V V COLLEGE OF ENGINEERING
(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)
V V Nagar, Arasoor, Tisaiyanvilai
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

College Vision and Mission Statement

Vision

"Emerge as a premier technical institution of global standards, producing enterprising, knowledgeable engineers and entrepreneurs."

Mission

- Impart quality and contemporary technical education for rural students.
- Have the state of the art infrastructure and equipment for quality learning.
- Enable knowledge with ethics, values and social responsibilities.
- Inculcate innovation and creativity among students for contribution to society.

Vision and Mission of the Department of Computer Science and Engineering

Vision

"Produce competent and intellectual computer science graduates by empowering them to compete globally towards professional excellence".

Mission

- Provide resources, environment and continuing learning processes for better exposure in latest and contemporary technologies in Computer Science and Engineering.
- Encourage creativity and innovation and the development of self-employment through knowledge and skills, for contribution to society
- Provide quality education in Computer Science and Engineering by creating a platform to enable coding, problem solving, design, development, testing and implementation of solutions for the benefit of society.

I. PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Graduates of Computer Science Engineering can

- Apply their technical competence in computer science to solve real world problems, with
  technical and people leadership.
- Conduct cutting edge research and develop solutions on problems of social relevance.
- Work in a business environment, exhibiting team skills, work ethics, adaptability and lifelong learning.

II. PROGRAM OUTCOMES (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

III. PROGRAM SPECIFIC OUTCOMES (PSOs)

The Students will be able to

- Exhibit design and programming skills to build and automate business solutions using cutting edge technologies.
- Strong theoretical foundation leading to excellence and excitement towards research, to provide elegant solutions to complex problems.
- Ability to work effectively with various engineering fields as a team to design, build and develop system applications.

V V COLLEGE OF ENGINEERING
(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)
V V Nagar, Arasoor, Tisaiyanvilai
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY

LIST OF EXPERIMENTS – R2021

PRACTICAL SUBJECT NAME	OPERATING SYSTEMS LABORATORY
PRACTICAL SUBJECT CODE	CS3461
SEMESTER/ YEAR	04 / SECOND
TOTAL HOURS	45
STAFF IN-CHARGE	Mrs. M. JANCYRANI MALLI
LAB INSTRUCTOR	Mrs. ANITHA
REGULATION	2021

CO1	Define and implement UNIX Commands.
CO2	Compare the performance of various CPU Scheduling Algorithms.
CO3	Compare and contrast various Memory Allocation Methods.
CO4	Define File Organization and File Allocation Strategies.
CO5	Implement various Disk Scheduling Algorithms.

V V COLLEGE OF ENGINEERING
(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)
V V Nagar, Arasoor, Tisaiyanvilai
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

S.No	Name of the Experiment	CO Mapping	PO Mapping
1	a) Basic UNIX Commands b) C programs to simulate UNIX commands like cp, ls, grep c) Simple Shell Programs	CO1	PO1-PO4, PSO1- PSO3
2	Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir	CO1	PO1-PO3, PSO1- PSO3
3	CPU Scheduling Algorithms a) First Come First Served (FCFS) Scheduling b) Shortest Job First (SJF) Scheduling c) Priority Scheduling d) Round Robin Scheduling	CO2	PO1-PO5, PSO1- PSO3
4	Inter Process Communication	CO2	PO1-PO4, PSO1- PSO3
5	Mutual Exclusion	CO2	PO1-PO5, PSO1-PSO3
6	Bankers Algorithm for Deadlock Avoidance	CO2	PO1-PO5, PSO1- PSO3
7	Algorithm for Deadlock Detection	CO2	PO1-PO5, PSO1- PSO3
8	Implementation of Threading	CO2	PO1-PO5, PSO1-PSO3
9	Implementation of Paging Technique	CO3	PO1-PO5, PSO1-PSO3
10	a) Implementation Of First Fit Memory Allocation Methods b) Implementation Of Worst Fit Memory Allocation Methods c) Implementation Of Best Fit Memory Allocation Methods	CO3	PO1-PO5, PSO1- PSO3
11	a) Implementation Of FIFO Page Replacement Algorithm	CO3	PO1-PO5, PSO1- PSO3

	b) Implementation Of LRU Page Replacement Algorithm c) Implementation Of Optimal Page Replacement Algorithm		
12	a) Implementation Of File Organization – Single Level Directory b) Implementation Of File Organization – Two Level Directory	CO4	PO1-PO5, PSO1- PSO3
13	a) Implementation Of Sequential File Allocation b) Implementation Of Linked File Allocation c) Implementation Of Indexed File Allocation	CO4	PO1-PO4, PSO1- PSO3
14	a) Implementation Of FCFS Disk Scheduling Algorithm b) Implementation Of SSTF Disk Scheduling Algorithm c) Implementation Of SCAN Disk Scheduling Algorithm d) Implementation Of C-SCAN Disk Scheduling Algorithm	CO5	PO1-PO5, PSO1- PSO3
15	Installation Of Windows Operating system	CO5	PO1-PO5, PSO1-PSO3
16	Installation Of Guest Operating System Using VMWARE	CO5	PO1-PO5, PSO1-PSO3

V V COLLEGE OF ENGINEERING

(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)

V V Nagar, Arasoor, Tisaiyanvilai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RUBRICS FOR ASSESSING LABORATORY

Sl. No.	Criteria	Total Marks	Excellent (25)	Good (20)	Average (10)	Poor (5)
			91% - 100%	71% - 90%	50% - 70%	<50%
1	Preparation	25	Gives clear idea about the aim and having good capability of executing experiments.	Capability of executing experiments but no proper clarification about the objective.	Gives clear idea about the target and has less capability of executing experiments.	Gives indistinct idea about the target and has less capability of executing experiments & who feel difficult to follow the objectives.
2	Viva	25	Have executed the experiments in an efficient way & make credible and unbiased judgments regarding the experiments.	Executed the experiments with less efficient & has partial judgments regarding the experiments.	Executed the experiments with less efficiency and has no judgments regarding experiments .	Incomplete experiments & lack of judgments regarding experiments.
3	Performance	25	Followed all the instructions given in the procedure and submitted the manual on time.	Followed all the instructions given in the procedure with some assisting.	Followed some of the instructions given in the procedure & late in submission of manual.	Unable to follow the instructions given in the procedure & late in submission of manual.

V V COLLEGE OF ENGINEERING

(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)

V V Nagar, Arasoor, Tisaiyanvilai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering		
Preparation	25	
Viva	25	
Performance	25	
Total	75	
Lab Incharge	Date	

INDEX

S.No	DATE	NAME OF THE EXPERIMENT	SIGN
1		a) Basic UNIX Commands b) C programs to simulate UNIX commands like cp, ls, grep c) Simple Shell Programs	
2		Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir	
3		CPU Scheduling Algorithms a) First Come First Served (FCFS) Scheduling b) Shortest Job First (SJF) Scheduling c) Priority Scheduling d) Round Robin Scheduling	
4		Inter Process Communication	
5		Mutual Exclusion	
6		Bankers Algorithm for Deadlock Avoidance	
7		Algorithm for Deadlock Detection	
8		Implementation of Threading	
9		Implementation of Paging Technique	
10		a) Implementation Of First Fit Memory Allocation Methods b) Implementation Of Worst Fit Memory Allocation Methods c) Implementation Of Best Fit Memory Allocation Methods	
11		a) Implementation Of FIFO Page Replacement Algorithm b) Implementation Of LRU Page Replacement Algorithm c) Implementation Of Optimal Page Replacement Algorithm	
12		a) Implementation Of File Organization –Single Level Directory b) Implementation Of File Organization –Two Level Directory	
13		a) Implementation Of Sequential File Allocation b) Implementation Of Linked File Allocation c) Implementation Of Indexed File Allocation	
14		a) Implementation Of FCFS Disk Scheduling Algorithm b) Implementation Of SSTF Disk Scheduling Algorithm c) Implementation Of SCAN Disk Scheduling Algorithm d) Implementation Of C-SCAN Disk Scheduling Algorithm	
15		Installation Of Windows Operating system	
16		Installation Of Guest Operating System Using VMWARE	

Ex. No. 1(a)

BASIC UNIX COMMANDS

AIM:

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

BASIC COMMANDS:

a) Display Commands

1) **Command :** date
Purpose : To check the date and time
Syntax : $date
Example : $date

2) **Command :** month
Purpose : To display only month
Syntax : $+%m
Example : $+%m

3) **Command :** Month Name
Purpose : To display month name
Syntax : $+%h
Example : $+%h

4) **Command :** Month Day
Purpose : To display day of month
Syntax : $+%d
Example : $+%d

5) **Command :** year
Purpose : To display last two digits of years
Syntax : $+%y
Example : $+%y

6) **Command :** hour
Purpose : To display hours
Syntax : $+%H
Example : $+%H

7) **Command :** Minutes
Purpose : To display minutes
Syntax : $+%M
Example : $+%M

8) **Command :** Seconds
Purpose : To display seconds
Syntax : $+%S
Example : $+%S

b) **Command :** cal
Purpose : To display the calendar
Syntax : $cal
Example : $cal

c) **Command :** echo
Purpose : To print the message on the screen.
Syntax : $echo "text"
Example : $echo HELLO

d) **Command :** ls
Purpose : To list the files. Your files are kept in a directory.
Syntax : $ls
Example : $ls

lsls–s All files (include files with prefix)

ls–l Lodetai (provide file statistics)

ls–t Order by creation time

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

ls–s Order by size

ls–r Reverse order

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

ls–s Show file size

ls– h " Human Readable", show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

e) **Command :** man
Purpose : To provide manual help on every UNIX commands.
Syntax : $man unix command
Example : $man cat

f) **Command :** who
Purpose : To displays data about all users who have logged into the

system currently.

Syntax : $who

Example : $ who -H. To show only hostname

$ who -m. To show active processes spawned by init

$ who -p. To show user's message status as

$ who -T. Show or list users logged in.

$ who -u. Show time of last system boot.

g) **Command :** whoami

Purpose : To displays about current user only

Syntax : $whoami

Example : $whoami

h) **Command :** uptime

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

Syntax : $uptime

Example : $uptime

i) **Command :** uname

Purpose : To displays the system information such as hardware platform, system name and processor, OS type.

Syntax : $uname–a

Example : $uname–a

j) **Command :** hostname

Purpose : To displays and set system host name

Syntax : $ hostname

Example : $ hostname

k) **Command :** bc

Purpose : To stands for "best calculator"

Syntax : $bc

Example : $bc

l) **Command :** id

Purpose : To display the login name.

Syntax : $id

Example : $id

m) **Command :** clear

Purpose : To clear the screen.

Syntax : $clear
Example : $clear

n) **Command :** finger
Purpose : To gathers and displays information about the users which includelogin name, name of user, home directory etc..
Syntax : $finger username
Example : $finger student

FILE MANIPULATION COMMANDS

a) **Command :** cat
Purpose : To create, view, and edit files.
Syntax : CREATE :$cat>filename

VIEW: $cat filename

EDIT $cat>>filename
Example : $ cat>aaa

cat aaa

cat>> aaa

b) **Command :** concatenate
Purpose : To add two file content into new file
Syntax : $cat file1file2>file3
Example : $cat aaa bbb>ccc

c) **Command :** grep
Purpose : To search a particular word or pattern related to that word from the file.
Syntax : $grep search word filename
Example : $grep anu student

d) **Command :** rm
Purpose : To deletes a file from the file system
Syntax : $rm filename
Example : $rm student

e) **Command :** touch
Purpose : To create a blank file.
Syntax : $touch filename
Example : $touch student

f) **Command :** cp
Purpose : To copies the files or directories
Syntax : $cp source file destination file
Example : $cp student stud

g) **Command :** mv
Purpose : To rename the file or directory
Syntax : $mv old file new file
Example : $$mv student stu

h) **Command :** cut
Purpose : To cuts or pickup a given number of character or fields of the file.
Syntax : $cut<option><filename>
Example : $cut –c filename (-c cutting columns)

$cut–c1-10emp

$cut–f 3,6emp (-f cutting fields)

$ cut –f 3-6 emp

i) **Command :** wc
Purpose : To counts the number of lines, words, character in a specified file(s) with the options as –l,-w,-c
Syntax : $wc filename
Example : $$wc student -l
$$wc student -w

$$wc student -c

DIRECTORY COMMANDS

a) **Command :** mkdir
Purpose : To create a directory
Syntax : $mkdir <directory name>
Example : $mkdir student

b) **Command :** rmdir
Purpose : To delete a directory
Syntax : $rmdir <directory name>
Example : $rmdir student

c) **Command :** cd
Purpose : To change the current directory
Syntax : $cd
Example : $cd ~ (changes path to your home directory)

cd (changes parent directory)

d) **Command :** pwd (Print working Directory)
Purpose : To display the absolute pathname of current working directory.
Syntax : $pwd
Example : **$**pwd

PROCESS COMMANDS

a) **Command :** exit
Purpose : To terminate a process
Syntax : $exit
Example : $exit

b) **Command :** kill
Purpose : To terminates or send a signal to process
Syntax : $kill
Example : $kill

c) **Command :** passwd
Purpose : To create or change a password
Syntax : $passwd
Example : $passwd

d) **Command :** semicolon (;)
Purpose : To execute more than one command at a time
Syntax : $;
Example : $who; date;

FILTER COMMANDS

a) **Command :** head
Purpose : To display lines from the head(top)of a given file
Syntax : $head filename
Example : $head student

$head -2student (To display the top two lines:)

b) **Command :** tail
Purpose : To display last 10 lines of the file
Syntax : $tail filename
Example : $tail student

$tail -2filename(To display the bottom two lines)

c) **Command :** chmod

Purpose : To change the permissions of a file or directory.

Syntax : $ch mod category operation permission file

(Category–is the user type, Operation–is used to assign or remove permission, Permission–is the type of permission, File–are used to assign or remove permission all)

Example : $ch modu+rw,g+rwstudent

Assigns read and write permission for users and groups

$chmodg=rwx student

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

$chmodu-wx student

Removes write and execute permission for users

RESULT:

Ex. No. 1(b)

C programs to simulate UNIX commands like cp, ls, grep

AIM:

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

a) PROGRAM FOR SIMULATION OF CP UNIX COMMANDS

ALGORITHM:

STEP 1: Start the program
STEP 2: Declare the variables ch, *fp, sc=0
STEP 3: Open the file in read mode
STEP 4: Get the character
STEP 5: If ch== " " then increment sc value by one
STEP 6: Print no of spaces
STEP 7: Close the file

PROGRAM:

```
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
main(int argc,char *argv[])
{
        FILE *fp;
        char ch;
        int sc=0;
        fp=fopen(argv[1],"r");
        if(fp==NULL)
        printf("unable to open a file",argv[1]);
        else
        {
                while(!feof(fp))
                {
                        ch=fgetc(fp);
                        if(ch==' ')
                        sc++;
                }
        printf("no of spaces %d",sc);
        printf("\n");
        fclose(fp);
        }
}
```

b) PROGRAM FOR SIMULATION OF LS UNIX COMMANDS

ALGORTIHM:

STEP 1 : Start the program
STEP 2 : Open the directory with directory object dp
STEP 3 : Read the directory content and print it.
STEP 4: Close the directory.

PROGRAM:

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

C) PROGRAM FOR SIMULATION OF GREP UNIX COMMANDS

ALGORITHM

STEP 1: Start the program
STEP 2: Declare the variables fline[max], count=0, occurrences=0 and pointers *fp,*newline.
STEP 3: Open the file in read mode.
STEP 4: In while loop check fgets(fline,max,fp)!=NULL
STEP 5: Increment count value.
STEP 6: Check newline=strchr(fline, „\n")
STEP 7: print the count,fline value and increment the occurrence value.
STEP 8: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define max 1024
void usage()
{
```

```
        printf("usage:\t. /a.out filename word \n ");
}
int main(int argc, char *argv[])
{
        FILE *fp;
        char fline[max];
        char *newline;
        int count=0;
        int occurrences=0;
        if(argc!=3)
        {
                usage();
                exit(1);
        }
        if(!(fp=fopen(argv[1],"r")))
        {
                printf("grep: couldnot open file : %s \n",argv[1]);
                exit(1);
        }
        while(fgets(fline,max,fp)!=NULL)
        {
                count++;
                if(newline=strchr(fline, „\n"))
                *newline="\0";
                if(strstr(fline,argv[2])!=NULL)
        {
                printf("%s: %d %s \n", argv[1],count, fline);
                occurrences++;
        }
}
}
```

RESULT:

Ex. No. 1(c)

SIMPLE SHELL PROGRAMS

AIM:

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

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

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of n.
STEP 3: Calculate „r=expr $n%2".
STEP 4: If the value of r equals 0 then print the number is even
STEP 5: If the value of r not equal to 0 then print the number is odd.

PROGRAM:

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

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

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of year.
STEP 3: Calculate „b=expr $y%4".
STEP 4: If the value of b equals 0 then print the year is a leap year
STEP 5: If the value of r not equal to 0 then print the year is not a leap year.

PROGRAM:

```
echo "Enter the year"
read y
b=`expr $y % 4`
if [ $b -eq 0 ]
then
```

```
echo "$y is a leap year"
else
echo "$y is not a leap year"
fi
```

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

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of n.
STEP 3: Calculate „i=expr $n-1".
STEP 4: If the value of i is greater than 1 then calculate „n=expr $n * $i" and „i=expr $i – 1"
STEP 5: Print the factorial of the given number.

PROGRAM:

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

4.Write a Shell program to swap the two integers

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of a,b.
STEP 3: Calculate the swapping of two values by using a temporary variable temp.
STEP 4: Print the value of a and b.

PROGRAM:

```
echo "Enter Two Numbers"
read a b
temp=$a
a=$b
b=$temp
```

```
echo "after swapping"
echo $a $b
```

5.Write a Shell program for Academic and Personal Details

ALGORITHM:

STEP 1. Get name, age, and address from the user.
STEP 2. Print that message as similar.
STEP 3. Get mark1, mark2, and mark3 from the user.
STEP 4. Print that message as similar.

PROGRAM:

```
echo –n “Enter the name”
read s
echo –n “Enter the age”
read a
echo –n “Enter the address”
read adr
echo –n “The name is $s”
echo –n “The age is $a”
echo –n “The address is $adr”
echo –n “Enter the mark1”
read m1
echo –n “Enter the mark2”
read m2
echo –n “Enter the mark3”
read m3
echo –n “The mark1 is $m1”
echo –n “The mark2 is $m2”
echo –n “The mark3 is $m3”
```

6.Write a Shell program for Greatest Among Three Numbers

ALGORITHM:

STEP 1. Start the program.
STEP 2. Enter any three numbers.
STEP 3. Read the values as a, b and c.
STEP 4. If a greater than b and greater than c, print the value of a as the Greatest number.
STEP 5. Else if b is greater than c, print the value of b as greatest number.
STEP 6. Else print the value of c as the greatest number.
STEP 7. Stop the program.

PROGRAM:

```
echo greatest of 3 numbers
echo enter the numbers
read a
read b
read c
if test $a -gt $b -a $a -gt $c
then
echo $a is greater
elif test $b -gt $c
then
echo $b is greater
else
echo $c is greater
fi
```

7.Write a Shell program to Check Prime Number

ALGORITHM:

STEP 1. Start the program
STEP 2. Input number n
STEP 3. Is i>2, repeat the following steps.
STEP 4. p=n%i.
STEP 5. Is p=0 then increment t value by 1.
STEP 6. Decrement i value by one
STEP 7. Is t value > zero then print number is prime
STEP 8. Otherwise print number is not a prime
STEP 9. Stop the program

PROGRAM:

```
echo "Enter a Number"
read n
i=`expr $n - 1`
t=0
while [ $i -ge 2 ]
do
p=`expr $n % $i`
if [ $p -eq 0 ]
then
t=`expr $t + 1`
fi
i=`expr $i - 1`
done
```

```
if [ $t -gt 0 ]
then
echo "The Number $n is not a Prime Number"
else
echo "The Number $n is a Prime Number"
fi
```

8.Write a Shell program for Sum of N Numbers

ALGORITHM:

STEP 1. Start the program
STEP 2. Get the limit number 'n' from the user
STEP 3. Initialize sum=0 and i=1
STEP 4. If I is less than or equal to n get the number to be summed from the user and increment I value by i
STEP 5. Add the number to the sum value
STEP 6. Repeat step 4 and 5
STEP 7. Print the sum value
STEP 8. Stop the program.

PROGRAM:

```
echo "enter the limit"
read n
echo "enter the $n numbers"
sum=0
i=1
while test $i -le $n
do
read num
sum=`expr $num + $sum`
i=`expr $i + 1`
done
echo "the sum of the numbers are $sum"
```

9.Write a Shell program for Fibonacci Series

ALGORITHM:

STEP 1. Input the range(n)
STEP 2. Initialize b=1,a=0,s=0
STEP 3. Do the following in a loop, until s less than or equal to n
a=b;b=s
Print the Fibonacci series value i.e.(s)
Let s=a+b
STEP 4. Stop.

PROGRAM:

```
echo "Enter the range to be displayed"
read n
a=0
b=1
s=0
echo "Fibonacci series"
while test $s -le $n
do
a=$b
b=$s
echo $s
s=`expr $a + $b`
done
```

10.Write a Shell program for Armstrong Number

ALGORITHM:

STEP 1. Start the program.
STEP 2. Get the input value(num)
STEP 3. Assign value of num to x
STEP 4. Assign value of sum equal to zero.
STEP 5. Repeat the following steps till the num greater than zero
STEP 6. Find y equal to num modulus 10.Find z equal to cube of y. Then find num equal to num divided by 10.
STEP 7. If x equal to sum then print the value is Armstrong Otherwise print the number is not armstrong
STEP 8. Stop the program.

PROGRAM:

```
echo "Enter a Number"
read num
x=$num
sum=0
while [$num -gt 0]
do
y=`expr $num % 10`
z=`expr $y \* $y \* $y`
sum=`expr $sum + $z`
num=`expr $num / 10`
done
if [$x -eq $sum]
then
```

```
echo "$x is an armstrong Number"
else
echo "$x is not an armstrong Number"
fi
```

11.Write a Shell program for Arithmetic Operations Using Switch Case

ALGORITHM:

STEP 1. Enter the input
STEP 2. Enter the choice
STEP 3. Perform corresponding arithmetic operation on the inputs
STEP 4. Display the result
STEP 5. Stop

PROGRAM:

```
echo "Performing Arithmetic Manipulation"
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo -n "Enter the first no:"
read a
echo -n "Enter the second no:"
read b
echo "Arithmetic Operation - Menu"
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "0.Exit"
echo -n "Enter your Choice:"
read ch
case $ch in
1) echo " Option 1 Performs Addition" ;
      c=`expr $a + $b`;
      echo "The sum of $a and $b is $c";
exit ;;
2) echo " Option 2 Performs Subtraction" ;
      c=`expr $a - $b`;
      echo "The difference of $a and $b is $c";
exit ;;
3) echo " Option 3 Performs Multiplication" ;
      c=`expr $a \* $b`;
      echo "The product of $a and $b is $c";
exit ;;
4) echo " Option 4 Performs Division" ;
      c=`expr $a / $b`;
```

```
        echo "The Division of $a by $b is $c";exit ;;
        echo "You are exiting from the Arithmetic Manipulation" ;
    echo "Thankyou. Visit again"
    exit ;;
    esac
```

12.Write a Shell program for Palindrome Number

ALGORITHM:

STEP 1. Start the program.
STEP 2. Get the input value(a)
STEP 3. Assign value of a to d
STEP 4. Assign value of c equal to zero and b equal to one.
STEP 5. Repeat the following steps till the value of a greater than zero
STEP 6. Find b equal to a modulus 10.Find c equal to c *10 + b . Then find a equal to a divided by 10.
STEP 7. If d equal to c then print the value is Palindrome Otherwise print the number is not Palindrome
STEP 8. Stop the program.

PROGRAM:

```
    echo "Enter a number"
    read a
    d=$a
    c=0
    b=1
    while [ $a -gt 0 ]
    do
    b=`expr $a % 10`
    c=`expr $c \* 10 + $b`
    a=`expr $a / 10`
    done
    if [ $d -eq $c ]
    then
    echo "PALINDROME"
    else
    echo "NOT PALINDROME"
    fi
```

RESULT

Ex No: 2

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

AIM:

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

1. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (opendir, readdir, closedir)

ALGORITHM:

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

PROGRAM:

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

OUTPUT:

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

ALGORITHM:

STEP 1: Start the program.
STEP 2: Declare the variables pid,pid1,pid2 .
STEP 3: Call fork() system call to create process.
STEP 4: If pid==-1, exit.
STEP 5: Ifpid!=-1 , get the process id using getpid().
STEP 6: Print the process id.
STEP 7: Stop the program

PROGRAM:

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

OUTPUT:

RESULT:

Ex No :3 **CPU SCHEDULING ALGORITHMS**

1. FIRST COME FIRST SERVED (FCFS) SCHEDULING

AIM:

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

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.
Step 3: Get the value of „n" assign pid as I and get the value of p[i].btime.
Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time
and turnaround time.
Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.
Step 6: Print total wait time and total turnaround time.
Step 7: Stop the program.

PROGRAM:

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

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

OUTPUT:

2. SHORTEST JOB FIRST (SJF) SCHEDULING

AIM:

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

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.
Step 3: Get the value of „n" assign pid as I and get the value of p[i].btime.
Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time
and turnaround time.
Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.
Step 6: Print total wait time and total turnaround time.
Step 7: Stop the program.

PROGRAM:

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

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

OUTPUT:

3. PRIORITY SCHEDULING

AIM:

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

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.
Step 3: Get the value of „n" assign p and allocate the memory.
Step 4: Inside the for loop get the value of burst time and priority.
Step 5: Assign wtime as zero .
Step 6: Check p[i].pri is greater than p[j].pri .
Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
        int pno;
        int pri;
        int pri;
        int btime;
        int wtime;
}sp;
int main()
{
        int i,j,n;
        int tbm=0,totwtime=0,totttime=0;
        sp *p,t;
        printf("\n PRIORITY SCHEDULING.\n");
        printf("\n enter the no of process....\n");
        scanf("%d",&n);
        p=(sp*)malloc(sizeof(sp));
        printf("enter the burst time and priority:\n");
        for(i=0;i<n;i++)
        {
                printf("process%d:",i+1);
                scanf("%d%d",&p[i].btime,&p[i].pri);
                p[i].pno=i+1;
                p[i].wtime=0;
        }
        for(i=0;i<n-1;i++)
```

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

OUTPUT:

4. ROUND ROBIN SCHEDULING

AIM:

To write a C program for implementation of Round Robin scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.
Step 3: Get the value of „n‟ assign p and allocate the memory.
Step 4: Inside the for loop get the value of burst time and priority and read the time quantum.
Step 5: Assign wtime as zero.
Step 6: Check p[i].pri is greater than p[j].pri .
Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct rr
{
        int pno,btime,sbtime,wtime,lst;
}p[10];
int main()
{
        int pp=-1,ts,flag,count,ptm=0,i,n,twt=0,totttime=0;
        printf("\n round robin scheduling.....");
        printf("enter no of processes:");
        scanf("%d",&n);
        printf("enter the time slice:");
        scanf("%d",&ts);
        printf("enter the burst time");
        for(i=0;i<n;i++)
        {
                printf("\n process%d\t",i+1);
                scanf("%d",&p[i].btime);
                p[i].wtime=p[i].lst=0;
                p[i].pno=i+1;
                p[i].sbtime=p[i].btime;
        }
        printf("scheduling....\n");
        do
        {
                flag=0;
                for(i=0;i<n;i++)
                {
                        count=p[i].btime;
                        if(count>0)
                        {
                                flag=-1;
```

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

OUTPUT:

RESULT:

Ex No :4

INTER PROCESS COMMUNICATION

ALGORITHM

Step 1: Start the program.
Step 2: Define the key.
Step 3: Attach the client to the shared memory created by the server.
Step 4: Read the content from the shared memory.
Step 5: Display the content on the screen.
Step 6: Stop

PROGRAM:

MESSAGE QUEUE FOR WRITER PROCESS

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10
struct mesg_buffer {
        long mesg_type;
        char mesg_text[100];
} message;
int main()
{
        key_t key;
        int msgid;
        key = ftok("progfile", 65);
        msgid = msgget(key, 0666 | IPC_CREAT);
        message.mesg_type = 1;
        printf("Write Data : ");
        fgets(message.mesg_text,MAX,stdin);
        msgsnd(msgid, &message, sizeof(message), 0);
        printf("Data send is : %s \n", message.mesg_text);
        return 0;
}
```

MESSAGE QUEUE FOR READER PROCESS

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesg_buffer {
        long mesg_type;
        char mesg_text[100];
} message;
int main()
{
        key_t key;
        int msgid;
```

```
        key = ftok("progfile", 65);
        msgid = msgget(key, 0666 | IPC_CREAT);
        msgrcv(msgid, &message, sizeof(message), 1, 0);
        printf("Data Received is : %s \n", message.mesg_text);
        msgctl(msgid, IPC_RMID, NULL);
        return 0;
    }
```

OUTPUT:

RESULT:

Ex No :5

MUTUAL EXCLUSION

AIM:

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

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the required variables.
Step 3: Initialize the buffer size and get maximum item you want to produce.
Step 4: Get the option, which you want to do either producer, consumer or exit from the operation.
Step 5: If you select the producer, check the buffer size if it is full the producer should not produce the item or otherwise produce the item and increase the value buffer size.
Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not consume the item or otherwise consume the item and decrease the value of buffer size.
Step 7: If you select exit come out of the program.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
      {
        printf("\nEnter Your Choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1:
            if((mutex==1)&&(empty!=0))
             producer();
            else
              printf("Buffer is Full!!");
              break;
            case 2:
```

```
                        if((mutex==1)&&(full!=0))
                         consumer();
                        else
                        printf("Buffer is Empty!!");
                        break;
                case 3:
                        exit(0);
                        break;
                         }
                 }
                 return 0;
        }
        int wait(int s)
        {
                return (--s);
        }
        int signal(int s)
        {
          return(++s);
        }
        void producer()
        {
                mutex=wait(mutex);
                full=signal(full);
                empty=wait(empty);
                x++;
                printf("\nProducer Produces Item %d",x);
                mutex=signal(mutex);
        }
        void consumer()
        {
          mutex=wait(mutex);
          full=wait(full);
          empty=signal(empty);
          printf("\nConsumer Consumes Item %d",x);
          x--;
          mutex=signal(mutex);
    }
```

OUTPUT:

RESULT:

Ex No :6

BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

AIM:

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

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the memory for the process.
Step 3: Read the number of process, resources, allocation matrix and available matrix.
Step 4: Compare each and every process using the banker‟s algorithm.
Step 5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process
Step 6: Produce the result of state of process
Step 7: Stop the program

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int p, r, i, j, process, count;
  int completed[20], Max[20][20], alloc[20][20], need[20][20], safeSequence[20],
   avail[20];
 count = 0;
printf("Enter the no of Processes : ");
scanf("%d", &p);
for(i = 0; i< p; i++)
        completed[i] = 0;
printf("\n\nEnter the no of Resources : ");
scanf("%d", &r);
printf("\n\nEnter the Max Matrix for each Process : ");
for(i = 0; i < p; i++)
{
         printf("\nFor Process %d : ", i + 1);
         for(j = 0; j < r; j++)
         scanf("%d", &Max[i][j]);
}
printf("\n\nEnter the allocation for each Process : ");
        for(i = 0; i < p; i++)
 {
        printf("\nFor Process %d : ",i + 1);
        for(j = 0; j < r; j++)
        scanf("%d", &alloc[i][j]);
 }
printf("\n\nEnter the Available Resources : ");
```

```
for(i = 0; i < r; i++)
    scanf("%d", &avail[i]);
for(i = 0; i < p; i++)
    for(j = 0; j < r; j++)
        need[i][j] = Max[i][j] - alloc[i][j];
    do
    {
        printf("\n Max Matrix:\tAllocation Matrix:\n");
        for(i = 0; i < p; i++)
        {
            for( j = 0; j < r; j++)
                printf("%d ", Max[i][j]);
            printf("\t\t");
            for( j = 0; j < r; j++)
                printf("%d ", alloc[i][j]);
            printf("\n");
        }
        process = -1;
        for(i = 0; i < p; i++)
        {
            if(completed[i] == 0)
            {
                process = i ;
                for(j = 0; j < r; j++)
                {
                    if(avail[j] < need[i][j])
                    {
                        process = -1;
                        break;
                    }
                }
            }
            if(process != -1)
                break;
        }
        if(process != -1)
        {
            printf("\nProcess %d runs to Completion!", process + 1);
            safeSequence[count] = process + 1;
            count++;
            for(j = 0; j < r; j++)
            {
                avail[j] += alloc[process][j];
                alloc[process][j] = 0;
                Max[process][j] = 0;
                completed[process] = 1;
            }
        }
```

```
        }
        while(count != p && process != -1);
        if(count == p)
        {
            printf("\nThe system is in a Safe State!!\n");
            printf("Safe Sequence : < ");
            for( i = 0; i < p; i++)
                printf("%d ", safeSequence[i]);
            printf(">\n");
        }
        else
            printf("\nThe system is in an Unsafe State!!");
}
```

OUTPUT:

RESULT:

Ex No :7

ALGORITHM FOR DEADLOCK DETECTION

AIM:

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

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the memory for the process.
Step 3: Read the number of process, resources, allocation matrix and available matrix.
Step 4: Compare each and every process using the banker‟s algorithm.
Step 5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process
Step 6: Produce the result of state of process
Step 7: Stop the program

PROGRAM:

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;
int main()
{
        alloc[10][10],request[10][10],avail[10],r[10],w[10];
        printf("\nEnter the no of Process: ");
        scanf("%d",&np);
        printf("\nEnter the no of Resources: ");
        scanf("%d",&nr);
        for(i=0;i<nr;i++)
       {
              printf("\nTotal Amount of the Resource R%d: ",i+1);
              scanf("%d",&r[i]);
       }
       printf("\nEnter the Request Matrix:");
       for(i=0;i<np;i++)
       for(j=0;j<nr;j++)
       scanf("%d",&request[i][j]);
       printf("\nEnter the Allocation Matrix:");
       for(i=0;i<np;i++)
       for(j=0;j<nr;j++)
       scanf("%d",&alloc[i][j]);
       for(j=0;j<nr;j++)
       {
              avail[j]=r[j];
              for(i=0;i<np;i++)
       {
              avail[j]-=alloc[i][j];
       }
       }
```

```
for(i=0;i<np;i++)
{
        int count=0;
        for(j=0;j<nr;j++)
 {
         if(alloc[i][j]==0)
          count++;
           else
          break;
}
 if(count==nr)
 mark[i]=1;
}
for(j=0;j<nr;j++)
 w[j]=avail[j];
for(i=0;i<np;i++)
{
        int canbeprocessed=0;
         if(mark[i]!=1)
        {
                 for(j=0;j<nr;j++)
                {
                 if(request[i][j]<=w[j])
                 canbeprocessed=1;
        else
         {
                canbeprocessed=0;
                break;
         }
        }
if(canbeprocessed)
{
        mark[i]=1;
        for(j=0;j<nr;j++)
        w[j]+=alloc[i][j];
}
}
}
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;
if(deadlock)
printf("\n Deadlock Detected");
else
printf("\n No Deadlock Possible");
}
```

OUTPUT:

RESULT:

Ex No :8

IMPLEMENTATION OF THREADING

AIM:

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

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* trythis(void *arg)
{
        unsigned long i = 0;
        counter += 1;
        printf("\n Job %d has Started.....\n", counter);
        for(i=0; i<(0xFFFFFFFF);i++);
        printf("\n Job %d has Finished.....\n", counter);
        return NULL;
}
int main(void)
{
        int i = 0;
        int error;
        while(i < 2)
        {
                error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
                if (error != 0)
                printf("\nThread can't be created : [%s]", strerror(error));
                i++;
        }
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
return 0;
```

OUTPUT:

RESULT:

Ex No :9

IMPLEMENTATION OF PAGING TECHNIQUE

AIM:

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

ALGORITHM:

Step 1: Start the process
Step 2: Declare page number, page table, frame number and process size.
Step 3: Read the process size, total number of pages
Step 4: Read the relative address
Step 5: Calculate the physical address
Step 6: Display the address
Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
main()
{
        int memsize=15;
        int pagesize,nofpage;
        int p[100];
        int frameno,offset;
        int logadd,phyadd;
        int i;
        int choice=0;
        printf("\nYour Memory Size is %d ",memsize);
        printf("\nEnter Page Size:");
        scanf("%d",&pagesize);
        nofpage=memsize/pagesize;
        for(i=0;i<nofpage;i++)
        {
                printf("\nEnter the Frame of Page%d:",i+1);
                scanf("%d",&p[i]);
        }
        do
        {
                printf("\nEnter a logical address:");
                scanf("%d",&logadd);
                frameno=logadd/pagesize;
                offset=logadd%pagesize;
                phyadd=(p[frameno]*pagesize)+offset;
                printf("\nPhysical address is:%d",phyadd);
                printf("\nDo you want to continue(1/0)?:");
                scanf("%d",&choice);
                }while(choice==1);
        }
```

OUTPUT:

RESULT:

Ex No :10(a)

IMPLEMENTATION OF MEMORY ALLOCATION METHODS – FIRST FIT

AIM:

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

ALGORITHM:

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

PROGRAM:

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

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

OUTPUT:

RESULT:

Ex No :10(b)

IMPLEMENTATION OF MEMORY ALLOCATION METHODS – WORST FIT

AIM:

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

ALGORITHM:

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

PROGRAM:

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

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

OUTPUT:

RESULT:

Ex No :10(c)

IMPLEMENTATION OF MEMORY ALLOCATION METHODS – BEST FIT

AIM:

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

ALGORITHM:

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

PROGRAM:

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

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

OUTPUT:

RESULT:

Ex No :11(a)

IMPLEMENTATION OF FIFO PAGE REPLACEMENT ALGORITHM

AIM:

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

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
int main()
{
        int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("FIFO Page Replacement \n");
        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:

RESULT:

Ex No :11(b)

IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM

AIM:

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

ALGORITHM:

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

PROGRAM:

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

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

OUTPUT:

RESULT:

Ex No :11(c)

IMPLEMENTATION OF OPTIMAL PAGE REPLACEMENT ALGORITHM

AIM:

To write C program to implement optimal page replacement algorithm.

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
 int main()
{
        int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1,
        flag2, flag3, i, j, k, pos, max, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter page 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]){
                        flag1 = flag2 = 1;
                        break;
                }
        }
        if(flag1 == 0){
                for(j = 0; j < no_of_frames; ++j){
                        if(frames[j] == -1){
                                faults++;
                                frames[j] = pages[i];
```

```
                        flag2 = 1;
                        break;
                    }
                }
            }
            if(flag2 == 0){
                flag3 =0;
                for(j = 0; j < no_of_frames; ++j){
                    temp[j] = -1;
                    for(k = i + 1; k < no_of_pages; ++k){
                        if(frames[j] == pages[k]){
                            temp[j] = k;
                            break;
                        }
                    }
                }
            for(j = 0; j < no_of_frames; ++j){
                if(temp[j] == -1){
                    pos = j;
                    flag3 = 1;
                    break;
                }
            }
            if(flag3 ==0){
                max = temp[0];
                pos = 0;
                for(j = 1; j < no_of_frames; ++j)
                {
                    if(temp[j] > max)
                    {
                        max = temp[j];
                        pos = j;
                    }
                }
            }
            frames[pos] = pages[i];
            faults++;
        }
        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:

RESULT:

Ex No :12(a)

IMPLEMENTATION OF FILE ORGANIZATION -SINGLE LEVEL DIRECTORY

AIM:

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

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
struct
{
        char dname[10],fname[10][10];
        int fcnt;
}dir;
main()
{
        int i,ch;
        char f[30];
        dir.fcnt = 0;
        printf("\nEnter name of directory -- ");
        scanf("%s", dir.dname);
        while(1)
        {
                printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4.
                Display Files\t5. Exit\n Enter your choice -- ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter the name of the file -- ");
                        scanf("%s",dir.fname[dir.fcnt]);
                        dir.fcnt++;
                        break;
                        case 2: printf("\nEnter 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("\nEnter 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("\nDirectory Empty");
                else
                {
                        printf("\nThe Files are -- ");
                        for(i=0;i<dir.fcnt;i++)
                        printf("\t%s",dir.fname[i]);
                }
                break;
                default: exit(0);
                }
        }
    }
```

OUTPUT:

RESULT:

Ex No :12(b)

IMPLEMENTATION OF FILE ORGANIZATION -TWO LEVEL DIRECTORY

AIM:

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

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
struct
{
        char dname[10],fname[10][10];
        int fcnt;
}dir[10];
main()
{
        int i,ch,dcnt,k;
        char f[30], d[30];
        dcnt=0;
        while(1)
        {
        printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
        printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice -- ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("\nEnter name of directory -- ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt=0;
                dcnt++;
                printf("Directory created");
                break;
                case 2: 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",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);
        }
        }
        }
```

OUTPUT:

RESULT:

Ex No :13(a)

IMPLEMENTATION OF SEQUENTIAL FILE ALLOCATION

AIM:

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

ALGORITHM:

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

PROGRAM:

```
#include < stdio.h>
main()
{
        int f[50], i, st, len, j, c, k, count = 0;
        for(i=0;i<50;i++)
        f[i]=0;
        printf("Files Allocated are : \n");
        x: count=0;
        printf("Enter starting block and length of files: ");
        scanf("%d%d", &st,&len);
        for(k=st;k<(st+len);k++)
        if(f[k]==0)
                count++;
                if(len==count)
                {
                        for(j=st;j<(st+len);j++)
                        if(f[j]==0)
                        {
                                f[j]=1;
                                printf("%d\t%d\n",j,f[j]);
                         }
                        if(j!=(st+len-1))
                        printf(" The file is allocated to disk\n");
                }
        else
        printf(" The file is not allocated \n");
        printf("Do you want to enter more file(Yes - 1/No - 0)");
        scanf("%d", &c);
```

```
if(c==1)
goto  x;
else
exit();
}
```

OUTPUT:

RESULT:

Ex No :13(b)

IMPLEMENTATION OF LINKED FILE ALLOCATION

AIM:

To write a C program for implementation of linked file allocation.

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        int f[50], p,i, st, len, j, c, k, a;
        for(i=0;i<50;i++)
        f[i]=0;
        printf("Enter how many blocks already allocated: ");
        scanf("%d",&p);
        printf("Enter blocks already allocated: ");
        for(i=0;i<p;i++)
        {
                scanf("%d",&a);
                 f[a]=1;
        }
        x: printf("Enter index starting block and length: ");
        scanf("%d%d", &st,&len);
        k=len;
        if(f[st]==0)
        {
                for(j=st;j<(st+k);j++)
                {
                        if(f[j]==0)
                        {
                                f[j]=1;
                                printf("%d-------->%d\n",j,f[j]);
                        }
                        else
                         {
```

```
                                printf("%d Block is already allocated \n",j);
                                k++;
                        }
                }
        }
        else
                printf("%d starting block is already allocated \n",st);
                printf("Do you want to enter more file(Yes - 1/No - 0)");
                scanf("%d", &c);
                if(c==1)
                        goto x;
                else
                 exit(0);
        }
```

OUTPUT:

RESULT:

Ex No :13(c)

IMPLEMENTATION OF INDEXED FILE ALLOCATION

AIM:

To write a C program for implementation of indexed file allocation.

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
        for(i=0;i<50;i++)
        f[i]=0;
        x:printf("Enter the index block: ");
        scanf("%d",&ind);
        if(f[ind]!=1)
        {
                printf("Enter no of blocks needed and no of files for the index
                        %d on the disk : \n",ind);
                scanf("%d",&n);
        }
        else
        {
                printf("%d index is already allocated \n",ind);
                goto x;
        }
        y: count=0;
        for(i=0;i<n;i++)
        {
                scanf("%d", &index[i]);
                 if(f[index[i]]==0)
                count++;
        }
        if(count==n)
        {
```

```
                    for(j=0;j<n;j++)
                    f[index[j]]=1;
                    printf("Allocated\n");
                    printf("File Indexed\n");
                    for(k=0;k<n;k++)
                    printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
            }
        else
        {
            printf("File in the index is already allocated \n");
            printf("Enter another file indexed");
            goto y;
        }
        printf("Do you want to enter more file(Yes - 1/No - 0)");
            scanf("%d", &c);
            if(c==1)
            goto x;
            else
            exit(0);
        }
```

OUTPUT:

RESULT:

Ex No :14(a)

IMPLEMENTATION OF FCFS DISK SCHEDULING ALGORITHM

AIM:

To write a C program for implementation of FCFS Disk Scheduling Algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.

Step 3: Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.

Step 4: Increment the total seek count with this distance.

Step 5: Currently serviced track position now becomes the new head position.

Step 6: Go to step 3 until all tracks in request array have not been serviced.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
        int queue[20],n,head,i,j,k,seek=0,max,diff;
        float avg;
        printf("Enter the max range of disk\n");
        scanf("%d",&max);
        printf("Enter the size of queue request\n");
        scanf("%d",&n);
        printf("Enter the queue of disk positions to be read\n");
        for(i=1;i<=n;i++)
        scanf("%d",&queue[i]);
        printf("Enter the initial head position\n");
        scanf("%d",&head);
        queue[0]=head;
        for(j=0;j<=n-1;j++)
        {
                diff=abs(queue[j+1]-queue[j]);
                seek+=diff;
                printf("Disk head moves from %d to %d with seek %d n",
                                                queue[j],queue[j+1],diff);
        }
        printf("Total Seek Time is %d\n",seek);
        avg=seek/(float)n;
        printf("Average Seek Time is %f\n",avg);
        return 0;
}
```

OUTPUT:

RESULT:

Ex No :14(b)

IMPLEMENTATION OF SSTF DISK SCHEDULING ALGORITHM

AIM:

To write a C program for implementation of SSTF (Short Seek Time First) Disk Scheduling Algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Let Request array represents an array storing indexes of tracks that have been requested. 'head' is the position of disk head.
Step 3: Find the positive distance of all tracks in the request array from head.
Step 4: Find a track from requested array which has not been accessed/serviced yet and has minimum distance from head.
Step 5: Increment the total seek count with this distance.
Step 6: Currently serviced track position now becomes the new head position.
Step 7: Go to step 3 until all tracks in request array have not been serviced.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
        printf("Enter the number of Requests\n");
        scanf("%d",&n);
        printf("Enter the Requests sequence\n");
        for(i=0;i<n;i++)
                scanf("%d",&RQ[i]);
                printf("Enter initial head position\n");
                scanf("%d",&initial);
                while(count!=n)
                {
                        int min=1000,d,index;
                        for(i=0;i<n;i++)
                        {
                                d=abs(RQ[i]-initial);
                                if(min>d)
                                {
                                        min=d;
                                        index=i;
                                 }
                         }
                TotalHeadMoment=TotalHeadMoment+min;
                initial=RQ[index];
                RQ[index]=1000;
```

```
            count++;
            }
 printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

OUTPUT:

RESULT:

Ex No :14(c)

IMPLEMENTATION OF SCAN DISK SCHEDULING ALGORITHM

AIM:

To write a C program for implementation of SCAN Disk Scheduling Algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
Step 3: Let direction represents whether the head is moving towards left or right.
Step 4: In the direction in which head is moving service all tracks one by one.
Step 5: Calculate the absolute distance of the track from the head.
Step 6: Increment the total seek count with this distance.
Step 7: Currently serviced track position now becomes the new head position.
Step 8: Go to step 4 until we reach at one of the ends of the disk.
Step 9: If we reach at the end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.
Step 10: Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
        int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
        printf("Enter the no of tracks to be Traversed : ");
        scanf("%d'",&n);
        printf("\nEnter the position of Head: ");
        scanf("%d",&h);
        t[0]=0;
        t[1]=h;
        printf("\nEnter the Tracks: ");
        for(i=2;i<n+2;i++)
        scanf("%d",&t[i]);
        for(i=0;i<n+2;i++)
        {
              for(j=0;j<(n+2)-i-1;j++)
              {
                    if(t[j]>t[j+1])
                    {
                          temp=t[j];
                          t[j]=t[j+1];
                          t[j+1]=temp;
                    }
              }
        }
        for(i=0;i<n+2;i++)if(t[i]==h)
```

```
                j=i;k=i;
                p=0;
                while(t[j]!=0)
                {
                        atr[p]=t[j];
                        j--;
                        p++;
                }
                atr[p]=t[j];
                for(p=k+1;p<n+2;p++,k++)
                        atr[p]=t[k+1];
                        for(j=0;j<n+1;j++)
                        {
                         if(atr[j]>atr[j+1])
                                d[j]=atr[j]-atr[j+1];
                        else
                                d[j]=atr[j+1]-atr[j];
                                sum+=d[j];
                        }
                printf("\nAverage Header Movements:%f \n",(float)sum/n);
        }
```

OUTPUT:

RESULT:

Ex No :14(d)

IMPLEMENTATION OF C-SCAN DISK SCHEDULING ALGORITHM

AIM:

To write a C program for implementation of C-SCAN Disk Scheduling Algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
Step 3: The head services only in the right direction from 0 to the size of the disk.
Step 4: While moving in the left direction do not service any of the tracks.
Step 5: When we reach the beginning(left end) reverse the direction.
Step 6: While moving in the right direction it services all tracks one by one.
Step 7: While moving in the right direction calculate the absolute distance of the track from the head.
Step 8: Increment the total seek count with this distance.
Step 9: Currently serviced track position now becomes the new head position.
Step 10: Go to step 6 until we reach the right end of the disk.
Step 11: If we reach the right end of the disk reverse the direction and go to step 4 until all tracks in the request array have not been serviced.
Step 12: Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
        int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
        printf("\nEnter the no of tracks to be Traversed: ");
        scanf("%d'",&n);
        printf("\nEnter the position of head: ");
        scanf("%d",&h);
        t[0]=0;
        t[1]=h;
        printf("\nEnter total tracks: ");
        scanf("%d",&tot);
        t[2]=tot-1;
        printf("\nEnter the tracks: ");
        for(i=3;i<=n+2;i++)
                scanf("%d",&t[i]);
                for(i=0;i<=n+2;i++)
                        for(j=0;j<=(n+2)-i-1;j++)
                        if(t[j]>t[j+1])
                        {
                                temp=t[j];
                                t[j]=t[j+1];
```

```
                                t[j+1]=temp;
                        }
        for(i=0;i<=n+2;i++)
        if(t[i]==h)
        {
                j=i;
                break;
        }
        p=0;
        while(t[j]!=tot-1)
        {
                atr[p]=t[j];
                j++;
                p++;
        }
        atr[p]=t[j];
        p++;
        i=0;
        while(p!=(n+3) && t[i]!=t[h])
        {
                atr[p]=t[i];
                 i++;
                 p++;
        }
        for(j=0;j<n+2;j++)
        {
        if(atr[j]>atr[j+1])
                d[j]=atr[j]-atr[j+1];
                else
                d[j]=atr[j+1]-atr[j];
                sum+=d[j];
        }
        printf("\nTotal header movements: %d",sum);
        printf("\nAvg is:  %f",(float)sum/n);
    }
```

OUTPUT:

RESULT:

Ex No :15

INSTALLATION OF WINDOWS OPERATING SYSTEM

AIM:

To install windows 10 operating system.

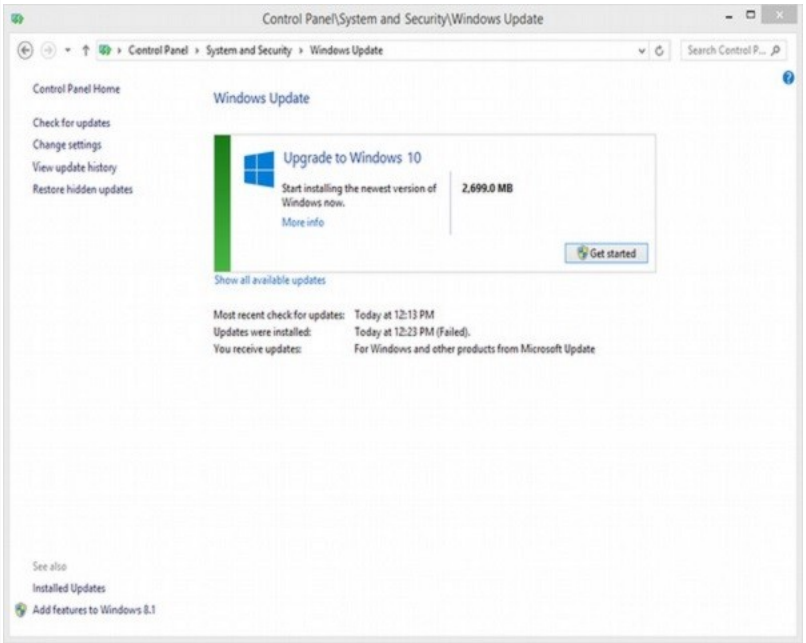
PROCEDURE:

Step 1 :

Look for the Windows 10 notification in the lower-right corner of the screen. This is a one-year-only offer that Microsoft is extending to valid users of Windows 7 and Windows 8.1.

Step 2 :

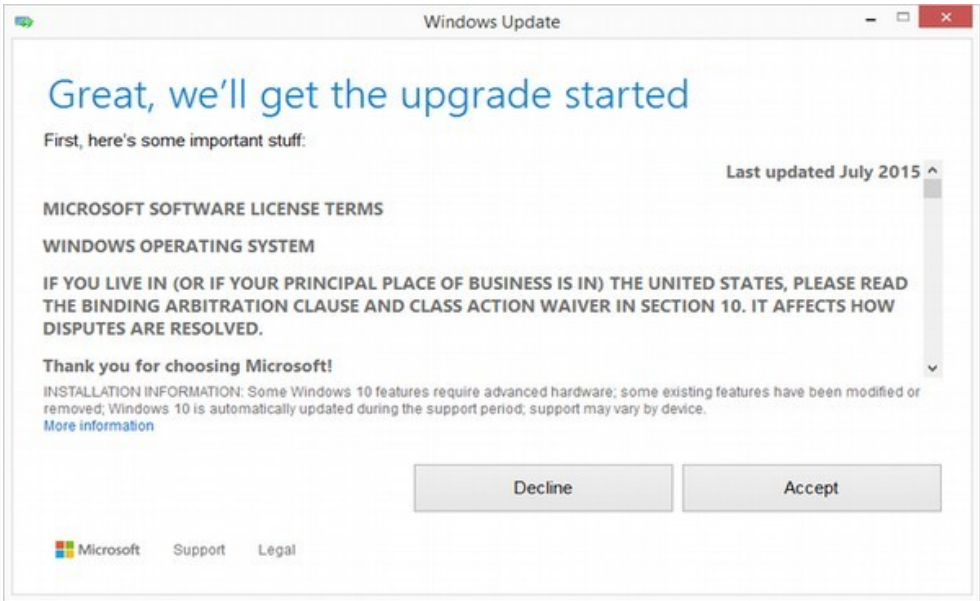
By clicking on the notification, it will start the download and installation process of Windows 10 in your system.



The download required for the upgrade is quite large, so make sure you have a stable Internet connection and continuous power for your computer to avoid interruptions during the process.

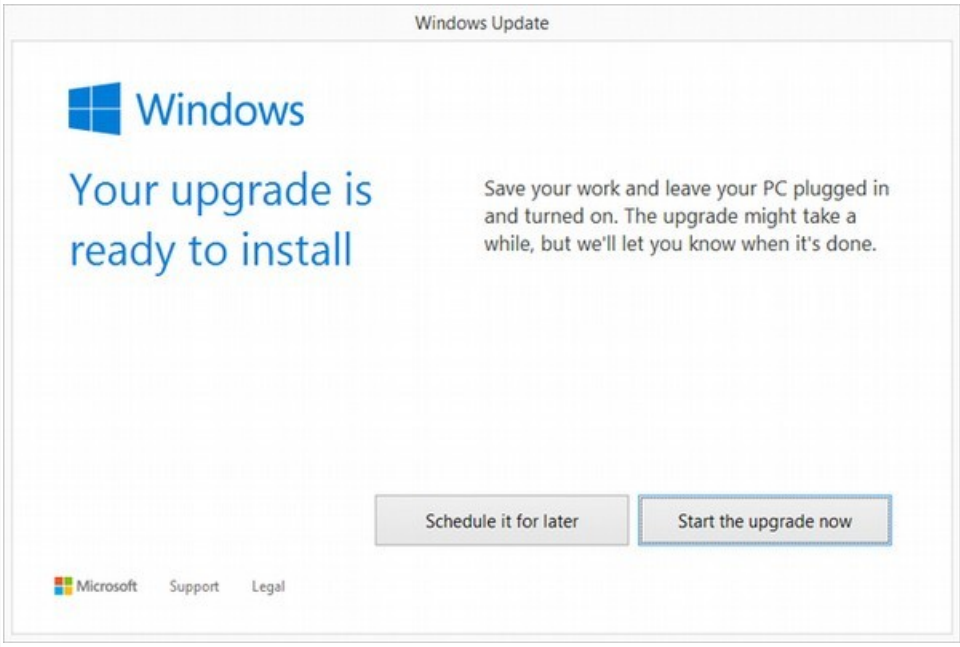
Step 3 :

After the download is complete, it will prompt you to accept Microsoft's license terms.



Step 4:

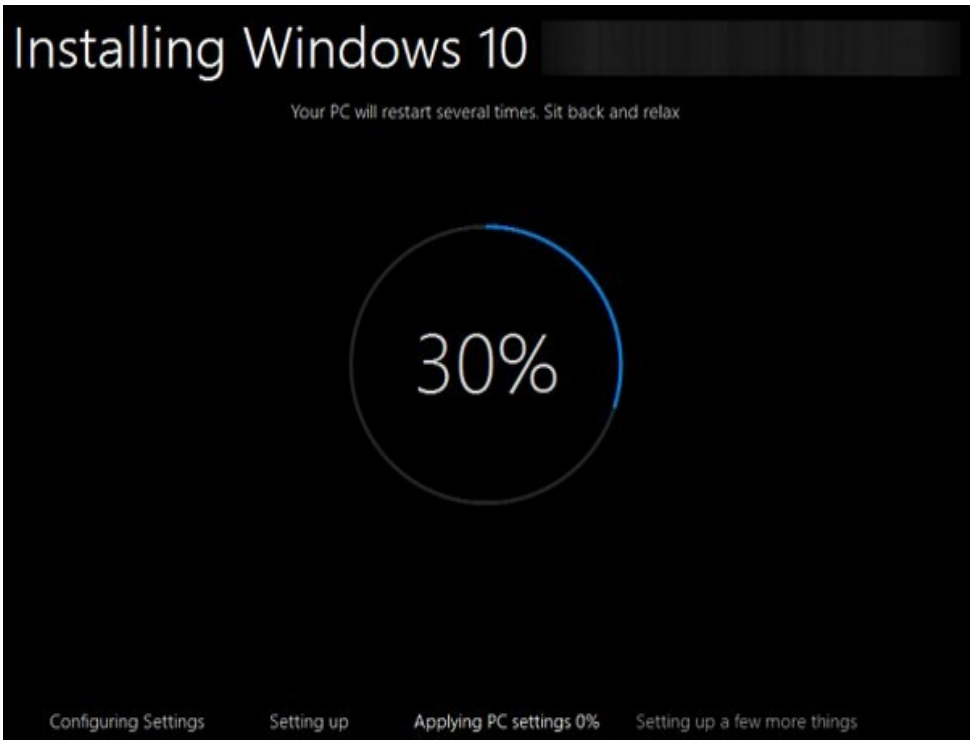
After agreeing to the terms, it will ask if you want to install the upgrade at that moment or schedule it for later.



Since the upgrade process can take approximately 2 hours, it will be helpful to schedule it for a time, that will be more suitable to you.

Step 5:

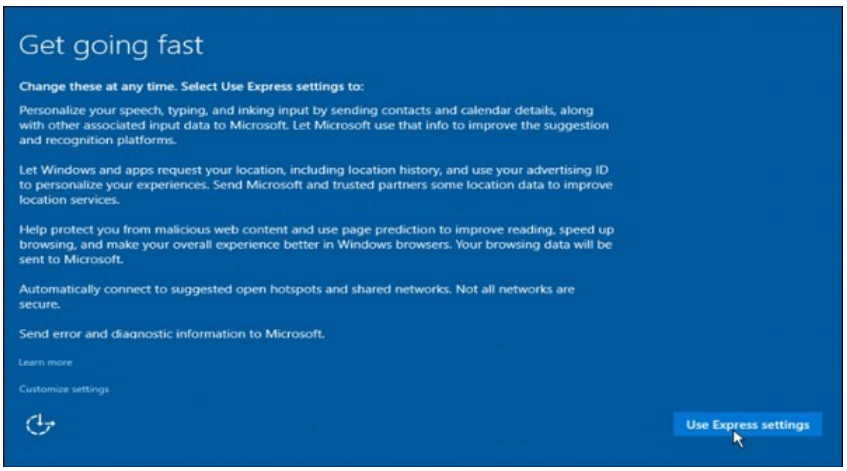
Once the upgrade starts, the system will perform a series of tasks, during which you will see the following screen.



During this time, your computer will reboot a couple of times, so don't worry. The process itself will take you through the steps to complete the upgrade.

Step 6:

As the upgrade approaches its end, it will ask you to configure some basic Windows settings. You can choose to use **Express settings**, which will apply the most common or recommended settings, or you can choose to **customize the settings** as you please.



Step 7:

After the upgrade finishes, you'll see the Windows welcome screen.

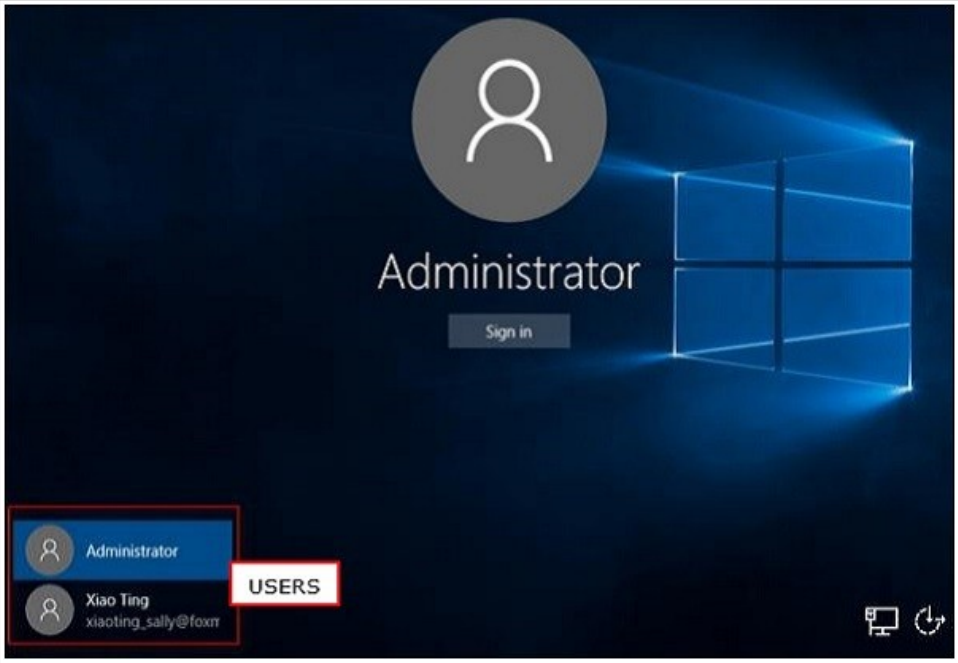
Clean Install

If your computer has an older operating system like Windows XP or Vista, you won't be able to upgrade for free. In these cases, you'll need to buy a boxed copy of Windows 10 to have a valid license for the installation. Windows 10 will have a starting price of $119.

But take in consideration that old computers that have either XP or Vista installed might have obsolete hardware components and might not be suitable for Windows 10. In this case, make sure you review the system requirements listed at the beginning of this chapter to check if your computer is qualified for a Windows 10 upgrade.

If you choose this type of an installation, just insert the disc in your computer and turn it on. Most computers will ask you to press a specific key to boot from the CD/DVD, but most systems use the F12 key. After accessing the disc, you just have to follow the steps which are very similar to the ones from the upgrade.

After you have installed or upgraded your Windows, you will get a Welcome Screen with the time and date. Just click anywhere to go to the User Accounts Screen.

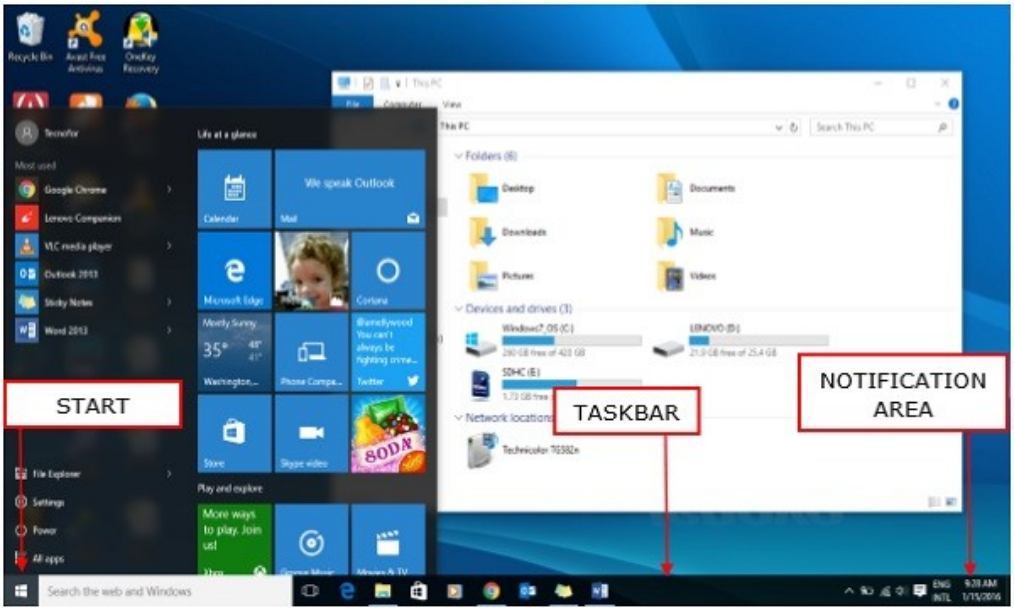


This screen lets you choose which user you want to log in to from the lower-left corner. After choosing the right user, and entering a password if necessary, you will see the Windows Desktop.

The Windows Desktop is simply your operating system main screen. Here you have access to an array of tools like the Start Menu, Taskbar, and other icons.

Windows 10 also introduces a search box in the Taskbar, which facilitates browsing both your computer and the Web.

Once you get to the Windows Desktop screen, here are some basic features you will see.



One of the most important parts of your Desktop is the Taskbar. By default, it sits at the bottom of your screen giving you access to the Start Menu, several application icons, and the Notification Area.

Windows

In Windows 10, if an application is active or opened, you will see a green line below its icon. Clicking the icon will bring the application window up.

Every open window features three buttons in the upper-right corner. These are used to minimize, maximize, or close the window −

- Minimizing means that the window will hide in the Taskbar.
- Maximizing will bring the window to a full-screen size.

Windows can be moved around or resized as you please −

- To move a window, just click on its Title Bar on the upper side of the window and drag it.
- To resize a window, move your mouse to any corner until you see a double-sided arrow. Then click and drag until you reach the desired size.

RESULT:

Thus the windows 10 operating system was installed successfully.

Ex No :16

INSTALLATION OF GUEST OPERATING SYSTEM USING VMWARE

AIM:

To install the guest operating system using vmware.

PROCEDURE:

Step 1:

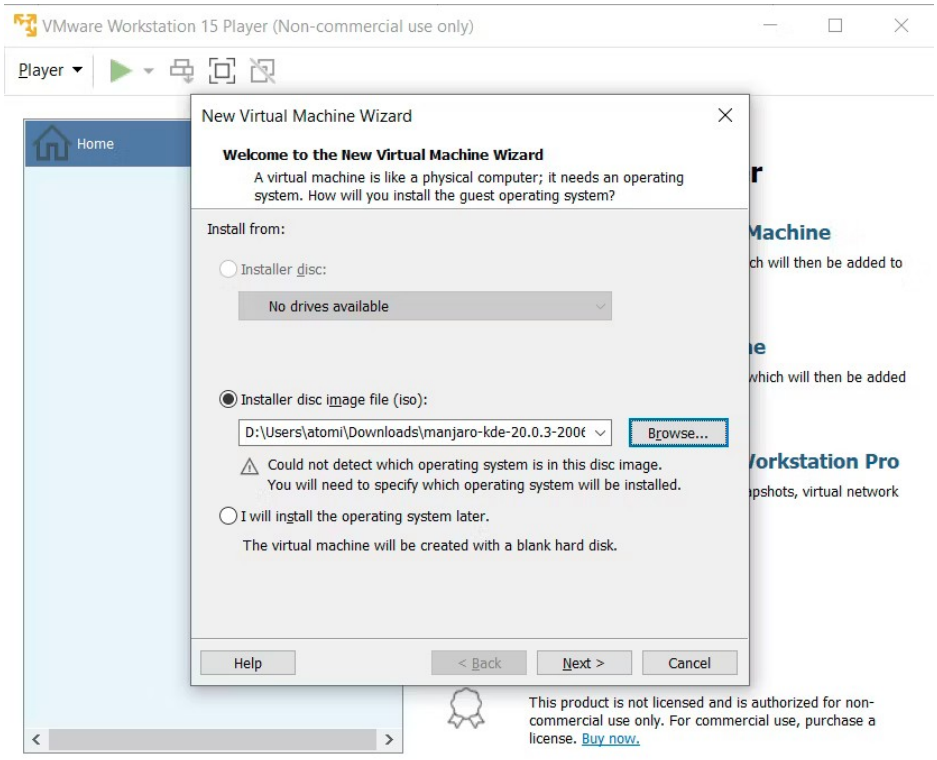
Click Create a New Virtual Machine

Step 2:

Select the default option, **Installer disc image file (iso)**

Step 3:

Click **Browse** to find the ISO file

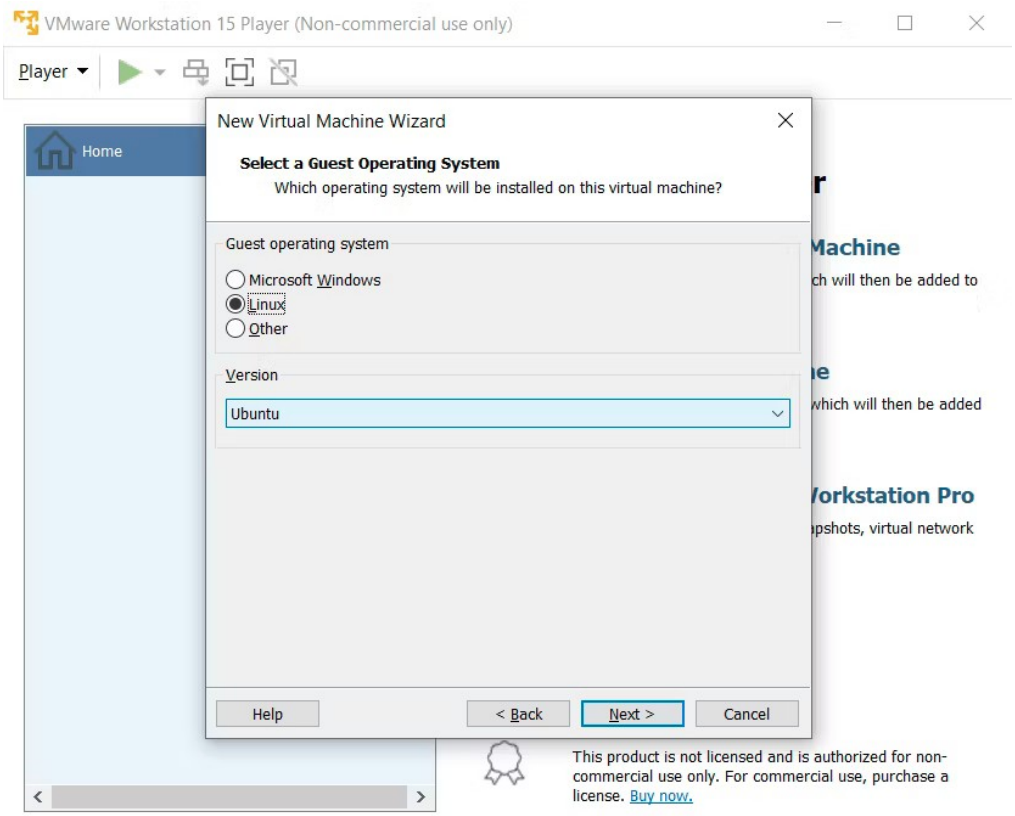


Step 4:

With "guest" OS selected, click **Next**

Step 5:

Select **Linux** as the Guest operating system type



Step 6:

Under **Version,** scroll through the list and select the OS.

Step 7:

Click **Next** to proceed and if necessary, input a **Virtual machine name**

Step 8:

Confirm the storage **Location** and change if needed

With the operating system selected and configured, it's time to build the virtual machine.

Step 9:

Under **Specify Disk Capacity** adjust **Maximum disk size** if required (the default should be enough)

Step 10:

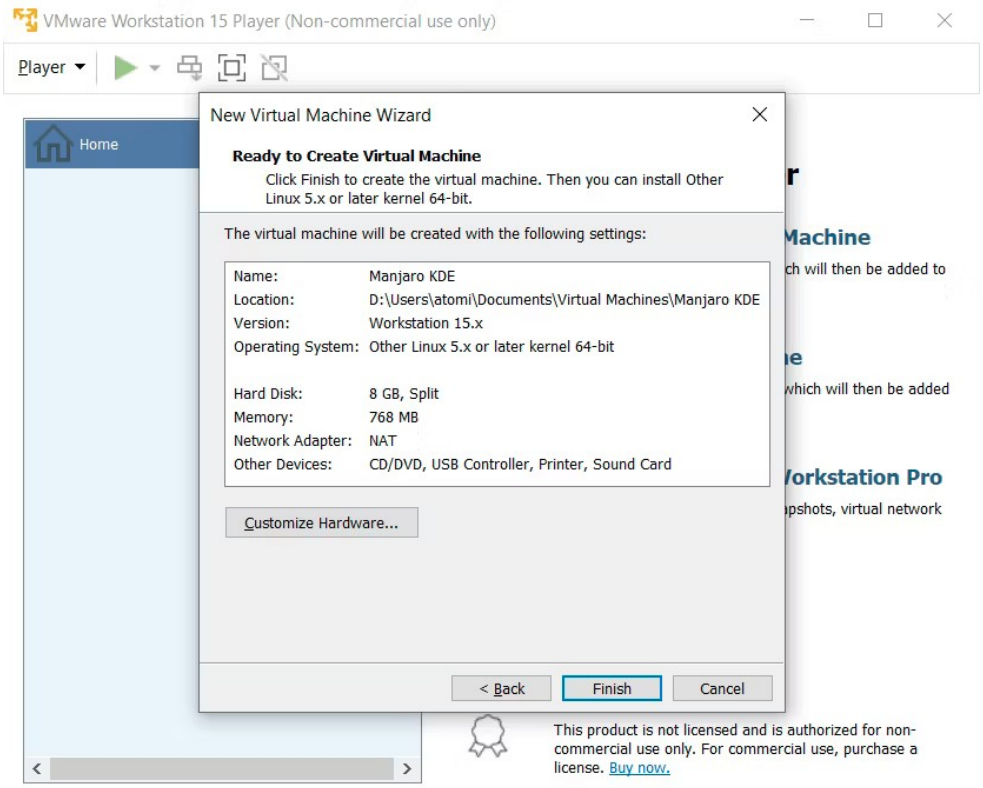
Select **Split virtual disk into multiple files** as this makes moving the VM to a new PC easy

Step 11:

Click **Next** then confirm the details on the next screen

Step 12:

If anything seems wrong click **Back,** otherwise click **Finish**

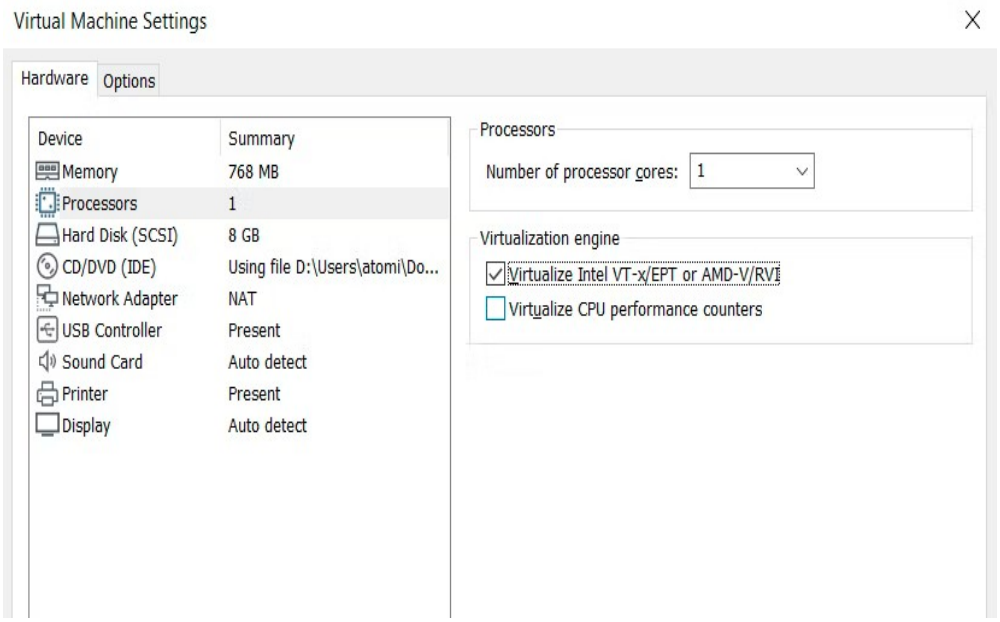


Your Linux virtual machine will be added to VMware Workstation Player.

Customize Your Virtual Hardware

In some cases, you might need to customize the virtual machine before installing Linux. Alternatively, you might install the OS and find there is something missing.

To fix this, right-click your virtual machine in VMware Workstation Player and select **Settings**.



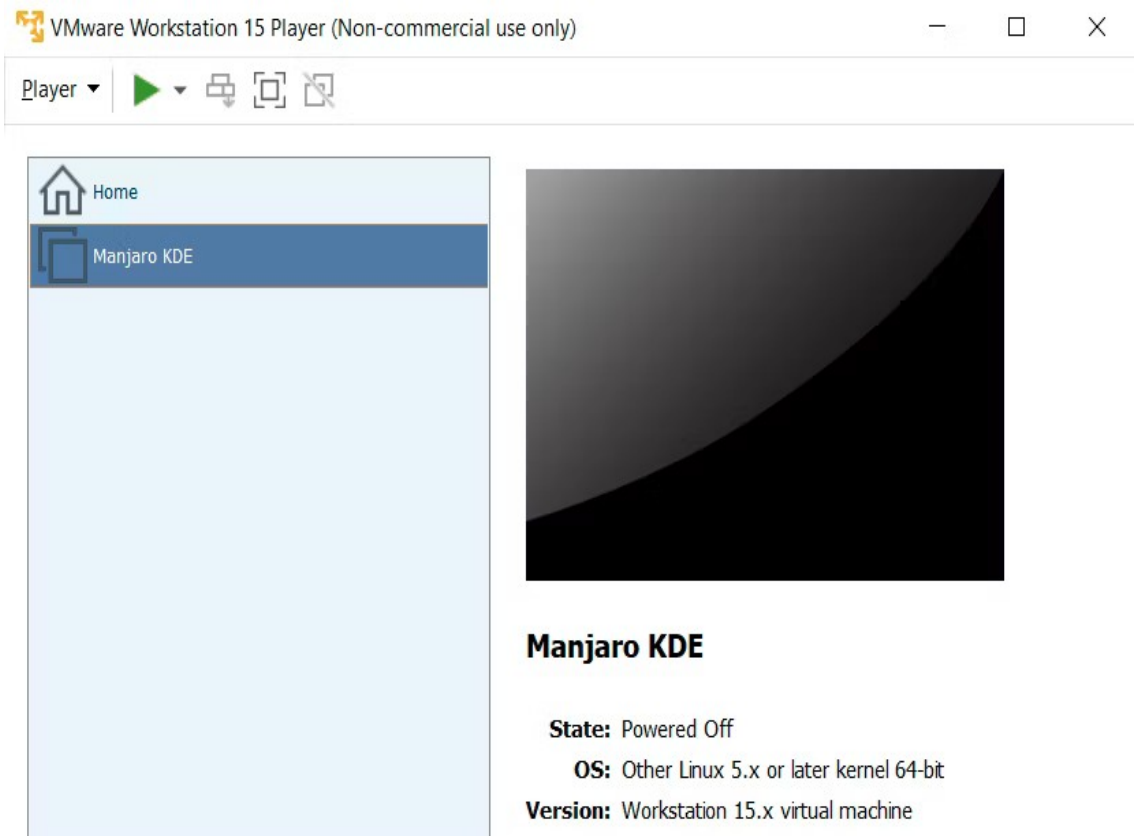
Here, you can tweak the virtual machine's hardware in other ways beyond the HDD. You have options for the **Memory**, **Processors**, **Network Adaptor** configuration, and much more.

It's worth taking a look at the **Processors** screen. In the right-hand pane, you'll spot a reference to a **Virtualization engine**. By default, this works automatically, but for troubleshooting set Intel VT-x or AMD-V, depending on your CPU.

You can address performance issues in the **Memory** screen. Here you'll spot an illustration of the suggested RAM size, as well as recommended options for your virtual machine. It's a good idea to stick to these recommendations. Going too small will prove a problem, while setting the RAM too high will impact on your PC's performance, slowing everything from standard system tasks to running the VM software!

Finally, spare a moment to check the **Display** settings. Default settings should be fine but if there is an issue with the display you can toggle 3D acceleration. Multiple monitors can be used and custom resolution set, but note that some modes will clash with some desktops.

Click **OK** to confirm changes, then select the virtual machine and click the **Play** button to begin.



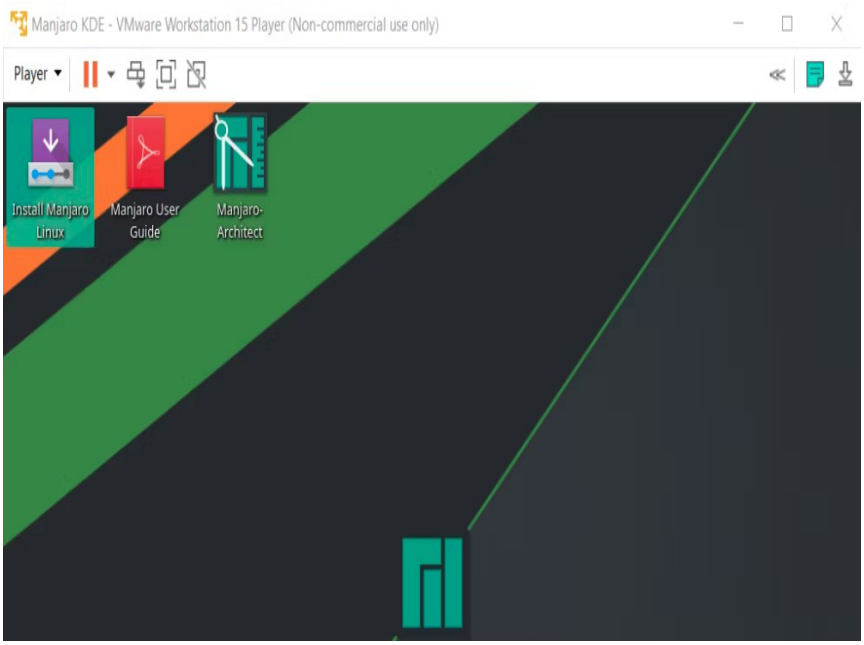
Download and Install VMware Tools

On the first boot of your virtual machine, you'll be prompted to **Download and Install** VMware Tools for Linux. Agree to this, then wait as it is downloaded.

VMware Tools will enhance the performance of the virtual machine while enabling shared folders between host and guest machines.

How to Install Linux in VMware

When the ISO boots in the virtual machine, it will boot into the live environment. This is a temporary Linux that exists only on the boot media and in the system memory. To ensure the environment persists, use the **Install** option on the desktop.



RESULT:

Thus the the guest operating system using vmware was installed successfully.

VIVA QUESTIONS

1. Explain the main purpose of an operating system?

Operating systems exist for two main purposes. One is that it is designed to make sure a computer system performs well by managing its computational activities. Another is that it provides an environment for the development and execution of programs.

2. Define a single-user system.

Single-user systems are the ones with an operating system to cater to only one user at a time. These are commonly our personal computers and became popular due to low hardware cost and wide range of software.

3. What is an Operating system?

An operating system is a program that manages the computer hardware. It also provides a basis for application programs and act as an intermediary between a user of a computer and the computer hardware. It controls and coordinates the use of the hardware among the various application programs for the various users.

4. List the services provided by an Operating System?

- Program execution
- I/O Operation
- File -System manipulation
- Communications
- Error detection

5. Name a few significant features of UNIX?

The following are a few features of UNIX:

- Hardware independent
- Multi-user operations
- Unix Shells
- Hierarchical file system
- Pipes and filters
- Utilities
- Development tools

6. Can you write a command to erase all files in the current directory including all its sub-directories?

rm –r* is used to erase all files in the current directory including all its sub-directories. rm is used for deleting files, but with the addition of option -r it erases all files in directories and subdirectories, and finally, an asterisk represents all entries.

7. What is the fork() system call?

This is used to create another process that duplicates the entire process structure and address space. The newly created process is called the child process and the one from which it got replicated is called the parent process. This was used to achieve parallelism before threads. The fork() system call takes no arguments and returns an integer.

- 0 means that the child process is created successfully, and 0 refers to the child process within the parent process.
- –1 means that the system was unable to create another process.
- default positive integer > 0 is returned to the parent process which represents the process ID of the child process.

8. What is meant by the term Super User?

The Super User is a user with access to all files and commands within the system. In general, this superuser login is to access root and it is secured with the root password.

9. What do chmod, chown, chgrp commands do?

These are file management commands. These are used for:

- chmod - It changes the permission set of a file
- chown - It changes the ownership of the file.
- chgrp - It changes the group of the file.

10. Can you name the important standard streams in the UNIX shell scripting?

These are Standard Input, Standard Output as well as Standard Error.

11. What is the 'nohup' in UNIX?

nohup is a special command to run a process in the background even when a user logs off from the system. It also helps in creating daemon processes or some cleanup script of logs.

12. Explain the term filter.

A program that takes input from the standard input, and displays results to the standard output by performing some actions on it. The most popular example of a Unix filter is the **grep** command. This is used to search for a pattern in files(standard input) and print them out in the output screen(standard output).

13. What can you tell about shell variables?

The shell has a few sets of predefined internal variables with the support of creating new ones as well. Some of these are environment variables whereas others are local variables. These can be accessed by different processes. The name of a variable can contain only letters(A-z), numbers(0-9), or the underscore character (_).

14. Enlist some file manipulation commands in UNIX.

These are few file manipulation commands:

- **cat filename** - Displays contents of the file.
- **cp source destination** - Copy the source file into the destination.
- **mv old_name new_name** - Move/rename.
- **rm filename** - Remove/delete filename.
- **touch filename** - creating/changing modification time.
- **ln [-s] old_name new_name** - Creating a soft link on an old name.
- **ls –F** - Displays information about the file type.
- **ls -ltr** - This will display in long format sorted by modified time with oldest first.

15. Can you explain the different scheduling algorithms?

There are a few different types of scheduling algorithms. The most common are first come first serve, shortest job first, and priority scheduling. First come first serve is exactly what it sounds like- the first process that arrives is the first one to be scheduled. Shortest job first is similar, but instead of being based on arrival time, it is based on the length of the process. Priority scheduling is a bit more complex- each process is assigned a priority, and the scheduler will run the process with the highest priority first.

16. What's your understanding of non-preemptive scheduling?

Non-preemptive scheduling is a type of scheduling algorithm where once a process has started, it cannot be interrupted until it has finished. This can be contrasted with preemptive scheduling, where a process can be interrupted at any time.

17. What are some examples of real-world applications that use non-preemptive scheduling?

Non-preemptive scheduling is often used in applications where the order of execution is important, such as in audio or video processing. Another example is when multiple processes are accessing the same resource, like a printer, and it is important to avoid process starvation.

18. How can you determine the processor utilization if all processes have equal priority?

In order to determine the processor utilization if all processes have equal priority, you will need to look at the total amount of time that each process spends running. This can be done by looking at the total amount of time that each process spends in the ready queue, as well as the total amount of time that each process spends running. By adding these two values together, you will be able to get an accurate measure of the processor utilization.

19. What are the goals of process scheduling?

The goals of process scheduling are to ensure that the processes are executed in a timely manner and that the resources of the system are used efficiently.

20. What do you understand about deadlock situations? How can they be avoided?

A deadlock situation is one where two or more processes are waiting on each other to release a resource before they can continue. This can lead to a situation where the processes are effectively frozen, as each is waiting on the other to make a move.

One way to avoid deadlock situations is to use a scheduling algorithm that prevents processes from holding on to resources for too long. Another way to avoid deadlock is to have a process that periodically checks for and releases resources that are no longer being used.

21. Can you give me an example of when an FCFS algorithm would be useful?

FCFS is often used when scheduling processes that have mutually exclusive resources, like a printer. In this case, it is more important to finish the process quickly rather than worry about response time, so FCFS is a good choice.

22. What are some limitations of using an FCFS algorithm?

FCFS can cause starvation if there are processes with very long run times. Additionally, FCFS can lead to convoy effects, where a long process can cause a number of shorter processes to have to wait a long time.

23. What types of resources can cause a deadlock situation?

A deadlock situation can occur when there are two or more processes that are each holding a resource and waiting to acquire a resource that is being held by the other process. This can happen with any type of resource, but is most commonly seen with computer resources like processors, memory, or files.

24. What are the three conditions required for a deadlock to occur?

The three conditions required for a deadlock to occur are:

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

25. What is starvation in the context of scheduling algorithms?

Starvation is a problem that can occur with scheduling algorithms. It happens when a process is unable to gain access to the CPU because it is continually being preempted

by other processes. This can lead to the process never getting a chance to run, and as a result, it can starve to death.

26. What is the best way to avoid starvation?

One of the best ways to avoid starvation is to use a priority-based scheduling algorithm. This type of algorithm ensures that the most important tasks are always given priority, which helps to prevent lower priority tasks from being starved for resources. Another way to avoid starvation is to use a round-robin scheduling algorithm, which ensures that each task is given a fair share of resources.

27. What is preemptive scheduling?

Preemptive scheduling is a type of scheduling algorithm where the scheduler can interrupt a running process in order to run a higher priority process. This can be contrasted with non-preemptive scheduling, where the scheduler will only run processes that are already waiting in the queue.

28. What are the problems with preemptive scheduling?

Preemptive scheduling can lead to issues with program stability and data corruption, as one process can be interrupted at any time by another process with a higher priority. This can cause problems if the first process was in the middle of writing to or reading from a file, for example, as the data may be left in an inconsistent state. Additionally, context switching between processes can be expensive in terms of time and resources, so if processes are being preempted frequently it can lead to performance issues.

29. What do you know about Round Robin scheduling?

Round Robin scheduling is a type of scheduling algorithm where each process is given a set amount of time to run (called a time slice) and then is preempted and added to the end of the queue. This process repeats until all processes have finished. Round Robin is often used in real-time systems where it is important to guarantee that all processes get a fair amount of time to run.

30. Why is round robin scheduling difficult to implement on platforms like Linux?

Round robin scheduling is difficult to implement on Linux because of the way that the Linux kernel is designed. The Linux kernel is a preemptive kernel, meaning that any process can be interrupted at any time by another process with a higher priority. This can cause issues with round robin scheduling, because it is difficult to guarantee that each process will get an equal amount of time if they can be interrupted at any time.

31. What are the advantages of round robin over FIFO scheduling?

Round robin scheduling has a few advantages over FIFO scheduling. First, round robin is more fair, because each process gets a turn in the CPU. Second, round robin is more efficient, because processes that can finish quickly do not have to wait for slower processes to finish. Finally, round robin is more flexible, because it can be adapted to different types of workloads.

32. What is the difference between process and program?

A program while running or executing is known as a process.

33. What is the use of paging in operating system?

Paging is used to solve the external fragmentation problem in operating system. This technique ensures that the data you need is available as quickly as possible.

34. What is the concept of demand paging?

Demand paging specifies that if an area of memory is not currently being used, it is swapped to disk to make room for an application's need.

35. What is the advantage of a multiprocessor system?

As many as processors are increased, you will get the considerable increment in throughput. It is cost effective also because they can share resources. So, the overall reliability increases.

36. What is virtual memory?

Virtual memory is a very useful memory management technique which enables processes to execute outside of memory. This technique is especially used when an executing program cannot fit in the physical memory.

37. What is thrashing?

Thrashing is a phenomenon in virtual memory scheme when the processor spends most of its time in swapping pages, rather than executing instructions.

38. What is a thread?

A thread is a basic unit of CPU utilization. It consists of a thread ID, program counter, register set and a stack.

39. What is FCFS?

FCFS stands for First Come, First Served. It is a type of scheduling algorithm. In this scheme, if a process requests the CPU first, it is allocated to the CPU first. Its implementation is managed by a FIFO queue.

40. What is SJF?

Shortest Job First (SJF) is a type of disk scheduling algorithm in the operating system in which the processor executes the job first that has the smallest execution time. In the shortest Job First algorithm, the processes are scheduled according to the burst time of these processes.

41. What is Round Robin Scheduling?

Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a quantum. Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

42. What is Priority Scheduling?

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis.

43. What is SMP?

SMP stands for Symmetric MultiProcessing. It is the most common type of multiple processor system. In SMP, each processor runs an identical copy of the operating system, and these copies communicate with one another when required.

44. What is deadlock? Explain.

Deadlock is a specific situation or condition where two processes are waiting for each other to complete so that they can start. But this situation causes hang for both of them.

45. What is Banker's algorithm?

Banker's algorithm is used to avoid deadlock. It is the one of deadlock-avoidance method. It is named as Banker's algorithm on the banking system where bank never allocates available cash in such a manner that it can no longer satisfy the requirements of all of its customers.

46. What is the difference between logical address space and physical address space?

Logical address space specifies the address that is generated by CPU. On the other hand physical address space specifies the address that is seen by the memory unit.

47. What is fragmentation?

Fragmentation is a phenomenon of memory wastage. It reduces the capacity and performance because space is used inefficiently.

48. How many types of fragmentation occur in Operating System?

There are two types of fragmentation:

- **Internal fragmentation:** It is occurred when we deal with the systems that have fixed size allocation units.
- **External fragmentation:** It is occurred when we deal with systems that have variable-size allocation units.

49. What is spooling?

Spooling is a process in which data is temporarily gathered to be used and executed by a device, program or the system. It is associated with printing. When different applications send output to the printer at the same time, spooling keeps these all jobs into a disk file and queues them accordingly to the printer.

50. What is the difference between internal commands and external commands?

Internal commands are the built-in part of the operating system while external commands are the separate file programs that are stored in a separate folder or directory.

51. What is semaphore?

Semaphore is a protected variable or abstract data type that is used to lock the resource being used. The value of the semaphore indicates the status of a common resource.

There are two types of semaphore:

- Binary semaphores
- Counting semaphores

52. What is a binary Semaphore?

Binary semaphore takes only 0 and 1 as value and used to implement mutual exclusion and synchronize concurrent processes.

53. What is Belady's Anomaly?

Belady's Anomaly is also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns.

54. What is starvation in Operating System?

Starvation is Resource management problem. In this problem, a waiting process does not get the resources it needs for a long time because the resources are being allocated to other processes.

55. What is aging in Operating System?

Aging is a technique used to avoid the starvation in resource scheduling system.

56. What are the advantages of multithreaded programming?

A list of advantages of multithreaded programming:

- Enhance the responsiveness to the users.
- Resource sharing within the process.
- Economical
- Completely utilize the multiprocessing architecture.

57. What is the difference between logical and physical address space?

Logical address specifies the address which is generated by the CPU whereas physical address specifies to the address which is seen by the memory unit.

58. What are overlays?

Overlays makes a process to be larger than the amount of memory allocated to it. It ensures that only important instructions and data at any given time are kept in memory.

59. When does trashing occur?

Thrashing specifies an instance of high paging activity. This happens when it is spending more time paging instead of executing.

60. Where is Batch Operating System used in Real Life

They are used in Payroll System and for generating Bank Statements.

61. What is the Function of Operating System?

The most important functions of Operating Systems are:

1. File Management
2. Job Management
3. Process Management
4. Device Management
5. Memory Management

62. What are the Services provided by the Operating System?

The services provided by the Operating Systems are:

1. Security to your computer
2. Protects your computer from external threats
3. File Management
4. Program Execution
5. Helps in Controlling Input Output Devices
6. Useful in Program Creation
7. Helpful in Error Detection
8. Operating System helps in communicating between devices
9. Analyzes the Performance of all devices

63. What are the Types of System Calls in Operating Systems?

The System Calls in Operating Systems are:

1. Communication
2. Information Maintenance
3. File Management
4. Device Management
5. Process Contro

64. What are the functions which are present in the Process Control and File Management System Call?

The Functions present in Process Control System Calls are:

1. Create
2. Allocate
3. Abort
4. End
5. Terminate
6. Free Memory

65. What are the functions which are present in the File Management System Call?

The Functions present in File Management System Calls are:

1. Create
2. Open
3. Read
4. Close
5. Delete

66. What are the types of Processes in Operating Systems?

The types of Operating System processes are:

1. Operating System Process
2. User Process

67. What are the Data Items in Process Control Block?

The Data Items in Process Control Block are:

1. Process State
2. Process Number
3. Program Counter
4. Registers
5. Memory Limits
6. List of Open Files

68. What are the Files used in the Process Control Block?

The Files used in Process Control Block are:

1. Central Processing Unit (CPU) Scheduling Information
2. Memory Management Information
3. Accounting Information
4. Input Output Status Information

69. What are types of Process Scheduling Techniques in Operating Systems?

The types of Process Scheduling Techniques in Operating Systems are:

1. Pre Emptive Process Scheduling
2. Non Pre Emptive Process Scheduling

70. What is Pre Emptive Process Scheduling in Operating Systems?

In this instance of Pre Emptive Process Scheduling, the OS allots the resources to a process for a predetermined period of time. The process transitions from running state to ready state or from waiting state to ready state during resource allocation. This switching happens because the CPU may assign other processes precedence and substitute the currently active process for the higher priority process.

71. What is Non Pre Emptive Process Scheduling in Operating Systems?

In this case of Non Pre Emptive Process Scheduling, the resource cannot be withdrawn from a process before the process has finished running. When a running process finishes and transitions to the waiting state, resources are switched.

72 What are the methods of Handling Deadlocks?

The methods of handling deadlock are:

1. Deadlock Prevention
2. Deadlock Detection and Recovery
3. Deadlock Avoidance
4. Deadlock Ignorance

73. How can we avoid Deadlock?

We can avoid Deadlock by using Banker's Algorithm.

74. How can we detect and recover the Deadlock occurred in Operating System?

- ✔ First, what we need to do is to allow the process to enter the deadlock state. So, it is the time of recovery.
- ✔ We can recover the process from deadlock state by terminating or aborting all deadlocked processes one at a time.
- ✔ Process Pre Emption is also another technique used for Deadlocked Process Recovery.

75. What is paging in Operating Systems?

Paging is a storage mechanism. Paging is used to retrieve processes from secondary memory to primary memory.

The main memory is divided into small blocks called pages. Now, each of the pages contains the process which is retrieved into main memory and it is stored in one frame of memory.

It is very important to have pages and frames which are of equal sizes which are very useful for mapping and complete utilization of memory.

76. What are the Disk Scheduling Algorithms used in Operating Systems?

The Disk Scheduling Algorithms used in Operating Systems are:

1. First Come First Serve
2. Shortest Seek Time First
3. LOOK
4. SCAN
5. C SCAN

77. What is a File?

A file is a named collection of related information that is recorded on secondary storage. A file contains either programs or data. A file has certain "structure" based on its type.

File attributes: Name, identifier, type, size, location, protection, time, date
File operations: creation, reading, writing, repositioning, deleting, truncating, appending, renaming
File types: executable, object, library, source code etc.

78. What is the information associated with an Open File?

Several pieces of information are associated with an open file which may be:

- ➢ File pointer
- ➢ File open count
- ➢ Disk location of the file Access rights

79. What are the different Accessing Methods of a File?

The different types of accessing a file are:

- **Sequential access:** Information in the file is accessed sequentially
- **Direct access:** Information in the file can be accessed without any particular order.
- **Other access methods**: Creating index for the file, indexed sequential access method (ISAM) etc.

80. What is Directory?

The device directory or simply known as directory records information- such as name, location, size, and type for all files on that particular partition. The directory can be viewed as a symbol table that translates file names into their directory entries.

81. Define UFD and MFD.

In the two-level directory structure, each user has own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. When a job starts the system's master file directory (MFD) is searched. The MFD is indexed by the user name or account number, and each entry points to the UFD for that user.

82. What is a Path Name?

A pathname is the path from the root through all subdirectories to a specified file. In a two-level directory structure a user name and a file name define a path name.

83. What is the importance of Disk Scheduling in Operating Systems?

1. One Input or Output request may be fulfilled by the disk controller at a time, even if several Input or Output requests could come in from other processes. Other Input or Output requests must thus be scheduled and made to wait in the waiting queue.
2. The movement of the disk arm might increase if two or more requests are placed far apart from one another.
3. Since hard disks are among the slower components of the computer system, they must be accessible quickly.

84. What is the purpose of disk scheduling?

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

85. What are the two components of disk Scheduling?

The disk access time has two components – seek time and rotational latency

86. What are the two types of scheduler in OS?

Three types of the scheduler are 1) Long term 2) Short term 3) Medium-term. Long term scheduler regulates the program and select process from the queue and loads them into memory for execution.

87. What is the function of scheduler?

A scheduler is the portion of the operating system that allocates computer resources to processes. It employs scheduling algorithms to determine which processes to allocate to the CPU. The scheduler selects which process is ready to be run next.

88. What is Disk Scheduling Algorithm?

A Process makes the I/O requests to the operating system to access the disk. Disk Scheduling Algorithm manages those requests and decides the order of the disk access given to the requests.

89. Why is dining philosophers problem also called?

The Dining philosopher problem is an example of process synchronization problem. Philosopher is an analogy for process and chopstick for resources, we can try to solve process synchronization problems using this. The solution of Dining Philosopher problem focuses on the use of semaphores.

90. What is the importance of dining philosophers problem?

The dining philosophers problem is a classic example in computer science often used to illustrate synchronization issues and solutions in concurrent algorithm design. It illustrates the challenges of avoiding a system state where progress is not possible, a deadlock.