ANNAI MIRA COLLEGE OF ENGINEERING AND TECHNOLOGY

NH-46, Chennai-Bengaluru National

Highways, Arappakkam, Ranipet-632517,

Tamil Nadu, India

Telephone : 04172-292925 Fax : 04172-292926

Email : amcet.rtet@gmail.com/info@amcet.in Web : www.amcet.in

DEPARTMENT OF INFORMATION TECHNOLOGY

CD3461 – OPERATING SYSTEM LABORATORY

Name : …….

Register Number : …….

Year & Branch : …….

Semester : …….

Academic Year : …….

ANNAI MIRA COLLEGE OF ENGINEERING AND TECHNOLOGY

NH-46, Chennai-Bengaluru National

Highways, Arappakkam, Ranipet-632517,

Tamil Nadu, India

Telephone : 04172-292925 Fax : 04172-292926

Email : amcet.rtet@gmail.com/info@amcet.in Web : www.amcet.in

DEPARTMENT OF INFORMATION TECHNOLOGY

CD3461 – OPERATING SYSTEM LABORATORY

CERTIFICATE

This is to Certify that the Bonafide record of the practical work done by ……. Register Number : ……. of IInd year **B.TECH (INFORMATION TECHNOLOGY)** submitted for the **B.TECH IT** practical examination (IV th Semester) in **CD3461 – OPERATING SYSTEM LABORATORY** during the academic year 2024 – 2025.

Staff in–Charge **Head of the Department**

Submitted for the practical examination held on _____ .

Internal Examiner **External Examiner**

INDEX

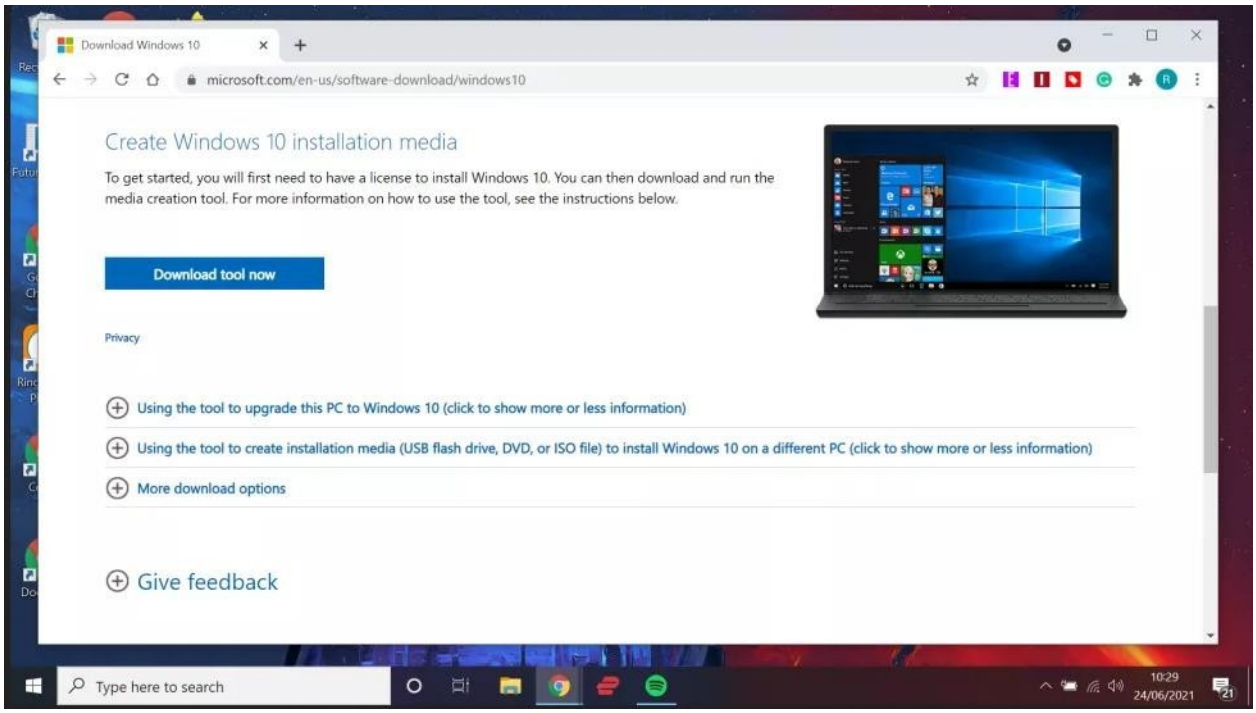
S.NO	DATE	EXPERIMENT NAME	SIGN
1.		Installation of Windows	
2.		Basic Unix commands and Shell programming	
3.		Multiple System Call	
4.		Implement the following CPU scheduling algorithms	
5.		Implement Shared memory and IPC	
6.		Implementation of semaphore	
7.		Implement Bankers Algorithm for Dead Lock Avoidance	
8.		Implement an Algorithm for Dead Lock Detection	
9.		Memory allocation techniques First Fit Allocation	
10.		FIFO & LRU Page Replacement	
11.		FILE ORGANIZATION	
12.		FILE ALLOCATION	
13.		Implement Paging Technique of memory management	
14.		Disk scheduling algorithm	
15.		Installation of Guest operating system Linux	

Ex No : 1	Installation of Windows
Date :	

1. Check your device meets the Windows 10 system requirements. Below you'll find the minimum specs needed to run Windows 10, so check your device is capable:

CPU: 1GHz or faster processor
RAM: 1GB for Windows 10 32-bit or 2GB for Windows 10 64-bit
Storage: 32GB of space or more
GPU: DirectX 9 compatible or later with WDDM 1.0 driver
Display: 800x600 resolution or higher

2. Create USB installation media. Visit Microsoft's Windows 10 download page (opens in new tab) and select "Download tool now" under the "create Windows 10 installation media" section. Transfer the downloaded installer tool to a USB drive.



3. Run the installer tool. Open the installer tool by clicking on it. Accept Microsoft's terms, and then **select "Create installation media for another PC"** on the "What do you want to do?" page. After selecting which language you want Windows 10 to run in, and which edition you want as well (32-bit or 62-bit), you'll be asked what type of media you want to use.

Installing from a USB drive is definitely the preferred option but you can also install from a CD or ISO file. Once you choose your device, the installer tool will download the required files and put them onto your drive.

4. Use your installation media. Insert your installation media into your device and then **access the computer's BIOS or UEFI**. These are the systems that allow you to control your computer's core hardware.

The process of accessing these systems is unique to each device, but the manufacturer's website should be able to give you a helping hand here. Generally, you'll need to **press the F2, F12 or Delete keys** as your computer boots up.

5. Change your computer's boot order. Once you have access to your computer's BIOS/UEFI you'll need to locate the settings for boot order. You need the Windows 10 installation tool to be higher up on the list than the device's current current boot drive: this is the SSD or HDD that your existing OS is stored on. You should **move the drive with the installer files to the very top of the boot order menu**. Now, when you restart your device the Windows 10 installer should load up first.

6. Restart your device. Save your settings in the BIOS/UEFI and reboot your device.

7. Complete the installation. Your device should now load up the Windows 10 installation tool on restart. This will guide you through the rest of the installation process.

Result:

Thus installation procedure for Windows operating system was studied.

Ex No : 2a	Basic Unix commands
Date :	

Aim:

To study about basic UNIX Commands.

I.FILE MANIPULATION COMMANDS

1. Command : Cat

Purpose : It is used to create display and concatenate the contents of the file(s). **Syntax : cat > <file name > , cat <file name > , cat file1 file2 > file3 Example :** $ cat > devi ,$ cat devi , $cat kala devi > rani

2. Command : More

Purpose : It is used to display the contents of the file on the screen at a time.
Syntax : more <file name >
Example : $ more devi

3. Command : Wc

Purpose : It is used to count the number of lines ,words and characters in a file or group of files.
Syntax : wc [options] <file name > Example : $ wc –l devi

4. Command : File

Purpose : It is used to determine the type of the file.
Syntax : file <file name > Example : $ file devi

5. Command : Spell

Purpose : It is used to find the spelling errors in the file.
Syntax : spell [options] <file name >
Example : $ spell -b devi

6. Command : Split

Purpose : It is used to split the given file into smaller pieces of given size.
Syntax : **split –size <file name > < splitted file name >**
Example : $ split –2 devi de

7. Command : Cp

Purpose : It is used to copy one or more files.
Syntax : cat <source file name > <destination file name>
Example : $ cp devi latha

8. Command : Mv

Purpose : It is used to move a file within a directory with different names and also used to move a file to different directory with its original name.
Syntax : mv <source file name > <destination file name>
Example : $ mv devi jeya

9. Command : Rm
Purpose : It is used to remove a file from the disk.
Syntax : rm <file name
> Example : $ rm devi

10. Command : Chmod
Purpose : It is used to set the three permissions for all categories of users for a file(s).
Syntax : > chmod category operation permission file(s)
Example : $ chmod u –wx exa.c

Category	Operation	Permission
u-users g-group o-others a-all	+ Assign - Remove = Assign absolutely	r – read w – write x - execute

II. GENERAL PURPOSE COMMANDS

1. Command : Banner
Purpose : It is used to display its argument in large letters.
Syntax : banner < string >
Example : $ banner BOOM

2. Command : Who
Purpose : It is used to get the information about all the users currently working in the system.
Syntax : who
Example : $ who

3. Command : Who am i
Purpose : It is used to know in which terminal the user is currently logged on.
Syntax : who am i
Example : $ who am I

4. Command : Date
Purpose : It is used to display the system date and time.
Syntax : date
Example : $ date

5. Command : Cal
Purpose : It prints the calender for the specified year and month.
Syntax : cal <month> <year>
Example : $ cal 05 2003

6. Command : Id
Purpose : It is used to display the login name.
Syntax : id
Example : $ id

7. Command : Clear
Purpose : It is used to clear the screen.
Syntax : clear
Example : $ clear

8. **Command :** Echo
   Purpose : It is used to specify the current path of the operating system.
   Syntax : echo PATH
   Example : $ echo
   PATH

III. COMMAND GROUPING & FILTER COMMANDS

1. **Command :** Head
   Purpose : It is used to display the top portion of the file.
   Syntax : head [options] <file name>
   Example : $ head -5 devi

2. **Command :** Tail
   Purpose : It is used to display the bottom portion of the file.
   Syntax : tail [options] <file
   name > Example : $ tail –5 devi

3. **Command :** Cut
   Purpose : It is used to extract selected fields or columns from each line of one or more files and display them on the standard output device.
   Syntax : cut [options] <file name
   > Example : $ cut –c5 devi

4. **Command :** Paste
   Purpose : It concatenates the line from each input file column by column with tab characters in between them.
   Syntax : paste [options] <file name >
   Example : $ paste f1 f2

5. **Command :** Join
   Purpose : It is used to extracts common lines from two sorted files and there should be the common field in both file.
   Syntax : join [options] <file name1 > <file name 2>
   Example : $ join –a1 f1 f2
6. **Command :** Uniq
   Purpose : It compares adjacent lines in the file and displays the output by eliminating duplicate adjacent lines in it.
   Syntax : uniq [options] <file name >
   Example : $ uniq -c devi

7. **Command :** Sort
   Purpose : It sorts one or more files based on ASCII sequence and also to merge the file.
   Syntax : sort [options] <file name >
   Example : $ sort -r devi

8. **Command :** Tee
   Purpose : It is used to read the contents from standard input or from output of another command and reproduces the output to boyh in standard output and direct into output to one or more files.

Syntax : tee [options] <file name > Example : $ tee date dat.txt

9. **Command :** grep
Purpose : It is used to search the specified pattern from one or more files.
Syntax : grep [options] <pattern> <file name >
Example : $ grep “anand” devi

IV. DIRECTORY COMMANDS

1. **Command :** mkdir
Purpose : It is used to create new directory or more than one directory.
Syntax : mkdir <directory name > Example : $ mkdir riya

2. **Command :** rmdir
Purpose : It is used to remove the directory if it is empty.
Syntax : rmdir <directory name > Example : $ rmdir riya

3. **Command :** cd
Purpose : It is used to change the control from one working directory to another specified directory.
Syntax : cd <directory name >
Example : $ cd riya

4. **Command :** cd ..
Purpose : It is used to quit from current directory and move to the previous directory.
Syntax : cd ..
Example : $ cd ..

V. COMMUNICATION COMMANDS

1. **Command :** Mesg
Purpose : It is used to send the message to another user’s terminal.
Syntax : mesg y
Example : $ mesg y

2. **Command :** Write
Purpose : It is used to communicate with other users that are logged in at the same time.
Syntax : write user_name Example :
$ write user2
Hai (ctrl+D)

3. **Command :** Wall
Purpose : It is used to send message to all users, those who are currently logged in.
Syntax : wall message
Example : $ wall
Hai

4. Command : News

Purpose : It permits user to read messages published on the system admin.

Syntax : news

Example : $

news

5. Command : Mail

Purpose : It is used to send mail to another user.

Syntax : mail

user_name Example :

$ mail user1

Hai

6. Command : Reply

Purpose : It is used to send reply to the specified user.

Syntax : reply

user_name Example :

$ reply user2

Fine

(ctrl+D)

RESULT:

Thus basic UNIX commands in UNIX were studied.

Ex No : 2b	Shell programming
Date :	

Aim :

To write simple shell scripts using shell programming fundamentals.

I) Area and Circumference of Circle

Aim :

To calculate area and circumference of circle using shell program.

Algorithm:

Step 1 : Start

Step 2 : Define constant pi = 3.14

Step 3 : Read the value of radius

Step 4 : Calculate area using formulae: pi × radius2

Step 5 : Calculate circumference using formulae: 2 × pi × radius

Step 6 : Print area and circumference

Step 7 : Stop

Program:

 # Area and Circumference of Circle

```
pi=`expr "scale=2; 22 / 7" | bc` readonly pi
echo  "Enter value for radius : " read radius
area=`expr "scale=2; $pi * $radius * $radius" | bc` circum=`expr "scale=2; 2 * $pi * $radius" | bc`
echo "Area : $area"
echo "Circumference : $circum"
```

Output

```
Enter value for radius : 12
Area : 452.16
Circumference : 75.36
```

Result:

Thus the shell program to calculate area and circumference of circle was executed successfully.

ii) Biggest of three numbers

Aim:

To find the biggest of three numbers using shell program.

Algorithm:

Step 1 : Start
Step 2 : Read values of a, b and
c Step 3 : If a > b and a > c then
Print "A is the biggest"
Step 4 : else if b > c then
Print "B is the biggest "
Step 5 : else
Print "C is the biggest "
Step 6 : Stop

Program:

Biggest of three numbers

```
echo "Give value for A B and
C: " read a b c
if [ $a -gt $b -a $a -gt $c ]
then
echo "A is the Biggest
number" elif [ $b -gt $c ]
then
echo "B is the Biggest number"
else
echo "C is the Biggest
number" fi
```

Output:

```
Give value for A B and C: 4
3 4 C is the Biggest number
```

Result:

Thus the shell program to find biggest of three numbers was executed successfully.

i) Armstrong Number

Aim:

To find the given number is Armstrong number using shell program.

Algorithm:

Step 1 : Start
Step 2 : Read number
Step 3 : Initialize 0 to sum and number to num
Step 4 : Extract lastdigit by computing number modulo 10 Step 5 : Cube the lastdigit and add it to sum
Step 6 : Divide number by 10
Step 7: Repeat steps 4–6 until number > 0 Step 8 : If sum = number then
Print “Armstrong number”
Step 8.1 : else
Print “Not an Armstrong number”
Step 9 : Stop

Program:

```
#Armstrong Number echo  "Enter a number : "  read n
a=$n s=0
while [ $n -gt 0 ] do
r=`expr $n % 10`
s=`expr $s + \( $r \* $r \* $r \)` n=`expr $n / 10`
done
if [ $a -eq $s ] then
echo "Armstrong Number" else
echo  "Not an Armstrong number" fi
```

Output:

Enter a number : 370 Armstrong Number

Result:

Thus the shell program to find the given number is Armstrong number was executed successfully.

ii) Fibonacci Series

Aim:

To find the Fibonacci series of a number using shell program.

Algorithm:

Step 1 : Start
Step 2 : Read number of terms as n
Step 3 : Initialize 0 to f1, 1 to f2 and 2 to i
Step 4 : Print initial fibonacci terms f1, f2
Step 5 : Generate next term using the formula f3 = f1 +
f2 Step 6 : Print f3
Step 7 : Increment i by
1 Step 8 : Assign f2 to
f1 Step 9 : Assign f3 to
f2
Step 10 : Repeat steps 5–9 until i ..
n Step 11 : Stop

Program:

#Fibonacci Series

```
echo  "Enter number of terms : " read n
echo "Fibonacci Series" f1=0
f2=1
echo -n "$f1 " echo -n " $f2 " i=2
while [ $i -lt $n ] do
f3=`expr $f1 + $f2` echo -n " $f3 " f1=$f2
f2=$f3  i=`expr $i + 1` done
echo
```

Output:

```
Enter number of terms : 8 Fibonacci Series
0 1 1 2 3 5 8 13
```

Result:

Thus the shell program to find Fibonacci series of a number was executed successfully.

iii) Prime Number

Aim:

To find the prime number using shell program.

Algorithm:

Step 1 : Start
Step 2 : Read the value of
n Step 3 : Initialize i to 2
Step 4 : If n is divisible by i then
Print “Not Prime” and Stop
Step 5 : Increment i by 1
Step 6 : Repeat steps 4 and 5 until i .. n/2
Step 7 : Print "Prime"
Step 8 : Stop

Program:

#Prime Number

```
echo  "Enter the number : " read n
i=2
m=`expr $n / 2` until [ $i -gt $m ] do
q=`expr $n % $i` if [ $q -eq 0 ] then
echo "Not a Prime number" exit
fi
i=`expr $i + 1` done
echo "Prime number"
```

Output:

Enter the number :
17 Prime number

Result:

Thus the shell program to find prime number was executed successfully.

vii) Factorial Value

Aim:

To find the factorial of a number using shell program.

Algorithm:

Step 1 : Start
Step 2 : Read number
Step 3 : Initialize 1 to fact and number to i
Step 4 : fact = fact * i
Step 5 : Decrement i by 1
Step 6: Repeat steps 4–6 until i
> 0 Step 7 : Print fact
Step 8 : Stop

Program:

```
#Factorial Value echo  "Enter a positive number : " read n
f=1
until [ $n -lt 1 ] do
f=`expr $f \* $n` n=`expr $n - 1` done
echo "Factorial value : $f"
```

Output:

```
[vijai@localhost loops]$ sh fact.sh
Enter a positive number : 10
Factorial value : 3628800
```

Result:

Thus the shell program to find factorial of a number was executed successfully.

viii) Convert Celsius to Fahrenheit

Aim:

To write a Shell program to convert Celsius to Fahrenheit

Algorithm

Step 1 : Start
Step 2 : Read the Celsius value
Step 3 : Convert into Fahrenheit simple interest using the formulae: (principal × rate × years) / 100
Step 4 : Print simple interest
Step 5 : Stop

Program:

```
echo -n "Enter temperature (C) : "
read tc
# formula Tf=(9/5)*Tc+32
tf=$(echo "scale=2;((9/5) * $tc) + 32" |bc) echo "$tc C = $tf F"
```

To write a Shell program to convert Celsius to Fahrenheit

Result:

Thus the shell program **to convert Celsius to Fahrenheit** was executed successfully.

ix) Odd or Even

Aim:

To find odd or even number using shell program.

Algorithm:

Step 1 : Start
Step 2 : Read number
Step 3 : If number divisible by 2 then Print "Number is Even"
Step 4 : else Print "Number is Odd"
Step 5 : Stop

Program:

```
echo  "Enter a non-zero number : " read num
rem=`expr $num % 2` if [ $rem -eq 0 ]
then
echo "$num is Even" else
echo "$num is Odd" fi
```

Output:

```
Enter a non-zero number : 12
12 is Even
```

Result:

Thus the shell program to find odd or even was executed successfully.

x) Date and Time

Aim

To Write a shell program to display date with time.

Algorithm

1. Start the program
2. Format the date
3. Assign the date to the variable
4. Printe the date and time
5. Stop the program

Program

```
date +"%FORMAT"
var=$
(date)
var=`date
` echo
"$var"
```

Result:

Thus the shell program to display date with time was entered and executed successfully

xi) Displaying natural numbers

Aim:

To write a shell program to display first 10 natural numbers.

Algorithm:

Step 1 : Start
Step 2 : Assign the initial value
Step 3: Print the numbers until condition fails
Step 4: Stop

Program:

```
for (( i=1; i<=10; i++ )) do
echo -n "$i "
done echo ""
```

Result:

Thus the above shell program to display first 10 natural numbers was entered and executed successfully.

Ex No : 3a	Fork system call
Date :	

Aim

To create a new child process using fork system call.

Algorithm

1. Declare a variable *x* to be shared by both child and parent.
2. Create a child process using fork system call.
3. If return value is -1 then
   Print "Process creation
   unsuccessfull"Termina
   te using exit system
   call.
4. If return value is 0 then
   Print "Child process"
   Print process id of the child using getpid
   system callPrint value of *x*
   Print process id of the parent using getppid system call
5. Otherwise
   Print "Parent process"
   Print process id of the parent using
   getpid system callPrint value of *x*
   Print process id of the shell using getppid system call.
6. Stop

Program :

```
#include <stdio.h> #include<stdlib.h> #include <unistd.h>
#include <sys/types.h> main()
{
pid_t            pid; int x = 5; pid = fork();x++; if (pid < 0)
{
printf("Process creation error"); exit(-1);
}
else if (pid == 0)
{
printf("Child process:");
 printf("\nProcess id is %d", getpid());
printf("\nValue of x is%d", x);
                        printf("\nProcess id of parent is %d\n", getppid());
}
else
{
                        printf("\nParent process:"); printf("\nProcess id is %d", getpid());
                        printf("\nValue of x is %d", x);
}                       printf("\nProcess id of shell is %d\n", getppid());
}
```

Output

```
$ gcc fork.c
$  ./a.out Child process:
Process id is 19499Value of x is 6
Process id of parent is 19498

Parent process: Process id is 19498Value of x is 6
Process id of shell is 3266
```

Result

Thus a child process is created with copy of its parent's address space.

Ex No : 3b	Wait system call
Date :	

Aim

To block a parent process until child completes using wait system call.

Algorithm

1. Create a child process using fork system call.
2. If return value is -1 then
   A) Print "Process creation unsuccessfull"
3. Terminate using exit system call.
4. If return value is > 0 then
   a) Suspend parent process until child completes using wait system call
   b) Print "Parent starts"
   c) Print even numbers from 0–10
   d) Print "Parent ends"
5. If return value is then
   a) Print "Child starts"
   b) Print odd numbers from 0–10
   c) Print "Child ends"
6. Stop

Program

```
/* Wait for child termination - wait.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

main()
{
int i, status;pid_t pid; pid = fork();

if (pid < 0)
{
printf("\nProcess creation failure\n");exit(-1);
}
else if(pid > 0)
{
wait(NULL);
```

```
printf ("\nParent starts\nEven Nos: ");for (i=2;i<=10;i+=2)
printf  ("%3d",i);
printf ("\nParent ends\n");
}
else if (pid == 0)
{
printf ("Child starts\nOdd Nos: ");for (i=1;i<10;i+=2)
printf  ("%3d",i); printf ("\nChild ends\n");
}
}
```

Output :

```
$ gcc wait.c
$ ./a.out Child starts
Odd Nos: 1 3 5 7 9
Child ends
Parent starts
Even Nos: 2 4 6 8 10
Parent ends
```

Result

Thus using wait system call zombie child processes were avoided.

Ex No : 3c	Exec system call
Date :	

Aim

To load an executable program in a child processes exec system call.

Algorithm

1. Create a child process using fork system call.
2. If return value is -1 then
   a. Print "Process creation unsuccessfull"
3. Terminate using exit system call.
4. If return value is > 0 then
   a. Suspend parent process until child completes using wait system call
   b. Print "Child Terminated".
   c. Terminate the parent process.
5. If return value is 0 then
   d. Print "Child starts"
   e. Load date program into child process using exec system call.
   f. Terminate the child process.
6. Stop

Program

/* Load a program in child process - exec.c */

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

main()
{
pid_t pid;

switch(pid = fork())
{
case -1:
perror("Fork failed");exit(-1);

case 0:
printf("Child process\n"); execl("/bin/date", "date", 0);exit(0);
```

```
default:
wait(NULL);
printf("Child  Terminated\n");exit(0);

}
}
```

Output

```
$ gcc exec.c

$ ./a.out Child process
Sat Feb 23 17:46:59 IST 2013
Child Terminated
```

Result

Thus the child process loads a binary executable file into its address space.

Ex No : 3d	Stat and exit system call
Date :	

Aim

To display file status using stat system call.

Algorithm

1. Get *filename* as command line argument.
2. If *filename* does not exist then stop.
3. Call stat system call on the *filename* that returns a structure
4. Display members st_uid, st_gid, st_blksize, st_block, st_size, st_nlink, etc.,
5. Convert time members such as st_atime, st_mtime into time using ctime function
6. Compare st_mode with mode constants such as S_IRUSR, S_IWGRP, S_IXOTH anddisplay file permissions.
7. Stop

Program

```
/* File status - stat.c */
#include<stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char*argv[])
{
struct stat file;int n; if (argc != 2)
{
printf("Usage: ./a.out <filename>\n");exit(-1);
}
if ((n = stat(argv[1], &file)) == -1)
{
perror(argv[1]);
exit(-1);
}
printf("User id : %d\n", file.st_uid);
printf("Group id : %d\n", file.st_gid);
printf("Block size : %d\n", file.st_blksize);
printf("Blocks allocated : %d\n", file.st_blocks);
printf("Inode no. : %d\n", file.st_ino);
printf("Last accessed : %s", ctime(&(file.st_atime)));
printf("Last modified : %s", ctime(&(file.st_mtime)));
printf("File size : %d bytes\n", file.st_size);
printf("No. of links : %d\n", file.st_nlink);
printf("Permissions : ");
printf( (S_ISDIR(file.st_mode)) ? "d" : "-");
```

```
printf( (file.st_mode & S_IRUSR) ? "r" : "-");
printf( (file.st_mode & S_IWUSR) ? "w" : "-");
printf( (file.st_mode & S_IXUSR) ? "x" : "-");

printf( (file.st_mode & S_IRGRP) ? "r" : "-");
printf( (file.st_mode & S_IWGRP) ? "w" : "-");
printf( (file.st_mode & S_IXGRP) ? "x" : "-");
printf( (file.st_mode & S_IROTH) ? "r" : "-");
printf( (file.st_mode & S_IWOTH) ? "w" : "-");
printf( (file.st_mode & S_IXOTH) ? "x" : "-");
printf("\n");

if(file.st_mode & S_IFREG) printf("File type : Regular\n");
if(file.st_mode & S_IFDIR) printf("File type : Directory\n");
}
```

Output

```
$ gcc stat.c

$ ./a.out fork.c
User id  :  0
 Group id : 0
Block size : 4096
Blocks allocated : 8I
node no. : 16627
Last accessed : Fri Feb 22 21:57:09 2013
Last modified : Fri Feb 22 21:56:13 2013
File size : 591 bytes
No. of links : 1 Permissions : - rw-r—r--
File type : Regular
```

Result

Thus attributes of a file is displayed using stat system call.

Ex No : 3e	Open system call
Date :	

Aim

To create a file and to write contents.

open()

- Used to open an existing file for reading/writing or to create a new file.
- Returns a file descriptor whose value is negative on error.
- The mandatory flags are O_RDONLY, O_WRONLY and O_RDWR
- Optional flags include O_APPEND, O_CREAT, O_TRUNC, etc
- The flags are ORed.
- The mode specifies permissions for the file.

creat()

- Used to create a new file and open it for writing.
- It is replaced with open() with flags O_WRONLY|O_CREAT | O_TRUNC

Algorithm

1. Declare a character buffer *buf* to store 100 bytes.
2. Get the new filename as command line argument.
3. Create a file with the given name using open system call with O_CREAT and O_TRUNC options.
4. Check the file descriptor.
   a) If file creation is unsuccessful, then stop.
5. Get input from the console until user types Ctrl+D
   a) Read 100 bytes (max.) from console and store onto buf using read system call
   b) Write length of *buf* onto file using write system call.
6. Close the file using close system call.
7. Stop

Program

```
/* File creation - fcreate.c */
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
int fd, n, len;char buf[100];

if (argc != 2) {
printf("Usage: ./a.out <filename>\n");exit(-1);
}
```

```
fd = open(argv[1], O_WRONLY|O_CREAT|O_TRUNC, 0644);

if(fd < 0)  {
printf("File creation problem\n");exit(-1);
}

printf("Press Ctrl+D at end in a new line:\n");

while((n = read(0, buf, sizeof(buf))) > 0)
{
len = strlen(buf); write(fd, buf, len);
}
close(fd);
}
```

Output

```
$ gcc fcreate.c

$ ./a.out helloFile I/O
Open system call is used to either open or create a file.
create system call is used  to create a file. It is seldom used.
^D
```

Result

Thus a file has been created with input from the user. The process can be verified by Using cat command

Ex No : 4	Implement the following CPU scheduling algorithms
Date :	

4a. Round Robin

AIM:

To write a program to implement the Round Robin CPU scheduling Algorithm

ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time(BT) of each processors
4. Get the Quantum time(QT)
5. Execute each processor until reach the QT or BT
6. Time of reaching process Us BT is its Turn Around Time(TAT)
7. Time waits to start the execution,is the waiting time(WT) of each processor
8. Calculation of Turn Around Time and Waiting Time tot_TAT = tot_TAT + cur_TAT
   avg_TAT = tot_TAT/num_of_proc tot_WT = tot_WT + cur_WT
   avg_WT = tot_WT/num_of_proc
9. Display the result
10. STOP the program

PROGRAM: (Round Robin Algorithm)

```
 #include<stdio.h>
#include<conio.h> int TRUE = 0;
int FALSE = -1;
int tbt[30],bt[30],tat[30],n=0,wt[30],qt=0,tqt=0,time=0,lmore,t_tat=0,t_wt=0;
void main()
{
int i,j; clrscr();
printf("\nEnter no. of processors:"); scanf("%d",&n);
printf("\nEnter Quantum Time:"); scanf("%d",&qt);  for(i=0;i<n;i++)
{
printf("\nEnter Burst Time of Processor[%d]:",i+1); scanf("%d",&bt[i]);
tbt[i] = bt[i];
wt[i] = tat[i] = 0;
}lmore = TRUE; while(lmore == TRUE)
{
lmore = FALSE; for(i=0;i<n;i++)
{
if(bt[i]  != 0)
wt[i] = wt[i] + (time - tat[i]); tqt = 1; while(tqt  <= qt && bt[i] !=0)
{
lmore = TRUE; bt[i] = bt[i] -1; tqt++;
time++; tat[i] = time;
```

```
}
}
}

printf("\nProcessor ID\tBurstTime\tTurnAroundTime\tWaitingTime\n");
for(i=0;i<n;i++)
{
printf("Processor%d\t\t%d\t\t%d\t\t%d\n",i+1,tbt[i],tat[i],wt[i]);
 t_tat = t_tat + tat[i]; t_wt = t_wt + wt[i];
}
printf("\nTotal Turn Around Time:%d",t_tat);
 printf("\nAverage Turn Around Time:%d",t_tat/n);
printf("\nTotal Waiting Time:%d",t_wt);
printf("\nAverage Waiting Time:%d",t_wt/n);
getch();
}
```

OUTPUT: (Round Robin Scheduling Algorithm)

```
Enter no. of processors: 5
Enter Quantum Time: 6
Enter Burst Time of Processor [1]:21
Enter Burst Time of Processor [2]:18
Enter Burst Time of Processor [3]:12
Enter Burst Time of Processor [4]:30
Enter Burst Time of Processor [5]:15
```

Processor ID	BurstTime	TurnAroundTime	WaitingTime
Processor1	21	84	63
Processor2	18	72	54
Processor3	12	48	36
Processor4	30	96	66
Processor5	15	81	66

```
Total Turn Around Time: 381
Average Turn Around Time: 76
Total Waiting Time: 285
Average Waiting Time: 57
```

RESULT:

Thus the program to implement the Round Robin CPU scheduling Algorithm was written, executed and the output was verified successfully.

4b. Shortest Job First

AIM:

To write a program to implement the SJF (Shortest Job First) CPU scheduling Algorithm

ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time of each processors
4. Sort the processors based on the burst time
5. Calculation of Turn Around Time and Waiting Time
   e) tot_TAT = tot_TAT + pre_TAT
   f) avg_TAT = tot_TAT/num_of_proc
   g) tot_WT = tot_WT + pre_WT + PRE_BT
   h) avg_WT = tot_WT/num_of_proc
6. Display the result
7. STOP the program

PROGRAM: (SJF Scheduling)

```
#include<stdio.h> #include<conio.h>
int p[30],bt[30],tot_tat=0,wt[30],n,tot_wt=0,tat[30],SJF_wt=0,SJF_tat=0; float
awt,avg_tat,avg_wt;
void main()
{
int i; clrscr();
printf("\nEnter the no.of processes \n"); scanf("%d",&n);
printf("Enter burst time for each process\n"); for(i=0;i<n;i++)
{
scanf("%d",&bt[i]); p[i] = i;
}
sort();
WT_TA T(& SJF _tat,& SJF _ wt);
printf("\n\nTotal Turn around Time:%d",SJF_tat); printf("\nAverage Turn around Time
:%d ", SJF_tat/n); printf("\nTotal Waiting Time:%d",SJF_wt);
printf("\nTotal avg. Waiting Time:%d",SJF_wt/n); getch();
}
int sort()
{
int t,i,j; for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(bt[i]>bt[j])
{
swap(&bt[j],&bt[i]);
swap(&p[j],&p[i]);
}
}
}
```

```
return 0;
}
int swap(int *a, int *b)
{
int t;
t = *a;
*a = *b;
*b = t; return 0;
}
int WT_TAT(int  *a, int *b)
{
int i; for(i=0;i<n;i++)
{
if(i==0)
tat[i] = bt[i]; else
tat[i] = tat[i-1] + bt[i]; tot_tat=tot_tat+tat[i];
}
*a = tot_tat; wt[0]=0; for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1]; tot_wt = tot_wt+wt[i];
}
*b = tot_wt;
printf("\nPROCESS\t\tBURST  TIME\tTURN  AROUND TIME\tWAITING  TIME");
for(i=0; i<n; i++) printf("\nprocess[%d]\t\t%d\t\t%d\t\t%d",p[i+1,bt[i],tat[i],wt[i]);
return 0;
}
```

OUTPUT: (SJF Scheduling Algorithm)

Enter the no. of processes 5
Enter burst time for each process 45
32
21
67
89

PROCESS	BURST TIME	TRUN AROUND TIME	WAITING TIME
Process [3]	21	21	0
Process [2]	32	53	21
Process [1]	45	98	53
Process [4]	67	165	98
Process [5]	89	254	165

Total Turn around Time: 591
Average Turn around Tim: 118
Total Waiting Time: 337
Total avg.Waiting Time: 67

RESULT:

Thus the program to implement the SJF (Shortest Job First) CPU scheduling Algorithm was written, executed and the output was verified successfully.

4c. First Come Fist Serve

AIM:

To write a program to implement the FCFS (First Come First Serve) CPU scheduling Algorithm

ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time of each processors
4. Calculation of Turn Around Time and Waiting Time
    a) tot_TAT = tot_TAT + pre_TAT
    b) avg_TAT = tot_TAT/num_of_proc
    c) tot_WT = tot_WT + pre_WT + PRE_BT
    d) avg_WT = tot_WT/num_of_proc
5. Display the result
6. STOP the program

PROGRAM: (FCFS Scheduling)

```
#include<stdio.h>
#include<conio.h>
int
p[30],bt[30],tot_tat=0,wt[30],n,tot_wt=0,tat[30],FCFS_wt=0,FCFS_tat=0;
float awt,avg_tat,avg_wt;
void main()
{
int i;
clrscr
();
printf("\nEnter the no.of processes \n");
scanf("%d",&n);
printf("Enter burst time for each process\n");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i])
; p[i] = i;
}
printf("\n FCFS Algorithm \n");
WT_TAT(&FCFS_tat,&FCFS_wt);  printf("\n\
nTotal Turn around Time:%d",FCFS_tat);
printf("\nAverage Turn around Time :%d ", FCFS_tat/n); printf("\nTotal
Waiting Time:%d",FCFS_wt); printf("\nTotal avg. Waiting Time:
%d",FCFS_wt/n); getch();
}
int WT_TAT(int  *a, int *b)
{
int i; for(i=0;i<n;i++)
{
if(i==0)
tat[i] =
bt[i]; else
tat[i] = tat[i-1] + bt[i];
tot_tat=tot_tat+tat[i];
```

```
}
*a = tot_tat; wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-
1]; tot_wt =
tot_wt+wt[i];
}
*b = tot_wt;
printf("\nPROCESS\t\tBURST TIME\tTURN AROUND TIME\tWAITING TIME");
for(i=0; i<n; i++) printf("\nprocess[%d]\t\t%d\t\t%d\t\t
%d",p[i],bt[i],tat[i],wt[i]); return 0;
}
```

OUTPUT: (FCFS Scheduling Algorithm)

Enter the no.of processes
5
Enter burst time for each process 12
14
2
6
5

FCFS Algorithm

PROCESS	BURST TIME	TURN AROUND TIME	WAITING TIME
process[0]	12	12	0
process[1]	14	26	12
process[2]	2	28	26
process[3]	6	34	28
process[4]	5	39	34

Total Turn around Time:139
Average Turn around Time :27
Total Waiting Time:100
Total avg. Waiting Time:20

RESULT:

Thus the program to implement the FCFS (First Come First Serve) CPU scheduling Algorithm was written, executed and the output was verified successfully.

4d. Priority

AIM:

To write a program to implement the Priority CPU scheduling Algorithm

ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time of each processors
4. Get the priority of all the processors
5. Sort the processors based on the priority
6. Calculation of Turn Around Time and Waiting Time
    i) tot_TAT = tot_TAT + pre_TAT
    j) avg_TAT = tot_TAT/num_of_proc
    k) tot_WT = tot_WT + pre_WT + PRE_BT
    l) avg_WT = tot_WT/num_of_proc
7. Display the result
8. STOP the program

PROGRAM: (Priority Scheduling)

```
#include<stdio.h>
#include<conio.h>
int
p[30],bt[30],tot_tat=0,pr[30],wt[30],n,tot_wt=0,tat[30],PR_wt=0,PR_tat=0;
float awt,avg_tat,avg_wt;
void main()
{
int i;
clrscr
();
printf("\nEnter the no.of processes \n"); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter burst time and priority of process[%d]:",i+1); scanf("%d
%d",&bt[i],&pr[i]);
p[i] = i;
}
sort();
WT_TA T(& PR_ tat,& PR_ wt);
printf("\n\nTotal Turn around Time:%d",PR_tat); printf("\nAverage Turn around Time :%d ",
PR_tat/n);
printf("\nTotal Waiting Time:%d",PR_wt); printf("\nTotal avg. Waiting Time:%d",PR_wt/n);
getch();
}
int sort()
{
int t,i,j,t2,t1; for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
```

```
{
if(pr[i]>pr[j])
{
swap(&bt[j],&bt[i]);
swap(&p[j],&p[i]);
swap(&pr[j],&pr[i]);
}
}
}
return 0;
}
int swap(int *a, int *b)
{
int t;
t = *a; *a = *b; *b
= t; return 0;
}
int WT_TAT(int  *a, int *b)
{
int i;
for(i=0;i<n;i+
+)
{
if(i==0)
tat[i] =
bt[i]; else
tat[i] = tat[i-1] + bt[i];
tot_tat=tot_tat+tat[i];
}
*a = tot_tat;
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-
1]; tot_wt =
tot_wt+wt[i];
}
*b = tot_wt;
printf("\nPROCESS\t\tBURST TIME\tPRIORITY\tTURN AROUND TIME\tWAITING TIME");
for(i=0; i<n; i++) printf("\nprocess[%d]\t\t%d\t\t%d\t\t%d\t\t%d",p[i]
+1,bt[i],pr[i],tat[i],wt[i]); return 0;
}
```

OUTPUT: (Priority Scheduling Algorithm)

Enter the no.of processes 5
Enter burst time and priority of process[1]:2 3
Enter burst time and priority of process[2]:4 5
Enter burst time and priority of process[3]:6 3
Enter burst time and priority of process[4]:4 8
Enter burst time and priority of process[5]:3 8

PROCESS	BURST TIME	PRIORITY	TURN AROUND TIME	WAITING TIME
process[1]	2	3	2	0
process[3]	6	3	8	2
process[2]	4	5	12	8
process[4]	4	8	16	12
process[5]	3	8	19	16

Total Turn around Time:57 Average Turn around Time :11 Total Waiting Time:38
Total avg. Waiting Time:7

RESULT:

Thus the program to implement the priority CPU scheduling Algorithm was written, executed and the output was verified successfully.

Ex No : 5	Implement Shared memory and IPC
Date :	

AIM:

To implement the Interprocess communication using shared memory.

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. shmat() and shmdt() are used to attach and detach shared memory segments. They are prototypes as follows:
   void *shmat(int shmid, const void *shmaddr, int shmflg); int shmdt(const void *shmaddr);
4. shmat() returns a pointer, shmaddr, to the head of the shared segment associated with a valid

shmid. shmdt() detaches the shared memory segment located at the address indicated by shmaddr

5. Shared1.c simply creates the string and shared memory portion.
6. Shared2.c attaches itself to the created shared memory portion and uses the string (printf)
7. Stop the program.

PROGRAM:

Shared1.c:

```
#include
<sys/types.h>
#include
<sys/ipc.h>
#include
<sys/shm.h>
#include <stdio.h>
#define SHMSZ  27
main()
{
char c;
int shmid;
  key_t key; char *shm, *s; key = 5678;
   if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) { perror("shmget");
      exit(1);
   }
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -
      1) { perror("shmat");
      exit(1);
      }}
```

Shared2.c

```
#include <sys/ipc.h> #include
<sys/shm.h> #include <stdio.h> #define
SHMSZ 27 main()
{
   int shmid; key_t key; char *shm,
  *s; key = 5678;
   if ((shmid = shmget(key, SHMSZ, 0666)) < 0) { perror("shmget");
      exit(1);
   }
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) { perror("shmat");
      exit(1);
   }
   for (s = shm; *s != NULL; s++) putchar(*s);
   putchar('\n'); *shm = '*'; exit(0);
}
```

OUTPUT:

Abcdefghijklmnopqrstuvwxyz

RESULT:

Thus the program has been executed successfully and the output is verified.

Ex No : 6	Implementation of semaphore
Date :	

Aim

To synchronize producer and consumer processes using semaphore.

Algorithm

1. Create a shared memory segment *BUFSIZE* of size 1 and attach it.
2. Obtain semaphore id for variables *empty*, *mutex* and *full* using semget function.
3. Create semaphore for *empty*, *mutex* and *full* as follows:
   a. Declare *semun*, a union of specific commands.
   b. The initial values are: 1 for mutex, N for empty and 0 for full
   c. Use semctl function with SETVAL command
4. Create a child process using fork system call.
   a. Make the parent process to be the *producer*
   b. Make the child process to the *consumer*
5. The *producer* produces 5 items as follows:
   a. Call *wait* operation on semaphores *empty* and *mutex* using semop function.
   b. Gain access to buffer and produce data for consumption
   c. Call *signal* operation on semaphores *mutex* and *full* using semop function.
6. The *consumer* consumes 5 items as follows:
   a. Call *wait* operation on semaphores *full* and *mutex* using semop function.
   b. Gain access to buffer and consume the available data.
   c. Call *signal* operation on semaphores *mutex* and *empty* using semop function.
7. Remove shared memory from the system using shmctl with IPC_RMID argument
8. Stop

Program

```
/* Producer-Consumer problem using semaphore – pcsem.c */#include <stdio.h>
      #include  <stdlib.h>
#include  <sys/types.h> #include
      <sys/ipc.h> #include
<sys/shm.h> #include
<sys/sem.h>

#define N 5
#define BUFSIZE 1
#define PERMS 0666

int *buffer;
int nextp = 0, nextc = 0;
int mutex, full, empty;                    /* semaphore variables */

void producer()
      {
     int data; if(nextp ==
           N)
          nextp = 0;
           printf("Enter data for producer to produce : ");scanf("%d",(buffer + nextp));
           nextp++;
      }


void consumer()
      {
     int  g; if(nextc == N)
          nextc = 0;
     g = *(buffer + nextc++); printf("\nConsumer consumes
           data %d", g);
      }


void sem_op(int id, int value)
      {
     struct sembuf op;int v;
           op.sem_num = 0;
           op.sem_op
     = value; op.sem_flg =
           SEM_UNDO;
     if((v = semop(id, &op, 1)) < 0)
          printf("\nError executing semop instruction");
      }
      void sem_create(int semid, int initval)
      {
           int semval;union
           semun
           {
                int val;
                struct semid_ds *buf;
                unsigned short *array;
```

```
    } s;

    s.val = initval;
    if((semval = semctl(semid, 0, SETVAL, s)) < 0)printf("\nError in executing
        semctl");
}

void sem_wait(int id)
{
    int value = -1; sem_op(id,
    value);
}

void sem_signal(int id)
{
    int value = 1; sem_op(id,
    value);
}

main()
{
    int shmid, i;pid_t
    pid;

    if((shmid = shmget(1000, BUFSIZE, IPC_CREAT|PERMS)) < 0)
    {
        printf("\nUnable to create shared memory");return;
    }
    if((buffer = (int*)shmat(shmid, (char*)0, 0)) == (int*)-1)
    {
        printf("\nShared memory allocation error\n");exit(1);
    }

    if((mutex = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
    {
        printf("\nCan't create mutex semaphore");exit(1);
    }
    if((empty = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
    {
        printf("\nCan't create empty semaphore");exit(1);
    }
    if((full = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
    {
        printf("\nCan't create full semaphore");exit(1);
    }

    sem_create(mutex,           1);
    sem_create(empty,

    N); sem_create(full, 0);
```

```
    if((pid = fork()) < 0)
    {
        printf("\nError in process creation");exit(1);
    }
    else if(pid > 0)
    {
        for(i=0; i<N; i++)
        {
            sem_wait(empty);
            sem_wait(mutex);
            producer();
            sem_signal(mutex);
            sem_signal(full);
        }
    }
    else if(pid == 0)
    {
        for(i=0; i<N; i++)
        {
            sem_wait(full); sem_wait(mutex);
            consumer(); sem_signal(mutex);
            sem_signal(empty);
        }
        printf("\n");
    }
}
```

Output

```
$ gcc pcsem.c

$ ./a.out
Enter data for producer to produce : 5

Enter data for producer to produce : 8Consumer
consumes data 5
Enter data for producer to produce : 4Consumer
consumes data 8
Enter data for producer to produce : 2Consumer
consumes data 4
Enter data for producer to produce : 9Consumer
consumes data 2
Consumer consumes data 9
```

Result

Thus synchronization between producer and consumer process for access to a sharedmemory segment is implemented.

Ex No : 7	Implement Bankers Algorithm for Dead Lock Avoidance
Date :	

AIM:

T o implement the Bankers Algorithm for Deadlock Avoidance.

ALGORITHM:

1. start the program
2. Get the required value
3. Check the condition block[p]=0 and eun[p]=0
4. Check the value is greater than or equal to 0
5. If it is fail deadlock will occur else
6. If m=0,res reg[p] [k]=newreg[k] deadlock will not occur
7. Stop the program.

PROGRAM:

```
#include<stdio.h> #include<conio.h> void main()
{
int n,r,i,j,k,p,u=0,s=0,m;
int block[10],run[10],active[10],newreq[10];
int max[10][10],resalloc[10][10],resreq[10][10];
int totalloc[10],totext[10],simalloc[10];
clrscr();
printf("Enter the no of processes:"); scanf("%d",&n);
printf("Enter the no of resource classes:"); scanf("%d",&r);
printf("Enter the total existed resource in each class:");
for(k=1;k<=r;k++)
scanf("%d",&totext[k]);
printf("Enter the allocated resources:");
 for(i=1;i<=n;i++)
for(k=1;k<=r;k++) scanf("%d",&resalloc);
printf("Enter the process making the new request:"); scanf("%d",&p);
printf("Enter the requested resource:");
for(k=1;k<=r;k++) scanf("%d",&newreq[k]);
printf("Enter the processes which are n blocked or running:");
 for(i=1;i<=n;i++)
{
if(i!=p)
{
printf("process %d:\n",i); scanf("%d%d",&block[i],&run[i]);
}
}
block[p]=0;
run[p]=0;
for(k=1;k<=r;k++)
{ j=0;
for(i=1;i<=n;i++)
{
```

```
totalloc[k]=j+resalloc[i][k]; j=totalloc[k];
}
}
for(i=1;i<=n;i++)
{
if(block[i]==1||run[i]==1) active[i]=1;
else active[i]=0;
}
for(k=1;k<=r;k++)
{
resalloc[p][k]+=newreq[k]; totalloc[k]+=newreq[k];
}
for(k=1;k<=r;k++)
{
if(totext[k]-totalloc[k]<0)
{
u=1;break;
}
}
if(u==0)
{
for(k=1;k<=r;k++) simalloc[k]=totalloc[k];
 for(s=1;s<=n;s++) for(i=1;i<=n;i++)
{
if(active[i]==1)
{ j=0;
for(k=1;k<=r;k++)
{
if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
{
j=1;break;
}
}
}
if(j==0)
{
active[i]=0; for(k=1;k<=r;k++) simalloc[k]=resalloc[i][k];
}
} m=0;
for(k=1;k<=r;k++) resreq[p][k]=newreq[k]; printf("Deadlock willn't occur");
}
else
{
for(k=1;k<=r;k++)
{
resalloc[p][k]=newreq[k]; totalloc[k]=newreq[k];
}
printf("Deadlock will occur");
}
getch();
}
```

OUTPUT:

Input:

Enter the no of resources: 4
Enter the no of resource classes: 3
Enter the total existed resources in each class: 3 2 2 Enter the allocated resources: 1 0 0 5 1 1 2 1 1 0 0 2 Enter the process making the new request: 2
Enter the requested resource: 1 1 2
Enter the processes which are n blocked or running:
Process 1: 1 2
Process 3: 1 0
Process 4: 1 0

Output:

Deadlock will occur

RESULT:

Thus the program has been executed successfully and the output is verified.

Ex No : 8	Implement an Algorithm for Dead Lock Detection
Date :	

AIM:

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

PROGRAM:

```
#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<=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 has been executed successfully and the output is verified.

Ex No : 9	Memory allocation techniques First Fit Allocation
Date :	

Aim

To allocate memory requirements for processes using first fit allocation.

Algorithm

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
    a. If hole size > process size then
        i. Mark process as allocated to that hole.
        ii. Decrement hole size by process size.
    b. Otherwise check the next from the set of hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Program

/* First fit allocation - ffit.c *

```
/#include <stdio.h> struct  process
{
int size; int flag; int holeid;
} p[10];
struct hole
{
int size; int actual;
} h[10];

main()
{
int i, np, nh, j;

printf("Enter the number of Holes : ");

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

```
for(i=0; i<nh; i++)
{
printf("Enter size for hole H%d : ",i);
scanf("%d", &h[i].size);
h[i].actual = h[i].size;
}

printf("\nEnter number of process : " );scanf("%d",&np); for(i=0;i<np;i++)
{
printf("enter the size of process P%d : ",i);scanf("%d", &p[i].size); p[i].flag = 0;
}

for(i=0; i<np; i++)
{
for(j=0; j<nh; j++)
{
if(p[i].flag != 1)
{
if(p[i].size <= h[j].size)
{
p[i].flag = 1; p[i].holeid = j;
h[j].size -= p[i].size;
}
}
}
}

printf("\n\tFirst fit\n"); printf("\nProcess\tPSize\tHole");for(i=0; i<np; i++)
{
if(p[i].flag != 1)
printf("\nP%d\t%d\tNot allocated", i, p[i].size);
else printf("\nP%d\t%d\tH%d", i, p[i].size, p[i].holeid);
}
printf("\n\nHole\tActual\tAvailable");
for(i=0; i<nh ;i++) printf("\nH%d\t%d\t%d", i, h[i].actual, h[i].size);printf("\n");
}
```

Output :

```
Enter number o          f process : 4
enter the size                  of process P0 :         212
enter the size                  of process P1 :         417
enter the size                  of process P2 :         112
enter the size                  of process P3 :         426

            First           fit

Process PSizeP0             Hole
            212             H1
P1          417             H4
P2          112             H1
P3          426             Not allocated
Hole        Actual          Available
H0          100             100
H1          500             176
H2          200             200
H3          300             300
H4          600             183
```

Result

Thus processes were allocated memory using first fit method.

9b. Best Fit Allocation

Aim

To allocate memory requirements for processes using best fit allocation.

Best fit

- Allocate the smallest hole that is big enough.
- The list of free holes is kept sorted according to size in ascending order.
- This strategy produces smallest leftover holes

Algorithm

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
    a. Sort the holes according to their sizes in ascending order
    b. If hole size > process size then
        i. Mark process as allocated to that hole.
        ii. Decrement hole size by process size.
    c. Otherwise check the next from the set of sorted hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Program

```
#include <stdio.h>

struct process
{
int size; int flag; int holeid;
} p[10];

struct hole
{
int  hid; int size; int actual;
} h[10];
```

```
main()
{
int i, np, nh, j;
void bsort(struct hole[], int); printf("Enter the number of Holes : ");
scanf("%d", &nh); for(i=0; i<nh; i++)
{
printf("Enter size for hole H%d : ",i); scanf("%d", &h[i].size);
h[i].actual =                h[i].size; h[i].hid = i;
}
printf("\nEnter number of process : " ); scanf("%d",&np);
for(i=0;i<np;i++)
{
printf("enter the size of process P%d : ",i);
scanf("%d", &p[i].size); p[i].flag = 0;
}
for(i=0; i<np; i++)
{
bsort(h, nh); for(j=0; j<nh; j++)
{
if(p[i].flag != 1)
{
if(p[i].size <= h[j].size)
{
p[i].flag = 1; p[i].holeid = h[j].hid;
h[j].size -= p[i].size;
}
}
}
}
printf("\n\tBest fit\n"); printf("\nProcess\tPSize\tHole");
for(i=0; i<np; i++)
{
if(p[i].flag != 1)
printf("\nP%d\t%d\tNot allocated", i, p[i].size);
else printf("\nP%d\t%d\tH%d", i, p[i].size, p[i].holeid);
}
printf("\n\nHole\tActual\tAvailable");
for(i=0; i<nh ;i++) printf("\nH%d\t%d\t%d", h[i].hid, h[i].actual,h[i].size);
printf("\n");
}

void bsort(struct hole bh[], int n)
{
struct hole temp;int i,j; for(i=0; i<n-1; i++)
{
for(j=i+1; j<n; j++)
{
if(bh[i].size > bh[j].size)
{
```

```
temp = bh[i];  bh[i] = bh[j];bh[j]
= temp;
}
}
}
}
```

Output

```
Enter the number of Holes : 5Enter size for hole H0 : 100
Enter size for hole H1 : 500 Enter size for hole H2 :
200 Enter size for hole H3 : 300 Enter size for hole H4 : 600
Enter number of process : 4
enter the size of process P0 : 212
enter the size of process P1 : 417
enter the size of process P2: 112
enter the size of process P3 : 426
 Best fit
```

Process	PSize	Hole
P0	212	H3
P1	417	H1
P2	112	H2
P3	426	H4

Hole	Actual	Available
H1	500	83
H3	300	88
H2	200	88
H0	100	100
H4	600	174

Result

Thus processes were allocated memory using best fit method.

Ex No : 10 a	FIFO
Date :	

Aim

To implement demand paging for a reference string using FIFO method.

FIFO

- Page replacement is based on when the page was brought into memory.
- When a page should be replaced, the oldest one is chosen.
- Generally, implemented using a FIFO queue.
- Simple to implement, but not efficient.
- Results in more page faults.
- The page-fault may increase, even if frame size is increased (Belady's anomaly)

Algorithm

1. Get length of the reference string, say *l*.
2. Get reference string and store it in an array, say *rs*.
3. Get number of frames, say *nf*.
4. Initalize *frame* array upto length *nf* to -1.
5. Initialize position of the oldest page, say *j* to 0.
6. Initialize no. of page faults, say *count* to 0.
7. For each page in reference string in the given order, examine:
    a. Check whether page exist in the *frame* array
    b. If it does not exist then
        i. Replace page in position *j*.
        ii. Compute page replacement position as (*j*+1) modulus *nf*.
        iii. Increment *count* by 1.
        iv. Display pages in *frame* array.
8. Print *count*.
9. Stop

Program

```
#include <stdio.h>

main()
{
    int  i,j,l,rs[50],frame[10],nf,k,avail,count=0;

    printf("Enter length of ref. string : ");scanf("%d", &l);
    printf("Enter reference string :\n");for(i=1; i<=l; i++)
        scanf("%d",  &rs[i]); printf("Enter
    number of frames : ");scanf("%d", &nf);
    for(i=0; i<nf; i+
        +) frame[i] =
        -1;
```

```
        j = 0;
        printf("\nRef. str            Page
        frames"); for(i=1; i<=l; i++)
        {
            printf("\n%4d\t", rs[i]);avail
            = 0;
            for(k=0; k<nf; k++) if(frame[k]
                == rs[i])
                    avail = 1;
            if(avail == 0)
            {
                frame[j] = rs[i];j =
                (j+1) % nf; count+
                +;
                for(k=0; k<nf; k++) printf("%4d",
                    frame[k]);
            }
        }
        printf("\n\nTotal no. of page faults : %d\n",count);
    }
```

Output

Enter length of ref. String : 20
Enter reference string :
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3
Enter number of frames : 5
Ref.

str Page frames

```
1     1   -1   -1   -1   -1
2     1    2   -1   -1   -1
3     1    2    3   -1   -1
4     1    2    3    4   -1
      2 1
```

```
5    1     2    3    4          5
6    6     2    3    4          5
2
1    6     1    3    4          5
2    6     1    2    4          5
3    6     1    2    3          5
7    6     1    2    3          7
6
3
```

Total no. of page faults : 10

Result

Thus page replacement was implemented using FIFO algorithm.

Ex No : 10 b	LRU Page Replacement
Date :	

Aim

To implement demand paging for a reference string using LRU method.

LRU

- **Pages used in the recent past are used as an approximation of future usage.**
- **The page that has not been used for a longer period of time is replaced.**
- **LRU is efficient but not optimal.**
- **Implementation of LRU requires hardware support, such as counters/stack.**

Algorithm

1. Get length of the reference string, say *len*.
2. Get reference string and store it in an array, say *rs*.
3. Get number of frames, say *nf*.
4. Create *access* array to store counter that indicates a measure of recent usage.
5. Create a function *arrmin* that returns position of minimum of the given array.
6. Initalize *frame* array upto length *nf* to -1.
7. Initialize position of the page replacement, say *j* to 0.
8. Initialize *freq* to 0 to track page frequency
9. Initialize no. of page faults, say *count* to 0.
10. For each page in reference string in the given order, examine:
    a. Check whether page exist in the *frame* array.
    b. If page exist in memory then
        i. Store incremented *freq* for that page position in *access* array.
    c. If page does not exist in memory then
        i. Check for any empty frames.
        ii. If there is an empty frame,
            - Assign that frame to the page
            - Store incremented *freq* for that page position in *access* array.
            - Increment *count*.
        iii. If there is no free frame then
            - Determine page to be replaced using *arrmin* function.
            - Store incremented *freq* for that page position in *access* array.
            - Increment *count*.
        iv. Display pages in *frame* array.
11. Print *count*.
12. Stop

Program

```
/* LRU page replacement – lrupr.c */

#include<stdio.h>


int arrmin(int[], int);main()
{
int i,j,len,rs[50],frame[10],nf,k,avail,count=0;int access[10], freq=0, dm;

printf("Length of Reference string : ");scanf("%d", &len);
printf("Enter reference string :\n");for(i=1; i<=len; i++) scanf("%d", &rs[i]);
printf("Enter no.
of frames : ");scanf("%d", &nf);

for(i=0; i<nf; i++) frame[i] = -1;
j = 0;

printf("\nRef. str              Page frames"); for(i=1; i<=len; i++)
{
printf("\n%4d\t", rs[i]);avail = 0;
for(k=0; k<nf; k++)
{
if(frame[k] == rs[i])
{
avail = 1; access[k] =
++freq;break;
}
}
if(avail == 0)
{
dm = 0;
for(k=0; k<nf; k++)
{
if(frame[k] == -1)
dm = 1; break;
}
if(dm == 1)
{
frame[k] = rs[i]; access[k] =
++freq;count++;
}

else{
j = arrmin(access, nf); frame[j] = rs[i];
access[j] =++freq; count++;

}
for(k=0; k<nf; k++) printf("%4d", frame[k]);
}
}
printf("\n\nTotal no. of page faults : %d\n", count);
}
```

```
int arrmin(int a[], int n)
{
int i, min = a[0];for(i=1; i<n; i++)if (min > a[i])
min = a[i]; for(i=0; i<n; i++)
if (min == a[i]) return i;
}
```

Output

Length of Reference string : 15
Enter reference string : 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3
Enter no. of frames : 5
Ref. str Page frames1

1		**-1**	**-1**	**-1**	**-1**
2	**1**	**2**	**-1**	**-1**	**-1**
3	**1**	**2**	**3**	**-1**	**-1**
4	**1**	**2**	**3**	**4**	**-1**
2					
1					
5	**1**	**2**	**3**	**4**	**5**
6	**1**	**2**	**6**	**4**	**5**
2					
1					
2					
3	**1**	**2**	**6**	**3**	**5**
7	**1**	**2**	**6**	**3**	**7**
6					
3					

Total no. of page faults : 8

Result :

Thus page replacement was implemented using LRU algorithm.

Ex No : 11	FILE ORGANIZATION
Date :	

11a. Single-Level Directory

Aim

To organize files in a single level directory structure, I,e., without sub-directories.

Algorithm

1. Get name of directory for the user to store all the files
2. Display menu
3. Accept choice
4. If choice =1 then
   Accept filename without any collission
   Store it in the directory
5. If choice =2 then
   Accept filename
   Remove filename from the directory array
6. If choice =3 then
   Accept filename
   Check for existence of file in the directory array
7. If choice =4 then
   List all files in the directory array
8. If choice =5
   thenStop

Program

```
#include <stdio.h> #include <stdlib.h> #include <conio.h> struct{
char  dname[10]; char fname[25][10];int fcnt;
}dir;

main()
{
int i, ch; char f[30]; clrscr(); dir.fcnt = 0;
printf("\nEnter name of directory -- "); scanf("%s", dir.dname);

while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5. Exit\
nEnter your choice--");scanf("%d",&ch);
```

```
switch(ch)
{
case 1:
printf("\n Enter the name of the file -- ");
scanf("%s", dir.fname[dir.fcnt]); dir.fcnt++;
break;

case 2:
printf("\n Enter the name of the file -- ");scanf("%s", f);
for(i=0; i<dir.fcnt; i++)
{
if(strcmp(f, dir.fname[i]) == 0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i], dir.fname[dir.fcnt-1]);break;
}
}
if(I == dir.fcnt)
printf("File %s not found", f);else dir.fcnt--;break;
case 3:
printf("\n Enter the name of the file -- ");scanf("%s", f);
for(i=0; i<dir.fcnt; i++)
{
if(strcmp(f, dir.fname[i]) == 0)
{
printf("File %s is found ", f);break;
}
}
if(I == dir.fcnt)
printf("File %s not found", f);break;

case 4:
if(dir.fcnt == 0)
printf("\n Directory Empty");else
{
printf("\n The Files are -- ");for(i=0; i<dir.fcnt; i++)
printf("\t%s", dir.fname[i]);
}
break;
default:
exit(0);
}
}
getch();
}
```

Output

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 1
Enter the name of the file – fcfs

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 1
Enter the name of the file -- sjf

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 1
Enter the name of the file – lru

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 3
Enter the name of the file -- sjfFile sjf is found

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 3
Enter the name of the file -- bankFile bank is not found

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 4
The Files are -- fcfs sjf lru bank

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice -- 2
Enter the name of the file -- lruFile lru is deleted

Result

Thus files were organized into a single level directory successfully.

11b. Two-Level Directory

Aim

To organize files as two-level directory with each user having his own user file directory(UFD).

Algorithm

1. Display menu
2. Accept choice
3. If choice =1 then
   - Accept directory name
   - Create an entry for that directory
4. If choice =2 then
   - Get directory name
   - If directory exist then accept filename without collision else report error
5. If choice =3 then
   - Get directory name
   - If directory exist then Get filename
     - If file exist in that directory then delete entry else report error
6. If choice =4 then
   - Get directory name
   - If directory exist then Get filename
     - If file exist in that directory then Display filename else report error
7. If choice =5 then Display files directory-wise
8. If choice =6 then Stop

Program

```
#include <stdio.h>
#include <conio.h>
#include  <stdlib.h>
struct
{
char dname[10], fname[10][10]; int fcnt;
}dir[10];

main()
{
int i, ch, dcnt, k;char f[30], d[30]; clrscr(); dcnt=0; while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3.Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \nEnter your choice -- );

scanf("%d", &ch);

switch(ch)
{
```

```
case 1:
printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname); dir[dcnt].fcnt = 0;
dcnt++;
printf("Directory  created");
.break; case 2:
printf("\n Enter name of the directory -- "); scanf("%s", d);
for(i=0; i<dcnt; i++) if(strcmp(d,dir[i].dname)
== 0)
{
printf("Enter name of the file -- ");
scanf("%s",  dir[i].fname[dir[i].fcnt]); dir[i].fcnt++;
printf("File created"); break;
}
if(i == dcnt)
                        printf("Directory %s not found",d);break;

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

                printf("Directory %s not found",d);jmp :
        break;

            case 4:
                printf("\nEnter name of the directory -- ");
                scanf("%s", d);
                for(i=0; i<dcnt; i++)
                {
                    if(strcmp(d,dir[i].dname) == 0)
                    {
                            printf("Enter the name of the file -- ");
```

```
                        scanf("%s", f);
                        for(k=0; k<dir[i].fcnt; k++)
                        {
                                if(strcmp(f, dir[i].fname[k]) == 0)
                                {
                                        printf("File %s is found ", f);
                                        goto jmp1;
                                }
                        }
                        printf("File %s not found", f);goto
                        jmp1;
                    }
                }
                printf("Directory %s not found", d);
                jmp1: break;

        case 5:
            if(dcnt == 0)
                printf("\nNo Directory's ");else
            {
                printf("\nDirectory\tFiles");
                for(i=0;i<dcnt;i++)
                {
                    printf("\n%s\t\t",dir[i].dname);
                    for(k=0;k<dir[i].fcnt;k++)
                            printf("\t%s",dir[i].fname[k]);
                }
            }
            break;

        default:
            exit(0);
        }
    }
    getch();
}
```

Output

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 1
Enter name of directory --CSE
Directory created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 1
Enter name of directory -- ECE
Directory created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 2
Enter name of the directory –ECE
Enter name of the file -- amruth File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 2
Enter name of the directory -- CSE
Enter name of the file -- kowshik File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice – 2
Enter name of the directory -- CSE
Enter name of the file -- pranesh File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 2
Enter name of the directory –ECE
Enter name of the file -- ajith File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 5

Directory	Files
CSE	kowshik pranesh
ECE	amruth ajith

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
Enter your choice -- 3
Enter name of the directory – ECE
Enter name of the file -- ajith
File ajith is deleted

Result

Thus use r files have been stored in their respective directories and retrieved easily.

Ex No : 12	FILE ALLOCATION
Date :	

12a. Contiguous Allocation

Aim

To implement file allocation on free disk space in a contiguous manner.

Algorithm

1. Assume no. of blocks in the disk as 20 and all are free.
2. Display the status of disk blocks before allocation.
3. For each file to be allocated:
   a. Get the *filename*, *start* address and file *length*
   b. If *start* + *length* > 20, then goto step 2.
   c. Check to see whether any block in the range (start, start + length-1) is allocated. If so, then go to step 2.
   d. Allocate blocks to the file contiguously from start block to start + length – 1.
4. Display directory entries.
5. Display status of disk blocks after allocation
6. Stop

Program

```
/* Contiguous Allocation - cntalloc.c */#include <stdio.h>
#include <string.h>

int num=0, length[10],
start[10];char fid[20][4], a[20][4];
void directory()
{
    int i;
    printf("\nFile Start Length\n");


    for(i=0; i<num; i++)
        printf("%-4s  %3d  %6d\n",fid[i],start[i],length[i]);
}
void display()
{
    int i;
    for(i=0; i<20; i++)printf("%4d",i); printf("\
    n"); for(i=0;
    i<20; i++)
        printf("%4s", a[i]);
```

```
}
main()
{
    int  i,n,k,temp,st,nb,ch,flag;char id[4];

    for(i=0; i<20; i++) strcpy(a[i],  "");
    printf("Disk space before allocation:\n");
    display();
    do
    {
        printf("\nEnter File name (max 3 char) : ");
        scanf("%s", id);
        printf("Enter start block : ");
        scanf("%d", &st);
        printf("Enter no. of blocks :
        ");
        scanf("%d",  &nb); strcpy(fid[num], id);
        length[num] = nb;flag = 0;

        if((st+nb) > 20)
        {
            printf("Requirement exceeds range\n");
            continue;
        }

        for(i=st; i<(st+nb); i++)
            if(strcmp(a[i], "") != 0)
                flag = 1;
        if(flag == 1)
        {
            printf("Contiguous allocation not possible.\n");
            continue;
        }
        start[num] =  st;
        for(i=st; i<(st+nb); i+
        +)

            strcpy(a[i], id);
             printf("Allocation  done\n");num++;

        printf("\nAny more allocation (1. yes / 2. no)? : ");
        scanf("%d", &ch);
    } while (ch == 1); printf("\n\t\t\
    tContiguous Allocation\n");
    printf("Directory:");
    directory();
    printf("\nDisk space after allocation:\n");
    display();
}
```

Output

Disk space before allocation:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Enter File name (max 3 char) :
cpEnter start block : 14
Enter no. of blocks : 3
Allocation done
Any more allocation (1. yes / 2. no)? : 1

Enter File name (max 3 char) :
trEnter start block :
18 Enter no. of
blocks : 3
Requirement
exceeds range

Enter File name (max 3 char) :
trEnter start block :
10 Enter no. of
blocks : 3 Allocation
done
Any more allocation (1. yes / 2. no)? : 1

Enter File name (max 3 char) :
mvEnter start block : 0
Enter no. of blocks : 2
Allocation done
Any more allocation (1. yes / 2. no)? : 1

Enter File name (max 3 char) :
psEnter start block : 12
Enter no. of blocks : 3
Contiguous allocation not possible.
Any more allocation (1. yes / 2. no)?
: 2

Contiguous Allocation

Directory:

File	Start	Length
cp	14	3
tr	10	3
mv	0	2

Disk space after allocation:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
m	m									tr	tr	tr		cp	cp	cp			
v	v																		

Result

Thus contiguous allocation is done for files with the available free blocks.

12b.LINKED FILE ALLOCATION

Aim

To implement file allocation on free disk space in a indexed manner

Algorithm

1. Get no. of files
2. Accept filenames and no. of blocks fo each file
3. Obtrain start block for each file
4. Obtain other blocks for each file
5. Check block availability before allocation
6. If block is unavailable then report error
7. Accept file name
8. Display linked file allocation blocks for that file
9. Stop

Program

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

main()
{
    static int b[20], i, j, blocks[20]
    [20]; char F[20][20], S[20], ch;
    int sb[20], eb[20], x, n;
    clrscr();
    printf("\n Enter no. of
    Files ::"); scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter file %d name ::", i+1);
        scanf("%s", &F[i]);
        printf("\n Enter No. of blocks::", i+1);
        scanf("%d",&b[i]);
    }

    for(i=0;i<n;i++)
    {
        printf("\n Enter Starting block of file%d::",i+1);
        scanf("%d", &sb[i]);
        printf("\nEnter blocks for file%d::\n", i+1);
        for(j=0; j<b[i]-1;)
        {
          printf("\n Enter the %dblock ::",j+2);       scanf("%d",&x);

            if(b[i] != 0)
            {
```

```
                }    blocks[i][j] = x;j++;
                els
                e
                    printf("\n Invalid
            }
        }
                    block::");



        printf("\nEnter the
        Filename :"); scanf("%s",
        &S);
        for(i=0; i<n; i++)
        {
            if(strcmp(F[i],S) == 0)
            {
                printf("\nFname\tBsize\tStart\tBlocks\n");
                printf("\n_____\
                n"); printf("\n%s\t%d\t%d\t", F[i], b[i], sb[i]);
                 printf("%d->",sb[i]);
                for(j=0; j<b[i]; j++)
                {
                     if(b[i] != 0)
                          printf("%d->", blocks[i][j]);
                }
            }
        }
        printf("\n_____\
        n"); getch();
}
```

Output

```
 Enter no. of Files

 ::2 Enter file 1

 name
 ::fcfsEnter No. of
 blocks::3

 Enter file 2 name
 ::sjfEnter No. of
 blocks::2

 Enter Starting block
of          file1::8Enter
blocks for file1::
 Enter the 2block
 ::3    Enter    the
 3block ::5

 Enter Starting block
```

of file2::2Enter
blocks for file2::
Enter the 2block

::6 Enter the

Filename

::fcfs

Fname	Bsize	Start	Blocks
fcfs	3	8	8->3->5

Result

Thus blocks for file were allocation using linked allocation method.

Ex No : 13	Implement Paging Technique of memory management
Date :	

AIM:

To implement the Memory management policy- Paging.

ALGORITHM:

Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into fixed - sized blocks.
Step 3: Frames – Physical memory is broken into fixed – sized blocks.
Step 4: Calculate the physical address using the following
Physical address = (Frame number * Frame size) + offset
Step 5: Display the physical address.
Step 6: Stop the process.

PROGRAM:

```
#include
<stdio.h>
#include
<conio.h>  struct
pstruct
{
        int
        fno;
        int
        pbit;
}ptable[10];

int

pmsize,lmsize,psize,frame,page,ftable[20],frameno;

void info()
{
printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
printf("\n\nEnter the Size of Physical memory: ");
scanf("%d",&pmsize);
printf("\n\nEnter the size of Logical memory: ");
scanf("%d",&lmsize);
printf("\n\nEnter the partition size:
"); scanf("%d",&psize);
frame = (int)
pmsize/psize; page = (int)
lmsize/psize;
printf("\nThe physical memory is divided into %d no.of frames\n",frame);
printf("\nThe Logical memory is divided into %d no.of pages",page);
}
void assign()
{
```

```
int i;
for (i=0;i<page;i++)
{
ptable[i].fno = -1;
ptable[i].pbit= -1;
}
for(i=0;
i<frame;i++)
ftable[i] = 32555;
for (i=0;i<page;i++)
{
printf("\n\nEnter the Frame number where page %d must be placed: ",i);
scanf("%d",&frameno);
ftable[frameno] = i;
if(ptable[i].pbit == -1)
{
ptable[i].fno = frameno;
ptable[i].pbit = 1;
}
}
getch();
//clrscr();
printf("\n\nPAGE TABLE\n\n");
printf("PageAddress  FrameNo. PresenceBit\n\
n"); for (i=0;i<page;i++)
printf("%d\t\t%d\t\t%d\
n",i,ptable[i].fno,ptable[i].pbit); printf("\n\n\n\
tFRAME TABLE\n\n"); printf("FrameAddress
PageNo\n\n"); for(i=0;i<frame;i++)
printf("%d\t\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
int
laddr,paddr,disp,phyaddr,baddr;
getch();
//clrscr();
printf("\n\n\n\tProcess to create the Physical Address\n\n");
printf("\nEnter the Base Address: ");
scanf("%d",&baddr);
printf("\nEnter theLogical Address:
"); scanf("%d",&laddr);
paddr = laddr /
psize; disp = laddr
% psize;
if(ptable[paddr].pbit == 1 )
phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
clrscr();
info();
assign();
```

```
cphyaddr();
getch();
}
```

OUTPUT:

```
   MEMORY MANAGEMENT USING PAGING
   Enter the Size of Physical memory: 16
   Enter the size of Logical memory: 8 Enter the partition size: 2
   The physical memory is divided into 8 no.of frames
   The Logical memory is divided into 4 no.of pages
   Enter the Frame number where page 0 must be placed: 5
   Enter the Frame number where page 1 must be placed: 6
   Enter the Frame number where page 2 must be placed: 7
   Enter the Frame number where page 3 must be placed: 2
PAGE TABLE

PageAddress          FrameNo.              PresenceBit

      0                  5                     1
      1                  6                     1
      2                  7                     1
      3                  2                     1

FRAME TABLE
 FrameAddress           PageNo

      0                 32555
      1                 32555
      2                   3
      3                 32555
      4                 32555
      5                   0
      6                   1
      7                   2
Process to create the Physical
  Address Enter the Base Address:
  1000
 Enter theLogical Address: 3

The Physical Address where the instruction present: 1013
```

RESULT:

Thus the program has been executed successfully and the output is verified.

Ex No : 14	Disk scheduling algorithm
Date :	

14 a. FCFS Disk Scheduling Algorithm

Program:

```
#include<stdio.h>
main()
{
int t[20], n, I, j, tohm[20], tot=0; float
avhm; clrscr();
printf(“enter the no.of tracks”);
scanf(“%d”,&n);
printf(“enter the tracks to be traversed”);
for(i=2;i<n+2;i++)
scanf(“%d”,&t*i+);
for(i=1;i<n+1;i++)
{
tohm[i]=t[i+1]-t[i];
if(tohm[i]<0)
tohm[i]=tohm[i]*(-
1);
}
for(i=1;i<n+1;i++)
tot+=tohm[i];
avhm=(float)tot/n;
printf(“Tracks traversed\tDifference between tracks\n”);
for(i=1;i<n+1;i++)
printf(“%d\t\t\t%d\n”,t*i+,tohm*i+); printf("\
nAverage header movements:%f",avhm);
getch();
}
```

Output

INPUT
Enter no.of tracks:9
Enter track position:55 58 60 70 18 90 150 160 184
Output
Tracks traversed Difference between
tracks 55 45
58 3
60 2
70 10
18 52
90 72
150 60
160 10
184 24
Average header movements: 30.888889

14b. SCAN DISK SCHEDULING ALGORITHM

Program

```
#include<stdio.h>
main()
{
int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
clrscr();
printf("enter the no of tracks to be traveresed");
scanf("%d'",&n);
printf("enter the position of head");
scanf("%d",&h);
t[0]=0;t[1]=h;
printf("enter the tracks");
for(i=2;i<n+2;i++)
scanf("%d",&t[i]);
for(i=0;i<n+2;i++)
{
for(j=0;j<(n+2)-i-1;j++)
{
if(t[j]>t[j+1])
{
temp=t[j];
t[j]=t[j+1];
t[j+1]=temp;
}
}
}
for(i=0;i<n+2;i++)
if(t[i]==h)
j=i;k
=i;
p=0;
while(t[j]!=0)
{
atr[p]=t[
j]; j--;
p++;
}
atr[p]=t[j];
for(p=k+1;p<n+2;p++,k+
+) atr[p]=t[k+1];
for(j=0;j<n+1;j++)
{
if(atr[j]>atr[j+1])
d[j]=atr[j]-atr[j+1];
else
d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("\nAverage header movements:%f",(float)sum/n);
getch();
}
```

Output:

```
INPUT
Enter no.of tracks:9
Enter track position:55 58 60 70 18 90 150 160
18 Tracks traversed Difference between tracks
150 50
160 10
184 24
90 94
70 20
60 10
58 20
55 3
18 37
Average header movements: 27.77
```

14c. C-SCAN Disk Scheduling

Algorithm Program

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

```
}
for(j=0;j<n+2;j++)
{
if(atr[j]>atr[j+1])
d[j]=atr[j]-atr[j+1];
else
d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("total header movements%d",sum);
printf("avg is %f",(float)sum/n);
getch();
}
```

Output

INPUT
Enter the track position : 55 58 60 70 18 90 150 160 184
Enter starting position : 100
Output
Tracks traversed Difference between
tracks 150 50
160 10
184 24
18 240
55 37
58 3
60 2
70 10
90 20
Average Seek time: 35.77777

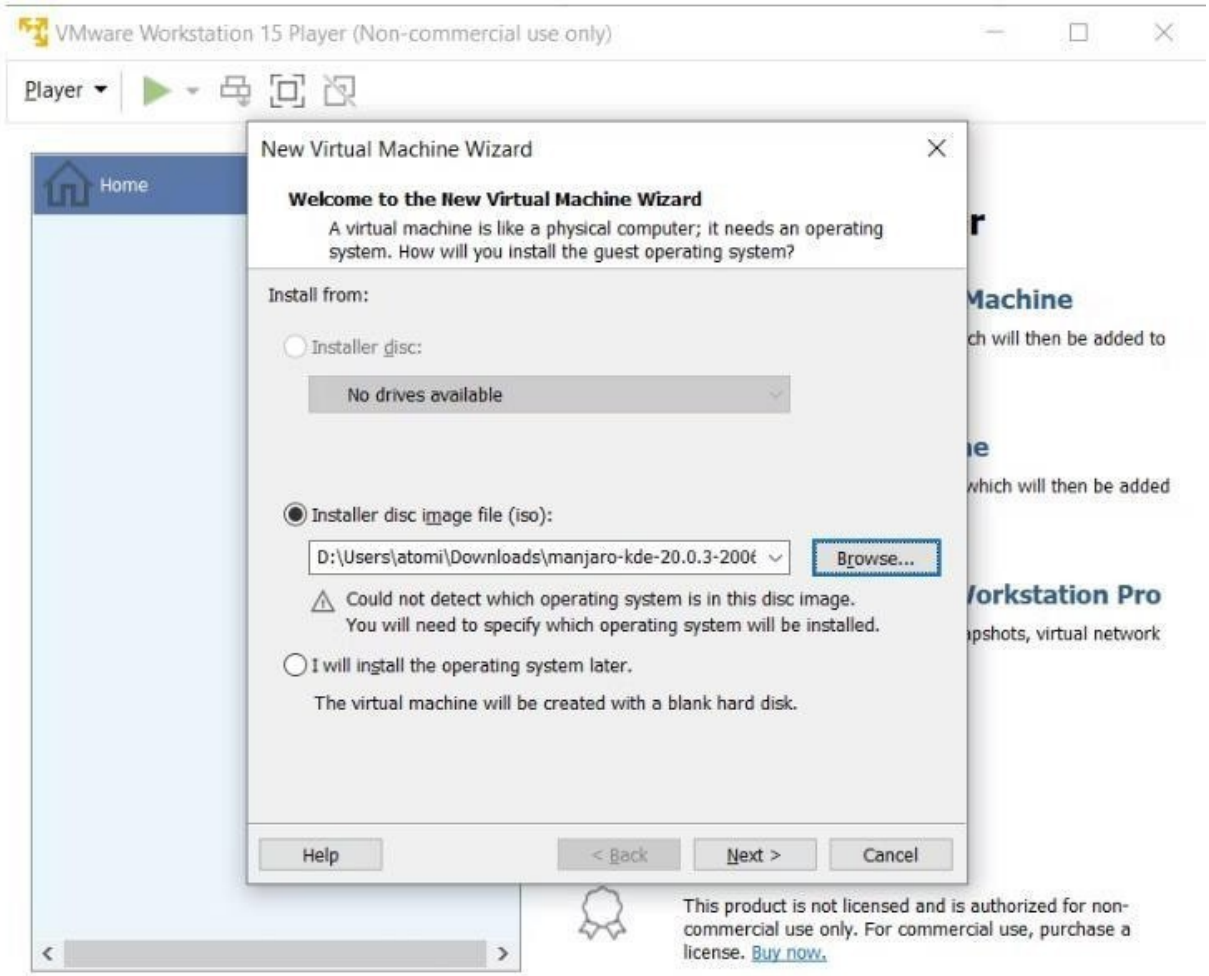
Ex No : 15	Installation of Guest operating system Linux
Date :	

Aim:

To install any guest operating system like Linux using VMware.

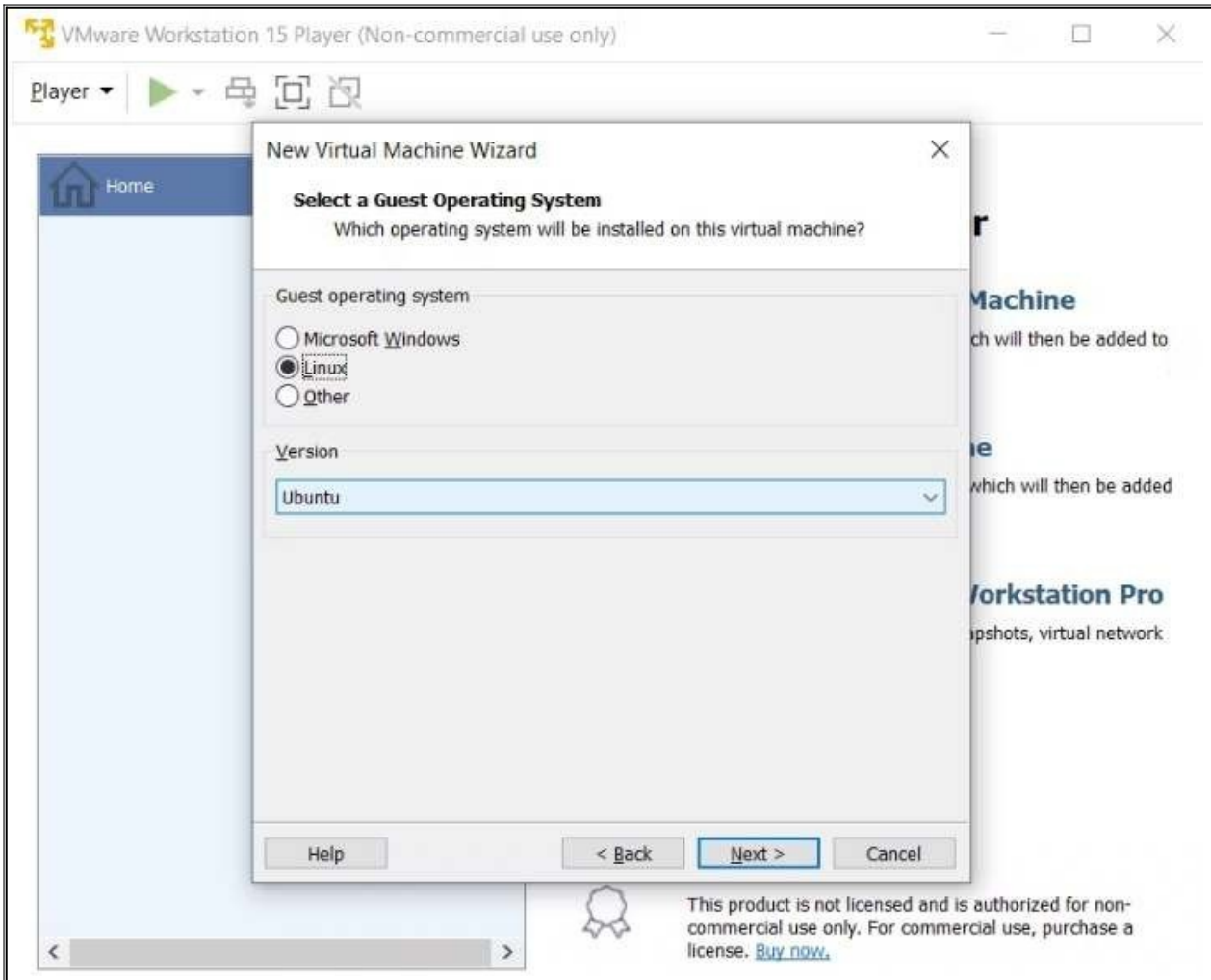
Procedure:

1. Install VMware Workstation Player
2. To start, head to the VMware website and download the latest version of their Workstation Player tool. We're using VMware Workstation 15 Player, which is around 150MB to download.
3. Create Your Linux Virtual Machine
4. Click Create a New Virtual Machine
5. Select the default option, Installer disc image file (iso)
6. Click Browse to find the ISO file

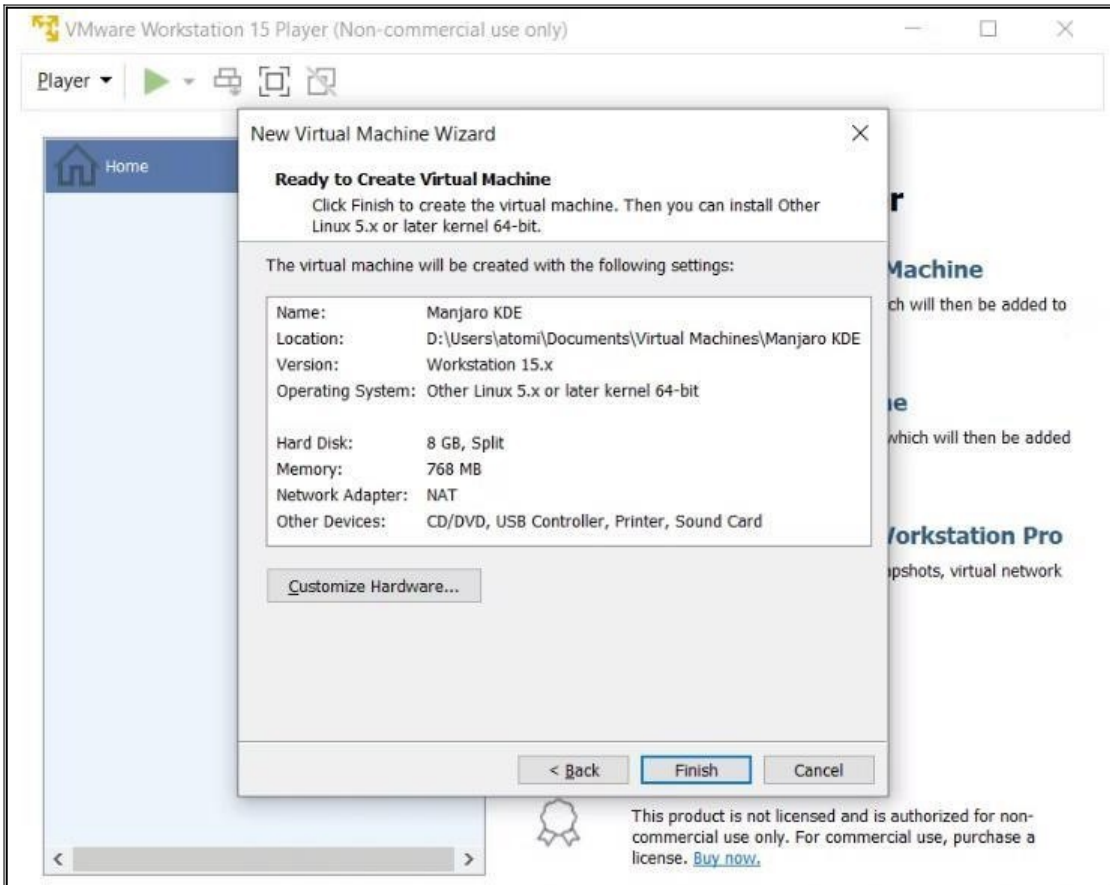


a. With "guest" OS selected, click Next

b. Select Linux as the Guest operating system type

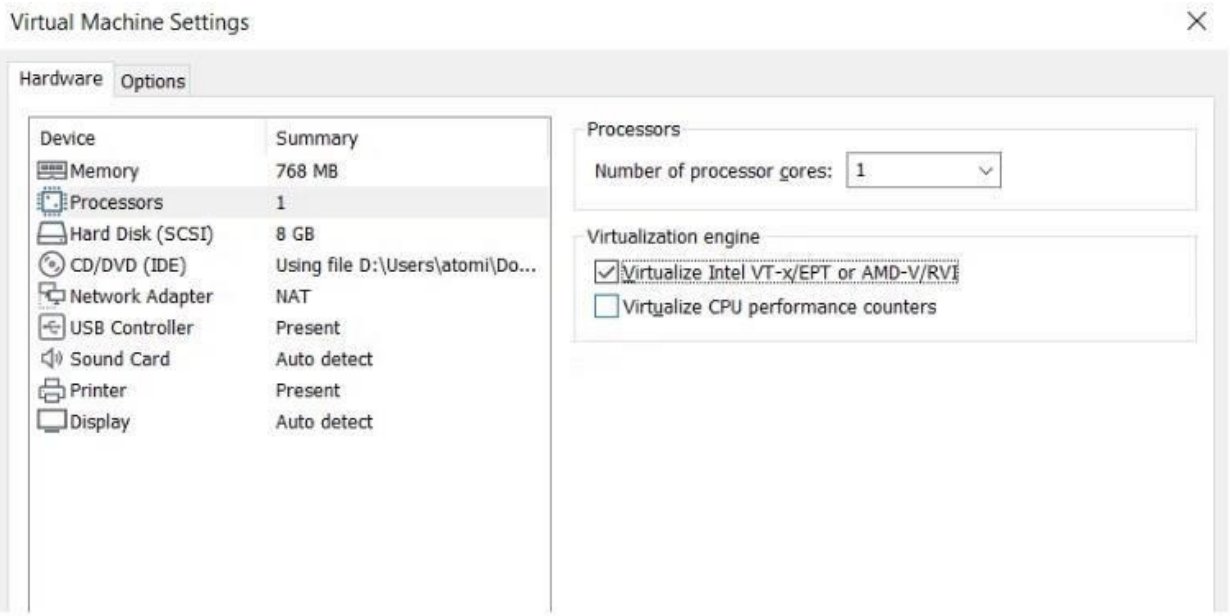


c. Under Version, scroll through the list and select the OS
d. Click Next to proceed and if necessary, input a Virtual machine name
e. Confirm the storage Location and change if needed
2. With the operating system selected and configured, it's time to build the virtual machine.
a. Under Specify Disk Capacity adjust Maximum disk size if required (the default should be enough)
b. Select Split virtual disk into multiple files as this makes moving the VM to a new PC easy
c. Click Next then confirm the details on the next screen
d. If anything seems wrong click Back, otherwise click Finish

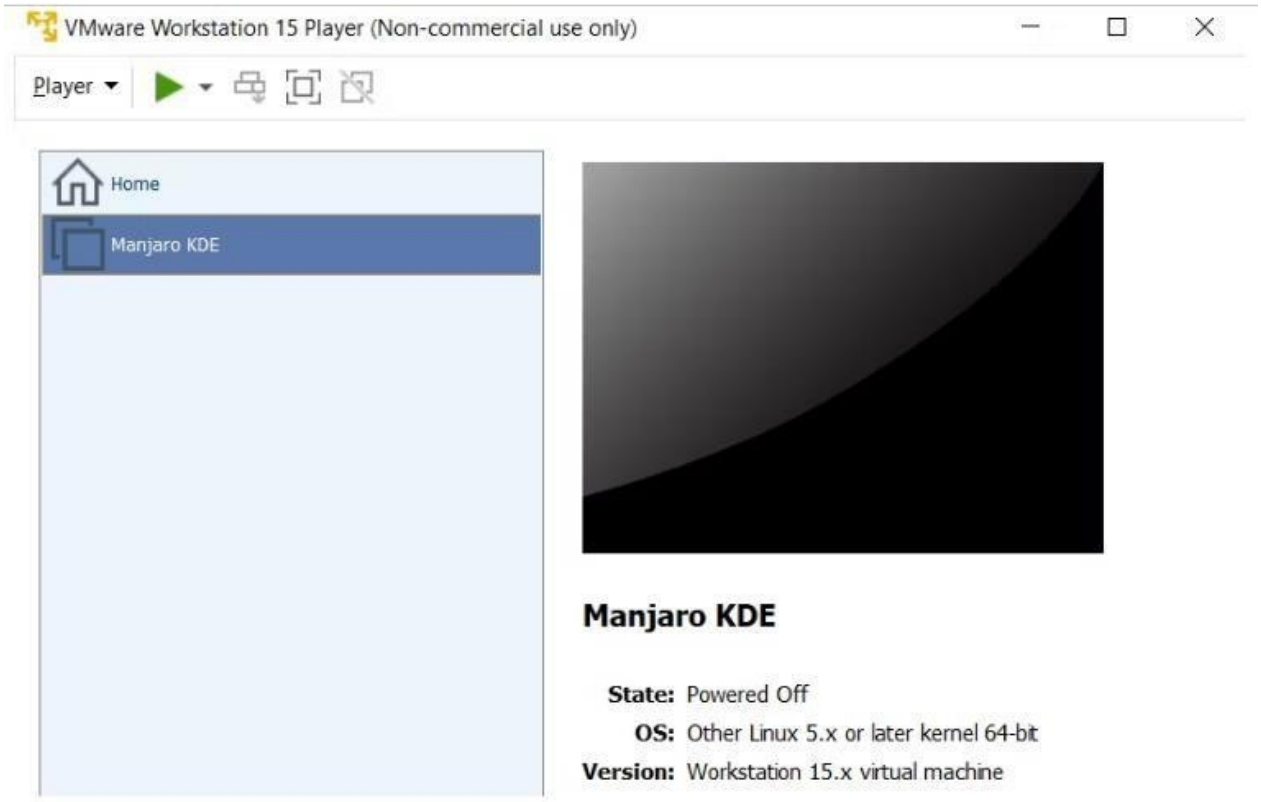


3. Customize Your Virtual Hardware

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



b. Click Processors. In the right-hand pane, click the check box Intel VT-x or
AMD-V, depending on your CPU.
c. Finally, check the Display settings. Click OK to confirm changes, then select the virtual
machine and click the Play button to begin.



ling Linux in VMware is simple. Let's run through the steps again:
d. Download the free VMware Workstation Player
e. Install, and restart Windows
f. Create and configure your virtual machine
g. Install Linux in the virtual machine**Result:**
h. Restart the virtual machine and use Linux

Result:

Thus the guest operating system Linux using VMWare was installed successfully.