

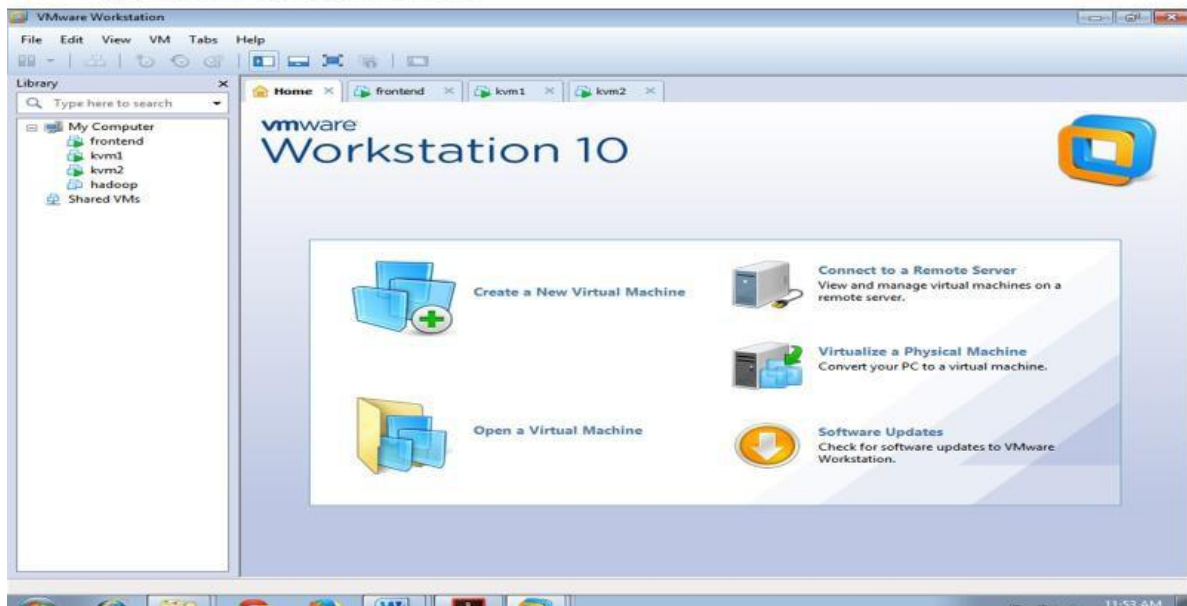
## EX. NO : 1) INSTALLATION OF LINUX USING VIRTUAL MACHINE

### AIM:

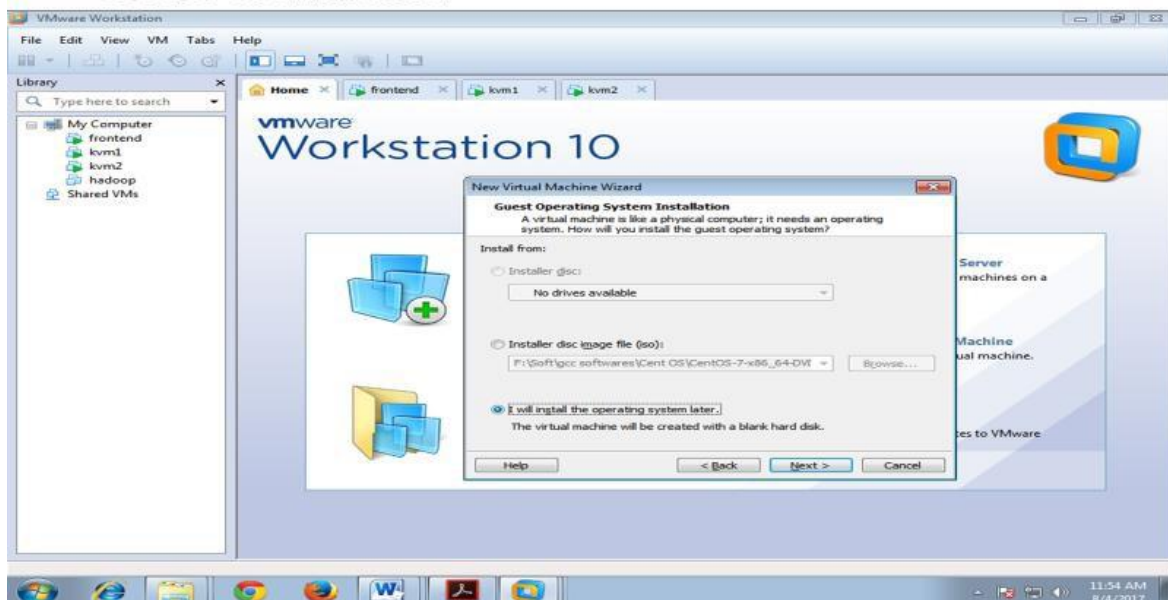
To Install VMware workstation with different flavours of linux on top of windows

### PROCEDURE:

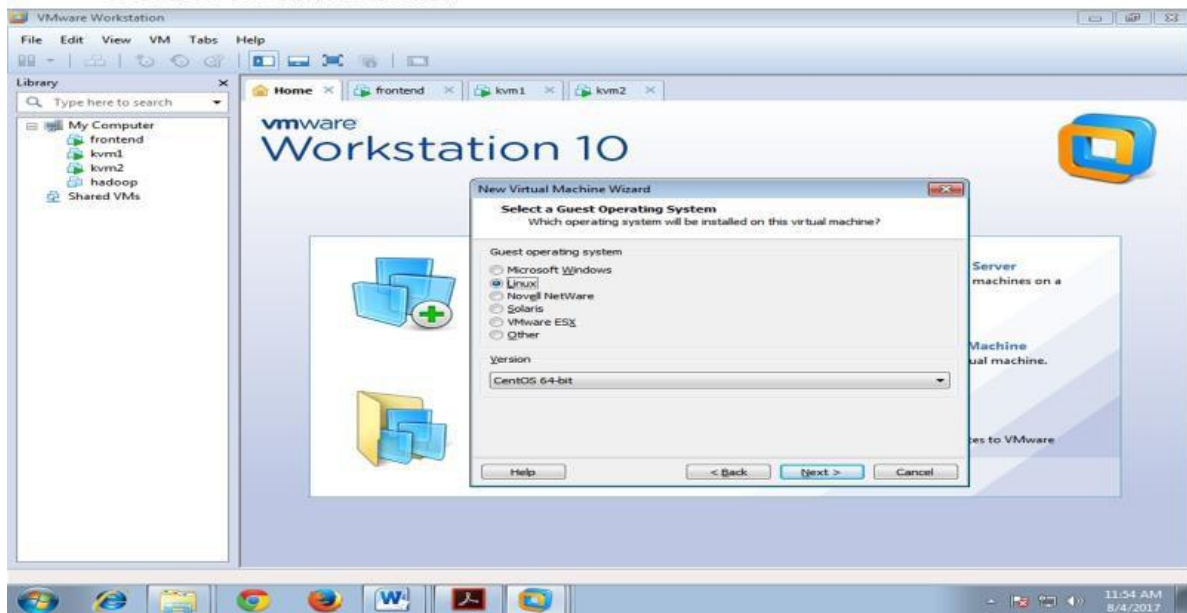
#### VMWare Workstation 10



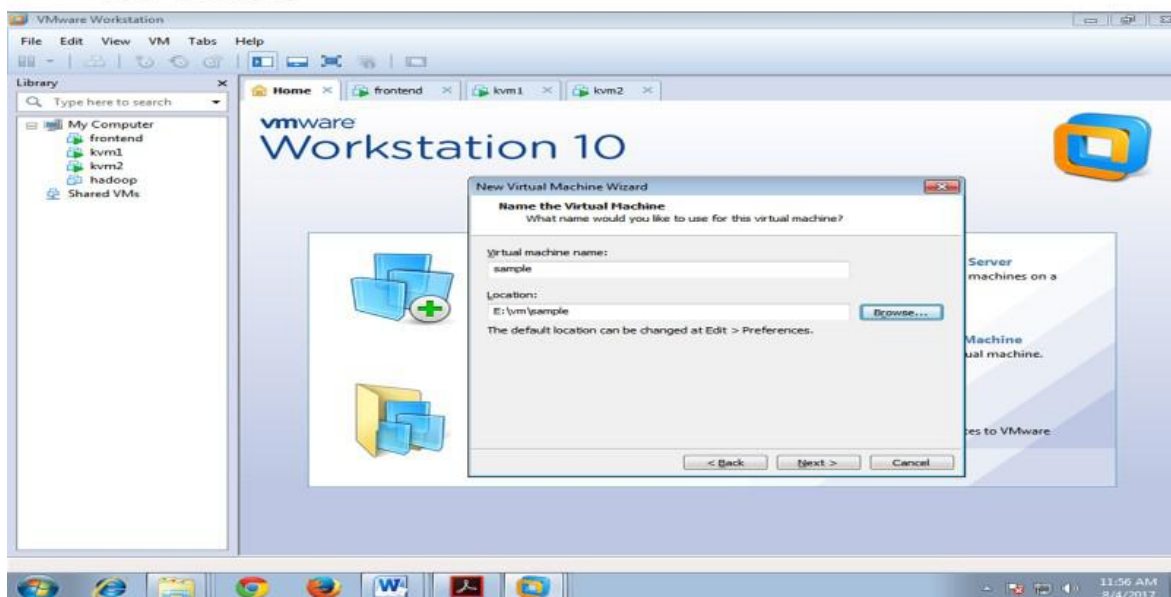
#### GUEST OS Installation



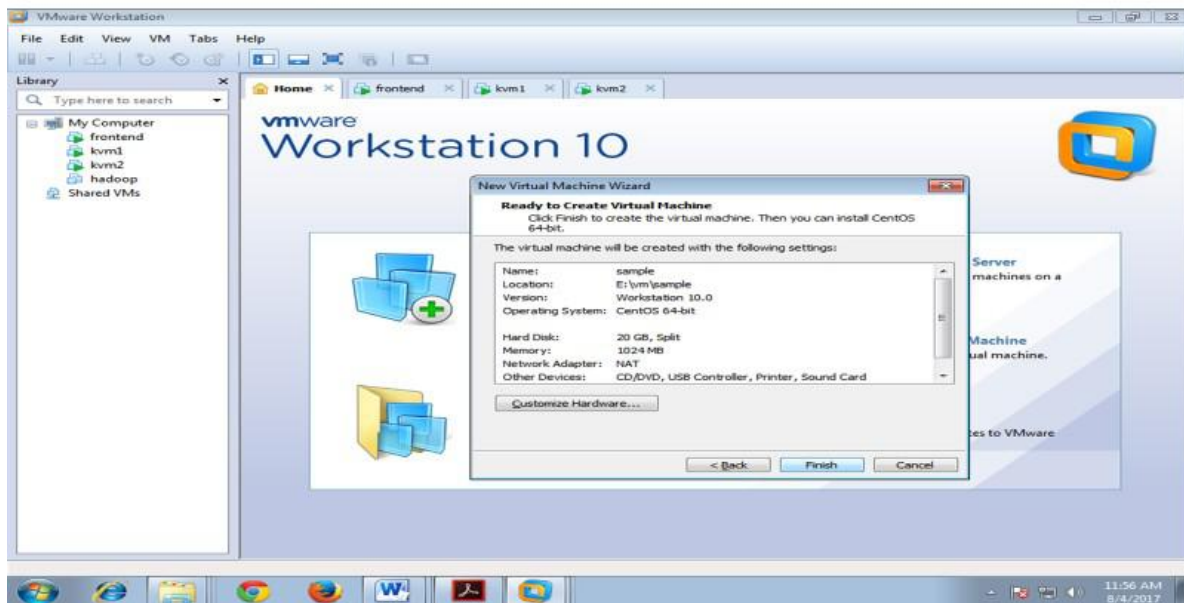
## GUEST OS Installation



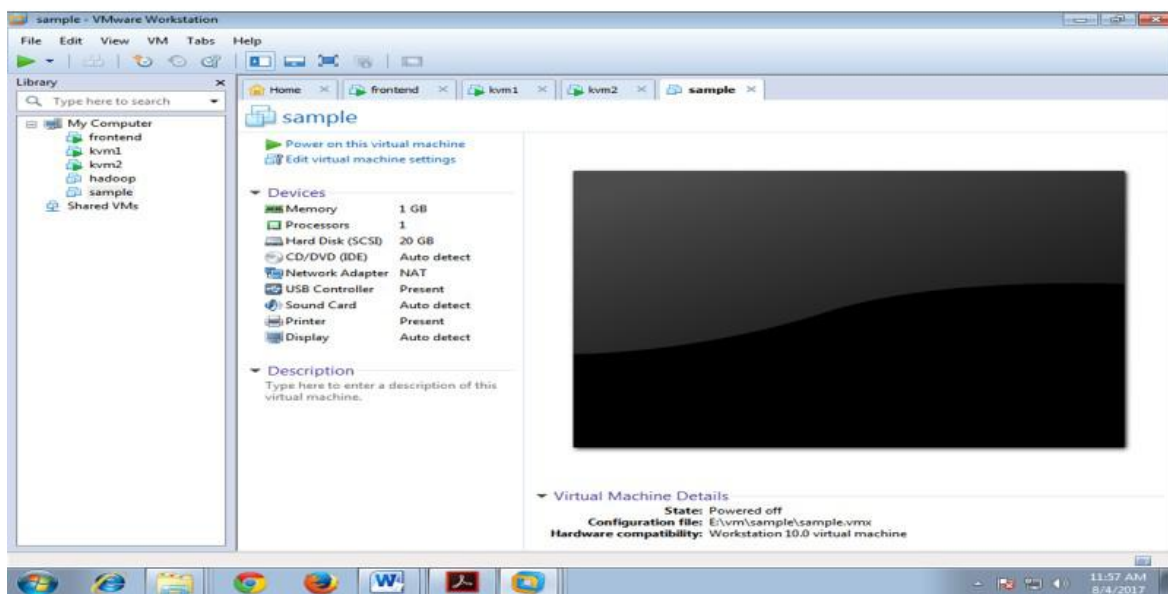
## VM Creation



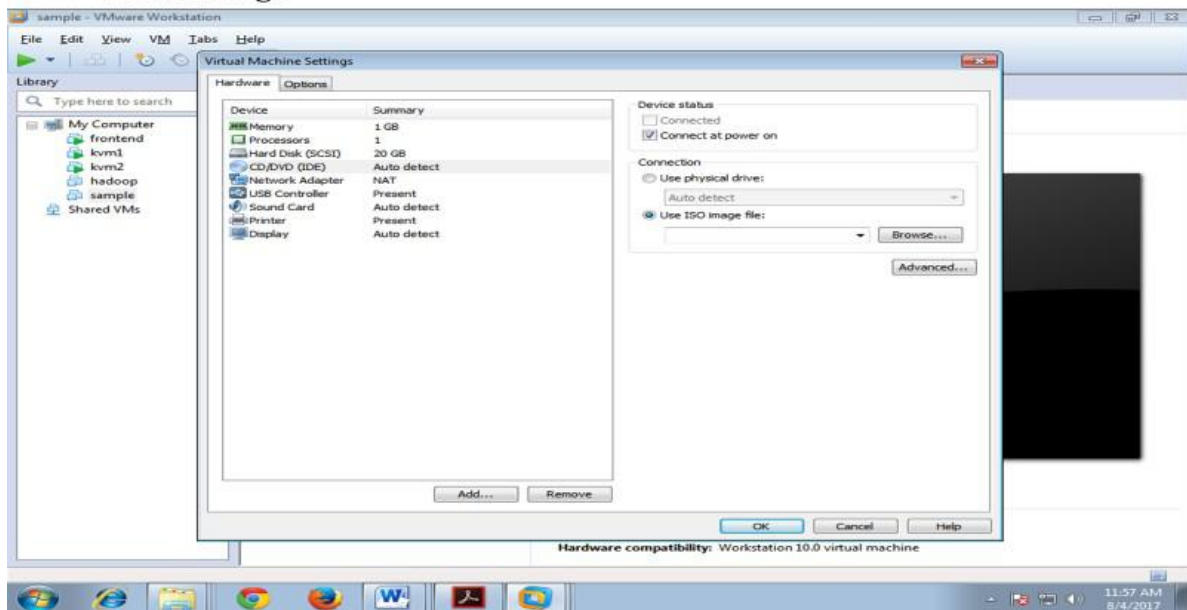
## VM Creation



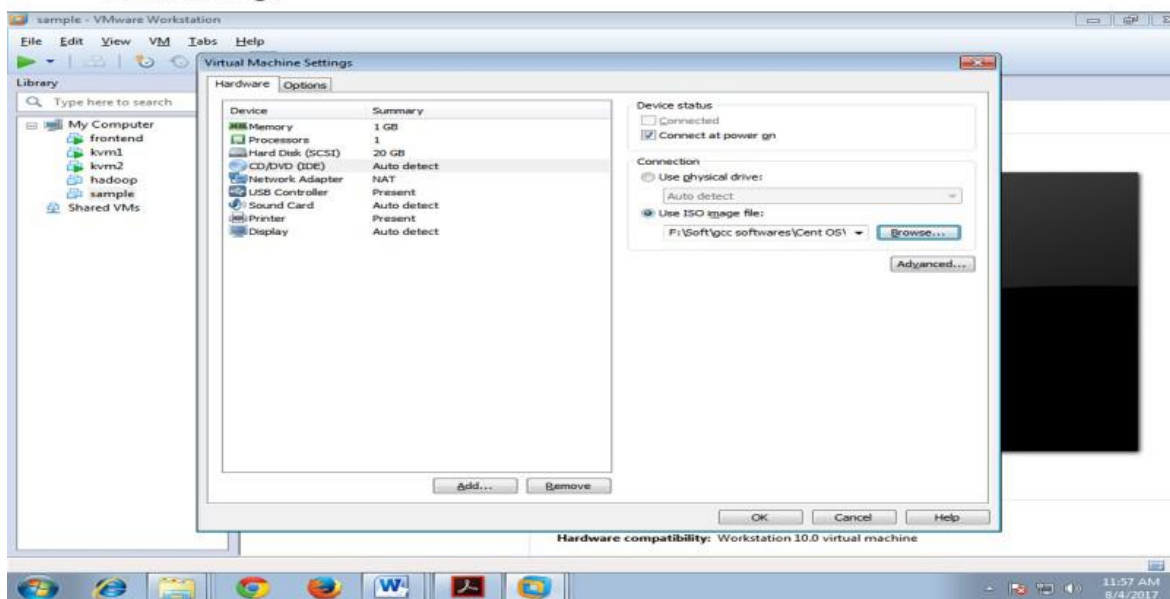
## VM Creation



## VM Settings



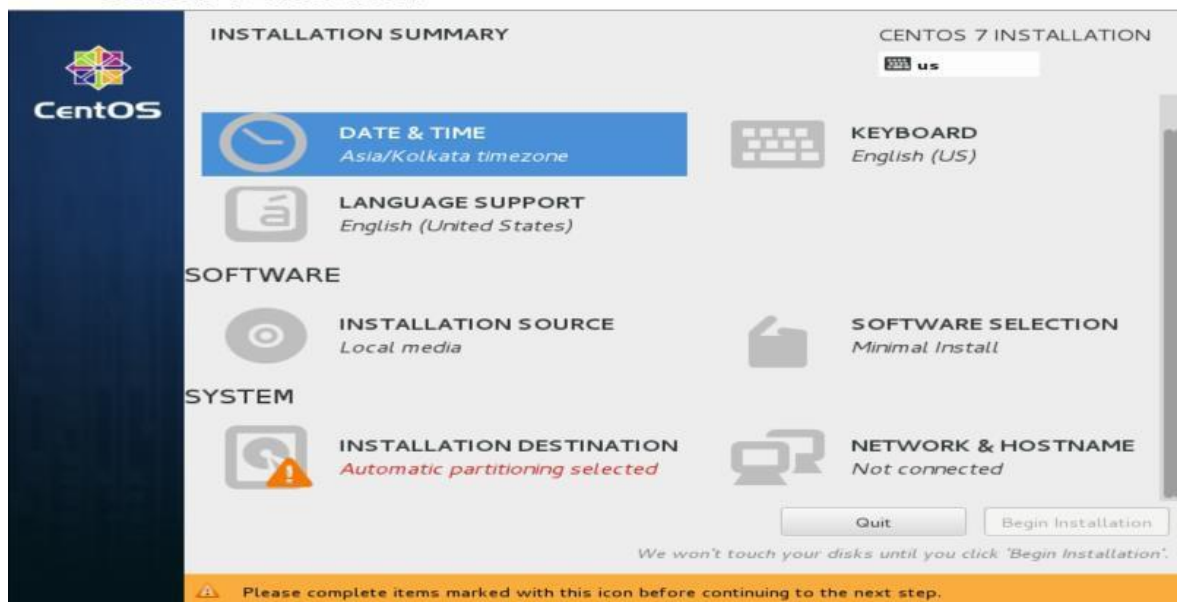
## VM Settings



## CentOS 7 Installation



## CentOS 7 Installation



## CentOS 7 Installation

**CentOS**

### INSTALLATION SUMMARY

**DATE & TIME**  
*Asia/Kolkata timezone*

**KEYBOARD**  
*English (US)*

**LANGUAGE SUPPORT**  
*English (United States)*

**SOFTWARE**

**INSTALLATION SOURCE**  
*Local media*

**SOFTWARE SELECTION**  
*Minimal Install*

**SYSTEM**

**INSTALLATION DESTINATION**  
*Automatic partitioning selected*

**NETWORK & HOSTNAME**  
*Not connected*

[Quit](#) [Begin Installation](#)

*We won't touch your disks until you click 'Begin Installation'.*


 Please complete items marked with this icon before continuing to the next step.

## CentOS 7 Installation

### INSTALLATION DESTINATION

[Done](#)

### CENTOS 7 INSTALLATION


 **us**

[Help!](#)

**Device Selection**

Select the device(s) you'd like to install to. They will be left untouched until you click on the main menu's "Begin Installation" button.

**Local Standard Disks**

**25 GiB**

**VMware, VMware Virtual S**  
sda / 992.5 KiB free

*Disks left unselected here will not be touched.*

**Specialized & Network Disks**

[Add a disk...](#)

*Disks left unselected here will not be touched.*

**Other Storage Options**

**Partitioning**


☒ Automatically configure partitioning. ☐ I will configure partitioning.

[Full disk summary and boot loader...](#)

1 disk selected; 25 GiB capacity; 992.5 KiB free




## CentOS 7 Installation


**CentOS**


**INSTALLATION SUMMARY**

**CENTOS 7 INSTALLATION**  
us


**LOCALIZATION**


**DATE & TIME**  
Asia/Kolkata timezone

**KEYBOARD**  
English (US)


**LANGUAGE SUPPORT**  
English (United States)


**SOFTWARE**

**INSTALLATION SOURCE**  
Local media

**SOFTWARE SELECTION**  
Minimal Install

**SYSTEM**

**INSTALLATION DESTINATION**  
Custom partitioning selected

**NETWORK & HOSTNAME**  
Not connected

Quit

Begin Installation

We won't touch your disks until you click 'Begin Installation'.

## CentOS 7 Installation

**NETWORK & HOSTNAME**  
Done

**CENTOS 7 INSTALLATION**  
us

**Ethernet (eno16777736)**  
Intel Corporation PRO/1000 MT Single Port Adapter

**Ethernet (eno16777736)** **ON**  
Connected

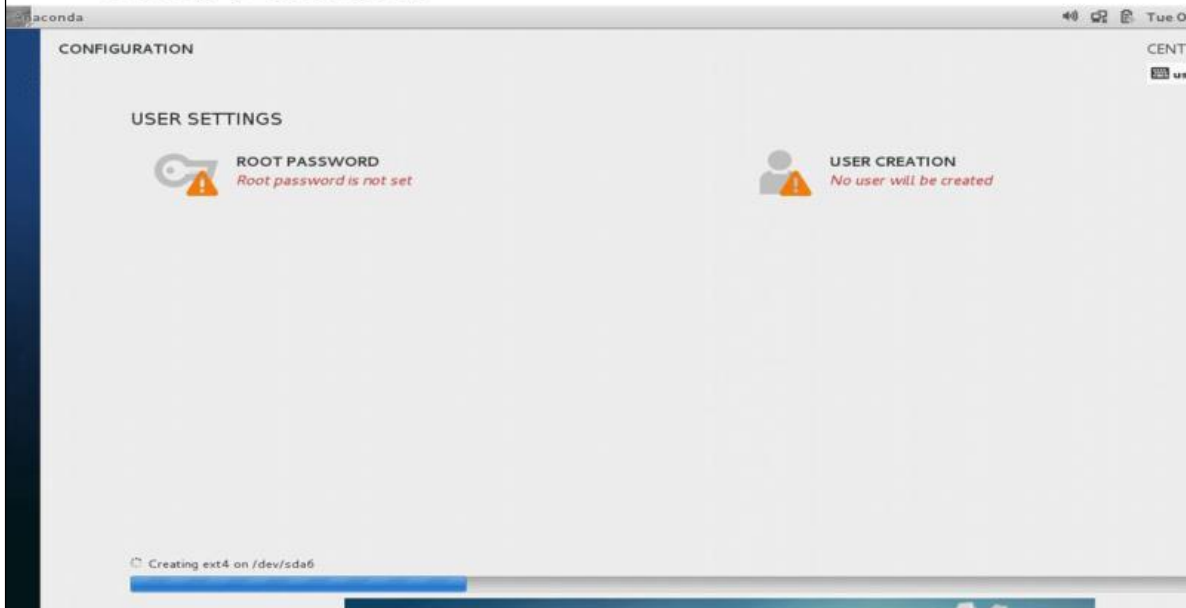
Hardware Address 00:0C:29:4D:74:55  
Speed 1000 Mb/s  
IP Address 192.168.124.147  
Subnet Mask 255.255.255.0  
Default Route 192.168.124.2  
DNS 192.168.124.2

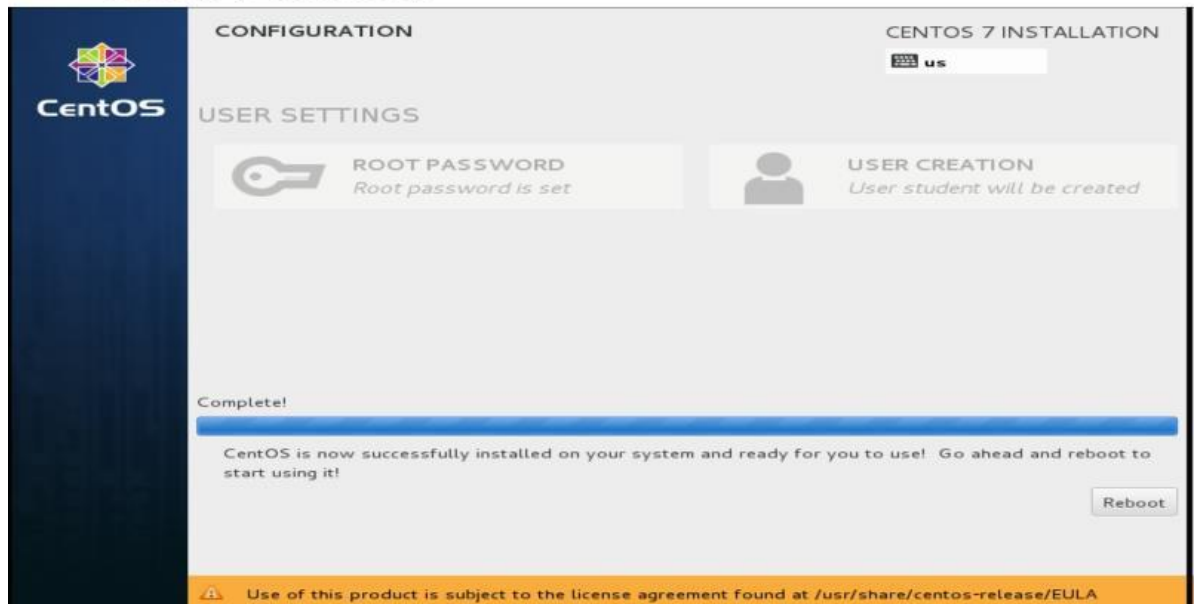
Configure...

Hostname:

## CentOS 7 Installation



## CentOS 7 Installation



### RESULT:

Thus linux on top of windows was successfully installed.



## EX.NO : 2) BASICS OF LINUX AND UNIX COMMANDS.

### AIM:

To execute the basic commands used in the UNIX operating system.

### 1. CAT COMMAND

The cat command is used to create a file.

```
$ cat > filename
```

#### **OUTPUT:**

```
$ cat > sample
```

```
This is 1st yr Mechanical.
```

```
Welcome to our College.
```

```
How are you ?
```

```
This is CP-II Lab.
```

```
^Z
```

```
[1]+  Stopped          cat > sample
```

### DISPLAY CONTENTS OF A FILE

The cat command is also to view the contents of a specified file.

```
$ cat filename
```

#### **OUTPUT:**

```
$ cat sample
```

```
This is 1st yr Mechanical.
```

```
Welcome to our College.
```

```
How are you ?
```

```
This is CP-II Lab.
```

The cat command also serves to concatenate multiple files into a single file.

```
$ cat file1 file2 > file3
```

If file 3 does not exist, it is created and contents of file1 and file2 are merged into a single file called file3.

#### **OUTPUT:**

```
$ cat>first
```

```
Hai
```

```
Hi Welcome
```

```
This is Mechanical dept ^Z
```

```
[2]+  Stopped          cat > first
```

```
[1mecha01@localhost ~]$ cat>second
```

```
hai
```

```
This is S.A.Engineering college ^Z
```

```
[3]+  Stopped          cat > second
```

```
[1mecha01@localhost ~]$ cat first second>third
```

```
[1mecha01@localhost ~]$ cat third
```

```
Hai
```

```
Hi Welcome
```

```
This is Mechanical dept
```

```
hai
```

```
This is S.A.Engineering college
```

There are two options available under the cat command

OPTIONS	USE
-v	Used to display non printing characters.
-n	Used for numbering the lines.

### **OUTPUT:**

```
$ cat -v sample
This is 1st yr Mechanical.
Welcome to our College.
How are you ?
This is CP-II Lab.
```

```
[1mecha01@localhost ~]$ cat -n sample
 1 This is 1st yr Mechanical.
 2 Welcome to our College.
 3 How are you ?
 4 This is CP-II Lab.
```

## **2. WORKING WITH DIRECTORIES**

### **CURRENT WORKING DIRECTORY**

The „pwd“ command is provided to know the current working directory. „pwd“ is abbreviation for print working directory

```
$ pwd
```

### **OUTPUT:**

```
$ pwd
/home/1mecha01
```

### **CREATE A DIRECTORY**

The „mkdir“ command is used to create an empty directory in a disk.

```
$ mkdir dirname
```

### **OUTPUT:**

```
$ mkdir new
[1mecha01@localhost ~]$ cd new
[1mecha01@localhost new]$
```

## **3. DATE AND TIME**

Displays the current date and time.

```
$date +option
```

### **Options**

%D - displays date as mm dd yyyy  
%T - displays time as hh:mm:ss  
%r - displays time in AM or PM  
%m - displays month of the year  
%d - displays day of the month  
%y - displays the year  
%H - displays the hours  
%M - displays the minutes  
%S - displays the seconds  
%w - displays the days of the week

**OUTPUT:**

```
[1mecha01@localhost new]$ date +%D
03/11/13
[1mecha01@localhost new]$ date +%T
15:16:41
[1mecha01@localhost new]$ date +%r
03:16:47 PM
[1mecha01@localhost new]$ date +%m
03
[1mecha01@localhost new]$ date +%d
11
[1mecha01@localhost new]$ date +%y
13
[1mecha01@localhost new]$ date +%H
15
[1mecha01@localhost new]$ date +%M
17
[1mecha01@localhost new]$ date +%S
24
[1mecha01@localhost new]$ date +%w
1
```

**4. CALENDAR**

Displays a simple calendar

**Syntax**

\$cal option

**Options**

- m                – displays the calendar and keeps Monday as first
- month year      – displays the specified month or year
- y                – displays calendar for current year
- I                – displays dates in continuous number
- YYYY            – displays the calendar for the specified year

**OUTPUT:**

\$ cal -m

JAN 2015

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

**5. WHO**

Displays the details of all the users who are logged in.

**Syntax**

\$who

**OUTPUT:**

\$ who

## WHO AM I

Displays the name of the current user

### **Syntax**

\$who am I

### **OUTPUT:**

\$ who am i

1mecha21 pts/0 2013-03-20 14:46 (172.15.128.61)

## 6. MAN

Used to display help on the commands. It displays pages from the linux reference manual, which is installed in the linux operating system.

### **Syntax**

\$ MAN

### **OUTPUT:**

\$ man dir

LS(1) User Commands LS(1)

#### NAME

ls - list directory contents

#### SYNOPSIS

ls [OPTION]... [FILE]...

#### DESCRIPTION

List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort.

Mandatory arguments to long options are mandatory for short options too.

-a, --all

do not ignore entries starting with .

-A, --almost-all

do not list implied . and ..

--author

with -l, print the author of each file

## 7. CLEAR

It is used to clear contents of the screen.

### **Syntax**

\$ clear

## 8. FILES AND DIRECTORIES

Listing of files, directories and sub directories. The ls command can be used to list the contents of the directory. It displays the names of files and directories.

## **Syntax**

\$ ls options

## **Options**

- l - displays a detailed list of files and directories
- a - displays all files
- A - displays files of the current year
- t - displays file type along with name
- r - displays file in sub directories
- R - displays the content of the specified directory
- m - list the files as a single line
- x - sort files horizontally

## **OUTPUT:**

\$ ls

```
8      a.out    cp    driven.sh gross.sh    temperature.sh
add.c   arms.sh  cp1   exno    oe.sh      triangle.sh
address.c big    cp5   exno1   palindrome.sh
alters.c big.sh  deena fact.sh rev.sh
alters.sh claw   Deena fibo.sh sa
ao.sh   colour.sh dig.sh filename swap.sh
```

[1mecha25@localhost ~]\$ ls -l

```
total 120
-rw-rw-r--. 1 1mecha25 1mecha25  0 Mar  4 09:30 8
-rw-rw-r--. 1 1mecha25 1mecha25 239 Mar 18 11:33 add.c
-rw-rw-r--. 1 1mecha25 1mecha25 382 Mar 18 10:55 address.c
-rw-rw-r--. 1 1mecha25 1mecha25 255 Mar 18 10:36 alters.c
-rw-rw-r--. 1 1mecha25 1mecha25 230 Mar 18 10:29 alters.sh
-rw-rw-r--. 1 1mecha25 1mecha25 247 Feb 25 10:56 ao.sh
-rwxrwxr-x. 1 1mecha25 1mecha25 5287 Mar 18 10:55 a.out
-rw-rw-r--. 1 1mecha25 1mecha25 281 Mar 11 11:01 arms.sh
-rw-rw-r--. 1 1mecha25 1mecha25   1 Mar  4 09:26 big
-rw-rw-r--. 1 1mecha25 1mecha25 107 Mar  4 09:36 big.sh
drwxrwxr-x. 2 1mecha25 1mecha25 4096 Feb 18 10:01 claw
-rw-rw-r--. 1 1mecha25 1mecha25 298 Feb 25 11:21 colour.sh
```

\$ ls -a

```
.      a.out      .big.swp Deena    filename  swap.sh
..     arms.sh    .ccache  .Deena.swp .gnome2   temperature.sh
8      .bash_history claw    dig.sh   gross.sh  triangle.sh
```

[1mecha25@localhost ~]\$ ls -A

```
8      arms.sh    .big.swp deena    fact.sh  palindrome.sh add.c    .bash_history .ccache  Deena
fibo.sh rev.sh address.c .bash_logout claw    .Deena.swp filename sa
```

**\$ ls -t**

add.c arms.sh driven.sh oe.sh temperature.sh deena cp1 a.out rev.sh palindrome.sh colour.sh  
triangle.sh exno1 exno

**[1mecha25@localhost ~]\$ ls -r**

triangle.sh palindrome.sh fact.sh Deena colour.sh a.out add.c temperature.sh oe.sh exno1  
deena claw ao.sh 8

**[1mecha25@localhost ~]\$ ls -R**

..  
8 a.out cp driven.sh gross.sh temperature.sh add.c arms.sh cp1 exno oe.sh  
triangle.sh address.c big cp5 exno1 palindrome.sh

./claw:

./sa:

**\$ ls -m**

8, add.c, address.c, alters.c, alters.sh, ao.sh, a.out, arms.sh, big, big.sh, claw, colour.sh, cp, cp1, cp5, deena, Deena, dig.sh, driven.sh, exno, exno1, fact.sh, fibo.sh, filename, gross.sh, oe.sh, palindrome.sh, rev.sh, sa, swap.sh, temperature.sh, triangle.sh

**[1mecha25@localhost ~]\$ ls -x**

8 add.c address.c alters.c alters.sh ao.sh a.out, arms.sh big big.sh claw  
colour.sh cp cp1, cp5 deena Deena dig.sh driven.sh exno exno1, fact.sh fibo.sh  
filename gross.sh oe.sh palindrome.sh rev.sh, sa swap.sh temperature.sh triangle.sh

## 8. PIPES AND FILTERS

A filter is a program that takes input from standard input process it and sends the output to the standard output.

Filters are used for:

1. Extract lines containing a specific pattern.
2. Short the contents of the file.
3. Replace existing characters with others.

Some filters are:

1. GREP
2. WC
3. CUT
4. TR
5. SORT

### GREP

It searches line by line for the specific pattern and outputs a line that matches the pattern. GREP stands for globally for Regular Expression and Printout.

#### Syntax

Grep[options] pattern[file]

If the file argument is omitted then grep will read from standard input.

Some of the characters used:

1. The carrat (^) – matches the beginning of the line



2. The dollar (\$) – matches the end of the line.
3. The period (.) – matches a single character.
4. The astric (\*) – matches zero or more occurrences of the previous character.

### **Options**

- V – displays only the lines which does not match.
- C – displays the count of lines that matches.
- n – displays the line that matches the pattern along.
- l – displays the lines that matches the pattern.

### **OUTPUT:**

```
$ grep -V
grep (GNU grep) 2.5.1
```

Copyright 1988, 1992-1999, 2000, 2001 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```
$ cat file
hai
hi
haii
hello
hello hai
```

```
$ grep -n ^h file
1:hai
2:hi
3:haii
4:hello
$ grep -n i$ file
2:hi
3:haii
5:hello hai
```

```
$ grep -n .e file
4:hello
5:hello hai
```

```
$ grep -n h*i file
1:hai
2:hi
3:haii
5:hello hai
```

```
$grep -c ^h file
5
```

```
$grep -c i$ file
3
```

```
$ grep -c .e file
2
```

```
$ grep -c h*i file
4
```

## WC FILTER

Used to count the number of lines, words or characters in a file.

### Syntax

```
$ wc[./wc][file_name]
```

### OUTPUT:

```
$ wc gross.sh
13 67 347 gross.sh
[lmecha25@localhost ~]$ wc palindrome.sh
18 50 291 palindrome.sh
```

## CUT FILTER

Used to extract specific columns from the output of certain commands.

### Syntax

```
$ cut[options][file_name]
```

### Options

-d<column delimiter>	-specifies the column delimiter
-f<column number>	-displays the specified number
-c<character number>	-displays the specified character

### OUTPUT:

```
$ cut -c 2 file
a
i
a
e
e
```

## TR FILTER

Used to translate one set of characters from the standard input to another.

### OUTPUT:

```
$ tr -d sample
p3rfm
3rf
eiffur
iffur
epfomri
fori
epfoift3
foift3
```

## SORT FILTER

It arranges the input taken from they standard input in an alphabetic order.

## Options

- n – arranges the number of alphabets in ascending order
- t – If we use a field separator other than a separator or a blank space
- r – it does sorting in reverse
- b -sorting in alphabetical order ignoring case
- u – it removes duplicate lines
- o -Output can be send to files with this

## OUTPUT:

\$ cat > sample

hai  
green  
yello  
blue  
white  
black  
green

\$ sort -n sample

black  
blue  
green  
green  
hai  
hello  
white

\$ sort -r sample

yello  
white  
hello  
hai  
green  
green  
blue

\$ sort -b sample

( IGNORING CASE- SORTING ALPHABETICAL ORDER)

black  
blue  
green  
green  
hai  
hello  
white

\$ sort -u sample

black  
blue  
green  
hai  
hello  
white  
yello

## **PIPES**

It is a mechanism in which the output of one command is send as an input of other.

## **TEE COMMAND**

Linux discards the intermediate output in a pipe, sometimes it is necessary to pipe the standard output of the command to another command and also saves it on disk for later use. It is also possible that we may want the output of the particular command in a long pipe line to be shared for later use.

## **9. REDIRECTION**

Input can be taken from sources other than standard input and can be passed to any source other than the standard output. It is called as redirection.

1. Input redirection
2. Output redirection
3. Error

## **INPUT REDIRECTION**

It implies taking input from a source other than the keyboard.

### **Syntax**

```
$ command<filename>
```

## **OUTPUT REDIRECTION**

It implies the redirecting the output to a source other than the monitor.

### **Syntax:**

```
$ command<filename>
```

## **ERROR**

It is possible to redirect both standard input and standard output for a command.

### **Syntax**

```
$ command<source><destination>
```

## **RESULT:**

Thus Basic Commands Used in UNIX operating system is executed successfully.

### EX.NO:3

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

#### i) PROGRAM USING SYSTEM CALL fork()

**AIM :** To write the program to create a Child Process using system call fork().

#### **ALGORITHM :**

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

#### **SYSTEM CALLS USED: 1. fork()**

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

**Syntax : fork()**

#### **2. execlp()**

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

**Syntax : execlp()**

#### **3. wait()**

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

**Syntax : wait( NULL)**

#### **4. exit()**

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

**Syntax: exit(0)**

#### **PROGRAM CODING :**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc,char *arg[])
{
int pid;
```

```
pid=fork();
if(pid<0)
{
printf("fork failed");
exit(1);
}
else if(pid==0)
{
execlp("whoami","ls",NULL);
exit(0);
}
else
{
printf("\n Process id is -%d\n",getpid());
wait(NULL);
exit(0);
}
}
```

**OUTPUT:**

[cse6@localhost Pgm]\$ cc prog4a.c

[cse6@localhost Pgm]\$ ./a.out

**RESULT:**

Thus the program was executed and verified successfully



### **EX.NO : 3)ii) PROGRAM USING SYSTEM CALLS getpid() & getppid()**

#### **AIM :**

To write the program to implement the system calls getpid() and getppid().

#### **ALGORITHM :**

- Step 1 : Declare the variables pid , parent pid , child id and grand chil id.
- Step 2 : Get the child id value using system call fork().
- Step 3 : If child id value is less than zero then print as “error at fork() child”.
- Step 4 : If child id !=0 then using getpid() system call get the process id.
- Step 5 : Print “I am parent” and print the process id.
- Step 6 : Get the grand child id value using system call fork().
- Step 7 : If the grand child id value is less than zero then print as “error at fork() grand child”.
- Step 8 : If the grand child id !=0 then using getpid system call get the process id.
- Step 9 : Assign the value of pid to my pid.
- Step 10 : Print “I am child” and print the value of my pid.
- Step 11 : Get my parent pid value using system call getppid().
- Step 12 : Print “My parent”s process id” and its value.
- Step 13 : Else print “I am the grand child”.
- Step 14 : Get the grand child”s process id using getpid() and print it as “my process id”. Step
- 15 : Get the grand child”s parent process id using getppid() and print it as “my parent”s process id

#### **SYSTEM CALLS USED :**

##### **1.getpid( )**

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

##### **2.getppid( )**

Used to get particular process parent”s id value.

##### **3.perror( )**

Indicate the process error.

#### **PROGRAM CODING:**

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main( )
{
int pid;
pid=fork( ); if(pid==
-1)
{
```

```

perror("fork failed"); exit(0);
}
if(pid==0)
{
printf("\n Child process is under execution");
printf("\n Process id of the child process is %d", getpid()); printf("\n
Process id of the parent process is %d", getppid());
}
else
{
printf("\n Parent process is under execution");
printf("\n Process id of the parent process is %d", getpid());
printf("\n Process id of the child process in parent is %d", pid());
printf("\n Process id of the parent of parent is %d", getppid());
}
return(0);
}

```

### **OUTPUT:**

```

Child process is under execution
Process id of the child process is 9314
Process id of the parent process is 9313 Parent
process is under execution
Process id of the parent process is 9313
Process id of the child process in parent is 9314 Process
id of the parent of parent is 2825

```

### **RESULT:**

Thus the program was executed and verified successfully

### **EX.NO : 3)iii) PROGRAM USING SYSTEM CALLS opendir() readdir() closedir()**

#### **AIM :**

To write the program to implement the system calls opendir( ), readdir( ).

#### **ALGORITHM :**

Step 1 : Start.

Step 2 : In the main function pass the arguments.

Step 3 : Create structure as stat buff and the variables as integer.

Step 4 : Using the for loop,initialization

#### **SYSTEM CALLS USED:**

##### **1.opendir( )**

Open a directory.

##### **2.readdir( )**

Read a directory.

##### **3.closedir( )**

Close a directory.

#### **PROGRAM CODING:**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/dir.h>
void main(int age,char *argv[])
{
    DIR *dir;
    struct dirent *rddir;
    printf("\n Listing the directory content\n");
    dir=opendir(argv[1]);
    while((rddir=readdir(dir))!=NULL)
    {
        printf("%s\t\n",rddir->d_name);
    }
    closedir(dir);
}
```

#### **OUTPUT:**

```
RP
roshi.c
first.c
pk6.c f2
abc FILE1
```

#### **RESULT:**

Thus the program was executed and verified successfully

### **EX.NO : 3 iv) PROGRAM USING SYSTEM CALL exec( )**

**AIM :** To write the program to implement the system call exec( ).

#### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Print execution of exec system call for the date Unix command.
- Step 3 : Execute the execlp function using the appropriate syntax for the Unix command date.
- Step 4 : The system date is displayed.

#### **SYSTEM CALL USED :**

##### **1.execlp( )**

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

**Syntax : execlp( )**

**PROGRAM CODING:** #include<stdio.h>  
#include<unistd.h> main( )  
{  
printf("\n exec system call");  
printf("displaying the date");  
execlp( "/bin/date", "date", 0);  
}

#### **RESULT:**

Thus the program was executed and verified successfully

#### **OUTPUT:**

Sat Dec 14 02 : 57 : 38 IST 2010

### **EX.NO : 3 v) PROGRAM USING SYSTEM CALLS wait() & exit()**

**AIM :** To write the program to implement the system calls wait() and exit().

#### **ALGORITHM :**

- Step 1 : Declare the variables pid and i as integers.
- Step 2 : Get the child id value using the system call fork().
- Step 3 : If child id value is less than zero then print “fork failed”.
- Step 4 : Else if child id value is equal to zero , it is the id value of the child and then start the child process to execute and perform Steps 6 & 7.
- Step 5 : Else perform Step 8.
- Step 6 : Use a for loop for almost five child processes to be called. Step
- 7 : After execution of the for loop then print “child process ends”.
- Step 8 : Execute the system call wait() to make the parent to wait for the child process to get over.
- Step 9 : Once the child processes are terminated , the parent terminates and hence print “Parent process ends”.
- Step 10 : After both the parent and the child processes get terminated it execute the wait() system call to permanently get deleted from the OS.

#### **SYSTEM CALL USED :**

##### **1. fork ()**

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

**Syntax: fork ()**

##### **2. wait ()**

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

**Syntax: wait (NULL)**

##### **3. exit ()**

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

**Syntax: exit(0)**

#### **PROGRAM CODING:**

```
#include<stdio.h> #include<unistd.h>
main()
{
    int i, pid;
    pid=fork();
    if(pid== -1)
    {
```

```

perror("fork failed");
exit(0);
}
else if(pid==0)
{
printf("\n Childprocessstarts");
for(i=0; i<5; i++)
{
printf("\n Child process %d is called", i);
}
printf("\n Child process ends");
}
else
{
wait(0);
printf("\n Parent process ends");
}exit(0);}

```

### **OUTPUT:**

```

Child process starts Child
process 0 is called Child
process 1 is called Child
process 2 is called Child
process 3 is called Child
process 4 is called Child
process ends Parent process
ends

```

### **RESULT:**

Thus the program was executed and verified successfully



### **EX. NO: 3 vi) PROGRAM USING SYSTEM CALL stat()**

**AIM:** To write the program to implement the system call stat().

#### **ALGORITHM :**

- Step 1 : Include the necessary header files to execute the system call stat().
- Step 2 : Create a stat structure thebuf. Similarly get the pointers for the two structures passwd and group.
- Step 3 : Declare an array named path[20] of character type to get the input file along with its extension and an integer i.
- Step 4 : Get the input file along with its extension.
- Step 5 : If not of stat of a particular file then do the Steps 6 to 18.
- Step 6 : Print the file's pathname as file's name along with its extension.
- Step 7 : To check the type of the file whether it is a regular file or a directory use S\_ISREG().
- Step 8 : If the above function returns true value the file is an regular file else it is directory.
- Step 9 : Display the file mode in octal representation using thebuf.st\_mode.
- Step 10 : Display the device id in integer representation using thebuf.st\_dev.
- Step 11 : Display the user id in integer representation using thebuf.st\_uid.
- Step 12 : Display the user's pathname.
- Step 13 : Display the group id in integer representation using thebuf.st\_gid.
- Step 14 : Display the group's pathname.
- Step 15 : Display the size of the file in integer representation using thebuf.st\_size.
- Step 16 : Display the last access in time and date format using ctime(&thebuf.st\_atime).
- Step 17 : Display the last modification in time and date format using ctime(&thebuf.st\_atime).
- Step 18 : Display the last status in time and date format using ctime(&thebuf.st\_atime).
- Step 19 : If Step 5 fails then print "Cannot get the status details for the given file".

#### **SYSTEM CALLS USED :**

- 1. stat()
- 2. creat()
- 3. open()
- 4. fstat(),

#### **PROGRAM CODING:**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
main()
{
```

```

int fd1,fd2,n;
char source[30],ch[5];
struct stat s,t,w;
fd1=creat("text.txt",0644);
printf("Enter the file to be copied\n");
scanf("%s",source);
fd2=open(source,0);
if(fd2==-1)
{
perror("file doesnot exist");
exit(0);
}
while((n=read(fd2,ch,1))>0)
write(fd1,ch,n);
close(fd2);
stat(source,&s);
printf("Source file size=%d\n",s.st_size);
fstat(fd1,&t);
printf("Destination file size =%d\n",t.st_size);
close(fd1);
}

```

### **RESULT:**

Thus the program was executed successfully.

### **EX.NO : 3 vii PROGRAM USING SYSTEM CALLS open(), read() & write( )**

**AIM :** To write the program to implement the system calls open( ),read( ) and write( ).

#### **ALGORITHM :**

- Step 1 : Declare the structure elements.
- Step 2 : Create a temporary file named temp1.
- Step 3 : Open the file named “test” in a write mode.
- Step 4 : Enter the strings for the file.
- Step 5 : Write those strings in the file named “test”.
- Step 6 : Create a temporary file named temp2.
- Step 7 : Open the file named “test” in a read mode.
- Step 8 : Read those strings present in the file “test” and save it in temp2.
- Step 9 : Print the strings which are read.

#### **SYSTEM CALLS USED :**

1. **open()**
2. **read()**
3. **write()**
4. **close()**
5. **gets()**
6. **lseek()**

#### **PROGRAM CODING:**

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
main()
{
int fd[2];
char buf1[25]= "just a test\n"; char
buf2[50];
fd[0]=open("file1", O_RDWR);
fd[1]=open("file2", O_RDWR);
write(fd[0], buf1, strlen(buf1));
printf("\n Enter the text now....");
gets(buf1);
write(fd[0], buf1, strlen(buf1));
lseek(fd[0], SEEK_SET, 0);
read(fd[0], buf2, sizeof(buf1));
write(fd[1], buf2, sizeof(buf2));
close(fd[0]);
close(fd[1]);
printf("\n");
return 0;
}
```

**OUTPUT:**

Enter the text now....progress

Cat file1 Just a

test progress

Cat file2 Just a test progress

**RESULT:**

Thus the program was executed successfully

#### **EX. NO: 4 i) SIMULATION OF UNIX COMMANDS “ls”**

**AIM:** To write a C program to simulate the operation of “ls” commands in Unix.

#### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Define the buffer size as 1024.
- Step 3 : Get the file name which has been created already.
- Step 4 : Open the file in read mode.
- Step 5 : Read the contents of the file and store it in the buffer.
- Step 6 : Print the contents stored in the buffer.
- Step 7 : Close the file.

#### **PROGRAM CODING:**

```
#include<stdio.h>
#include<dirent.h>
int main()
{
    struct dirent **namelist;
    int n,i;
    char pathname[100];
    scanf("%s",&pathname); getcwd(pathname);
    n=scandir(pathname,&namelist,0,alphasort);
    if(n<0)
        printf("Error");
    else
        for(i=0;i<n;i++)
            printf("%s\n",namelist[i]->d_name);
}
```

#### **RESULT:**

Thus the program was executed successfully

## **EX. NO: 4 ii SIMULATION OF UNIX COMMANDS “grep”**

**AIM:** To write a C program to simulate the operation of “grep” commands in Unix.

### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Define the buffer size.
- Step 3 : Get the file name which has been created already.
- Step 4 : Open the file in read mode.
- Step 5 : Read the contents of the file and store it in the buffer.
- Step 6 : Print the contents stored in the buffer.
- Step 7 : Close the file.

### **PROGRAM CODING:**

```
#include<stdio.h>
#include<stdlib.h>
main()
{
FILE *fp;
char c[100],pat[10];
int l,i,j=0,count=0,len,k,flag=0;
printf("\n enter the pattern");
scanf("%s",pat);
len=strlen(pat);
fp=fopen("nn.txt","r");
while(!feof(fp))
{
fscanf(fp,"%s",c);
l=strlen(c); count++;
for(i=0;i<count;i++)
{
if(c[i]==pat[j])
{
flag=0;
for(k=1;k<i;k++)
{
if(c[i+k]!=pat[k])
flag=1;
}
if(flag==0)
printf("\n the pattern %s is present in word %d",pat,count);
}}}}
}
```

### **RESULT:**

Thus the program was executed successfully



**EX. NO: 5 ) SHELL PROGRAMMIG****i BASIC ARITHMETIC OPERATION USIG SHELL PROGRAMMIG**

**AIM:** To write a shell program to solve arithmetic operation.

**ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : get the input
- Step 3 : perform the arithmetic calculation.
- Step 4 : print the result.
- Step 5 :stop the exeution.

**PROGRAM CODING:**

```
#!/bin/bash
echo "enter the a vale"
read a
echo "enter b value"
read b
c=`expr $a + $b`
echo "sum:"$c
c=`expr $a - $b`
echo "sub:"$c
c=`expr $a \* $b`
echo "mul:"$c
c=`expr $a / $b`
echo "div:"$c
```

**OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
Enter the a vale
10
Enter b value
50
sum:60
sub:-40
mul:500
div:0
```

**RESULT:**

Thus the program was executed successfully

## **EX. NO: 5 ) ii NUMBER CHECKIG USIG SHELL PROGRAM**

**AIM:** To write a shell program to check whether the number is odd or even.

### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : get the input
- Step 3 : perform the division by 2.
- Step 4 : print the result.
- Step 5 :stop the execution.

### **PROGRAM CODING:**

```
#!/bin/bash
num="1 2 3 4 5 6 7 8"
for n in $num
do
q=`expr $n % 2`
if [ $q -eq 0 ]
then
echo "even no"
continue
fi
echo "odd no"
done
```

### **OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
odd no
even no
odd no
even no
odd no
even no
odd no
even no
```

### **RESULT:**

Thus the program was executed successfully

### **EX. NO: 5 ) iii MULTIPLICATION TABLE USING SHELL PROGRAM**

**AIM:** To write a shell program to check whether the number is odd or even.

**ALGORITHM :**

**ALGORITHM :**

Step 1 : Include the necessary header files.

Step 2 : get the input

Step 3 : perform the multiplication .

Step 4 : print the result.

Step 5 :stop the execution.

#### **PROGRAM CODING:**

```
#!/bin/bash
echo " which table you want"
read n
for((i=1;i<10;i++))
do
echo $i "*" $n "=" `expr $i \* $n`
done
```

#### **OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
```

```
which table you want
```

```
5
```

```
1 * 5 = 5
```

```
2 * 5 = 10
```

```
3 * 5 = 15
```

```
4 * 5 = 20
```

```
5 * 5 = 25
```

```
6 * 5 = 30
```

```
7 * 5 = 35
```

```
8 * 5 = 40
```

```
9 * 5 = 45
```

### **EX. NO: 5 ) iv USING WHILE LOOP IN SHELL PROGRAM**

**AIM:** To write a shell program to print the number from 1 to 10 using while loop.

#### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Define the buffer size as 1024.
- Step 3 : Get the file name which has been created already.
- Step 4 : Open the file in read mode.
- Step 5 : Read the contents of the file and store it in the buffer.
- Step 6 : Print the contents stored in the buffer.
- Step 7 : Close the file.

#### **PROGRAM CODING:**

```
#!/bin/bash
a=1
while [ $a -lt 11 ]
do
echo "$a"
a=`expr $a + 1`
done
```

#### **OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
1
2
3
4
5
6
7
8
9
10
```

#### **RESULT:**

Thus the program was executed successfully

## **EX. NO:5 ) v USING IF STATEMENT IN SHELL PROGRAMING**

**AIM:** to write a shell program to use simple if statement .

### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Define the buffer size as 1024.
- Step 3 : Get the file name which has been created already.
- Step 4 : Open the file in read mode.
- Step 5 : Read the contents of the file and store it in the buffer.
- Step 6 : Print the contents stored in the buffer.
- Step 7 : Close the file.

### **PROGRAM CODING:**

```
#!/bin/bash
for var1 in 1 2 3
do
for var2 in 0 5
do
if [ $var1 -eq 2 -a $var2 -eq 0 ]
then
continue
else
echo "$var1 $var2"
fi
done
```

### **OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
1 0
1 5
2 5
3 0
3 5
```

### **RESULT:**

Thus the program was executed successfully

## **EX. NO: 5) vi SIMPLE FUNCTION IN SHELL PROGRAMING**

**AIM:** to write a shell program to add a two number using function .

### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Define the buffer size as 1024.
- Step 3 : Get the file name which has been created already.
- Step 4 : Open the file in read mode.
- Step 5 : Read the contents of the file and store it in the buffer.
- Step 6 : Print the contents stored in the buffer.
- Step 7 : Close the file.

### **PROGRAM CODING:**

```
#!/bin/bash
add()
{
c=`expr $1 + $2`
echo "addition = $c"
}
add 5 10
```

### **OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
addition = 15
```

### **RESULT:**

Thus the program was executed successfully

## **EX. NO: 5) vii SWITCH STATEMENT IN SHELL PROGRAMING**

**AIM:** to write a shell program to add a two number using function .

### **ALGORITHM :**

- Step 1 : Include the necessary header files.
- Step 2 : Define the buffer size as 1024.
- Step 3 : Get the file name which has been created already.
- Step 4 : Open the file in read mode.
- Step 5 : Read the contents of the file and store it in the buffer.
- Step 6 : Print the contents stored in the buffer.
- Step 7 : Close the file.

### **PROGRAM CODING**

```
#!/bin/bash
ch='y'
while [ $ch = 'y' ]
do
echo "enter your choice"
echo "1 no of user logged on"
echo "2 print calender"
echo "3 print date"
read d
case $d in
1) who | wc -l;;
2) cal 20;;
3) date;;
*) break;;
esac
echo "do you wish to continue (y/n)"
read ch
```

### **OUTPUT:**

```
[2mecse25@rhes3linux ~]$ sh case2.sh
enter your choice
1 no of user logged on
2 print calender
3 print date
Thu Apr 4 11:18:20 IST 2013
do you wish to continue
(y/n) n
```

### **RESULT:**

Thus the program was executed successfully

## **EX.NO:6) a CPU SCHEDULING: ROUND ROBIN SCHEDULING**

**AIM :** To write the program to simulate the Round Robin program.

### **PROBLEM DESCRIPTION:**

CPU scheduler will decide which process should be given the CPU for its execution. For this it use different algorithm to choose among the process. One among that algorithm is Round robin algorithm.

In this algorithm we are assigning some time slice .The process is allocated according to the time slice, if the process service time is less than the time slice then process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue .If the CPU burst of the currently running process is longer than time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed and the process will be put at the tail of the ready queue.

### **ALGORITHM:**

Step 1: Initialize all the structure elements

Step 2: Receive inputs from the user to fill process id,burst time and arrival time.

Step 3: Calculate the waiting time for all the process id.

- i) The waiting time for first instance of a process is calculated as:  
 $a[i].waittime = count + a[i].arrivt$
- ii) The waiting time for the rest of the instances of the process is calculated as:
  - a) If the time quantum is greater than the remaining burst time then waiting time is calculated as:  
 $a[i].waittime = count + tq$
  - b) Else if the time quantum is greater than the remaining burst time then waiting time is calculated as:

$a[i].waittime = count - remaining\ burst\ time$

Step 4: Calculate the average waiting time and average turnaround time

Step 5: Print the results of the step 4.

### **PROGRAM CODING:**

```
#include<stdio.h>
void main()
{
int i,tbt=0,nop,ts=0,flag[20], rem[20];
int from,wt[20],tt[20],b[20], twt=0,ttt=0;
int dur;
float awt,att;
printf("Enter no. of Processes: ");
scanf("%d",&nop);
printf("Enter the time slice: ");
scanf("%d",&ts);
printf("Enter the Burst times..\n");
for(i=0;i<nop;i++)
```



```

{
wt[i]=tt[i]=0;
printf("P%d\t: ",i+1);
scanf("%d",&b[i]);
rem[i]=b[i];
tbt+=b[i]; flag[i]=0;
}
from=0;
i=0;
printf("\n\t Gantt Chart");
printf("\n ProcessID\tFrom Time\tTo Time\n");
while(from<tbt)
{
if(!flag[i])
{
if(rem[i]<=ts)
{
dur=rem[i];
flag[i]=1;
tt[i]=dur+from;
wt[i]=tt[i]-b[i];
}
else
dur=ts;
printf("%7d%15d%15d\n",i+1, from,from+dur);
rem[i] -= dur;
from += dur;
}
i=(i+1)%nop;
}
for(i=0;i<nop;i++)
{
twt+=wt[i];
ttt+=tt[i];
}
printf("\n\n Process ID \t Waiting Time \t Turn Around Time");
for(i=0;i<nop;i++)
{
printf("\n\t%d\t\t%d\t\t%d",i+1,wt[i],tt[i]);
}
awt=(float)twf/(float)nop;
att=(float)ttf/(float)nop;
printf("\nTotal Waiting Time:%d",twf);
printf("\nTotal Turn Around Time:%d",ttf);
printf("\nAverage Waiting Time:%.2f",awt);
printf("\nAverage Turn Around Time:%.2f",att);
}

```

**OUTPUT:**

Enter no. of Processes: 3

Enter the time slice: 3

Enter the Burst times.. P1 : 24

P2 : 5

P3 : 3

ProcessID	<u>Gantt Chart</u>	
	From Time	To Time
1	0	3
2	3	6
3	6	9
1	9	12
2	12	14
1	14	17
1	17	20
1	20	23
1	23	26
1	26	29
1	29	32

Process ID	Waiting Time	Turn Around Time
1	8	32
2	9	14
3	6	9

Total Waiting Time:23

Total Turn Around Time:55

Average Waiting Time:7.67

Average Turn Around Time:18.33

**RESULT:**

Thus the program was executed successfully

## **EX.NO:6) b )CPU SCHEDULING: SHORTEST JOB FIRST.**

**AIM:** To write a C program to implement the CPU scheduling algorithm for shortest job first.

### **PROBLEM DESCRIPTION:**

Cpu scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is SJF algorithm.

In this algorithm the process which has less service time given the cpu after finishing its request only it will allow cpu to execute next other process.

### **ALGORITHM:**

- Step 1: Get the number of process.
- Step 2: Get the id and service time for each process.
- Step 3: Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.
- Step 4: Calculate the total time and waiting time of remaining process.
- Step 5: Waiting time of one process is the total time of the previous process.
- Step 6: Total time of process is calculated by adding the waiting time and service time of each process.
- Step 7: Total waiting time calculated by adding the waiting time of each process.
- Step 8: Total turn around time calculated by adding all total time of each process.
- Step 9: Calculate average waiting time by dividing the total waiting time by total number of process.
- Step 10: Calculate average turn around time by dividing the total waiting time by total number of process.
- Step 11: Display the result.

### **PROGRAM CODING:**

```
#include<stdio.h> int main()
{
    int n,w[100],tot[100],i,j,awt,atot; float avwt,avtot;
    struct
    {
        int p,bt; } sjf[10],temp;
    printf("Enter the number of Processes:"); scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("Enter the Burst time for Process%d : ",i); scanf("%d",&sjf[i].bt);
        sjf[i].p=i;
    }
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(sjf[j].bt>sjf[i].bt)
            {
```

```

        temp=sjf[i];
        sjf[i]=sjf[j];
        sjf[j]=temp;
    }
    w[1]=0;
    tot[1]=sjf[1].bt;
    for(i=2;i<=n;i++) tot[i]=tot[i-1]+sjf[i].bt;
    awt=0;
    atot=0;
    for(i=1;i<=n;i++)
    {
        w[i]=tot[i]-sjf[i].bt; awt+=w[i]; atot+=tot[i];
    }
    avwt=(float)awt/n;
    avtot=(float)atot/n;
    printf("\n\nProcessId\tWaiting time\t TurnaroundTime");
    for(i=1;i<=n;i++)
    printf("\n\t%d\t\t%d\t\t%d",sjf[i].p,w[i],tot[i]);
    printf("\n\nTotal Waiting Time :%d",awt);
    printf("\n\nTotal Turnaround Time :%d",atot);
    printf("\n\nAverage Waiting Time :%.2f",avwt);
    printf("\n\nAverage Turnaround Time :%.2f",avtot); }

```

### **OUTPUT:**

[cse6@localhost Pgm]\$ cc prog9b.c

[cse6@localhost Pgm]\$ ./a.out

Enter the number of Processes:3

Enter the Burst time for Process1 : 24

Enter the Burst time for Process2 : 5

Enter the Burst time for Process3 : 3

ProcessId	Waiting time	TurnaroundTime
3	0	3
2	3	8
1	8	32

Total Waiting Time :11

Total Turnaround Time :43

Average Waiting Time :3.67

Average Turnaround Time :14.33

[cse6@localhost Pgm]\$

### **RESULT:**

Thus the program is executed.

## **EX.NO: 6c) CPU SCHEDULING: FIRST COME FIRST SERVE WITHOUT ARRIVAL TIME**

**AIM:** to write a c program to implement the first come first serve without arrival TIME CPU scheduling algorithm

### **PROBLEM DESCRIPTION:**

Cpu scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is FCFS algorithm.

In this algorithm the process which arrive first is given the cpu after finishing its request only it will allow cpu to execute other process.

### **ALGORITHM:**

Step 1: Create the number of process.

Step 2: Get the ID and Service time for each process.

Step 3: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.

Step 4: Calculate the Total time and Processing time for the remaining processes. Step 5: Waiting time of one process is the Total time of the previous process. Step 6: Total time of process is calculated by adding Waiting time and Service time.

Step 7: Total waiting time is calculated by adding the waiting time for lackprocess.

Step 8: Total turn around time is calculated by adding all total time of each process. Step 9: Calculate Average waiting time by dividing the total waiting time by total

number of process.

Step 10: Calculate Average turn around time by dividing the total time by the number of process.

Step 11: Display the result.

### **PROGRAM CODING:**

```
#include<stdio.h> int
main()
{
    int n,b[10],t=0,i,w=0,r=0,a=0; float
    avg,avg1;
    printf("\nEnter number of processes:");
    scanf("%d",&n);
    printf("\nEnter the burst times : \n");
    for(i=1;i<=n;i++) scanf("%d",&b[i]);
    printf("\n Gantt chart ");
    for(i=1;i<=n;i++)
    printf("P%d\t",i);
    printf("\n\nProcess BurstTime WaitingTime TurnaroundTime\n");
    for(i=1;i<=n;i++)
```

```

    {
        t=t+w;
        r=r+b[i];
        printf("P%d\tt%d\tt%d\tt%d\tt\n",i,b[i],w,r);
        w=w+b[i];
        a=a+r;
    }
    avg=(float)t/n;
    avg1=(float)a/n;
    printf("\n Average WaitingTime is %f",avg); printf("\n
    Average TurnaroundTime is %f\n",avg1); return(0);}

```

### **OUTPUT:**

```

[cse6@localhost Pgm]$ cc prog9a.c -o prog9a.out
[cse6@localhost Pgm]$ ./prog9a.out
Enter number of processes : 3
Enter the burst times :
24
5
3

```

Gantt chart	P1	P2	P3
Process	BurstTime	WaitingTime	TurnaroundTime
P1	24	0	24
P2	5	24	29
P3	3	29	32

```

Average WaitingTime is 17.666666
Average TurnaroundTime is 28.333334
[cse6@localhost Pgm]$

```

### **RESULT:**

Thus the program is executed

## **EX.NO: 6 d) CPU SCHEDULING: FIRST COME FIRST SERVE WITH ARRIVAL TIME**

### **AIM:**

To write a C program to implement the array representation of the CPU scheduling algorithm first come first serve using arrival time.

### **PROBLEM DESCRIPTION:**

CPU scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is FCFS algorithm.

In this algorithm the process which arrive first is given the cpu after finishing its request only it will allow cpu to execute other process

### **ALGORITHM:**

- Step 1: Create the number of process.
- Step 2: Get the ID and Service time for each process.
- Step 3: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.
- Step 4: Calculate the Total time and Processing time for the remaining processes.
- Step 5: Waiting time of one process is the Total time of the previous process. Step 6: Total time of process is calculated by adding Waiting time and Service time.
- Step 7: Total waiting time is calculated by adding the waiting time for lack process.
- Step 8: Total turn around time is calculated by adding all total time of each process.
- Step 9: Calculate Average waiting time by dividing the total waiting time by total number of process.
- Step 10: Calculate Average turn around time by dividing the total time by the number of process.
- Step 11: Display the result.

### **PROGRAM CODING:**

```
#include<stdio.h>
struct process
{
char name[5]; int at,
bt, wt, tt;
}
int main( )
{
int i, j, n, t;
float awt=0, att=0;
struct process p[10], temp;
printf("\n Enter the number of process: ");
scanf("%d", &n);
```

```

for(i=0; i<=n; i++)
{
printf("\n Enter the name, arrival time and burst time of process %d:", i+1);
scanf("%s %d %d", &p[i].name, &p[i].at, &p[i].bt);
}
for(i=0; i<n-1; i++)
for(j=0; j<n-1; j++)
if(p[j].at>p[j+1].at)
{
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
p[0].wt=0;
t=p[0].tt=p[0].bt;
att+=p[0].bt;
for(i=1; i<n; i++)
{
p[i].wt=t-p[i].at;
t+=p[i].bt;
p[i].tt=p[i].wt+p[i].bt;
awt+=p[i].wt;
att+=p[i].tt;
}
printf("\n Process Name Arrival time burst time Waiting Time Turnaround Time\n");
for(i=0; i<n; i++)
printf("%s \t\t %d \t\t %d \t\t %d \t\t %d \n", p[i].name, p[i].at, p[i].bt, p[i].wt, p[i].tt);
awt/=n;
att/=n;
printf("\n Average waiting Time : %f", awt);
printf("\n Average Turnaround Time : %f\n", att);

```

### **RESULT:**

Thus the program was executed successfully.



## **EX.NO: 6 e)CPU SCHEDULING: PRIORITY SCHEDULING**

**AIM:** To write a C program to implement the CPU scheduling algorithm for Priority.

### **PROBLEM DESCRIPTION:**

Cpu scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the process. One among that algorithm is Priority algorithm.

In this algorithm the processes will be given the priorities. The process which is having the highest priority is allocated the cpu first.

After finishing the request the cpu is allocated to the next highest priority and so on.

### **ALGORITHM:**

- Step 1: Get the number of process
- Step 2: Get the id and service time for each process.
- Step 3: Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.
- Step 4: Calculate the total time and waiting time of remaining process.
- Step 5: Waiting time of one process is the total time of the previous process.
- Step 6: Total time of process is calculated by adding the waiting time and service time of each process.
- Step 7: Total waiting time calculated by adding the waiting time of each process.
- Step 8: Total turn around time calculated by adding all total time of each process.
- Step 9: Calculate average waiting time by dividing the total waiting time by total number of process.
- Step 10: Calculate average turn around time by dividing the total waiting time by total number of process.
- Step 11: Display the result.

### **PROGRAM CODING:**

```
#include<stdio.h> int
main()
{
int n,temp=0,w[20],b[20], p[20],
t2[20],j,t1,d[20],i,
te=0,b1[20],t3=0;
float t,r;
w[1]=0;
printf("\nEnter no. of processes:");
scanf("%d",&n);
printf("\nEnter the burst times : ");
for(i=1;i<=n;i++)
{
printf("P%d : ",i);
scanf("%d",&b[i]); d[i]=i;
}
```

```

printf("Enter the priorities:");
for(i=1;i<=n;i++)
{
printf("P%d : ",i);
scanf("%d",&p[i]);
}
for(i=1;i<=n;i++)
for(j=i+1;j<=n;j++)
if(p[i]<p[j])
{
temp=p[i];
t1=d[i];
te=b[i];
p[i]=p[j];
d[i]=d[j];
b[i]=b[j];
p[j]=temp;
d[j]=t1;
b[j]=te;
}

printf("\nGantt Chart : ");
for(i=1;i<=n;i++) printf("P%d\t",d[i]);
printf("\nProcess \t Priority\tBurst Time\t Waiting Time\t Turnaround Time");
for(i=1;i<=n;i++)
{
t=d[i];
w[i+1]=w[i]+b[i];
t2[i]=b[i]+w[i];
t3+=t2[i];
printf("\nP%d\tt%d\tt%d\tt%d\tt%d",d[i],p[i],b[i],w[i],t2[i]);
}
temp=0;
for(i=1;i<=n;i++)
temp+=w[i];
t=(float)temp/n;
r=(float)t3/n;
printf("\nAverage Waiting time : %.2f",t);
printf("\nAverage Turnaround time : %.2f",r);
}

```

### **RESULT:**

Thus the program was executed successfully.

### **OUTPUT:**

```
[cse6@localhost Pgm]$ cc prog10a.c
[cse6@localhost Pgm]$ ./a.out
Enter the no. of processes : 3 Enter the
burst times
P1 : 24
P2 : 5
P3 : 3
Enter the priorities
```

P1 : 2

P2 : 1

P3 : 3

Gantt Chart : P2          P1P3

ProcessID	Priority	BurstTime	WaitingTime	TurnaroundTime
P2	1	5	0	5
P1	2	24	5	29
P3	3	3	29	32

Average Waiting Time : 11.33

Average Turnaround Time : 22.00

```
[cse6@localhost Pgm]$
```

### **RESULT:**

Thus the program is executed

## EX.NO: 7 IMPLEMENTATION OF SEMAPHORE

**AIM:** To write a C program to implement the Producer & consumer Problem (Semaphore)

### ALGORITHM:

Step 1: The Semaphore mutex, full & empty are initialized.

Step 2: In the case of producer process

- i) Produce an item in to temporary variable.
- ii) If there is empty space in the buffer check the mutex value for enter into the critical section.
- iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.

Step 3: In the case of consumer process

- i) It should wait if the buffer is empty
- ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
- iii) Signal the mutex value and reduce the empty value by 1.
- iv) Consume the item.

Step 4: Print the result

### PROGRAM CODING

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n 1.producer\n2.consumer\n3.exit\n");
    while(1)
    {
        printf(" \nenter ur choice");
        scanf("%d",&n);
        switch(n)
        {
            case 1:if((mutex==1)&&(empty!=0))
                producer();
            else
                printf("buffer is full\n");
            break;
            case 2:if((mutex==1)&&(full!=0))
                consumer();
            else
                printf("buffer is empty");
            break;
            case 3:exit(0);

            break;
        }
    }
}
int wait(int s)
{

```

```
return(--s);
}
int signal(int s)
{
return (++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\n producer produces the items %d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes the item %d",x);
x--;
mutex=signal(mutex);
}
```

**RESULT:**

Thus the program was executed successfully

## EX.NO: 8) SHARED MEMORY AND IPC

### AIM

To write a C program to implement shared memory and inter process communication.

### ALGORITHM:

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution

### PROGRAM CODING

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/sem.h>
int main()
{
    int id,semid,count=0,i=1,j;
    int *ptr;
    id=shmget(8078,sizeof(int),IPC_CREAT|0666);
    ptr=(int *)shmat(id,NULL,0);
    union semun
    {
        int val;
        struct semid_ds *buf;
        ushort *array;
    }u;
    struct sembuf sem;
    semid=semget(1011,1,IPC_CREAT|0666);
    ushort a[1]={1};
    u.array=a;
    semctl(semid,0,SETALL,u);
    while(1)
    {
        sem.sem_num=0;
        sem.sem_op=-1;
        sem.sem_flg=0;
        semop(semid,&sem,1);
        *ptr=*ptr+1;
        printf("process id:%d count is :%d \n",getpid(),*ptr);
        for(j=1;j<=1000000;j++)
        {
            sem.sem_num=0;
            sem.sem_op+=1;
            sem.sem_flg=0;
            semop(semid,&sem,1);
```

```
}  
}  
shmdt(ptr);  
}
```

### **OUTPUT:**

```
[cse6@localhost Pgm]$ cc ex11.c -o ex11.out  
[cse6@localhost Pgm]$ ./ex11.out
```

```
Produced element a  
Consumed element a  
Produced element b  
Consumed element b  
Produced element c  
Consumed element c  
Produced element d  
Consumed element d  
Produced element e  
Consumed element e  
Produced element f  
Consumed element f  
Produced element g  
Consumed element g  
Produced element h  
Consumed element h
```

### **RESULT:**

Thus the program was executed

## **EX.NO: 9) IMPLEMENTATION BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE**

### **AIM**

To write a C program to implement bankers algorithm for dead lock avoidance

### **ALGORITHM:**

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

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

Step 7: Stop the execution

### **PROGRAM CODING**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int work[5],avl[5],alloc[10][10],l;
int need[10][10],n,m,I,j,avail[10],max[10][10],k,count,fcount=0,pr[10];
char finish[10]={,,f",f",f",f",f",f",f",f",f",f};
clrscr();
printf("enter the no of process");
scanf("%d",&n);
printf("enter the no of resources");
scanf("%d",&m);
printf("enter the total no of resources");
for(i=1;i<=m;i++)
scanf("%d",&avail[i]);
printf("enter the max resources req by each pr alloc matrix");
for(i=1;i<=n;i++)
for(j=1;j<=m;j++)
scanf("%d",&max[i][j]);
printf("process allocation matrix");
for(i=1;i<=n;i++)
for(j=1;j<=m;j++)
scanf("%d",&alloc[i][j]);
for(i=1;i<=n;i++)
for(j=1;j<=m;j++)
need[i][j]=max[i][j]-alloc[i][j];
for(i=1;i<=n;i++)
{ k=
0;
for(j=1;j<=m;j++)
{
```



```

k=k+alloc[i][j];
}
avl[i]=avl[i]-k;
work[i]=avl[i];
}
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
{
count=0;
for(j=1;j<=m;j++)
{
if((finish[i]=="f")&&(need[i][j]<=work[i])) ofcount++;
}
if(count==m)
{
for(l=1;l<=m;l++)
work[l]=work[l]+alloc[i][l];
finish[i]="t";
pr[k]=i;
JBLET
break;
}
}
for(i=1;i<=n;i++)
if(finish[i]=="t")
fcount++;
if(fcount==n)
{
printf("the system is in safe state");
Dept
for(i=1;i<=n;i++)
printf("%d",pr[i]);
}
else
printf("the system is not in safe state");
getch();
}

```

### **RESULT:**

Thus the program was executed successfully.

## EX.NO: 10) IMPLEMENTATION OF DEADLOCK DETECTION

### ALGORITHMAIM

To write a C program to implement Deadlock Detection algorithm

### ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

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

Step 7: Stop the execution

//Deadlock Detection algorithm implementation

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

```
#include <conio.h>
```

```
void main()
```

```
{
```

```
int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
```

```
clrscr();
```

```
printf("Enter total no of processes");
```

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

```
printf("Enter total no of resources");
```

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

```
printf("Enter claim (Max. Need) matrix\n");
```

```
for(i=1;i<=tp;i++)
```

```
{
```

```
printf("process %d:\n",i);
```

```
for(j=1;j<=tr;j++)
```

```
scanf("%d",&c[i][j]);
```

```
}
```

```
printf("Enter allocation matrix\n");
```

```
for(i=1;i<=tp;i++)
```

```
{
```

```
printf("process %d:\n",i);
```

```
for(j=1;j<=tr;j++)
```

```
scanf("%d",&p[i][j]);
```

```
}
```

```
printf("Enter resource vector (Total resources):\n");
```

```
for(i=1;i<=tr;i++)
```

```
{
```

```
scanf("%d",&r[i]);
```

```
}
```

```
printf("Enter availability vector (available resources):\n");
```

```
for(i=1;i<=tr;i++)
```

```
{
```

```
scanf("%d",&a[i]);
```

```
temp[i]=a[i];
```

```
}
```

```
for(i=1;i<=tp;i++)
```

```

{
    sum=0;
    for(j=1;j<=tr;j++)
    {
        sum+=p[i][j];
    }
    if(sum==0)
    {
        m[k]=i;
        k++;
    }
}
for(i=1;i<=tp;i++)
{
    for(l=1;l<=k;l++)
    if(i!=m[l])
    {
        flag=1;
        for(j=1;j<=tr;j++)
        if(c[i][j]<temp[j])
        {
            flag=0;
            break;
        }
    }
    if(flag==1)
    {
        m[k]=i;
        k++;
        for(j=1;j<=tr;j++)
        temp[j]+=p[i][j];
    }
}
printf("deadlock causing processes are:");
for(j=1;j<=tp;j++)
{
    found=0;
    for(i=1;i<=k;i++)
    {
        if(j==m[i])
        found=1;
    }
    if(found==0)
    printf("%d\t",j);
}
getch();
}

```

**OUTPUT:**

Enter total no. of processes : 4

Enter total no. of resources : 5

Enter claim (Max. Need) matrix :

0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1 0 1 0 1

Enter allocation matrix :

1 0 1 1 0

1 1 0 0 0

0 0 0 1 0

0 0 0 0 0

Enter resource vector (Total resources) :

2 1 1 2 1

Enter availability vector (available resources) :

0 0 0 0 1

deadlock causing processes are : 2 3

**RESULT:**

Thus the program was executed successfully.

## **EX.NO: 11) THREADING & SYNCHRONIZATION**

### **AIM:**

To write a C program to implement Threading & Synchronization

### **ALGORITHM:**

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

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

Step 7: Stop the execution

### **PROGRAM CODING**

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* doSomething(void *arg)
{
    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d started\n", counter);

    for(i=0; i<(0xFFFFFFFF);i++);
    printf("\n Job %d finished\n", counter);
    return NULL;
}
int main(void)
{
    int i = 0;
    int err;
    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &doSomething, NULL);
        if (err != 0)
            printf ("\ncan't create thread :[%s]", strerror(err));
        i++;
    }
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    return 0;
}
```

**RESULT:** Thus the program was executed successfully

## **EX.NO: 12) Implementation of the following Memory Allocation Methods for fixed partition**

**a) First Fit b) Worst Fit c) Best Fit**

### **AIM:**

To implement the following Memory Allocation Methods for fixed partition

a) First Fit b) Worst Fit c) Best Fit

### **ALGORITHM:**

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process.

In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

### **a) WORST-FIT**

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

```

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

```

### INPUT

Enter the number of blocks: 3  
Enter the number of files: 2

Enter the size of the blocks:-  
Block 1: 5  
Block 2: 2  
Block 3: 7

Enter the size of the files:-  
File 1: 1  
File 2: 4

### OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

### b) Best-fit

```

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

```

```

}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

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

```

### INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

### OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	2	2	1
2	4	1	5	1

### c) First-fit

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

```
#include<conio.h>
```



```

#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t%d\t%d\t%d\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}

```

## INPUT

Enter the number of blocks: 3  
Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

### **OUTPUT**

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

### **RESULT:**

Thus the program was executed successfully.

## **EX.NO: 13) IMPLEMENTATION OF PAGING TECHNIQUE OF MEMORY MANAGEMENT**

### **AIM:**

To write a C program to implement the concept of Paging

### **ALGORITHM:**

Step 1: The Semaphore mutex, full & empty are initialized.

Step 2: In the case of producer process

- i) Produce an item in to temporary variable.
- ii) If there is empty space in the buffer check the mutex value for enter into the critical section.
- iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.

Step 3: In the case of consumer process

- i) It should wait if the buffer is empty
- ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
- iii) Signal the mutex value and reduce the empty value by 1.
- iv) Consume the item.

Step 4: Print the result

### **PROGRAM CODING**

```
#include<stdio.h>
main()
{
int i,j,arr[100],pt[20],val,pgno,offset,phymem,fs,nf;
printf("\n Memory Management paging\n");
printf("enter the size of physical memory");
scanf("%d",&phymem);
for(i=20,j=0;i<phymem+20,j<phymem;i++,j++)
arr[i]=j;
printf("\n Enter size of frame or page");
scanf("%d",&fs);
nf=phymem/fs;
printf("No of frame available are\t%d",nf);
printf("\n Enter the page table");
for(i=0;i<nf;i++)
{
scanf("%d",&pt[i]);
}
printf("\n Enter the page no");
scanf("%d",&pgno);
printf("\n Enter the offset");
scanf("%d",&offset);
val=(fs*pt[pgno])+offset;
printf("the physical address is:%d\n",arr[val]);
}
```

**RESULT:** Thus the program is executed

## EX.NO: 14)i PAGE REPLACEMENT ALGORITHMS - FIFO

To write a C program to implement page replacement FIFO (First In First Out) algorithm

### ALGORITHM:

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution

### PROGRAM CODING

```
#include<graphics.h>
#include<stdlib.h>
void main()
{
    //declare pages for store page nos, frames to store frame details
    int pages[10], frames[10][10], ppos[10], fillcount=0, least;
    //no_p stands for no of pages, no_f stands for no of frames
    int I,j,k,m,pos=0,no_p,no_f,faults=0,gd=DETECT,gm,no;
    int x1=20,y1=100;
    float ratio;
    char found,
    str[30],ch; clrscr();
    //GRAPHICS initialise method
    initgraph(&gd,&gm,"../bgi");
    setbkcolor(BLUE);
    puts("Enter no of pages do u have"); scanf("%d",&no_p);
    puts("Enter no of frames do u have");
    scanf("%d",&no_f);
    //initializing each frame
    with 0 for(i=0;i<no_f;i++)
    for(j=0;j<no_p;j++)
    frames[i][j]=0;
    for(i=0;i<no_p;i++)
    {
        puts("Enter page num");
        scanf("%d",&pages[i]);
        clrscr();
        cleardevice();
        x1=20,y1=100;
        found='f';
        for(j=0;j<no_f;j++)
        {
```

```

if(i!=0) frames[j][i]=frames[j][i-1];
//checking whether page is there in frames
or not if(frames[j][i]==pages[i])
found='t';
}
//if PAGE is not there in
frames if(found=='f')
{
faults++;
fillcount++;
if(fillcount<=no_f)
{
frames[pos][i]=pages[i];
pos++;
}
IT
else
{
for(j=0;j<no_f;j++)
ppos[j]=0;
for(j=0;j<no_f;j++)
{
for(k=i-3;k<i;k++)
{
if(frames[j][i]==pages[k])
ppos[j]=k;
}
}
least=ppos[0];
for(j=0;j<no_f;j++)
{
if(ppos[j]<least)
least=ppos[j];
}
for(j=0;j<no_f;j++)
if(pages[least]==frames[j][i])
pos=j;
frames[pos][i]=pages[i];
}
}
//printing frames each time we
enter a no settextstyle(2,0,6);
for(k=0;k<no_f;k++)

```

```

{
for(j=0;j<=i;j++)
{
rectangle(x1,y1,x1+40,y1+45);
if(frames[k][j]!=0)
{
//changing text color in case of replacement
if(j==i&&frames[k][j]==pages[i]&&found=='f')
setcolor(MAGENTA);
else
setcolor(WHITE);
itoa(frames[k][j],str,10);
outtextxy(x1+15,y1+15,str);
}
else
outtextxy(x1+10,y1+10,"");
setcolor(WHITE);
x1+=55;
}
y1+=45;
x1=20;
}
}
//printing page fault ratio
printf("/n/n page fault
ratio=%f",(float)faults/(float)no_p); getch();
}

```

### **RESULT:**

Thus the program was executed successfully.

## EX.NO: 14)ii PAGE REPLACEMENT ALGORITHMS - LRU

### AIM

To write a C program to implement page replacement LRU (Least Recently Used) algorithm.

### ALGORITHM:

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution

### PROGRAM CODING

```
#include<graphics.h>
#include<stdlib.h>
void main()
{
//declare pages for stores page nos, frames to store frame
details int pages[10],frames[10][10],ppos[10],fillcount=0,least;
//no_pstands for no of pages, no_f stands for no of frames
int i,j,k,m,pos=0;no_p,no_f,faults=0;gd=DETECT,gm,no;
int x1=20,y1=100;
float ratio;
char found,
str[30],ch,occur; clrscr();
//GRAPHICS initialise method
initgraph(&gd,&gm,"../bgi");
setbkcolor(BLUE);
puts("Enter no of pages do u
have"); scanf("%d",&no_p);
puts("Enter no of frames do u
have"); scanf("%d",&no_f);
//initializing each frame
with 0 for(i=0;i<no_f;i++)
for(j=0;j<no_p;j++)
frames[i][j]=0;
for(i=0;i<no_p;i++)
{
puts("Enter page num");
scanf("%d",&pages[i]);
clrscr();
cleardevice();
x1=20,y1=100;
```

```

found='f';
for(j=0;j<no_f;j++)
{
if(i!=0) frames[j][i]=frames[j][i-1];
//checking whether page is there in frames
or not if(frames[j][i]==pages[i])
found='t';
}
//if PAGE is not there in
frames if(found=='f')
{
faults++;
fillcount++;
if(fillcount<=no_f)
frames[pos][i]=pages[i];
pos++;
}
else
{
for(j=0;j<no_f;j++)
ppos[j]=0;
for(j=0;j<no_f;j++)
{
for(k=0;k<i;k++)
{
if(frames[j][i]==pages[k])
Computer Networks & Operating Systems Lab Manual
ppos[j]++;
}
}
least=ppos[0];
for(j=0;j<no_f;j++)
{

if(least>ppos[j])
least=ppos[j];
}
ocurs='n';
for(j=0;j<1&&occur=='n';j++)
{
for(k=0;k<no_f;k++)
{
if(pages[j]==frames[k][i]&&ppos[k]==least)

```



```

{
pos=k;
occur='y';
}
}
}
frames[pos][i]=pages[i];
}
}
//printing frames each time we
enter a no settextstyle(2,0,6);
for(k=0;k<no_f;k++)
{
for(j=0;j<=i;j++)
{
rectangle(x1,y1,x1+40,y1+45);
if(frames[k][j]!=0)
{
//changing the text color when page is replaced of
if(j==i&&frames[k][j]==pages[i]&&found=='f')
setcolor(MAGENTA);
else
setcolor(WHITE);
itoa(frames[k][j],str,10);
outtextxy(x1+15,y1+15,str);
}
else
outtextxy(x1+10,y1+10,"");
setcolor(WHITE);
x1+=55;
}
y1+=45;
x1=20;
}
}
//printing page fault ration
printf("page fault ration
%f", (float)faults/(float)no+p); getch();
}

```

#### RESULT:

Thus the program was executed successfully

## **EX.NO: 14)iii PAGE REPLACEMENT ALGORITHMS - LFU**

### **AIM**

To write a C program to page replacement LFU (Least Frequently Used) algorithm.

### **ALGORITHM:**

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution

### **PROGRAM CODING**

```
#include<graphics.h>
#include<stdlib.h>
void main()
{
//declare pages for stores page nos, frames to store
frame details int pages[10],frames[10][10];
//no_p stands for no of pages, no_f stands for no of frames
int i,j,k,m,pos=0;no_p,no_f,faults=0;gd=DETECT,gm,no; int
x1=20,y1=100;
float ratio;
char found,
str[30],ch; clrscr();
//GRAPHICS initialise method
initgraph(&gd,&gm,"../bgi");
setbkcolor(BLUE);
puts("Enter no of pages do u have");
scanf("%d",&no_p);
puts("Enter no of frames do u have");
scanf("%d",&no_f);
//fill all frames with 0
for(i=0;i<no_f;i++)
for(j=0;j<no_p;j++)
frames[i][j]=0;
for(i=0;i<no_p;i++)
{
puts("Enter page num");
scanf("%d",&pages[i]);
clrscr();
```

```

cleardevice();
x1=20,y1=100;
found='f';
for(j=0;j<no_f;j++)
{
if(i!=0) frames[j][i]=frames[j][i-1];
//checking whether page is there in frames or not
if(frames[j][i]==pages[i])
found='t';
}

//if PAGE is not there in
frames if(found=='f')
{
frames[pos][i]=pages[i];
faults++;
if(pos<no_f)
pos++;
else pos=0;
}
//printing frames each time we
enter a no settextstyle(2,0,6);
for(k=0;k<no_f;k++)
{
for(j=0;j<=i;j++)
{
rectangle(x1,y1,x1+40,y1+45);
if(frames[k][j]!=0)
{

//changing the text color when page is replaced
if(j==i&&frames[k][j]==pages[i]&&found=='f')
JBLET
setcolor(MAGENTA);
else
setcolor(WHITE);
itoa(frames[k][j],str,10);
outtextxy(x1+15,y1+15,str);
}
else
outtextxy(x1+10,y1+10,"");
setcolor(WHITE);
x1+=55;
}
}

```

```
        y1+=45;
        x1=20;
    }
}

//printing page fault ratio
printf("page fault ration %f",(float)faults/(float)no_p);
getch();
}
```

**RESULT:** Thus the program was executed successfully

## **EX.NO: 15 a FILE ORGANIZATION TECHNIQUES - SINGLE LEVEL DIRECTORY**

### **AIM**

To write a C program to implement File Organization concept using the technique Single level directory.

### **ALGORITHM:**

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution

### **PROGRAM CODING**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm,count,i,j,mid,cir_x;
char fname[10][20];
clrscr();
initgraph(&gd,&gm,"c:\\tc\\bgi");
cleardevice();
setbkcolor(GREEN);
puts("Enter no of files do u have?");
scanf("%d",&count);
for(i=0;i<count;i++)
{
cleardevice();
setbkcolor(GREEN);
printf("Enter file %d name",i+1);
scanf("%s",fname[i]);
setfillstyle(1,MAGENTA);
mid=640/count;
cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);
settextjustify(1,1);
outtextxy(320,125,"Root
Directory"); setcolor(BLUE);
```

```
for(j=0;j<=i;j++,cir_x+=mid)
{
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[j]);
}
getch();
}
}
```

**RESULT:**

Thus the program was executed successfully.

## EX.NO: 15 b) FILE ORGANIZATION TECHNIQUES : TWO

### LEVEL AIM

To write a C program to implement File Organization concept using the technique two level directory.

### ALGORITHM:

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

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

Step 7: Stop the execution

### PROGRAM CODING

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element
node; void main()
{
int gd=DETECT,gm;
node *root;
root=NULL; clrscr();
create(&root,0,"null",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\bgi");
display(root);
getch();
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node*)malloc(sizeof(node));
printf("enter name of dir/file(under %s):",dname); fflush(stdin);
gets((*root)->name);
```

```

if(lev==0||lev==1)
(*root)->ftype=1; else
(*root)->ftype=2;
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x; (*root)->lx=lx; (*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
if(lev==0||lev==1)
{
if((*root)->level==0)
printf("How many users");
else
printf("hoe many files");
printf("(for%s):",(*root)->name);
scanf("%d",&(*root)->nc);
}
else (*root)->nc=0; if((*root)->nc==0) gap=rx-lx;
else gap=(rx-lx)/(*root)->nc; for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name);

```



```
        for(i=0;i<root->nc;i++)  
        {  
            display(root->link[i]);  
        }  
    }  
}
```

**RESULT:**

Thus the program was executed successfully.

## **EX.NO: 15c) FILE ORGANIZATION TECHNIQUES : HIERARCHICAL**

### **AIM**

To write a C program to implement File Organization concept using the technique hierarchical level directory.

### **ALGORITHM:**

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution

### **PROGRAM CODING**

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

```

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

```

```

{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]); } } }

```

**RESULT:**

Thus the program was executed successfully.

## **EX.NO: 15 d) FILE ORGANIZATION TECHNIQUES : DAG**

### **AIM**

To write a C program to implement File Organization concept using the technique using DAG

### **ALGORITHM:**

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

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

Step 7: Stop the execution

### **PROGRAM CODING**

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
struct tree_element
{
    char name[20];
    int x,y,ftype,lx,rx,nc,level;
    struct tree_element *link[5];
};
typedef struct tree_element node;
typedef struct
{
    char from[20];
    char to[20];
}link;
link L[10]; int nofl;
```

```

node * root;
void main()
{
int gd=DETECT,gm;
root=NULL; clrscr();
create(&root,0,"root",0,639,320);
read_links();
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
draw_link_lines();
display(root);
getch();
closegraph();
}
read_links()
{
int i;
printf("how many links");
scanf("%d",&nofl);
for(i=0;i<nofl;i++)
{
printf("File/dir:");
fflush(stdin);
gets(L[i].from);
printf("user name:");
fflush(stdin);
gets(L[i].to);
}
}
draw_link_lines()
{
int i,x1,y1,x2,y2;
for(i=0;i<nofl;i++)
{
search(root,L[i].from,&x1,&y1);
search(root,L[i].to,&x2,&y2);
setcolor(LIGHTGREEN);
setlinestyle(3,0,1);
line(x1,y1,x2,y2);
setcolor(YELLOW);
setlinestyle(0,0,1);
}
}

```

```

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

```

```

else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create( & ( (*root)->link[i] ) , lev+1 ,
(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
/* displays the constructed tree in graphics mode */
display(node *root)
{

int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14); if(root
!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root-
>name); for(i=0;i<root->nc;i++)

{
display(root->link[i]);

}
}
}

```



## **EX.NO : 15 e) FILE ALLOCATION TECHNIQUE-INDEXED ALLOCATION**

### **AIM:**

To write a C program to implement File Allocation concept using the technique indexed allocation Technique..

### **ALGORITHM:**

Step 1: Start the Program

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

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

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

Step 7: Stop the execution.

### **PROGRAM CODING:**

```
#include<stdio.h>
void main()
{
char a[10];
int i,ib,cib[10];
printf("\n enter the file name:");
scanf("%s",a);
printf("\n index block:");
scanf("%d",&ib);
for(i=1;i<=5;i++)
{
printf("\n enter the child of index block %d:",i);
scanf("%d",&cib[i]);
}
printf("\n the list of files\t index block\n");
printf("%s\t\t %d",a,ib);
printf("\n the above file utilization index block of child blocks followin\t");
printf("\n");
for(i=1;i<=5;i++)
{
printf("%d\t\t",cib[i]);
}
printf("\n");
}
```

### **OUTPUT:**

Enter the name:Testing

Index block:19

Enter the child of index block 1:9

Enter the child of index block 2:16

Enter the child of index block 3:1

Enter the child of index block 4:10

Enter the child of index block 5:25

The list of files      index block  
Testing                      19

The above file utilization index block of child blocks following:

9            16        1            10        25

**RESULT:**

Thus the program was executed successfully.  
36

## EX.NO : 15 f) FILE ALLOCATION TECHNIQUE-LINKED

### ALLOCATION AIM:

To write a C program to implement File Allocation concept using the technique Linked List Technique..

### ALGORITHM:

Step 1: Start the Program

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

Step 3: Enter the filename, starting block ending block.

Step 4: Print the free block using loop.

Step 5: "for" loop is created to print the file utilization of linked type of entered type .

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

Step 7: Stop the execution.

### PROGRAM CODING

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

### OUTPUT:

```
Enter the
filename:binary Enter
the starting block:19
Enter the ending
block:25 Enter the free
block:1:12 Enter the free
block:2:34 Enter the free
block:3:21 Enter the free
```

block:4:18 Enter the free  
block:5:35

File name	starting block	ending block
Binary	19	25

Binary file utilization of linked type of the following  
blocks: 19 12 34 21 18 35 25

**RESULT:**

Thus the program was executed successfully.

**CONTENT BEYOND SYLLABI:****IMPLEMENTATION OF DISK SCHEDULING ALGORITHMS****EX.NO:16****DATE:****AIM:**

To write a 'C' program to implement the Disk Scheduling algorithm for First Come First Served (FCFS), Shortest Seek Time First (SSTF), and SCAN.

**PROBLEM DESCRIPTION:**

Disk Scheduling is the process of deciding which of the cylinder request is in the ready queue is to be accessed next.

The access time and the bandwidth can be improved by scheduling the servicing of disk I/O requests in good order.

**Access Time:**

The access time has two major components: **Seek time and Rotational Latency.**

**Seek Time:**

Seek time is the time for disk arm to move the heads to the cylinder containing the desired sector.

**Rotational Latency:**

Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

**Bandwidth:**

The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

**ALGORITHM:**

1. Input the maximum number of cylinders and work queue and its head starting position.
2. **First Come First Serve Scheduling (FCFS) algorithm** – The operations are performed in order requested.
3. There is no reordering of work queue.
4. Every request is serviced, so there is no starvation.
5. The seek time is calculated.
6. **Shortest Seek Time First Scheduling (SSTF) algorithm** – This algorithm selects the request with the minimum seek time from the current head position.
7. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
8. The seek time is calculated.
9. **SCAN Scheduling algorithm** – The disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.

10. At the other end, the direction of head movement is reversed, and servicing continues.
11. The head continuously scans back and forth across the disk.
12. The seek time is calculated.
13. Display the seek time and terminate the program

**Program :**

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

void fcfs(int noq, int qu[10], int st)
{
    int i,s=0;
    for(i=0;i<noq;i++)
    {
        s=s+abs(st-qu[i]);
        st=qu[i];
    }
    printf("\n Total seek time :%d",s);
}

void sstf(int noq, int qu[10], int st, int visit[10])
{
    int min,s=0,p,i;
    while(1)
    {
        min=999;
        for(i=0;i<noq;i++)
            if (visit[i] == 0)
            {
                if(min > abs(st - qu[i]))
                {
                    min = abs(st-qu[i]);
                    p = i;
                }
            }
        if(min == 999)
            break;
        visit[p]=1;
        s=s + min;
        st = qu[p];
    }
    printf("\n Total seek time is: %d",s);
}

void scan(int noq, int qu[10], int st, int ch)
{
    int i,j,s=0;
```

```

        for(i=0;i<noq;i++)
        {
            if(st < qu[i])
            {
                for(j=i-1; j>= 0;j--)
                {
                    s=s+abs(st - qu[j]);
                    st = qu[j];
                }
                if(ch == 3)
                {
                    s = s + abs(st - 0);
                    st = 0;
                }
                for(j = 1;j < noq;j++)
                {
                    s= s + abs(st - qu[j]);
                    st = qu[j];
                }
                break;
            }
        }
    }
    printf("\n Total seek time : %d",s);
}

int main()
{
    int n,qu[20],st,i,j,t,noq,ch,visit[20];
    printf("\n Enter the maximum number of cylinders : ");
    scanf("%d",&n);
    printf("enter number of queue elements");
    scanf("%d",&noq);
    printf("\n Enter the work queue");
    for(i=0;i<noq;i++)
    {
        scanf("%d",&qu[i]);
        visit[i] = 0;
    }
    printf("\n Enter the disk head starting position: \n");
    scanf("%d",&st);
    while(1)
    {
        printf("\n\n\t\t MENU \n");
        printf("\n\n\t\t 1. FCFS \n");
        printf("\n\n\t\t 2. SSTF \n");
        printf("\n\n\t\t 3. SCAN \n");
        printf("\n\n\t\t 4. EXIT \n");
        printf("\nEnter your choice: ");
        scanf("%d",&ch);
        if(ch > 2)

```

```

        {
            for(i=0;i<noq;i++)
            for(j=i+1;j<noq;j++)
            if(qu[i]>qu[j])
            {
                t=qu[i]; qu[i]
                = qu[j]; qu[j]
                = t;
            }
        }
switch(ch)
{
    case 1: printf("\n FCFS \n");
    printf("\n*****\n");
    fcfs(noq,qu,st);
    break;

    case 2: printf("\n SSTF \n");
    printf("\n*****\n");
    sstf(noq,qu,st,visit);
    break;
    case 3: printf("\n SCAN \n");
    printf("\n*****\n");
    scan(noq,qu,st,ch);
    break;
    case 4: exit(0);
}
}
}
}

```

## Output

Enter the maximum number of cylinders : 200  
 enter number of queue elements5

Enter the work queue23  
 89  
 132  
 42  
 187

Enter the disk head starting position:  
 100

MENU

1. FCFS

2. SSTF



3. SCAN

4. EXIT

Enter your choice: 1

FCFS

\*\*\*\*\*

Total seek time : 421

MENU

1. FCFS

2. SSTF

3. SCAN

4. EXIT

Enter your choice: 2

SSTF

\*\*\*\*\*

Total seek time is: 273

MENU

1. FCFS

2. SSTF

3. SCAN

4. EXIT

Enter your choice: 3

SCAN

\*\*\*\*\*

Total seek time : 287

MENU

1. FCFS
2. SSTF
3. SCAN
4. EXIT

Enter your choice: 4

**Result:**

Thus the C program to implement the Disk scheduling algorithms namely FCFS, SSTF, SCAN algorithms was written and executed. The obtained outputs were verified.