Display the details of the current working directory.

Syntax: $who am i

6.’tty’ Command:

It will display the terminal name.

Syntax: $tty

7.’Binary’ Calculator Command:

It will change the „$‟ mode and in the new mode, arithmetic operations such as +,- ,*,/,%,n,sqrt( ),length( ),=, etc can be performed . This command is used to go to the binary calculus mode.

Syntax:

$bc operations
^d
$

1 base –inputbase

0 base – outputbase are used for base conversions.

Base:

Decimal = 1 Binary = 2 Octal = 8 Hexa = 16

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. MANIPULATION Command:

It is used to manipulate the screen.

Syntax: $tput <argument>

Arguments:

1. Clear – to clear the screen.
2. Longname – Display the complete name of the terminal.
3. SMSO – background become white and foreground become black color.
4. rmso – background become black and foreground becomes white color.

11. 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 column wise.
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

The > symbol is redirectory we use cat command.

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 <filename source>>><destination filename>
$ cat <source filename>>><destination filename> it is avoid overwriting.

Options: -

-n content of file with numbers included with blank lines.

Syntax:

$cat –n <filename>

4. SORTING A FILE:

To sort the contents in alphabetical order in reverse order.

Syntax:

$sort <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>

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 screen full of information is displayed after which the pagecommand 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 barkey.

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 data 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 anothercommand.

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 1. MESG

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

Syntax: $mesg y

2. Command: WRITE

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

Syntax: $write <user name>

3. Command: WALL

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

Syntax: $wall <message>

4. Command: MAIL

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

Syntax: $mail <user name>

5. Command: REPLY

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

Syntax: $reply<user name>

INFERENCE:

RESULT:

Thus the program to study and execute Unix commands is done successfully.

Ex. No: 2	SYSTEM CALLS OF UNIX OS

AIM:

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

ALGORITHM :

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

SYSTEM CALLS USED:

1. fork( ) Used to create new processes. The new process consistsof a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero. **Syntax** : fork( )

2. execlp( ) Used after the fork( ) system call by one of the two processes to replace the process‟ memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution.The child process overlays its address space with the UNIX command /bin/ls using the execlp system call. **Syntax** : execlp( )

3. wait( ) The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated. **Syntax** : wait(NULL)

4. exit( ) A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call). **Syntax**: exit(0)

5. getpid( )Each process is identified by its id value. This function is used to get the id value of a particular process.

6. getppid( ) Used to get particular process parent‟s id value. perror( )

7. perror( ) Indicate the process error.

8. opendir( ) Open a directory.

9. readdir( ) Read a directory.

10. closedir( ) Close a directory.

11. execlp( ) Used after the fork( ) system call by one of the two processes to replace the process‟‟memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution. The child process overlays its address space with the UNIX command /bin/ls using the execlp system call.
    Syntax : execlp( )

12. fork ( ) Used to create new process. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.
    Syntax: fork ( )

13. wait ( ) The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated. **Syntax**: wait (NULL)

14. exit ( ) A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call). **Syntax**: exit(0)

PROGRAM:

1)FORK SYSTEM CALL PROGRAM:
```
#include<stdio.h>
#include<stdlib.h>
int main( )
{
        int pid;
        pid=fork( );
        printf("\n THIS LINE EXECUTED TWICE");
        if(pid==-1)        {
                printf("\n CHILD PROCESS NOT CREATED\n");
                        exit(0);
            }
            if(pid==0)        {
                        printf("\n I AM CHILD PROCESS AND MY ID IS %d \n",getpid( ));
                        printf("\n THE CHILD PARENT PROCESS ID IS:%d \n",getppid( ));
            }
            else
            {
                        printf("\n I AM PARENT PROCESS AND MY ID IS:%d\n",getpid( ));
                        printf("\n THE PARENTS PARENT PROCESS ID IS:%d\n",getppid( ));
            }
            printf("\n IT CAN BE EXECUTED TWICE");
            printf("\n");
}
```

OUTPUT:

2)SYSTEM CALL WITH ARGUMENT PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,int *argv[])
{
	int pid,i;
	pid=fork( );
	printf("\n THIS LINE EXECUTED TWICE");
	if(pid==-1)	{
		printf("\n CHILD PROCESS NOT CREATED\n");
			exit(0);
	   }
	   if(pid==0)	{
			printf("\n CHILD PROCESS IS IN PROGRESS\n");
			for(i=0;i<5;i++)
				printf("\n THE CHILD PROCESSING VALUE IS:%d \n",i);
			execvp("ls",argv);
	   }
	   else	{
			printf("\n PARENT PROCESS IS IN WAITING\n");
			printf("\n CHILD PROCESS COMPLETED ITS TASK\n");
	   }
	   exit(0);
}
```

OUTPUT:

3)WORKING WITH DIRECTORY PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/dir.h>
void main(int argc,char *argv[])
{
	DIR *dir;
	struct dirent *rddir;
	printf("LISTING THE DIRECTORY CONTENT\n");
	dir=opendir(argv[1]);
	printf("THE CURRENT DIRECTORY FILES ARE:\n");
	while((rddir=readdir(dir))!=NULL)
	{
		printf("%s\n",rddir->d_name);
	}
	closedir(dir);
}
```

OUTPUT:

4)STAT SYSTEM CALL PROGRAM:

```
#include<stdio.h>
#include<sys/stat.h>
int main( )
{
        struct stat sfile;
        stat("stat.c",&sfile);
        printf("file st_uid:%d\n",sfile.st_uid);
        printf("file st_uid:%d\n",sfile.st_gid);
        printf("file st_size:%ld\n",sfile.st_size);
        printf("file st_blocks:%ld\n",sfile.st_blocks);
        printf("file serialno:%ld\n",sfile.st_ino);
        printf("file recent access time :%ld\n",sfile.st_atime);
        printf("file permission change time:%ld\n",sfile.st_ctime);
        printf("file recent modified time:%ld\n",sfile.st_mtime);
}
```

OUTPUT:

INFERENCE:

RESULT:
Thus the program for system call using Unix as been executed and output is verified successfully.

Ex. No: 3	SIMULATE UNIX COMMAND IN OS

AIM:
To write C Program to Simulate the **ls** Command in UNIX Operating system.

ALGORITHM :

```
Step 1: Start the process
Step 2: create a directory entry using dirent structure.
Step 3: To define a pointer to a structure, dp predefined structure DIR and another pointer to a
            structure called ep
 Step 4: The directory is opened using the opendir( ) function
Step 5: In the while loop read each entry using the readdir( ) function which returns a pointer
Step 6: If no pointer is returned the program is exited, else it will list the files inside the directory.
Step 7: The closedir( ) function closes this open directory.
Step 8: Stop the process.
```

PROGRAM CODING:

```
#include<stdio.h>
#include<string.h>
int main(int argc, char *argv[])
{
        FILE *fp1, *fp2;
        char ln[80];
        fp1=fopen(argv[1],"r");
        fp2=fopen(argv[2],"w");
        while(fgets(ln,80,fp1))
        {
                fputs(ln,fp2);
        }
        printf("FILE HAS BEEN COPIED SUCCESSFULLY\n");
        fclose(fp1);
        fclose(fp2);
}
```

OUTPUT:

3b) GREP COMMAND

AIM:
To write C Program to Simulate the grep Command in UNIX Operating system

ALGORITHM:

Step 1: Include necessary header files.

Step 2: Make necessary declarations.

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

Step 4: Read the pattern from the user.

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

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

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

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int strmat(char ln[],char pa[])
{
  int i,j,k;
  for(i=0;ln[i]!='\0';i++)
  {
    for(j=i,k=0;ln[j]==pa[k]&&pa[k]!='\0';j++,k++);
                if(k>0&&pa[k]=='\0')
                        return(1);
  }
  return 0;
}
int main(int argc,char *argv[])
{
  FILE *fp1;
  char ln[80];
  if(argc<3)
  {
    printf("USAGE: FILE_NAME SOUCRCE_FILE SEARCH PATTERN\n");
  }
  else
  {
            fp1=fopen(argv[2],"r");
            if(fp1=='\0')
            {
                        printf("\n SOURCE FILE CANNOT BE OPENED\n");
                         exit(0);
            }

  }
  while(fgets(ln,80,fp1))
  {
   if(strmat(ln,argv[1]))
     printf("%s",ln);
  }
  fclose(fp1);
}
```

OUTPUT:

INFERENCE:

RESULT:

The program for simulation of UNIX commands has been executed and output is verified successfully.

Ex. No: 4	SHELL PROGRAMMING

AIM:

To write and execute basic shell programs.

PROCEDURE:

1. GREATEST AMONG THREE NUMBERS

ALGORITHM:

```
Step 1. Read 3 numbers n1,n2 and n3
Step 2. Check if n1 is greater than n2

check n1 is greater than n3 also
  announce n1 as greatest
announce n3 as greatest
else, check whether n2 is greater than n3
     if yes, announce n2 as greatest

else announce n3 as greatest
```

PROGRAM:

```
echo "ENTER THREE NUMBERS"
read a b c
if [ $a -gt $b ] && [ $a -gt $c ]
then
echo "$a is greater"
elif [ $b -gt $c ]
then
echo "$b is greater"
else
echo "$c is greater"
fi
```

OUTPUT:

2.FACTORIAL OF A GIVEN NUMBER

ALGORITHM:

```
Step 1: Read n
Step 2: Initialize fact to 1 and i to n
Step 3: Repeat the following until i>0
Assign fact * i to fact
Decrement i by 1
```

PROGRAM:

```
echo "ENTER THE NUMBER:"
read n
```

```
fact=1
while [ $n -gt 1 ]
do
fact=`expr $fact \* $n | bc`
n=`expr $n - 1`
done
echo "FACTORIAL OF THE GIVEN NUMBER IS  $fact"
```

OUTPUT:

3.SUM OF ODD NUMBERS UPTO N

ALGORITHM:

```
Step 1: Read n
Step 2: Initialize x=1 and sum=0
Step 3: Repeat the following until x < n Assign sum + x to sum Increment x by2
```

PROGRAM:

```
echo "enter the range"
read n
x=1
sum=0
while [ $x -le $n ]
do
sum=`expr $sum + $x`
x=`expr $x + 2`
done
echo "sum=$sum"
```

OUTPUT:

4.GENERATION OF FIBONACCI NUMBERS

ALGORITHM:

```
Step 1: Read n
Step 2: Initialize p=-1, q=1 and I=1
Step 3: Repeat the following until I < n
Assign p + q to r
        Assign q to p
Assign r to q
Increment I by 1
```

PROGRAM:

```
echo "ENTER THE LIMIT:"
read n
p=-1
```

```
q=1
i=1
while [ $i -le $n ]
do
r=`expr $p + $q`
p=$q
q=$r
echo "$r"
i=`expr $i + 1`
done
```

OUTPUT:

5.IMPLEMENT THE ARITHMETIC CALCULATOR

ALGORITHM:
Step 1: Read a, b and option
Step 2: According to the option perform the operation

PROGRAM:

```
echo "ENTER THE VALUE OF A:"
read a
echo "ENTER THE VALUE OF B:"
read b
c=0
echo "ENTER THE OPTION TO PERFORM 1.ADDITION  2.SUBTRACTION 3.MULTIPLICATION
4.DIVISION"
read op
case "$op" in
1)c=`expr $a + $b`
echo $c;;
2)c=`expr $a - $b`
echo $c;;
3)c=`expr $a \* $b`
echo $c;;
4)c=`expr $a / $b`
echo $c;;
esac
exit
```

OUTPUT:

6.WRITE A SHELL PROGRAM TO FIND THE LARGEST DIGIT OF A NUMBER

ALGORITHM:

```
Step 1: Get a number from the user
Step 2: Obtain individual digit for the above number using modulo operator
Step 3: Initialize variable max with first digit
Step 4: Compare the value of max with the other digits, if the value of max is lesser update the value of max
Step 5: Display the value of max
```

PROGRAM:

```
echo "ENTER THE NUMBER"
read a
max=0
while [ $a -gt 0 ]
do
r=`expr $a % 10`
if [ $r -gt $max ]
then
max=$r
fi
a=`expr $a / 10`
done
echo "THE LARGEST DIGIT OF THE NUMBER:$max"
```

OUTPUT:

7. CHECK WHETHER GIVEN STRING IS A PALINDROME OR NOT.

ALGORITHM:

```
Step 1: Read a String
Step 2: Find the length of the string
Step 3: Start reading from the last character to the first character and store it as a new string in temp
Step 4: Compare both the strings, if it is same the given string is palindrome
```

PROGRAM:

```
echo "ENTER THE STRING TO CHECK PALINDROME"
read str
len=`echo $str | wc -c`
len=`expr $len-1 | bc`
i=1
j=`expr $len/2 | bc`
while [ $i -le $j ]
```

```
do
k=`echo $str | cut -c $i`
l=`echo $str | cut -c $len`
if [ $k != $l ]
then
echo "$str is not a palindrome"
exit
fi
i=`expr $i+1 | bc`
len=`expr $len-1 | bc`
done
echo "$str is  a palindrome"
```

OUTPUT:

8.WRITE A SHELL PROGRAM TO FIND OUT THE REVERSE OF A GIVEN NUMBER

ALGORITHM:

```
Step 1: Get a number from the user
Step 2: Set a loop upto the number is not equal to zero
Step 3: reminder=number%10
Step 4:rnum=rnum*10+reminder
Step 5: number=number/10
Step 6: if number==rnum print both are same
```

PROGRAM:

```
echo "ENTER THE NUMBER"
read n
rnum=0
while [ $n -ne 0 ]
do
remainder=`expr $n % 10 | bc`
rnum=`expr $rnum \* 10 + $remainder | bc`
n=`expr $n / 10 | bc`
done
echo "REVERSE OF THE NUMBER IS $rnum"
```

OUTPUT:

INFERENCE:

RESULT:

Thus the program to perform shell programming is executed and verified successfully.

Ex. No: 5	**IMPLEMENTATION OF CPU SCHEDULING ALGORITHM**

AIM:

To implement Scheduling algorithms using C are

a. FCFS Scheduling Algorithm
b. SJF Scheduling Algorithm
c. Priority Scheduling Algorithm
d. Round Robin Scheduling Algorithm

5.a.FCFSAlgorithm:

AIM:

To implement FCFS Scheduling algorithm using C.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the burst time

Step 4: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate

a) Waiting timess(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 6: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
int n,pid[10],at[10],bt[10],ft[10],wt[10],ta[10],i,j,t,stt=0,totta=0,totwt=0;
float avgta,avgwt;
printf("ENTER THE NO.OF PROCESSES:");
scanf("%d",&n);
for(i=1;i<=n;i++)          {
        pid[i]=i;
        printf("\n ENTER THE ARRIVAL TIME:");
        scanf("%d",&at[i]);
        printf("\n ENTER THE BURST TIME:");
        scanf("%d",&bt[i]);
}
for(i=1;i<=n;i++)
{
            for(j=i+1;j<=n;j++)
{
                        if(at[i]>at[j])
                        {
                           t=pid[i];
                           pid[i]=pid[j];
                           pid[j]=t;
                           t=at[i];
                           at[i]=at[j];
                           at[j]=t;
                           t=bt[i];
                           bt[i]=bt[j];
                           bt[j]=t;
                        }
             stt=at[1];                    }
}
printf("\nTHE VALUES OF THE ARRIVAL TIME IS %d",stt);
for(i=1;i<=n;i++)
{
             ft[i]=stt+bt[i];
             wt[i]=stt-at[i];
             ta[i]=ft[i]-at[i];
             totta=totta+ta[i];
             totwt=totwt+wt[i];
             stt=ft[i];           }
 avgta=(float)totta/n;
 avgwt=(float)totwt/n;
 printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\t\tWAIT TIME\tTAT\n");
 for(i=1;i<=n;i++)
            printf("%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
 printf("\nAVERAGE TURN AROUND TIME=%f",avgta);
 printf("\nAVERAGE WAITING TIME=%f",avgwt);
}
```

OUTPUT:

5. b Shortest Job First(NON-PRE-EMPTIVE SCHEDULING)

AIM:

To implement SJF Scheduling algorithm in C.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as _0‘ and its turnaround time as its burst time.

Step 6: Sort the processes names based on their Burt time Step 7 :For each process in the ready queue, calculate

a) Waiting timess(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 9: Stop the process

PROGRAM:

```
#include<stdio.h>
void main( )
{
        int i,j,t,n,stt=0,pid[10],at[10],bt[10],ft[10],att,wt[10],ta[10],totwt=0,totta=0;
        float avgwt,avgta;
        printf("ENTER THE NUMBER OF PROCESSES:");
        scanf("%d",&n);
     printf("\nENTER THE ARRIVAL TIME:");
        scanf("%d",&att);



for(i=1;i<=n;i++)
        {
                pid[i]=i;
                at[i]=att;
                printf("\nENTER THE BURST TIME:");
                scanf("%d",&bt[i]);
        }
        for(i=1;i<=n;i++)
        {
                for(j=i+1;j<=n;j++)
                {


if(bt[i]>bt[j])
                        {
                                t=pid[i];
                                pid[i]=pid[j];
                                pid[j]=t;

                                t=bt[i];
                                bt[i]=bt[j];
                                bt[j]=t;
                        }
                              stt=att;
                }       }

        for(i=1;i<=n;i++)
        {
                ft[i]=stt+bt[i];
                wt[i]=stt-at[i];
                ta[i]=ft[i]-at[i];
                totta=totta+ta[i];
                totwt=totwt+wt[i];
                stt=ft[i];
        }
        avgwt=(float)totwt/n;
```

```
	avgta=(float)totta/n;
	printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\tWAIT TIME\tTAT");
	for(i=1;i<=n;i++)
	{
		printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
	}
	 printf("\nAVERAGE TURN AROUND TIME=%f",avgta);
	 printf("\nAVERAGE WAITING TIME=%f",avgwt);
}
```

OUTPUT:

5.c Priority Scheduling

AIM:

To write a c program to simulate the CPU Scheduling priority algorithm.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Sort the ready queue according to the priority number.

Step 5: Set the waiting of the first process as ‗0' and its burst time as its turnaround time

Step 6: Arrange the processes based on process priority

Step 7:For each process in the Ready Q calculate

Step 8: for each process in the Ready Q calculate

a) Waiting timess(n)= waiting time (n-1) + Burst time (n-1) b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.

Step 10: Stop the process

PROGRAM:

```
#include<stdio.h>
void main( )
{
        int i,j,t,n,stt=0,pid[10],pr[10],at[10],bt[10],ft[10],att;
        int wt[10],ta[10],totwt=0,totta=0;
        float avgwt,avgta;
        printf("ENTER THE NO.OF.PROCESS:");
        scanf("%d",&n);
        printf("\nENTER THE ARRIVAL TIME:");
        scanf("%d",&att);
        for(i=1;i<=n;i++)
        {
                pid[i]=i;




at[i]=att;
                printf("\nENTER THE BURST TIME:");
                scanf("%d",&bt[i]);
                printf("\nENTER THE PRIORITY OF THE PROCESS:");
                scanf("%d",&pr[i]);
        }
```

```
	for(i=1;i<=n;i++)
	{
		 for(j=i+1;j<=n;j++)
		 {
			if(pr[i]>pr[j])
			{
				t=pid[i];
				pid[i]=pid[j];
				pid[j]=t;
				t=bt[i];
				bt[i]=bt[j];
				bt[j]=t;
				t=pr[i];


pr[i]=pr[j];
				pr[j]=t;
			}			}
	}
	stt=att;
	for(i=1;i<=n;i++)
	{
		ft[i]=stt+bt[i];
		wt[i]=stt-at[i];
		ta[i]=ft[i]-at[i];
		if(wt[i]<0)
			wt[i]=0;
		totwt=totwt+wt[i];
		totta=totta+ta[i];
		stt=ft[i];
	}
	avgwt=(float)totwt/n;
	avgta=(float)totta/n;
	printf("PNO\tARR TIME\tBURST TIME\tFINISH TIME\tWAIT TIME\tTURN TIME\n");
	for(i=1;i<=n;i++)
	{
		printf("\n%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
	}
	printf("\n THE AVERAGE WAITING TIME IS:%f\n",avgwt);
	printf("\n THE AVERAGE TURN TIME IS:%f\n",avgta);
}
```
OUTPUT:

5.d. Round Robin Scheduling

AIM:

To implement Round Robin Scheduling Algorithm in C

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1. Step 6: Consider the ready queue is a circular Q, calculate

(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n)

Step 7: Calculate(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

PROGRAM:

```
#include<stdio.h>
main( )
{
        int i,j,n,wt[10],ta[10],at[10],bt[10],tot_wt=0,tot_ta=0,ft[10],t;
        int s[10],prid[10],p[10],max=0,temp,stt=0,ts=0,x=0;
        float avg_wt,avg_ta;
        printf("Enter the no. of process:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                prid[i]=i;
                printf("\n Enter the Arrival time of process %d:",i);
                scanf("%d",&at[i]);
                printf("\n Enter the Burst time of the process %d:",i);
                scanf("%d",&bt[i]);
                wt[i]=0;
                p[i]=0;

if(at[i]>max)
                {
                        max=at[i];
                }
        }
        printf("\n Enter the time Slice:");
        scanf("%d",&ts);
        for(i=1;i<n;i++)
        {
                for(j=i+1;j<=n;j++)
                {
                        if(at[i]>at[j])
                        {
                                t=at[i];
                                at[i]=at[j];
                                at[j]=t;
                                t=bt[i];
                                bt[j]=bt[j];
```

```
                                        bt[j]=t;
                                }                       }               }
                for(i=1;i<=n;i++)
                {
                        s[i]=bt[i];
                }
                i=1;
                x=0;

        while(x<n)
                {
                        if(p[i]==1)
                        goto con;
                        if(at[i]>stt)
                        {
                                temp=max;
                                for(i=1;i<=n;i++)
                                {
                                        if(p[i]==0 && at[i]<=temp)
                                        {
                                                temp=at[i];
                                        }
                                }
                                if(temp>stt)
                                {
                                        stt=temp;
                                }
                                if(at[i]>stt)
                                        goto con;
                        }               if(s[i]>ts)
                        {
                                s[i]=s[i]-ts;
                                stt=stt+ts;
                        }
                        else
                        {
                                stt=stt+s[i];
                                ft[i]=stt;


        s[i]=0;
                                p[i]=1;
                                x++;
                        }
                        con:
                        i++;
                        if(i>n)
                        i=1;    }
                for(i=1;i<=n;i++)
                {
                        ta[i]=ft[i]-at[i];
                        wt[i]=ta[i]-bt[i];
                        tot_ta+=ta[i];
                        tot_wt+=wt[i];
                }
                avg_wt=(float)tot_wt/n;
                avg_ta=(float)tot_ta/n;
                printf("\n PNO\tARR TIME\tBURST TIME\tWAIT TIME\t TURN TIME\t FINISH        TIME");
                for(i=1;i<=n;i++)
                {
                        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t%d",i,at[i],bt[i],wt[i],ta[i],ft[i]);




        }
                printf("\n The Average Waiting Time is:%0.2f",avg_wt);

                printf("\n The Average Turn Time is   :%0.2f",avg_ta);
        }
```

OUTPUT:

INFERENCE:

RESULT:

Thus to write a c program to simulate the CPU Scheduling priority algorithm is executed and output is verified successfully.

Ex. No: 6	IMPLEMENTATION OF SEMAPHORES

AIM:

To implement the Producer and Consumer Problem using semaphores

ALGORITHM:

```
Step 1: Start
Step 2: Initialize the semaphore variable S
Step 3: In the producer function ,
        While s ==1 do nothing
        Produce the value
        Assign s=1
        Return
Step 4: In the Consumer function
        While s==0 do nothing
        Display the consumed value
        Assign s=0
        Return
Step 5: Create threads for producer and consumer function to make it run concurrently
Step 6: Stop
```

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#define num_loops 5
union semun
{
        int val;
        struct semid_ds *buf;
        short *array;
};
int main(int argc,char *argv[]){
        int semset_id;
        union semun sem_val;
        int child_pid;
        int i;
struct sembuf sem_op;
int rc;
        struct timespec delay;
        semset_id=semget(IPC_PRIVATE,1,0600);
        if(semset_id==-1)       {
                perror("semget");
                exit(1); }
        printf("SEMAPHORE SET CREATED SEMAPHORE SET ID%d\n",semset_id);
        sem_val.val=0;
        rc=semctl(semset_id,0,SETVAL,sem_val);
        child_pid=fork( );
        switch(child_pid)       {
                case 1:perror("fork");exit(1);
                case 0:for(i=0;i<num_loops;i++)                 {
                        sem_op.sem_num=0;
                        sem_op.sem_op=-1;
                        sem_op.sem_flg=0;
                        semop(semset_id,&sem_op,1);
                        printf("Consumer consumed item %d\n",i);
                        fflush(stdout);                 }
                       break;
                default:for(i=0;i<num_loops;i++)
        {
                                printf("Producer produced item %d\n",i);
                                fflush(stdout);
                                sem_op.sem_num=0;
                                sem_op.sem_op=1;
                                sem_op.sem_flg=0;
                                semop(semset_id,&sem_op,1);
                                if(rand( )>3*(RAND_MAX/4))                      {
                                        delay.tv_sec=0;
                                        delay.tv_sec=10;
                                        nanosleep(&delay,NULL);
                                }                       }break;
        }       return 0;}
```

OUTPUT:

INFERENCE:

RESULT:

Thus, the program for simulation of producer-consumer problem using semaphorehas been executed and output is verified successfully.

Ex. No: 7	SHARED MEMORY & INTERPROCESS COMMUNICATION

AIM:

To develop a client-server application program, this uses shared memory using IPC

ALGORITHM:

Server:

1. Define shared memory size of 30 bytes
2. Define the key to be 5600
3. Create a shared memory using shmget ( ) system calls and gets the shared memory id in variable shmid.
4. Attach the shared memory to server data space
5. Get the content to be placed in the shared memory from the user of the server.
6. Write the content in the shared memory, which will read out by the client.
7. Stop

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/msg.h>
#include<sys/ipc.h>
#include<string.h>
#define msgsz 3
typedef struct msgbuffer
{
        long mtype;
        char mtext[msgsz];
}mbuffer;
main( )
{
        int msgflg=IPC_CREAT|0666;
        key_t key;
        int msqid,buffer;
        mbuffer mb;
        key=1234;
        if((msqid=msgget(key,msgflg))<0)
        {
                perror("msgget");
                return 1;
        }
        else
                mb.mtype=1;
        strcpy(mb.mtext,"HAI");
        buffer=strlen(mb.mtext);
if(msgsnd(msqid,&mb,buffer,msgflg)<0)
        {
                printf("%d%ld%s%d",msqid,mb.mtype,mb.mtext,buffer);
                perror("msgsnd");
                return 1;
        }
        else
                printf(" MESSAGE : %s IS SENT ",mb.mtext);
        return 0;
}
```

OUTPUT:

CLIENT

Step1: Initialize size of shared memory shmsize to 27.

Step2:Initialize key to 2013 (same value as in server)

Step3: Obtain access to the same shared memory segment using same key.

If obtained then display the shmid else print "Server not started"

Step4: Attach client process to the shared memory using shmmat with shmid as parameter.

If pointer to the shared memory is not obtained, then stop.

Step5: Read contents of shared memory and print it.

Step6: After reading, modify the first character of shared memory to '*'

Step7: Stop

PROGRAM FOR CLIENT:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/msg.h>
#include<sys/ipc.h>
#include<stdio.h>
#define msgsz 8
typedef struct msgbuffer
{
        long mtype;
        char mtext[msgsz];
}
mbuffer;
main( )
{
        key_t key;
        int msqid,buffer;
        mbuffer mb;
        key=1234;
        if((msqid=msgget(key,0666))<0)
        {
                perror("msgget");
                return 0;



}

if(msgrcv(msqid,&mb,msgsz,1,0)<0)
        {


perror("msgrev");
                return 1;
        }
        printf(" MESSAGE : %s IS RECEIVED\n",mb.mtext);
        return 0;
}
```

<u>OUTPUT:</u>

INFERENCE:

RESULT:

Thus the program to implement interprocess communication using shared memory is executed and verified successfully.

Ex. No: 8	BANKER'S ALGORITHM DEADLOCK AVOIDANCE

AIM:

To write a program to implement deadlock avoidance using Banker's Algorithm.

ALGORITHM:

Safety algorithm

1. Start
2. Initialize a temporary vector W (Work) to equal the available vector A.
3. Find an index i (row i) such that

   Need i < = W

   If no such row exists, the system will deadlock, since no process can run to completion.

4. If such a row is found, mark this process as finished, and add all its resources to W Vector i,

   W = W + Ci

   Go to step 2, until either all processes are marked terminated (in this case initial state is safe), or until a deadlock occurs, in which the state is not safe.

Resource – request algorithm

1. If Request i < = Needi , go to step 2. Otherwise, error
2. If Request i < = Available, go to step 3. Otherwise, Pi must wait, since resources are not available
3. Modify the state (the system pretend to have allocated the requested resources to process

   Pi) Available = Available - Requesti

   Allocationi = Allocationi +

   Requesti Needi = Needi -

   Requesti

4. If the resulting state is safe, the transaction is completed and process Pi is allocated its resources. If new state is unsafe, then Pi must wait for Requesti and the old state is restored.
5. Stop the program.

PROGRAM:

```
#include<stdio.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 Avoidance ************\n");
        input( );
        show( );
        cal( );
        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;
		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("p%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:

INFERENCE:

RESULT:

Thus the program to implement deadlock avoidance using Banker’s Algorithm is implemented and executed successfully.

Ex. No: 9	DEADLOCK DETECTION

AIM:

To write a program to implement deadlock detection.

ALGORITHM 1: Simply detects the existence of a Cycle:
Step1: Start at any vertex finds all its immediate neighbors.
Step2: From each of these find all immediate neighbors, etc.
Step3: Until a vertex repeats (there is a cycle) or one cannot continue (there is no cycle)
Step4: Stop.

ALGORITHM 2: On a copy of the graph:
Step1: See if any Processes NEEDs can all be satisfied.
Step2: If so satisfy the needs with holds and remove that Process and all the Resources it holds from the graph.
Step3: If any Process are left Repeat step a
Step4: If all Processes are finally removed by this procedure there is no Deadlock in the original graph, if not there is.
Step5: Stop.

PROGRAM:
```
#include<stdio.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 Algorithm ************\n");
        input( );
        show( );
        cal( );
        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;
        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("p%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:

INFERENCE:

RESULT:

Thus the program to implement deadlock detection is implemented and executed successfully.

Ex. No: 10	**THREADING AND SYNCHRONIZATION**

AIM:

To implement Banking system involving Concurrency (Thread)

ALGORITHM:

Step 1: In Client side creating threads which runs a method to receive files:

```
int k;
for(k=0; k< fNameCounter; k++)
{
    pthread_t thread_id;
int status = pthread_create(&thread_id,NULL,&receiveFile. fName);
if(status !=0)
{
printf("Thread Creation Failed \n");
exit(0);
}
}
```

Step 2:In server side

```
int k;
for(k=0; k< fNameCounter; k++)
{
int status = pthread_create(&thread_id,NULL,&sendFile. fName);
if(status !=0)
{
printf("Thread Creation Failed \n");
exit(0);
}
}
```

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* doSomeThing(void *arg)
```

```
{           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);
             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));
                        i++;      }
             pthread_join(tid[0], NULL);
            pthread_join(tid[1], NULL);
             return 0;
}
```

OUTPUT:

INFERENCE:

RESULT:

Thus the program to implement threading and synchronization is implemented and executed successfully.

EX.NO:11	MEMORY ALLOCATION METHODS

AIM:

To create a simulator for memory management algorithm.(First Fit, Best Fit, Worst Fit)

ALGORITHM

1) FIRST FIT

Allocates the first hole that is big enough. Searching can be started either at the beginning or where the previous first fit search end read.

1.Start the process.
2.Declare the size.
3.Get the number of processes to be inserted.
4.Allocate the first hole that is big enough searching.
5.Start at the beginning of the set of holes.
6.If not start at the hole that is sharing the pervious first fit search end.
7.If large enough then stop searching in the procedure.
8.Display the values.
9.Stop the process.

2) BEST FIT

Best Fit allocates the smallest hole that is big enough. We must search the entire list, unless the list is kept ordered by size.

1. Start the program.
2. Declare the size.
3. Get the number of processes to be inserted.
4. Allocate the best hole that is small enough searching.
5. Start at the best of the set of holes.
6. If not start at the hole that is sharing the previous best fit search end.
7. Compare the hole in the list.
8. If small enough then stop searching in the procedure.
9. Display the values.
10. Stop the program.

3) WORST FIT

Allocates to the largest hole.

1. Start the program.
2. Declare the size.
3. Get the number of processes to be inserted.

4. Allocate the first hole that is small enough searching.

5. If small enough then stop searching in the procedure.
6. Display the values.
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
main( )
{
        int p[10],np,b[10],nb,ch,c[10],d[10],alloc[10],flag[10],i,j;
        printf("\nEnter the no of process:");
        scanf("%d",&np);
        printf("\nEnter the no of blocks:");
        scanf("%d",&nb);
        printf("\nEnter the size of each process:");
        for(i=0;i<np;i++)
                printf("\nProcess %d:",i);
                scanf("%d",&p[i]);
        printf("\nEnter the block sizes:");
        for(j=0;j<nb;j++)
                printf("\nBlock %d:",j);
                scanf("%d",&b[j]);c[j]=b[j];d[j]=b[j];
        if(np<=nb)
        {
                printf("\n1.First fit 2.Best fit 3.Worst fit");
                do
                {
                        printf("\nEnter your choice:");
                        scanf("%d",&ch);
                        switch(ch)
                        {
                                case 1: printf("\nFirst Fit\n");
                                        for(i=0;i<np;i++)
                                        {
                                                for(j=0;j<nb;j++)
                                                {
                                                        if(p[i]<=b[j])
                                                        {
```

```
      alloc[j]=p[i];printf("\n\nAlloc[%d]",alloc[j]);
       printf("\n\nProcess %d of size %d is allocated in block:%d of size:%d",i,p[i],j,b[j]);
flag[i]=0,b[j]=0;break;
                                                    }
                                                    else
                                                            flag[i]=1;
                                            }
                                    }
                                    for(i=0;i<np;i++)
                                    {
                                            if(flag[i]!=0)
                            printf("\n\nProcess %d of size %d is not allocated",i,p[i]);
                                    }
                                    break;
                            case 2: printf("\nBest Fit\n");
                            for(i=0;i<nb;i++)
                                    {
                                            for(j=i+1;j<nb;j++)
                                            {
                                                    if(c[i]>c[j])
                                                    {
                                                            int temp=c[i];c[i]=c[j];c[j]=temp;
                                                    }
                                            }
                                     }
                                    printf("\nAfter sorting block sizes:");
                                    for(i=0;i<nb;i++)
                                            printf("\nBlock %d:%d",i,c[i]);
                                    for(i=0;i<np;i++)
                                    {
                                            for(j=0;j<nb;j++)
                                            {
                                                    if(p[i]<=c[j])
                                                    {

      alloc[j]=p[i];printf("\n\nAlloc[%d]",alloc[j]);
       printf("\n\nProcess %d of size %d is allocated in block %d of size %d",i,p[i],j,c[j]);
                    flag[i]=0,c[j]=0;break;
```

```
                }
                                else
                                        flag[i]=1;
                        }
                }
                for(i=0;i<np;i++)
                {
                        if(flag[i]!=0)
        printf("\n\nProcess %d of size %d is not allocated",i,p[i]);
                }
                break;
        case 3: printf("\nWorst Fit\n");
                for(i=0;i<nb;i++)
                {
                        for(j=i+1;j<nb;j++)
                        {
                                if(d[i]<d[j])
                                {
                                        int temp=d[i];d[i]=d[j];d[j]=temp;
                                }
                        }
                }
                printf("\nAfter sorting block sizes:");
                for(i=0;i<nb;i++)
                        printf("\nBlock %d:%d",i,d[i]);
                for(i=0;i<np;i++)
                {
                        for(j=0;j<nb;j++)
                        {
                                if(p[i]<=d[j])
                                {
                              alloc[j]=p[i];printf("\n\nAlloc[%d]",alloc[j]);
printf("\n\nProcess %d of size %d is allocated in block %d of size %d",i,p[i],j,d[j]);
                                        flag[i]=0,d[j]=0;break;
                                }
                                else
                                        flag[i]=1;
                        }
        }
                                        for(i=0;i<np;i++)
                                        {
                                              if(flag[i]!=0)
                              printf("\n\nProcess %d of size %d is not allocated",i,p[i]);
                                        }
                                        break;
                                  default:printf("Invalid Choice…!");break;
                              }           }
                  while(ch<=3);
        } }
```

OUTPUT:

INFERENCE:

RESULT:

Thus the implementation of memory allocation methods for fixed partition is executed successfully.

Ex. No: 12	PAGING TECHNIQUE

AIM:

To implement memory allocation with pages

ALGORITHM:

Step1: Read all the necessary input from the keyboard.

Step2: Pages - Logical memory is broken into fixed - sized blocks.

Step3: Frames – Physical memory is broken into fixed – sized blocks.

Step4: Calculate the physical address using the following

Physical address = (Frame number * Frame size) + offset

Step5: Display the physical address.

Step6: Stop the process.

PROGRAM:

```
#include<stdio.h>
int main( )
{
        int ms, ps, nop, np, rempages, i, j, x, y,pa, offset;
        int s[10], fno[10][20];
        printf("\nEnter the memory size --");
        scanf("%d",&ms);
        printf("\nEnter the page size --");
        scanf("%d",&ps);
        nop = ms/ps;
        printf("\nThe no. of pages available in memory are --%d ",nop);
        printf("\nEnter number of processes --");
        scanf("%d",&np);
        rempages = nop;
        for(i=1;i<=np;i++)
        {
                printf("\nEnter no. of pages required for p[%d]--",i);
                scanf("%d",&s[i]);
                if(s[i] >rempages)
                {
                        printf("\nMemory is Full");
break;
                    }
                    rempages = rempages -s[i];
                    printf("\nEnter pagetable for p[%d] ---",i);
                    for(j=0;j<s[i];j++)
                              scanf("%d",&fno[i][j]);            }
          printf("\nEnter Logical Address to find Physical Address ");
          printf("\nEnter process no. and pagenumber and offset --");
          scanf("%d %d %d",&x,&y, &offset);
          if(x>np || y>=s[i] || offset>=ps)
                    printf("\nInvalid Process or Page Number or offset");
          else
          {
                    pa=fno[x][y]*ps+offset;
                    printf("\nThe Physical Address is --%d",pa);                 }
          return (0);        }
```

OUTPUT:

INFERENCE:

RESULT:

Thus the program to implement paging concept is implemented and executed successfully.

Ex. No: 13	PAGE REPLACEMENT ALGORITHM

AIM

To write a c program to implement page replacement algorithms are

a. FIFO page replacement algorithm
b. LRU page replacement algorithm
c. LFU page replacement algorithm

FIFO page replacement algorithm

ALGORITHM:

```
Step1:Start the process
Step2:Read number of pages n
Step3:Read number of pages no
Step4:Read page numbers into an array a[i]
Step5:Initialize avail[i]=0 .to check page hit
Step6: Replace the page with circular queue, while re-placeing check page availability in the  frame
Place avail[i]=1 if page is placed in the frame Count page faults.
Step7: Print the results.
Step8: Stop the process
```

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;
}
```

OUTPUT:

b. LRU page replacement

algorithm ALGORITHM:

ALGORITHM :

```
Step1:Start the process
Step2:Declare the size
Step3:Get the number of pages to be inserted
Step4:Get the value
Step5:Declare counter and stack
Step6:Select the least recently used page by counter value
Step7:Stack them according the selection.
Step8:Display the values
Step9. Stop the process
```

PROGRAM:

```
#include<stdio.h>
int findLRU(int time[], int n)
```

```
{
        int i, minimum = time[0], pos = 0;
        for(i = 1; i < n; ++i)
        {
                if(time[i] < minimum)
                {
                        minimum = time[i];
                        pos = i;
                }
        }


        return pos;

}
int main( )
{
        int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,
pos, faults = 0;
        printf("Enter number of frames: ");
        scanf("%d", &no_of_frames);
        printf("Enter number of pages: ");
        scanf("%d", &no_of_pages);
        printf("Enter reference string: ");
        for(i = 0; i < no_of_pages; ++i)
                scanf("%d", &pages[i]);
        for(i = 0; i < no_of_frames; ++i)
                frames[i] = -1;
        for(i = 0; i < no_of_pages; ++i)
        {
                flag1 = flag2 = 0;
                for(j = 0; j < no_of_frames; ++j)
                {
                        if(frames[j] == pages[i])
                        {
                                counter++;
                                time[j] = counter;
                                flag1 = flag2 = 1;
                                break;
                        }
                }
                if(flag1 == 0)
                {
                        for(j = 0; j < no_of_frames; ++j)
                        {
                                if(frames[j] == -1)
                                {
                                        counter++;

                                        faults++;
                                        frames[j] = pages[i];
                                        time[j] = counter;
                                        flag2 = 1;
                                        break;
                                }
                        }
                }
                if(flag2 == 0)
                {
                        pos = findLRU(time, no_of_frames);
                        counter++;
                        faults++;
                        frames[pos] = pages[i];
                        time[pos] = counter;
                }
                printf("\n");
                for(j = 0; j < no_of_frames; ++j)
                        printf("%d\t", frames[j]);

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

OUTPUT:

c. LFU page replacement algorithm

ALGORITHM:

Step1:Start Program

Step2:Read Number Of Pages And Frames

Step3:Read Each Page Value

Step4:Search For Page In The Frames

Step5:If Not Available Allocate Free Frame

Step6:If No Frames Is Free Replace The Page With The Page that is leastly Used

Step7:Print Page Number Of Page Faults

Step8:Stop process.

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;
        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 THE NO.OF PAGE FAULTS ARE:%d",pf);

printf("\n PAGE FAULT RATE:%f",pfr);
}
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;
}
```

OUTPUT:

INFERENCE:

RESULT:

Thus the program to implement Page Replacement algorithm is implemented and executed successfully.

Ex. No: 14	FILE ORGANIZATION TECHNIQUES

AIM:

To implement File Organization Structures in C are

a. Single Level Directory
b. Two-Level Directory
c. Hierarchical Directory Structure
d. Directed Acyclic Graph Structure

a. Single Level

ALGORITHM:

```
Step 1:Start
Step 2: Initialize values gd=DETECT,gm,count,i,j,mid,cir_x; Initialize character array fname[10][20];
Step 3: Initialize graph function as Initgraph(& gd, &gm," c:/tc/bgi"); Clear device( );
Step 4:set back ground color with setbkcolor( );
Step 5:read number of files in variable count.
Step 6:if check i<count
Step 7: for i=0 & i<count i increment;
Cleardevice( ); setbkcolor(GREEN); read file name;
setfillstyle(1,MAGENTA);
Step 8: mid=640/count;
cir_x=mid/3; bar3d(270,100,370,150,0,0); settextstyle(2,0,4); settextstyle(1,1);
outtextxy(320,125,"rootdirectory"); setcolor(BLUE);
i++;
Step 9:for j=0&&j<=i&&cir_x+=mid j increment; line(320,150,cir_x,250);
        fillellipse(cir_x,250,30,30); outtextxy(cir_x,250,fname[i]);
Step 10: End
```

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main( )
```

```
{
int i,ch;
char f[30];
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice --
");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
case 5: exit(0);
break
default:
printf("\nterminate");
}
}
}
```

<u>OUTPUT</u>

b. Two-level directory Structure

ALGORITHM:

Step 1:Start

Step 2:Initialize structure elements

```
struct tree_ element{ char name[20];

   Initialize integer variables x, y, ftype, lx, rx, nc, level;
            struct tree_element *link[5];}
            typedef structure tree_element node;
Step 3:start main function
Step 4: Step variables gd=DETECT,gm; node *root;
root=NULL;
Step 5:create structure using create(&root,0,"null",0,639,320);
Step 6:initgraph(&gd, &gm,"c:\tc\bgi"); display(root);
closegraph( );
Step 7:end main function
Step 8:Initialize variables i,gap;
Step 9:if check *root==NULL (*root)=(node*)malloc(sizeof(node));
enter name of ir file name in dname; fflush(stdin);
gets((*root)->name);
Step 10 if check lev==0||lev==1 (*root)->ftype=1; else(*root)->ftype=2;
 (*root)->level=lev; (*root)->y=50+lev*5; (*root)->x=x;
(*root)->lx=lx; (*root)->rx=rx;
Step 11:for i=0&&i<5 increment i
(*root)->link[i]=NULL;
if check (*root)->ftype==1
Step 12: if check (lev==0||lev==1) if check(*root)->level==0 print "how many users"
else print "how many files" print (*root)->name
read (*root)->nc
Step 13:Then (*root)->nc=0; if check(*root)->nc==0 gap=rx-lx;
else gap=(rx-lx)/(*root)->nc;
Step 14:for i=0&&i<(*root)->nc increment i;
create(&((*root)->link[i]),lev+1,(*root)->name, lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
then
(*root)->nc=0;
Step 15: Initialize e display function Initialize i
set textstyle(2,0,4); set textjustify(1,1); set fillstyle(1,BLUE);
setcolor(14); step 13:if check root!=NULL
Step 16:for i=0&&i<root->nc increment i
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
Step 17: if check root->ftype==1
bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
```

```
else fill ellipse(root->x,root->y,20,20);
out textxy(root->x,root->y,root->name);
Step 18:for i=0&&i<root->nc increment i display(root->link[i]);
Step 19:End
```

PROGRAM:

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main( )
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr( );
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
```

```
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
        printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
```

```
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
    printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
getch( );
}
```

OUTPUT:

c. Hierarchical directory Structure
ALGORITHM:

Step 1:Start

Step 2: define structure and declare structure variables

Step 3: start main and declare variables

Node *root Root = NULL

Step 4: create root null Initgraph &gd,&gm Display root

Step 5:create a directory tree structure If check *root==NULL

```
Display dir/file mane
Step 6: gets *root->name
*root-> level=lev *root->y=50+lev*50 *root->x=x
*root->lx=lx *root->rx = rx
Step7: for i=0 to i<5
Root->link[i]=NULL Display sub dir/ files
Step8: if check *root->nc==0 Gap=rx-lx
Then
Gap =rx-lx/*root->nc
Step9: for i=0 to i<*root->nc
Then     *rot->nc=0
Step10: display the directory tree in graphical mood Display nood *root
If check rooy !=NULL
Step 11: foe i=0 to i<root->nc
Line of root->x, root->y, root->link[i]->x, root->link[i]-y
Step12: if check root->ftype==1
Bard of root->x-20, root->y-10,root->x+20,root->y+10,0 Then
Display root->link[i]
Step13: End
```

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,"..\\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 name of dir/file(under%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;
}
return;
}
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]);
}
}
return;
}
```

OUTPUT:

Directed Acyclic Graph (DAG)

ALGORITHM:

```
Step 1:Start
Step 2: declare structure and structure variables
Step 3: start main and declare variables Node *root
Root = NULL Create root null
Read the shared file information
Step 4: for i=0; to i<no.f1
Draw link lines for shared files Search root find the first and second
Step 5: if check root !-NULL
If check string comparing root ->name, s==0 Then
For i=0 to i<root->nc search root->link
Step 6: creates a directory tree structure If check *root==NULL
Display directory mane or file name
Step7: if check *root ->nc==0 Gap=rx-lx
Then
Gap=rx-lx/ (root ->nc
Step 8: for i=0 to i<*root-> link[i] Then *root->nc=0
Step9: display the directory tree in graphical mode Display node *root
Step10: if check root !=NULL
For i=0 to i<root->nc draw lines
Step 11: line of root->x,root->y,root->link[i]->x, root->link[i]->y If check root->ftype==1 if it is
directory
Step12: then root->x,root->y,20,20 if it a file Root->x,root->y,root->name
Step13: for 0 to i<root->nc display children Display root->link[i]
Step14: Stop process.
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
structtree_element
{
char name[20];
```

```
intx,y,ftype,lx,rx,nc,level; structtree_element *link[5]; };
typedefstructtree_element node; typedefstruct
{
char from[20]; char to[20]; }link;
link L[10]; intnofl; node * root; void main( )
{
intgd=DETECT,gm; root=NULL; clrscr( );
create(&root,0,"root",0,639,320); read_links( );
clrscr( );
initgraph(&gd,&gm,"c:\\tc\\BGI");
draw_link_lines( );
display(root);
getch( );
closegraph( );
}
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);
}
}


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);
}
}
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);
}
}
}
create(node **root,intlev,char *dname,intlx,intrx,int x)
{
inti,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 */ 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

INFERENCE:

RESULT:

Thus various file organization techniques is executed and verified successfully.

Ex. No: 15	FILE ALLOCATION STRATEGIES

AIM:

To implement File allocation in C are

a. Sequential File Allocation
b. Indexed File Allocation
c. Linked File Allocation

a. Sequential File Allocation:

ALGORITHM:

```
Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
  Step 4: Allocate the required locations to each in sequential order
a) Randomly select a location from available location
s1= random(100);
b) Check whether the required locations are free from the selected location. if(b[s1].flag==0)
{
for(j=s1;j<s1+p[i];j++) {
if((b[j].flag)==0) count++;
} if(count==p[i]) break;
}
c) Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)
{
k[i][j]=s; j=j+1; b[s].bno=s; b[s].flag=1;
}
Step 5: Print the results fileno, length ,Blocks allocated. Step 6: Stop the program
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
    int st[20],b[20],b1[20],ch,i,j,n,blocks[20][20],sz[20];
    char F[20][20],S[20];
    clrscr( );
    printf("\n Enter no. of Files ::");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter file %d name ::",i+1);

        scanf("%s",&F[i]);
        printf("\n Enter file%d size(in kb)::",i+1);
        scanf("%d",&sz[i]);
        printf("\n Enter Starting block of %d::",i+1);
        scanf("%d",&st[i]);
        printf("\n Enter blocksize of File%d(in bytes)::",i+1);
        scanf("%d",&b[i]);
    }
    for(i=0;i<n;i++)
        b1[i]=(sz[i]*1024)/b[i];
    for(i=0;i<n;i++)
    {
        for(j=0;j<b1[i];j++)
            blocks[i][j]=st[i]+j;
    }
    do
    {
        printf("\nEnter the Filename ::");
        scanf("%s",S);
        for(i=0;i<n;i++)
        {
            if(strcmp(S,F[i])==0)
            {
                printf("\nFname\tStart\tNblocks\tBlocks\n");
                printf("\n---------------------------------------------\n");
                printf("\n%s\t%d\t%d\t",F[i],st[i],b1[i]);
                for(j=0;j<b1[i];j++)
                    printf("%d->",blocks[i][j]);
            }

        }
        printf("\n---------------------------------------------\n");
        printf("\nDo U want to continue ::(Y:n)");
        scanf("%d",&ch);
        if(ch!=1)
            break;
    }while(1);
}
```

OUTPUT:

b. Indexed File Allocation:

ALGORITHM:

```
Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
Step 4: Allocate the required locations by selecting a location randomly q= random(100);
a) Check whether the selected location is free .
b) If the location is free allocate and set flag=1 to the allocated locations. q=random(100);
{
if(b[q].flag==0) b[q].flag=1; b[q].fno=j; r[i][j]=q;
Step 5: Print the results fileno, length ,Blocks allocated.
Step 6: Stop the program
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int n;
void main( )
{
   int b[20],b1[20],i,j,blocks[20][20],sz[20];
   char F[20][20],S[20],ch;
   clrscr( );
   printf("\n Enter no. of Files ::");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\n Enter file %d name ::",i+1);
      scanf("%s",&F[i]);
      printf("\n Enter file%d size(in kb)::",i+1);
      scanf("%d",&sz[i]);
      printf("\n Enter blocksize of File%d(in bytes)::",i+1);
      scanf("%d",&b[i]);
   }
   for(i=0;i<n;i++)
   {
      b1[i]=(sz[i]*1024)/b[i];
      printf("\n\nEnter blocks for file%d",i+1);
      for(j=0;j<b1[i];j++)
```

```
        {
           printf("\n Enter the %dblock ::",j+1);
           scanf("%d",&blocks[i][j]);
        }   }
    do
    {
        printf("\nEnter the Filename ::");
        scanf("%s",&S);
        for(i=0;i<n;i++)
        {
           if(strcmp(F[i],S)==0)
           {
               printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
               printf("\n---------------------------------------------\n");
               printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);
               for(j=0;j<b1[i];j++)
                   printf("%d->",blocks[i][j]);
           }       }
        printf("\n---------------------------------------------\n");
        printf("\nDo U want to continue ::(Y:n)");
        scanf("%d",&ch);
    }while(ch!=0);
}
```

<u>OUTPUT:</u>

c. Linked File Allocation

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
Step 4: Allocate the required locations by selecting a location randomly q= random(100);
a) Check whether the selected location is free .
b) If the location is free allocate and set flag=1 to the allocated locations. While allocating next location address to attach it to previous location

```
for(i=0;i<n;i++) {
for(j=0;j<s[i];j++) {
q=random(100); if(b[q].flag==0) b[q].flag=1; b[q].fno=j; r[i][j]=q;
if(j>0) {
p=r[i][j-1]; b[p].next=q; }
} }
```

Step 5: Print the results fileno, length ,Blocks allocated.
Step 6: Stop the program

PROGRAM:

```
#include<stdio.h>
void main( )
{
char a[10];
            int i,sb,eb,fb1[10];
printf("\n enter the file name:");
scanf("%s",a);
 printf("\n Enter the starting block:");
scanf("%d",&sb);
 printf("Enter the ending Block:");
scanf("%d",&eb);
for(i=0;i<5;i++)
{
printf("Enter the free block %d",i+1);
 scanf("%d",&fb1[i]);    }
printf("\n File name \t Starting block \t Ending block \n");
printf("%s \t\t %d\t\t %d",a,sb,eb);
printf("\n %s File Utilization of Linked type of following blocks:",a);
printf("\n %d->",sb);
for(i=0;i<5;i++)
{
```

```
printf("%d->",fb1[i]);
 }
printf("%d\n",eb);
}
```

OUTPUT:

INFERENCE:

RESULT:

Thus the program to implement File allocation in C is executed successfully.

{

CONTENT BEYOND THE SYLLABUS

Ex. No: 16	**VIRTUALIZATION**

AIM:

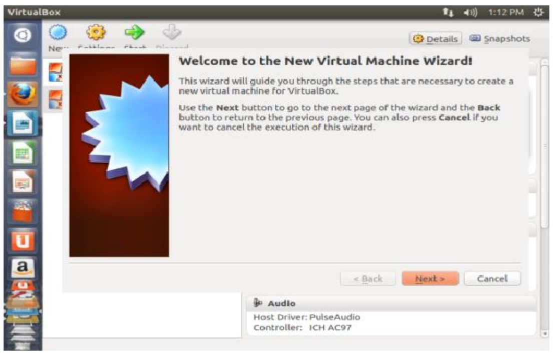
To implement Virtualization concept using Virtual box.

PROCEDURE:

```
cse@ubuntu-WIV37205-0125:~$ sudo apt-get install virtual box
[sudo] password for oslab:
Reading package lists... Done
Building dependency tree
Reading state information... Done
virtualbox is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 61 not upgraded.
```

ALGORITHM :

Step 1: open virtualbox and click next.
Step 2: enter the name and click next
Step 3: in the memory dialog box, click next.
Step 4: select vmdk and click next.
Step 5: select start-up disk->create new hard disk and click next.
Step 6: select dynamically allocated storage and click next.
Step 7: in virtual disk file browse the location by selecting 26gb volume and click save.
Step 8: the path of the location is shown and click next.
Step 9: a summary dialog box opens and click create.
Step 10: click next to run the virtual machine.
Step 11: installation of windows starts.
Step 12: set the name and organization.
Step 13: set the administration password.
Step 14: Virtual OS starts.


VirtualBox 1:12 PM

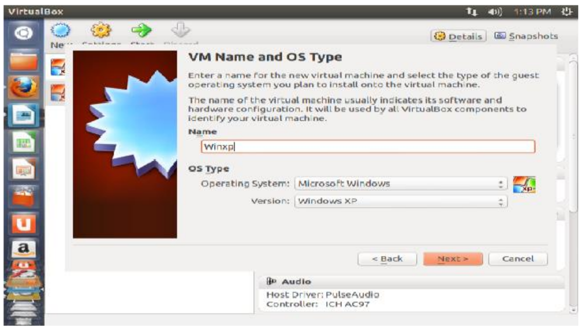
Welcome to the New Virtual Machine Wizard!

This wizard will guide you through the steps that are necessary to create a new virtual machine for VirtualBox.

Use the **Next** button to go to the next page of the wizard and the **Back** button to return to the previous page. You can also press **Cancel** if you want to cancel the execution of this wizard.

< Back Next > Cancel

Audio
Host Driver: PulseAudio
Controller: ICH AC97


VirtualBox 1:13 PM

VM Name and OS Type

Enter a name for the new virtual machine and select the type of the guest operating system you plan to install onto the virtual machine.

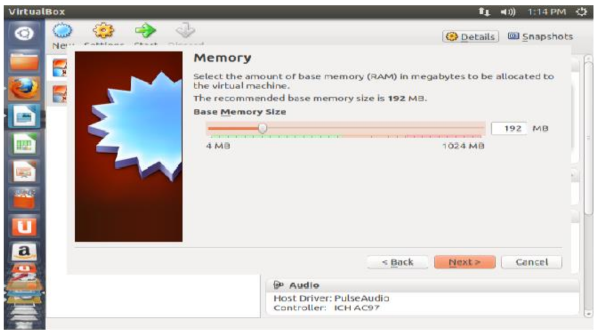
The name of the virtual machine usually indicates its software and hardware configuration. It will be used by all VirtualBox components to identify your virtual machine.

Name: Winxp

OS Type
Operating System: Microsoft Windows
Version: Windows XP

< Back Next > Cancel

Audio
Host Driver: PulseAudio
Controller: ICH AC97


VirtualBox 1:14 PM

Memory

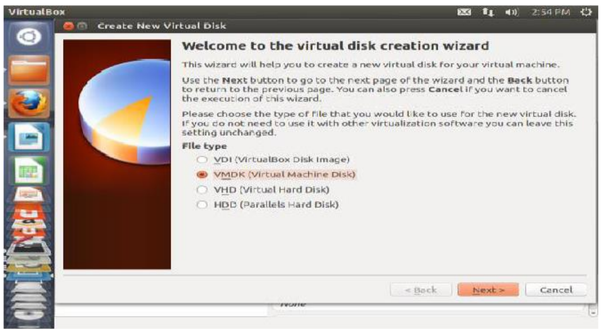
Select the amount of base memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended base memory size is **192** MB.

Base Memory Size: 192 MB (4 MB – 1024 MB)

< Back Next > Cancel

Audio
Host Driver: PulseAudio
Controller: ICH AC97


VirtualBox 2:54 PM — Create New Virtual Disk

Welcome to the virtual disk creation wizard

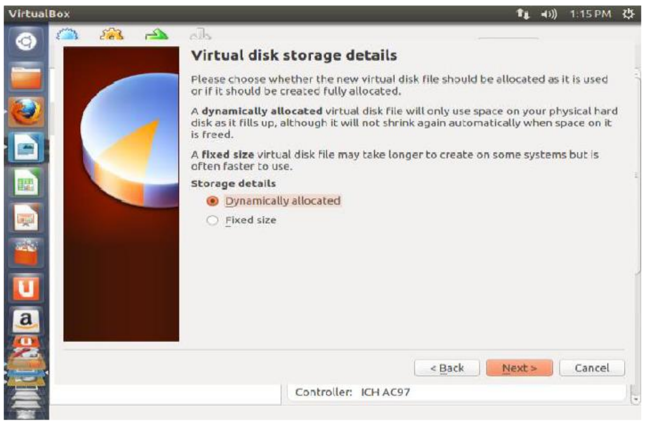
This wizard will help you to create a new virtual disk for your virtual machine.

Use the **Next** button to go to the next page of the wizard and the **Back** button to return to the previous page. You can also press **Cancel** if you want to cancel the execution of this wizard.

Please choose the type of file that you would like to use for the new virtual disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

File type
- VDI (VirtualBox Disk Image)
- [✓ VMDK (Virtual Machine Disk)]
- VHD (Virtual Hard Disk)
- HDD (Parallels Hard Disk)

< Back Next > Cancel


VirtualBox 1:15 PM

Virtual disk storage details

Please choose whether the new virtual disk file should be allocated as it is used or if it should be created fully allocated.

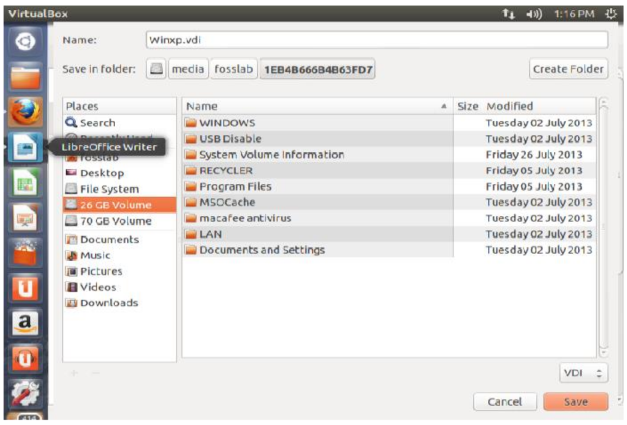
A **dynamically allocated** virtual disk file will only use space on your physical hard disk as it fills up, although it will not shrink again automatically when space on it is freed.

A **fixed size** virtual disk file may take longer to create on some systems but is often faster to use.

Storage details
- [✓ Dynamically allocated]
- Fixed size

< Back Next > Cancel

Controller: ICH AC97


VirtualBox 1:16 PM

Name: Winxp.vdi

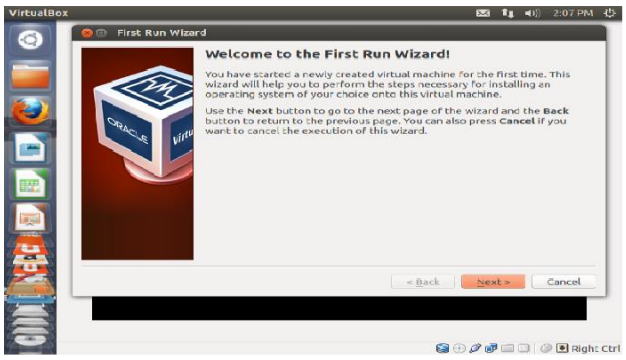
Save in folder: media | fosslab | 1EB4B666B4B63FD7 — Create Folder

Places	Name	Size	Modified
Search	WINDOWS		Tuesday 02 July 2013
Recently Used	USB Disable		Tuesday 02 July 2013
fosslab	System Volume Information		Friday 26 July 2013
Desktop	RECYCLER		Friday 05 July 2013
File System	Program Files		Friday 05 July 2013
26 GB Volume	MSOCache		Tuesday 02 July 2013
70 GB Volume	macafee antivirus		Tuesday 02 July 2013
Documents	LAN		Tuesday 02 July 2013
Music	Documents and Settings		Tuesday 02 July 2013
Pictures			
Videos			
Downloads			

LibreOffice Writer

VDI

Cancel Save


VirtualBox 2:07 PM — First Run Wizard

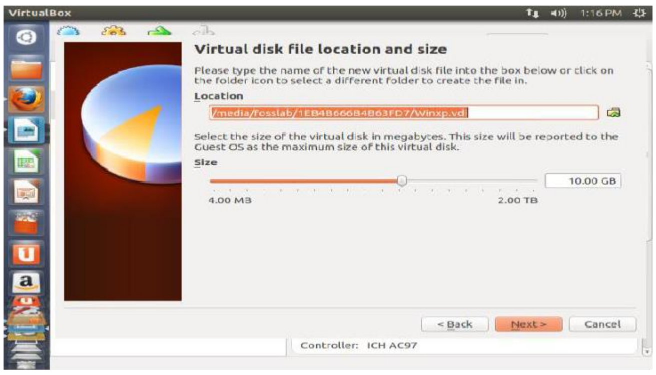
Welcome to the First Run Wizard!

You have started a newly created virtual machine for the first time. This wizard will help you to perform the steps necessary for installing an operating system of your choice onto this virtual machine.

Use the **Next** button to go to the next page of the wizard and the **Back** button to return to the previous page. You can also press **Cancel** if you want to cancel the execution of this wizard.

< Back Next > Cancel

Right Ctrl


VirtualBox 1:16 PM

Virtual disk file location and size

Please type the name of the new virtual disk file into the box below or click on the folder icon to select a different folder to create the file in.

Location: /media/fosslab/1EB4B666B4B63FD7/Winxp.vdi

Select the size of the virtual disk in megabytes. This size will be reported to the Guest OS as the maximum size of this virtual disk.

Size: 10.00 GB (4.00 MB – 2.00 TB)

< Back Next > Cancel

Controller: ICH AC97

INFERENCE:

RESULT:

Thus the program to implement Virtualization concept using Virtual box is done and executed successfully.

Ex. No: 17	KERNEL CONFIGURATION

AIM :

To download / access The latest kernel source code from kernel.org,compile the kernel and install it in the local system and to Try viewing the source code of the kernel.

PROCEDURE

TO KNOW THE CURRENT VERSION OF THE KERNEL:

```
cse@ubuntu -desktop:~$ uname -r
 3.5.0-17-generic
```

DOWNLOAD THE LATEST LINUX KERNEL CODE AND EXTRACT IT IN THE SAME LOCATION:

http://www.berkes.ca/guides/linux_kernel.html

CHANGE THE DIRECTORY TO THE OSLAB:
TO LIST ALL THE FOLDERS AVAILABLE:

```
cse@ubuntu-desktop:~$ ls
3yr it.doc          k2.pl   k5.html~                 ppp        select.php~
3yr result.odt        k2.pl~      k.pl                 pro.php        Templates
build_systems         k2.py       k.pl~                           pro.php~   tmp
compfrmsrc            k2.py~      linux-3.2.17.tar.bz2          Public        Videos
Desktop             k3.html  linux-3.6.2.tar.bz2  sam1.php                 VirtualBox VMs
Documents             k3.html~ Music                         sam1.php~
Downloads             k4.php      Pictures                  sam.php
```

CHANGE THE DIRECTORY TO THE EXTRACTED LINUX FILE cse@ubuntu-desktop:~$ cd Downloads cse@ubuntu-desktop:~/Downloads$ cd linux-3.11-rc6

TO INSTALL LIBNCURSES5:
cse@ubuntu -desktop:~/Downloads$ cd linux-3.11-rc6 sudo apt-get install libncurses5-dev

TO COMPILE KERNEL REMOTELY:
cse@ubuntu -desktop:~/Downloads/linux-3.11-rc6$ makes menuconfig

TO UNCOMPRESS KERNEL TO HIGH MEMORY: cse@ubuntu -desktop:~/Downloads/linux-3.11-rc6$ makes bzImage make [1]:
Nothing to be done for `all'.
make [1]: Nothing to be done for `relocs'. CHK
include/generated/uapi/linux/version.h VDSOSYM
arch/x86/vdso/vdso32-int80-syms.lds GZIP
arch/x86/boot/compressed/vmlinux.bin.gz

```
BUILD   arch/x86/boot/bzImage
Setup is 16652 bytes (padded to 16896 bytes).
System is 5234 kB
CRC 88e6028e
Kernel: arch/x86/boot/bzImage is ready (#2)
```

TO MAKE KERNEL MODULES: cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ make modules make [1]:

```
Nothing to be done for `all'.
make [1]: Nothing to be done for `relocs'.
CHK     include/generated/uapi/linux/version.h
CHK     include/generated/utsrelease.h
CALL    scripts/checksyscalls.sh
LD [M]  sound/pci/echoaudio/snd-indigodjx.ko
 CC      sound/pci/echoaudio/snd-indigoio.mod.o
 IHEX firmware/cpia2/stv0672_vp4.bin IHEX
 firmware/yam/1200.bin
   IHEX   firmware/yam/9600.bin
```

TO INSTALL KERNEL MODULES:
cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ sudo make modules_install

```
[sudo] password for fosslab:
INSTALL arch/x86/crypto/ablk_helper.ko
INSTALL net/netfilter/xt_tcpudp.ko
INSTALL net/netfilter/xt_time.ko
INSTALL /lib/firmware/cpia2/stv0672_vp4.bin
INSTALL /lib/firmware/yam/1200.bin
INSTALL /lib/firmware/yam/9600.bin
DEPMOD 3.11.0-rc5
```

TO BOOT THE KERNEL:
cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ sudo cp arch/x86/boot/bzImage /boot/vmlinux-3.11-rc6 cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ sudo cp System.map /boot

TO INSTALL KERNEL:
cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ sudo make install

```
[sudo]password for fosslab:
Installing packages.
```

TO BOOT THE SPECIFIC KERNEL CONFIGURATION:
cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ sudo update -grub

```
[sudo]password for fosslab
updating packages
```

TO REBOOT THE KERNEL:
cse@ubuntu --desktop:~/Downloads/linux-3.11-rc6$ sudo reboot

```
The system is going down for reboot now.
```

TO CHECK THE NEWEST KERNEL VERSION:

```
cse@ubuntu --desktop:~$ uname -r 3.11.0-rc6
```

INFERENCE:

RESULT:

Thus to implement Kernel Configuration in Linux Environment is executed successfully.