

**EX.NO:1**

**DATE:**

## **INSTALLATION OF WINDOWS OPERATING SYSTEM**

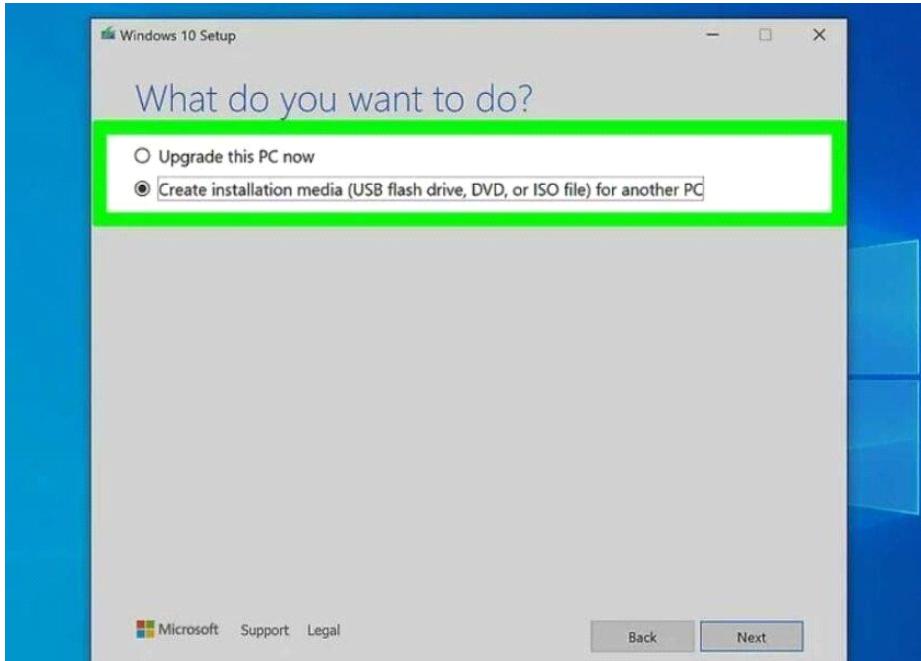
### **AIM:**

To install windows operating system.

### **Operating System**

In simple words, Operating System is system software that is required to run applications programs and utilities.

### **Create a window installation media**



### **Requirement:**

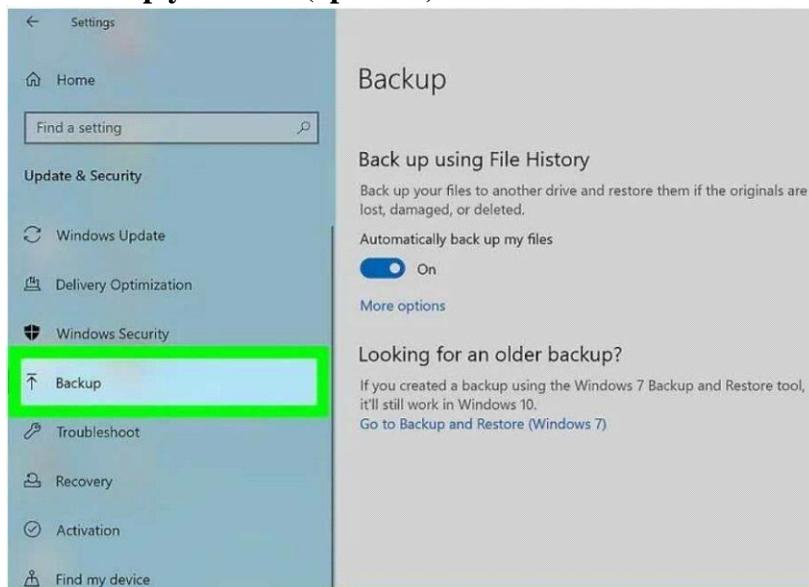
- Minimum 8 GB of Pen Drive.
- A computer with an internet connection.

download windows from the official website of Microsoft. After successfully downloaded the .exe file, follow the steps below:

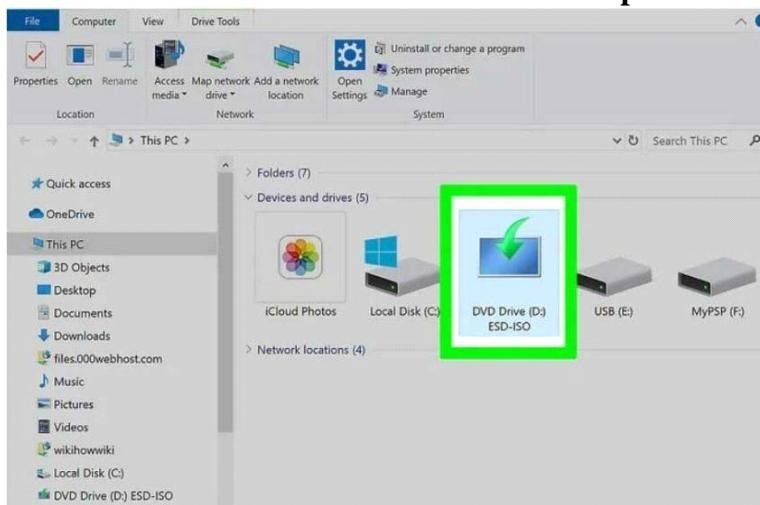
## Steps to create installation media

- Plugin a flash drive
- Download a tool [MediaCreationTool.exe](#)
- Run it, and click on accept
- Select installation media ( DVD/ISO, USB Drive) from another PC and click next
- Select your language, windows edition and PC architecture and click NEXT
- Now, you need to select the USB Pen drive and click NEXT

### Backup your files (optional)

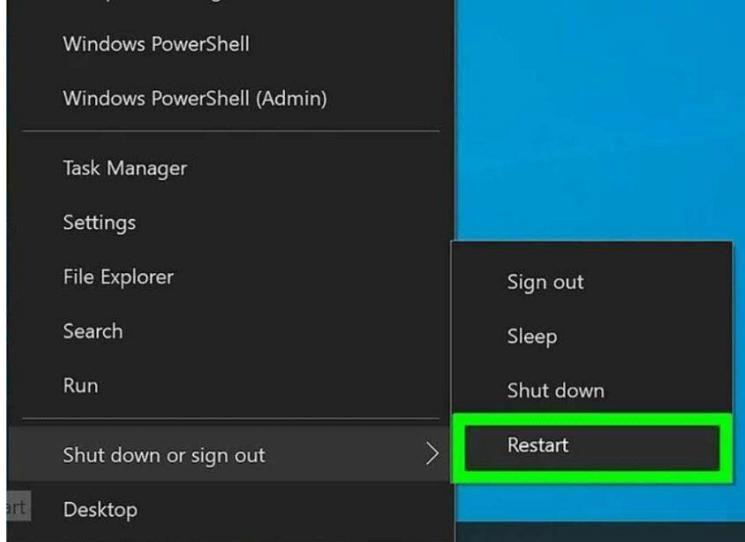


### Insert Window installation media to Computer



Now, It's time to insert a media installation (USB or DVD) into the computer. Make sure all the windows fileshave been copied to the flash drive.

## Boot the computer



click on the windows icon and choose the option “**Shut down or sign out**” and then “**Restart**“. as pc reboot,press **F10, F11, F12, or ESC** in order to enter the boot menu.Press **F1, F2, F3, or ESC** to enter the BIOS (Basic Input Output System).

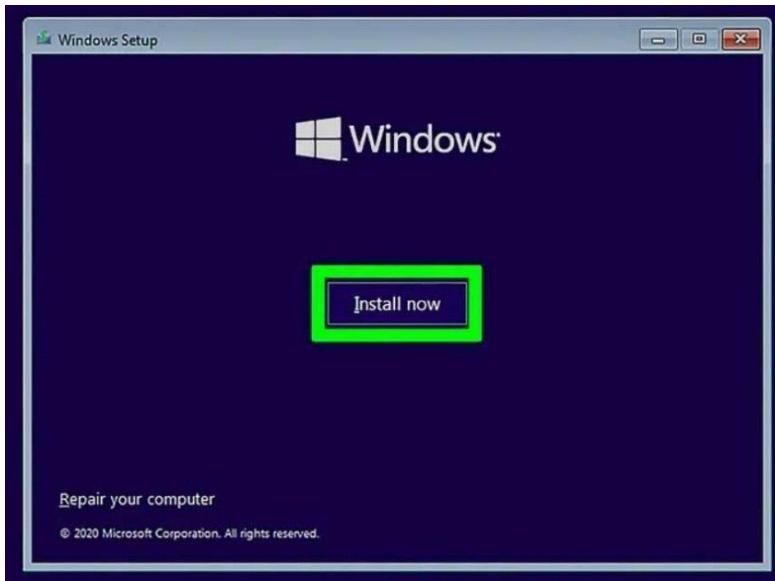
## Select the USB Drive



Select BIOS Features option

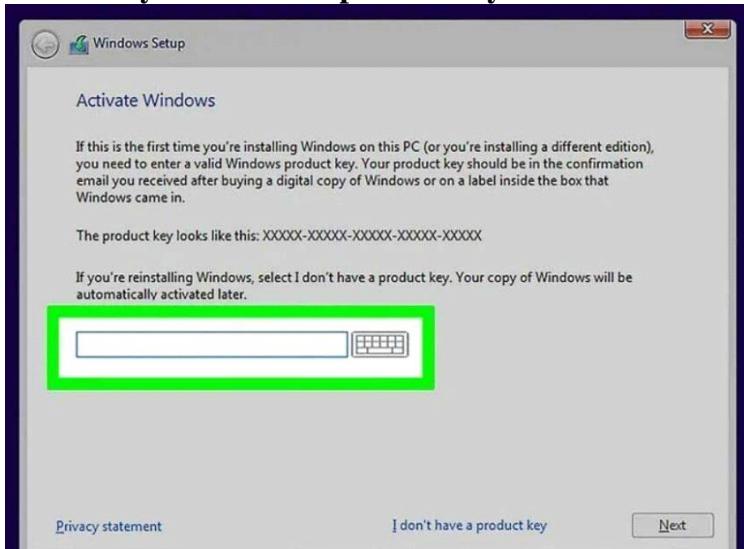
In Boot Options Priorities, Select the Boot Option #1. Click it and select the flash drive option.

## Install Now

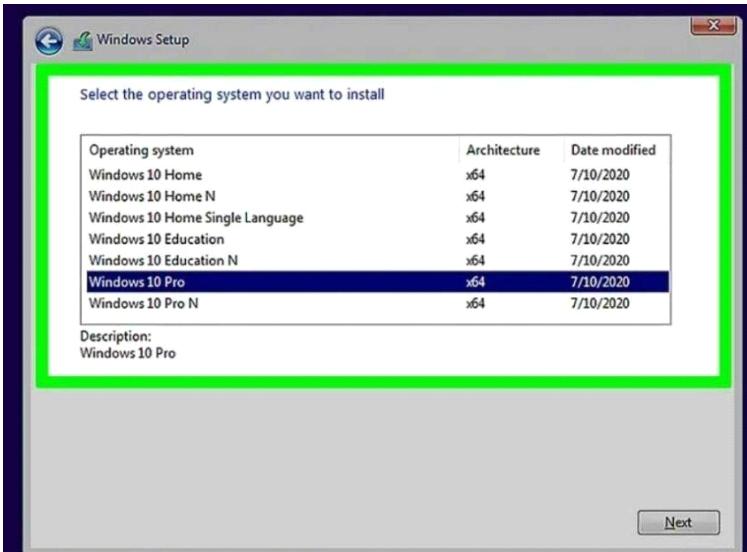


Click on “Install now” and continue to the next step.

### Enter your windows product key and click next



### Select the edition of window to install

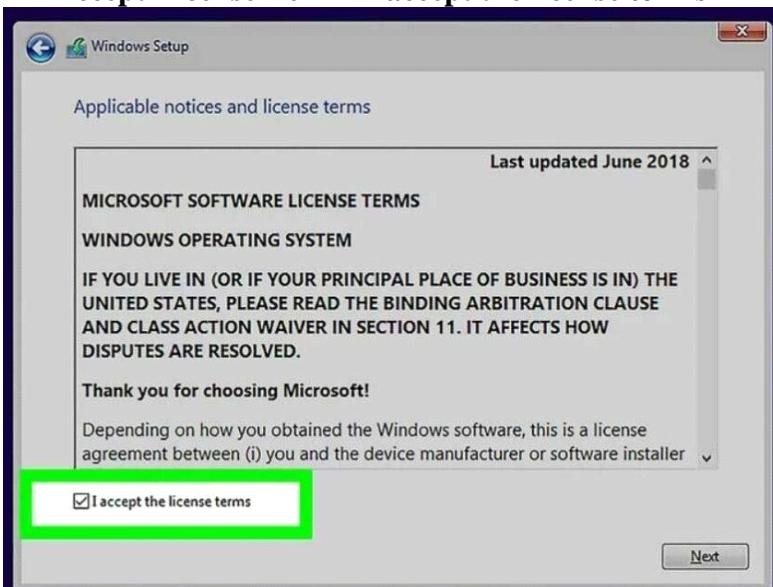


Windows operating system comes in two-bit options 32-bit and 64-bit. x86 denotes 32-bit and x64 denotes 64-bit.

### Difference between 32-bit and 64-bit

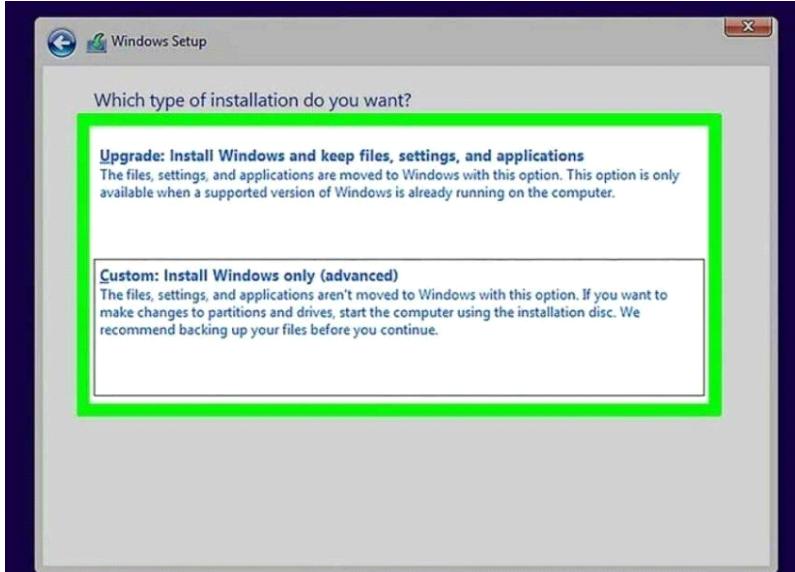
- “The 32 bit OS can store and handle lesser data than the 64 bit OS. it addresses a maximum of 4,294,967,296 bytes (4 GB) of RAM. The 64 bit OS, on the other hand, can handle more data than the 32 bitOS.”
- x86 or 32-bit operating system supports only 32-bit software programme
- x64 or 64-bit operating system supports both 32-bit and 64-bit software programme. go for “**Windows 10 Home**” mostly used OS for personal computers. now click “NEXT”.

### Accept License Term “I accept the license terms”



Before proceeding further, accept the license terms and then click next to continue the installation.

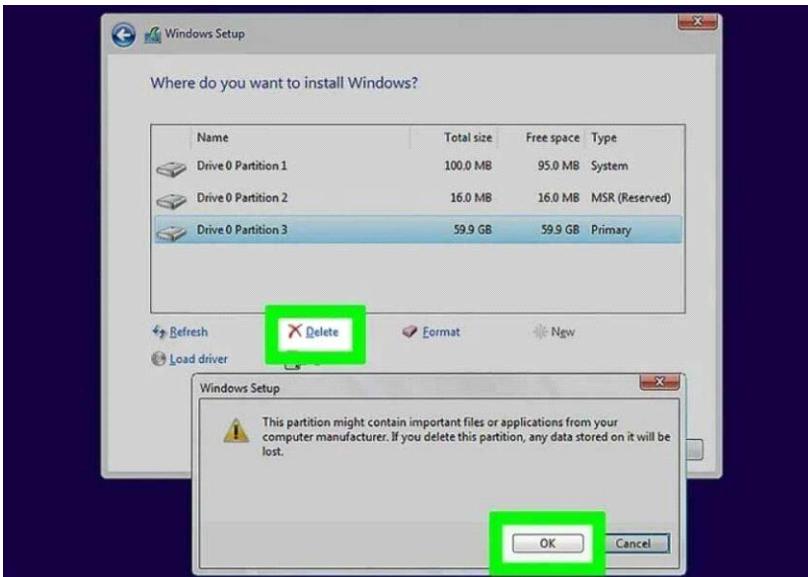
### Choose: Custom install window only or Upgrade:



**Upgrade:** This option is useful when installing the latest OS to your existing supporting versions of the operating system.

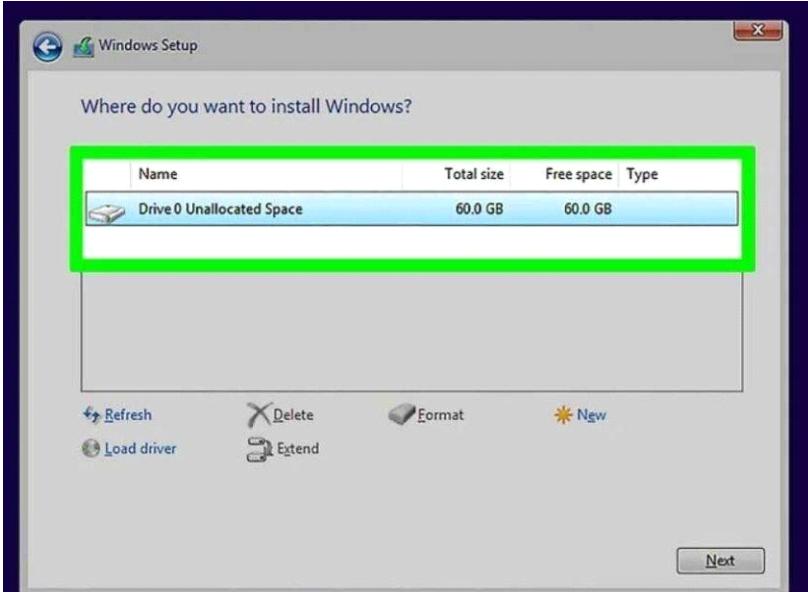
**Custom Installation:** This option is useful while installing OS to brand new or existing computer, which doesn't have an OS.

### Select a drive or partition



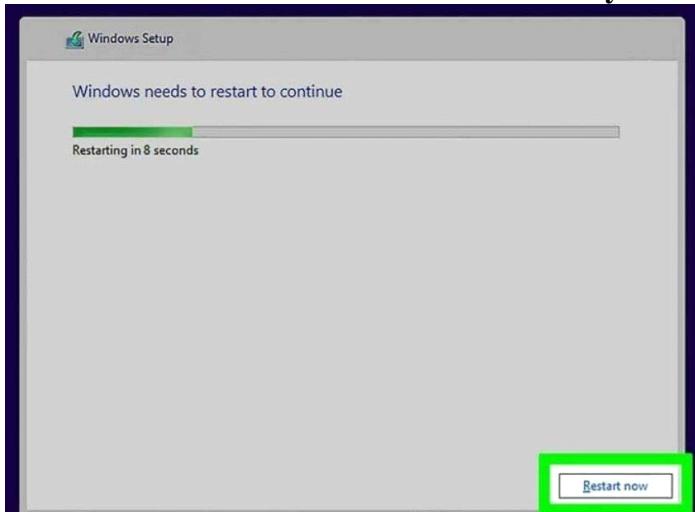
Choose the correct partition where you are going to install your operating system and then click on “Delete”.

### Select a drive with unallocated space and click “Next”



Choose the drive with unallocated space and click “Next” to continue the installation of windows. This might take little time to copy data to hard drive, as it's done move on to the next step.

**Remove the USB flash drive and restart your computer.**



As you can see, you have successfully installed the operating system to the computer system. Now, remove the USB flash drive and restart your computer.

## Setting Up Windows



For setting up windows, only a few steps as mentioned below:

- Verify your region and keyboard input
- Connect to your Wireless network.
- Select for Personal Use or Organizational use and click next

## Sign in to Windows

- Click “Accept” to Set up Cortana
- Click “Yes” and follow instructions to set up the Windows timeline
- Choose your privacy settings and click “Accept”

This will take a few more times to set up your windows. Now, wait for the setting up. As it's done.

### **RESULT:**

Thus the installation of windows operating system was executed successfully.

<b>EX.NO:2(a)</b>	<b>BASICS OF UNIX COMMANDS</b>
<b>DATE:</b>	

### **AIM:**

To study and execute Unix commands.

### **PROCEDURE:**

Unix is security conscious, and can be used only by those persons who have an account.

Telnet (Telephone Network) is a Terminal emulator program for TCP/IP networks that enables users to log on to remote servers.

To logon, type telnet server\_ip address in run window.

User has to authenticate himself by providing username and password. Once verified, a greeting and \$ prompt appears. The shell is now ready to receive commands from the user. Options suffixed with a hyphen (-) and arguments are separated by space.

### **GENERAL COMMANDS**

<b>Command</b>	<b>Function</b>
Date	Used to display the current system date and time.
date +%D date +%T date +%Y date +%H Cal	Displays date only Displays time only Displays the year part of date Displays the hour part of time Calendar of the current month
Calyear calmonth year Who	Displays calendar for all months of the specified year Displays calendar for the specified month of the year Login details of all users such as their IP, Terminal No, Username,
who am i Tty Uname uname -r	Used to display the login details of the user Used to display the terminal name Displays the Operating System Shows version number of the OS (kernel).
uname -n echo "txt"	Displays domain name of the server Displays the given text on the screen
echo \$HOME Bc	Displays the user's home directory Basic calculator. Press <b>Ctrl+d</b> to quit
Lpfile	Allows the user to spool a job along with others in a printqueue.
man cmdname	Manual for the given command. Press <b>q</b> to exit
History Exit	To display the commands used by the user since log on. Exit from a process. If shell is the only process then logs out

## DIRECTORY COMMANDS

<b>Command</b>	<b>Function</b>
Pwd	Path of the present working directory
Mkdirdir	A directory is created in the given name under the currentdirectory
mkdirdir1 dir2	A number of sub-directories can be created under one stroke
cd subdir	Change Directory. If the subdirstarts with / then path startsfrom <b>root</b> (absolute) otherwise from current working directory.
Cd	To switch to the home directory.
cd /	To switch to the root directory.
cd..	To move back to the parent directory
rmdirsubdir	Removes an empty sub-directory.

## FILE COMMANDS

<b>Command</b>	<b>Function</b>
cat >filename	To create a file with some contents. To end typing press <b>Ctrl+d</b> . The >symbol means redirecting output to a file. (<for input)
cat filename cat >>filename	Displays the file contents. Used to append contents to a file
cpsrc des  cp -i src des cp -r src des mv old new  mv f1 f2 f3 dir	Copy files to given location. If already exists, it will be overwritten  Warns the user prior to overwriting the destination file Copies the entire directory, all its sub-directories and files. To rename an existing file or directory. -i option can also be used  To move a group of files to a directory.
mv -v old new Rmfile	Display name of each file as it is moved. Used to delete a file or group of files. -i option can also be used
rm *	To delete all the files in the directory.
rm -r *	Deletes all files and sub-directories
rm -f * Ls	To forcibly remove even write-protected files Lists all files and subdirectories (blue colored) in sorted manner.
Lsname lsname* ls -a ls -x dirname ls -R	To check whether a file or directory exists. Short-hand notation to list out filenames of a specific pattern. Lists all files including hidden files (files beginning with .) To have specific listing of a directory. Recursive listing of all files in the subdirectories
ls -l	Long listing showing file access rights (read/write/execute-rwx for user/group/others-ugo).
cmpfile1 file2	Used to compare two files. Displays nothing if files are identical.

## OUTPUT

### GENERAL COMMANDS

[student@veccse ~]date

Sat May 06 06:10:34 UTC 2023

[student@veccse ~]date +%D

05/06/23

[student@veccse ~]date +%T

10:13:11

[student@veccse ~]date +%Y

2023

[student@veccse ~]date +%H

10

[student@veccse ~]who

studentpts/1 May 06 10:05 (172.16.1.14)

[student@veccse ~]who am i

studentpts/1 May 16 10:05 (172.16.1.14)

[student@veccse ~]tty

/dev/pts/1

[student@veccse ~]uname

Linux

[student@veccse ~]echo "hello"

hello

[student@veccse ~]echo \$HOME

/home/s

tudent

[studen

t@vecc

se

~]bcbc

1.06

Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc. This is

freesoftwarewith ABSOLUTELY NO WARRANTY.

For details type `warranty'.

[student@veccse ~]man lp

lp(1) Easy

Software Products lp(1)

NAME

lp - print files cancel - cancel jobs SYNOPSIS

```
lp [ -E ] [ -c ] [ -d destination ] [ -h server ] [ -m ] [ -n num- copies [ -o option ] [ -q priority ] [
```

```
-s ] [ -t title ] [ -H handling
```

```
] [ -P page-list ] [ file(s) ]
```

```
lp [ -E ] [ -c ] [ -h server ] [ -i job-id ] [ -n num-copies [ -o option ] [ -q priority ] [ -ttitle ] [ -H handling ] [ -P page-list ] cancel [ -a ] [ -h server ] [ -u username ] [ id ] [ destination ] [ destination-id ]
```

## DESCRIPTION

lpsubmits files for printing or alters a pending job. Use a filename of "-" to forceprintingfrom the standard input.

cancel cancels existing print jobs. The -a option will remove all jobs from thespecifieddestination.

## OPTIONS

The following options are recognized by lp:

```
[student@veccse ~]history
```

- date
  - date +%D
  - date +%T
  - date +%Y
  - date +%H
  - who
  - who am i
  - tty
  - uname
  - uname -r
  - uname -n
  - echo "helloi"
  - echo \$HOME
  - bc
  - man lp
- history

## DIRECTORY

## COMMANDS

```
[student@veccse]
```

```
$ pwd  
/home/student
```

```
[student@veccse]
```

```
~]mkdir san
```

```
[student@veccse]
```

```
~]mkdir s1 s2
```

```
[student@veccse ~]ls
```

```
s1 s2 san
```

```
[student@veccs]
```

```
e ~]cd s1
```

```
[student@veccs]
```

```
e s1]$ cd /
```

```
[student@veccs]
```

```
e /]$ cd ..
```

```
[student@veccs]
```

```
e /]$ rmdir s1
```

```
[student@veccs]
```

```
e ~]$ ls
```

```
s2 san
```

## FILE COMMANDS

```
[student@vecit ~]$ cat>test
```

```
hi welcome operating systems lab
```

```
[student@vecit ~]$ cat test
```

```
hi welcome operating systems lab
```

```
[student@vecit ~]$ cat>>test fourth  
semester[student@vecit ~]$ cat testhi  
welcome operating systems lab fourth  
semester [student@vecit ~]$ cat>test1
```

```

[student@vecit
~]$ cp test test1
[student@vecit
~]$ cat test1
hi welcome operating systems lab fourth semester
[student@vecit ~]$ cp -i test test1 cp: overwrite `test1'?
y[student@vecit ~]$ cp -r test test1
[student@vecit ~]$ ls

s s2 san swap.sh temp.sh test TEST test1
[student@vecit ~]$ mv san san1
[student@vecit ~]$ ls
s s2 san1 swap.sh temp.sh test TEST test1
[student@vecit ~]$ mv test test1 san1 [student@vecit
~]$ mv -v san1 sannew
`san1' -> `sannew'

[student@vecit ~]$ ls
s s2 sannew swap.sh temp.sh TEST
[student@vecit ~]$ cmp test test1cmp:
test: No such file or directory

```

## **RESULT**

Thus the study and execution of Unix commands has been completed successfully.

<b>EX. NO: 1(b)</b>	<b>SIMPLE SHELL PROGRAMS</b>
<b>DATE:</b>	

### **AIM:**

To write simple shell scripts using shell programming fundamentals.

### **DESCRIPTION:**

The activities of a shell are not restricted to command interpretation alone. The shell also has Rudimentary programming features. When a group of commands has to be executed regularly, they are stored in a file (with extension .sh). All such

files are called shell scripts or shell programs. Shell programs run in interpretive mode.

## Preliminaries

- Comments in shell script start with #. It can be placed anywhere in a line; the shell ignores contents to its right. Comments are recommended but not mandatory
- Shell variables are loosely typed i.e. not declared. Their type depends on the value assigned. Variables when used in an expression or output must be prefixed by \$.
- The read statement is shell's internal tool for making scripts interactive.
- Output is displayed using echo statement. Any text should be within quotes. Escape sequence should be used with -e option.
- Commands are always enclosed with `` (back quotes).
- Expressions are computed using the expr command. Arithmetic operators are + - \* / %. Meta characters \* ( ) should be escaped with a \.
- Multiple statements can be written in a single line separated by ;
- The shell scripts are executed using the sh command (sh filename).

## Swapping values of

### two variables

#### **Algorithm**

Step 1 : Start

Step 2 : Read the values of a and b

Step 3 : Interchange the values of a and b using another variable t as follows:  
t = a, a = b,

b = t

Step

4 :

Prin

t a

and

b

Step

5 :  
Stop  
**Program (swap.sh) #**  
**Swapping values** echo -n  
"Enter value for A : "  
read a echo -n "Enter  
value for B : " read bt=\$a

a=\$b b=\$t  
echo "Values after Swapping" echo "A Value  
is \$a"echo"B Value is \$b"  
**Output**

[student@vecit ~]\$ sh swap.sh Enter Value for  
A:5 Enter Value for B:6 Values after  
Swapping A value is 6B values is5  
[student@vecit ~]\$

### **Farenheit to Centigrade**

#### **ConversionAlgorithm**

Step 1: Start

Step 2: Read Fahrenheit value

Step 3: Convert Fahrenheit to centigrade using the formulae:

(Fahre  
nheit –  
32) ×  
5/9

Step 4:

Print

centigr

ade

Step 5:

Stop

#### **Program**

# Degree conversion

```

echo -n "Enter
Fahrenheit : " read f
c=`expr\($f - 32 \) \* 5
/ 9`
echo "Centigrade is : $c"

```

### **Output**

```
[student@vecit ~]$ sh temp.sh Enter Fahrenheit:4 Centigrade is: -15[student@vecit ~]$
```

## **RESULT**

Thus using programming basics, simple shell scripts were executed

## **CONDITIONAL CONSTRUCTS**

### **AIM:**

To write shell scripts using decision-making constructs.

### **DESCRIPTION:**

Shell supports decision-making using if statement. The if statement like its counterpart in programming languages has the following formats. The first construct executes the statements when the condition is true. The second construct adds an optional else to the first one that has different set of statements to be executed depending on whether the condition is true or false. The last one is an elif ladder, in which conditions are tested in sequence, but only one set of statements is executed.

<i>if [ condition Jthen</i>	<i>if [ condition Jthen</i>	<i>if [condition Jthen</i>
<i>statements fi</i>	<i>statements else</i>	<i>statements elif [ condition ]</i>

	<i>statements</i>	<i>then</i>
	<i>fi</i>	<i>statements ...</i>
		<i>else</i>
		<i>statements</i>
		<i>fi</i>

### Operator Description

-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal to
-a	Logical AND
-o	Logical OR
!	Logical

**Odd or  
even  
Algorit  
hm**

Step1:Start  
 Step 2 : Read number  
 Step 3 : If number divisible by 2 then Print "Number is Even"  
 Step 3.1 : else Print  
 "Number is Odd"Step 4  
 : Stop Program

### # Odd or even using if-else

```
echo -n "Enter a non-zero number : " readnumrem=`expr
$num % 2`  

if [ $rem -  

eq 0 ]then
```

```
echo "$num is
Even" elseecho
"$num is Odd"
fi
```

### **Output**

```
[student@vecit ~]$ sh oddeven.sh
Enter a non-zero number : 12 12 is Even
```

**Stri**

**ng**

**co**

**mp**

**aris**

**on**

**Alg**

**orit**

**hm**

Step 1 : Start

Step 2 : Read strings str1 and str2

Step 3 : If str1 = str2 then Print "Strings are the same"

Step 3.1 : else Print "Strings

are distinct"Step 4 : Stop

### **Program**

```
echo -n "Enter the first string : "
read s1 echo -n"Enter the second
string : " read s2if [ $s1 ==
$s2 ] then
```

```
echo "Strings are the
same" elseecho"Strings
are distinct" fi
```

### **Output**

```
[student@vecit ~]$ sh strcomp.sh
```

```
Enter the first string :ece-a Enter the second string : ECE-A Strings are distinct
```

## **RESULT**

Thus using if statement scripts with conditional expressions were executed

## **MULTI-WAY BRANCHING**

### **AIM:**

To write shell scripts using case construct to match patterns.

### **DESCRIPTION:**

The case statement is used to compare a variables value against a set of constants (integer, character, string, range). If it matches a constant, then the set of statements followed after ) is executed till a ;; is encountered. The optional default block is indicated by \*. Multiple constants can be specified in a single pattern separated by |.

*c*

*a*

*s*

*e*

*v*

*a*

*r*

*i*

*a*

*b*

*l*

*e*

*i*

*n*

*c*

*o*

*n*

*s*

*t*

*a*

*n*

*t*

*I*

)

*s*

*t*

*a*

*t*

*e*

*m*

*e*

*n*

*t*

*s*

;

;

*c*

*o*

*n*

*s*

*t*

*a*

*n*

*t*  
2  
)

*s*  
*t*  
*a*  
*t*  
*e*  
*m*  
*e*  
*n*  
*t*  
*s*

;  
;  
...

*constantN statements ;;*

\*

)

*s*  
*t*  
*a*  
*t*  
*e*  
*m*  
*e*  
*n*  
*t*  
*s*

*e*

**s**

**a**

**c**

**Si**

**mp**

**le**

**Ca**

**lcu**

**lat**

**or**

**Al**

**gor**

**ith**

**m**

Step 1 : Start

Step 2 : Read operands a and b

Step 3 :

Display

operation menu

Step 4 : Read

option

Step 5 : If option = 1 then Calculate  $c = a + b$

Step 5.1 : else if option = 2 then

Calculate  $c = a - b$  Step 5.2 : else if

option = 3 then Calculate  $c = a * b$

Step 5.3 : else if option = 4 then

Calculate  $c = a / b$  Step 5.4 : else if

option = 5 then Calculate  $c = a \% b$

Step 5.5 : else Print  
"Invalid option"Step 6 :  
Print c  
Step 7 : Stop

### **Program**

```
# Arithmetic operations--multiple statements in a block echo -n "Enter the  
two numbers : "read a b  
echo " 1. Addition" echo " 2.  
Subtraction" echo" 3.  
Multiplication" echo " 4.  
Division"  
echo " 5. Modulo Division" echo -n "Enter the option : " read optioncase  
$option in  
1)      c=`expr $a + $b` echo "$a + $b = $c";;  
2)      c=`expr $a - $b` echo "$a - $b = $c";;  
3)      c=`expr $a \* $b` echo "$a * $b = $c";;  
4)      c=`expr $a / $b` echo "$a / $b = $c";;  
5)      c=`expr $a % $b` echo "$a % $b = $c";;  
*) echo "Invalid Option" esac
```

### **Output**

```
[student@vecit ~]$  
shsimplecal.shEnter the  
two numbers : 2 4  
•      Addition  
•      Subtraction  
•      Multiplication  
•      Division  
•      Modulo Division Enter the option : 1 2 + 4 = 6
```

## **RESULT**

Thus using case statement, shell scripts were executed.

## **LOOPING**

### **AIM**

To write shell scripts using looping statements.

### **DESCRIPTION:**

Shell supports a set of loops such as for, while and until to execute a set of statements repeatedly. The body of the loop is contained between do and done statement. The for loop doesn't test a condition, but uses a list instead.

*For  
variable  
in list  
do  
statements  
done*

The while loop executes the statements as long as the condition remains true.

*while [  
condition  
] do  
statements  
done*

The until loop complements the while construct in the sense that the statements are executed as long as the condition remains false.  
*until [ condition ]*

*d  
o  
  
s  
t  
a*

*t*  
*e*  
*m*  
*e*  
*n*  
*t*  
*s*

*d*  
*o*  
*n*  
*e*

**Ar**  
**mst**  
**ron**  
**g**  
**Nu**  
**mbe**  
**r**  
**Alg**  
**orit**  
**hm**

Step 1 : Start

Step 2 : Read number

Step 3 : Initialize 0 to sum and number to num

Step 4 : Extract last digit by computing number modulo 10

Step 5 : Cube the lastdigitand  
add it to sumStep 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.sh)**

```
# Armstrong number using
while loopecho -n"Enter a
number : "
read n a=$n s=0 while [
$n -gt 0 ] dor=`expr $n
% 10`
s=`expr $s + \$r \* \$r \* \$r \`"
n=`expr $n / 10`doneif [ $a -eq $s ]
then
echo "Armstrong Number" else
echo -n "Not an Armstrong number" fi
```

#### **OUTPUT:**

```
[student@vecit ~]$ sh
armstrong.sh Enter a number
: 370 Armstrong Number
```

**RESULT:**

Thus using loops, iterative scripts were executed

<b>EX.NO:3</b>	<b>IMPLEMENTATION OF FORK, EXEC, GETPID, EXIT, WAIT, AND CLOSE SYSTEM CALLS.</b>
<b>DATE:</b>	

**AIM:**

To write a program for implementing process management using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

**ALGORITHM:**

- Start the program.
- Read the input from the command line.
- Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID
- execvp() system call used to execute that command given on that command line argument
- execlp() system call used to execute specified command.
- Open the directory at specified in command line input.
- Display the directory contents.
- Stop the program.

**PROGRAM:**

```
#include<sys  
stdio.h>  
main(int  
argc,char*arg  
[])  
{  
  
int pid; char s[100];  
pid=fork();if(pid<0)  
printf("error");else  
if(pid>0)
```

```
{  
wait(NULL);  
  
printf("\n Parent Process:\n");  
printf("\n\tParent Process  
id:%d\n",getpid());execlp("cat","cat",ar[  
1],(char*)0); error("can't execute cat  
%s,",ar[1]);  
}  
  
else  
{  
  
}  
}
```

```
printf("\nChild process:");

printf("\n\tChild process parent id:\t %d",getppid());
printf(s,"n\tChild process id :%d",getpid());
write(1,s,strlen(s));
printf(" ");

printf(" ");

printf(" "); execvp(ar[2],&ar[2]);
error("can't execute %s",ar[2]);
```

### **OUTPUT:**

[root@localhost ~]# ./a.out tst date Child process:

Child process id :

3137 Sat Apr 10 02:45:32 IST 2010

Parent Process:

Parent Process id:3136

sd

dsaASD[root@localhos  
t ~]# cat tst sddsaASD

### **RESULT:**

Thus the program for process management was written and successfully executed.

<b>EX.NO:4A</b>	<b>IMPLEMENTATION OF FCFS SCHEDULING ALGORITHM</b>
<b>DATE:</b>	

### **AIM**

To write a C program to implement First Come First Serve scheduling algorithm.

### **DESCRIPTION:**

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

### **ALGORITHM:**

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on order in which it requests CPU.

Step 4: Compute the waiting time and turnaround time for

each process.

Step 5: Calculate the average waiting time and

average turnaround time.

Step 6: Print the details about all

the processes.

Step 7: Stop the

program.

### **PROGRAM**

#### **PROGRAM:**

```
#includ
e<stdio.
h>Void
main()
{
```



```
%d\
t\",
wt[i
]);
printf("%d",wt[n]+bt[n]); printf("\n
\n");
```

p

r

i

n

t

f

(

"

\

n

"

)

;

f

o

r

(

i

=

0

;

i

<

n

;

i

+

+

)

w

a

t

[

i

]

=

w

t

[

i

]

-

a

t

[

i

]

;

f

o

r

(

i

=

0

;

i

<

n

;

i

```

+
+
)

t
a
t
[
i
]
=
w
a
t
[
i
]
-
a
t
[
i
]
;
printf("\n FIRST COME FIRST SERVE\n");

printf("\n Process Burst-time Arrival-time Waiting-time Finish-time Turnaround-time\n");
for(i=0;i<n;i++)
printf("\n\n%5d%5s%5d%5d%5d%5d%5d%5d%5d",i+1,p[i],bt[i],at[i],wat[i],ft[i],tat[i]);
for(i=0;i<n;i++)
sum=sum+wat[i];awt=sum/n;
for(i=0;i<n;i++)
sum1=sum1+bt[i]+wt[i];
;att=sum1/n;

```

```
printf("\n\nAverage waiting  
time:%f",awt); printf("\n\nAverage  
turnaround time:%f",att);  
}
```

**OUTPUT:**

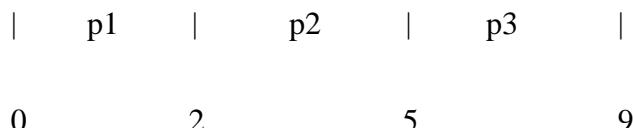
enter the number of process 3

Enter the process name  
and burst-time:p12

p2 3

p3 4

Enter the  
arrival-  
time:0 1 2  
GANTT  
CHART



**FIRST COME FIRST SERVE**

Proce ss	Burst-time	Arrival-time	Waiting-time	Finish-time	Turnaround-time
p1	2	0	0	2	2
p2	3	1	1	5	4
p3	4	2	3	9	7

Average waiting  
time:1.333333 Average  
turnaround  
time:5.333333

**RESULT:**

The FCFS scheduling algorithm has been implemented in C.

<b>EX.NO:4B</b>	<b>IMPLEMENTATION OF SJF SCHEDULING ALGORITHM</b>
<b>DATE:</b>	

**AIM:**

To write a C program to implement shortest job first (non-pre-emptive) scheduling algorithm.

**DESCRIPTION:**

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

**ALGORITHM:**

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on burst time.

Step 4: Compute the waiting time and turnaround time for each process.  
Step 5: Calculate the average waiting time and average turnaround time.  
Step 6: Print the details about all the processes.

Step 7: Stop the program.

**PROGRAM:**

```
#includ
e<stdio.
h>void
main()
{
    int
    i,j,n,bt[30],at[30],st[30],ft[30],wat[30],wt[30],temp,temp1,tot,tt[30];float
    awt, att;int p[15];
    wat[1]=0;
    printf("ENTER THE NO.OF PROCESS");
    scanf("%d",&n);
    printf("\nENTER THE PROCESS NUMBER,BURST TIME AND
ARRIVALTIME");
    for(i=1;i<=n;i++)
    {
        scanf("%d\t %d\t %d",&p[i],&bt[i],&at[i]);
    }
    printf("\nPROCESS\tBURSTTIME\tAR
RIVALTIME");
    for(i=1;i<=n;i++)
    {
        printf("\np% d\t% d\t% d",p[i],bt[i],at[i]);
    }
    for(i=1;i<=n;i++)
```

```
{  
for(j=i+1;j<=n;j++)  
{  
if(bt[i]>bt[j])  
{  
temp=bt[i]; bt[i]=bt[j]; bt[j]=temp;  
temp1=p[i];p[i]=p[j]; p[j]=temp1;  
}  
}  
if(i==1)  
{  
  
}  
else  
{
```

```

st[1]=0;
ft[1]=bt[1]; wt[1]=0;

st[i]=ft[i-1];
ft[i]=st[i]+bt[i];
wt[i]=st[i];
}

}

printf("\n\n\t\tGANTT CHART\n"); printf("\n
                                \n");
for(i=1;i<=n;i++)
printf("\t\t\t",p[i
]); printf("\t\n");
printf("\n
                                \n");
printf("\n
                                \n");
for(i=1;
i<=n;i+
)
printf(
%d
\t\t,wt[
i]);
printf("%d",wt[n]+bt[n]); printf("\n
                                \n");
for(i=2;i<=n;i++)
wat[i]=wt[i]-at[i];for(i=1;i<=n;i++)
tt[i]=wat[i]+bt[i]-at[i];
printf("\nPROCESS\tBURSTTIME\tARRIVALTIME\tWAITINGTIME\tT
URNAROUNDTIME\n");

```

```

for(i=1;i<=n;i++)
{
    printf("\np%d %5d %15d %15d %15d",p[i],bt[i],at[i],wat[i],tt[i]);
}

for(i=1,tot=0;i<=n;i++)t
ot+=wt[i];
awt=(float)tot/n;
printf("\n\n AVERAGE WAITING TIME=%f",awt);for(i=1,tot=0;i<=n;i++)
tot+=tt[i];att=(float)tot/n;
printf("\n\n AVERAGE TURNAROUND TIME=%f",att);

}

```

**OUTPUT:**

enter the no.of process3

enter the process number,burst time and

arrival time  
1 8 12 5 1  
3 3 1

PROCESS	BURSTTIME	ARRIVALTIME	WAITINGTIME	TURNAROUNDTIME
p3	3	1	0	2
p2	5	1	2	6
p1	8	1	7	14

AVERAGE WAITING  
TIME=3.666667 AVERAGE  
TURNAROUND TIME=7.333333

**RESULT:**

The SJF scheduling algorithm has been implemented in C.

<b>EX.NO.4C:</b>	<b>IMPLEMENTATION OF ROUND ROBIN SCHEDULING ALGORITHM</b>
<b>DATE:</b>	

**AIM:**

To write a C program to implement Round Robin scheduling algorithm.

**DESCRIPTION:**

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

**ALGORITHM:**

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on priority.

Step 4: Compute the waiting time and turnaround time for

each process.

Step 5: Calculate the average waiting time and

average turnaround time. Step 6: Print the details about all the processes.  
Step 7: Stop the program.

**PROGRAM:**

```
#include<stdio.h> void main()
{
    int ct=0,y[30],j=0,bt[10],cwt=0; int
    tq,i,max=0,n,wt[10],t[10],at[10],tt[10],b[1
    0]; float a=0.0,s=0.0;
    char p[10][10];

    printf("\n enter the no
    of process:");
    scanf("%d",&n);
    printf("\n enter the time
    quantum");scanf("%d",&tq);
    printf("\n enter the process
    name,burst time,arrival time");

    for(i=0;i<n;i++)
    {
        scanf("%s",p[i]);
        scanf("%d",&bt[i]);
        scanf("%d",&at[i]);
        wt[i]=t[i]=0;b[i]=bt[i];
    }
    printf("\n\t\tGANTT CHART");printf("\n
                                \n");
    for(i=0;i<n;i++)
    {
        i
        f
        (
    }
```

m  
a  
x  
<  
b  
t  
[  
i  
]  
)

m  
a  
x  
=  
b  
t  
[  
i  
]  
;  
}  
while(max!=0)  
{  
for(i=0;i<n;i++)  
{  
if(bt[i]>0)  
{  
i  
f  
(  
c  
t  
=

=  
0  
)

w  
t  
[  
i  
]  
=  
w  
t  
[  
i  
]  
+  
c  
w  
t  
;

e  
l  
s  
e  
wt[i]=wt[i]+(cwt-t[i]);  
}

if(bt[i  
]==0)  
cwt=c  
wt+0;  
else  
if(bt[i

```
]==m  
ax)  
{  
if(bt[i]>tq)  
{  
cwt=cwt+tq;  
  
}  
else  
{  
  
}  
printf("\t%s",p[i]); y[j]=cwt;j++;  
}  
else if(bt[i]<tq)  
{
```

```
bt[i]=bt[i]-tq;max=max-tq;
```

```
cwt=cwt+bt[i];bt[i]=0; max=0;
```

```
cwt=cwt+bt[i]; bt[i]=0;
```

```
printf("\t%s",p[i]);
```

```
y[j]=cwt;
```

```
j++;
```

```
}
```

```
else if(bt[i]>tq)
```

```
{
```

```
cwt=cwt+tq
```

```
;
```

```
bt[i]=bt[i]-
```

```
tq;
```

```
printf("\t%
```

```
s",p[i]);
```

```
y[j]=cwt;
```

```
j++;
```

```
}
```

```
else if(bt[i]==tq)
```

```
{
```

```
cwt=cwt+bt[i];
```

```
printf("\t%s",p[i]); bt[i]=0;y[j]=cwt; j++;
```

```
}
```

```
t[i]=cwt;
```

```
}
```

```
ct=ct+1;
```

```

}

for(i=0;i<n;i++)
{
    wt[i]=wt[i]-at[i]; a=a+wt[i];
    tt[i]=wt[i]+b[i]-at[i];s=s+tt[i];
}

a=a/n;
s=s/n;
printf("\n          ");printf("\n0"); for(i=0;i<j;i++)
printf("\t%d",y[i]);printf("\n");
printf("\n          "); printf("\n\t\t ROUND ROBIN\n");

printf("\n          Process    Burst-time    Arrival-time    Waiting-
time           Turnaround-time\n");
for(i=0;i<n;i++)

printf("\n\n %d%s \t %d\t\t %d \t\t %d\t\t %d", i+1, p[i], b[i], at[i],wt[i], tt[i]);
printf("\n\nAvg waiting time=%f",a);
printf("\n\nAvgturn around time=%f",s);
}

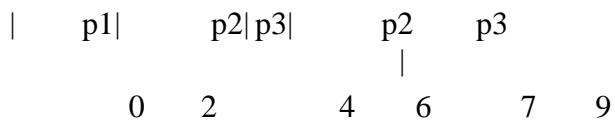
```

## **OUTPUT:**

enter the no of  
process:3 enter  
the time  
quantum2  
enter the process name, bursttime, arrival time

p1	2	0
p2	3	1
p3	4	2

## **GANTT CHART**



## ROUND ROBIN

Process	Burst-time	Arrival-time	Waiting-time	Turnaround-time
p1	2	0	0	2
p2	3	1	3	5
p3	4	2	3	5

Avg Waiting

Time=2.000000 Avg

Turnaround

Time=4.000000

## RESULT

The Round Robin scheduling algorithm has been implemented in C.

<b>EX.NO:4D</b>	<b>IMPLEMENTATION OF PRIORITY</b>
<b>DATE:</b>	<b>SCHEDULING ALGORITHM</b>

### AIM:

To write a C program to implement Priority Scheduling algorithm.

### DESCRIPTION:

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order

with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

### **ALGORITHM:**

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on priority.

Step 4: Compute the waiting time and turnaround time for each process.  
Step 5: Calculate the average waiting time and average turnaround time.  
Step 6: Print the details about all the processes.

Step 7: Stop the program.

### **PROGRAM:**

```
#inclu
de<std
io.h>
#inclu
de<str
ing.h>
void
main()
{
int bt[30],pr[30],np;
intwt[30],tat[30],wat[30],at[30],ft[30];int i,j,x,z,t;float
sum1=0,sum=0,awt,att;char
p[5][9],y[9];
printf("\nenter the number of
process");scanf("%d",&np);
printf("\nEnter the process,burst-time and
priority:");
for(i=0;i<np;i++)
{
```



```
(  
"  
\n  
"  
)  
;  
f  
o  
r  
(  
i  
=  
0  
;  
i  
<  
=  
n  
p  
;  
i  
+  
+  
)  
printf("%d\t",wt[i]  
)printf("\n");printf("\n");  
for(i=0;i<np;i++)  
  
wat[i]=wt[i]-  
at[i];for(i=0;i<np;i++)  
tat[i]=wat[i]-at[i];  
printf("\nPRIORITY  
SCHEDULING:\n");
```

```

printf("\nProcess Priority Burst-time Arrival-time Waiting-time Turnaround-time");
for(i=0;i<np;i++)
printf("\n\n%d %d %d %d %d %d",i+1,p[i],pr[i],bt[i],at[i],wt[i],tat[i]);
for(i=0;i<np;i++)
sum=sum+wat
[i];awt=sum/np
;
for(i=0;i<np;i+
++)
sum1=sum1+ta
t[i];att=sum1/n
p;
printf("\n\nAverage waiting time:%f",awt); printf("\n\nAverage turnaround time is:%f",att);
}

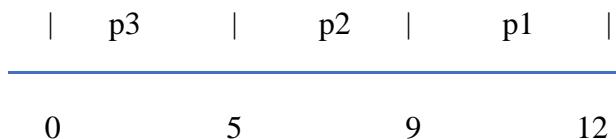
```

**OUTPUT:**

Enter the number of process3

Enter the process, burst-time  
and priority:p1 3 3p2 4 2  
p3 5 1

Enter the  
arrival-  
time: 0 1 2  
GANTT  
CHART



**PRIORITY SCHEDULING:**

Process	Priority	Burst-time	Arrival-time	Waiting-time	Turnaround-time
p3	1	5	0	0	0

p2	2	4	1	5	3
p1	3	3	2	9	5

Average waiting time:

3.666667 Average turnaround

time is: 2.666667

## RESULT

The Priority scheduling algorithm has been implemented in C.

**AIM:**

To write a program for developing Application using Inter Process communication with pipes.

**ALGORITHM:**

- Start the program.
- Read the input from parent process and perform in child process.
- Write the date in parent process and read it in child process.
- Data is read.
- Stop the program.

**SHARED MEMORY FOR WRITER PROCESS**

```
#includ
e
<iostrea
m>
#includ
e
<sys/ipc
.h>
#includ
e
<sys/sh
m.h>
#includ
e
<stdio.h
> using
namesp
ace
std;int
main()
{
```

```

// ftok to generate unique key key_tkey
=ftok("shmfile",65);
// shmget returns an identifier in shmid
int shmid = shmget(key,1024,0666|IPC_CREAT);
// shmat to attach to shared memory

char *str = (char*)
shmat(shmid,(void*)0,0);printf("WriteData :
");
gets(str);
printf("Data written in memory: %s\n",str);

//detach from shared
memoryshmdt(str);
return 0;

}

```

## **SHARED MEMORY FOR READER PROCESS**

```

#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

```

```

> using
namesp
ace
std::int
main()
{
    // ftok to generate unique key key_tkey
    =ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*)
    shmat(shmid,(void*)0,0);printf("Data read
from memory: %s\n",str);
    // detach from shared memory shmdt(str);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL); return
    0;
}

```

O  
U  
T  
P  
U  
T  
:

The image displays two terminal windows side-by-side. Both windows have a title bar 'andres@andres: ~/Programs/OS'. The left terminal window has a green title bar icon. It contains the following text:  
 andres@andres:~/Programs/OS\$ ./writer  
 Write Data : Geeks for Geeks  
 Data written in memory: Geeks for Geeks  
 andres@andres:~/Programs/OS\$ █

The right terminal window has a blue title bar icon. It contains the following text:  
 andres@andres:~/Programs/OS\$ ./reader  
 Data read from memory: Geeks for Geeks  
 andres@andres:~/Programs/OS\$ █

**RESULT:**

Thus the program was executed successfully.

**AIM:**

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

**DESCRIPTION:**

Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

**ALGORITHM:**

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

Step 2: In the case of producer process

- Produce an item in to temporary variable.
- If there is empty space in the buffer check the mutex value for enter into the critical section.
- 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

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

- Consume the item.
- Step 4: Print the result

### **PROGRAM :**

```
#define BUFFERSIZE 10

int
mutex,n,empty,full=0,ite
m,item1;intbuffer[20];
int
in=0,out=0,mut
ex=1;void
wait(int s)
{
while(s<0)
{
}

s--;
}
```

```
printf("\nCannot add an item\n");exit(0);
```

```
void signal(int s)
```

```
{
```

```
s
```

```
+
```

```
+
```

```
;
```

```
}
```

```
void producer()
```

```
{
```

```
do
```

```
{
```

```
wait (empty);wait(mutex);
```

```
printf("\nEnter an  
item:");scanf("%d",&item);
```

```
buffer[in]=item;  
in=in+1; signal(mutex);signal(full);
```

```
}
```

```
while(in<n);
```

```
}
```

```
void consumer()
```

```
{
```

```
do
```

```
{
```

```
wait(f
```

```
ull);
```

```
wait(
```

```
mutex
```

```
);
```

```
item1  
=buff  
er[out  
];  
printf("\nConsumed item  
=%d",item1);out=out+1;  
signal(mutex);signal(empty);  
}  
while(out<n);  
}  
void main()  
{  
  
printf("Enter the value  
of n:");scanf("%d",&n);  
empty=  
n;  
while(in  
<n)  
produce  
r();whil  
e(in!=o  
ut)  
consum  
er();  
}
```

### **OUTPUT:**

```
$ cc prco.c  
$ a.out
```

```
Enter the  
value of n  
:3Enter the  
item:2
```

Enter

the

item:5

Enter

the

item:9

consum

ed

item=2

consum

ed

item=5

consum

ed

item=9

\$

**RESULT:**

Thus the program for solving producer and consumer problem using semaphore was executed successfully.

**AIM:**

To Simulate Algorithm for Deadlock avoidance

**DESCRIPTION:**

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type

**ALGORITHM:**

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

Step 5: Check whether its possible to allocate. If possible it is safe state

Step 6: If the new request comes then check that the system is in safety or not if weallow therequest.

Step 7:

Stop the

execution

10.Stop

the

program

## **PROGR**

### **AM:**

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

```
void main()
```

```
{
```

```
int
```

```
pno,rno,i,j,prc,count,t,tot
```

```
al;count=0;clrscr();
```

```
printf("\n Enter number of
```

```
process:");scanf("%d",&pno); printf("\n
```

```
Enter number of
```

```
resources:");scanf("%d",&rno);for(i=1;i<
```

```
=pno;i++)
```

```
{
```

```
flag[i]=0;
```

```
}
```

```
printf("\n Enter total numbers of each
```

```
resources:");for(i=1;i<= rno;i++)scanf("%d",&tres[i]);
```

```
printf("\n Enter Max resources for each process:");for(i=1;i<= pno;i++)
```

```
{
```

```

printf("\n for process %d:",i);for(j=1;j<=
rno;j++)scanf("%d",&max[i][j]);
}

printf("\n Enter allocated resources for each
process:");for(i=1;i<=pno;i++)
{
printf("\n for process %d:",i);for(j=1;j<=
rno;j++)scanf("%d",&allocated[i][j]);
}

printf("\n available resources:\n");for(j=1;j<= rno;j++)
{
avail[j]
=0;total
=0;
for(i=1;
i<=
pno;i++)
{
total+=allocated[i][j];
}

avail[j]=tres[j]-
total;work[j]=avail[j];
printf("           %d \t",work[j]);
}

do
{
for(i=1;i<= pno;i++)
{
for(j=1;j<= rno;j++)
{
need[i][j]=max[i][j]-allocated[i][j];
}
}
}

```

```

    }
}

printf("\n Allocated matrix Max need");

for(i=1;i<= pno;i++)
{
printf("\n"); for(j=1;j<= rno;j++)
{
printf("%4d",allocated[i][j]);
}
printf(" |"); for(j=1;j<= rno;j++)
{
printf("%4d",max[i][j]);
}

printf(" |"); for(j=1;j<= rno;j++)
{

}

prc=0; for(i=1;i<= pno;i++)
{
if(flag[i]==0)
{
prc=i;
for(j=1;j<= rno;j++)
{
if(work[j]< need[i][j])
{
prc=0; break;
}
}
}
}
```

```
 }  
 }  
 if(prc!=0)break;  
 }
```

```

printf("%4d",need[i][j]);
}

if(prc!=0)
{
printf("\n Process %d
completed",i);count++; printf("\n
Available matrix:");for(j=1;j<=
rno;j++)

{
work[j]+=allocated[prc][j];

allocated[prc][j]=
0; max[prc][j]=0;

flag[prc]=1;
printf(" %d",work[j]);
}

}

}while(count!
=pno&&prc!
=0);
if(count==pn
o)
printf("\nThe system is in a
safe state!!");else
printf("\nThe system is in an
unsafe state!!");getch();
}

```

**OUTPUT:**

Enter number of

process:5 Enter

number of

resources:3

Enter total numbers of each resources:10

5 7Enter Maxresources for each process:

for process 1: 7 5 3

for process 2: 3 2 2

for process 3: 9 0 2

for process 4: 2 2 2

for process 5: 4 3 3

Enter allocated resources for

each process:forprocess 1: 0 1

0

for process 2: 3 0 2

for process 3: 3 0 2

for process 4: 2 1 1

for process 5: 0 0 2

available resources:

2            3            0

Allocated matrix

Max

need

0	1	0	7	5	3	7	4	3
3	0	2	3	2	2	0	2	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 2 completed

Available matrix: 5 3 2 Allocated

matrix			Max			need		
0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 4 completed

Available matrix: 7 4 3 Allocated

matrix			Max			need		
0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 1 completed

Available matrix: 7 5 3 Allocated

matrix			Max			need		
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	3	1	

Process 3 completed

Available matrix: 10 5 5

Allocated matrix                    Max      need

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 5 completed

Available matrix:    10            5 7

The system is in a safe state!!

## **RESULT:**

Thus the program to implement the deadlock avoidance was executed and verified.

## **AIM:**

To Simulate Algorithm for Deadlock detection

## **ALGORITHM:**

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value..

Step 4: After allocation find the

need value. Step 5: Check

whether its possible to allocate.

Step 6: If it is possible then the system is in  
safe state. Step 7: Stop the execution

## **PROGRAM**

```
#includ
```

```
e<stdio.
```

```
h>
```

```
#includ
```

```
e<conio
```

```
.h> int
```

```
max[10
```

```
0][100];
```

```
in
```

```
t
```

```
al
```

```
lo
```

```
c[
```

```
1
```

```
0
```

```
0]
```

```
[1
```

```
0
```

```
0]
```

```
;
```

```
in
```

t

n

e

e

d[

1

0

0]

[1

0

0]

;

i

n

t

a

v

a

i

l

[

1

0

0

]

;

i

n

t

n

,

r

;

v

o

i

d

i

n

p

u

t

(

)

;

v

o

i

d

s

h

o

```
w
(
)
;

v
o
i
d

c
a
l
(
)
;
int main()
{
    int i,j;
    printf("***** Deadlock Detection Algo *****\n");
    input();show();cal();
    getch(); return 0;
}

void input()
{
    int i,j;
```

```
printf("Enter the no of Processes\t");
scanf("%d",&n);

printf("Enter the no of resource
instances\t");scanf("%d",&r);printf("Enter the
Max Matrix\n");
for(i=0;i<n;i++)

{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}

}

printf("Enter the
Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}

}

printf("Enter the available Resources\n");for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}

}

void show()

{
int i,j;
```

```
printf("Process\t Allocation\t Max\t
Available\t");for(i=0;i<n;i++)
{
printf("\nP%d\t",i+1);

for(j=0;j<r;j++)
{
printf("%d ",alloc[i][j]);
}

printf("\t");

for(j=0;j<r;j++)
{
printf("%d ",max[i][j]);
}

printf("\t");

if(i==0)
{
for(j=0;j<r;j++) printf("%d ",avail[j]);
}
}

void cal()
{

int
finish[100],temp,need[100][100],fla
g=1,k,c1=0;int dead[100];
int safe[100];

i
n
```

```
t  
  
i  
,  
j  
;  
  
f  
o  
r  
(  
i  
= 0  
;  
i  
< n  
;  
i  
+  
+  
)  
{  
finish[i]=0;  
}  
//find need matrix  
  
for(i=0;i<n;i++)  
{  
for(j=0;j<r;j++)  
{  
need[i][j]=max[i][j]-alloc[i][j];  
}
```

```
}

while(flag)
{
flag=0;

for(i=0;i<n;i++)
{
int c=0;

for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{

c
+
+
;
if(c==r)
{
for(k=0;k<r;k++)
{



av
ail[
k]
+=
all
oc[
i][j
];
}
```

```
fin  
ish  
[i]  
=1;  
  
flag=1;  
}  
//printf("\nP%d",i); if(finish[i]==1)  
{  
  
}
```

```
i=n;  
}  
}  
}  
}  
}  
} j=0;  
flag=0; for(i=0;i<n;i++)  
{  
if(finish[i]==0)  
{  
d  
e  
a  
d  
[  
j  
]  
=  
i  
;  
j  
+  
+  
;  
  
f  
l  
a  
g
```

```
=  
1  
;  
}  
}  
if(flag==1)  
{  
    printf("\n\nSystem is in Deadlock and the Deadlock process are\n");  
  
}  
else  
{  
  
}  
}
```

```
for(i=0;i<n;i++)
```

```
{
```

```
}
```

```
printf("P%d\t",dead[i]);
```

**OUTPUT:**

Enter the no. Of processes 3

Enter the no of resources

instances 3Enter the max

matrix

3 6 8

4 3 3

3 4 4

Enter the allocation matrix 3 3 3

2 0 3

1 2 4

Enter the available resources 1 2 0

Process	allocation	max	available
P1	3 3 3	3 6 8	1 2 0
P2	2 0 3	4 3 3	
P3	1 2 4	3 4 4	

System is in deadlock and deadlock

process are P1 P2 P3

## **RESULT:**

Thus the program to implement the deadlock detection was executed successfully.

<b>EX.NO: 9</b>	<b>IMPLEMENTATION OF THREADING</b>
<b>DATE:</b>	

### **AIM:**

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

### **ALGORITHM:**

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

**PROGRAM:**

```
#i  
n  
cl  
u  
d  
e  
<  
st  
di  
o.  
h  
>  
#i  
n  
cl  
u  
d  
e  
<  
st  
ri  
n  
g.  
h  
>  
#i  
n  
cl  
u  
d  
e  
<  
pt  
hr  
e  
a  
d.  
h  
>  
#i  
n  
cl  
u  
d
```

```
e
<
st
dl
ib
.h
>
#i
n
cl
u
d
e
<
u
ni
st
d.
h
>
pt
h
re
a
d
_t
ti
d
[
2
];
void* doSomeThing(void *arg)
{
unsigned
long i =
0;
pthread_t
id =
pthread_s
elf();

if(pthread_equal(id,tid[0]))
{
printf("\n First thread processing\n");
}
else
```

```
{  
printf("\n Second thread processing\n");  
}
```

```
f  
o  
r  
(  
i  
=  
0  
;  
  
i  
<  
(  
0  
x  
F  
F  
F  
F  
F  
F  
F  
F  
F  
)  
;  
i++);  
return NULL;  
}  
int main(void)  
{  
int i = 0;  
  
i  
n  
t  
  
e  
r  
r  
;
```

```
w
h
i
l
e
(
i

<

2
)
{

err = pthread_create(&(tid[i]), NULL,
    &doSomeThing, NULL);if (err !=
    0)
printf("\ncan't create thread
:[%s]", strerror(err));else
printf("\n Thread created successfully\n");

i++;
}

s
l
e
e
p
(
5
)
;

r
e
t
u
```

r

n

0

;

}

### **OUTPUT:**

Thread created successfully

First  
thread  
processing  
Thread  
created  
successfull  
y

Second thread processing

**RESULT:**

Thus the implementation of threading using c program was executed successfully.

<b>EX.NO:10</b>	<b>IMPLEMENTATION OF PAGING TECHNIQUE</b>
<b>DATE:</b>	

**AIM:**

To write a C program to implement paging concept for memory management.

**ALGORIHTM:**

Step 1: Start the program.

Step 2: Enter the logical memory address.

Step 3: Enter the page table which has offset and page frame.

Step 4: The corresponding physical address can be calculate by,  $PA = [pageframe * No. of pagesize] + Page\ offset$ .

Step 5: Print the physical address for the corresponding logical address.

Step 6: Terminate the program.

**PROGRAM:**

```
#  
i  
n  
c  
l  
u  
d  
e  
<  
s  
t  
d
```

i  
o  
.h  
>

#  
i  
n  
c  
l  
u  
d  
e  
<  
c  
o  
n  
i  
o  
.h  
>

m  
a  
i  
n  
(  
)  
{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

int  
s[10]

```
,  
fno[1  
0][20  
];  
clrscr  
( );  
printf("\nEnter the memory size -- ");  
  
scanf("%d",&ms);  
  
printf("\nEnter the page  
size -- ");  
scanf("%d",&ps);  
nop = ms/ps;  
  
printf("\nThe no. of pages available in memory are -- %d ",nop);  
  
printf("\nEnter number of processes -- ");  
scanf("%d",&np);  
  
r  
e  
m  
p  
a  
g  
e  
s  
  
=  
  
n  
o  
p  
;  
  
f  
o
```

```
r
(
i
=
1
;

i
<
=
n
p
;
i
+
+
)

{

printf("\nEnter no. of pages required for
p[%d]-- ",i);scanf("%d",&s[i]);
if(s[i] >rmpages)
{
printf("\n
Memory is
Full");
break;
}

rmpages = rmpages -
s[i]; printf("\nEnter
```

```
pagetable for p[%d] ---
```

```
",i); for(j=0;  
j<s[i];  
j++)  
scanf(  
"%d",  
&fno[  
i][j]);  
}
```

```
printf("\nEnter    Logical    Address    to    find  
          Physical    Address    ");printf("\nEnter  
process no. and pagenumber and offset -- ");  
scanf("%d %d  
%d",&x,&y, &offset);  
if(x>np || y>=s[i] ||  
offset>=ps)  
printf("\nInvalid Process or Page  
Number or offset");else  
{  
    pa=fno[x][y]*ps+offset;  
    printf("\nThe Physical Address is -- %d",pa);  
}  
  
getch();  
}
```

### **OUTPUT:**

Enter the memory

size – 1000Enter

the page size -- 100

The no. of pages available in memory are 10

Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3]-- 5

Memory is Full

Enter Logical Address to find Physical Address Enter process no. and pagenumberand offset

-

- 2

3

60

The Physical Address is----- 760

## **RESULT:**

Thus C program for implementing paging concept for memory management has been executed successfully.

<b>Ex.NO: 11</b>	<b>IMPLEMENTATION OF MEMORY ALLOCATION TECHNIQUES</b>
<b>DATE:</b>	

## **AIM:**

To write a C program to implement Memory Management concept using the technique best fit, worst fit and first fit algorithms.

## **ALGORITHM:**

- Get the number of process.
- Get the number of blocks and size of process.
- Get the choices from the user and call the corresponding switch cases.
- First fit -allocate the process to the available free block match with the size of the process
- Worst fit – allocate the process to the largest block size available in the list
- Best fit -allocate the process to the optimum size block available in the list
- Display the result with allocations

## **PROGRAM:**

```
#include<stdio.h>
main()
{
    int p[10],np,b[10],nb,ch,c[10],d[10],alloc[10],flag[10],i,j;

    printf("\nEnter the no
    of process:");
    scanf("%d",&np);
    printf("\nEnter the no of blocks:");

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

```

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

printf("\nEnter the block sizes:");for(j=0;j<nb;j++)
{
printf("\nBlock
%d:",j);
scanf("%d",&b[j])
;c[j]=b[j];d[j]=b[j]
;
}

if(np<=nb)
{
printf("\n1.First fit 2.Best fit 3.Worst fit");do
{
printf("\nEnter your
choice:");scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nFirst Fit\n");for(i=0;i<np;i++)
{
for(j=0;j<nb;j++)
{
if(p[i]<=b[j])
{
alloc[j]=p[i];
printf("\n\nAllo
");
}
}
}
}
}

```

```
c[%d]",all  
oc[j]);  
printf("\n\nProcess %d of size %d is allocated in block:%d of size:%d",i,p[i],j,b[j]);  
  
f  
l  
a  
g  
[  
i  
]  
=  
0  
,b  
[  
j  
]  
=  
0  
;  
  
b  
r  
e  
a  
k  
;  
}  
  
e  
l  
s  
e
```

```
f
l
a
g
[
i
]
=
1
;

}

for(i=0;i<np;i++)
{
```

```
}
```

```
if(flag[i]!=0)
```

```
printf("\n\nProcess %d of size %d is not allocated",i,p[i]);
```

```
}
```

```
break;
```

```
case 2: printf("\nBest Fit\n");for(i=0;i<nb;i++)
```

```
{
```

```
for(j=i+1;j<nb;j++)
```

```
{
```

```
if(c[i]>c[j])
```

```
{
```

```
int temp=c[i];
```

```
c[i]=c[j]; c[j]=temp;
```

```
}
```

```
}
```

```
}
```

```
printf("\nAfter sorting block sizes:");
```

```
for(i=0;i<nb;i
```

```
++)
```

```
printf("\nBloc
```

```
k
```

```
%d:%d",i,c[i]
```

```
);
```

```
for(i=0;i<np;i  
++)  
{  
for(j=0;j<nb;j++)  
{  
if(p[i]<=c[j])  
{  
}  
}  
else  
}  
for(i=0;i<np;i++)  
{  
}  
break;
```

```

    alloc[j]=p[i];

    printf("\n\nAlloc[%d]",alloc[j]);

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

}

flag[i]=1;

if(flag[i]!=0)

printf("\n\nProcess %d of size %d is notallocated",i,p[i]);

case 3: printf("\nWorst Fit\n");for(i=0;i<nb;i++)

{

for(j=i+1;j<nb;j++)

{

if(d[i]<d[j])

{

int temp=d[i];d[i]=d[j]; d[j]=temp;

}

}

printf("\nAfter sorting block
sizes:");
for(i=0;i<nb;i++)printf("\nBlock
%d:%d",i,d[i]);
for(i=0;i<np;i++)

{

for(j=0;j<nb;j++)

```

```
{  
if(p[i]<=d[j])  
{  
alloc[j]=p[i];  
printf("\n\nAll  
oc[%d]",alloc[  
j]);  
printf("\n\nProcess %d of size %d is allocated in block %d of size  
%d",i,p[i],j,d[j]  
  
flag[i]=0,d[j]=0;break;  
}  
else flag[i]=1;  
}  
for(i=0;i<np;i++)  
{  
}  
break; default:  
printf("Invalid Choice...!");break;
```

}

if(flag[i]!=0)

```
printf("\n\nProcess %d of size  
%d is not allocated",i,p[i]);  
}  
}  
}  
}
```

### **OUTPUT:**

Enter the no of  
process:3Enter  
the no of  
blocks:3  
Enter the size of each process:

Process 0:100

Process 1:150

Process 2:200

Enter the block sizes:

Block 0:300

Block 1:350

Block 2:200

1.First fit 2.Best

fit 3.Worst fit

Enter your

choice:1

Alloc[100]

Process 0 of size 100 is allocated in block 0 of size

300Alloc[150]Process 1 of size 150 is allocated in block

1 of size 350Alloc[200]

Process 2 of size 200 is allocated in block 2 of

size 200Enter your choice:2

Best Fit

After sorting block

sizes are:Block0:200

Block 1:300

Block 2:350

Alloc[100]

Process 0 of size 100 is allocated in block:0 of

size:200Alloc[150]Process 1 of size 150 is allocated in

block:1 of size:300Alloc[200]

Process 2 of size 200 is allocated in block:2

of size:350enter your choice:3

Worst Fit

After sorting block sizes are:

Block 0:350

Block 1:300

B

l

o

c

k

2  
:  
2  
0  
0

A  
1  
1  
o  
c  
[  
1  
0  
0  
]

Process 0 of size 100 is allocated in block 0 of size  
350Alloc[150]Process 1 of size 150 is allocated in block  
1 of size 300Alloc[200]  
Process 2 of size 200 is allocated in block 2 of  
size 200Enter your choice:6  
Invalid Choice...!

**RESULT:**

Thus a UNIX C program to implement memory management scheme using Best fitworst fitand first fit were executed successfully.

**AIM:**

To write a UNIX C program to implement FIFO page replacement algorithm.

**DESCRIPTION :**

The FIFO Page Replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen . There is not strictly necessary to record the time when a page is brought in. By creating a FIFO queue to hold all pages in memory and by replacing the page at the head of the queue. When a page is brought into memory, insert it at the tail of the queue.

**ALGORITHM:**

- Start the process
- Declare the size with respect to page length
- Check the need of replacement from the page to memory
- Check the need of replacement from old page to new page in memory
- Format queue to hold all pages
- Insert the page require memory into the queue
- Check for bad replacement and page fault
- Get the number of processes to be inserted
- Display the values
- Stop  
the  
process

**PROGR****AM:**

```
#include<
stdio.h>
#include<
conio.h>
main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

clrscr();
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);

printf("\n Enter the reference string -- ");
");for(i=0;i<n;i++)scanf("%d",&rs[i]);
printf("\n Enter no. of
frames --
");scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;

printf("\n The Page Replacement Process is -- \n");for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
i
f
(
m
[
k
]
=
=
r
s
```

```

[
i
]
)

b
r
e
a
k
;
}

if(k==f)
{
m[count++]=rs[i];pf++;
}

for(j=0;j<f;
j++)
printf("\t%d",m[j]);if(
k==f)
printf("\tP
F No.
%d",pf);
printf("\n");
;if(count==
f) count=0;
}

printf("\n The number of Page Faults using FIFO are %d",pf);getch();
}

```

### **OUTPUT:**

Enter the length of reference string – 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter no. of frames -- 3

The Page Replacement Process is –

7	-1	-1	PF No. 1
7	0	-1	PF No. 2
7	0	1	PF No. 3
2	0	1	PF No. 4
2	0	1	
2	3	1	PF No. 5
2	3	0	PF No. 6
4	3	0	PF No. 7
4	2	0	PF No. 8
4	2	3	PF No. 9
0	2	3	PF No. 10
0	2	3	
0	2	3	
0	1	3	PF No. 11
0	1	2	PF No. 12
0	1	2	
0	1	2	
7	1	2	PF No. 13
7	0	2	PF No. 14
7	0	1	PF No. 15

The number of Page Faults using FIFO are 15

### **RESULT:**

Thus a UNIX C program to implement FIFO page replacement is executed successfully.

### **AIM:**

To write UNIX C program a program to implement LRU page replacement algorithm.

**DESCRIPTION:**

The Least Recently Used replacement policy chooses to replace the page which has not been referenced for the longest time. This policy assumes the recent past will approximate the immediate future. The operating system keeps track of when each page was referenced by recording the time of reference or by maintaining a stack of references.

**ALGORITHM:**

- Start the process
- Declare the size
- Get the number of pages to be inserted
- Get the value
- Declare counter and stack
- Select the least recently used page by counter value
- Stack them according the selection.
- Display the values
- Stop the process

**PROGRAM:**

```
#include<
stdio.h>
#include<
conio.h>
main()
{
    int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0,
next=1;clrscr();printf("Enter the length of reference string -- ");

    scanf("%d",&n);
    printf("Enter the reference string -- ");for(i=0;i<n;i++)
    {
        scanf("%d",&rs[i]); flag[i]=0;
```

```
}

printf("Enter the number of frames --
");scanf("%d",&f);for(i=0;i<f;i++)
{
count[i]=0;m[i]=-1;
}

printf("\nThe Page Replacement process is -- \n");for(i=0;i<n;i++)
{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1; count[j]=next;next++;
}
}
}

if(flag[i]==0)
{
if(i<f)
{
    m
    [
    i
    ]
    =
    r
    s
    [
    i
    ]
    ;
}
}
```

c

```
    o
    u
    n
    t
    [
    i
    ]
    =
    n
    e
    x
    t
    ;
    n
    e
    x
    t
    +
    +
    ;
}

else
{
}

pf++;
```

}

```

min=0; for(j=1;j<f;j+
+)

if(count[min] > count[j])min=j;m[min]=rs[i];
count[min]=next;
next++;

for(j=0;j<f;j++)
printf("%d\t", m[j]);if(flag[i]==0)

printf("PF No. --
%d", pf);
printf("\n");
}

printf("\nThe number of page faults using LRU are %d",pf);getch();
}

```

### **OUTPUT:**

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

0 1 7 0 1Enter the number of frames -- 3

The Page Replacement process is --

7	-1	-1	PF No. -- 1
7	0	-1	PF No. -- 2
7	0	1	PF No. -- 3
2	0	1	PF No. -- 4
2	0	1	
2	0	3	PF No. -- 5
2	0	3	
4	0	3	PF No. -- 6

4	0	2	PF No. -- 7
4	3	2	PF No. -- 8
0	3	2	PF No. -- 9
0	3	2	
0	3	2	
1	3	2	PF No. -- 10
1	3	2	
1	0	2	PF No. -- 11
1	0	2	
1	0	7	PF No. -- 12
1	0	7	
1	0	7	

The number of page faults using LRU are 12

## **RESULT:**

Thus a UNIX C program to implement LRU page replacement is executed successfully.

## **AIM:**

To write a program in C to implement LFU page replacement algorithm.

## **ALGORITHM**

Step1: Start the program

Step2: Declare the required variables and initialize it.

Step3; Get the frame size and reference string

from the userStep4: Keep track of entered data

elements

Step5: Accommodate a new element look for the element that is not to be used infrequently replace.

Step6: Count the number of page fault and

display the valueStep7: Terminate the program

## **PROGRAM**

```
#include
<stdio.h>
#include
<conio.h>
main()
{
    int rs[50], i, j, k, m, f, cntr[20], a[20], min,
    pf=0;clrscr(); printf("\nEnter number of page
references -- ");scanf("%d",&m);printf("\nEnter
the reference string -- ");for(i=0;i<m;i++)
scanf("%d",&rs[i]);
printf("\nEnter the available no. of frames --
");scanf("%d",&f);for(i=0;i<f;i++)
{
    cntr[i]=0;a[i]=-1;
}
Printf("\nThe Page Replacement Process is
- \n");for(i=0;i<m;i++)
```

{

f

o

r

(

j

=

0

;

j

<

f

;

j

+

+

)

i

f

(

r

s

[

i

]

=

=

a

[

j

]

)

{

$\min = 0;$

}

if(j==f)

{

```

cntr[j]++;break;

for(k=1;k<f;k++)
if(cntr[k]<cntr[min])mi
n=k;
a[min]=rs[i];cntr[min]=
1;pf++;
}
printf("\n"); for(j=0;j<f;j++)

printf("\t%d",a[j
]);if(j==f)
printf("\tPF No.
%d",pf);

}

printf("\n\n Total number of page faults -- %d",pf);getch();
}

```

### **OUTPUT:**

Enter number of page references -- 10

Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3

Enter the available no. of frames 3

The Page Replacement Process is –

1	-1	-1	PF No. 1
1	2	-1	PF No. 2
1	2	3	PF No. 3
4	2	3	PF No. 4
5	2	3	PF No. 5
5	2	3	

5	2	3	
5	2	1	PF No. 6
5	2	4	PF No. 7
5	2	3	PF No. 8

Total number of page faults----- 8

## **RESULT:**

Thus the C programs to implement LFU page replacement algorithm wasexecuted successfully.

<b>EX NO: 13</b>	<b>IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUES</b>
<b>DATE:</b>	

- SINGLE LEVEL DIRECTORY:**

**AIM:**

Program to simulate Single level directory file organization technique.

**DESCRIPTION:**

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub

directories. Advantage of single level directory system is that it is easy to find a file in the directory.

### **PROGRAM:**

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

void main()
{
int      i,ch;  charf[30]; clrscr(); dir.fcnt
= 0;
printf("\nEnter      name  of  directory  --  ");scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
4. Display Files\t5.
Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the
name of the file -- ");
scanf("%s",dir.fname[dir.fcn
t]);
dir.fcnt++; break;
case 2: printf("\nEnter the
name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f); strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break;
}
if(i==dir.fcnt)
```

```
printf("File %s not found",f);else
```

```
dir.fcnt
--;
break;

case 3:           printf("\nEnter the
name of the file -- ");scanf("%s",f); for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);break;
case 4:           if(dir.fcnt==0)
printf("\nDirectory Empty");else
{
printf("\nThe Files are --
");for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;

}
getch();}
```

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

**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 -- A  
1. Create File 2. Delete File 3. Search File  
4. Display Files 5. Exit Enter your choice – 1

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

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

The Files are -- A B C  
1. Create File 2. Delete File 3. Search File  
4. Display Files 5. Exit Enter your choice – 3

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

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

• **TWO LEVEL DIRECTORY**

**AIM:** Program to simulate two level file organization technique

**Description:**

In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block

contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched.

### **PROGRAM:**

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

void main()
{
int          i,ch,dcnt,k;  char
f[30],
d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3.
Delete File"); printf("\n4. Search File\t\t5.
Display\t6. Exit\t Enter your choice --");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter name of directory -- ");
scanf("%s",
d
i
r
[
d
c
n
t
]
.
f
c
n
t
        dir[dcnt].dname);}
```

```

=
0
;

d
c
n
t
+
+
;
printf("Directory created"); break;
case 2: printf("\nEnter name of
the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");scanf("%s"
dir[i].fcnt++; printf("File created");
}
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++)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].f
cnt]);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
Director
y's ");
else
{
printf("\nDirectory\tFiles");for(i=0;i<dcnt;i++)
{

```

```
 }  
 }  
break;
```

```

        printf("\n%s\t\t",dir[i].dname);
        for(k=0;k<dir[i].fcnt;k++)
            printf("\t%s",dir[i].fname[k]);

    default:exit(0);
}
}
}

```

**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 -- DIR1 Directory created

1. Create Directory 2. Create File 3. Delete File  
 4. Search File 5. Display 6. Exit Enter  
 your choice -- 1Enter name of  
 directory -- DIR2 Directory created  
 1. Create Directory 2. Create File 3. Delete File  
 4. Search File 5. Display 6. Exit Enter  
 your choice -- 2Enter name of the  
 directory – DIR1  
 Enter name of  
 the file --  
 A1Filecreated  
 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 – DIR1

Enter name of  
 the file --  
 A2Filecreated  
 1. Create Directory 2. Create File 3. Delete File

4. Search File 5.

Display 6.Exit

Enter your

choice – 6

### **RESULT:**

Thus the implementation of file organization techniques executed successfully

### **AIM:**

To implement sequential file allocation technique.

### **ALGORITHM:**

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order.

a). Randomly select a location from available location s1= random(100); b).

Check whether the required locations are free from the selected location. c).

Allocate and set flag=1 to the allocated locations.

Step 5: Print the results fileno, length ,

Blocks allocated.Step 6: Stop the progr

## **PROGRAM**

```
#include<stdio.h>main()
{
    int
    f[50],i,st,j,len,c,k;clrscr(
    );for(i=0;i<50;i++)
        f
    [
    i
    ]
    =
    0
    ;
}

X
:
printf("\n Enter the starting block & length of file");scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)if(f[j]==0)
{
    f
    [
    j
    ]
    =
    1
    ;
    printf("\n%d->%d",j,f[j]);
}
else
{
    printf("Block already allocated");break;
}
```

```
if(j==(st+len))  
printf("\n the file is allocated to disk");
```

```
printf("\n if u want to enter more files?(y-
```

```
1/n-0)");scanf("%d",&c);
```

```
i
```

```
f
```

```
(
```

```
c
```

```
=
```

```
=
```

```
1
```

```
)
```

```
g
```

```
o
```

```
t
```

```
o
```

```
X
```

```
;
```

```
getch();
```

```
}
```

else

```
exit();
```

### **OUTPUT:**

Output: Enter the starting block & length of

file 4 104->15->1

6->1

7->1

8->1

9->1

10->1

11->1

12->1

13->1

The file is allocated to disk

If you want to enter more files? (Y-1/N-0)

### **RESULT:**

Thus the program to implement the Sequential file allocation was executed successfully.

**AI**

**M**

:

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

### **ALGOR**

#### **ITHM:**

Step 1: Start the Program

Step 2: Get the number of files.

Step 3: Allocate the required locations by selecting a location randomly

Step 4: Check whether the selected location is free.

Step 5: If the location is free allocate and set flag =1 to the allocated

locations.Step 6: Print the results file no, length, blocks allocated.

Step 7: Stop the execution

**PROGRAM:**

```
#include<stdio.h>main()
{
    int
    f[50],p,i,j,k,a,st,len,n,c;
    clrscr();
    for(i=0;i<50;i++)
        f[i]=0;

    printf("Enter how many blocks that are already
allocated");scanf("%d",&p);printf("\nEnter the blocks no.s that are
already allocated");for(i=0;i<p;i++)
    {
        scanf("%d",&a);f[a]=1;
    }

    X: printf("Enter the starting index block & length");scanf("%d%d",&st,&len);

    k=len;

    for(j=st;j<(k+st);j++)
    {
        if(f[j]==0)
        {

        }
        else
        {
    }
```

```
f[j]=1;
printf("\n%d->%d",j,f[j]);

printf("\n %d->file is already allocated",j); k++;

}

}

printf("\n If u want to enter one more file? (yes-1/no-0)");

sca
nf(
%d
",&
c);
if(c
==
1)
goto X;

getch( );
}
```

else

```
exit();
```

**OUTPUT:**

Enter how many blocks are already  
allocated 3 Enter theblocks no's that are  
already allocated 4 7 9Enter the starting  
index block & length 3 7

3-> 1

4-> File is already allocated 5->1

6->1

7-> File is already allocated 8->1

9-> File is already allocated 10->1

11->1

12->1

If u want to enter one more file? (yes-1/no-0)

## **RESULT:**

Thus the program to implement the linked file allocation was executed successfully

## **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: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly.

Step 5: Print the results file no, length,  
blocks allocated. Step 6: Stop the  
execution.

## **PROGRAM**

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

```
int
f[50],i,k,j,inde[50],
n,c,count=0,p;
main()
{
clrscr();
for(i=0;i<5
0;i++)
f[i]=0;
x: printf("enter index
block\t");scanf("%d",&p);if(f[p]==0)
{
    f[p]=1;
    printf("enter no of files on index\t"); scanf("%d",&n);

}
else
{
```

```
printf("Block already allocated\n"); goto x;

}

for(i=0;i<n;i++)
scanf("%d",&inde[i]);fo
r(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("Block already allocated");goto x;
}
for(j=0;j<n;j++)

f[inde[j]]=1;
printf("\n
allocated");
printf("\n file
indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]); printf("Enter 1 to enter more files and 0 to exit\t"); s
scanf("%d",&c);
i
f
(
c
=
=
1
)

g
o
t
```

o

x

;

getch();

}

else

```
exit();
```

**OUTPUT:**

Enter index block 9

Enter no of files on

index 3 1 2 3Allocated

File indexed 9-> 1:1

9-> 2:1

9->3:1

Enter 1 to enter more files and 0 to exit.

**RESULT :**

Thus the program to implement the indexed file allocation was executed successfully

**AIM:**

Write a C program to simulate disk scheduling algorithms

- a) FCFS
- b) SCAN
- c) C-SCAN

**DESCRIPTION**

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.

When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip

## PROGRAM

### FCFS DISK SCHEDULING ALGORITHM

```
#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];a
    vhm=(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:**

Enter no.of tracks:9

Enter track position:55      58      60      70      18      90      150      160      184

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

## SCAN DISK SCHEDULING ALGORITHM

```
#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;  
}  
}  
}  
  
f  
o  
r  
(  
i  
=  
0  
;  
i  
<
```

n

+

2

;

i

+

+

)

i

f

(

t

[

i

]

=

=

h

)

j

=

i

;

k

=

i

;

p

=

```
0  
;  
while(t[j]!=0)
```

```
{
```

```
a
```

```
t
```

```
r
```

```
[
```

```
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

]

=

a

tr

[j

]

-

a

tr

[j

+

1

];
e

l

s

e

d

[j

]

=

a

tr

[j

```

+
1
]
-
a
tr
[j
];
s
u
m
+
=
d
[j
];
}

printf("\nAverage header movements:%f",(float)sum/n);getch();
}

```

**OUTPUT:**

Enter no.of tracks:9

Enter track position:55 58 60 70 18 90 150 160 184

Tracks traversed	Difference between tracks
------------------	---------------------------

150	50
160	10
1841	24
90	94
70	20
60	10

58	2
55	3
18	37

Average header movements: 27.77

### C-SCAN DISK SCHEDULING ALGORITHM

```
#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]=temp;  
}  
}
```

fo

r(i

=

0;

i<

=

n

+

2;

i+

+)\p

if(

t[i

]=

=

h)

j=

i;

br

ea

k;

p=0;

```
while(t[j]!=tot-1)
```

```
{
```

```
a
```

```
t
```

```
r
```

```
[
```

```
p
```

```
]
```

```
=
```

```
t
```

```
[
```

```
j
```

```
]
```

```
;
```

```
j
```

```
+
```

```
+
```

```
;
```

```
p
```

```
+
```

```
+
```

```
;
```

```
}
```

```
a
```

```
t
```

```
r
```

```
[
```

```
p
```

```
]
```

```
=
```

```
t
[
j
]
;
p
+
+
;
i
=
0
;
while(p!=(n+3) && t[i]!=t[h])
{
    a
    t
    r
    [
    p
    ]
    =
    t
    [
    i
    ]
    ;
    i
    +
    +
```

;

p

+

+

;

}

for(j=0;j<n+2;j++)

{

if(atr[j]>atr[j+1])

d

[j

]

=

a

tr

[j

]

-

a

tr

[j

+

1

];

e

l

s

e

d

[j

```
]
=
a
tr
[j
+
1
]
-
a
tr
[j
];
s
u
m
+
=
d
[j
];
}

printf("total header movements%d",sum);

printf("avg is %f",(float)sum/n);getch();

}
```

**OUTPUT:**

Enter the track position: 55                    58        60        70        18        90        150      160      184

Enter starting position : 100

Tracks traversed      Difference Between tracks

150                    50

160	10
184	24
18	240
55	37
58	3
60	2
70	10
90	29

Average seek time : 35.7777779

### **RESULT:**

Thus the program to implement disk Scheduling algorithm has been executed and verified

<b>EX.NO: 16</b>	<b>INSTALLATION OF UBUNTU OPERATING SYSTEM USING VMWARE</b>
<b>DATE:</b>	

### **Ubuntu:**

[Ubuntu](#) is one of the popular Linux distributions developed on the [Debian Linux](#) platform. Ubuntu is a full operating system available in multiple editions: Desktop edition, Servers edition, IoT, and Cloud. These editions can run on either a computer or virtual machines.

### What Is VMWare Workstation?

VMWare Workstation is an application developed by VMWare to create virtual machines, containers, and Kubernetes clusters on any desktop or server system.

VMWare released VMWare Workstation in two products: VMWare Workstation Pro and VMWare Workstation Player. VMWare Workstation Player is released on a free license with limited features, whereas VMWare Workstation Pro is an enterprise paid version that has loaded with a lot of features.

### Prerequisites To Install Ubuntu On VMWare Workstation:

To install [Ubuntu](#) Linux on VMWare Workstation you just need a physical machine, VMWare WorkstationPro or Player, and Ubuntu Linux ISO image.

#### 1. A host system with minimum:

- 8 GB of memory
  - A quad core CPU
  - 500 GB of Hard Drive
2. VMWare Workstation Pro or Player application:

1. [Download](#) VMWare Workstation
3. Ubuntu Operating System to install on VMWare Workstation.

1. [Download](#) Ubuntu Linux

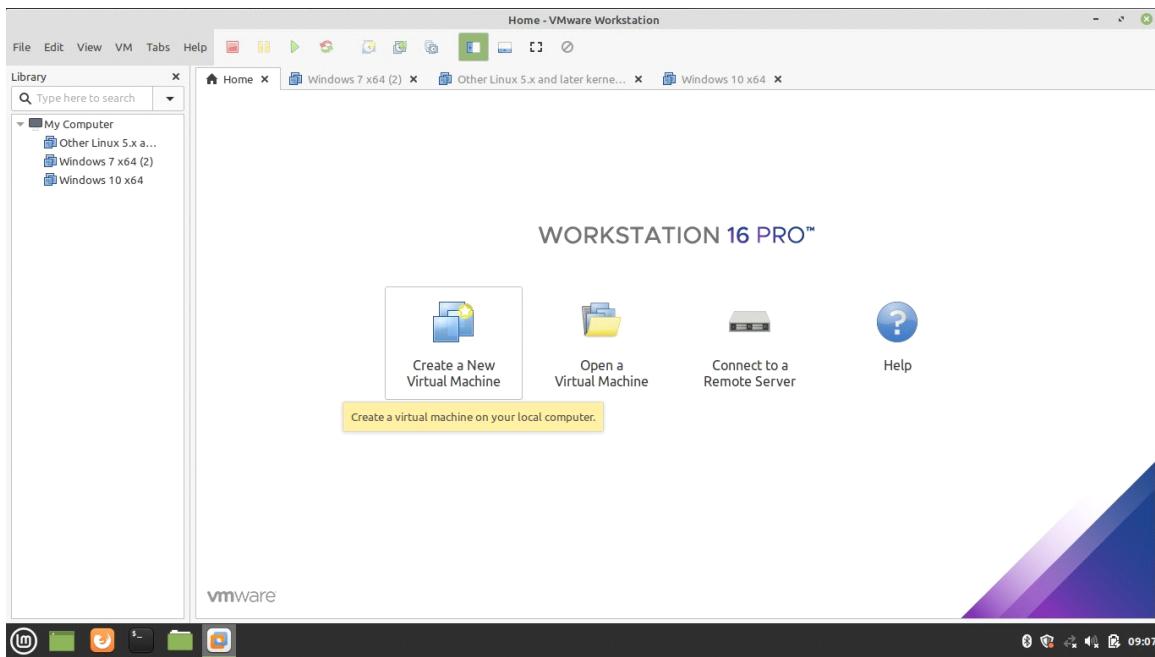
How To Install Ubuntu Linux On  
VMWare Workstation? Time needed:

30 minutes.

### How to Install Ubuntu Linux on VMWare Workstation?

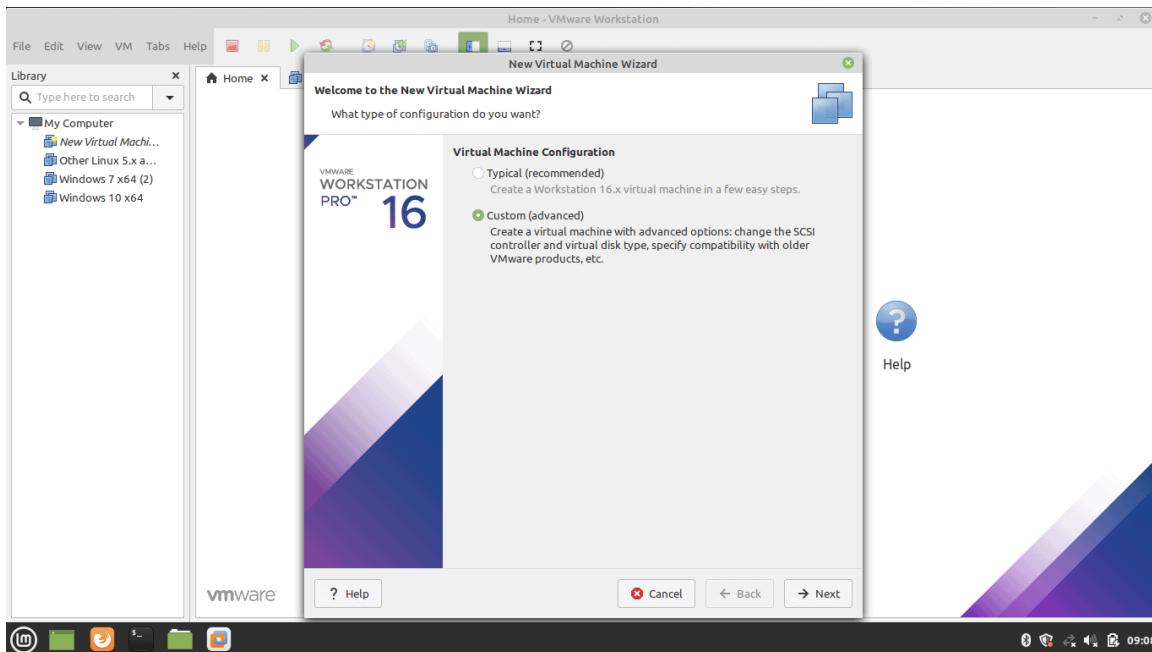
#### • **Fier up VMWare Workstation**

[Download](#) the VMWare Workstation application for your host operating system and install it on your machine. The installation procedure is pretty simple and straight. Read the [documentation](#) for more details. Open the app after installation. Create a new Virtual Machine.



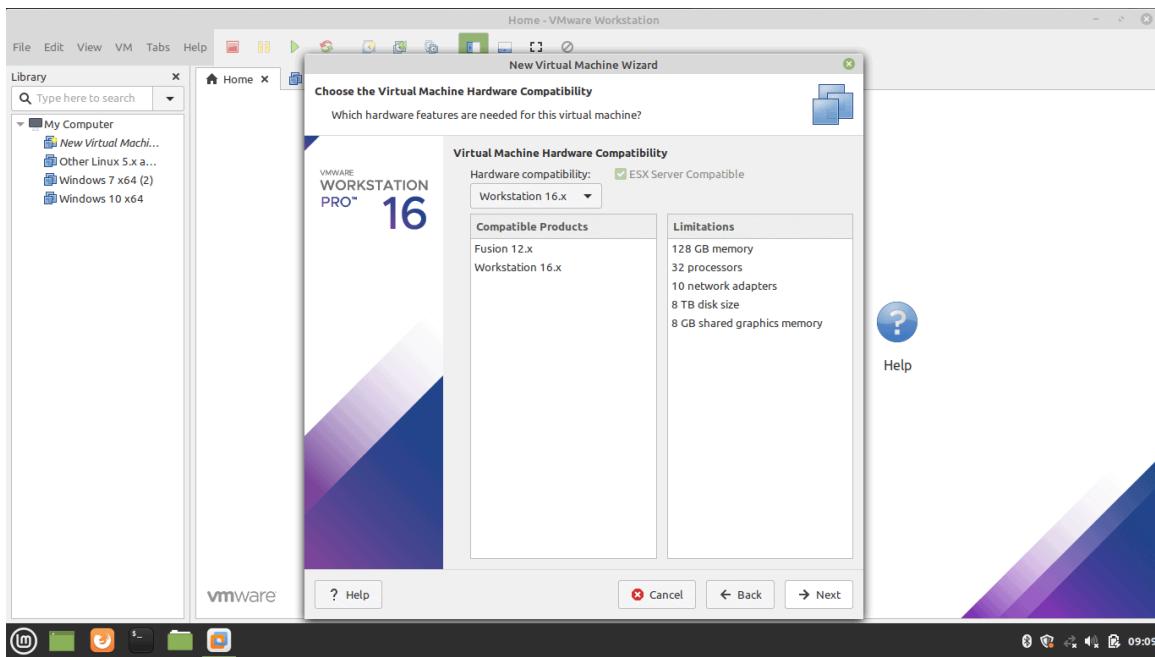
- **Select Custom Configuration Wizard**

You can choose either Typical or Custom Wizard. We recommend selecting Custom if you want to install with all the configurations. If you are okay with default configurations then go ahead with Typical configurations.



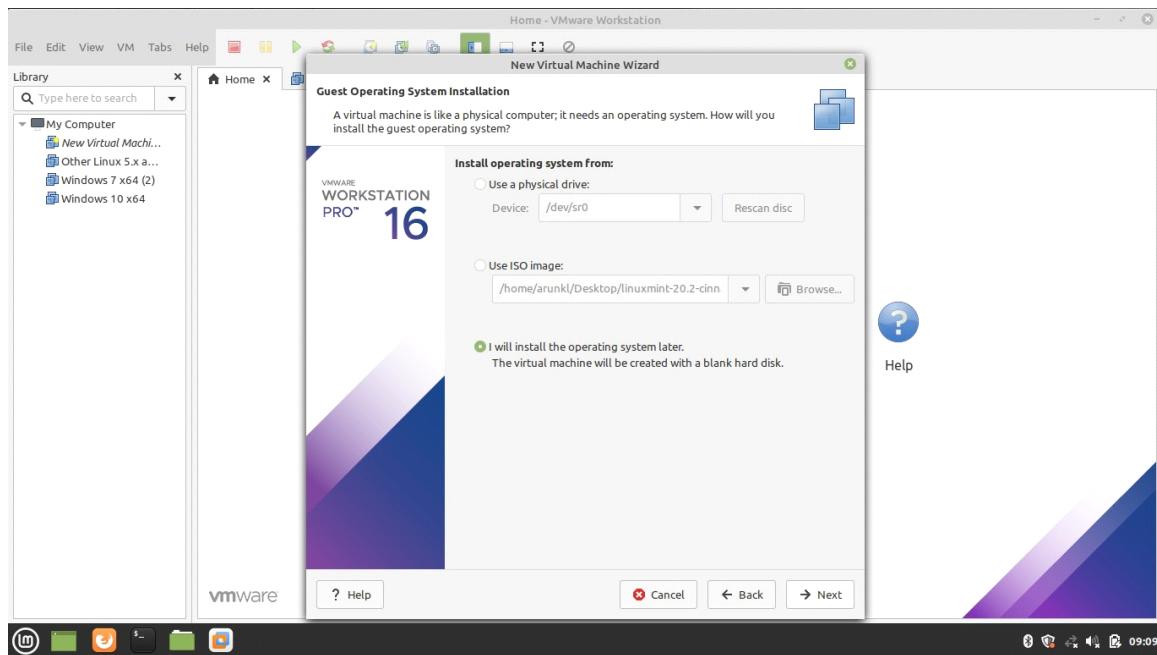
- **Select Virtual Machine Hardware Compatibility**

Go with the default option if you don't have the choice.

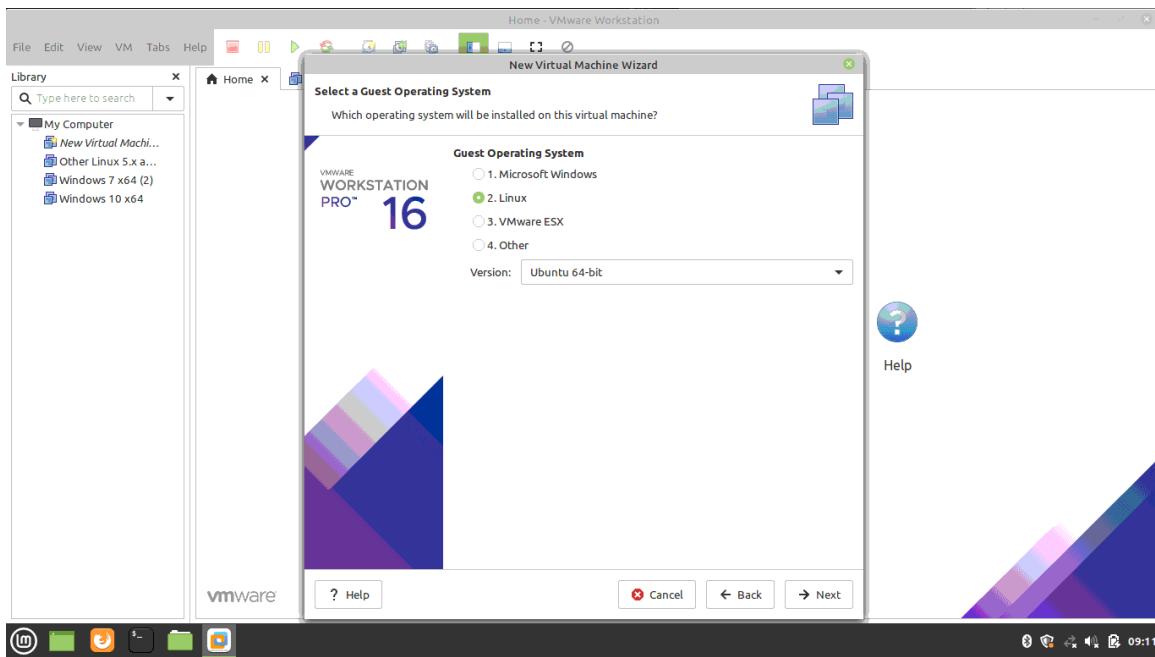


- **Select the Operating System Media**

Select 'I will install the operating system later' for an interactive installation.

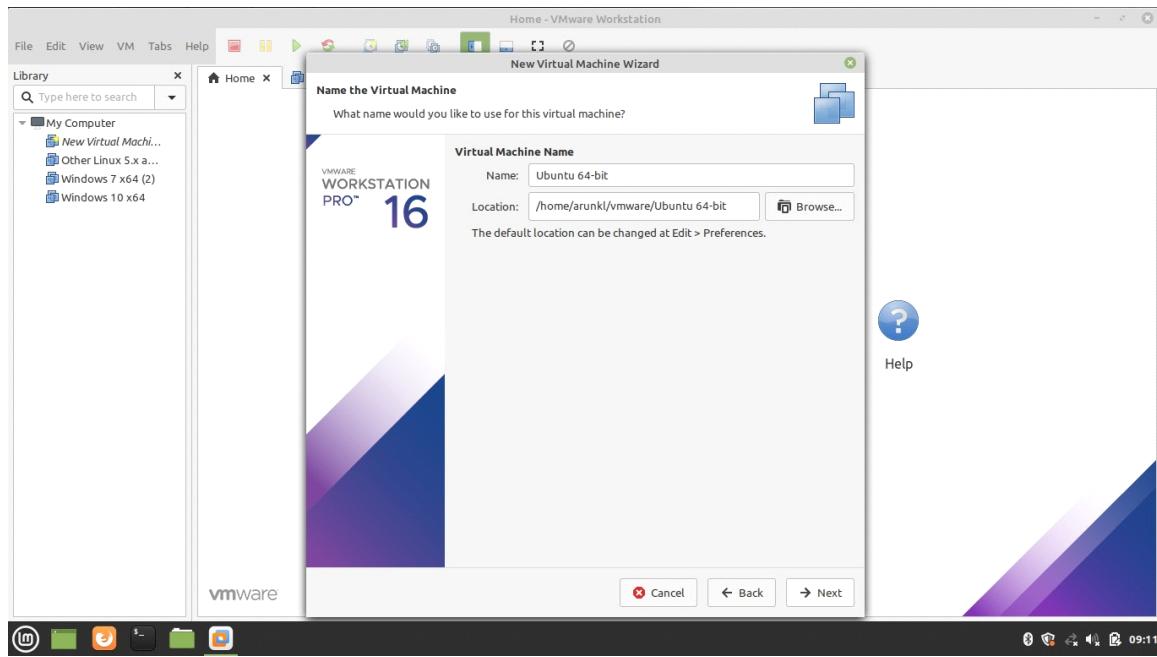


- **Select Guest Operating System**



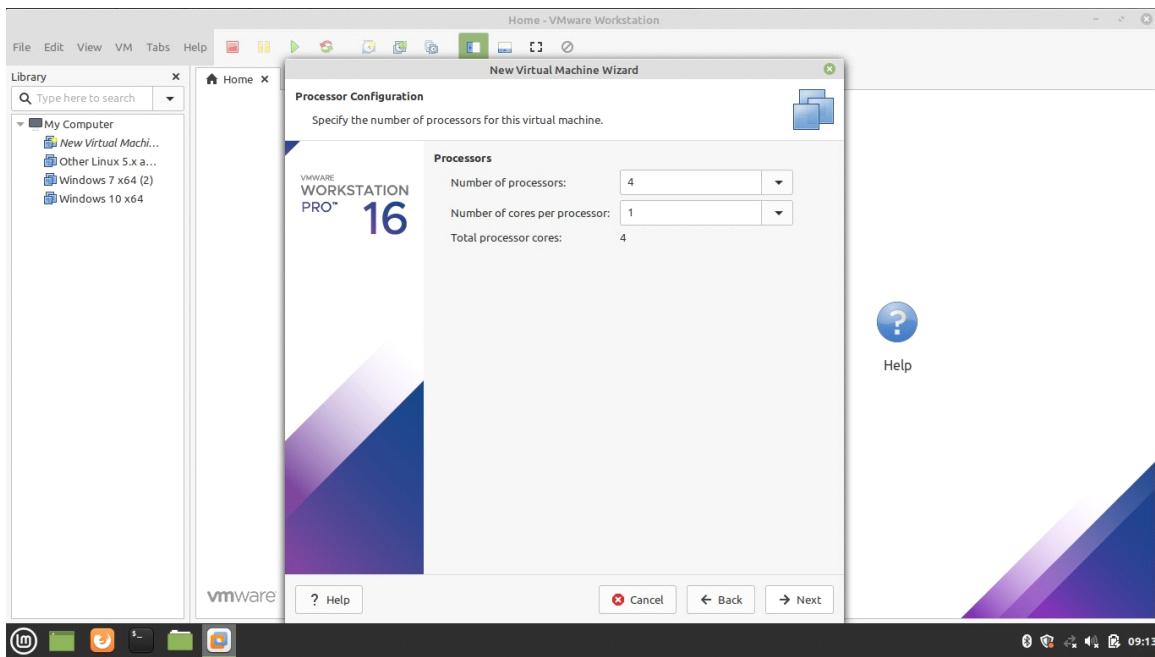
- **Name the Virtual Machine Name and location**

Type a name and give the location details.



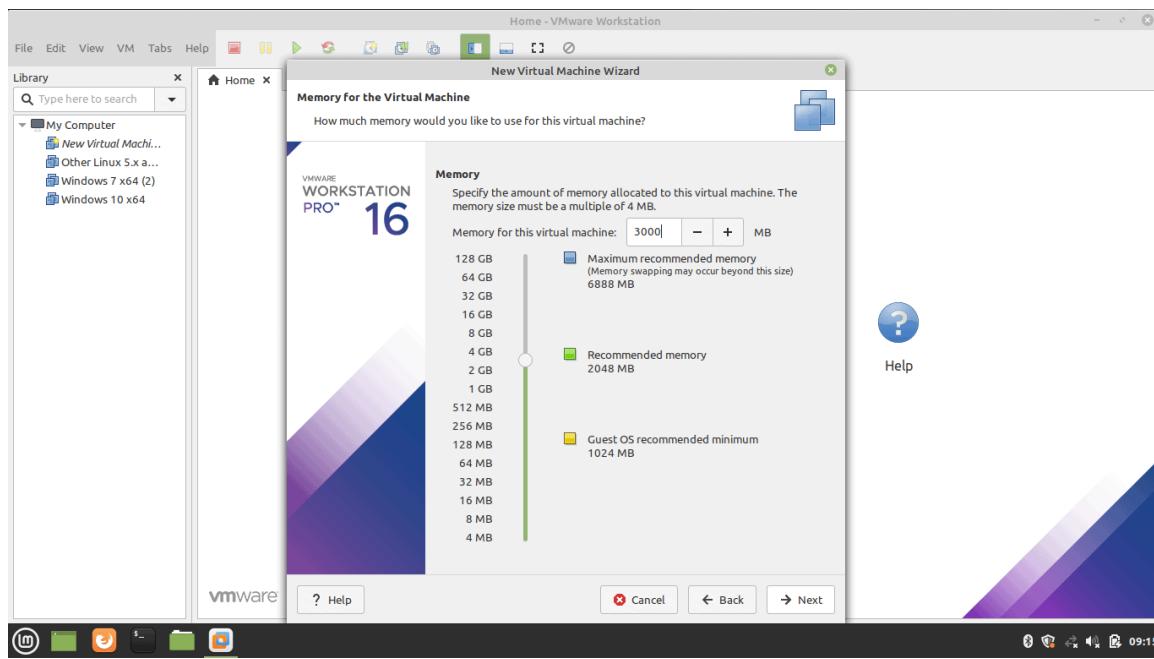
- **Allocate the Processors**

Assign the processors, Calculate the processor required to run the host machine. Assign the leftover resources to the virtual machine.



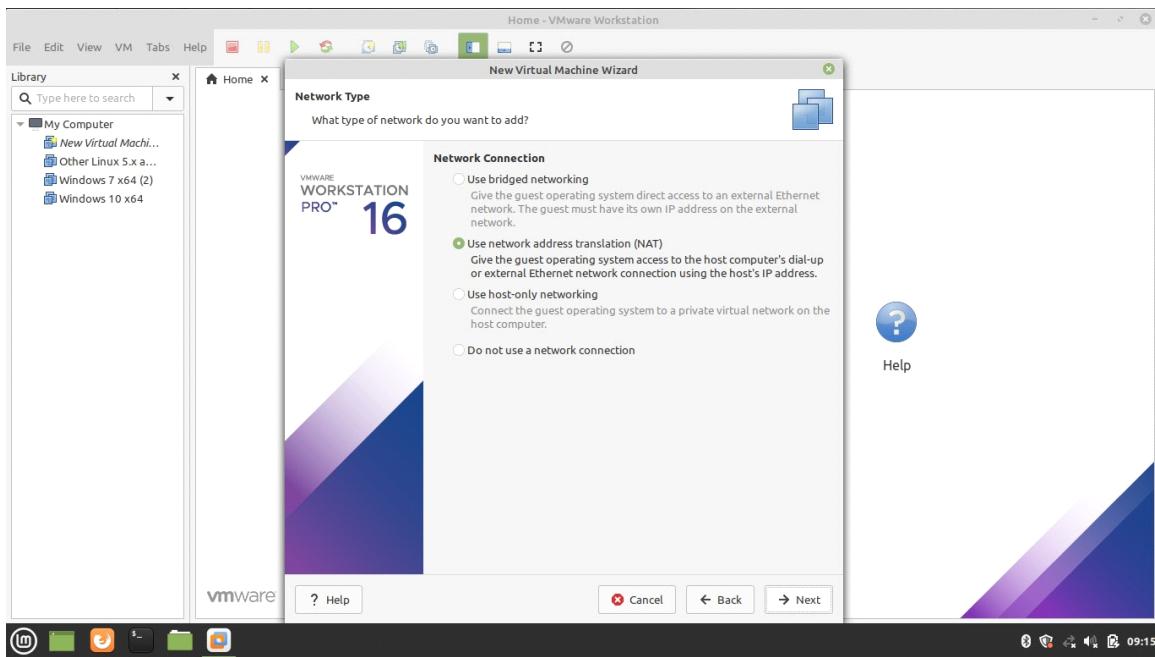
- **Allocate the Memory for Virtual Machine**

Memory allocation calculation is the same as the processor allocation. Leave sufficient memory for the host system and allocate the remaining memory for the virtual machine.

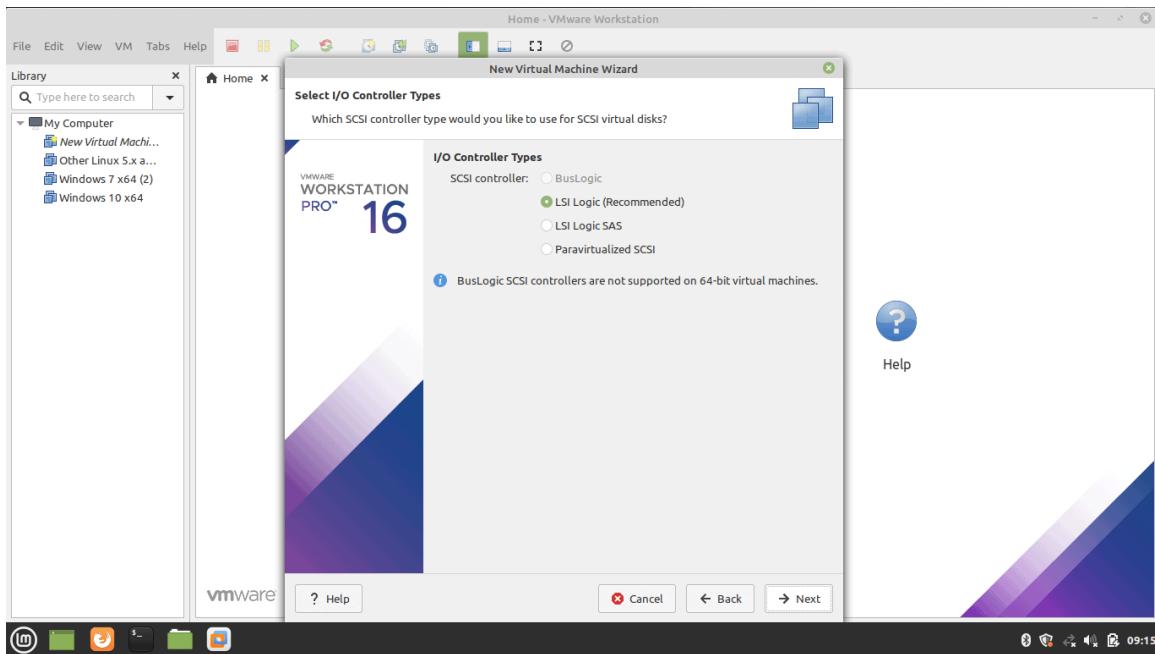


- **Choose the Network Configuration**

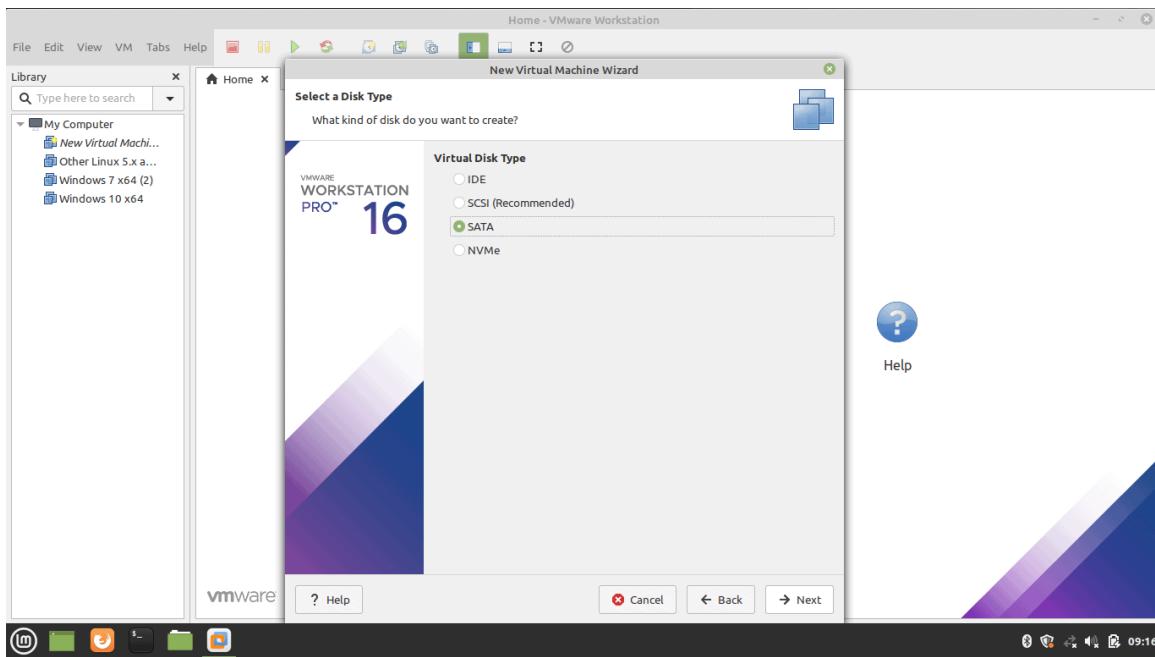
Select any one of the network configurations as per your requirement.



- Select the I/O Controller Type

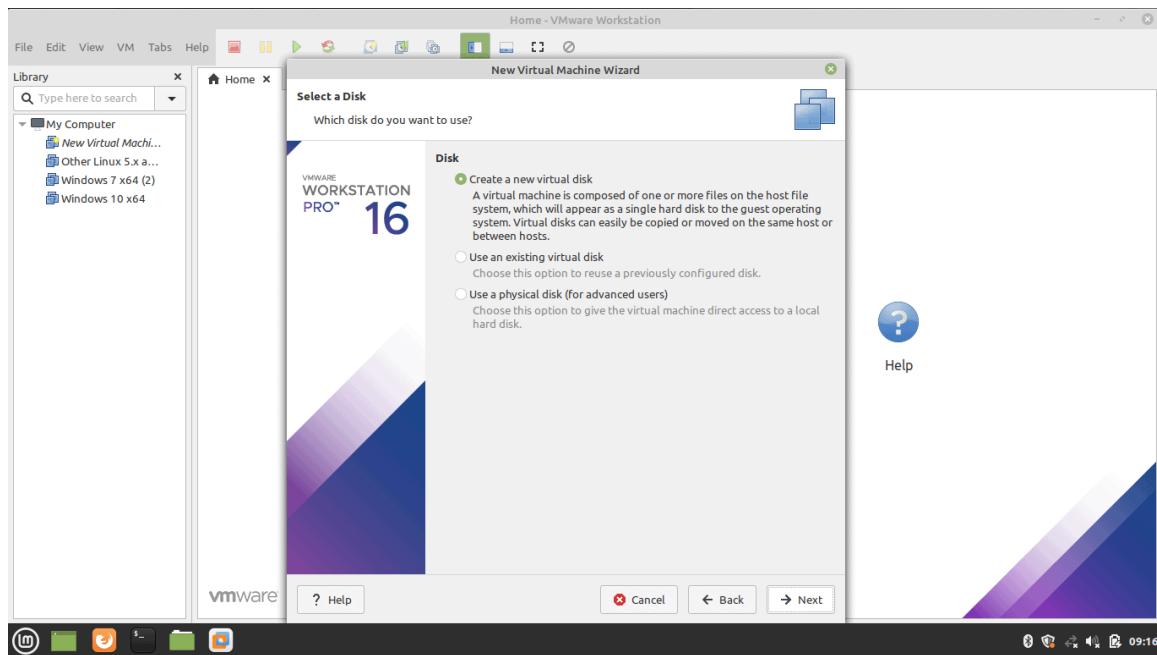


- Select Disk Type



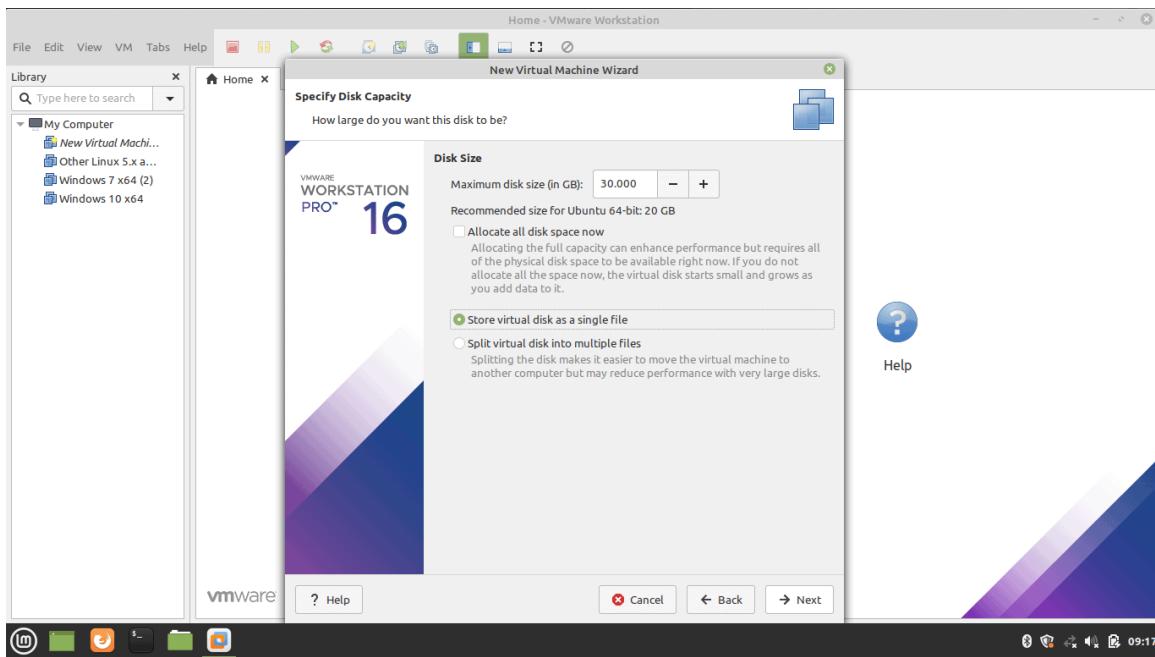
- **Select Virtual Disk**

Select the Virtual Disk if you have or create one.

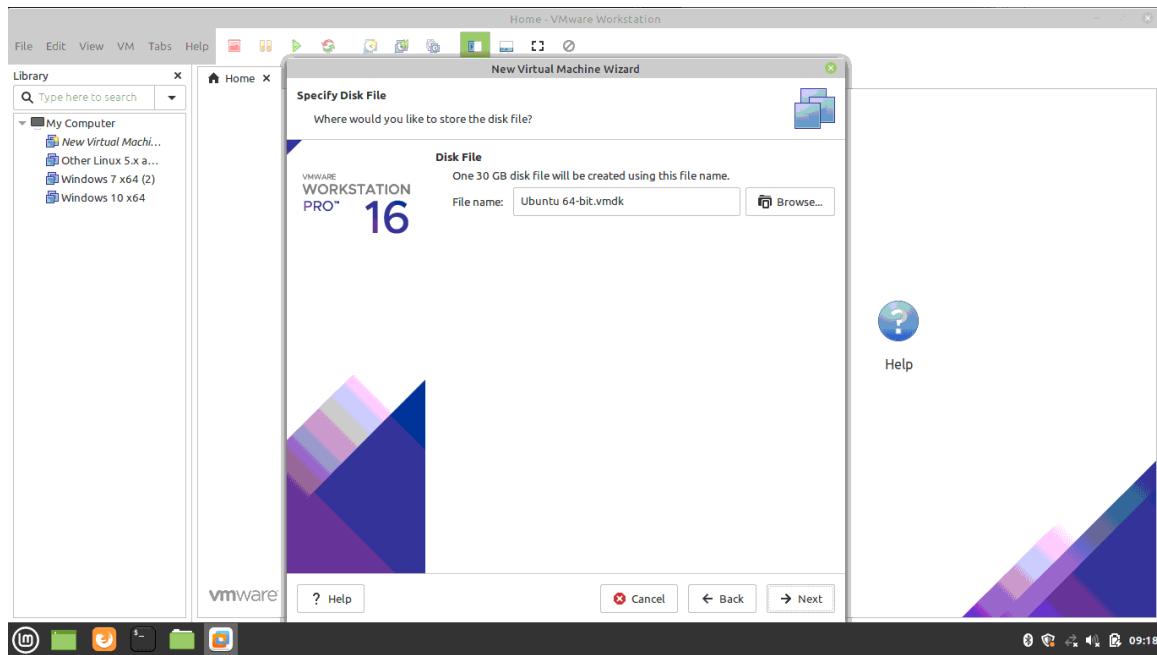


- **Select Disk Capacity**

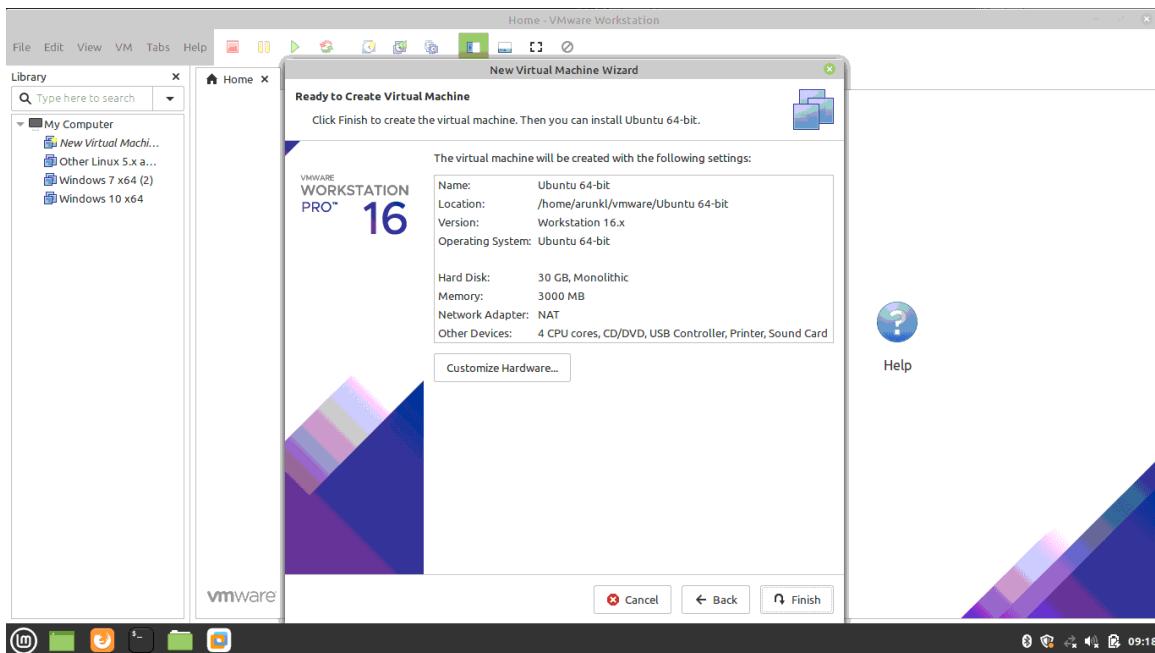
Select the disk size. Selecting a single disk will increase the performance. However, selecting a split disk will help in the disk transfer scenario.



- **Specify Virtual Disk File**

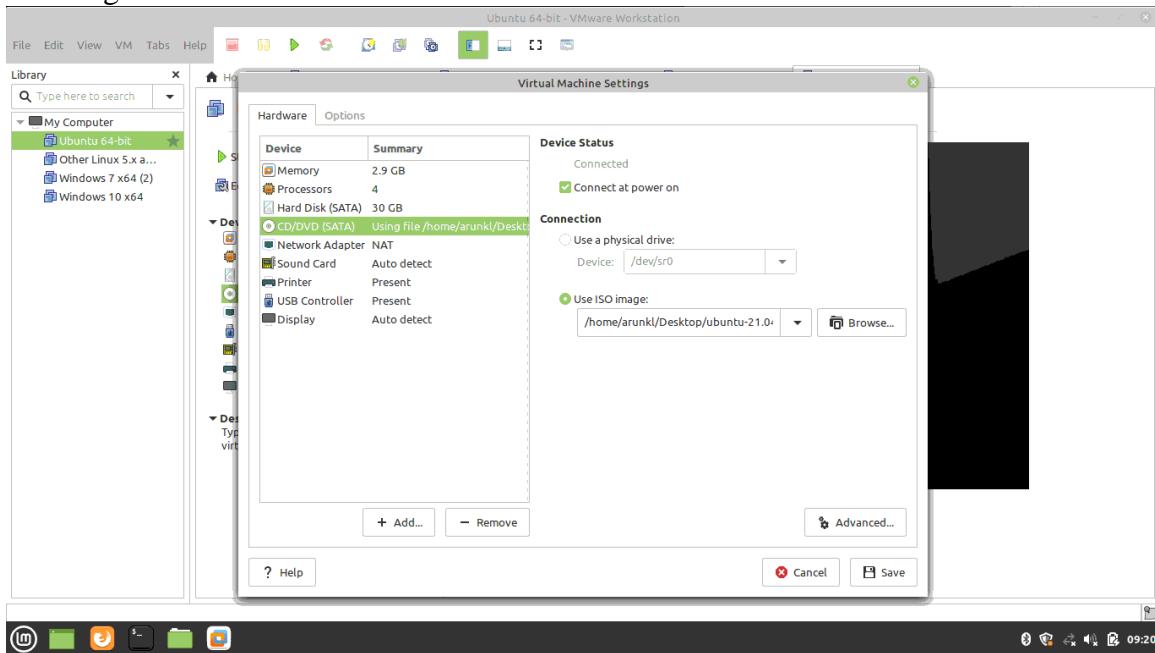


- **Create Virtual Machine**

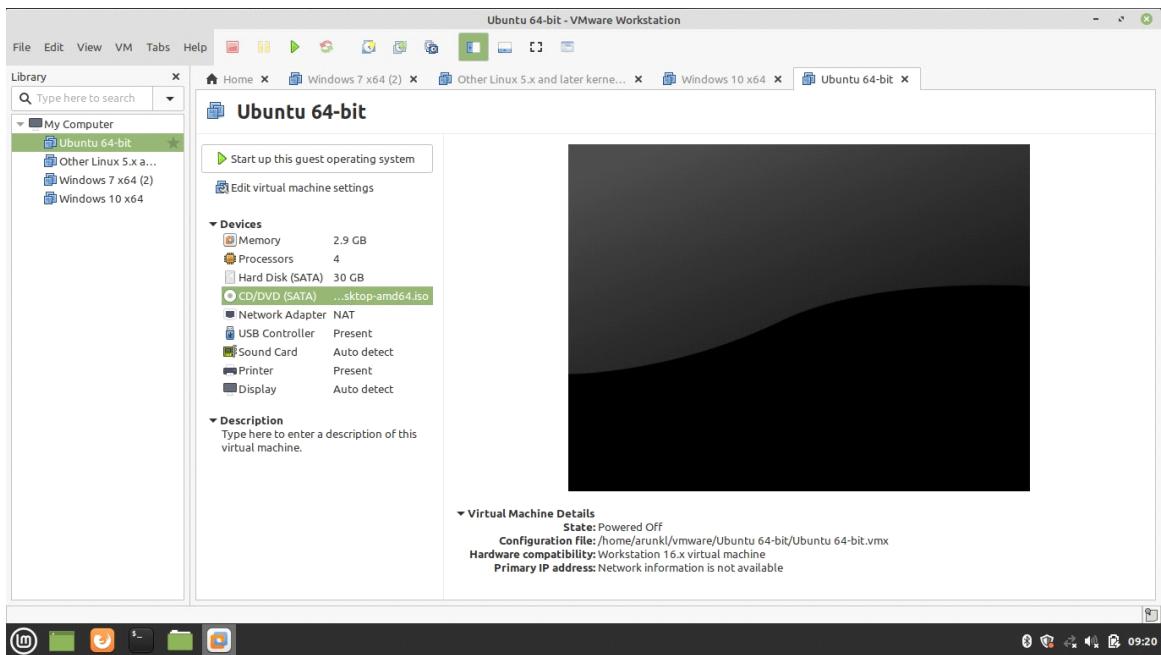


- **Supply Ubuntu ISO Image to Virtual Machine**

Download Ubuntu image. Edit the CD/DVD settings and import the downloaded Ubuntu image.

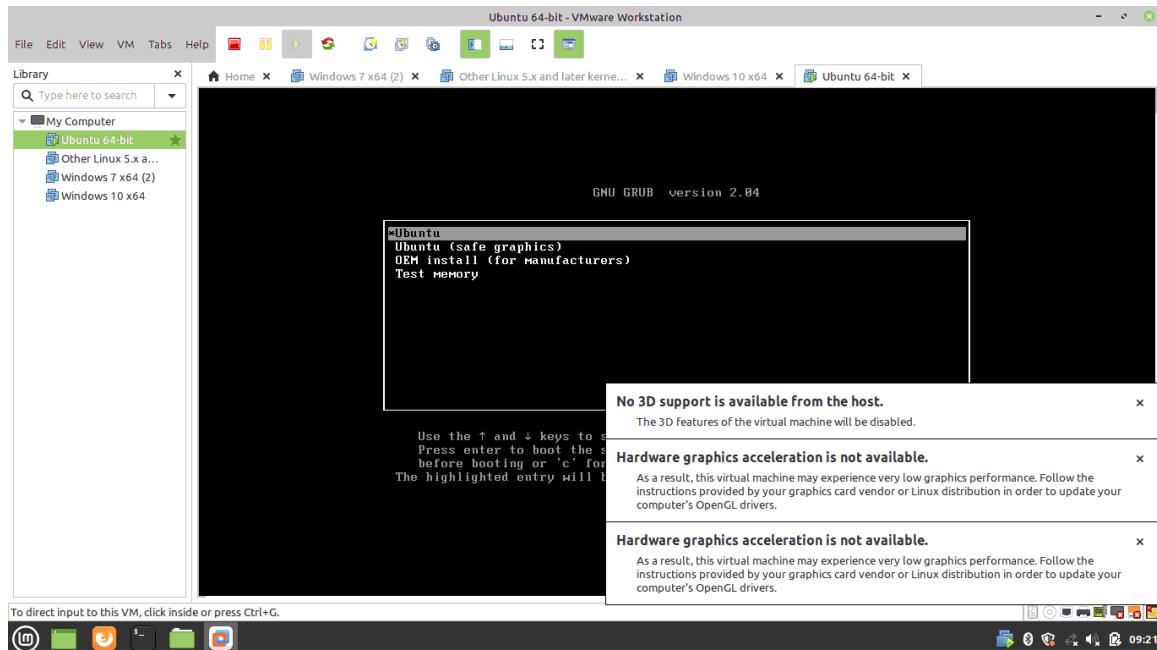


- **Install Ubuntu Linux on VMWare Workstation**



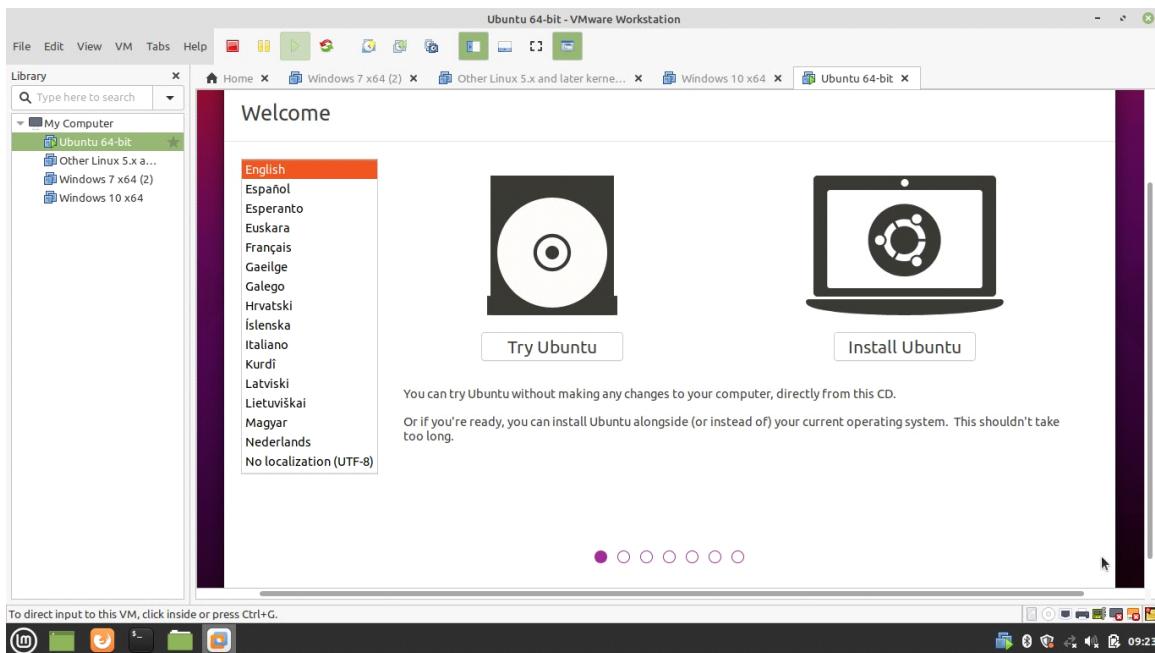
- **Power On the Virtual Machine**

Press the Play button to power on the Virtual Machine.

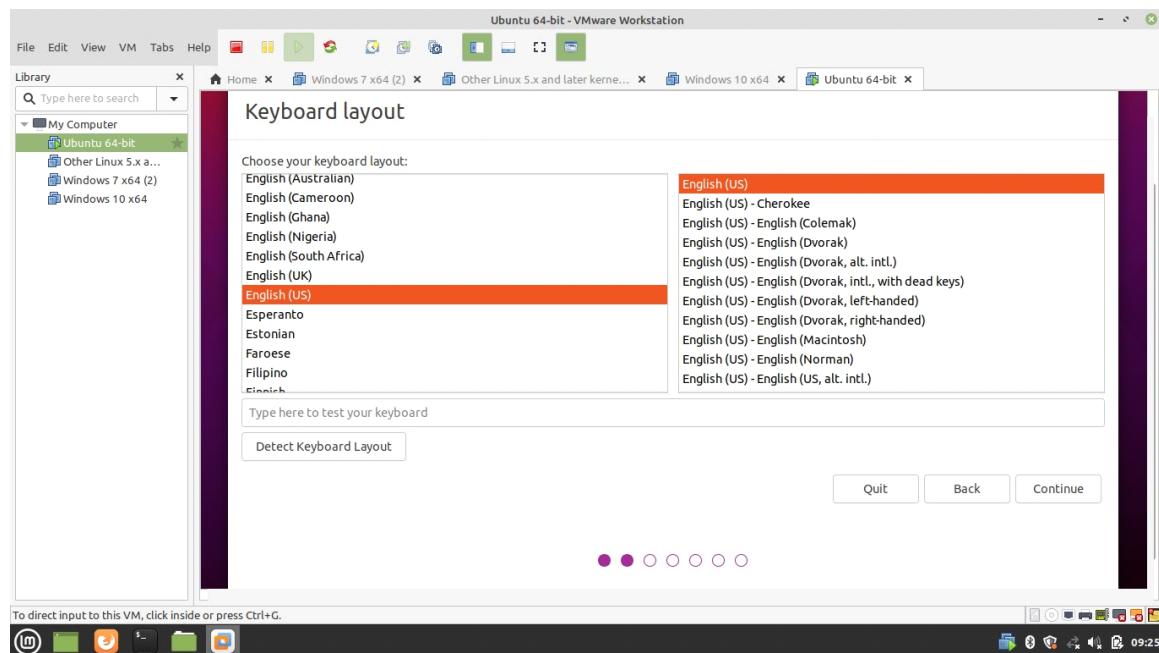


- **Welcome Ubuntu Virtual Machine**

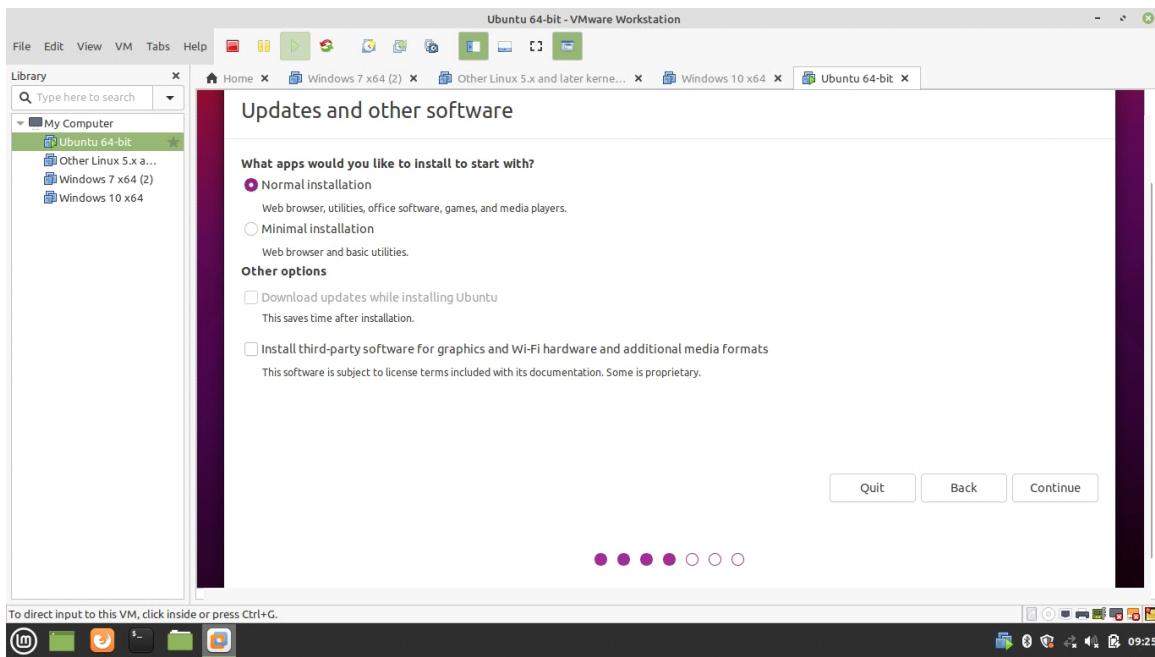
After powering on the Virtual machine, you will be treated with a welcome screen on which you will see two options: Try Ubuntu and Install Ubuntu. Select Try Ubuntu if you want to run Ubuntu in live mode. Select Install to continue the installation process.



- **Select Keyboard Layout**

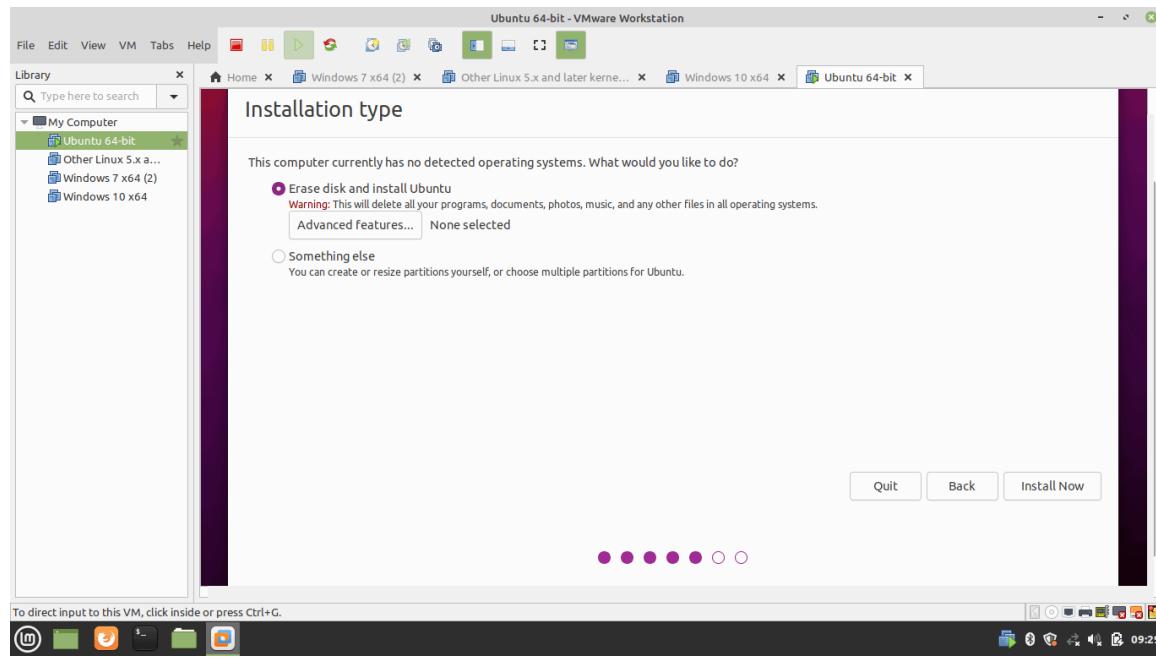


- **Software Update and Package Selection in Virtual Machine**

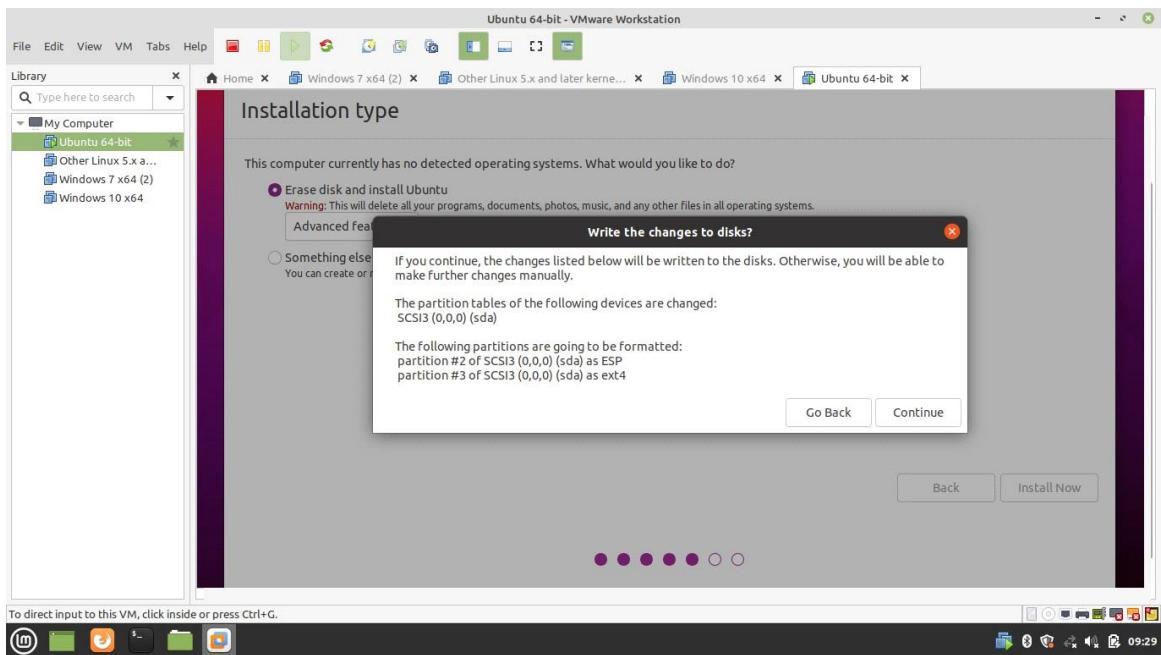


## • Partition the Disk

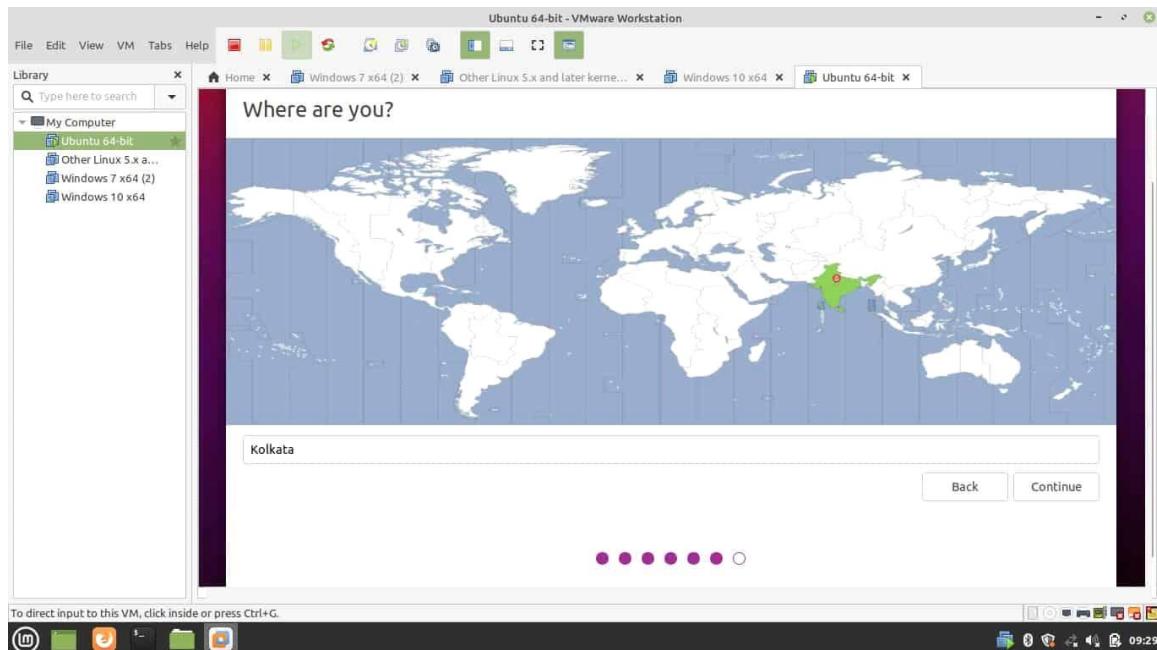
Select Erase the Disk for auto partition. Or Select the Advance option to create the custom partition.



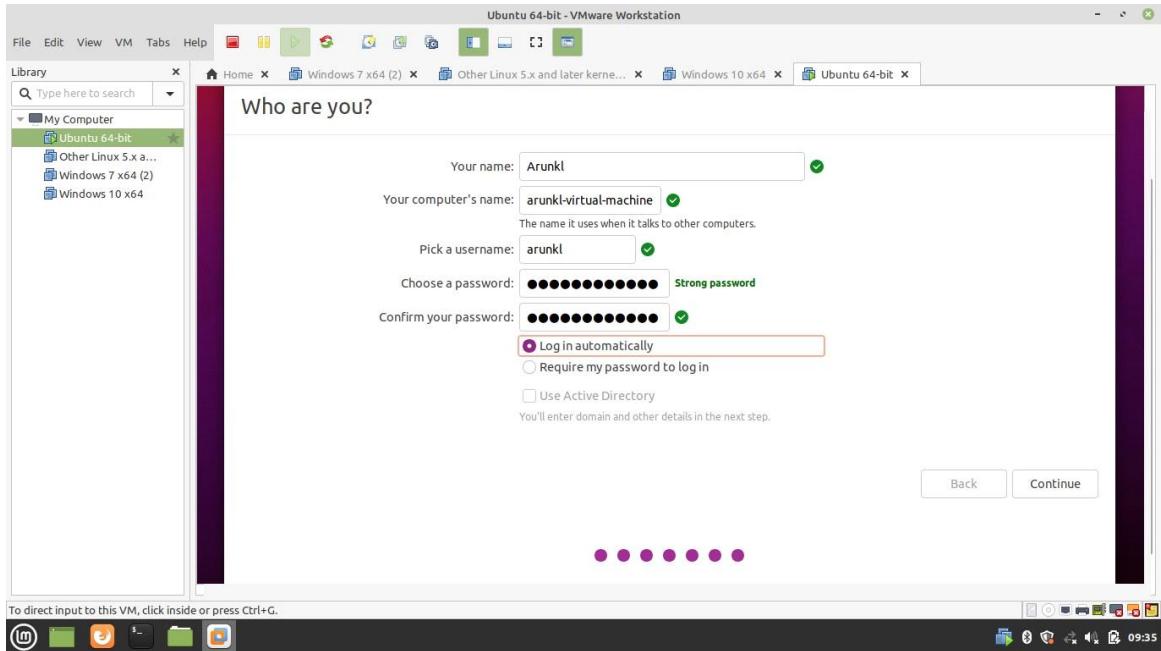
## • Write Changes to Disk



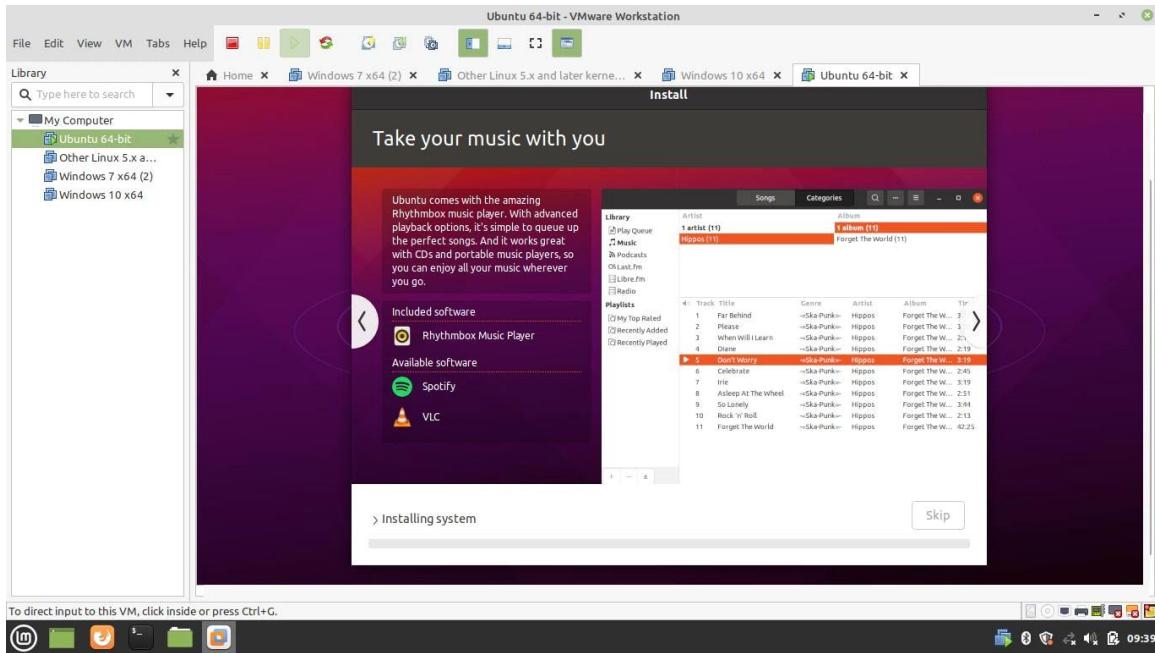
- **Select Time Zone**



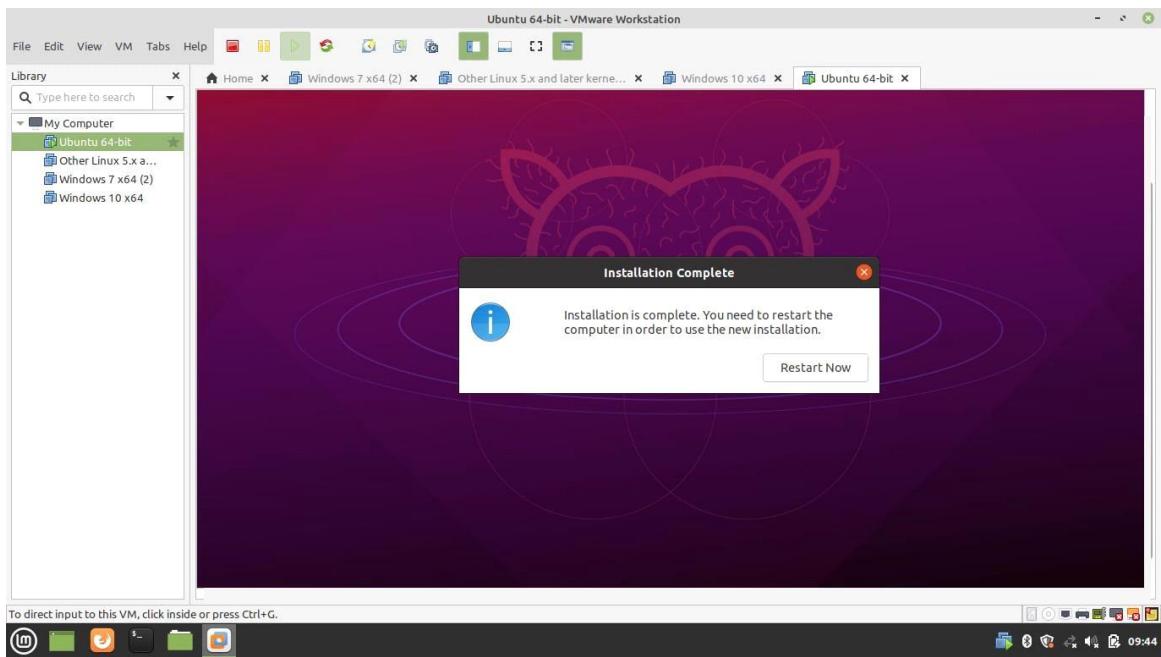
- **Create an Admin Account**



- **Installation of Ubuntu in Progress**

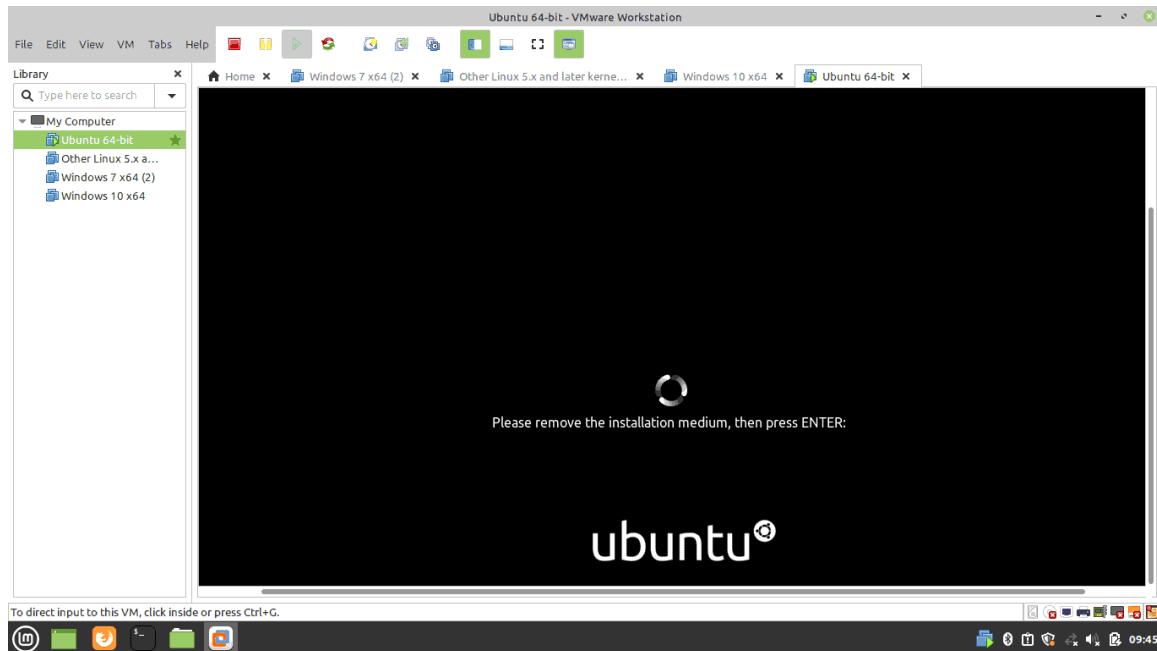


- **Reboot Virtual Machine**
- Reboot the machine after installation.

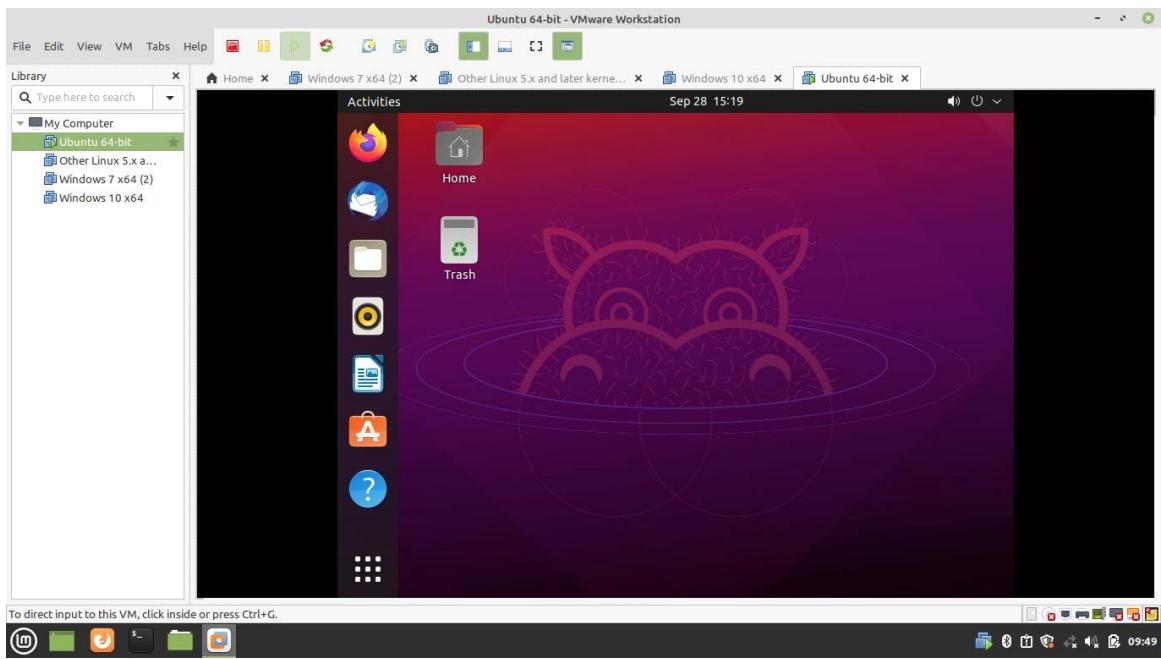


- **Remove the installation media**

Remove the installation media before reboot.



- **Boot Ubuntu**



## RESULT:

Thus the installation of ubuntu operating system using VMware executed successfully.

