HAZARA UNIVERSITY, MANSEHRA
DEPARTMENT OF CS & IT

Lab Manual
Operating System (CS202)

Course Name	Operating System
Course Code	CS202
Instructor	Miraj Gul
Credit Hours	4(3+1)
Semester	Spring 2024

Lab Task 1

Learning MS-DOS Basics - A Tutorial

Objective: This tutorial gives you an opportunity to try basic MS-DOS commands. By following the procedures in this section, you will learn to:

- View the contents of a directory
- Change from one directory to another
- Create and delete directories
- Change from one drive to another
- Copy files
- Rename files
- Delete files
- Format a device

The Command Prompt

When you first turn on your computer, you will see some cryptic information flash by. MS-DOS displays this information to let you know how it is configuring your computer. You can ignore it for now. When the information stops scrolling past, you'll see the following:

```
C:\>
```

This is called the **command prompt** or **DOS prompt**. The flashing underscore next to the command prompt is called the **cursor**. The cursor shows where the command you type will appear.

Typing a Command

This section explains how to type a command at the command prompt and demonstrates the "Bad command or file name" message.

- **To type a command at the command prompt**

1. Type the following at the command prompt (you can type the command in either uppercase or lowercase letters):

   ```
   nul
   ```

   If you make a typing mistake, press the **BACKSPACE** key to erase the mistake, and then try again.

2. Press **ENTER**.

   You must press **ENTER** after every command you type. The following

   message appears:

   ```
   Bad command or file name
   ```

   The "Bad command or file name" message appears when you type something that MS- DOS does not recognize. Because **nul** is not a valid MS-DOS command, MS-DOS displays the "Bad command or file name" message.

3. Now, type the following command at the command prompt:

```
ver
```

The following message appears on your screen:

```
MS-DOS version 6.22
```

Viewing the Contents of a Directory

In this section, you will view the contents of a directory by using the **dir** command. The **dir** command stands for "**directory**."

- **To view the contents of a directory**

1. Type the following at the command prompt:

```
dir
```

A list similar to the following appears:

```
Volume in drive C is MS-DOS_6.22 Volume Serial Number is lE49-15E2 Directory of
C:\

WINDOWS     <DIR>     09-08-92  10:27p  TEMP     <DIR>
05-15-92 12:09p CONFIG SYS 278 09-23-92 10:50a
COMMAND COM 53014 09-18-92 6:00a
WINA20  386  9349 11-11-91 5:00a DOS      <DIR> 09-
02-92 4:23p AUTOEXEC BAT            290 09-23-92
10:54a

               7 file(s) 62931 bytes
                       8732672 bytes free
```

This is called a ***directory list***. A directory list is a list of all the files and subdirectories that a directory contains. In this case, you see all the files and directories in the main or ***root*** directory of your drive. All the files and directories on your drive are stored in the root directory.

Changing Directories

Look at the list on your screen. All the names that have <DIR> beside them are directories. You can see a list of the files in another directory by changing to that directory, and then using the dir command again. In this case, you will change to the DOS directory. Before you begin this section, make sure you have a directory named DOS by carrying out the following procedure.

- **To make sure you have a directory named Windows**

If you do not see a line in the directory list indicating that you have a directory named Windows, type the following at the command prompt:

  - **dir /s Windows**

- You will see a message that includes a line such as the following:

  - **Directory of C:\DIRNAME**

- **To change from the root directory to the WINDOWS directory**

To change directories, you will use the **cd** command. The **cd** command stands for "***change directory***."

1. Type the following at the command prompt:

```
cd windows
```

The command prompt changes. It should now look like the following:

```
C:\WINDOWS>
```

Next, you will use the dir command to view a list of the files in the DOS directory.

- **To view a list of the files in the WINDOWS directory**

1. Type the following at the command prompt:

```
dir
```

A list of the files in the WINDOWS directory appears, but scrolls by too quickly to read. You can modify the **dir** command so that it displays only one screen of information at a time.

- **To view the contents of a directory one screen at a time**

1. Type the following at the command prompt:

```
dir /p
```

One screen of information appears. At the bottom of the screen, you will see the following message:

```
Press any key to continue . . .
```

To view the next screen of information, press any key on your keyboard. Repeat this step until the command prompt appears at the bottom of your screen.

When you typed the **dir** command this time, you included the **/p** switch after the command. A ***switch*** modifies the way MS-DOS carries out a command. Generally, a switch consists of a forward slash (/) that is followed by one or more letters or numbers.

When you used the **/p** switch with the **dir** command, you specified that MS-DOS should pause after it displays each screen of directory list information. The **p** actually stands for "page"

- Another helpful switch you can use with the **dir** command is the **/w** switch. The **/w** switch indicates that MS-DOS should show a wide version of the directory list.

o **To view the contents of a directory in wide format**

1. Type the following at the command prompt:

```
dir /w
```

The directory list appears, with the filenames listed in wide format. Note that only filenames are listed. No information about the files' size or date and time of creation appears.

2. If the directory contains more files than will fit on one screen, you can combine the /p and /w switches as follows:

```
dir /w /p
```

Changing Back to the Root Directory

Next, you will change from the DOS directory to the root directory. The root directory is the directory you were in before you changed to the DOS directory.

Before you begin this section, make sure your command prompt looks like the following:

```
C:\DOS>
```

o **To change to the root directory**

1. Type the following at the command prompt:

```
cd \
```

Note that the slash you type in this command is a backslash (\), not a forward slash (/).

No matter which directory you are in, this command always returns you to the root directory of a drive. The root directory does not have a name. It is simply referred to by a backslash (\).

The command prompt should now look like the following:

```
C:\>
```

When your command prompt appears similar to this---that is, when it does not contain the name of a directory---you are in the root directory.

Creating a Directory

In this section, you will create two directories. Creating a directory is helpful if you want to organize related files into groups to make them easy to find. Before you begin this section, make sure the command prompt looks like the following:

```
C:\>
```

To create a directory, you will use the **md** command. The **md** command stands for "make directory."

o To create and change to a directory named FRUIT

1. Type the following at the command prompt:

```
md fruit
```

You have now creat ed a directory named FRUIT. You won't see the new FRUIT directory until you carry out the **dir** command in the next step.

2. To confirm that you successfully created the FRUIT directory, type the following at the command prompt:

```
dir
```

or

```
dir /p
```

Look through the directory list. A new entry somewhere in the list should look similar to the following:

```
FRUIT <DIR> 09-25-93 12:09p
```

3. To change to the new FRUIT directory, type the following at the command prompt:

```
cd fruit
```

The command prompt should now look like the following:

```
C:\FRUIT>
```

You will now create a directory within the FRUIT directory, named GRAPES.

- **To create and work with a directory named GRAPES**

1. Type the following at the command prompt:

```
md grapes
```

You will not see the new GRAPES directory until you carry out the **dir** command in the next step.

2. To confirm that you successfully created the GRAPES directory, type the following at the command prompt:

```
dir
```

A list similar to the following appears:

```
Volume in drive C is MS-DOS-6 Volume Serial Number is
lE49-15E2 Directory of C:\FRUIT

.            <DIR>            09-25-93   12:08p
..           <DIR>            09-25-93   12:08p
GRAPES       <DIR>            09-25-93   12:10p
                    3 file(s) 0 bytes 11534336 bytes free
```

Note that there are three entries in the FRUIT directory. One is the GRAPES directory that you just created. There are two other entries---one looks like a single period (**.**) and the other looks like a double period (**..**). These directory entries are important to MS- DOS, but you can ignore them. They appear in every directory and cont ain information relation to the directory structure.

The GRAPES directory is a ***subdirectory*** of the FRUIT directory. A subdirectory is a directory within another directory. Subdirectories are useful if you want to further subdivide information.

3. To change to the GRAPES directory, type the following at the command prompt:

```
cd grapes
```

The command prompt should now look like the following:

```
C:\FRUIT\GRAPES>
```

4. To switch back to the FRUIT directory, type the following:

```
cd ..
```

The command prompt should now look like the following:

```
C:\FRUIT>
```

When the **cd** command is followed by two periods (**..**), MS-DOS moves up one level in the directory structure. In this case, you moved up one level from the GRAPES directory to the FRUIT directory.

Deleting a Directory

If you no longer use a particular directory, you may want to delete it to simplify your directory structure. Deleting a directory is also useful if you type the wrong name when you are creating a directory and you want to delete the incorrect directory befo re creating a new one.

In this section, you will delete the GRAPES directory. Before you begin this section, make sure the command prompt looks like the following:

```
C:\FRUIT>
```

To delete a directory, use the **rd** command. The **rd** command stands for "***remove directory***."

o To delete the GRAPES directory

1. Type the following at the command prompt:

```
rd grapes
```

2. To confirm that you successfully deleted the GRAPES directory, type the following at the command prompt:

```
dir
```

The GRAPES directory should no longer appear in the directory list.

Note You cannot delete a directory if you are in it. Before you can delete a directory, you must make the directory that is one level higher the current directory. To do this, type **cd..** at the command prompt.

Changing Drives

This section describes how to change drives. Changing drives is useful if you want to work with files that are on a different drive.

So far, you have been working with drive C. You have other drives you can use to store information. For example, drive A is your first floppy disk drive. The files and directories on drive A are located on the floppy disk in the drive. (You might also have a drive B, which contains the files and directories stored on the floppy disk in that drive.)

Before you begin this section, make sure your command prompt looks like the following:

```
C:\FRUIT>
```

o **To change to and view files on a different drive**

1. Insert a 3.5" floppy disk in drive A label- side up. Make sure the disk clicks into the drive.

1. Type the following at the command prompt:

```
a:
```

Note that the command prompt changed to the following:

```
A:\>
```

This message may appear:

```
Not ready reading drive A Abort, Retry, Fail?
```

If you see this message, the disk may not be inserted properly. Place the disk label-side up in the disk drive, and make sure the disk clicks into the disk drive. Then, type r for Retry. If this message appears again, press F for Fail, and then type b: at the command prompt. If you no longer see this message, type b: instead of a: throughout the rest of the tutorial.

There must be a floppy disk in the drive that you want to change to.

2. Change back to drive C by typing the following at the command prompt:

```
c:
```

Your command prompt should return to the following:

```
C:\FRUIT>
```

When you type a drive letter followed by a colon, you change to that drive. The drive letter that appears in the command prompt shows which drive is the ***current drive***. Unless you specify otherwise, any commands you type are carried out on the current drive and in the current directory.

So far, all the commands you typed were carried out on the current drive and in the current directory. You can also carry out a command on a drive that isn't current. For example, you can view the files on a disk in drive A without switching to drive A by following this procedure.

- To view the contents of the WINDOWS directory on drive C

1. Type the following at the command prompt:

```
dir c:\windows
```

A list of the files in the DOS directory on drive **C** should scroll past on your screen.

Copying Files

This section describes how to copy a single file and a group of files. Copying files creates a duplicate of the original file and does not remove the original file. This is useful for many reasons. For example, if you want to work on a document at home, you can copy it from your computer at work to a floppy disk and then take the floppy disk home.

To copy a file, you will use the copy command. When you use the copy command, you must include two parameters. The first is the location and name of the file you want to copy, or the source. The second is the location to which you want to copy the file, or the destination. You separate the source and destination with a space. The copy command follows this pattern:

copy source destination

- Copying a Single File

In this section, you will copy the notepad.exe files from the WINDOWS directory to the FRUIT directory. You will specify the source and destination of these files in two different ways. The difference between the two methods is explained at the end of this section.

Before you begin this section, make sure the command prompt looks like the following:

```
C:\FRUIT>
```

To copy the NOTEPAD.EXE files from the WINDOWS directory to the FRUIT directory

1. Return to the root directory by typing the following at the command prompt:

```
cd\
```

The command prompt should now look like the following:

C:\>
Change to the DOS directory by typing the following at the command prompt:

cd windows

The command prompt should now look like the following:

C:\WINDOWS>

2. Make sure the file you are going to copy, NOTEPAD.EXE, is located in the WINDOWS directory by using the dir command followed by a filename.

dir notepad.exe

A list similar to the following appears:

3. To copy the NOTEPAD.EXE file from the WINDOWS directory to the FRUIT directory, type the following at the command prompt:

copy c:\windows\notepad.exe c:\fruit

The following message appears:

1 file(s) copied

4. To confirm that you copied the files successfully, view the contents of the FRUIT directory by typing the following at the command prompt:

dir \fruit

You should see the file listed in the FRUIT directory.

Renaming Files

This section explains how to rename files. You may want to rename a file if the information in it changes or if you decide you prefer another name.

To rename a file, you will use the **ren** command. The **ren** command stands for "**rename**." When you use the **ren** command, you must include two parameters.

The first is the file you want to rename, and the second is the new name for the file. You separate the two names with a space. The **ren** command follows this pattern:

ren oldname newname

- **Renaming a File**

In this section, you will rename the README.TXT file.

Before you begin this section, make sure your command prompt looks like the following:

```
C:\FRUIT>
```

1. To rename the NOTEPAD.EXE file to PADNOTE.TXT, type the following at the command prompt:

   ren notpad.exe padnote.txt

2. To confirm that you renamed the file successfully, type the following at the command prompt:

   dir

Deleting Files

This section explains how to delete, or remove, a file that you no longer want on your disk. If you don't have very much disk space, deleting files you no longer use is essential.

To delete a file, you will use the del command. The del command stands for "delete."

- **Deleting a File**

  In this section, you will delete two files using the **del** command.

Before you begin, make sure your command prompt looks like the following:

C:\FRUIT>

To delete the PEARCOM and PEAR.HLP files

1. Delete the PADNOTE.TXT file by typing the following at the command prompt:

   del PADNOTE.TXT

2. To confirm that you deleted the files successfully, type the following at the command prompt:

   dir

- **Deleting a Group of Files**

  In this section, you will use wildcards to delete a group of files.

  Before you begin this section, make sure your command prompt looks like the following:

  C:\FRUIT>

To delete files in the current directory that end with the extension OLD by using wildcards

1. View all files that end with the extension OLD by typing the following at the command prompt:

   dir *.old

   A list of all the files that end with the extension OLD appears. Make sure that these are the files you want to delete. When you are deleting files by using wildcards, this step is very important. It will prevent you from deleting files accidentally.

2. Delete all files ending with OLD by typing the following at the command prompt:

   del *.old

3. To confirm that all the files with the extension OLD have been deleted, type the following at the command prompt:

   dir

The FRUIT directory should contain no files.

Now that the FRUIT directory is empty, you can delete it by using the **rd** (remove directory) command that you learned to use in "Deleting a Directory" earlier in this chapter.

- **To delete the FRUIT directory**

1. Return to the root directory by typing the following at the command prompt:

   ***cd ***

2. You can see the FRUIT directory in the directory list by typing the following at the command prompt:

   `dir` or **`dir /p`**

3. Remove the FRUIT directory by typing the following at the command prompt:

   rd fruit

4. To verify that the FRUIT directory has been removed, type the following at the command prompt:

   `dir` or **`dir /p`**

   The FRUIT directory should not appear in the directory list.

Lab Task No. 2

Installing Ubuntu

Objective: Is to understand the process of Linux installation.

Scope: Linux distribution Ubuntu 18.04 installation.

Task:

Step 1) Download Ubuntu 18.04 LTS ISO File

Please make sure you have the latest version of Ubuntu 18.04 LTS, If not, please download the ISO file from the link here

https://www.ubuntu.com/download/desktop

Since Ubuntu 18.04 LTS only comes in a 64-bit edition, so you can install it on a system that supports 64-bit architecture.

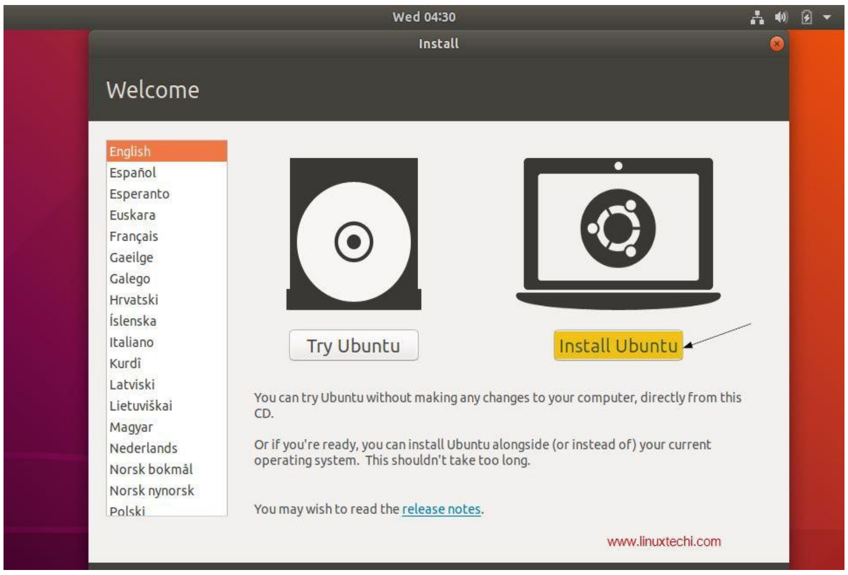
Step 2) Create a Bootable Disk

Once the ISO file is downloaded then next step is to burn the downloaded ISO image into the USB/DVD or flash drive to boot the computer from that drive.

Also make sure you change the boot sequence so that system boots using the bootable CD/DVD or flash drive.

Step 3) Boot from USB/DVD or Flash Drive

Once the system is booted using the bootable disk, you can see the following screen presented before you with options including "Try Ubuntu" and "Install Ubuntu" as shown in the image below,



You can try Ubuntu without making any changes to your computer, directly from this CD.

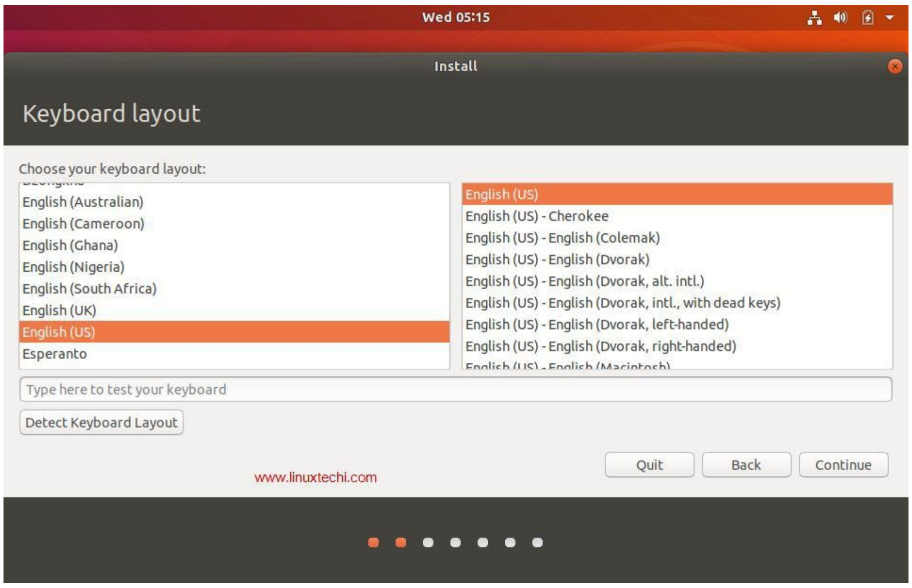
Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the release notes.

Even though when you click “Try Ubuntu” you can have a sneak peek into the 18.04 LTS without installing it in your system, our goal here is to install Ubuntu 18.04 LTS in your system. So click “Install Ubuntu” to continue with the installation process.

Step 4) Choose your Keyboard layout

Choose your favorite keyboard layout and click “Continue”. By default English (US) keyboard is selected and if you want to change, you can change here and click “Continue”,


Wed 05:15
Install
Keyboard layout
Choose your keyboard layout:
English (Australian)
English (Cameroon)
English (Ghana)
English (Nigeria)
English (South Africa)
English (UK)
English (US)
Esperanto
English (US)
English (US) - Cherokee
English (US) - English (Colemak)
English (US) - English (Dvorak)
English (US) - English (Dvorak, alt. intl.)
English (US) - English (Dvorak, intl., with dead keys)
English (US) - English (Dvorak, left-handed)
English (US) - English (Dvorak, right-handed)
Type here to test your keyboard
Detect Keyboard Layout
www.linuxtechi.com
Quit
Back
Continue

Step 5) Preparing to Install Ubuntu and other Software

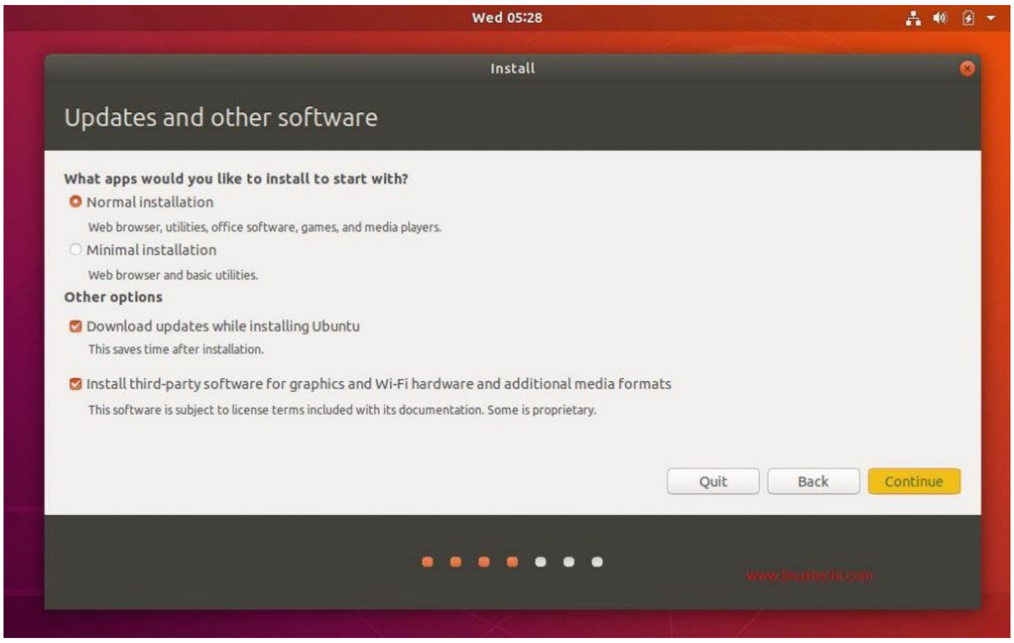
In the next screen, you’ll be provided following beneath options including:

Type of Installation:

Normal Installation or Minimal installation, If you want a minimal installation then select second option otherwise go for the Normal Installation. In my case I am doing Normal Installation

Download Updates While Installing Ubuntu (select this option if your system has internet connectivity during installation)

Install third party software for graphics and Wi-Fi hardware, MP3 and additional media formats Select this option if your system has internet connectivity)


Wed 05:28

Install

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.

click on “Continue” to proceed with installation

Step 6) Select the appropriate Installation Type

Next the installer presents you with the following installation options including:

- Erase Disk and Install Ubuntu
- Encrypt the new Ubuntu installation for security
- Use LVM with the new Ubuntu installation
- Something Else

Where,

Erase Disk and Install Ubuntu

- Choose this option if your system is going to have only Ubuntu and erasing anything other than that is not a problem. This ensures a fresh copy of Ubuntu 18.04 LTS is installed in your system.

Encrypt the new Ubuntu installation for security

- Choose this option if you are looking for extended security for your disks as your disks will be completely encrypted. If you are beginner, then it is better not to worry about this option.

Use LVM with the new Ubuntu installation

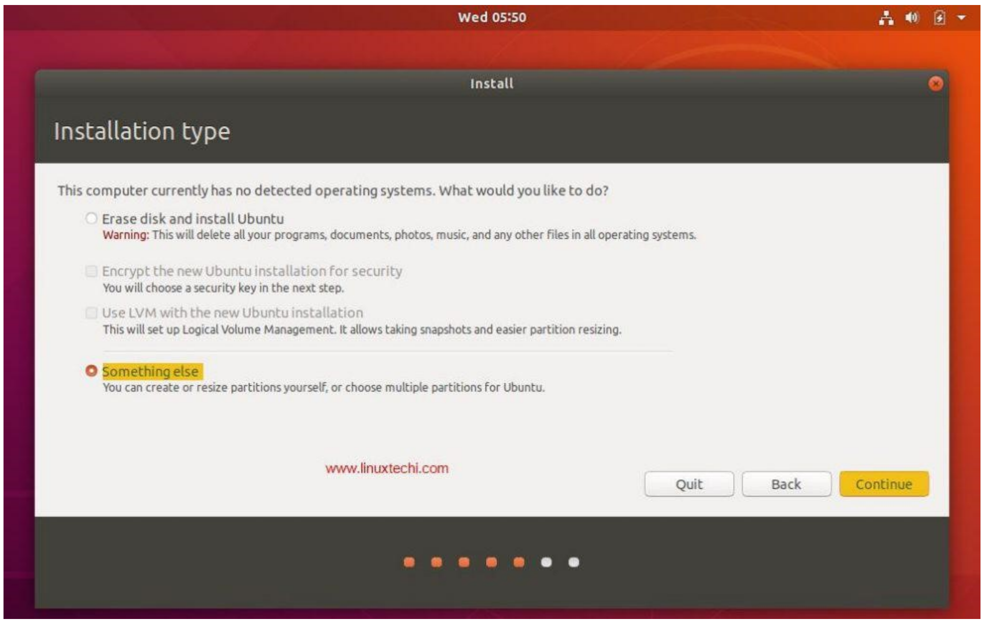
- Choose this option if you want to use LVM based file systems.

Something Else

- Choose this option if you are advanced user and you want to manually create your own partitions and want to install Ubuntu along with existing OS

In this article, we will be creating our custom partitions on a hard disk of 40 GB and the following partitions are to be created:

/boot	1 GB (ext4 files system)
/home	18 GB (ext4 file system) / 12 GB (ext4 file system)
/var	6 GB (ext4 file system)
Swap	2 GB



Wed 05:50

Install

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.

Encrypt the new Ubuntu installation for security
You will choose a security key in the next step.

Use LVM with the new Ubuntu installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

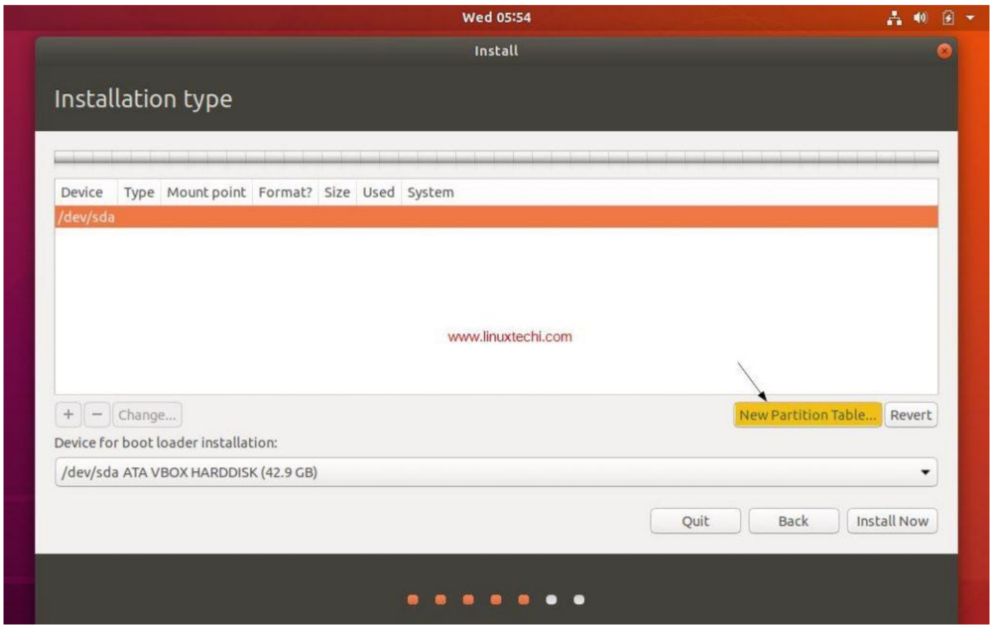
[✓ Something else]
You can create or resize partitions yourself, or choose multiple partitions for Ubuntu.

www.linuxtechi.com

Now, Choose "Something Else" and Click on continue

You can see the available disk size for Ubuntu in the next window as shown below:

Now in order to create your own partitions, click on "New Partition Table"



Wed 05:54

Install

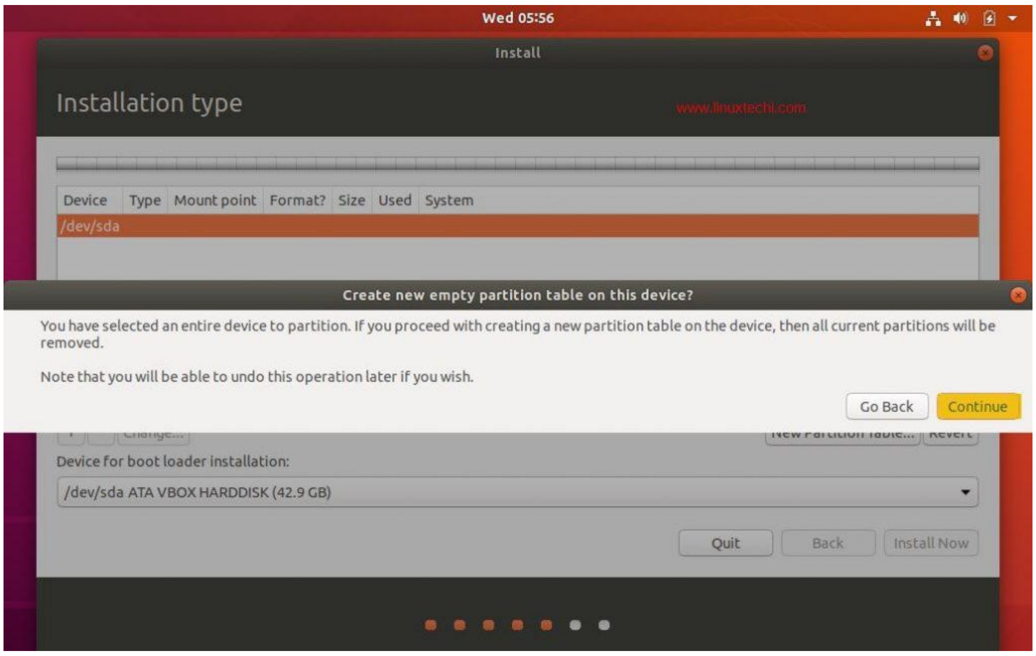
Installation type

Device	Type	Mount point	Format?	Size	Used	System
/dev/sda						

www.linuxtechi.com

Device for boot loader installation:

/dev/sda ATA VBOX HARDDISK (42.9 GB)


Wed 05:56
Install
Installation type
www.linuxtechi.com
Device | Type | Mount point | Format? | Size | Used | System
/dev/sda
Create new empty partition table on this device?

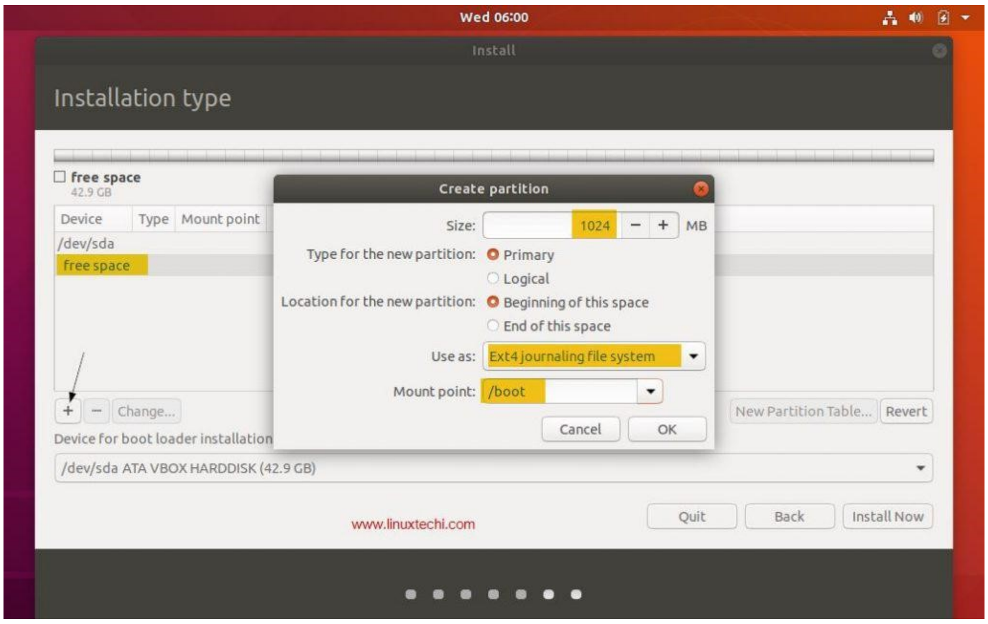
You have selected an entire device to partition. If you proceed with creating a new partition table on the device, then all current partitions will be removed.

Note that you will be able to undo this operation later if you wish.

Go Back
Continue
Device for boot loader installation:
/dev/sda ATA VBOX HARDDISK (42.9 GB)
Quit
Back
Install Now

Click on Continue

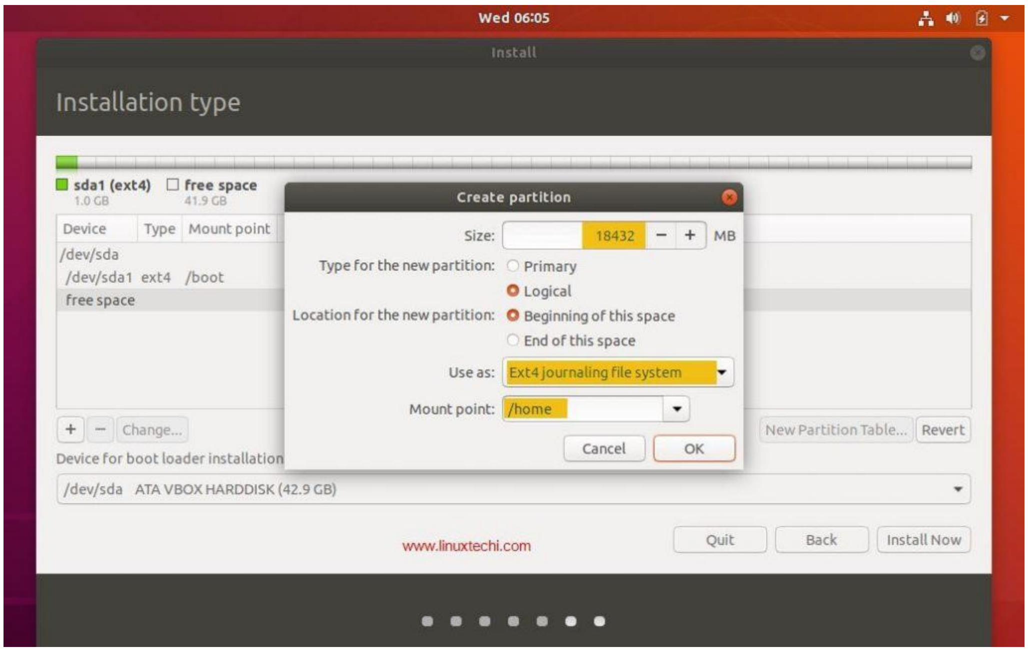
Create /boot partition of size 1GB, Select the free space and then Click on the “+” symbol to create a new partition



Size: 1024 MB
Type for the new partition: [✓ Primary] Logical
Location for the new partition: [✓ Beginning of this space] End of this space
Use as: Ext4 journaling file system
Mount point: /boot

Click on “OK”

Let’s create /home partition of size 18 GB,


Wed 06:05

Install

Installation type

🟩 sda1 (ext4) 1.0 GB ☐ free space 41.9 GB

Device	Type	Mount point
/dev/sda		
/dev/sda1	ext4	/boot
free space		

Create partition

Size: 18432 MB

Type for the new partition: Primary [✓ Logical]

Location for the new partition: [✓ Beginning of this space] End of this space

Use as: Ext4 journaling file system

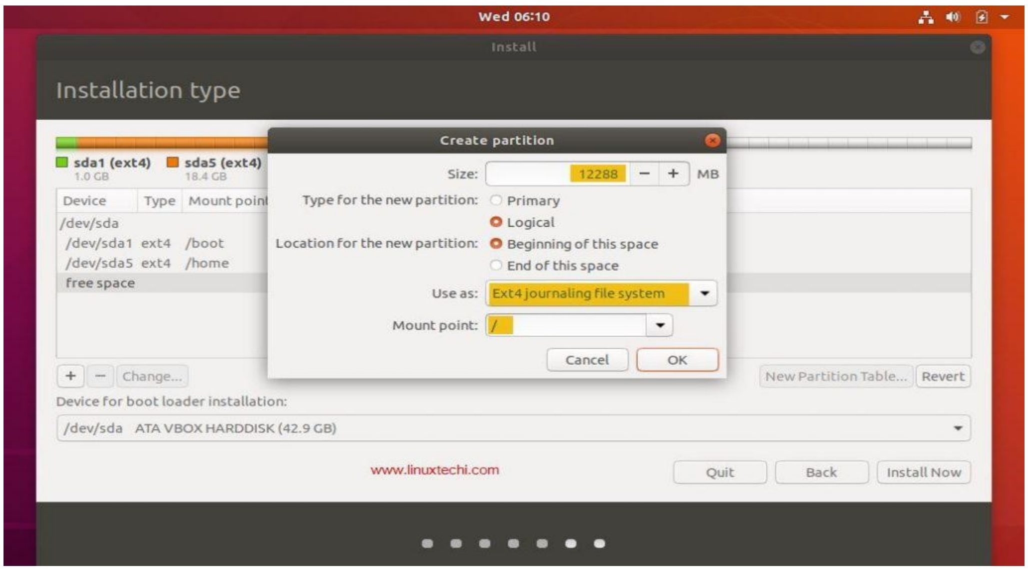
Mount point: /home

Device for boot loader installation:

/dev/sda ATA VBOX HARDDISK (42.9 GB)

www.linuxtechi.com

In the same way create / & /var file system of size 12 GB & 6 GB respectively


Wed 06:10

Install

Installation type

🟩 sda1 (ext4) 1.0 GB 🟧 sda5 (ext4) 18.4 GB

Device	Type	Mount point
/dev/sda		
/dev/sda1	ext4	/boot
/dev/sda5	ext4	/home
free space		

Create partition

Size: 12288 MB

Type for the new partition: Primary [✓ Logical]

Location for the new partition: [✓ Beginning of this space] End of this space

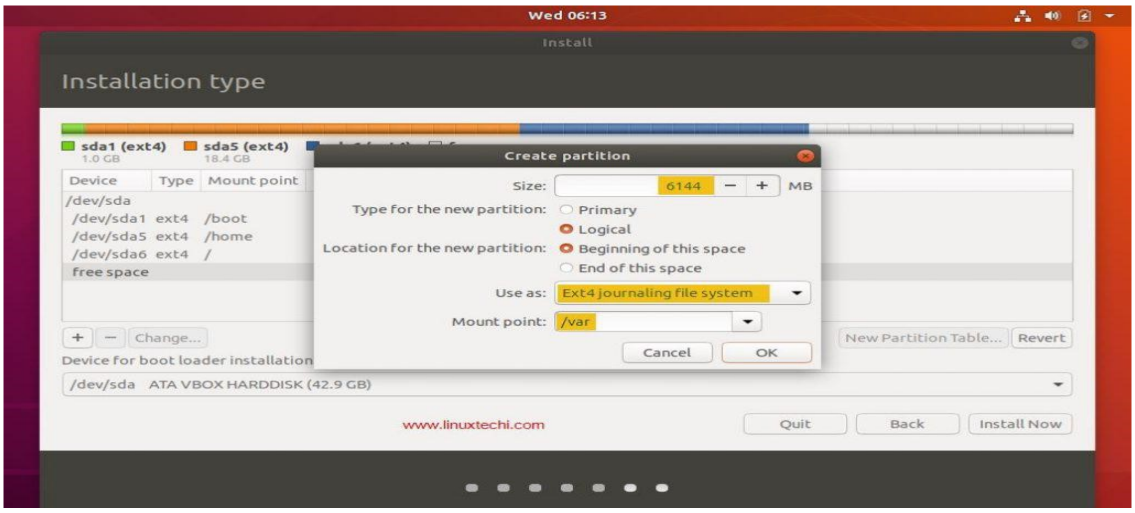
Use as: Ext4 journaling file system

Mount point: /

Device for boot loader installation:

/dev/sda ATA VBOX HARDDISK (42.9 GB)

www.linuxtechi.com


Wed 06:13

Install

Installation type

🟩 sda1 (ext4) 1.0 GB 🟧 sda5 (ext4) 18.4 GB

Device	Type	Mount point
/dev/sda		
/dev/sda1	ext4	/boot
/dev/sda5	ext4	/home
/dev/sda6	ext4	/
free space		

Create partition

Size: 6144 MB

Type for the new partition: Primary [✓ Logical]

Location for the new partition: [✓ Beginning of this space] End of this space

Use as: Ext4 journaling file system

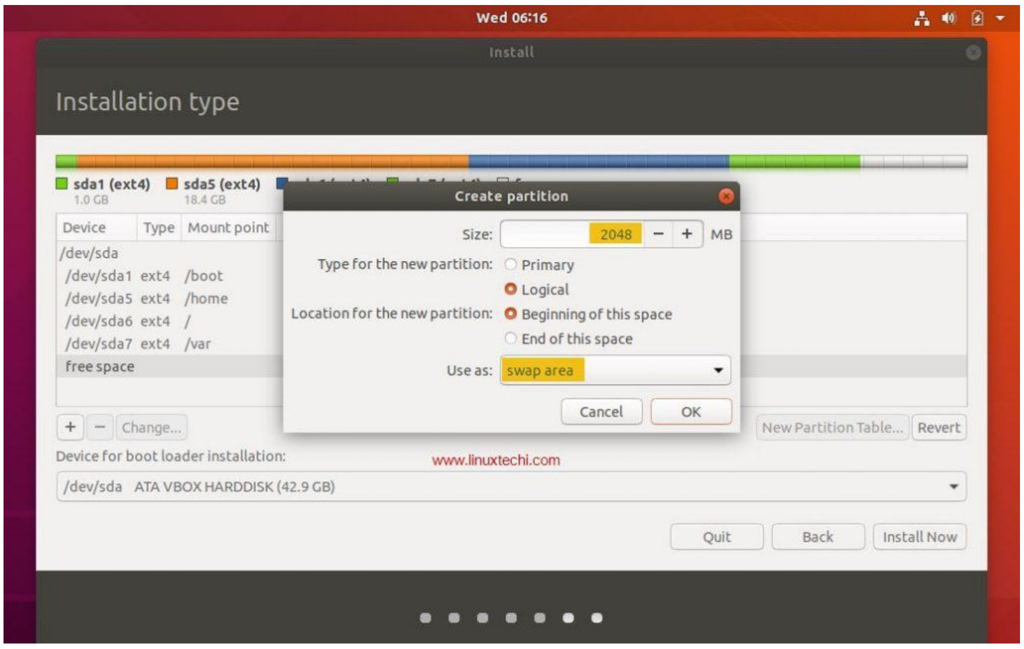
Mount point: /var

Device for boot loader installation:

/dev/sda ATA VBOX HARDDISK (42.9 GB)

www.linuxtechi.com

Now create last partition as swap of size 2 GB,



sda1 (ext4) 1.0 GB, sda5 (ext4) 18.4 GB

Device	Type	Mount point
/dev/sda		
/dev/sda1	ext4	/boot
/dev/sda5	ext4	/home
/dev/sda6	ext4	/
/dev/sda7	ext4	/var
free space		

Create partition

Size: 2048 MB

Type for the new partition: Primary [✓ Logical]

Location for the new partition: [✓ Beginning of this space] End of this space

Use as: swap area

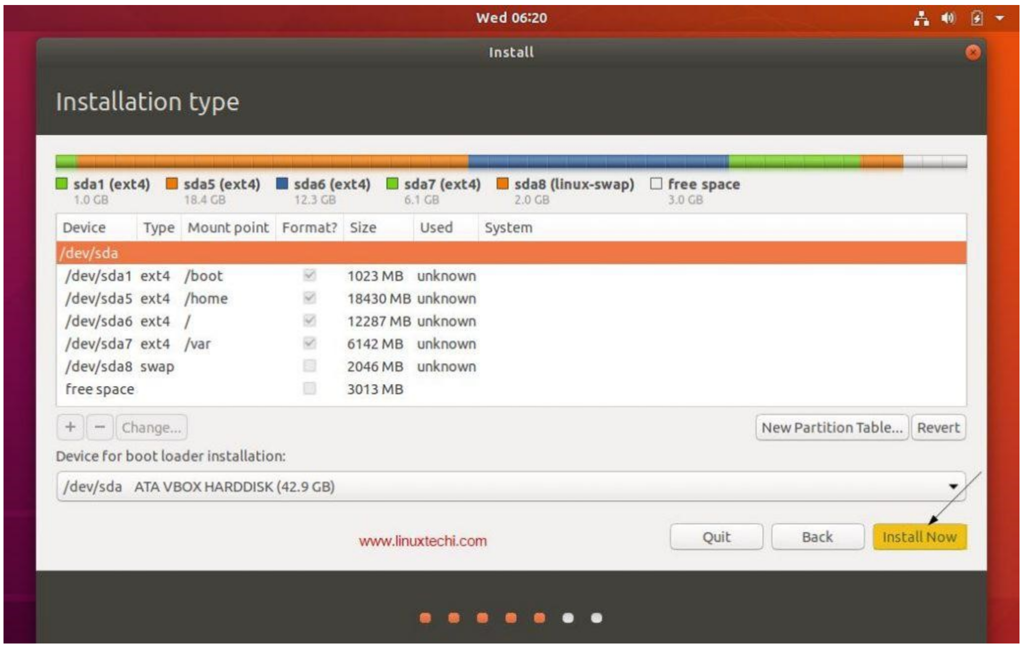
Device for boot loader installation:

www.linuxtechi.com

/dev/sda ATA VBOX HARDDISK (42.9 GB)

Click on OK

Once you are done with the partition creation task , then click on “Install Now” option to proceed with the installation



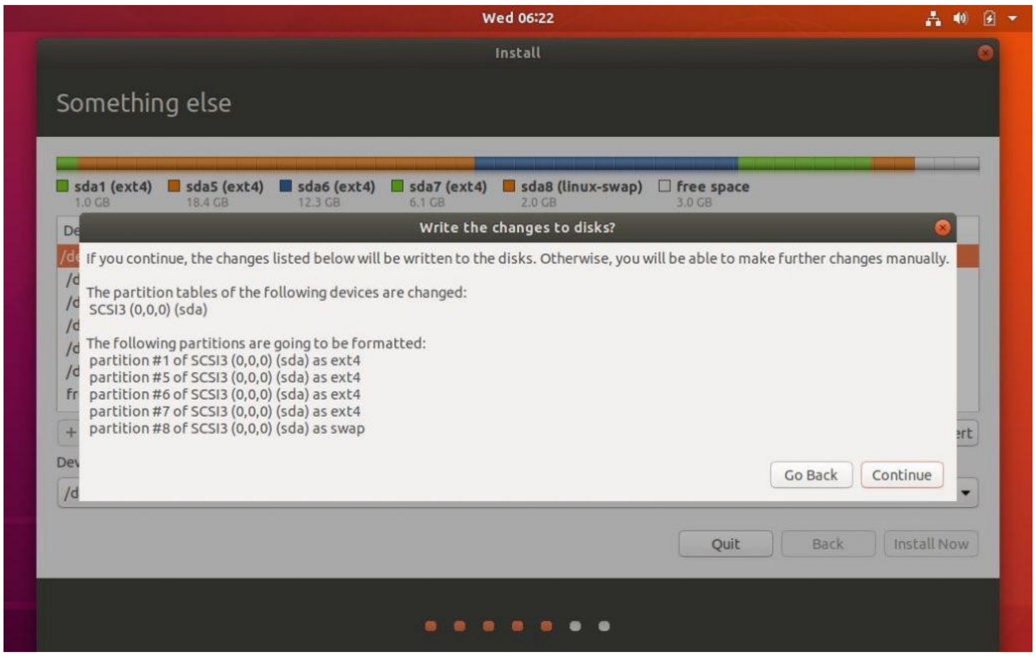
sda1 (ext4) 1.0 GB, sda5 (ext4) 18.4 GB, sda6 (ext4) 12.3 GB, sda7 (ext4) 6.1 GB, sda8 (linux-swap) 2.0 GB, free space 3.0 GB

Device	Type	Mount point	Format?	Size	Used	System
/dev/sda						
/dev/sda1	ext4	/boot	✓	1023 MB	unknown	
/dev/sda5	ext4	/home	✓	18430 MB	unknown	
/dev/sda6	ext4	/	✓	12287 MB	unknown	
/dev/sda7	ext4	/var	✓	6142 MB	unknown	
/dev/sda8	swap			2046 MB	unknown	
free space				3013 MB		

Device for boot loader installation:

/dev/sda ATA VBOX HARDDISK (42.9 GB)

www.linuxtechi.com



sda1 (ext4) 1.0 GB, sda5 (ext4) 18.4 GB, sda6 (ext4) 12.3 GB, sda7 (ext4) 6.1 GB, sda8 (linux-swap) 2.0 GB, free space 3.0 GB

Write the changes to disks?

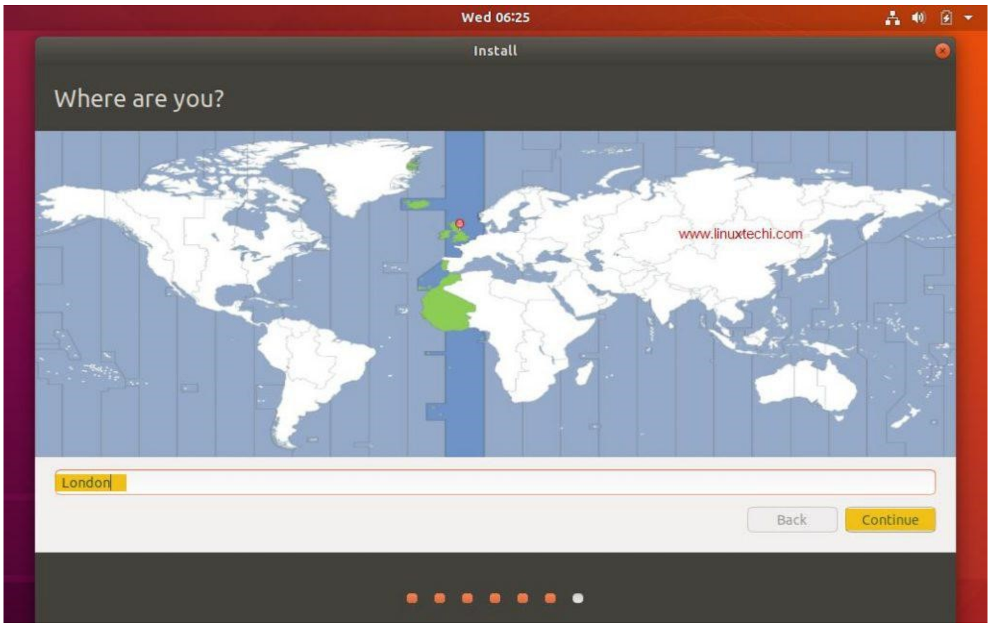
If you continue, the changes listed below will be written to the disks. Otherwise, you will be able to make further changes manually.

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

The following partitions are going to be formatted:
partition #1 of SCSI3 (0,0,0) (sda) as ext4
partition #5 of SCSI3 (0,0,0) (sda) as ext4
partition #6 of SCSI3 (0,0,0) (sda) as ext4
partition #7 of SCSI3 (0,0,0) (sda) as ext4
partition #8 of SCSI3 (0,0,0) (sda) as swap

Now click on “Continue” to write all the changes to the disks

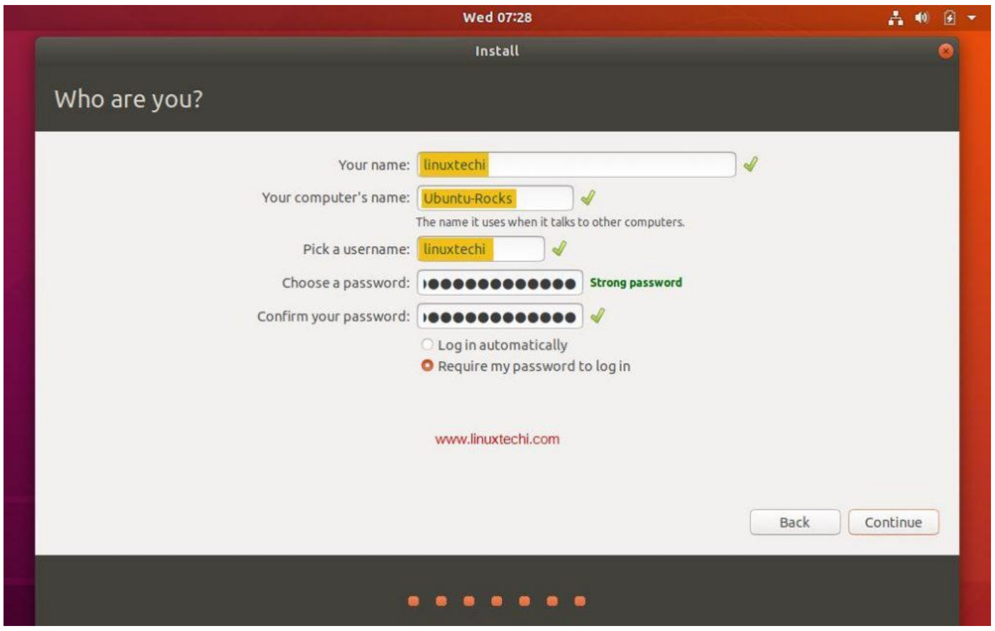
Step 7) Select Your Time zone



London

Choose your favorite time zone and then click on “Continue” Step 8) Provide your User Credentials

In the next screen you will be prompted to provide your user credentials. In this screen provide your name, computer name, username and the password to login into Ubuntu 18.04 LTS



Your name: linuxtechi

Your computer's name: Ubuntu-Rocks

The name it uses when it talks to other computers.

Pick a username: linuxtechi

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

Confirm your password: ●●●●●●●●●●●●●

Log in automatically [✓ Require my password to log in]

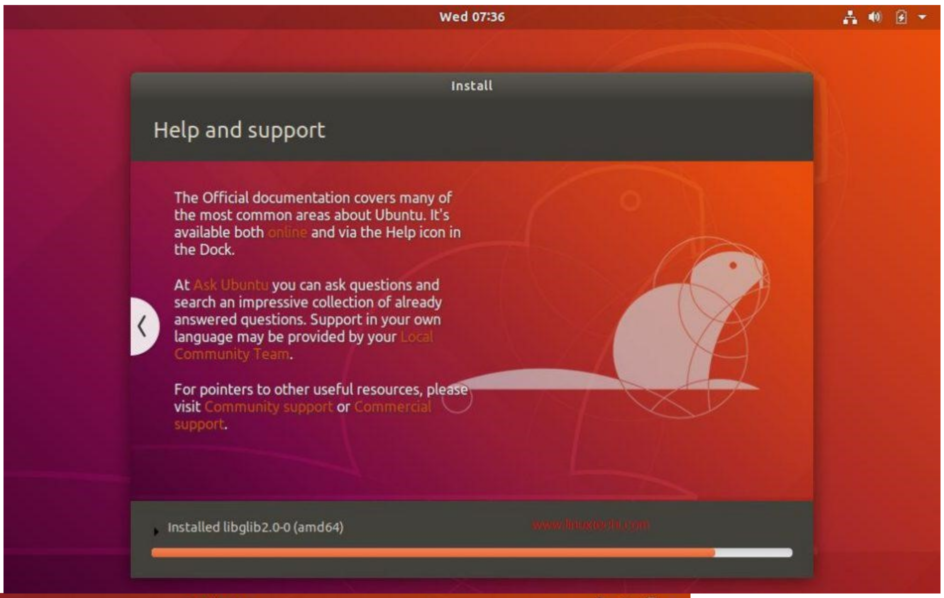
Click “Continue” to begin the installation process.

Step 9) Start Installing Ubuntu 18.04 LTS

The installation of Ubuntu 18.04 LTS starts now and will take around 5-10 mins depending on the speed of your computer,

Step 10) Restart Your System

Once the installation is completed, remove the USB/DVD from the drive and Click “Restart Now” to restart your system.



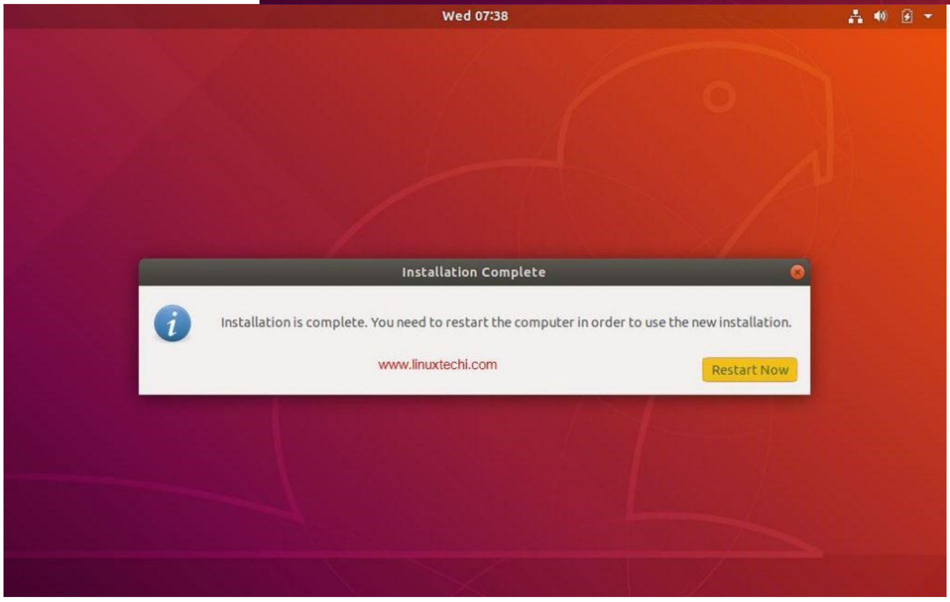
Help and support

The Official documentation covers many of the most common areas about Ubuntu. It's available both online and via the Help icon in the Dock.

At Ask Ubuntu you can ask questions and search an impressive collection of already answered questions. Support in your own language may be provided by your Local Community Team.

For pointers to other useful resources, please visit Community support or Commercial support.

Installed libglib2.0-0 (amd64)

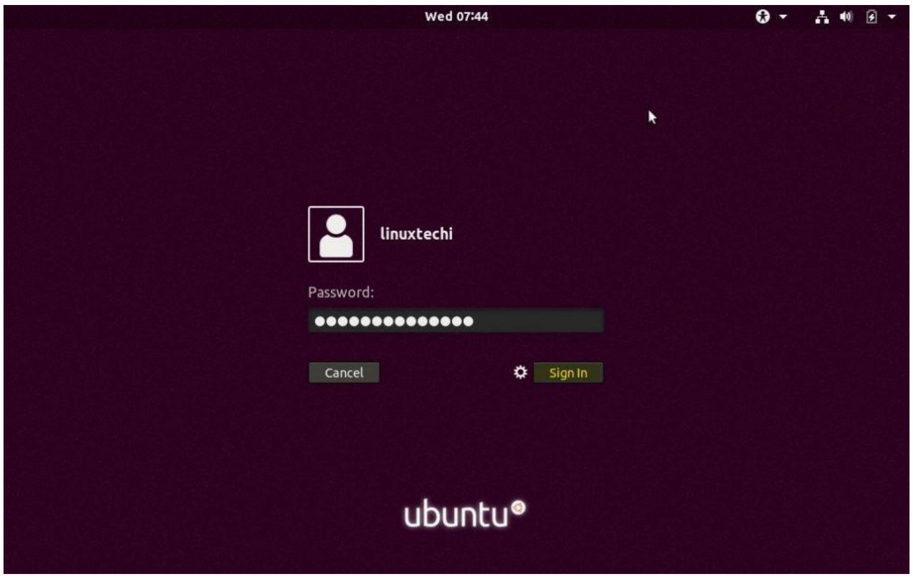


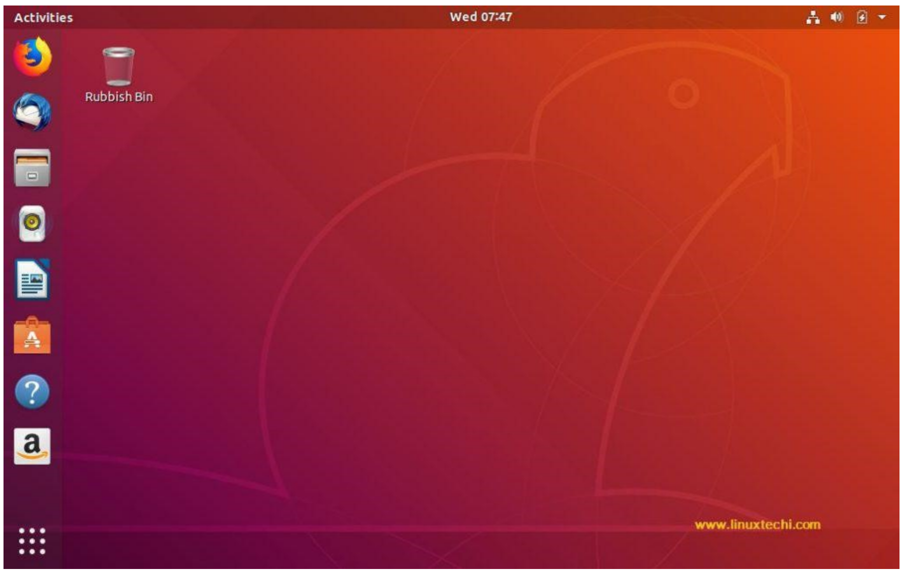
Installation is complete. You need to restart the computer in order to use the new installation.

Read More on : How to Install VirtualBox 6.0 on Ubuntu 18.04 LTS / 18.10 / CentOS 7

Step:11) Login to Your Ubuntu 18.04 desktop

Once your system has been rebooted after the installation then you will get the beneath login screen, enter the User name and password that you have set during installation (Step 8)


Wed 07:44
linuxtechi
Password:
Cancel
Sign In
ubuntu


Activities
Wed 07:47
Rubbish Bin
www.linuxtechi.com

Lab Task No. 3

CPU SCHEDULINGALGORITHMS

A). FIRST COME FIRST SERVE:

AIM: To write a c program to simulate the CPU scheduling algorithm First Come First Serve (FCFS)

DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFSmainly says first come first serve the algorithm which came first will be served first.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time Step 4: Set the waiting of the first process as _0‘and its burst time as its turnaround time Step 5: for each process in the Ready Q calculate
a). Waiting time (n) = waiting time (n-1) + Burst time (n-1) b).
Turnaround time (n)= waiting time(n)+Burst time(n)
Step 6: Calculate
a) Average waiting time = Total waiting Time / Number of process

b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
 float wtavg, tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;tat[0]
= tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
        printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();
}
```

INPUT

```
Enter the number of processes --        3
Enter Burst Time for Process 0 --       24
Enter Burst Time for Process 1 --       3
Enter Burst Time for Process 2 --       3
```

OUTPUT

```
PROCESS         BURST TIME      WAITING TIME    TURNAROUND
                                                TIME
P0              24              0               24
P1              3               24              27
P2              3               27              30
```

```
Average Waiting Time--      17.000000
Average Turnaround Time --                              27.000000
```

B). SHORTEST JOB FIRST:

AIM: To write a program to stimulate the CPU scheduling algorithm Shortest job first (Non- Preemption)

DESCRIPTION:

To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
Step 5: Set the waiting time of the first process as _0‘ and its turnaround time as its burst time.
Step 6: Sort the processes names based on their Burt time
Step 7: For each process in the ready queue, calculate
a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)
b) Turnaround time (n)= waiting time(n)+Burst time(n)
Step 8: Calculate
c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number of process
Step 9: Stop the process

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
main()
{
int  p[20],  bt[20],  wt[20],  tat[20],  i,  k,  n,  temp;  float  wtavg,
tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);

}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;

temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
 tat[i]  =  tat[i-1]  +bt[i];
wtavg  =  wtavg  +  wt[i];
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
        printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();}
```

INPUT

```
Enter the number of processes --          4
Enter Burst Time for Process 0 --         6
Enter Burst Time for Process 1 --         8
Enter Burst Time for Process 2 --         7
Enter Burst Time for Process 3 --         3
```

OUTPUT

```
PROCESS              BURST     WAITING     TURNARO
                     TIME      TIME        UND TIME
P3                   3         0           3
P0                   6         3           9
P2                   7         9           16
P1                   8         16          24
Average Waiting Time --        7.000000
Average Turnaround Time --     13.000000
```

Lab Task No. 4

C). ROUND ROBIN:

AIM: To simulate the CPU scheduling algorithm round-robin.

DESCRIPTION:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process (n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

a) Waiting time for process (n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU fromprocess(n-1)

b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number ofprocess Step

8: Stop the process

SOURCE CODE

```
#include<stdio.h>
main()
{
int    i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
float awt=0,att=0,temp=0;
clrscr();
printf("Enter  the  no  of  processes  --  ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter  Burst  Time  for  process  %d  --  ",  i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnter  the  size  of  time  slice  --  ");
scanf("%d",&t);
max=bu[0];
for(i=1;i<n;i++)
if(max<bu[i])
max=bu[i];
for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
 if(bu[i]<=t)    {
tat[i]=temp+bu[i];
temp=temp+bu[i];
bu[i]=0;
 }
 else {
bu[i]=bu[i]-t;
temp=temp+t;
 }
for(i=0;i<n;i++){
wa[i]=tat[i]-
ct[i]; att+=tat[i];
awt+=wa[i];}
printf("\nThe Average Turnaround time is -- %f",att/n);
printf("\nThe Average Waiting time is -- %f ",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
getch();}
```

INPUT:

```
Enter the no of processes – 3
 Enter Burst Time for process 1 – 24
Enter Burst Time for process 2 -- 3 Enter
Burst Time for process 3 – 3 Enter the
size of time slice – 3
```

OUTPUT:

```
PROCESS        BURST TIME        WAITING TIME     TURNAROUNDTIME
1                  24                   6               30
2                  3                   4               7
3                 3                    7              10
The Average Turnaround time is – 15.666667 The
Average Waiting time is------------5.666667
```

D). PRIORITY:

AIM: To write a c program to simulate the CPU scheduling priorityalgorithm.

DESCRIPTION:

To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the processes. The waiting time of each process is obtained by summing up the burst times of all the previous processes.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as ‗0‘ and its burst time as its turnaround time
Step 6: Arrange the processes based on process priority
Step 7: For each process in the Ready Q calculate Step 8:
for each process in the Ready Q calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)
b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.

Step10: Stop

SOURCE CODE:

```
#include<stdio.h>
main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg,
tatavg;
clrscr();
printf("Enter the number of  processes --- ");
scanf("%d",&n);
for(i=0;i<n;i++){
p[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ",i); scanf("%d
%d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(pri[i] > pri[k]){
temp=p[i];
p[i]=p[k];
p[k]=temp;
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;
}
wtavg   =   wt[0]   =   0;
tatavg  =  tat[0]  =  bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i];
 tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND
TIME");
for(i=0;i<n;i++)
printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n); printf("\nAverage
Turnaround Time is --- %f",tatavg/n);
getch();}
```

INPUT

```
Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10      3
Enter the Burst Time & Priority of Process 1 --- 1       1
Enter the Burst Time & Priority of Process 2 --- 2       4
Enter the Burst Time & Priority of Process 3 --- 1       5
Enter the Burst Time & Priority of Process 4 --- 5       2
```

OUTPUT

```
PROCESS     PRIORITY    BURST TIME     WAITIN     TURN
                                       G  TIME    UND TIME
1           1           1              0          1
4           2           5              1          6
0           3           10             6          16
2           4           2              16         18
3           5           1              18         19
Average Waiting Time is ---             8.200000
Average Turnaround Time is----------------12.000000
```

Lab Task No. 5

MEMORY MANAGEMENT

A). MEMORY MANAGEMENT WITH FIXED PARTITIONING TECHNIQUE (MFT)

AIM: To implement and simulate the MFT algorithm.

DESCRIPTION:

In this the memory is divided in two parts and process is fit into it. The process which is best suited will be placed in the particular memory where it suits. In MFT, the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. In MVT, each job gets just the amount of memory it needs. That is, thepartitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient'' user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation.

ALGORITHM:

```
Step1: Start the process.
Step2: Declarevariables.
Step3: Enter total memory size ms.
Step4: Allocate memory for os.
Ms=ms-os
Step5: Read the no partition to be divided n Partition size=ms/n.
Step6: Read the process no and process size.
Step 7: If process size is less than partition size allot alse blocke the process. While allocating
update memory wastage-external fragmentation.
if(pn[i]==pn[j])f=1;
if(f==0){ if(ps[i]<=siz)
{
extft=extft+size-
ps[i];avail[i]=1; count++;
}
}
Step 8: Print the results
```

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
main()
{
int     ms,     bs,     nob,     ef,n,
mp[10],tif=0; int i,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
scanf("%d",&mp[i]);
}
printf("\nNo.        of        Blocks        available        in        memory--%d",nob);
printf("\n\nPROCESS\tMEMORYREQUIRED\tALLOCATED\tINTERNAL
FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
printf("\t\tYES\t%d",bs-mp[i]);
 tif = tif + bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
 printf("\nTotal External Fragmentation is %d",ef);
getch();
}
```

INPUT

```
          Enter the total memory available (in Bytes) --                1000
          Enter the block size (in  Bytes)--    300
          Enter the number of processes – 5
Enter memory required for process 1 (in Bytes) --           275
Enter memory required for process 2 (in Bytes) --           400
Enter memory required for process 3 (in Bytes) --           290
Enter memory required for process 4 (in Bytes) --           293
Enter memory required for process 5 (in Bytes) --           100
No. of Blocks available in memory --               3
```

OUTPUT

```
PROCESS                                          ALLOCAT        INTERNAL
                MEMORY REQUIRED                    ED           FRAGMENTATION
1               275                                YES          25
2               400                                NO           -----
3               290                                YES          10
4               293                                YES          7
          Memory is Full, Remaining Processes cannot be accommodated Total
          Internal Fragmentation is 42
          Total External Fragmentation is 100
```

B) MEMORY VARIABLE PARTIONING TYPE (MVT)

AIM: To write a program to simulate the MVT algorithm

ALGORITHM:

Step1: start the process.
Step2: Declare variables.
Step3: Enter total memory size ms.
Step4: Allocate memory for os.
Ms=ms-os
Step5: Read the no partition to be divided n Partition size=ms/n.
Step6: Read the process no and process size.
Step 7: If process size is less than partition size allot alse blocke the process. While allocating update memory wastage-external fragmentation.

```
if(pn[i]==pn[j])        f=1;
if(f==0){ if(ps[i]<=size)
{
extft=extft+size-
ps[i];avail[i]=1; count++;
}
}
```

Step 8: Print the results
Step 9: Stop the process.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
main()
 {
 int                ms,mp[10],i,
 temp,n=0; char ch = 'y';
 clrscr();
 printf("\nEnter the total memory available (in Bytes)-- ");
 scanf("%d",&ms);
 temp=ms;
 for(i=0;ch=='y';i++,n++)
 {
 printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
 scanf("%d",&mp[i]);
 if(mp[i]<=temp)
 {
 printf("\nMemory is allocated for Process %d ",i+1);
 temp = temp - mp[i];
 }
 else
 {
 printf("\nMemory is Full"); break;
 }
 printf("\nDo you want to continue(y/n) -- ");
  scanf(" %c", &ch);
 }
 printf("\n\nTotal     Memory     Available     --     %d",     ms);
 printf("\n\n\tPROCESS\t\t     MEMORY     ALLOCATED     ");
 for(i=0;i<n;i++)
 printf("\n \t%d\t\t%d",i+1,mp[i]);
 printf("\n\nTotal     Memory     Allocated     is     %d",ms-temp);
 printf("\nTotal External Fragmentation is %d",temp);
 getch();
 }
```

OUTPUT:

```
Enter the total memory available (in Bytes) – 1000
Enter memory  required for process 1 (in Bytes) – 400
 Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275
Memory is allocated for Process 2
Do you want to continue(y/n) – y
 Enter memory required for process 3 (in Bytes) – 550


Memory is Full

 Total Memory Available – 1000

PROCESS              MEMORY ALLOCATED

 1                   400
 2              275


  Total Memory Allocated is 675
  Total External Fragmentation is 325
```

Lab Task No. 6

MEMORY ALLOCATION TECHNIQUES

AIM: To Write a C program to simulate the following contiguous memory allocation techniques

a) Worst-fit b) Best-fit c) First-fit

DESCRIPTION

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

PROGRAM

WORST-FIT

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

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

INPUT

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

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

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

OUTPUT

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

BEST-FIT

```
 #include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
        static int bf[max],ff[max];
        clrscr();
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of files:");
        scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
     printf("Block %d:",i);
     scanf("%d",&b[i]);
        printf("Enter the size of the files :-\n");
        for(i=1;i<=nf;i++)
        {
                printf("File %d:",i);
                scanf("%d",&f[i]);
        }
        for(i=1;i<=nf;i++)
        {
                 for(j=1;j<=nb;j++)
                {
                        if(bf[j]!=1)
                        {
                                temp=b[j]-f[i];
                                if(temp>=0)
                                        if(lowest>temp)
                                        {
                                        ff[i]=j;
                                         lowest=temp;
                                         }
                      }}
                 frag[i]=lowest;  bf[ff[i]]=1;  lowest=10000;
        }
        printf("\nFile No\tFile Size \tBlock No\tBlock
        Size\tFragment"); for(i=1;i<=nf && ff[i]!=0;i++)

                printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
        getch();
}
```

INPUT

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

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

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

OUTPUT

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

FIRST-FIT

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
        int
        frag[max],b[max],f[max],i,j,nb,nf,temp,highes
        t=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)
                                            {
                            }
                    }
                     frag[i]=highest; bf[ff[i]]=1; highest=0;
            }
            ff[i]=j; highest=temp;
         }
            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();
    }
```

INPUT

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

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

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

OUTPUT

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

Lab Task No. 7

PAGE REPLACEMENT ALGORITHMS

AIM: To implement FIFO page replacement technique.

a) FIFO b) LRU c) OPTIMAL

DESCRIPTION:

Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults.

FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

LRU-In this algorithm page will be replaced which is least recently used

OPTIMAL- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. This algorithm will give us less page fault when compared to other page replacement algorithms.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the frame Place avail[i]=1 if page is placed in theframe Count page faults
7. Print the results.
8. Stop the process.

A) FIRST IN FIRST OUT

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h> int fr[3];
void main()
{
void display();
int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
int
flag1=0,flag2=0,pf=0,frsize=3,top=0;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0; flag2=0; for(i=0;i<12;i++)
{
if(fr[i]==page[j])
{
flag1=1; flag2=1; break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j]; flag2=1; break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0;
}
display();
}
printf("Number of page faults : %d ",pf+frsize);
 getch();
}
void display()
{
int     i;     printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
5  3  1
5  2  1
5  2  4
5  2  4
3  2  4
3  2  4
3  5  4
3  5  2

Number of page faults: 9
```

B) LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM:

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

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
 clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1; break;
}
}
if(flag1==0)
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];       flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k]) fs[i]=1;
}}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf+frsize);
getch();
}
void display()
{
int i; printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
2  5  1
2  5  1
2  5  4
2  5  4
3  5  4
3  5  2
3  5  2
3  5  2

No of page faults: 7
```

Lab Task No. 8

C) OPTIMAL

AIM: To implement optimal page replacement technique.

ALGORTHIM:

1. Start Program
2. Read Number Of Pages And Frames
3.Read Each Page Value
4. Search For Page In The Frames
5. If Not Available Allocate Free Frame
6. If No Frames Is Free Repalce The Page With The Page That Is Leastly Used
7.Print Page Number Of Page Faults
8.Stop process.

SOURCE CODE:

```
/* Program to simulate optimal page replacement */
#include<stdio.h>
#include<conio.h>
int fr[3], n, m;
void
display();
void main()
{
 int i,j,page[20],fs[10];
 int
 max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
 float pr;
 clrscr();
 printf("Enter length of the reference string: ");
 scanf("%d",&n);
 printf("Enter the reference string: ");
 for(i=0;i<n;i++)
 scanf("%d",&page[i]);
 printf("Enter no of frames: ");
 scanf("%d",&m);
 for(i=0;i<m;i++)
 fr[i]=-1; pf=m;
```

```
for(j=0;j<n;j++)
 {
 flag1=0;           flag2=0;
 for(i=0;i<m;i++)
 {
 if(fr[i]==page[j])
 {
 flag1=1;  flag2=1;
 break;
 }
 }
 if(flag1==0)
 {
 for(i=0;i<m;i++)
 {
 if(fr[i]==-1)
 {
fr[i]=page[j];  flag2=1;
break;
 }
 }
 }
 if(flag2==0)
 {
 for(i=0;i<m;i++)
 lg[i]=0;
 for(i=0;i<m;i++)
 {
 for(k=j+1;k<=n;k++)
 {
 if(fr[i]==page[k])
 {
 lg[i]=k-j;
 break;
 }
 }
 }
 found=0;
 for(i=0;i<m;i++)
 {
 if(lg[i]==0)
 {
index=i;
found = 1;
```

```
 break;
 }
 }
if(found==0)
 {
 max=lg[0];    index=0;
 for(i=0;i<m;i++)
 {
 if(max<lg[i])
{
 max=lg[i];
 index=i;
}
 }
 }
 fr[index]=page[j];
 pf++;
}
display();
}
printf("Number    of    page    faults    :    %d\n",    pf);
pr=(float)pf/n*100;
printf("Page fault rate = %f \n", pr); getch();
}
void display()
{
 int    i;    for(i=0;i<m;i++)
 printf("%d\t",fr[i]);
 printf("\n");
}
```

OUTPUT:

```
Enter length of the reference string: 12
Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5
Enter no of frames: 3
1 -1 -1
1 2 -1
1 2 3
1 2 4
1 2 4
1 2 4
1 2 5
1 2 5
1 2 5
3 2 5
4 2 5
4 2 5
Number of page faults : 7 Page fault rate = 58.333332
```

Lab Task No. 9

FILE ORGANIZATION TECHNIQUES

A) SINGLE LEVEL DIRECTORY:

AIM: Program to simulate Single level directory file organization technique.

DESCRIPTION:

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

SOURCE CODE :

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

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

```
        }
        if(i==dir.fcnt)
        printf("File %s not found",f);
                                   else

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

OUTPUT:

```
Enter name of directory -- CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

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

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

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

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

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

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

Lab Task No. 10

TWO LEVEL DIRECTORY

AIM: Program to simulate two level file organization technique

Description:
In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFDis searched.

SOURCE CODE :

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

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

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

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

```
                printf("Directory  %s not found",d); jmp1: break;
            case 5: if(dcnt==0)
           printf("\nNo Directory's ");
                else
                {
                        printf("\nDirectory\tFiles");
                        for(i=0;i<dcnt;i++)
                        {
                            printf("\n%s\t\t",dir[i].dname);
                            for(k=0;k<dir[i].fcnt;k++)
                            printf("\t%s",dir[i].fname[k]);
                        }
                }
                break;

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

OUTPUT

```
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 1
Enter name of directory -- DIR1 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 -- DIR2 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 – DIR1
Enter name of the file -- A1
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 – DIR1

Enter name of the file -- A2
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6.
Exit Enter your choice – 6
```

VIVA QUESTIONS

1. Define directory?
2. List the different types of directory structures?
3. What is the advantage of hierarchical directory structure?
4. Which of the directory structures is efficient? Why?
5. What is acyclic graph directory?

Lab Task No. 11

FILE ALLOCATION STRATEGIES

A) SEQUENTIAL:

AIM: To write a C program for implementing sequential file allocation method

DESCRIPTION:

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
Step 4: Allocate the required locations to each in sequential order a).
Randomly select a location from availablelocation s1= random(100);
a) Check whether the required locations are free from the selected
location.
 if(b[s1].flag==0){
for                    (j=s1;j<s1+p[i];j++){
if((b[j].flag)==0)count++;
      }
if(count==p[i]) break;
}
b) Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)
{
k[i][j]=s; j=j+1; b[s].bno=s;
b[s].flag=1;
}

Step 5: Print the results file no, length, Blocks allocated. Step
6: Stop the program
```

<u>SOURCE CODE :</u>

```
#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1
;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch();
}
```

OUTPUT:

```
Enter the starting block & length of file 4 10
4->1
5->1
6->1
7->1
8->1
9->1
10->1
11->1
12->1
13->1
The file is allocated to disk.
```

B) INDEXED:

AIM: To implement allocation method using chained method

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation.

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 q= random(100);
    a) Check whether the selected location is free .
    b) If the location is free allocate and set flag=1 to the allocated locations.

q=random(100);
{
if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
Step 5: Print the results file no, length ,Blocks allocated.
Step 6: Stop the program
```

SOURCE CODE :

```
#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");
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
```

Lab Task No. 12

C) LINKED:

AIM: To implement linked file allocation technique.

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation

ALGORTHIM:

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 q= random(100);

a) Check whether the selected location is free .
b) If the location is free allocate and set flag=1 to the allocated locations.

While allocating next location address to attach it to previous location

```
for(i=0;i<n;i++)
{
for(j=0;j<s[i];j++)
{
q=random(100);               if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
             if(j>0)
             {
}
}
p=r[i][j-1]; b[p].next=q;}
```

Step 5: Print the results file no, length ,Blocks allocated.
Step 6: Stop the program

SOURCE CODE :

```
#include<stdio.h>
main()
{
int f[50],p,i,j,k,a,st,len,n,c;
clrscr();
for(i=0;i<50;i++) f[i]=0;
printf("Enter how many blocks that are already
allocated"); scanf("%d",&p);
printf("\nEnter the blocks no.s that are already allocated");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
X:
printf("Enter the starting index block &
length"); scanf("%d%d",&st,&len); k=len;
for(j=st;j<(k+st);j++)
{
if(f[j]==0)
{ f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n %d->file is already
allocated",j);
k++;
}
}
printf("\n If u want to enter one
more file? (yes-1/no-0)");
scanf("%d",&c);
if(c==1)
goto
X;
else
exit();
getch( );}
```

OUTPUT:

```
Enter how many blocks that are already allocated 3 Enter the blocks no.s
that are already allocated 4 7  Enter the starting index block & length 3 7  9
3->1
4->1 file is already allocated
5->1
6->1
7->1 file is already allocated
8->1
9->1file is already allocated
10->1
11->1
12->1
```

Lab Task No. 13

DEAD LOCK AVOIDANCE

AIM: To Simulate bankers algorithm for Dead Lock Avoidance (Banker‘s Algorithm)

DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type Rj is available. Max: If
- max[i, j]=k, Pi may request at most k instances resource Rj.
- Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj Need: If Need[I,
- j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]- Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   - Finish[i] =False
   - Need<=Work If no such I exists go to step 4.
3. work= work + Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I;
Allocation I=Allocation +Request I;
Need i=Need i- Request I;
If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the state is unsafe, the Pi must wait for Request i and the

old resource-allocation state is restored.

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.
9. or not if we allow the request.
10. stop the program.
11. *end*

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int      count=0,c=0;
char      finish[10];
clrscr();
printf("Enter    the    no.    of    processes    and    resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter      the      claim      matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter      the      allocation      matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0;          for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A:
for(i=0;i<n;i++)
{
 c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
if(c==m)
{
printf("All the resources can be allocated to Process %d", i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
 else
printf("\n   System   is   in   safe   mode");
printf("\n The given state is safe state");
getch();
}
```

OUTPUT

```
Enter the no. of processes and resources: 4 3
Enter the claim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2
Available resources are: 6 2 3
Process 2 executed?:y
All the resources can be allocated to Process 3 Available resources
are: 8 3 4
Process 3 executed?:y
All the resources can be allocated to Process 4 Available resources
are: 8 3 6
Process 4 executed?:y
All the resources can be allocated to Process 1
Available resources are: 9 3 6
Process 1 executed?:y
System is in safe mode
The given state is safe state
```

Lab Task No. 14

DEAD LOCKPREVENTION

AIM: To implement deadlock prevention technique

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

DESCRIPTION:

Data structures

-
- n-Number of process, m-number of resource types.
-
- Available: Available[j]=k, k – instance of resource type Rj is available.
- Max: If max[i, j]=k, Pi may request at most k instances resource Rj.

  Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.

Find an i such that both
- Finish[i] =False
- Need<=Work

If no such I exists go to step 4.

5. work=work+Allocation, Finish[i] =True;

if Finish[1]=True for all I, then the system is in safe state

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. Stop the process.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
void main()
{
char job[10][10];
int time[10],avail,tem[10],temp[10]; int safe[10];
int        ind=1,i,j,q,n,t;
clrscr();
printf("Enter no of jobs: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter   name   and   time:   ");
scanf("%s%d",&job[i],&time[i]);
}
printf("Enter    the    available    resources:");
scanf("%d",&avail);
for(i=0;i<n;i++)
{
temp[i]=time[i];
tem[i]=i;
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(temp[i]>temp[j])
{
t=temp[i];
temp[i]=temp[j];
temp[j]=t;  t=tem[i];
tem[i]=tem[j];
tem[j]=t;
}
 }
for(i=0;i<n;i++)
{
q=tem[i];
if(time[q]<=avail)
{
safe[ind]=tem[i];
avail=avail-tem[q];
printf("%s",job[safe[ind]]);
ind++;
}
else
{
printf("No safe sequence\n");
}
}
printf("Safe sequence is:");
for(i=1;i<ind; i++)
printf("%s %d\n",job[safe[i]],time[safe[i]]);
getch();
}
```

OUTPUT:

```
Enter no of jobs:4
Enter name and time: A 1
Enter name and time: B 4
Enter name and time: C 2
Enter name and time: D 3
Enter   the   available   resources:   20
Safe sequence is: A 1, C 2, D 3, B 4.
```

Lab Task No. 15

AIM: To Write a C program to simulate disk scheduling algorithms
a) FCFS b) SCAN c) C-SCAN

DESCRIPTION

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reacheseach cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip

PROGRAM

A) FCFS DISK SCHEDULING ALGORITHM

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

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
```

B) SCAN DISK SCHEDULING ALGORITHM

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

C) C-SCAN DISK SCHEDULING ALGORITHM

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

        p=0;
        while(t[j]!=tot-1)
        {
                atr[p]=t[j];
                j++;
                p++;
        }
        atr[p]=t[j];
        p++;
        i=0;
        while(p!=(n+3) && t[i]!=t[h])
        {
                atr[p]=t[i]; i++;
                p++;
        }
        for(j=0;j<n+2;j++)
        {
                if(atr[j]>atr[j+1])
                        d[j]=atr[j]-atr[j+1];
                else
                        d[j]=atr[j+1]-atr[j];
                sum+=d[j];
        }
        printf("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
```