EX.NO:1 **INSTALLATION OF WINDOWS OPERATING SYSTEM**

DATE:

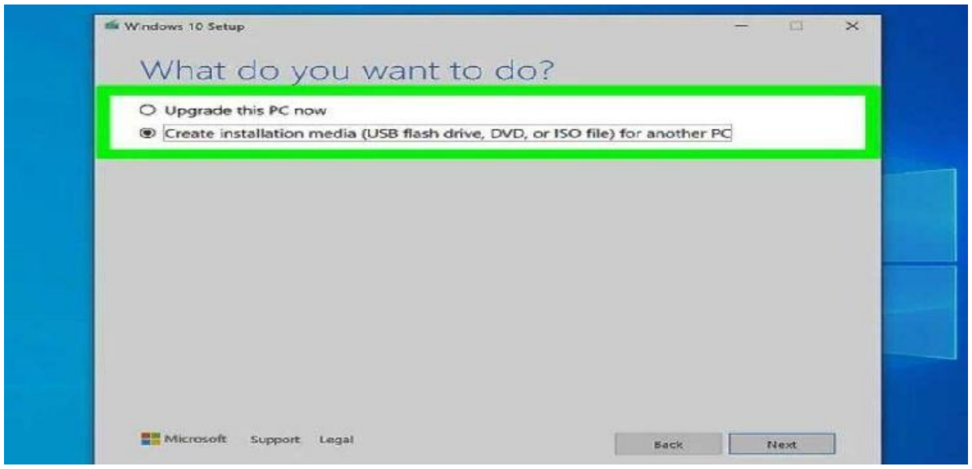
AIM:

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



What do you want to do?

○ Upgrade this PC now

● Create installation media (USB flash drive, DVD, or ISO file) for another PC

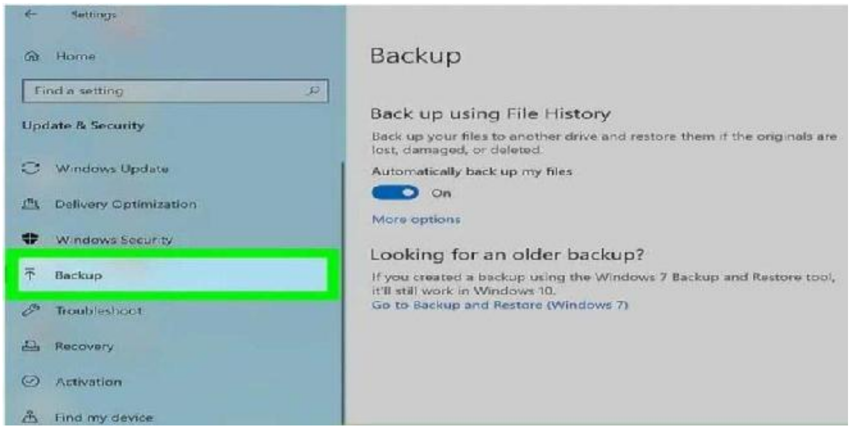
Requirement:

- Minimum 8gb of pendrive
- A computer with an internet connection
- Download window from the officaly website Microsoft. After successfully download the exe file
- Follows the steps below

Steps to create installation media

- Plugin a flash drive
- Download a tool media creation tool.exe
- Run it and click on accept
- Select installiation media(DVD,ISO/USB Drive)from another pc and click next
- Select your language windows edition and pc architecture and click next
- Now u need to select the USB pendrive and click next

Backup ur files(optional)



Backup

Back up using File History

Back up your files to another drive and restore them if the originals are lost, damaged, or deleted.

Automatically back up my files

On

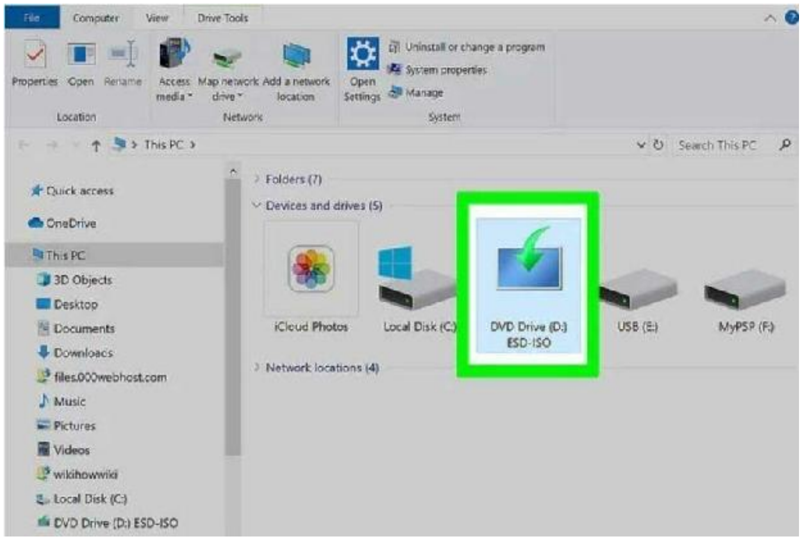
More options

Looking for an older backup?

If you created a backup using the Windows 7 Backup and Restore tool, it'll still work in Windows 10.

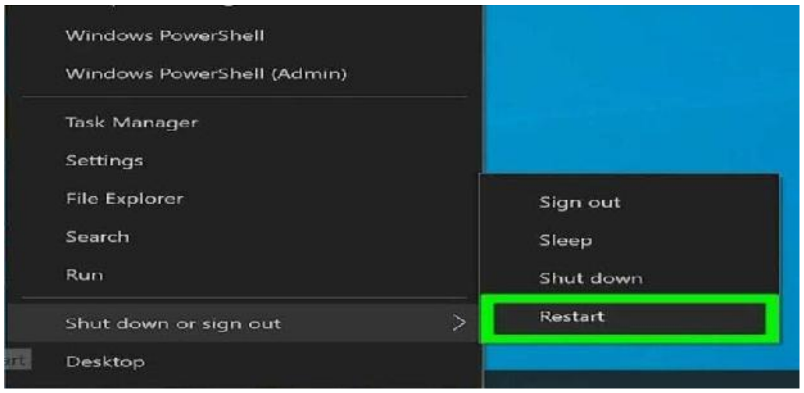
Go to Backup and Restore (Windows 7)

Insert Window installation media to Computer



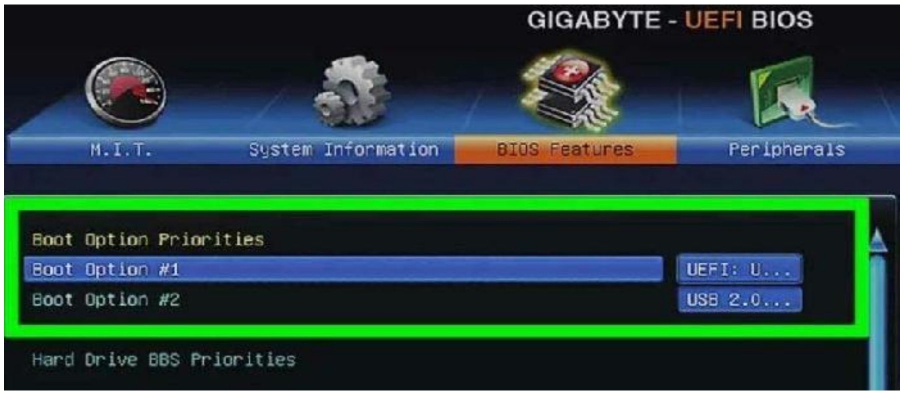
Now, it’s time to insert a media installation (USB or DVD) into the computer. Make sure all the windows files have been copied to the flash drive.

Boot the computer



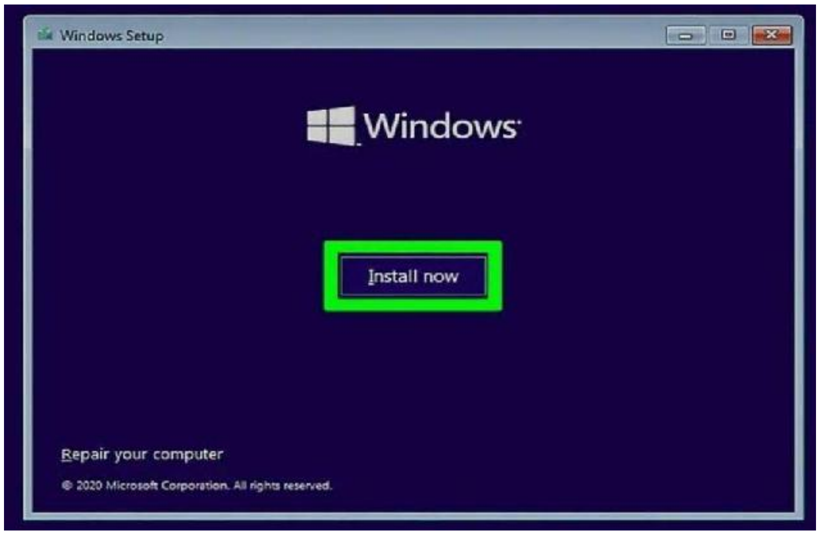
Click on the windows icon and choose the option “**Shut down or sign out** and then “**Restart**“. as pc reboot, press **F10, F11, F12, or ESC** in order to enter the boot menu. Press **F10, F11, F12, or ESC** to enter the BIOS (Basic Input Output System).

Select the USB Drive



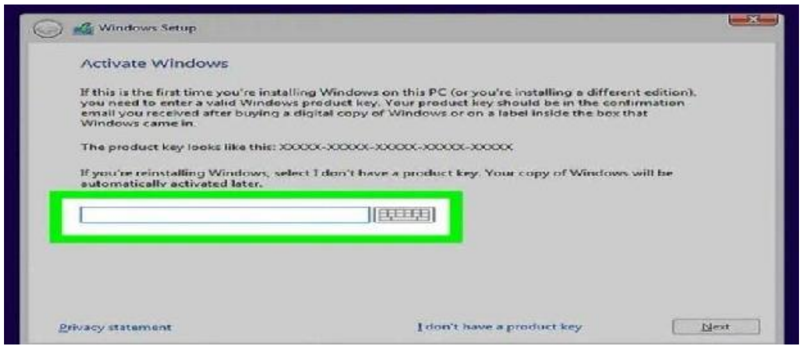
Select BIOS Features option
In Boot Options Priorities, Select the Boot Option #1. Click it and select the flash drive option.

Install Now



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

Enter your windows product key and click next



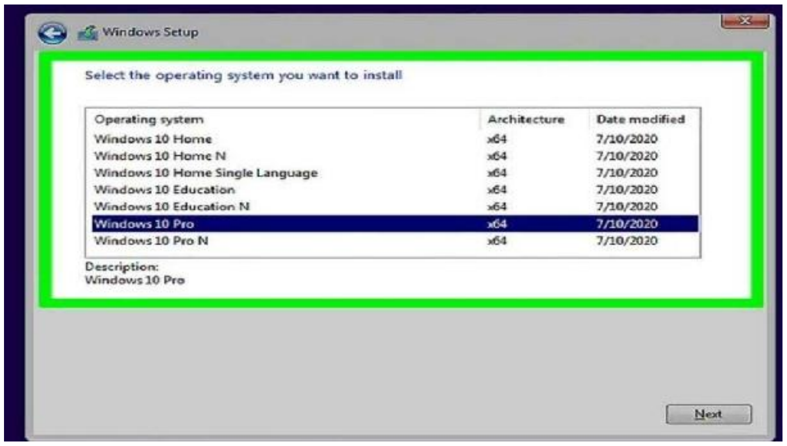
Activate Windows

If this is the first time you're installing Windows on this PC (or you're installing a different edition), you need to enter a valid Windows product key. Your product key should be in the confirmation email you received after buying a digital copy of Windows or on a label inside the box that Windows came in.

The product key looks like this: XXXXX-XXXXX-XXXXX-XXXXX-XXXXX

If you're reinstalling Windows, select I don't have a product key. Your copy of Windows will be automatically activated later.

Select the edition of window to install


Windows Setup

Select the operating system you want to install

Operating system	Architecture	Date modified
Windows 10 Home	x64	7/10/2020
Windows 10 Home N	x64	7/10/2020
Windows 10 Home Single Language	x64	7/10/2020
Windows 10 Education	x64	7/10/2020
Windows 10 Education N	x64	7/10/2020
Windows 10 Pro	x64	7/10/2020
Windows 10 Pro N	x64	7/10/2020

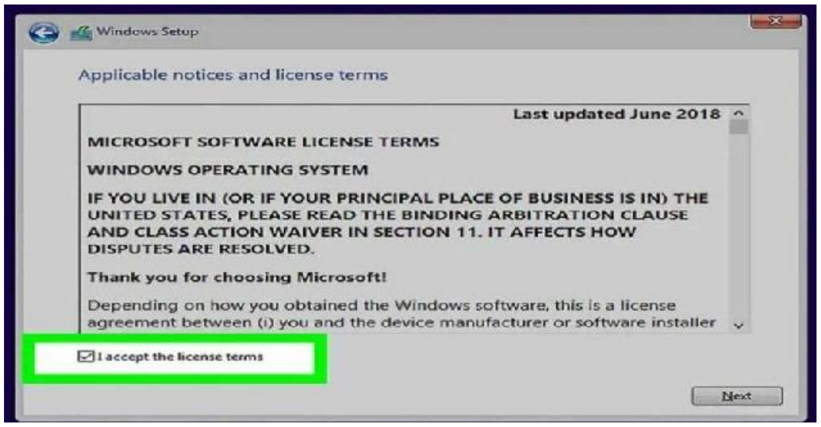
Description:
Windows 10 Pro

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 bit x86 or 32-bit operating system supports only 32-bit software program x64 or 64-bit operating system supports both 32-bit and 64-bit software program. go for "**Windows 10 Home**" mostly used OS for personal computers. Now click "NEXT".

Accept License Term "I accept the license terms"


Windows Setup

Applicable notices and license terms

Last updated June 2018

MICROSOFT SOFTWARE LICENSE TERMS

WINDOWS OPERATING SYSTEM

IF YOU LIVE IN (OR IF YOUR PRINCIPAL PLACE OF BUSINESS IS IN) THE UNITED STATES, PLEASE READ THE BINDING ARBITRATION CLAUSE AND CLASS ACTION WAIVER IN SECTION 11. IT AFFECTS HOW DISPUTES ARE RESOLVED.

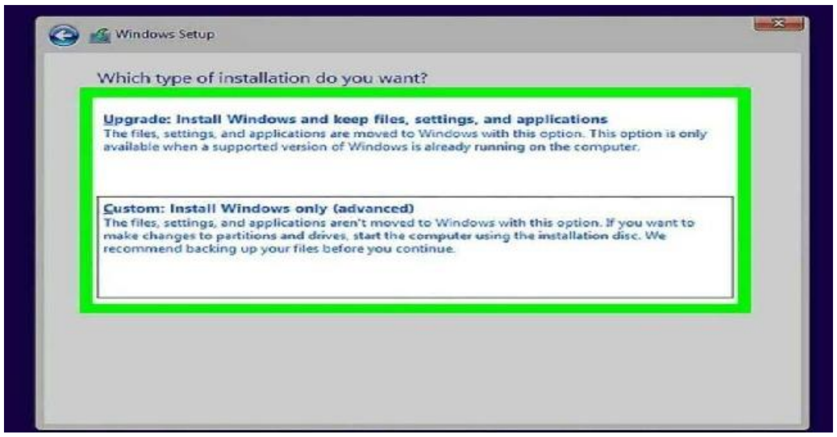
Thank you for choosing Microsoft!

Depending on how you obtained the Windows software, this is a license agreement between (i) you and the device manufacturer or software installer

[✓] I accept the license terms

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

Choose: Custom installs window only or Upgrade:



Which type of installation do you want?

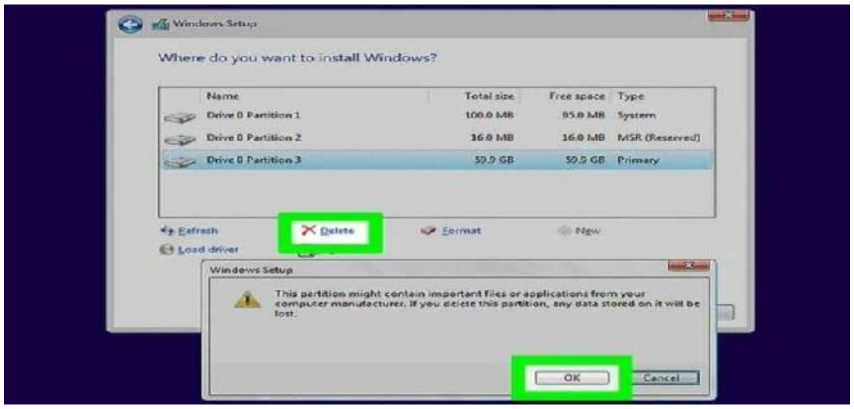
Upgrade: Install Windows and keep files, settings, and applications
The files, settings, and applications are moved to Windows with this option. This option is only available when a supported version of Windows is already running on the computer.

Custom: Install Windows only (advanced)
The files, settings, and applications aren't moved to Windows with this option. If you want to make changes to partitions and drives, start the computer using the installation disc. We recommend backing up your files before you continue.

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



Where do you want to install Windows?

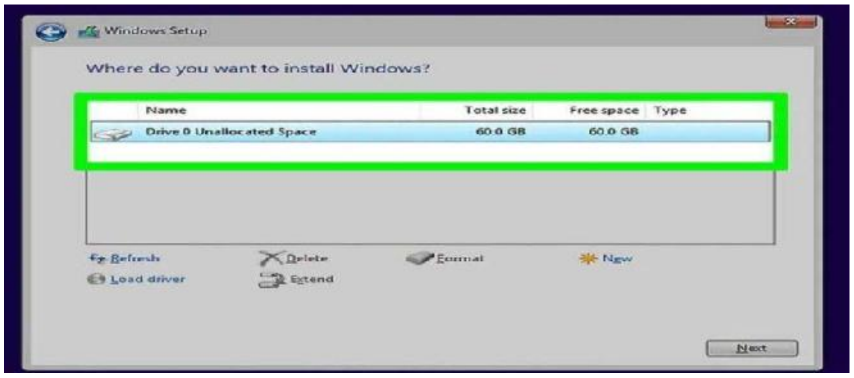
Name	Total size	Free space	Type
Drive 0 Partition 1	100.0 MB	95.0 MB	System
Drive 0 Partition 2	16.0 MB	16.0 MB	MSR (Reserved)
Drive 0 Partition 3	59.9 GB	59.9 GB	Primary

Windows Setup

This partition might contain important files or applications from your computer manufacturer. If you delete this partition, any data stored on it will be lost.

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"

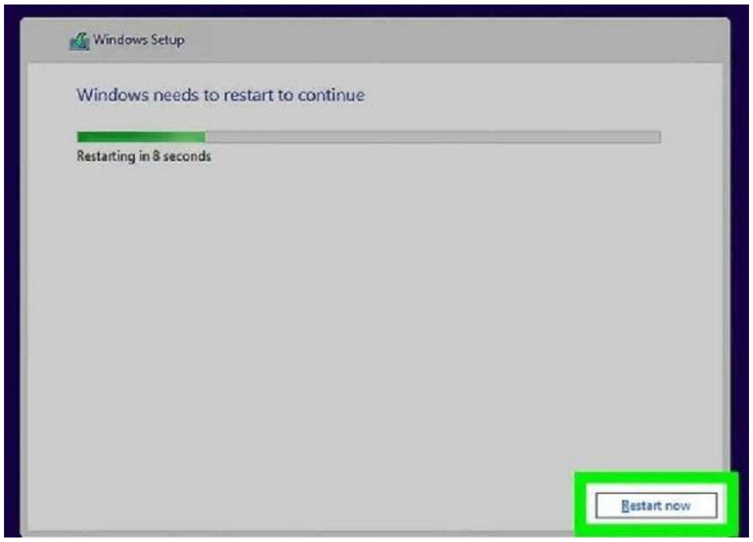


Where do you want to install Windows?

Name	Total size	Free space	Type
Drive 0 Unallocated Space	60.0 GB	60.0 GB	

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.



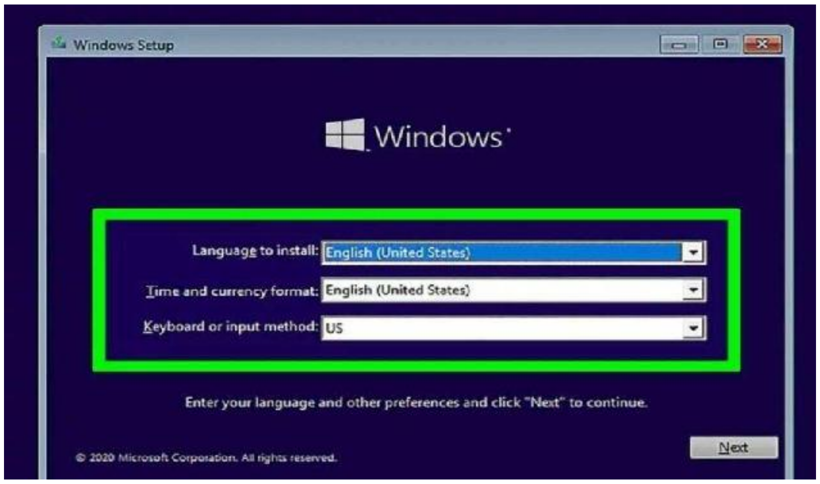
Windows Setup

Windows needs to restart to continue

Restarting in 8 seconds

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



Windows Setup

Language to install: English (United States)

Time and currency format: English (United States)

Keyboard or input method: US

Enter your language and other preferences and click "Next" to continue.

© 2020 Microsoft Corporation. All rights reserved.

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.

EX NO: 2 ILLUSTRATE UNIX COMMANDS & SHELL PROGRAMMING

DATE:

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 log on, 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:

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

DIRECTORY COMMANDS:

Command	Function
Pwd	Path of the present working directory
Mkdirdir	A directory is created in the given name under the current directory
mkdirdir1 dir2	A number of sub-directories can be created under one stroke
cd subdir	Change Directory. If the subdirstarts with **/** then path startsfrom **root** (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

Command	Function
cat >filename	To create a file with some contents. To end typing press **Ctrl+d**. The >symbol means redirecting output to a file. (<for input)
cat filename	Displays the file contents.
cat >>filename	Used to append contents to a file
cpsrc des	Copy files to given location. If already exists, it will be overwritten
cp –i src des	Warns the user prior to overwriting the destination file
cp –r src des	Copies the entire directory, all its sub-directories and files.
mv old new	To rename an existing file or directory. –i option can also be used
mv f1 f2 f3 dir	To move a group of files to a directory.
mv –v old new	Display name of each file as it is moved.
Rmfile	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 *	To forcibly remove even write-protected files
Ls	Lists all files and subdirectories (blue colored) in sorted manner.
Lsname	To check whether a file or directory exists.
lsname*****	Short-hand notation to list out filenames of a specific pattern.
ls –a	Lists all files including hidden files (files beginning with **.**)
ls –x dirname	To have specific listing of a directory.
ls –R	Recursive listing of all files in the subdirectories

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/student
[student@veccse  ~]bcbc
1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc. This is freesoftware
with 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] [-qpriority] [

-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

Lp submits files for printing or alters a pending job. Use a filename of "-" to force printing from the standard input. Cancel cancels existing print jobs. The -a option will remove all jobs from the specified destination.

OPTIONS

The following options are recognized by lp:

[student@veccse ~] history

```
1        date
2        date +%D
3        date +%T
4        date +%Y
5        date +%H
6        who
7        who am i
8        tty
9        uname
10       uname -r
11       uname -n
12       echo "helloi"
13       echo $HOME
14       bc
15       man lp
16       history
```

DIRECTORY COMMANDS

[student@veccse]$ pwd

/home/student

[student@veccse ~]mkdir san

[student@veccse ~]mkdir s1 s2

```
[student@veccse ~]ls
s1  s2  san [student@veccse  ~]cd s1
[student@veccse  s1]$ cd / [student@veccse
/]$ cd . . [student@veccse  /]$ rmdir s1
[student@veccse  ~]$ 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 test hi 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 test1 cmp: test: No such file or directory
```

RESULT

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

EX NO: 2 **ILLUSTRATE UNIX COMMANDS & SHELL PROGRAMMING**
DATE:

Aim:
To implement the Unix Commands & Shell programming.

Step 1 - Format the drive and set the primary partition as active

1. Connect the USB flash drive to your technician PC.
2. Open Disk Management: Right-click on **Start** and choose **Disk Management**.
3. Format the partition: Right-click the USB drive partition and choose **Format**. Select the **FAT32** file system to be able to boot either BIOS-based or UEFI-based PCs.
4. Set the partition as active: Right-click the USB drive partition and click **Mark Partition as Active**.

Step 2 - Copy Windows Setup to the USB flash drive

1. Use File Explorer to copy and paste the entire contents of the Windows product DVD or ISO to theUSB flash drive.

Step 3 - Install Windows to the new PC

1. Connect the USB flash drive to a new PC.
2. Turn on the PC and press the key that opens the boot-device selection menu for the computer, suchas the Esc/F10/F12 keys. Select the option that boots the PC from the USB flash drive.Windows Setupstarts. Follow the instructions to install Windows.
3. Remove the USB flash drive 4.

.

If your Windows image is larger than 4GB

Windows USB install drives are formatted as FAT32, which has a 4GB filesize limit. If your image is larger than the filesize limit:

1. Copy everything except the Windows image file (sources\install.wim) to the USB drive (either drag and drop, or use this command, where D: is the mounted ISO and E: is the USB flash drive.)

   command
   ```
   robocopy D: E: /s /max:3800000000
   ```

2. Split the Windows image file into smaller files, and put the smaller files onto the USB drive:
3. Command

   ```
   Dism /Split-Image /ImageFile:D:\sources\install.wim /SWMFile:E:\sources\install.swm
   /FileSize:3800
   ```

BASIC UNIX COMMANDS

Format	Purpose	Example	Result
+%m	To display only month	$date+%m	06
+%h	To display month name	$date+%h	June
+%d	To display day of month	$date+%d	01
+%y	To display last two digits of years	$date+%y	09
+%H	To display hours	$date+%H	10
+%M	To display minutes	$date+%M	45
+%S	To display seconds	$date+%S	55

date

–used to check the date and time
Syn:$date

a) **cal**

–used to display the
calendarSyn:$cal 2 2009

c) echo

–used to print the message on the screen.
Syn:$echo "text"

d)ls

–used to list the files. Your files are kept in a directory.

Syn:$lsls–s
All files (include files with prefix) ls–l Lodetai (provide file statistics) ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l) ls–s Order by size ls–r Reverse order

ls–f Mark directories with /,executable with* , symbolic links with @, local sockets with =, name
dpipes(FIFOs)with ls–s Show file size

ls– h" Human Readable", show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or) ls[a- m]*List all the files whose name begin with alphabets From „a" to „m" ls[a]*List all the files whose name begins with „a" or „A"
Eg:$ls>my list Output of „ls" command is stored to disk file named „my list"

e) **lp**

–used to take printouts Syn:$lp filename f)**man**

–used to provide manual help on every UNIX commands.
Syn:$man unix command
$man cat

g) **who** & **whoami**

–it displays data about all users who have logged into the system currently. The next command displays aboutcurrent user only.
Syn:$who$whoami

h) **uptime**

–tells you how long the computer has been running since its last reboot or power-off. Syn:$uptime

i) **uname**

–it displays the system information such as hardware platform, system name and processor, OStype. Syn:$uname–a

j)**hostname**

–displays and set system host nameSyn:$ hostname

k) **bc**

–stands for „best calculator‟

```
$bc          $ bc         $ bc          $ bc
10/2*3       scale =1     ibase=2       sqrt(196
                                        )
15           2.25+1       obase=16      14 quit
             3.35         11010011
              quit        89275
                           1010

$bc                    $ bc-l
for(i=1;i<3;i=i+1)I   scale=
    1                  2
                     s(3.14)
   2                      0
  3                 quit
```

FILE MANIPULATION COMMANDS a)**cat–**
this create, view and concatenate files.
Creation:
Syn:$cat>filename
Viewing:
Syn:$cat filename
Add text to an existing file:
Syn:$cat>>filename

Concatenate:
Syn:$catfile1file2>file3
$catfile1file2>>file3 (no over writing of file3)

b) **grep**–used to search a particular word or pattern related to that word from the file.

Syn:$grep search word
filenameEg:$grep anu student

c) **rm**–deletes a file from the file
systemSyn:$rm filename

d) **touch**–used to create a blank
file.Syn:$touch file names

e) **cp**–copies the files or
directories Syn:$cpsource file
destination file Eg:$cp student
stud

f) **mv**–to rename the file or directory syn:$mv old file new
fileEg:$mv–i student student list(-i prompt when
overwrite)

g) **cut**–it cuts or pickup a given number of character or fields of the
file.Syn:$cut<option><filename>
Eg: $cut –c filename
$cut–c1-10emp
$cut–f 3,6emp
$ cut –f 3-6 emp
-c cutting columns
-f cutting fields

h) **head**–displays10 lines from the head(top)of a given
fileSyn:$head filename
Eg:$head student

To display the top two lines:

Category	Operation	Permission
u– user sg–group o– others	+assign -remove =assign absolutely	r– read w– write x-execute

Syn:$head-i)**tail**–Syn:$tail filenameEg:$tail student
To display the bottom two lines;
Syn:$ tail -2 student

j) **chmod**–used to change the permissions of a file or director y.

2student
displays last 10 lines of the file

Syn:$ch mod category operation permission file
Where, Category–is the user type
Operation–is used to assign or remove permissionPermission–is the type of permission
File–are used to assign or remove permission allExamples:
$chmodu-wx student
Removes write and execute permission for users
$ch modu+rw,g+rwstudent
Assigns read and write permission for users and groups
$chmodg=rwx student
Assigns absolute permission for groups of all read, write and execute permissions

k) **wc**–it counts the number of lines, words, character in a specified file(s)
l) with the options as –l,-w,-cSyn: $wc –l filename

$wc –w filename
$wc–c filename

Result:

Thus the program has been executed successfully.

EX.NO: 3a PROCESS CREATION – FORK ( ) SYSTEM CALL

DATE:

AIM: To create a new child process using fork( ) system call.

fork()

The fork system call is used to create a new process called *child* process.

- The return value is 0 for a child process.
- The return value is negative if process creation isunsuccessful.
- For the parent process, return value is positive

The child process is an exact copy of the parent process.

Both the child and parent continue to execute the instructions following fork call.

The child can start execution before the parent or vice-versa.

getpid()and getppid()

The getpid system call returns process ID of the calling process

The getppid system call returns parent process ID of the calling process

Algorithm:

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

Program:

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

Output:

```
$ gcc fork.c
$ ./a.out
Child process:
Process id is 19499
Value of x is 6
Process id of parent is 19498
Parent process:
Process id is 19498
Value of x is 6
Process id of shell is 3266
```

Result:

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

Ex.No: 3b **Implementation of wait ( ) system call**

Date:

Aim:

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

wait()

- The wait system call causes the parent process to be blocked until a child terminates.
- When a process terminates, the kernel notifies the parent by sending the SIGCHLD signal to the parent.
- Without wait, the parent may finish first leaving a *zombie* child, to be adopted by init process

Algorithm:

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

Program:

```
/* Wait for child termination - wait.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
```

```
#include <sys/wait.h>
main()
{
    int i, status;
    pid_t pid;
    pid = fork();
    if (pid <  0)
    {
        printf("\nProcess creation failure\n");
        exit(-1);
    }
    else if(pid > 0)
    {
        wait(NULL);
        printf ("\nParent starts\nEven Nos: ");
        for (i=2;i<=10;i+=2)
            printf  ("%3d",i);
        printf ("\nParent ends\n");
    }
    else if (pid == 0)
    {
        printf ("Child starts\nOdd Nos: ");
        for (i=1;i<10;i+=2)
            printf ("%3d",i);
        printf ("\nChild  ends\n");
    }
}
```

Output:

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

Result:

Thus using wait system call zombie child processes were avoided.

Ex.No: 3c IMPLEMENTATION OF EXEC ( ) SYSTEM CALL

Date:

Aim:

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

execl()

The exec family of function (execl, execv, execle, execve, execlp, execvp) is used by the child process to load a program and execute. execl system call requires path, program name and null pointer.

Algorithm:

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

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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
main()
{
    pid_t pid;
    switch(pid = fork())
    {

        case -1:
            perror("Fork failed");
            exit(-1);

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

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

Output:

```
$ gcc exec.c
$ ./a.out
Child process
Sat Apr 09 13:46:59 IST 2022
Child Terminated
```

Result:

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

Ex.No: 3d **Implementation of stat( ) system call**

Date:

Aim:

To display file status information using stat( ) system call.

exit()

- The exit system call is used to terminate a process either normally or abnormally.
- Closes all standard I/O streams.

stat()

- The stat system call is used to return information about a file as a structure.

Algorithm:

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

Program:

```
/* File status - stat.c */
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char*argv[])
{
    struct stat file;
    int n;
    if (argc != 2)
    {
        printf("Usage: ./a.out <filename>\n");
        exit(-1);
    }
    if ((n = stat(argv[1], &file)) == -1)
    {
        perror(argv[1]);
        exit(-1);
    }
```

```
        printf("User id : %d\n", file.st_uid);
        printf("Group id : %d\n", file.st_gid);
        printf("Block size : %d\n", file.st_blksize);
        printf("Blocks allocated : %d\n", file.st_blocks);
        printf("Inode no. : %d\n",  file.st_ino);
        printf("Last accessed : %s", ctime(&(file.st_atime)));
        printf("Last modified : %s", ctime(&(file.st_mtime)));
        printf("File size : %d bytes\n", file.st_size);
        printf("No. of links : %d\n", file.st_nlink);
        printf("Permissions : ");
        printf( (S_ISDIR(file.st_mode)) ? "d" : "-");
        printf( (file.st_mode & S_IRUSR) ? "r" : "-");
        printf( (file.st_mode & S_IWUSR) ? "w" : "-");
        printf( (file.st_mode & S_IXUSR) ? "x" :  "-");
        printf( (file.st_mode & S_IRGRP) ? "r" :  "-");
        printf( (file.st_mode & S_IWGRP) ? "w" : "-");
        printf( (file.st_mode & S_IXGRP) ? "x" :  "-");
        printf( (file.st_mode & S_IROTH) ? "r" :  "-");
        printf( (file.st_mode & S_IWOTH) ? "w" : "-");
        printf( (file.st_mode & S_IXOTH) ? "x" : "-");
        printf("\n");
        if(file.st_mode & S_IFREG)
        printf("File type : Regular\n");
        if(file.st_mode & S_IFDIR)
        printf("File type :  Directory\n");
    }
```

Output:

```
 $ cat file2
Functions Programming
it0201 pts/0 2021-03-10 13:28 (192.168.8.84)
 $ gcc stat.c
$ ./a.out  file2 User
id : 736 Group id :
736 Block size :
4096 Blocks
allocated : 8
Inode no. : 15335432
Last accessed : Mon Mar 8 20:01:39 2021
Last modified : Wed Mar 10 13:44:18 2021
File size : 76 bytes
No. of links : 1
Permissions : -rw-rw-r--
File type : Regular
```

Result:

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

Ex.No: 3e Implementation of opendir( ), readdir(), closedir() system call

Date:

Aim:

To display directory contents using opendir( ), readdir( ), closedir( ) system call.

- **opendir(), readdir()and closedir()**
  - The opendir() system call is used to open a directory.
  - It returns a pointer to the first entry.
  - It returns NULL on error.
- **The readdir() system call is used to read a directory as a dirent structure.**
  - It returns a pointer pointing to the next entry in directory stream.
  - It returns NULL if an error or end-of-file occurs.
- **The closedir() system call is used to close the directory stream.**
- **Write to a directory is done only by the kernel.**

Algorithm:

1. Get directory name as command line argument.
2. If directory does not exist then stop.
3. Open the directory using opendir system call that returns a structure
4. Read the directory using readdir system call that returns a structure
5. Display d_name member for each entry.
6. Close the directory using closedir system call.
7. Stop

Program:

```
/* Directory content listing - dirlist.c */
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h> main(int
argc, char *argv[])
{
struct dirent *dptr;
DIR *dname;
if (argc != 2)
{
printf("Usage: ./a.out <dirname>\n");
exit(-1);
}
if((dname = opendir(argv[1])) == NULL)
{perror(argv[1]);
exit(-1);
  }
while(dptr=readdir(dname))
printf("%s\n", dptr->d_name);
closedir(dname);
}
```

Output:

```
$ gcc dirlist.c
$ ./a.out dir1 wait.c
a.out
..
stat.c dirlist.c
fork.c
.
exec.c new.txt
```

Result:

Thus files and subdirectories in the directory were listed that includes hidden files

EX. No: 4a CPU Scheduling Algorithms- FCFS Scheduling
Date:

Aim:

To schedule snapshot of processes queued according to FCFS scheduling.

Process Scheduling:

- CPU scheduling is used in multi-programmed operating systems.
- By switching CPU among processes, efficiency of the system can be improved.
- Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.
- Gantt chart provides a way of visualizing CPU scheduling and enables tounderstand better.

First Come First Serve (FCFS):

- Process that comes first is processed first
- FCFS scheduling is non-preemptive
- Not efficient as it results in long average waiting time.
- Can result in starvation, if processes at beginning of the queue have longbursts.

Algorithm:

```
1.  Define an array of structure process with members pid, btime, wtime & ttime.
2.  Get length of the ready queue, i.e., number of process (say n)
3.  Obtain btime for each process.
4.  The wtime for first process is 0.
5.  Compute wtime and ttime for each process as:
      a. $wtime_{i+1} = wtime_i + btime_i$
      b. $ttime_i = wtime_i + btime_i$
6.  Compute average waiting time awat and average turnaround time atur
7.  Display the btime, ttime and wtime for each process.
8.  Display GANTT chart for the above scheduling
9.  Display awat time and atur
10. Stop
```

Program

```
/* FCFS Scheduling- fcfs.c */
#include <stdio.h>
struct process
{
    int  pid;
    int btime;
    int wtime;
    int ttime;
} p[10];

main()
{
    int i,j,k,n,ttur,twat; float
    awat,atur;
    printf("Enter no. of process : ");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        printf("Burst time for process P%d (in ms) : ",(i+1));
        scanf("%d", &p[i].btime);
        p[i].pid =  i+1;
    }

    p[0].wtime = 0;
    for(i=0; i<n;  i++)
    {
        p[i+1].wtime = p[i].wtime + p[i].btime;
        p[i].ttime = p[i].wtime +  p[i].btime;
    }
    ttur = twat = 0;
    for(i=0; i<n;  i++)
    {
        ttur += p[i].ttime;
        twat +=  p[i].wtime;
    }
    awat = (float)twat / n;
    atur = (float)ttur / n;

    printf("\n FCFS Scheduling\n\n");
    for(i=0; i<28; i++)
        printf("-");
    printf("\n Process B-Time T-Time W-Time\n");
    for(i=0; i<28; i++)
        printf("-");
for(i=0; i<n; i++)
printf("\nP%d\t%4d\t%3d\t%2d",p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
printf("\n");
for(i=0; i<28;  i++)
printf("-");
printf("\n\n Average waiting time: %5.2fms", awat);
printf("\n Average turn around time : %5.2fms\n", atur);
printf("\n\n GANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n");
printf("|");
for(i=0; i<n;  i++)
{
k = p[i].btime/2;
for(j=0; j<k; j++)
printf(" ");
printf("P%d",p[i].pid);
for(j=k+1; j<p[i].btime;j++)
printf(" ");
printf("|");
}
printf("\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n");
printf("0");
for(i=0; i<n;  i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}
}
```

Output:

```
$ cc fcfs.c
$./a.out
Enter no. of process : 4
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 4
Burst time for process P3 (in ms) : 11
Burst time for process P4 (in ms) :  6
```

FCFS Scheduling

```
-----------------------------
Process B-Time T-Time W-Time
-----------------------------
  P1       10     10      0
  P2        4     14     10
  P3       11     25     14
  P4        6     31     25

-----------------------------
Average waiting time: 12.25ms
Average turnaround time :  20.00ms


GANTT   Chart
--------------------------------------------
|       P1       |    P2 |       P3       |    P4    |
--------------------------------------------
0               10     14                25         31
```

Result:

Thus waiting time & turnaround time for processes based on FCFS scheduling was computed and the average waiting time was determined.

Ex. No:4b **SJF SCHEDULING**
Date:

Aim:

To schedule snapshot of processes queued according to SJF scheduling.

Shortest Job First (SJF):

- Process that requires smallest burst time is processed first.
- SJF can be preemptive or non–preemptive
- When two processes require same amount of CPU utilization, FCFS is used to break the tie.
- Generally efficient as it results in minimal average waiting time.
- Can result in starvation, since long critical processes may not be processed.

Algorithm:

```
1.  Define an array of structure process with members pid, btime, wtime &
    ttime.
2.  Get length of the ready queue, i.e., number of process (say n)
3.  Obtain btime for each process.
4.  Sort the processes according to their btime in ascending order.
      a. If two process have same btime, then FCFS is used to resolve the tie.
5.  The wtime for first process is 0.
6.  Compute wtime and ttime for each process as:
      a. $wtime_{i+1} = wtime_i + btime_i$
      b. $ttime_i = wtime_i + btime_i$
7.  Compute average waiting time awat and average turn around time atur.
8.  Display btime, ttime and wtime for each process.
9.  Display GANTT chart for the above scheduling.
10. Display awat and atur.
11. Stop.
```

Program:

```
    /* SJF Scheduling – sjf.c */
    #include <stdio.h>
    struct process
    {
        int  pid;
        int btime;
        int wtime;
        int ttime;
    } p[10], temp;

  main()
  {
  int i,j,k,n,ttur,twat;
  float awat,atur;
  printf("Enter no. of process : ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  {
  printf("Burst time for process P%d (in ms) : ",(i+1));
  scanf("%d", &p[i].btime);
  p[i].pid = i+1;
  }
  for(i=0; i<n-1; i++)
  {
  for(j=i+1; j<n; j++)
  {
  if((p[i].btime > p[j].btime) ||(p[i].btime == p[j].btime && p[i].pid >  p[j].pid))
  {
  temp = p[i];
  p[i] = p[j];
  p[j] = temp;
  }
  }
  }
  p[0].wtime = 0;
  for(i=0; i<n;  i++)
  {
  p[i+1].wtime = p[i].wtime + p[i].btime;
  p[i].ttime = p[i].wtime +  p[i].btime;
  }
  ttur = twat = 0;
  for(i=0; i<n; i++)
  {
  ttur += p[i].ttime;
  twat +=  p[i].wtime;
  }
```

```
awat = (float)twat / n;
atur = (float)ttur / n;
printf("\n SJF Scheduling\n\n");
for(i=0; i<28; i++)
printf("-");
printf("\n Process B-Time T-Time W-Time\n");
for(i=0; i<28; i++)
printf("-");
for(i=0; i<n;  i++)
printf("\n P%-4d\t%4d\t%3d\t%2d", p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
printf("\n");
for(i=0; i<28;  i++)
printf("-");
printf("\n\n Average waiting time: %5.2fms", awat);
printf("\n Average turn around time : %5.2fms\n", atur);
printf("\n\n GANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n|");
for(i=0; i<n;  i++)
{
k = p[i].btime/2;
for(j=0; j<k; j++)
printf(" ");
printf("P%d",p[i].pid);
for(j=k+1; j<p[i].btime;j++)
printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n0");
for(i=0; i<n;  i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}
}
```

Output:
$ cc sjf.c
$./a.out

```
Enter no. of process :5
Burst time for process      P1 (in ms) :    10
Burst time for process      P2 (in ms) :    6
Burst time for process      P3 (in ms) :    5
Burst time for process      P4 (in ms) :    6
Burst time for process      P5 (in ms) :    9
     SJF Scheduling

  Process B-Time T-Time W-Time
  ------------------------

     P3          5        5        0
     P2          6       11        5
     P4          6       17       11
     P5          9       26       17
     P1         10       36       26

  ________________________
Average waiting time: 11.08ms
Average turnaround time: 19.00ms

  GANTT   Chart
  |--P3---|----P2---|---P4--|----P5-----|-----P1----

  0-------5-------11------17---------26---------- 3
```

Result:

Thus waiting time & turnaround time for processes based on SJF scheduling was computed and the average waiting time was determined.

Ex. No: 4c **PRIORITY SCHEDULING**

Date:

Aim:

To schedule snapshot of processes queued according to Priority scheduling.

Priority:

- Process that has higher priority is processed first.
- Priority can be preemptive or non–preemptive.
- When two processes have same priority, FCFS is used to break the tie.
- Can result in starvation, since low priority processes may not be processed.

Algorithm:

```
1.  Define an array of structure process with members pid, btime, pri, wtime & ttime.
2.  Get length of the ready queue, i.e., number of process (say n)
3.  Obtain btime and pri for each process.
4.  Sort the processes according to their pri in ascending order.
      a. If two process have same pri, then FCFS is used to resolve the tie.
5.  The wtime for first process is 0.
6.  Compute wtime and ttime for each process as:
      a. $wtime_{i+1} = wtime_i + btime_i$
      b. $ttime_i = wtime_i + btime_i$
7.  Compute average waiting time awat and average turn around time atur
8.  Display the btime, pri, ttime and wtime for each process.
9.  Display GANTT chart for the above scheduling
10. Display awat and atur
11. Stop
```

Program:

```
/* Priority Scheduling- priority.c */
#include <stdio.h>
struct process
{
        int pid; int
        btime; int
        pri; int
        wtime; int
        ttime;
} p[10], temp;

main()
{
        int i,j,k,n,ttur,twat;
        float awat,atur;
        printf("Enter no. of process : ");
        scanf("%d", &n);
        for(i=0; i<n; i++)
        {
                printf("Burst time for process P%d (in ms) : ", (i+1));
                scanf("%d", &p[i].btime);
                printf("Priority for process P%d : ", (i+1));
                scanf("%d", &p[i].pri);
                p[i].pid = i+1;
        }
        for(i=0; i<n-1; i++)
        {
        for(j=i+1; j<n; j++)
                {
           if((p[i].pri > p[j].pri) ||(p[i].pri == p[j].pri && p[i].pid > p[j].pid))
                        {
                                temp = p[i];
                                p[i] = p[j];
                                p[j] = temp;
                        }
                }
        }
        p[0].wtime = 0;
        for(i=0; i<n;  i++)
        {
                p[i+1].wtime = p[i].wtime + p[i].btime;
                p[i].ttime = p[i].wtime +  p[i].btime;
        }
        ttur = twat = 0;
        for(i=0; i<n;  i++)
        {ttur += p[i].ttime;
                twat +=  p[i].wtime;
        }
awat = (float)twat / n;
atur = (float)ttur / n;
printf("\n\t Priority Scheduling\n\n");
for(i=0; i<38; i++)
printf("-");
printf("\nProcess B-Time Priority T-Time W-Time\n");
for(i=0; i<38; i++)
printf("-");
for (i=0; i<n; i++)
printf("\nP%4d\t%4d\t%3d\t%4d\t%4d",p[i].pid,p[i].btime,p[i].pri,p[i].ttime,
p[i].wtime);
printf("\n");
for(i=0; i<38;  i++)
printf("-");
printf("\n\nAverage waiting time: %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n", atur);
printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n|");
for(i=0; i<n;  i++)
{
k = p[i].btime/2;
for(j=0; j<k; j++)
printf(" ");
printf("P%d",p[i].pid);
for(j=k+1; j<p[i].btime;  j++)
printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n0");
for(i=0; i<n;  i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}
}
```

Output:

```
   $ cc priority.c
$./a.out
   Enter no. of process : 5
   Burst time for process P1 (in ms) :  10
   Priority for process P1 : 3
   Burst time for process P2 (in ms) :  7
   Priority for process P2 : 1
   Burst time for process P3 (in ms) :  6
   Priority for process P3 : 3
   Burst time for process P4 (in ms) :  13
   Priority for process P4 : 4
   Burst time for process P5 (in ms) :  5
   Priority for process P5 : 2
            Priority Scheduling

        ------------------------------------
        Process B-Time  Priority  T-Time      W-Time
        ------------------------------------
           P2        7        1        7        0

           P5        5        2       12        7
           P1       10        3       22       12
           P3        6        3       28       22
           P4       13        4       41       28
        ------------------------------------
Average waiting time: 13.80ms Average
        turnaround time:  22.00ms

        GANTT Chart
        ------------------------------------------------

        |    P2    |   P5   |      P1      |   P3   |       P4       |
        0          7        12             22       28               41
```

Result:

Thus waiting time & turnaround time for processes based on Priority scheduling was computed and the average waiting time was determined.

Ex. No: 4d **ROUND ROBIN SCHEDULING**

Date:

Aim:

To schedule snapshot of processes queued according to Round robin scheduling.

Round Robin:

- All processes are processed one by one as they have arrived, but in rounds.
- Each process cannot take more than the time slice per round.
- Round robin is a fair preemptive scheduling algorithm.
- A process that is yet to complete in a round is preempted after the time slice and put at the end of the queue.
- When a process is completely processed, it is removed from the queue.

Algorithm

1. Get length of the ready queue, i.e., number of process (say *n*)
2. Obtain *Burst* time B_i for each processes P_i.
3. Get the *time slice* per round, say *TS.*
4. Determine the number of rounds for each process.
5. The wait time for first process is 0.
6. If $B_i > TS$ then process takes more than one round. Therefore turnaround and waiting time should include the time spent for other remaining processes in the same round.
7. Calculate *average* waiting time and turnaround time
8. Display the GANTT chart that includes,
   a. Order in which the processes were processed in progression of rounds.
   b. Turnaround time T_i for each process in progression of rounds.
9. Display the *burst* time, *turnaround* time and *wait* time for each process (in order of rounds they were processed).
10. Display *average* wait time and turnaround time.
11. Stop.

Program:

```
/* Round robin scheduling- rr.c */
#include <stdio.h>
main()
{
    int i,x=-1,k[10],m=0,n,t,s=0;
    int a[50],temp,b[50],p[10],bur[10],bur1[10];
    int  wat[10],tur[10],ttur=0,twat=0,j=0;
    float awat,atur;
    printf("Enter no. of process : ");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        printf("Burst time for process P%d : ", (i+1));
        scanf("%d", &bur[i]);
        bur1[i] = bur[i];
    }
    printf("Enter the time slice (in ms) : ");
    scanf("%d", &t);
    for(i=0; i<n; i++)
    {
        b[i] = bur[i] / t;
        if((bur[i]%t) != 0)
            b[i] += 1;
        m += b[i];
    }
    printf("\n\t\tRound Robin Scheduling\n");
    printf("\nGANTT Chart\n");
    for(i=0; i<m; i++)
    printf(" --------------------- ");
    printf("\n");
    a[0] = 0;
    while(j <  m)
    {
        if(x == n-1)
        x = 0;
        else
            x++;
        if(bur[x] >=  t)
        {
            bur[x] -=  t;
            a[j+1] = a[j] + t;
            if(b[x] == 1)
            {
                p[s] = x;
            }     k[s] = a[j+1]; s++;
            j++;
            b[x] -= 1;
printf("P%d|", x+1);
}
else if(bur[x] != 0)
{
a[j+1] = a[j] + bur[x];
bur[x] = 0;
if(b[x] == 1)
{
p[s] = x;
k[s] = a[j+1];
s++;
}
j++;
b[x] -= 1;
printf("P%d|",x+1);
}
}
printf("\n");
for(i=0;i<m;i++)
printf(" ---------------------------- ");
printf("\n");
for(j=0; j<=m; j++)
printf("%d\t", a[j]);
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(p[i] >  p[j])
{
temp = p[i];
p[i] = p[j];
p[j] = temp;
temp = k[i];
k[i] = k[j];
k[j] = temp;
}
}
}
for(i=0; i<n; i++)
{
wat[i] = k[i] - bur1[i];
tur[i] = k[i];
}
for(i=0; i<n;  i++)
{
ttur += tur[i];
twat +=  wat[i];
}
printf("\n\n");
```

```
for(i=0; i<30;  i++) printf("-");
printf("\nProcess\tBurst\tTrnd\tWait\n");
for(i=0; i<30; i++)
printf("-");
for (i=0; i<n; i++)
printf("\nP%-4d\t%4d\t%4d\t%4d", p[i]+1, bur1[i], tur[i],wat[i]);
printf("\n");
for(i=0; i<30;  i++)
printf("-");
awat = (float)twat / n;
atur = (float)ttur / n;
printf("\n\nAverage waiting time: %.2f ms", awat);
printf("\nAverage turnaround time : %.2f ms\n", atur);
}
```

Output:

```
Enter no. of process : 5
Burst time for process P1 : 10
Burst time for process P2 : 29
Burst time for process P3 : 3
Burst time for process P4 : 7
Burst time for process P5 : 12
Enter the time slice (in ms) :  10

                 Round Robin Scheduling
GANTT Chart
-------------------------------------------------------------
P1   |  P2  |   P3   |   P4   |   P5   |   P2   |   P5   |   P2   |
-------------------------------------------------------------
0     10      20      23      30      40      50      52      61

-------------------------------
Process Burst        Trnd     Wait
-------------------------------
P1          10         10        0
P2          29         61       32
P3           3         23       20
P4           7         30       23
P5          12         52       40

-------------------------------
Average waiting time: 23.00 ms
Average turnaround time: 35.20  ms
```

Result:

Thus waiting time and turnaround time for processes based on Round robin scheduling was computed and the average waiting time was determined.

Ex. No: 5a **INTER PROCESS COMMUNICATION**
Date:

Aim:
To generate 25 Fibonacci numbers and determine prime amongst them using pipe.

Inter process Communication:

- Inter-Process communication (IPC), is the mechanism whereby one process can communicate with another process, i.e exchange data.
- IPC in linux can be implemented using pipe, shared memory, message queue, semaphore, signal or sockets.

Pipe:

- Pipes are unidirectional byte streams which connect the standard output from one process into the standard input of another process.
- A pipe is created using the system call *pipe* that returns a pair of file descriptors.
- The descriptor pfd[0] is used for reading and pfd[1] is used for writing.
- Can be used only between parent and child processes.

Algorithm:

```
1.  Declare a array to store Fibonacci numbers
2.  Declare an array pfd with two elements for pipe descriptors.
3.  Create pipe on pfd using pipe function call.
        a. If return value is -1 then stop
4.  Using fork system call, create a child process.
5.  Let the child process generate 25 Fibonacci numbers and store them in a
    array.
6.  Write the array onto pipe using write systemcall.
7.  Block the parent till child completes using waitsystem call.
8.  Store Fibonacci nos. written by child from the pipe in an array using read
    system call
9.  Inspect each element of the Fibonacci array and check whether they are
    prime
        a. If prime then print the Fibonacci term.
10. Stop
```

Program:

```
/* Fibonacci and Prime using pipe - fibprime.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
main()
{
    pid_t pid;
    int pfd[2];
    int i,j,flg,f1,f2,f3;
    static unsigned int ar[25],br[25];
    if(pipe(pfd) == -1)
    {printf("Error in pipe");
        exit(-1);
    }
    pid=fork(); if
    (pid ==  0)
    { printf("Child process generates Fibonacci series\n" );
        f1 = -1;
        f2 = 1;
        for(i = 0;i < 25; i++)
        {f3 = f1 + f2;
        printf("%d\t",f3);
            f1 =  f2;
            f2 = f3;
            ar[i] = f3;
        }write(pfd[1],ar,25*sizeof(int));
    }
    else if (pid > 0)
    {wait(NULL);
        read(pfd[0], br, 25*sizeof(int));
        printf("\nParent prints Fibonacci that are Prime\n");
        for(i = 0;i < 25; i++)
        {flg = 0;
            if (br[i] <= 1)
            flg = 1;
            for(j=2; j<=br[i]/2; j++)
            {if (br[i]%j == 0)
                {flg=1; break;
                }}
            if (flg == 0) printf("%d\t", br[i]);
        }
    elseprintf("\n");}
    {
        printf("Process creation failed"); exit(-1);
    }
}
```

Output:

```
   $ cc fibprime.c
$ ./a.out

   Child process generates Fibonacci series
     0          1          1               2          3              5             8             13
     21         34         55              89         144            233           377           610
     987        1597       2584            4181       6765           10946         17711         28657
    46368
    Parent    prints    Fibonacci     that       are Prime
     2            3            5            13          89             233           1597          28657
```

Result:

Thus generation of Fibonacci numbers that are prime is determined using IPC pipe.

Ex. No: 5b   Implementation of commands using pipes (who | wc -l)
Date:

Aim:

To determine number of users logged in using pipe.

Algorithm:

1. Declare an array *pfd* with two elements for pipe descriptors.
2. Create pipe on *pfd* using pipe function call.
   a. If return value is -1 then stop
3. Using fork system call, create a child process.
4. Free the standard output (1) using close system call to redirect the output to pipe.
5. Make a copy of write end of the pipe using dup system call.
6. Execute who command using execlp system call.
7. Free the standard input (0) using close system call in the other process.
8. Make a close of read end of the pipe using dup system call.
9. Execute wc –l command using execlp system call.
10. Stop

Program:

```
/* No. of users logged - cmdpipe.c */
#include <stdio.h>
#include  <stdlib.h>
#include <unistd.h>
int main()
{
    int  pfds[2];
    pipe(pfds);
    if (!fork())
    {
        close(1);
        dup(pfds[1]);
        close(pfds[0]);
    }   execlp("who", "who", NULL);
    else
    {
        close(0);
        dup(pfds[0]);
        close(pfds[1]);
    }   execlp("wc", "wc", "-l", NULL);
}
```

Output:

```
   $ cc cmdpipe.c
$ ./a.out
   15
```

Result:

Thus standard output of who is connected to standard input of wc using pipe to compute number of users logged in.

Ex. No: 5c **CHAT MESSAGING**
Date:

Aim:

To exchange message between server and client using message queue.

Message Queue:

- A message queue is a linked list of messages stored within the kernel.
- A message queue is identified by a unique identifier.
- Every message has a positive long integer type field, a non-negative length, and the actual data bytes.
- The messages need not be fetched on FCFS basis. It could be based on type field.

Algorithm:

<u>Server</u>

```
1. Declare a structure mesgq with type and text fields.
2. Initialize key to 2013 (some random value).
3. Create a message queue using msgget with key & IPC_CREAT as parameter.
    a. If message queue cannot be created then stop.
4. Initialize the message type member of mesgq to 1.
5. Do the following until user types Ctrl+D
    a. Get message from the user and store it in text member.
    b. Delete the newline character in text member.
    c. Place message on the queue using msgsend for the client to read.
    d. Retrieve the response message from the client using msgrcv function.
    e. Display the text contents.
6. Remove message queue from the system using msgctl with IPC_RMID as parameter.
7. Stop.
```

<u>Client</u>

```
1. Declare a structure mesgq with type and text fields.
2. Initialize key to 2013 (same value as in server).
3. Open the message queue using msgget with key as parameter.
    a. If message queue cannot be opened then stop.
4. Do while the message queue exists
    a. Retrieve the response message from the server using msgrcv function
    b. Display the text contents.
    c. Get message from the user and store it in text member.
    d. Delete the newline character in text member.
    e. Place message on the queue using msgsend for the server to read.
5. Print "Server Disconnected".
6. Stop
```

Program:

Server

```
/* Server chat process - srvmsg.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesgq
{
	long type;
	char text[200];
} mq;
main()
{
	int msqid, len;
	key_t key = 2013;
	if((msqid = msgget(key, 0644|IPC_CREAT)) == -1)
	{
		perror("msgget");
		exit(1);
	}
	printf("Enter text, ^D to quit:\n"); mq.type = 1;
	while(fgets(mq.text, sizeof(mq.text), stdin) != NULL)
	{
		len = strlen(mq.text);
		if (mq.text[len-1] == '\n')
			mq.text[len-1] = '\0'; msgsnd(msqid,
		&mq, len+1, 0); msgrcv(msqid, &mq,
		sizeof(mq.text), 0, 0);
		printf("From Client: \"%s\"\n", mq.text);
	}
	msgctl(msqid, IPC_RMID, NULL);
}
```

Client

```
/* Client chat process - climsg.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesgq
{
	long type;
	char text[200];
} mq;
```

```
main()
{
      int msqid, len;
      key_t key = 2013;
      if ((msqid = msgget(key, 0644)) == -1)
      {
            printf("Server not active\n");
            exit(1);
      }
      printf("Client ready :\n");
      while (msgrcv(msqid, &mq, sizeof(mq.text), 0, 0) != -1)
      {
            printf("From Server: \"%s\"\n", mq.text);
            fgets(mq.text, sizeof(mq.text), stdin);
            len = strlen(mq.text);
            if (mq.text[len-1] == '\n')
                  mq.text[len-1] = '\0';
            msgsnd(msqid, &mq, len+1, 0);
      }
      printf("Server Disconnected\n");
}
```

Output:

<u>Server</u>

```
$ cc srvmsg.c -o srvmsg
$ ./srvmsg
Enter text, ^D to quit: hi
From Client: "hello" Where r u?
From Client: "I'm where i am" bye
From Client: "ok"
^D
```

<u>Client</u>

```
$ cc climsg.c -o climsg
$ ./climsg
Client ready:
From Server: "hi" hello
From Server: "Where r u?" I'm where i am
From Server: "bye" ok
Server Disconnected
```

Result:

Thus chat session between client and server was done using message queue.

Ex. No: 5d Process Communication using Shared Memory
Date:

Aim:

To demonstrate communication between processes using shared memory.

Shared Memory:

- Two or more processes share a single chunk of memory to communicate randomly.
- Semaphores are generally used to avoid race condition amongst processes.
- Fastest amongst all IPCs as it does not require any system call.
- It avoids copying data unnecessarily.

Algorithm:

<u>Server</u>

```
1.  Initialize size of shared memory shmsize to 27.
2.  Initialize key to 2013 (some random value).
3.  Create a shared memory segment using shmget with key & IPC_CREAT as
    parameter.
      a. If shared memory identifier shmid is -1, then stop.
4.  Display shmid.
5.  Attach server process to the shared memory using shmmat with shmid as
    parameter.
      a. If pointer to the shared memory is not obtained, then stop.
6.  Clear contents of the shared region using memset function.
7.  Write a–z onto the shared memory.
8.  Wait till client reads the shared memory contents.
9.  Detatch process from the shared memory using shmdt system call.
10. Remove shared memory from the system using shmctl with IPC_RMID
    argument.
11. Stop.
```

<u>Client</u>

```
1.  Initialize size of shared memory shmsize to 27.
2.  Initialize key to 2013 (same value as in server).
3.  Obtain access to the same shared memory segment using same key.
      a. If obtained then display the shmid else print "Server not started"
4.  Attach client process to the shared memory using shmmat with shmid as
    parameter.
      a. If pointer to the shared memory is not obtained, then stop.
5.  Read contents of shared memory and print it.
6.  After reading, modify the first character of shared memory to '*'
7.  Stop.
```

Program:

Server

```
/* Shared memory server - shms.c */
#include <stdio.h>
#include <stdlib.h>
#include <sys/un.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define shmsize 27
main()
{
        char c;
        int shmid;
        key_t key =2013;
        char *shm, *s;
        if ((shmid = shmget(key, shmsize, IPC_CREAT|0666)) <  0)
        { perror("shmget"); exit(1);
        }
        printf("Shared memory id : %d\n", shmid);
        if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
        { perror("shmat"); exit(1);
        }
        memset(shm, 0, shmsize);
        s = shm;
        printf("Writing (a-z) onto shared memory\n");
        for (c = 'a'; c <= 'z'; c++)
                *s++ = c;
                *s = '\0';
        while (*shm != '*');
        printf("Client finished reading\n");
        if(shmdt(shm) != 0)
                fprintf(stderr, "Could not close memory segment.\n");
        shmctl(shmid, IPC_RMID, 0);
}
```

Client

```
/* Shared memory client - shmc.c */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define shmsize 27
main()
{
        int shmid;
        key_t key = 2013;
        char *shm, *s;
```

```
            if ((shmid = shmget(key, shmsize, 0666)) <  0)
            {
                  printf("Server not started\n");
            }     exit(1);
            else
                  printf("Accessing shared memory id : %d\n",shmid);
            if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
            {
                  perror("shmat");
                  exit(1);
            }
            printf("Shared memory contents:\n");
            for (s = shm; *s != '\0'; s++)
                  putchar(*s);
                  putchar('\n');
                  *shm = '*';
      }
```

Output:

Server

```
$ cc shms.c -o shms
  $ ./shms
   Shared memory id : 196611
   Writing (a-z) onto shared memory
   Client finished reading
```

Client

```
   $ cc shmc.c -o shmc
$ ./shmc
   Accessing shared memory id : 196611
   Shared memory contents:
   abcdefghijklmnopqrstuvwxyz
```

Result:

Thus contents written onto shared memory by the server process is read by the client process using shared memory technique successfully.

Ex. No: 5e Implementation of Producer-Consumer Process

Date:

Aim:

To synchronize producer and consumer processes using semaphore.

Semaphores:

- A semaphore is a counter used to synchronize access to a shared data amongst multiple processes.
- To obtain a shared resource, the process should:
  - Test the semaphore that controls the resource.
  - If value is positive, it gains access and decrements value of semaphore.
  - If value is zero, the process goes to sleep and awakes when value is > 0.
- When a process relinquishes resource, it increments the value of semaphore by 1.

Producer-Consumer Process:

- A producer process produces information to be consumed by a consumer process.
- A producer can produce one item while the consumer is consuming another one.
- With bounded-buffer size, consumer must wait if buffer is empty, whereas producer must wait if buffer is full.
- The buffer can be implemented using any IPC facility.

Algorithm:

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

Program:

```
/* Producer-Consumer problem using semaphore – pcsem.c */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#define N 5
#define BUFSIZE 1
#define PERMS 0666
int *buffer;
int nextp = 0, nextc = 0;
int mutex, full, empty;
void producer()
{
    int data;
    if(nextp == N)
        nextp = 0;
    printf("Enter data for producer to produce : ");
    scanf("%d",(buffer + nextp));
    nextp++;
}
void consumer()
{
    int g;
    if(nextc ==  N)
        nextc = 0;
    g = *(buffer + nextc++);
    printf("\nConsumer consumes data %d",  g);
}
void sem_op(int id, int value)
{ struct sembuf op;
    int v; op.sem_num =
    0; op.sem_op =
    value;
    op.sem_flg =  SEM_UNDO;
    if((v = semop(id, &op, 1)) < 0)
        printf("\nError executing semop instruction");
}
void sem_create(int semid, int initval)
{
    int semval;
    union semun
    {
        int  val;
        struct semid_ds *buf;
    } s;  unsigned short *array;
```

```
        s.val = initval;
        if((semval = semctl(semid, 0, SETVAL, s)) < 0)
        printf("\nError in executing semctl");
}
void sem_wait(int id)
{
        int value = -1;
        sem_op(id, value);
}
void sem_signal(int id)
{
        int value = 1;
        sem_op(id, value);
}
main()
{
        int shmid, i;
        pid_t pid;
        if((shmid = shmget(1000, BUFSIZE, IPC_CREAT|PERMS)) <  0)
        {
                printf("\nUnable to create shared memory");
                return;
        }
        if((buffer = (int*)shmat(shmid, (char*)0, 0)) == (int*)-1)
        {
                printf("\nShared memory allocation error\n");
                exit(1);
        }
        if((mutex = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
        {
                printf("\nCan't create mutex semaphore");
                exit(1);
        }
        if((empty = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
        {
                printf("\nCan't create empty semaphore");
                exit(1);
        }
        if((full = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) ==-1)
        {
                printf("\nCan't create full semaphore");
                exit(1);
        }
        sem_create(mutex,1);
        sem_create(empty,N);
        sem_create(full, 0);
        if((pid = fork()) < 0)
        { printf("\nError in process creation");
            exit(1);
        }
        else if(pid > 0)
        { for(i=0; i<N; i++)
            { sem_wait(empty); sem_wait(mutex);
                producer();
                sem_signal(mutex);
                sem_signal(full);
            }
        }
        else if(pid == 0)
        { for(i=0; i<N; i++)
            { sem_wait(full); sem_wait(mutex);
                consumer();
                sem_signal(mutex);
                sem_signal(empty);
            }
            printf("\n");
        }
    }
```

Output:

```
  $ cc pcsem.c
$ ./a.out
    Enter data for producer to produce : 5
    Enter data for producer to produce : 8
    Consumer consumes data 5
    Enter data for producer to produce : 4
    Consumer consumes data 8
    Enter data for producer to produce : 2
    Consumer consumes data 4
    Enter data for producer to produce : 9
    Consumer consumes data  2
    Consumer consumes data 9
```

Result:

Thus synchronization between producer and consumer process using semaphore technique was implemented and executed successfully.

Ex. No: 6 **SEMAPHORE IMPLEMENTATION**
Date:

Aim:
To demonstrate the utility of semaphore in synchronization and multithreading.

Semaphore:

- ➢ The POSIX system in Linux has its own built-in semaphore library.
- ➢ To use it, include semaphore.h.
- ➢ Compile the code by linking with -lpthread -lrt.
- ➢ To lock a semaphore or wait, use the **sem_wait** function.
- ➢ To release or signal a semaphore, use the **sem_post** function.
- ➢ A semaphore is initialised by using **sem_init**(for processes or threads)
- ➢ To declare a semaphore, the data type is sem_t.

Algorithm:

1. 2 threads are being created, one 2 seconds after the first one.
2. But the first thread will sleep for 4 seconds after acquiring the lock.
3. Thus the second thread will not enter immediately after it is called, it will enter 4 – 2 = 2 secs after it is called.
4. *Stop*.

Program:/* C program to demonstrate working of Semaphores */
```
  #include <stdio.h>
  #include <pthread.h>
  #include <semaphore.h>
  #include <unistd.h>
  sem_t mutex;
  void* thread(void* arg)
  {//wait
  sem_wait(&mutex);
  printf("\nEntered..\n");
//critical section
sleep(4);
//signal
printf("\nJust Exiting...\n");
sem_post(&mutex);
}
int main()
{sem_init(&mutex, 0, 1);
        pthread_t t1,t2;
        pthread_create(&t1,NULL,thread,NULL);
        sleep(2);
        pthread5_create(&t2,NULL,thread,NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
        sem_destroy(&mutex);
        return 0;
  }
```

Output:

```
$ cc sem.c -lpthread
$ ./a.out
Entered..
Just Exiting...
Entered..
Just Exiting...
```

Result:

Thus semaphore implementation has been demonstrated using c program successfully.

Ex. No: 7 DEAD LOCK AVOIDANCE USING [Bankers Algorithm]
Date:

Aim:

To implement deadlock avoidance by using Banker's Algorithm.

Algorithm:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.
9. Stop.

Program:

```
#include<stdio.h>
#include<stdio.h>
main()
{
    int r[1][10], av[1][10];
    int all[10][10], max[10][10], ne[10][10], w[10],safe[10];
    int i=0, j=0, k=0, l=0, np=0, nr=0, count=0, cnt=0;
    clrscr();
    printf("Enter the number of processes in a system");
    scanf("%d", &np);
    printf("Enter the number of resources in a system");
    scanf("%d",&nr);
    for(i=1; i<=nr; i++)
    { printf("Enter no. of instances of resource R%d " ,i);
        scanf("%d", &r[0][i]);
        av[0][i] = r[0][i];
    }
    for(i=1; i<=np; i++)
    for(j=1; j<=nr;  j++)
            all[i][j] = ne[i][j] = max[i][j] =  w[i]=0;
    printf("Enter the allocation matrix");
    for(i=1; i<=np; i++)
    { for(j=1; j<=nr; j++)
        { scanf("%d", &all[i][j]);
              av[0][j] = av[0][j] - all[i][j];
        }
    }
    printf("Enter the maximum matrix");
```

```
     for(i=1; i<=np; i++)
    { for(j=1; j<=nr; j++)
        {
                scanf("%d",&max[i][j]);
        }
    }
    for(i=1; i<=np; i++)
    { for(j=1; j<=nr; j++)
        { ne[i][j] = max[i][j] - all[i][j];
        }
    }
    for(i=1; i<=np; i++)
    { printf("process P%d", i);
        for(j=1; j<=nr; j++)
        {
            printf("\n allocated %d\t",all[i][j]);
            printf("maximum  %d\t",max[i][j]);
            printf("need %d\t",ne[i][j]);
        }
        printf("\n________________________________________\n");
    }
    printf("\nAvailability ");
    for(i=1; i<=nr; i++)
        printf("R%d %d\t", i, av[0][i]);
        printf("\n_______________ ");
        printf("\n safe sequence");
    for(count=1; count<=np; count++)
    { for(i=1; i<=np; i++)
      { cnt = 0;
            for(j=1; j<=nr; j++)
            {
                if(ne[i][j] <= av[0][j] && w[i]==0)
                cnt++;
            }
            if(cnt == nr)
            {
                k++;
                safe[k] = i;
                for(l=1; l<=nr; l++)
                    av[0][l] = av[0][l] + all[i][l];
                    printf("\n P%d ",safe[k]);
                    printf("\t Availability ");
                    for(l=1; l<=nr; l++)
                    printf("R%d %d\t", l, av[0][l]);
                    w[i]=1;
            }
        }
    }
    getch();
}
```

Output:

```
   $ cc deadlock.c
$ ./a.out
      Enter the number of processes in a system 3
      Enter the number of resources in a system 3
      Enter no. of instances of resource R1 10
      Enter no. of instances of resource R2 7
      Enter no. of instances of resource R3  7

      Enter the allocation matrix
      3 2 1
      1 1 2
      4 1 2

      Enter the maximum matrix
      4 4 4
      3 4 5
      5 2 4

      process P1
        allocated 3          maximum 4          need 1
        allocated 2          maximum 4          need 2
        allocated 1          maximum 4          need 3
      ________________________________
      process  P2
        allocated 1        maximum   3          need 2
        allocated 1        maximum   4          need 3
        allocated 2        maximum   5          need 3
      ________________________________
      process P3
        allocated 4        maximum   5          need 1
        allocated 1        maximum   2          need 1
        allocated 2        maximum   4          need 2
      ________________________________

      Availability R1 2      R2 3        R3 2
        safe sequence
      ________________________________

        P3          Availability R1 6              R2 4      R3 4
        P1          Availability R1 9              R2 6      R3 5
        P2          Availability R1 10             R2 7      R3 7
```

Result:

Thus banker’s algorithm for dead lock avoidance was executed successfully.

Ex. No: 8 **DEAD LOCK DETECTION**

Date:

Aim:

To check whether the process and their request for resources are in a deadlocked state or safe state.

Algorithm:

1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vector W to equal the Available vector.
3. Find an index i such that process i is currently unmarked and the ith row of Q is less than or equal to W . That is, Qik ... Wk, for 1 ... k ... m. If no such row is found, terminate the algorithm.
4. If such a row is found, mark process i and add the corresponding row of the allocation matrix to W. That is, set Wk = Wk + Aik, for 1 ... k ... m. Return to step 3.

Program:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n, r;
void input();
void show();
void cal();
main()
{
      int i,j;
      printf("Deadlock Detection Algorithm\n");
      input();
      show();
      cal();
      getch();
}
void input()
{
      int i,j;
      printf("Enter the no of Processes\t");
      scanf("%d",&n);
      printf("Enter the no of resource instances\t");
      scanf("%d", &r);
      printf("Enter the Max Matrix\n");
      for(i=0; i<n; i++)
      for(j=0; j<r;  j++)
               scanf("%d", &max[i][j]);
       printf("Enter the Allocation Matrix\n");
```

```
    for(i=0; i<n; i++) for(j=0; j<r;
        j++) scanf("%d",
        &alloc[i][j]);
    printf("Enter the available Resources\n");
    for(j=0;j<r;j++)
        scanf("%d",&avail[j]);
}
void show()
{
    int i, j;
    printf("Process\t Allocation\t Max\t Available\t");
    for(i=0; i<n; i++)
    {
        printf("\nP%d\t", i+1);
        for(j=0; j<r; j++)
        {
            printf("%d ", alloc[i][j]);
        }
        printf("\t");
        for(j=0; j<r; j++)
        {
            printf("%d ", max[i][j]);
        }
        printf("\t");
        if(i == 0)
        {
            for(j=0; j<r; j++)
            printf("%d ", avail[j]);
        }
    }
}
void cal()
{
    int finish[100], temp, need[100][100], flag=1, k, c1=0;
    int dead[100];
    int safe[100];
    int i, j;
    for(i=0; i<n; i++)
    {
        finish[i] = 0;
    }
    /*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++)
                        {
                            avail[k] += alloc[i][j];
                            finish[i]=1;
                            flag=1;
                        }
                        if(finish[i] == 1)
                        {
                            i=n;
                        }
    j = 0;          }}}}}
    flag = 0;
    for(i=0; i<n; i++)
    {
        if(finish[i] == 0)
        {
            dead[j] = i;
            j++;
            flag = 1;
        }
    }
    if(flag ==  1)
    {
    printf("\n\n System is in Deadlock and the Deadlock process are\n");
        for(i=0;i<n;i++)
        {
            printf("P%d\t", dead[i]);
        }
   }
   else
   {
        printf("\n No Deadlock Occurs");
   }
}
```

Output:

```
********** Deadlock Detection Algorithm ************
Enter the no  of Processes 3
Enter the no of resource instances 3
Enter the Max Matrix
3 6 0
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 0          1 2 0
 P2                 2 0 3                  4 3 3
 P3                 1 2 4                  3 4 4

System is in Deadlock and the Deadlock processes are P0  P1  P2
```

Result:

Thus using given state of information deadlocked processe s w e r e determinedsuccessfully.

Ex. No: 9 **THREADING**

Date:

Aim:

To demonstrate the concept of threading and synchronization using mutual exclusion (mutex).

Description:

- Thread synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as critical section.
- Processes access to critical section is controlled by using synchronization techniques.
- When one thread starts executing the critical section (serialized segment of the program) the other thread should wait until the first thread finishes.
- If proper synchronization techniques are not applied, it may cause a race condition where the values of variables may be unpredictable.
- A Mutex is a lock that we set before using a shared resource and release after using it.
- When the lock is set, no other thread can access the locked region of code. So this ensures a synchronized access of shared resources in the code.

Algorithm:

1. Create two threads.
2. Let the threads share a common resource, say counter.
3. Even if thread2 si scheduled to start while thread was not done, access to shared resource is not done as it is locked by mutex.
4. Once thread1 completes, thread2 starts execution.
5. Stop.

Program:

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

```
        for(i=0; i<(0xFFFFFFFF);i++)
        printf("\n Job %d has finished\n", counter);
        pthread_mutex_unlock(&lock);
        return NULL;
    }
    main()
    {
        int i = 0;
        int error;
        if (pthread_mutex_init(&lock, NULL) != 0)
        {
            printf("\n mutex init has failed\n");
            return 1;
        }
        while(i < 2)
        {
            err = pthread_create(&(tid[i]), NULL, &trythis, NULL);
            if (error != 0)
            printf("\nThread can't be created :[%s]", strerror(error));
            i++;
        }
        pthread_join(tid[0], NULL);
        pthread_join(tid[1], NULL);
        pthread_mutex_destroy(&lock);
        return 0;
    }
```

Output:

```
    $ cc filename.c -lpthread
$ ./a.out
    Job 1 started
    Job 1 finished
    Job 2 started
    Job 2 finished
```

Result:

Thus c program for implementing thread synchronization using mutex lock was executed successfully.

Ex. No: 10 **PAGING TECHNIQUE**

Date:

Aim:

To implement the concept of paging i.e physical address of a given page to be determined using page table.

Algorithm:

1. Get process size.
2. Compute number of pages available and display it.
3. Get relative address.
4. Determine the corresponding page.
5. Display page table.
6. Display the physical address.

Program:

```
#include <stdio.h>
#include <math.h>
main()
{
int size, m, n, pgno, pagetable[3]={5,6,7}, i, j, frameno;
double m1;
int ra=0, ofs;
printf("Enter process size (in KB of max 12KB):");
scanf("%d", &size);
m1 = size / 4;
n = ceil(m1);
printf("Total No. of pages: %d", n);
printf("\nEnter relative address (in hexa) \n");
scanf("%d",  &ra);
pgno = ra / 1000;
ofs = ra % 1000;
printf("page no=%d\n", pgno);
printf("page table");
for(i=0;i<n;i++)
printf("\n %d [%d]", i, pagetable[i]);
frameno = pagetable[pgno];
printf("\nPhysical address: %d%d", frameno, ofs);
}
```

Output:

```
$ cc paging.c
$ ./a.out
Enter process size (in KB of max 12KB):12
Total No. of pages: 3
Enter relative address (in hexa): 2643
page no=2
page table
0 [5]
1 [6]
2 [7]
Physical address: 7643
```

Result:

Thus physical address for the given logical address is determined using pagingtechnique**.**

Ex. No: 11a MEMORY ALLOCATION USING [First Fit]

Date:

Aim:

To allocate memory requirements for processes using first fit allocation strategy.

Memory Management:

- The first-fit, best-fit, or worst-fit strategy is used to select a free hole from the set of available holes.

First fit:

- Allocate the first hole that is big enough.
- Searching starts from the beginning of set of holes.

Algorithm:

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

Program:

```
/* First fit allocation - ffit.c */
#include <stdio.h>
struct process
{ int size;
int flag;
int holeid;
} p[10];
struct hole
{ int size;
 int actual;
} h[10];
```

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

Output:

```
   $ cc ffit.c
$ ./a.out
   Enter the number of Holes : 5
   Enter size for hole H0 : 100
   Enter size for hole H1 : 500
   Enter size for hole H2 : 200
   Enter size for hole H3 : 300
   Enter size for hole H4 : 600

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

   First    fit

   Process PSize      Hole
     P0        212         H1
     P1        417         H4
     P2       112         H2
     P3       426    Not allocated

   Hole    Actual Available
    H0       100       100
    H1       500       176
    H2       200       200
    H3       300       300
    H4       600       183
```

Result:

Thus memory allocation using first fit method was implemented and executed successfully.

Ex. No: 11b MEMORY ALLOCATION USING [Best Fit]

Date:

Aim:

To allocate memory requirements for processes using best fit allocation.

Best fit:

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

Algorithm:

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

Program:

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

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

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

Output:

```
   $ cc bfit.c
$ ./a.out
     Enter the number of Holes: 5
     Enter size for hole H0 : 100
     Enter size for hole H1 : 500
     Enter size for hole H2 : 200
     Enter size for hole H3 : 300
     Enter size for hole H4 : 600

     Enter number of process:  4
     Enter the size of process P0 : 212
     Enter the size of process P1 : 417
     Enter the size of process P2 : 112
     Enter the size of process P3 :  426
          Best fit

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

         H1        500            83
         H3        300            88
         H2        200            88
         H0        100           100
         H4        600           174
```

Result:

Thus memory allocation using best fit method was implemented and executed successfully.

Ex. No: 11b **MEMORY ALLOCATION USING [Worst Fit]**
Date:

Aim:

To allocate memory requirements for processes using Worst fit allocation stratedy.

Program:

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

}
```

OUTPUT:

```
Memory Management Scheme - Worst Fit
Enter the number of blocks:5
Enter the number of files:3
Enter the size of the blocks:-
Block 1:34
Block 2:24
Block 3:35
Block 4:39
Block 5:45
Enter the size of the files:-
File 1:21
File 2:12
File 3:34

  File_no:      File_size :     Block_no:     Block_size:    Fragement
  1                     21                    5                    45                    24
  2                     12                    4                    39                    27
  3                     34                    3                    35                    1
```

Result:

Thus memory allocation using Worst fit method has implemented and executed successfully.

Ex. No: 12a PAGE REPLACEMENT - FIFO (First In First Out)
Date:

Aim:

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

FIFO:

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

Algorithm:

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

Program:

```
#include <stdio.h>
main()
{
    int i, j, l, rs[50], frame[10], nf, k, avail, count=0;
    printf("Enter length of reference string : ");
    scanf("%d", &l);
    printf("Enter reference string :\n");
    for(i=1; i<=l; i++)
    scanf("%d", &rs[i]);
    printf("Enter number of frames : ");
    scanf("%d", &nf);
    for(i=0; i<nf; i++)
        frame[i] = -1;
        j = 0;
    printf("\n Ref. str Page frames");
    for(i=1; i<=l; i++)
    { printf("\n%4d\t", rs[i]);
        avail = 0;
        for(k=0; k<nf; k++)
        if(frame[k] == rs[i])
                avail = 1;
        if(avail == 0)
        { frame[j] = rs[i];
            j = (j+1) % nf;
            count++;
            for(k=0;    k<nf;    k++)
            printf("%4d",frame[k]);
        }}
    printf("\n\nTotal no. of page faults : %d\n",count);
}
```

Output:

```
  $ cc fifo.c
$ ./a.out
    Enter length of reference string: 20
    Enter reference string:
    1 2 3 4 2 1 5 6 2 1 2 3 7 6 3
    Enter number of frames: 5
         Ref          Page frames
          .
         str
          1     1     -1    -1   -1    -1
          2     1      2    -1   -1    -1
          3     1      2     3   -1    -1
          4     1      2     3    4    -1
          2
          1
          5     1      2     3    4     5
          6     6      2     3    4     5
          2
          1     6      1     3    4     5
          2     6      1     2    4     5
          3     6      1     2    3     5
          7     6      1     2    3     7
          6
          3
    Total no. of page faults: 10
```

Result:

Thus page replacement using FIFO algorithm was implemented and executed successfully.

Ex. No: 12b PAGE REPLACEMENT - LRU (Least Recently Used)
Date:

Aim:

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

LRU:

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

Algorithm:

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

Program:

```
/* LRU page replacement - lrupr.c */
#include <stdio.h>
int arrmin(int[], int);
main()
{
    int i,j,len,rs[50],frame[10],nf,k,avail,count=0;
    int access[10], freq=0, dm;
    printf("Length of Reference string : ");
    scanf("%d", &len);
    printf("Enter reference string :\n");
    for(i=1; i<=len; i++)
        scanf("%d", &rs[i]);
        printf("Enter no. of frames : ");
    scanf("%d", &nf);
    for(i=0; i<nf; i++)
        frame[i] = -1;
        j = 0;
    printf("\nRef.  Str     Page frames");
    for(i=1; i<=len; i++)
    {
        printf("\n%4d\t", rs[i]);
        avail = 0;
        for(k=0; k<nf; k++)
        {
            if(frame[k] == rs[i])
            {
                avail  =  1;
                access[k] = ++freq;
                break;
            }}
        if(avail == 0)
        { dm = 0;
            for(k=0; k<nf; k++)
            { if(frame[k] == -1)
                    dm = 1;
                    break;
            }
            if(dm ==  1)
            {     frame[k] = rs[i];
                access[k] = ++freq;
                count++;
            }
            else
            {     j = arrmin(access, nf);
                frame[j] = rs[i];
                access[j] = ++freq;
                count++;
            }
```

```
                    for(k=0; k<nf; k++)
                    printf("%4d", frame[k]);
                }}
            printf("\n\nTotal no. of page faults : %d\n", count);
        }
        int arrmin(int a[], int n)
        {
          int i, min = a[0];
          for(i=1; i<n; i++)
                if (min >  a[i])
                    min = a[i];
            for(i=0; i<n;  i++)
                if (min == a[i])
                    return i;
        }
```

Output:

```
  $ cc lrupr.c
$ ./a.out
  Length of Reference string: 15
  Enter reference string:
  1 2 3 4 2 1 5 6 2 1 2 3 7 6 3
  Enter no. of frames: 5

      Ref. str          Page frames
         1          1    -1    -1    -1    -1
         2          1     2    -1    -1    -1
         3          1     2     3    -1    -1
         4          1     2     3     4    -1
         2
         1
         5          1     2     3     4     5
         6          1     2     6     4     5
         2
         1
         2
         3          1     2     6     3     5
         7          1     2     6     3     7
         6
         3
Total no. of page faults: 8
```

Result:

Thus page replacement using LRU algorithm was implemented and executed successfully.

Ex. No: 13a File Organization - Single-Level Directory

Date:

Aim:

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

Algorithm:

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

Program:

```
 #include  <stdio.h>
 #include <stdlib.h>
 #include <conio.h>
 struct
 {
     char dname[10];
     char fname[25][10];
     int fcnt;
 }dir;
main()
{
int i, ch;
char f[30];
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{ printf("\n 1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5. Exit\n
Enter your choice--");
scanf("%d",&ch);
switch(ch)
{ case 1:
printf("\n Enter the name of the file -- ");
scanf("%s", dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2:
printf("\n Enter the name of the file -- ");
scanf("%s", f);
for(i=0; i<dir.fcnt; i++)
{
if(strcmp(f, dir.fname[i]) == 0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i], dir.fname[dir.fcnt-1]);
break;
}
}
if(i == dir.fcnt)
printf("File %s not found", f);
else
dir.fcnt--; break;

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

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

Output:

```
   $ cc sld.c
$ ./a.out
      Enter name of directory -- CSE
       1. Create File 2. Delete File 3. Search File
       4. Display Files      5. Exit
      Enter your choice -- 1
       Enter the name of the file -- fcfs

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

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

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

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

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

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

Result:

Thus files organization based on single level directory structure technique was implemented and executed successfully.

Ex. No: 13b File Organization - Two-level directory

Date:

Aim:

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

Algorithm:

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

Program:

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

```
dcnt++;
printf("Directory created");
break;

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

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

```
case 4:
printf("\nEnter name of the directory -- ");
scanf("%s", d);
for(i=0; i<dcnt; i++)
{
if(strcmp(d,dir[i].dname) == 0)
{
printf("Enter the name of the file -- ");
scanf("%s", f);
for(k=0; k<dir[i].fcnt; k++)
{
if(strcmp(f, dir[i].fname[k]) ==  0)
{
printf("File %s is found ", f);
goto jmp1;
}}
printf("File %s not found", f);
goto jmp1;
}}
printf("Directory %s not found", d); jmp1: break;

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

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

Output:

```
   $ cc tld.c
$ ./a.out
   1. Create Directory      2. Create File    3. Delete File
   4. Search File           5. Display        6. Exit
   Enter your choice -- 1
   Enter name of directory – CSE
   Directory created

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

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

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

   1. Create Directory      2. Create File    3. Delete File
   4. Search File           5. Display        6. Exit
   Enter your choice -- 5
     Directory      Files
        CSE         kowshik
        ECE         amruth

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

Result:

Thus user files have been stored in their respective directories and retrieved easily using two level directory structure of file organization.

Ex. No: 14a F I L E ALLOCATION [Sequential]

Date:

Aim:

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

File Allocation:

The three methods of allocating disk space are:

1. Sequential allocation
2. Linked allocation
3. Indexed allocation

Sequential Allocation:

- ➢ Each file occupies a set of Sequential block on the disk.
- ➢ The number of disk seeks required is minimal.
- ➢ The directory contains address of starting block and number of Sequential block (length) occupied.
- ➢ Supports both sequential and direct access.
- ➢ First / best fit is commonly used for selecting a hole.

Algorithm:

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

Program:

```
/* Sequential Allocation - Seqalloc.c */
#include <stdio.h>
#include <string.h>
int num=0, length[10], start[10];
char fid[20][4], a[20][4];
void directory()
{
    int i;
    printf("\n File Start Length\n");
    for(i=0; i<num; i++)
        printf("%-4s %3d %6d\n",fid[i],start[i],length[i]);
}
```

```
void display()
{      int i;
       for(i=0; i<20; i++)
       printf("%4d",i);
       printf("\n");
       for(i=0; i<20;  i++)
              printf("%4s", a[i]);
}
main()
{      int i,n,k,temp,st,nb,ch,flag;
       char id[4];
       for(i=0; i<20; i++)
       strcpy(a[i], "");
       printf("Disk space before allocation:\n");
       display();
       do
       {     printf("\nEnter File name (max 3 char) : ");
              scanf("%s", id);
              printf("Enter start block : ");
              scanf("%d", &st);
              printf("Enter no. of blocks : ");
              scanf("%d", &nb);
              strcpy(fid[num], id);
              length[num] = nb;
              flag = 0;
              if((st+nb) > 20)
              { printf("Requirement exceeds range\n");
                 continue;
              }
              for(i=st; i<(st+nb); i++)
                     if(strcmp(a[i], "") != 0)
                            flag = 1;
              if(flag == 1)
              {
                     printf("Sequential allocation not possible.\n");
                     continue;
              }
              start[num] = st;
              for(i=st; i<(st+nb);i++)
                     strcpy(a[i], id);;
              printf("Allocation done\n");
                     num++;
              printf("\n Any more allocation (1. yes / 2. no)? : ");
              scanf("%d", &ch);
       } while (ch == 1);
       printf("\n\t\t\t Sequential Allocation\n");
       printf("Directory:");
       directory();
       printf("\nDisk space after allocation:\n");
       display(); }
```

Output:

```
   $ cc Seqalloc.c
$ ./a.out
      Disk space before allocation:
          0    1    2    3    4    5    6    7    8    9 10  11  12  13  14  15  16  17  18  19
      Enter File name (max 3 char) : cp
      Enter start block : 14
      Enter no. of blocks : 3
      Allocation done
      Any more allocation (1. yes / 2. no)? : 1

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

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

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

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

                              Sequential  Allocation
      Directory:
      File Start Length
      cp      14        3
      tr      10        3
      mv       0        2
      Disk space after allocation:
          0    1    2     3   4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19
         mv   mv                                           tr   tr   tr        cp   cp   cp
```

Result:

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

Ex. No: 14b **FILE ALLOCATION [Linked]**

Date:

Aim:

To implement file allocation on free disk space using linked file allocation strategy.

Linked Allocation:

- Each file is a linked list of disk blocks.
- The directory contains a pointer to first and last blocks of the file.
- The first block contains a pointer to the second one, second to third and so on.
- File size need not be known in advance, as in contiguous allocation.
- No external fragmentation.
- Supports sequential access only.

Indexed Allocation:

- In indexed allocation, all pointers are put in a single block known as index block.
- The directory contains address of the index block.
- The ith entry in the index block points to ith block of the file.
- Indexed allocation supports direct access.
- It suffers from pointer overhead, i.e wastage of space in storing pointers.

Algorithm:

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

Program:

```
#include   <stdio.h>
#include  <conio.h>
#include <string.h>
main()
{
     static int b[20], i, j, blocks[20][20];
     char F[20][20], S[20], ch;
     int sb[20], eb[20], x, n;
     printf("\n Enter no. of Files :");
     scanf("%d",&n);
```

```
            for(i=0;i<n;i++)
            {
                  printf("\n Enter file %d name ::",
                  i+1);scanf("%s", &F[i]);
                  printf("\n Enter No. of blocks:",
                  i+1);scanf("%d",&b[i]);
            }
            for(i=0;i<n;i++)
            {
                  printf("\n Enter Starting block of file%d::",i+1);
                  scanf("%d", &sb[i]);
                  printf("\n Enter blocks for file%d::\n",
                  i+1);for(j=0; j<b[i]-1;)
                  {
                        printf("\n Enter the %dblock ::",
                        j+2);scanf("%d", &x);
                        if(b[i] != 0)
                        {

                        }
                        else

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

Output:

$ cc linked.c

$ ./a.out

```
 Enter no. of Files :: 2

 Enter file 1 name :: fcfs
 Enter No. of blocks::3

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

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

 Enter Starting block of file2::2
 Enter blocks for file2::
 Enter the 2block :: 6 Enter
the Filename ::fcfs
Fname Bsize   Start    Blocks
--------------------------------------------
  fcfs      3        8      8->3->5
--------------------------------------------
```

Result:

Thus blocks for file were allocated using linked allocation method and executed successfully.

EX.NO:14C **FILE ALLOCATION [Indexed]**

DATE:

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<50;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]);for(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);
if(c==1)
goto x;
                else
                         exit();
getch();
}
```

OUTPUT:

```
Enter index block 9
Enter no of files on index 3 1 2 3
Allocated
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.

Ex. No: 15a FCFS Disk Scheduling

Date:

Aim:

To implement FCFS disk scheduling algorithms using C program.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.

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

3. Increment the total seek count with this distance.

4. Currently serviced track position now becomes the new head position.

5. Go to step 2 until all tracks in request array have not been serviced.

Program:

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

Input:

```
Enter no. of tracks:9
Enter track position: 55     58        60        70        18        90        150      160 184
```

Output:

```
Tracks traversed    Difference between tracks

        55                    45
        58                    3
        60                    2
        70                    10
        18                    52

        90                    72
        150                   60
        160                   10
        184                   24

        Average header movements:30.888889
```

Result:

Thus FCFS disk scheduling algorithm is implemented and executed successfully.

Ex. No: 15b SCAN DISK SCHEDULING ALGORITHM

Date:

Aim:

To implement SCAN disk scheduling algorithms using C program.

Algorithm:

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

Program:

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

Input:

```
Enter no.of tracks:9
Enter track position: 55    58       60       70       18       90       150     160 184
```

Output:

```
Tracks traversed   Difference between tracks

            150           50
            160           10
            184           24

            90            94
            70            20
            60            10
            58            2
            55            3
            18            37

Average header movements: 27.77
```

Result:

Thus SCAN disk scheduling algorithm is implemented and executed successfully.

Ex. No: 15c C-SCAN DISK SCHEDULING ALGORITHM

Date:

Aim:

To implement C-SCAN disk scheduling algorithms using C program.

Algorithm:

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

Program:

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

```
 temp=t[j]; t[j]=t[j+1];
t[j+1]=temp
}
p=0;
while(t[j]!=tot-1)
{
atr[p]=t[j]; j++;
p++;
}
atr[p]=t[j]; p++;
i=0;
while(p!=(n+3) && t[i]!=t[h])
{
atr[p]=t[i]; i++; p++;
}

for(j=0;j<n+2;j++)
{
if(atr[j]>atr[j+1])
d[j]=atr[j]-atr[j+1];
else

d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("total header movements%d",sum);
printf("avg is %f",(float)sum/n);
getch();
}
```

Input:

```
Enter the track position : 55     58     60     70     18     90     150     160     184
Enter starting position : 100
```

Output:

```
Tracks traversed    Difference Between tracks
      150                 50
      160                 10
      184                 24
      18                  240
      55                  37
      58                  3
      60                  2
      70                  10
      90                  20
Average seek time: 35.7777779
```

Result:

Thus C-SCAN disk scheduling algorithm is implemented and executed successfully.

Ex No: 16 **INSTALLATION OF UBUNTU**

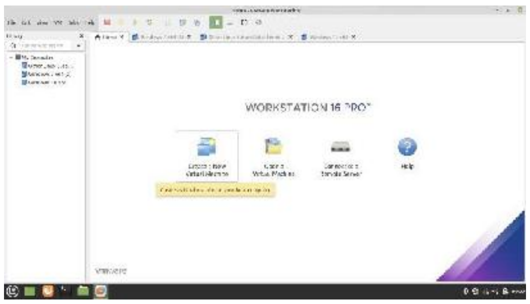
DATE:

AIM:

To install and implement Ubuntu using vmware.

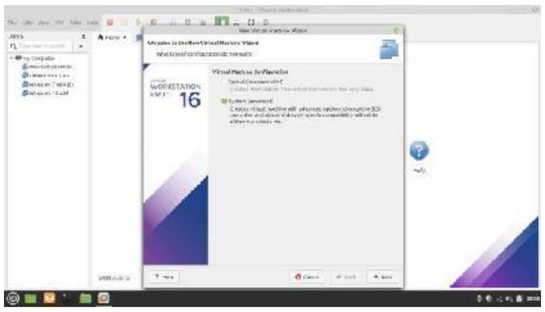
Fier up VMWare Workstation

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.



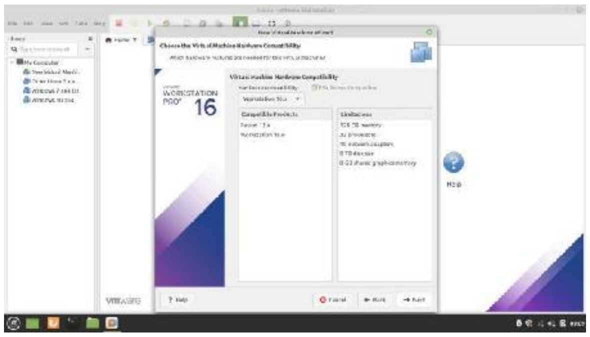
1. Select Custom Configuration Wizard

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.



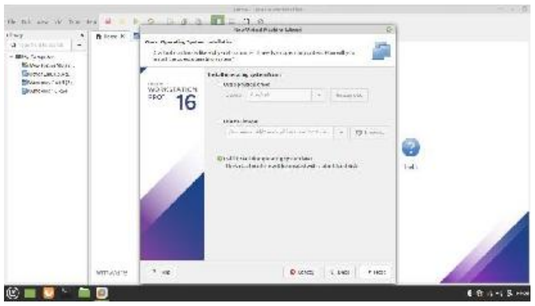
2. **Select Virtual Machine Hardware Compatibility**

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

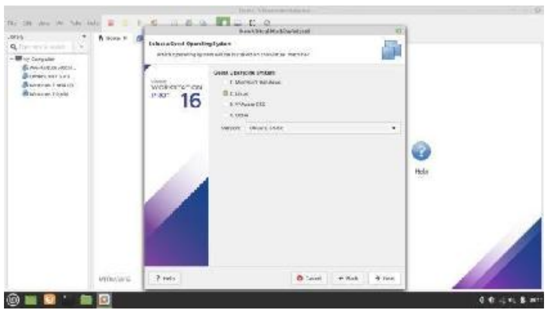


3. **Select the Operating System Media**

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

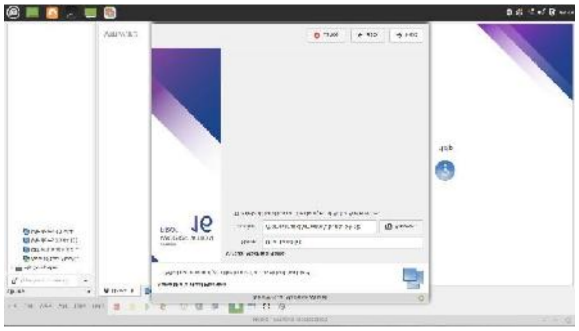


4. **Select Guest Operating System**



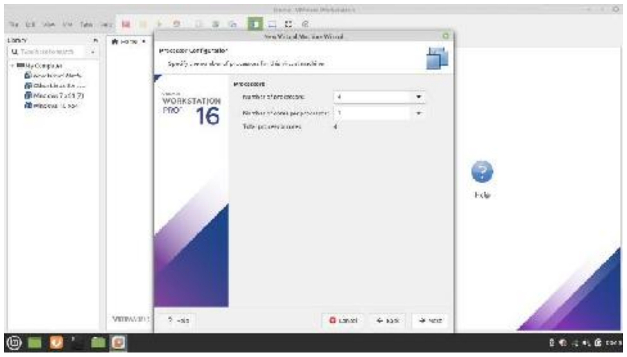
1. **Name the Virtual Machine Name and location**

Type a name and give the location details.



5. **Allocate the Processors**

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



Processor Configuration

Specify the number of processors for this virtual machine.

Processors

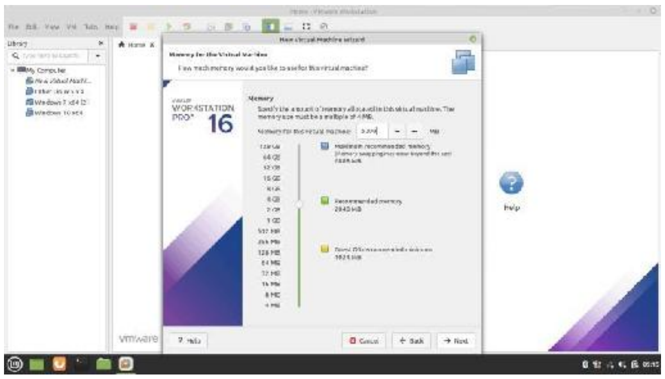
Number of processors: 4

Number of cores per processor: 1

Total processor cores: 4

6. **Allocate the Memory for Virtual Machine**

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



Memory for the Virtual Machine

How much memory would you like to use for this virtual machine?

Memory

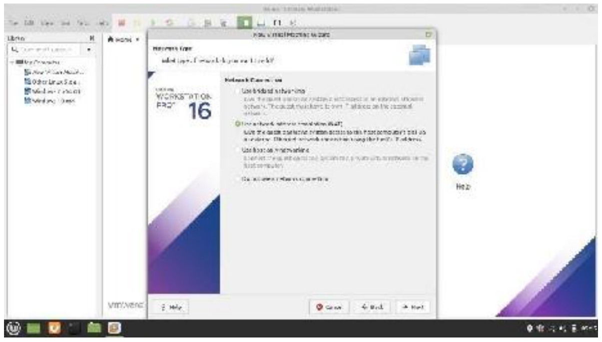
Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: [illegible] MB

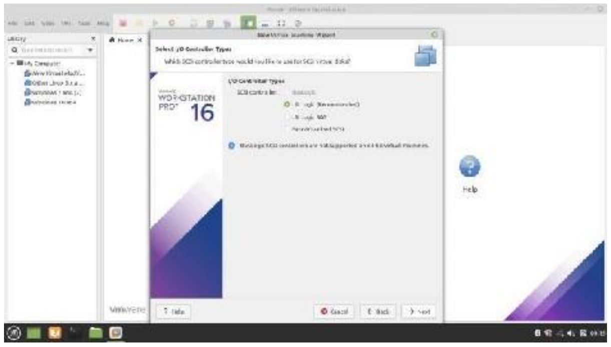
Recommended memory [illegible] MB

7. **Choose the Network Configuration**

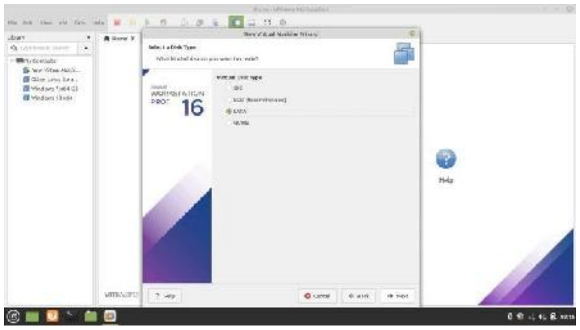
Select any one of the network configurations as per your requirement.



8. **Select the I/O Controller Type**



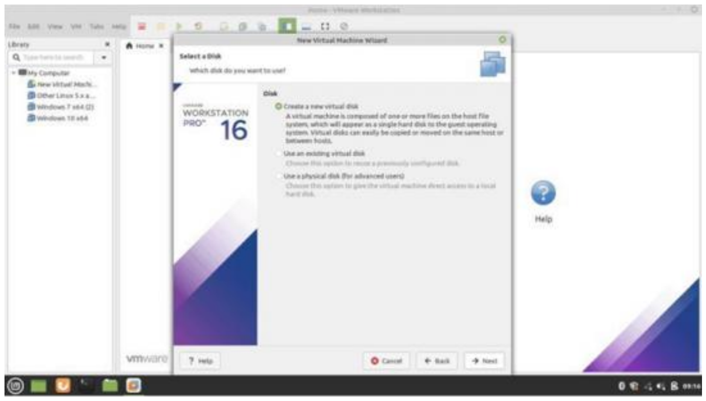
9. **Select Disk Type**



IDE
SCSI (Recommended)
[✓ SATA]
NVMe

10. Select Virtual Disk

Select the Virtual Disk if you have or create one.



New Virtual Machine Wizard

Select a Disk

Which disk do you want to use?

Disk

(●) Create a new virtual disk

A virtual machine is composed of one or more files on the host file system, which will appear as a single hard disk to the guest operating system. Virtual disks can easily be copied or moved on the same host or between hosts.

( ) Use an existing virtual disk

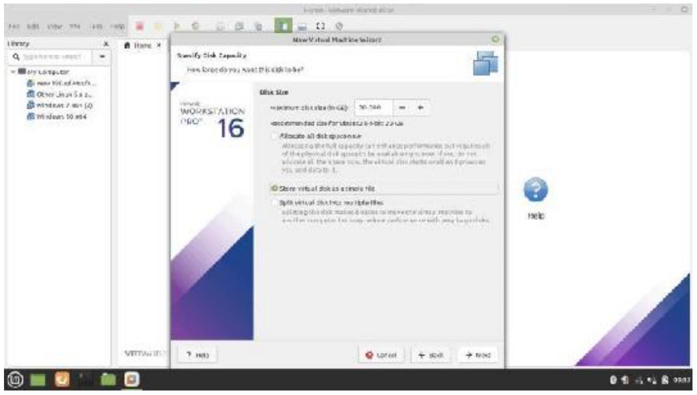
Choose this option to reuse a previously configured disk.

( ) Use a physical disk (for advanced users)

Choose this option to give the virtual machine direct access to a local hard disk.

11. Select Disk Capacity

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



New Virtual Machine Wizard

Specify Disk Capacity

How large do you want this disk to be?

Disk Size

Maximum disk size (in GB): [illegible]

Recommended size for Ubuntu 64-bit: [illegible]

[ ] Allocate all disk space now

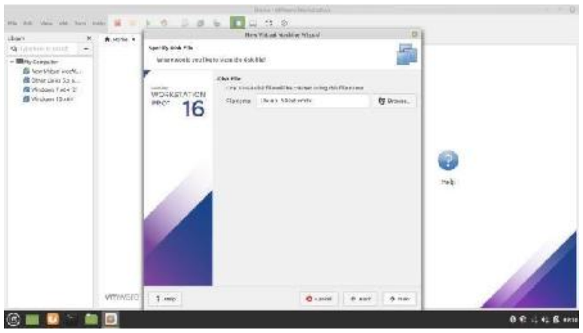
[illegible]

(●) Store virtual disk as a single file

( ) Split virtual disk into multiple files

[illegible]

12. Specify Virtual Disk File



New Virtual Machine Wizard

Specify Disk File

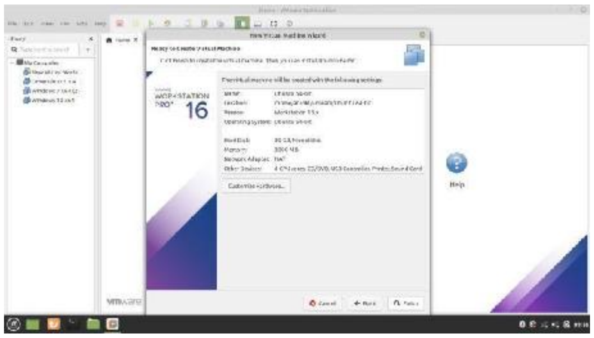
Where would you like to store the disk file?

Disk File

[illegible]

File name: [illegible]

13. Create Virtual Machine



New Virtual Machine Wizard

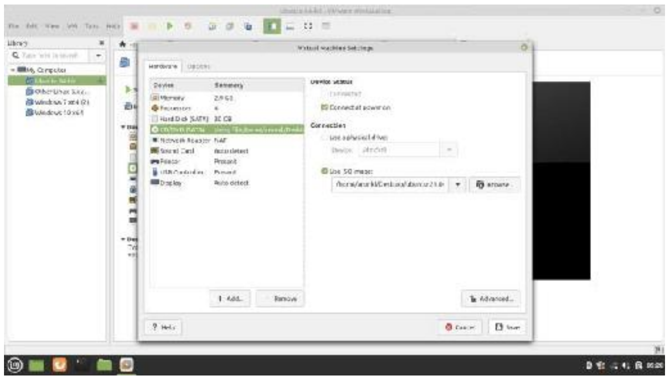
Ready to Create Virtual Machine

The virtual machine will be created with the following settings:

Name:	Ubuntu 64-bit
Location:	[illegible]
Version:	Workstation 15.x
Operating System:	Ubuntu 64-bit
Hard Disk:	30 GB, Preallocated
Memory:	3000 MB
Network Adapter:	NAT
Other Devices:	4 CPU cores, CD/DVD, USB Controller, Printer, Sound Card

14. Supply Ubuntu ISO Image to Virtual Machine

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



Virtual Machine Settings

Device	Summary
Memory	2.9 GB
Processors	4
Hard Disk (SATA)	30 GB
CD/DVD (SATA)	Using file [illegible]
Network Adapter	NAT
Sound Card	Auto detect
Printer	Present
USB Controller	Present
Display	Auto detect

Device status

Connected

Connect at power on

Connection

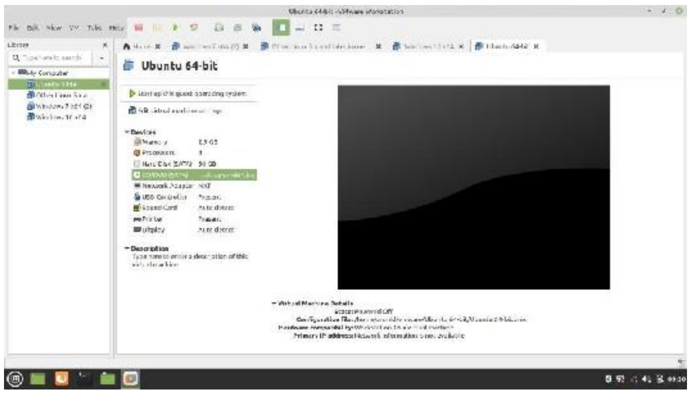
Use a physical drive:

Device: [illegible]

Use ISO image:

[illegible]

15. Install Ubuntu Linux on VMWare Workstation



Ubuntu 64-bit

Start up this guest operating system

Edit virtual machine settings

Devices

Memory	2.9 GB
Processors	4
Hard Disk (SATA)	30 GB
CD/DVD (SATA)	[illegible]
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Description

Type here to enter a description of this virtual machine.

Virtual Machine Details

State: Powered Off

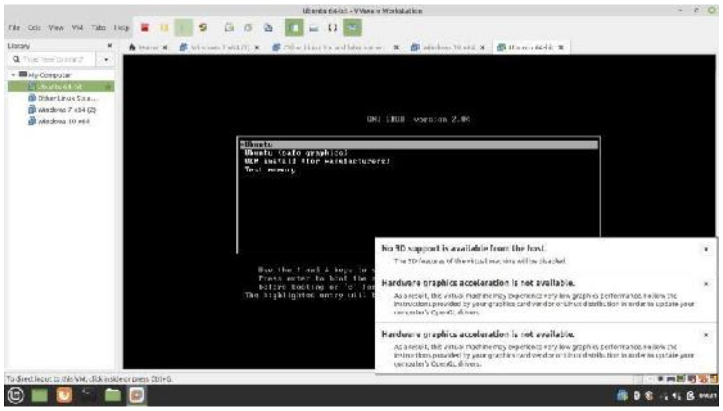
Configuration file: [illegible]

Hardware compatibility: Workstation 16.x virtual machine

Primary IP address: Network information is not available

16. Power On the Virtual Machine

Press the Play button to power on the Virtual Machine.



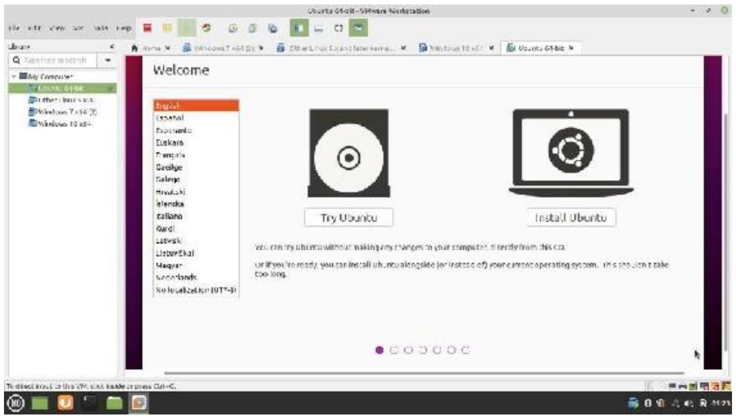
```
GNU GRUB  version 2.04

*Ubuntu
 Ubuntu (safe graphics)
 OEM install (for manufacturers)
 Test memory
```

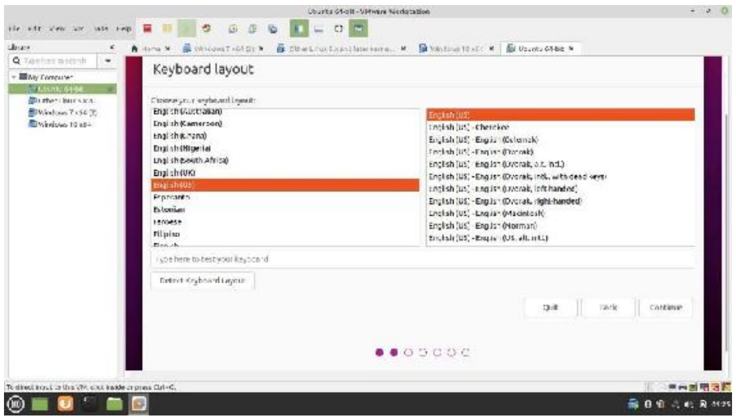
No 3D support is available from the host.
Hardware graphics acceleration is not available.
Hardware graphics acceleration is not available.

17. Welcome Ubuntu Virtual Machine

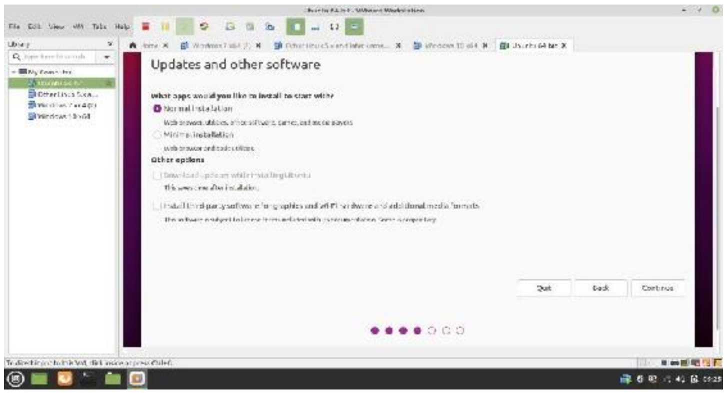
After powering on the Virtual machine, you will be treated with a welcome screen on which you will see two options: Try Ubuntu and Install Ubuntu. Select Try Ubuntu if you want to runUbuntu in live mode. Select Install to continue the installation process.



18. Select Keyboard Layout



19. Software Update and Package Selection in Virtual Machine



Updates and other software

What apps would you like to install to start with?

[✓ Normal installation]
Web browser, utilities, office software, games, and media players

Minimal installation
Web browser and basic utilities

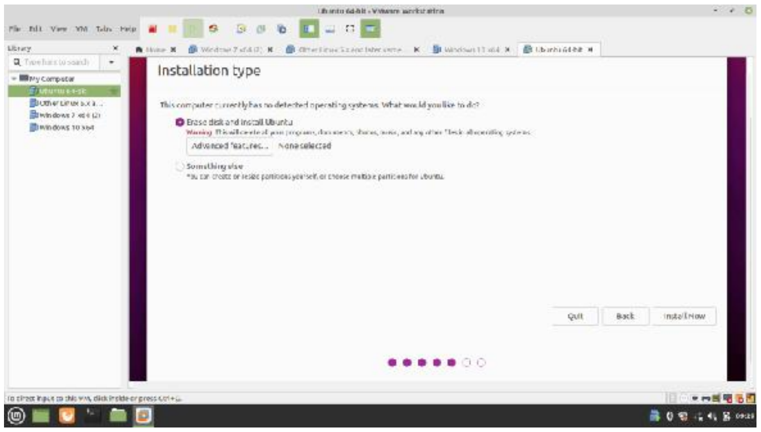
Other options

Download updates while installing Ubuntu
This saves time after installation.

Install third-party software for graphics and Wi-Fi hardware and additional media formats
This software is subject to license terms included with its documentation. Some is proprietary.

20. 1Partition the Disk

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



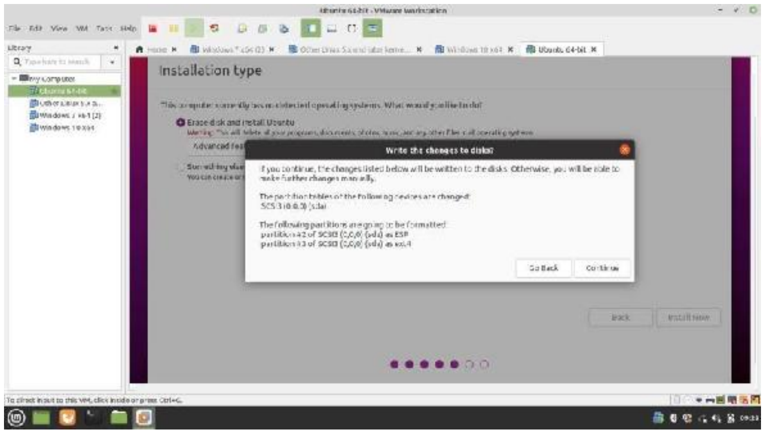
Installation type

This computer currently has no detected operating systems. What would you like to do?

[✓ Erase disk and install Ubuntu]
Warning: This will delete all your programs, documents, photos, music, and any other files in all operating systems.
Advanced features... None selected

Something else
You can create or resize partitions yourself, or choose multiple partitions for Ubuntu.

21. Write Changes to Disk



Write the changes to disks?

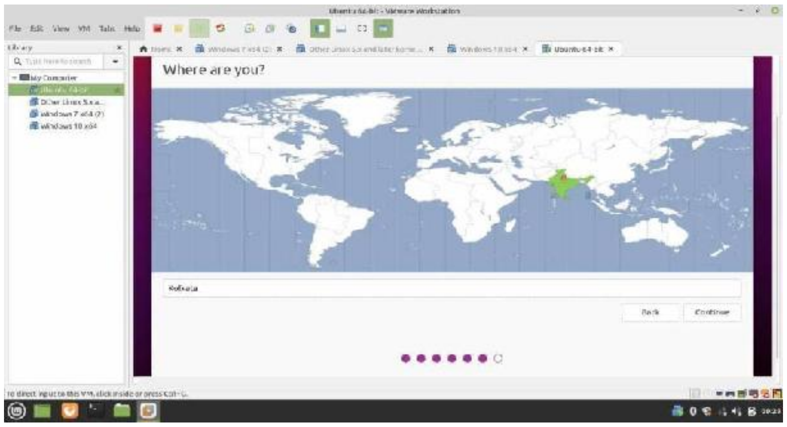
If you continue, the changes listed below will be written to the disks. Otherwise, you will be able to make further changes manually.

The partition tables of the following devices are changed:
SCSI3 (0,0,0) (sda)

The following partitions are going to be formatted:
partition #2 of SCSI3 (0,0,0) (sda) as ESP
partition #3 of SCSI3 (0,0,0) (sda) as ext4

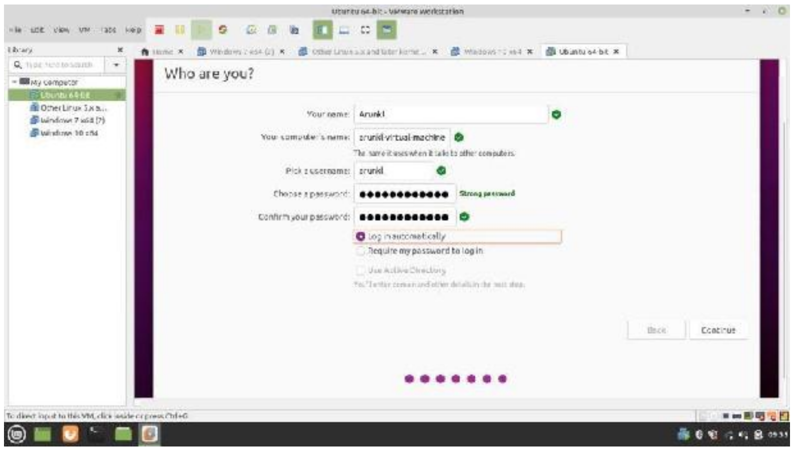
22.22.

23. Select Time Zone


Where are you?

Kolkata

24. Create an Admin Account


Who are you?

Your name: Arunkl

Your computer's name: arunkl-virtual-machine

The name it uses when it talks to other computers.

Pick a username: arunkl

Choose a password: ●●●●●●●●●●●● Strong password

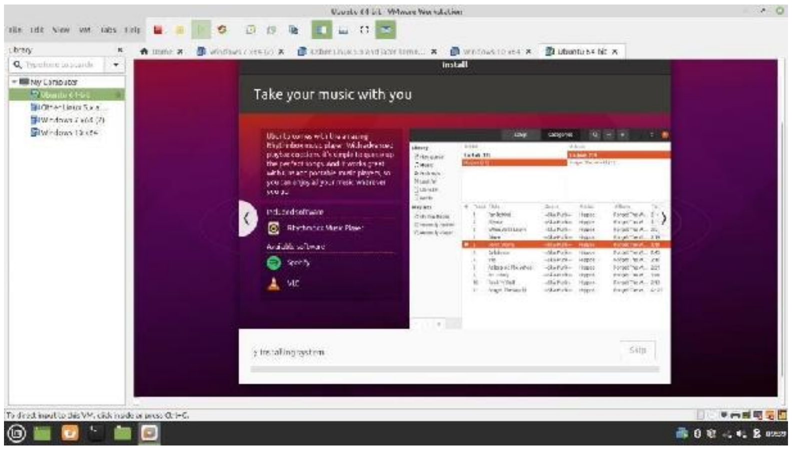
Confirm your password: ●●●●●●●●●●●●

[✓ Log in automatically]

Require my password to log in

Use Active Directory

25. Installation of Ubuntu in Progress

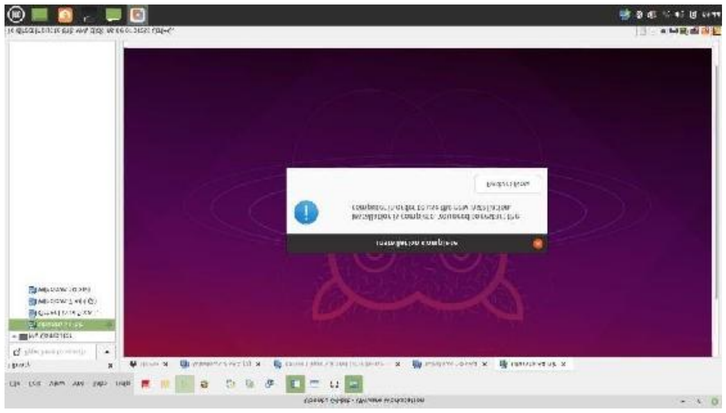

Install

Take your music with you

Installing system

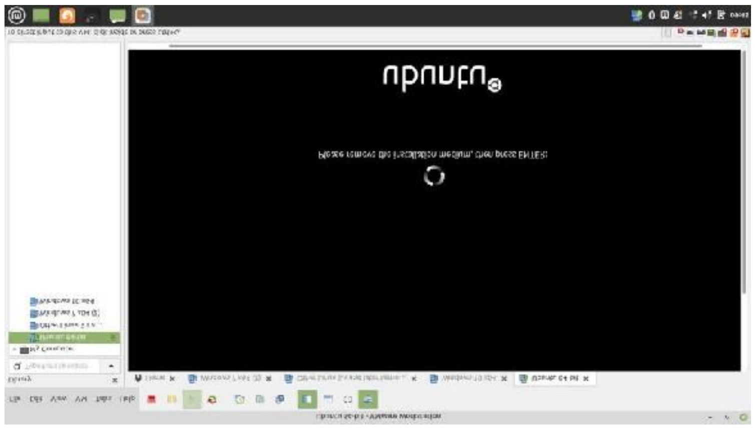
26. Reboot Virtual Machine

Reboot the machine after installation.

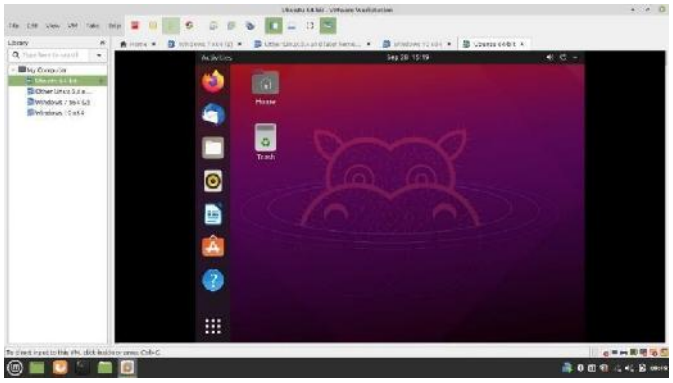


27. Remove the installation media

Remove the installation media before reboot.



28. **Boot Ubuntu**



Result:

Thus, the Ubuntu OS has been successfully installed.