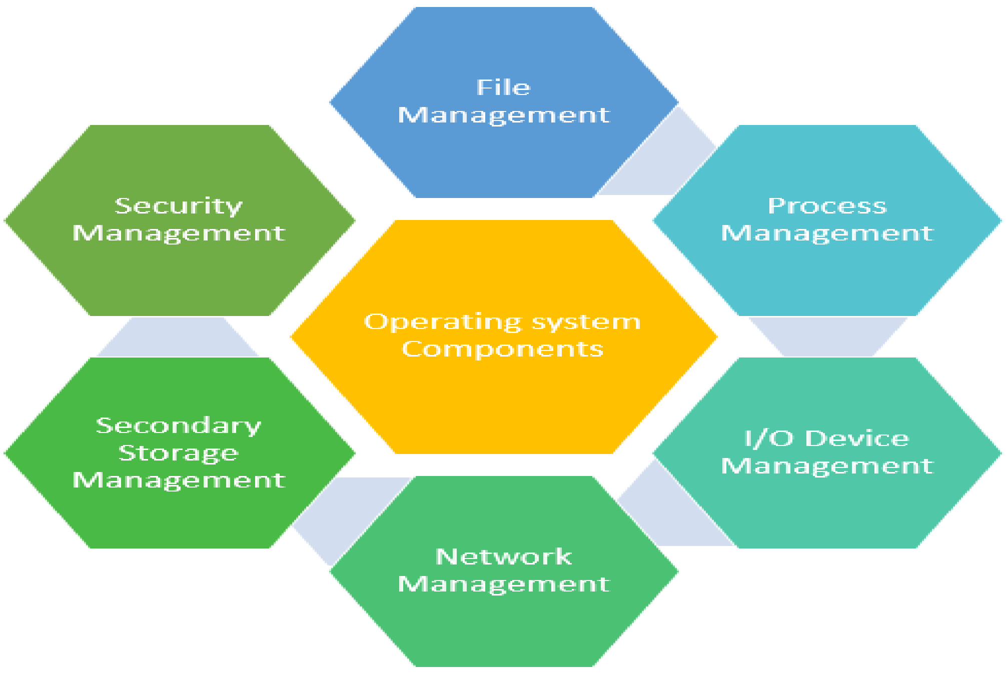
# DEBRE MARKOS UNIVERSITY

# INSTITUTE OF TECHNOLOGY

## SCHOOL OF COMPUTING

## Information Technology Academic Program



**Laboratory Manual for operating system (ITec2022)**

**Prepared by: - Minichel Yibeyin**

**Email Address: -minichelyb@gmail.com**

**DEBRE MARKOS, ETHIOPIA**

**NOVEMBER, 2020**

# Table of Contents

## LIST OF ACRONYMS

|  |  |
|---|---|
| OS | OPERATING SYSTEM |
| CPU | CENTRAL PROCESSING UNIT |
| MVT | MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS |
| MFT | MULTIPROGRAMMING WITH A FIXED NUMBER OF TASKS |
| SJF | Shortest Job First |
| CD | Compact Disk |

# Chapter One

## 1. Overview of Operating Systems (OS)

**What is Operating System?**

Operating system is the core software component of your computer. It performs many functions and is, in very basic terms, an interface between your computer and the outside world. A computer is a modern system consists of one or more processor, some main memory, disks, printers, a keyboard, display and other I/O systems. This is a complex system. Thus writing programs that keeps track of all these components and use them correctly is a mandatory. For these reason, computers are provided with a layer of software called the ***operating system (OS***).The job of OS is to manage all these devices and provide user program with a simpler interface to the hardware.

An operating system is the actual software that controls the allocation and use of a computer's hardware. It keeps components working in unity, acting as a communicator between the user, the computer's hardware and software.

Operating system is system software that controls the execution of programs and that provides services such as resource allocation, scheduling, I/O control and data/file management. It hides the complexity of how hardware components work.

Operating System interprets commands and instructions. It also coordinates compilers, assemblers, utility programs, and other software to the various user of the computer system. OS provides easy communication between the computer system and the computer operator (human). It also establishes data security and integrity.

## Basic Functions of operating system

Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

For large systems, the operating system has even greater responsibilities and powers. It makes sure that different program and users running at the same time do not interfere with each other. The operating system is also responsible for security, ensuring that unauthorized users do not access the system. Today most operating systems perform the following important functions:

1. **Process management**:

That is, assignment of processor to different tasks/processes being performed by the computer system. This allows two or more processes to be executed at a time. Here the operating system must decide if it can run the different processes on single processor at a time, using multitasking concept.

The operating system needs to allocate enough of the processor's time to each process and application so that they can run as efficiently as possible. This is particularly important for multitasking. When the user has multiple applications and processes running, it is up to the operating system to ensure that they have enough resources to run properly.

## 2. **Memory management**:

That is, allocation of main memory and other storage areas to the system programs as well as user programs and data. This involves allocating, and often to create a virtual memory for program. Paging which means organizing data so that the program data is loaded into pages of memory. Another method of managing memory is swapping. This involves swapping the content of memory to disk storage.

The operating system needs to ensure that each process has enough memory to execute the process, while also ensuring that one process does not use the memory allocated to another process. This must also be done in the most efficient manner. The operating system must balance the needs of each process with the different types of memory available.

## 3. **Input/output management or Device Management**:

That is, co-ordination and assignment of the different output and input device while one or more programs are being executed. Most computers have additional hardware, such as printers and scanners, connected to them. These devices require drivers, or special programs that translate the electrical signals sent from the operating system or application program to the hardware device. The operating system manages the input to and output from the computer.

## 4. **File management**:

That is, the storage of file on various storage devices. It also allows all files to be easily changed and modified through the use of text editors or some other files manipulation. An Operating System can create and maintain a file System, where users can create, delete and move files around a structured file system.

## 5. **Protection and Security**

The processes in an operating system must be protected from one another's activities, and to provide such protection, we must ensure that only processes that have gained proper authorization from the operating system can operate on the files, memory, CPU and other resources of the system. Protection provides a mechanism for controlling access to the resources provided by a computer system. Historically, protection was a concern only on multiprogrammed computer systems with several users. However, with the advent of networking and Internet, all computer systems, from servers to PADs, must be concerned with protection. Security ensure the authentication of system users to protect the integrity of the information stored in the system (code and data), as well as the physical resources of the computer system.

At the simplest level, an operating system does two things:

1. It manages the hardware and software resources of the system.

2. It provides a stable, consistent way for applications, for users, to deal with the hardware without having to know all the details of the hardware.

## Types of Operating Systems

There are several types of operating systems, with Windows, Linux and Macintosh group being the most widely used. Here is an overview on each system:

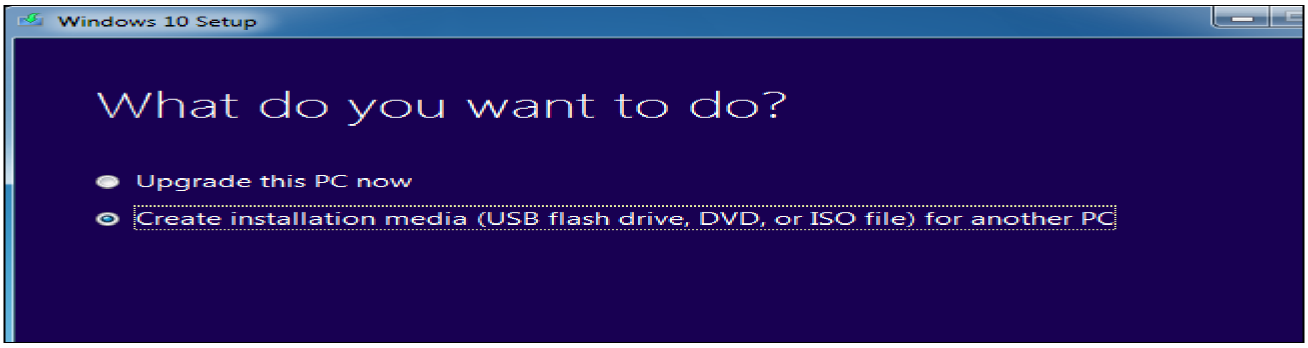
**Windows:** Windows is the popular Microsoft brand preferred by most personal users. This system has come a long way from version 1.0 all the way up to the new Windows 10. Although Windows has made steps in regard to security, it has a reputation for being one of the most vulnerable systems.

## Experiment 1.1:-Installation processes for window 10 on PC

## AIM: to discuss Installation processes for window 10 on PC

### How to Get Installation Media and Do a Clean Install of Windows 10

If you don't want to upgrade from an existing Windows installation, you can download the official Windows 10 installation media for free from Microsoft and perform a clean install. To do this, visit Microsoft's Download Windows 10 page, click "Download Tool Now", and run the downloaded file. Select "Create installation media for another PC".



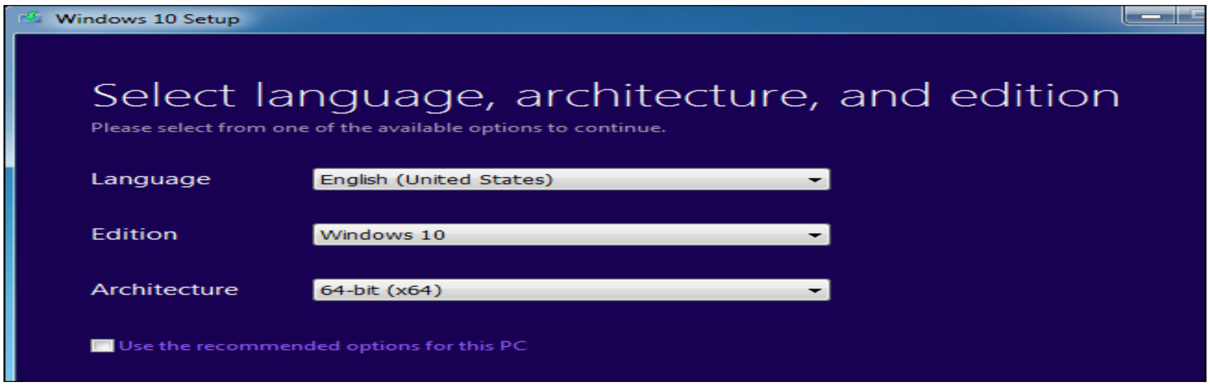
Windows 10 Setup

What do you want to do?

Upgrade this PC now

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

Be sure to select the language, edition, and architecture you want to install of Windows 10. If you're installing it on a PC with a 64-bit CPU, you probably want the 64-bit version. If you're installing it on a PC with a 32-bit CPU, you'll need the 32-bit version. If you're installing Windows 10 on the current PC, just keep the "Use the recommended options for this PC" box checked and the tool will automatically download the correct version for your current PC.


Windows 10 Setup

Select language, architecture, and edition

Please select from one of the available options to continue.

Language: English (United States)

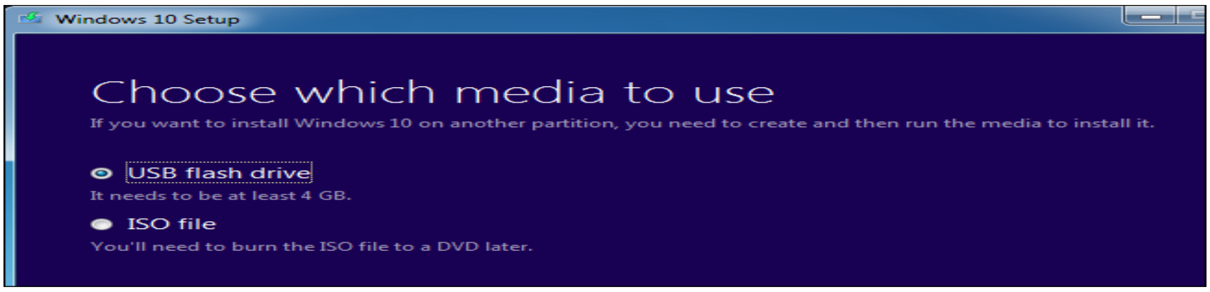
Edition: Windows 10

Architecture: 64-bit (x64)

Use the recommended options for this PC

The tool will allow you to copy the Windows 10 installation files to a USB drive or burn them to a DVD. If you're using a USB drive, it must be 4 GB or larger in size. All files on the USB drive will be erased as part of this process.

If you want to install Windows 10 in a virtual machine, select the "ISO file" option here. The tool will download an ISO file, and you can then boot the downloaded ISO in a virtual machine to install Windows 10 inside it


Windows 10 Setup

Choose which media to use

If you want to install Windows 10 on another partition, you need to create and then run the media to install it.

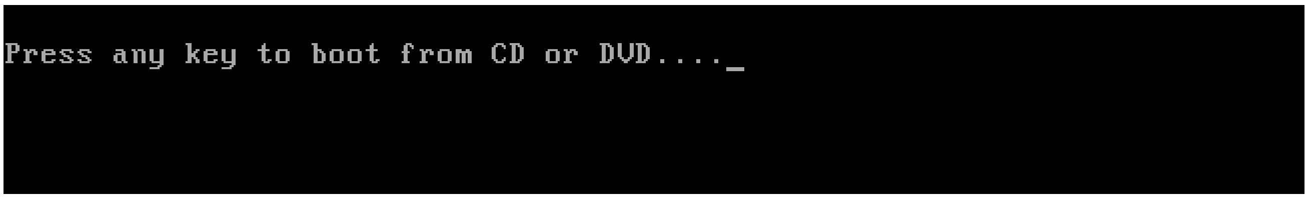
USB flash drive

It needs to be at least 4 GB.

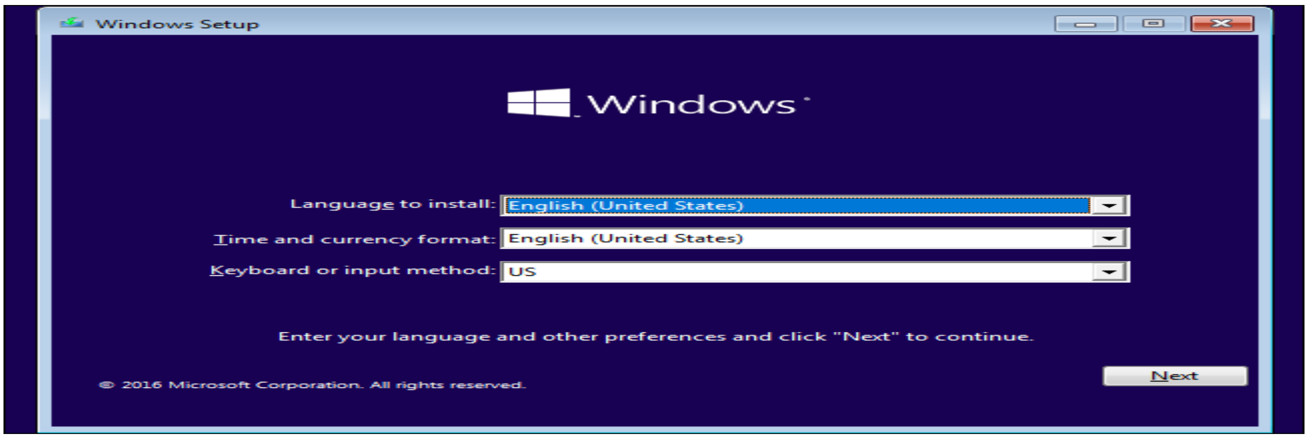
ISO file

You'll need to burn the ISO file to a DVD later.

Once you've created installation media, you'll need to insert it into the PC you want to install Windows 10 on. You then boot from the installation media. This may require modifying the boot order in your PC's BIOS or UEFI firmware.


```
Press any key to boot from CD or DVD...._
```

On the Windows Setup screen, select your language, time and currency format, and keyboard layout. Click "Next" to continue.


Windows Setup

Windows

Language to install: English (United States)

Time and currency format: English (United States)

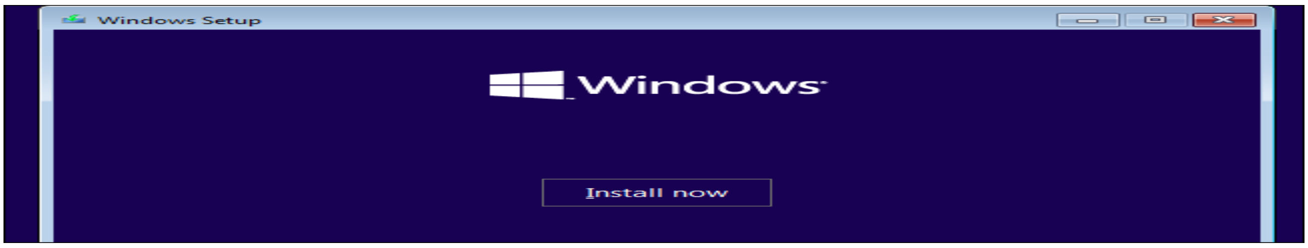
Keyboard or input method: US

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

© 2016 Microsoft Corporation. All rights reserved.

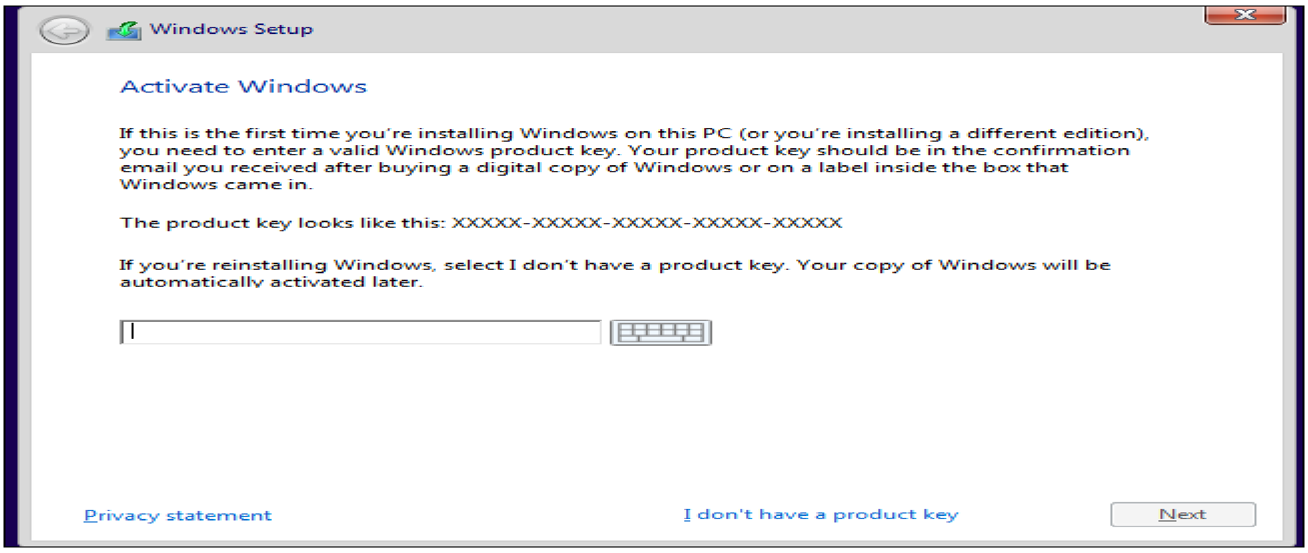
Next

When you reach the installer screen, select "Install Now" and follow the instructions to install Windows 10 on your PC.



When you see the Activate Windows screen, you'll need to either enter a key or skip it. You may not see this screen if Windows 10 automatically detects a key associated with your PC's hardware.

- If you've never installed and activated Windows 10 on this computer before, enter your Windows 10 key here. If you don't have one, but you have a valid Windows 7, 8, or 8.1 key, enter it here instead.
- If you've previously taken advantage of the free Windows 10 upgrade offer on this PC, click "I don't have a product key". Windows will automatically activate with a "digital license" associated with your PC's hardware on Microsoft's servers once it's installed.



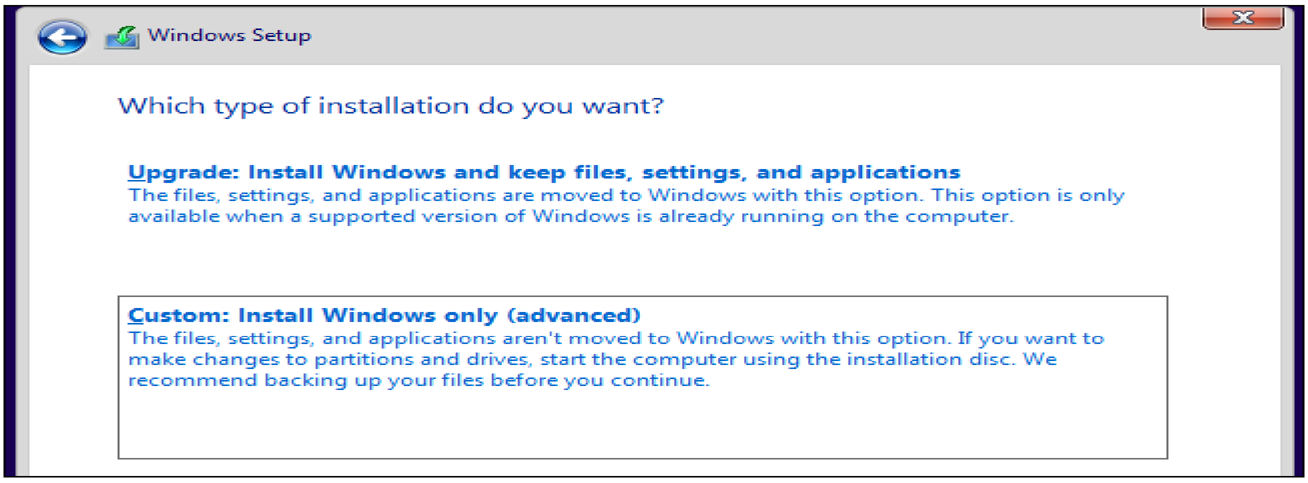
Activate Windows

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

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

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

When you reach the "Which type of installation do you want?" screen, click "Custom" to perform a clean installation and remove everything on your PC. (If you've changed your mind and want to upgrade your existing installation, you can click "Upgrade".)



Which type of installation do you want?

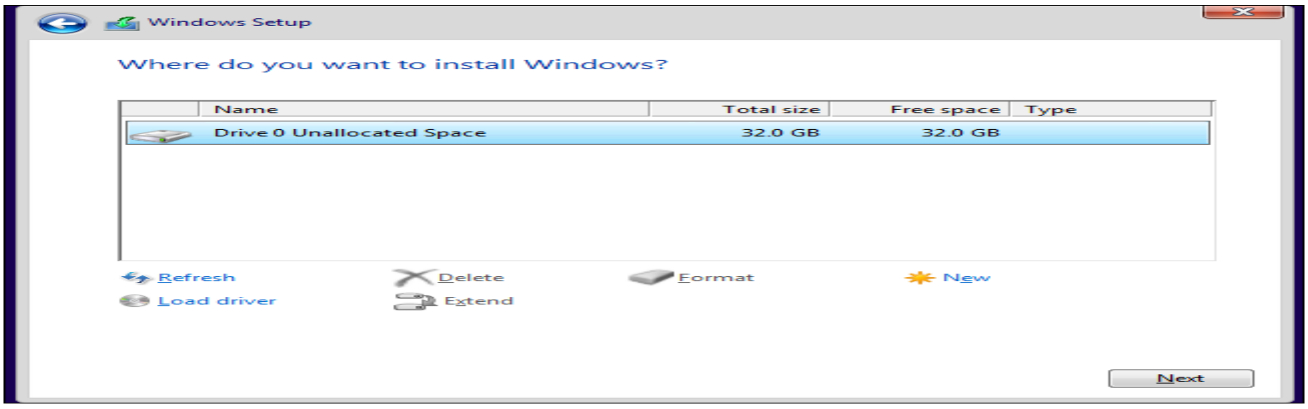
**Upgrade: Install Windows and keep files, settings, and applications**
The files, settings, and applications are moved to Windows with this option. This option is only available when a supported version of Windows is already running on the computer.

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

On the next screen, select the hard drive you want to install Windows on and erase it. If you have multiple partitions on that drive, you may want to erase those as well.

**Warning**: When you delete a partition, you're also deleting all the files on that partition. Be sure you have backups of any important files before doing this!

When you're done erasing partitions, you should have a big block of "Unallocated Space". Select that, click "New", and once it's formatted your drive, click Next.

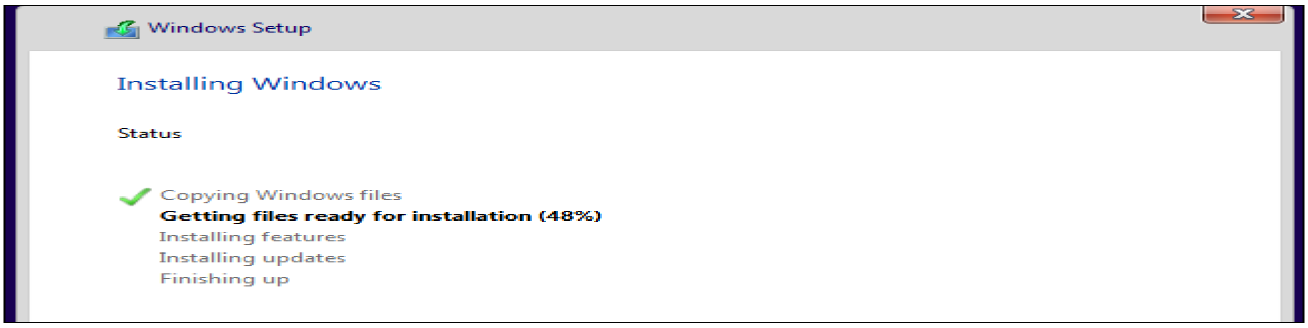


Windows Setup

Where do you want to install Windows?

| | Name | Total size | Free space | Type |
|---|---|---|---|---|
| | Drive 0 Unallocated Space | 32.0 GB | 32.0 GB | |

Windows 10 will install itself, and may restart a few times during this process. When it's done, you'll see the normal setup interface you see when setting up Windows 10 on any new PC, where you can add user accounts and adjust various settings.



Windows Setup

Installing Windows

Status

✓ Copying Windows files
**Getting files ready for installation (48%)**
Installing features
Installing updates
Finishing up

**Unix/Linux:** The UNIX operating system is well known for its stability. UNIX is often used more as a server than a workstation. Linux was based on the UNIX system, with the source code being a part of GNU open-source project. Both systems are very secure yet far more complex than Windows.

## Experiment 1.2:- installation processes for Any Linux Distribution

## AIM: to discuss installation processes for Any Linux Distribution

### Installation process for Any Linux Distribution

1. **Download the Linux distribution of your choice.** If you're new to Linux, consider trying a lightweight and easy to use distribution, such as Ubuntu or Linux Mint. Linux distributions (known as "distros") are typically available for free to download in ISO

format. You can find the ISO for the distribution of your choice at the distribution's website. This format needs to be burned to a CD or USB stick before you can use it to install Linux. This will create a Live CD or Live USB.

- ✓ A Live CD or Live USB is a disk that you can boot into, and often contains a preview version of the operating system that can be run directly from the CD or USB stick.
- ✓ Install an image burning program, or use your system's built-in burning tool if you are using Windows 7, 8, or Mac OS X. Pen Drive Linux and UNetBootin are two popular tools for burning ISO files to USB sticks.

2. **Boot into the Live CD or Live USB.** Most computers are set to boot into the hard drive first, which means you will need to change some settings to boot from your newly-burned CD or USB. Start by rebooting the computer.

   Once the computer reboots, press the key used to enter the boot menu. The key for your system will be displayed on the same screen as the manufacturer's logo. Typical keys include F12, F2, or Del.

   a. For Windows 8 users, hold the Shift key and click restart. This will load the Advanced Startup Options, where you can boot from CD.
   b. For Windows 10 users, go to advanced boot in settings and click "Restart Now."
   c. If your computer doesn't give you direct access to the boot menu from the manufacturer's splash screen, it's most likely hidden in the BIOS menu. You can access the BIOS menu in the same way that you would get to the boot menu. At the manufacturer splash screen, the key should be listed in one of the bottom corners.

Once you're in the boot menu, select your live CD or USB. Once you've changed the settings, save and exit the BIOS setup or boot menu. Your computer will continue with the boot process.

3. **Try out the Linux distribution before installing.** Most Live CDs and USBs can launch a "live environment", giving you the ability to test it out before making the switch. You won't be able to create files, but you can navigate around the interface and decide if it's right for you.

4. **Start the installation process.** If you're trying out the distro, you can launch the installation from the application on the desktop. If you decided not to try out the distribution, you can start the installation from the boot menu.

- You will be asked to configure some basic options, such as language, keyboard layout, and timezone.

5. **Create a username and password.** You will need to create login information to install Linux. A password will be required to log into your account and perform administrative tasks.
6. **Set up the partition.** Linux needs to be installed on a separate partition from any other operating systems on your computer if you intend dual booting Linux with

another OS. A partition is a portion of the hard drive that is formatted specifically for that operating system. You can skip this step if you don't plan on dual booting.

- Distros such as Ubuntu will set a recommended partition automatically. You can then adjust this manually yourself. Most Linux installations require at least 20 GB, so be sure to set aside enough room for both the Linux operating system and any other programs you may install and files you may create.
- If the installation process does not give you automatic partitions, make sure that the partition you create is formatted as Ext4. If the copy of Linux you are installing is the only operating system on the computer, you will most likely have to manually set your partition size.

7. **Boot into Linux.** Once the installation is finished, your computer will reboot. You will see a new screen when your computer boots up called "GNU GRUB". This is a boot loader that handles Linux installations. Pick your new Linux distro from the list. This screen may not show up if you only have one operating system on your computer. If this screen isn't being presented to you automatically, then you can get it back by hitting shift right after the manufacturer splash screen.

- If you install multiple distros on your computer, they will all be listed here.

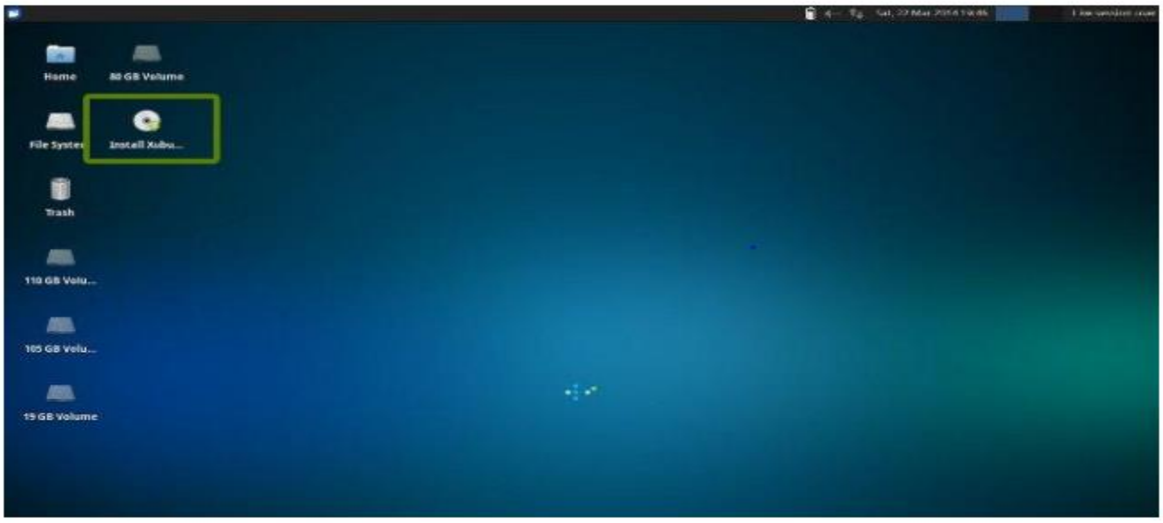
8. **Check your hardware.** Most hardware should work out of the box with your Linux distro, though you may need to download some additional drivers to get everything working.

- Some hardware requires proprietary drivers to work correctly in Linux. This is most common with graphics cards. There is typically an open source driver that will work, but to get the most out of your graphics cards you will need to download the proprietary drivers from the manufacturer.
- In Ubuntu, you can download proprietary drivers through the System Settings menu. Select the Additional Drivers option, and then select the graphics driver from the list. Other distros have specific methods for obtaining extra drivers.
- You can find other drivers from this list as well, such as Wi-Fi drivers.

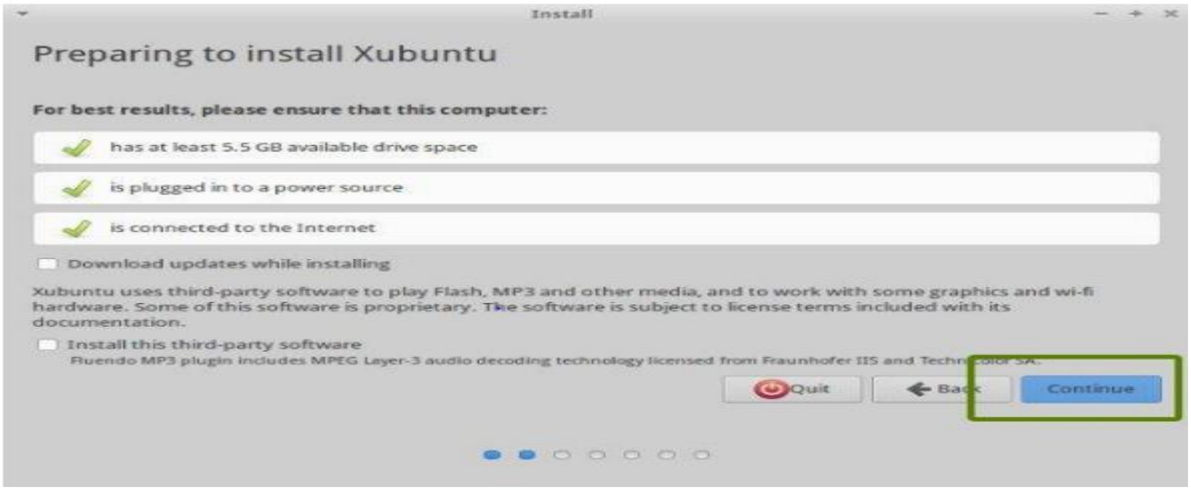
9. **Start using Linux.** Once your installation is complete and you've verified that your hardware is working, you're ready to start using Linux. Most distros come with several popular programs installed, and you can download many more from their respective file repositories.

## Experiment 1.3:- installation processes for Ubuntu

**Step 1: Create a live Ubuntu USB or disk if it is not available**
Download and create a live USB or DVD. In Windows, you can use tools like "universal USB installer "to create a live Ubuntu USB
**Step 2: Boot in to live USB**
Plug the live USB or disk in to the computer and restart the computer. While booting the computer press F10 or F12 function key (defers from computer to computer) to go to the boot menu. Now, choose the option to boot from USB or CD/DVD.
**Step 3: Start the installation**
It will take some time to boot in to the live USB or disk. Once booted, you will be immediately provided with option to either try Ubuntu or install Ubuntu. Even if you choose to try, you can find the option to install on the desktop:



First few screens are pretty straight forward. Just choose press continue:


Install

Preparing to install Xubuntu

For best results, please ensure that this computer:

✓ has at least 5.5 GB available drive space

✓ is plugged in to a power source

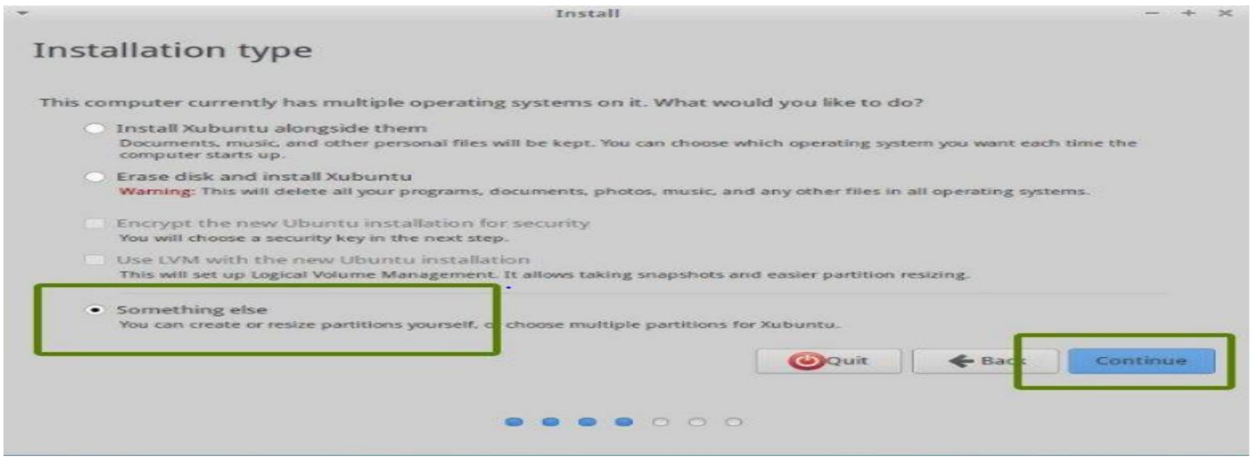
✓ is connected to the Internet

☐ Download updates while installing

Xubuntu uses third-party software to play Flash, MP3 and other media, and to work with some graphics and wi-fi hardware. Some of this software is proprietary. The software is subject to license terms included with its documentation.

☐ Install this third-party software
Fluendo MP3 plugin includes MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Technicolor SA.

**Step 4: Prepare the partition**
This is the most important part of the whole dual boot installation. Where to install Ubuntu? Windows is already installed here, so, we'll prepare a new partition for Ubuntu. In the Installation Type window, choose Something Else:

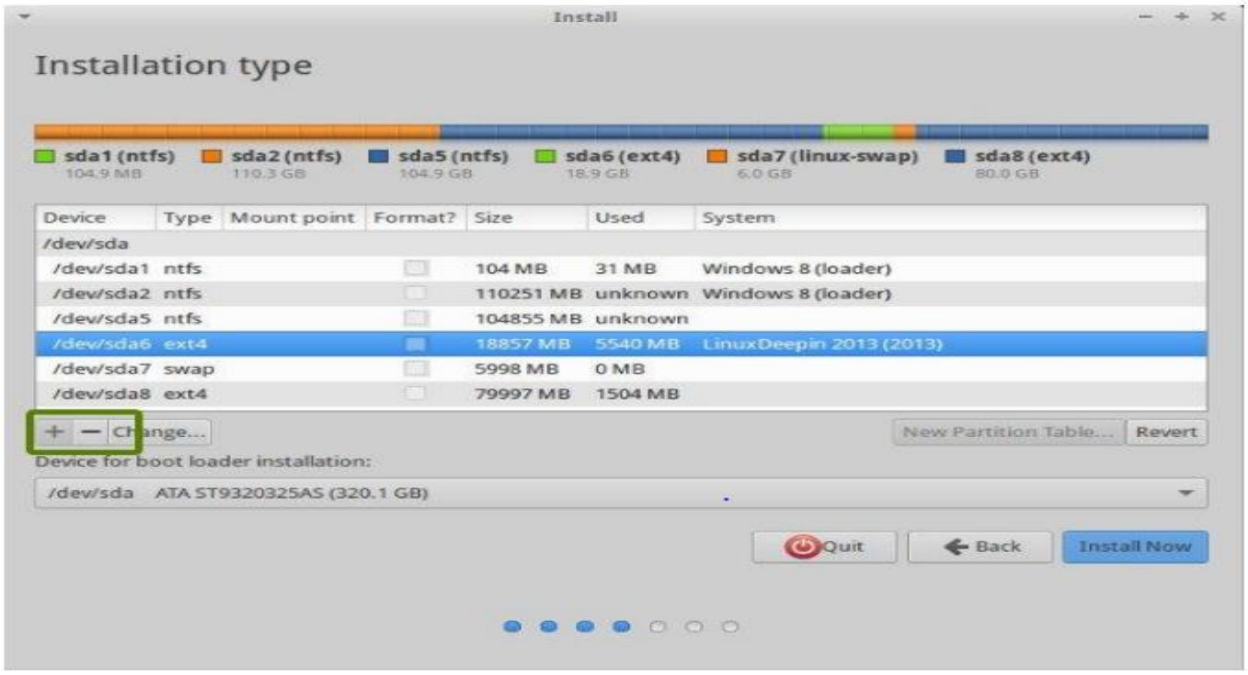


Installation type

This computer currently has multiple operating systems on it. What would you like to do?

- ○ Install Xubuntu alongside them
  Documents, music, and other personal files will be kept. You can choose which operating system you want each time the computer starts up.
- ○ Erase disk and install Xubuntu
  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 Xubuntu.

As you can see, it has 3 NTFS and some ext4 partitions. One of the NTFS partition consists of Windows installation. This should be untouched if you want to keep your Windows installation safe.
If there is no “free space”, you need to delete a NTFS or existing ext4 partition and create some free space. This will delete all the data in that partition
Click on the desired partition and press the – to delete the partition.



Installation type

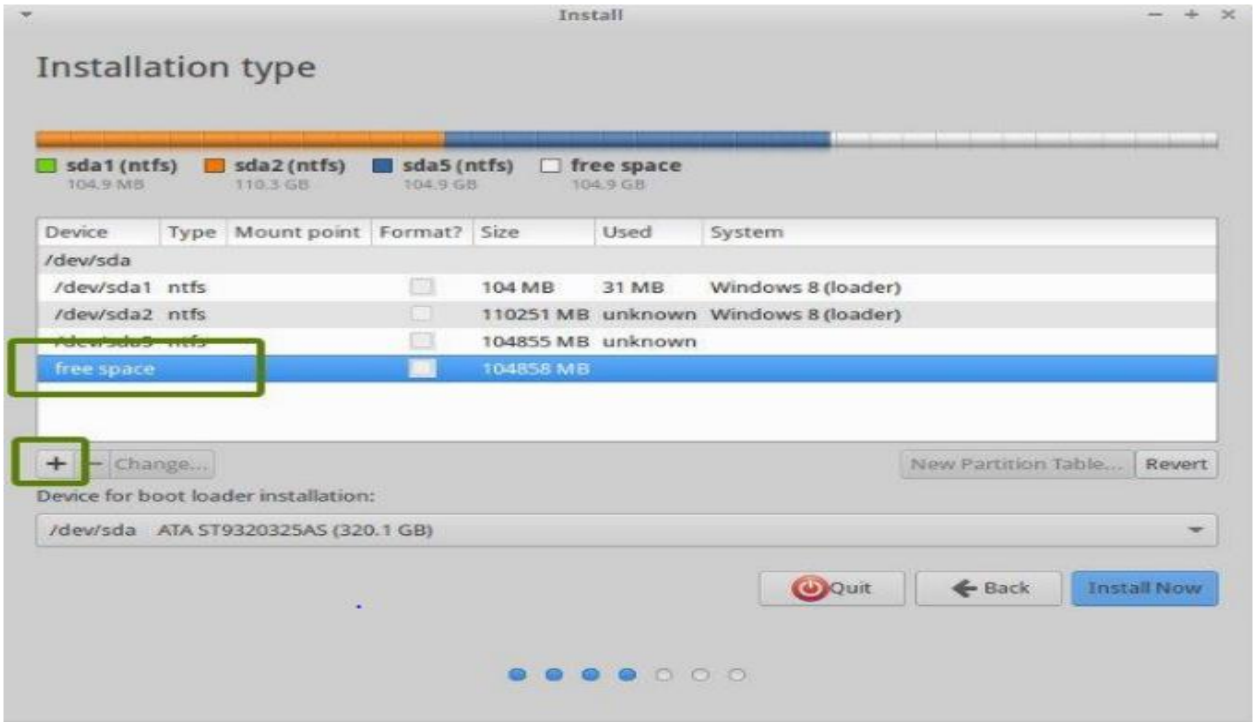
🟩 sda1 (ntfs) 104.9 MB 🟧 sda2 (ntfs) 110.3 GB 🟦 sda5 (ntfs) 104.9 GB 🟩 sda6 (ext4) 18.9 GB 🟧 sda7 (linux-swap) 6.0 GB 🟦 sda8 (ext4) 80.0 GB

| Device | Type | Mount point | Format? | Size | Used | System |
|---|---|---|---|---|---|---|
| /dev/sda | | | | | | |
| /dev/sda1 | ntfs | | ☐ | 104 MB | 31 MB | Windows 8 (loader) |
| /dev/sda2 | ntfs | | ☐ | 110251 MB | unknown | Windows 8 (loader) |
| /dev/sda5 | ntfs | | ☐ | 104855 MB | unknown | |
| /dev/sda6 | ext4 | | ☐ | 18857 MB | 5540 MB | LinuxDeepin 2013 (2013) |
| /dev/sda7 | swap | | ☐ | 5998 MB | 0 MB | |
| /dev/sda8 | ext4 | | ☐ | 79997 MB | 1504 MB | |

Device for boot loader installation:

/dev/sda ATA ST9320325AS (320.1 GB)

**Step 5: Create root, swap and home**
Once you have some free space on your hard drive, it is time to install Ubuntu on it. Now, there are several ways to do it. We have to create minimum a Root, a Swap and a Home partition.
Create a root partition first. Choose the free space available and click on +.


Install

Installation type

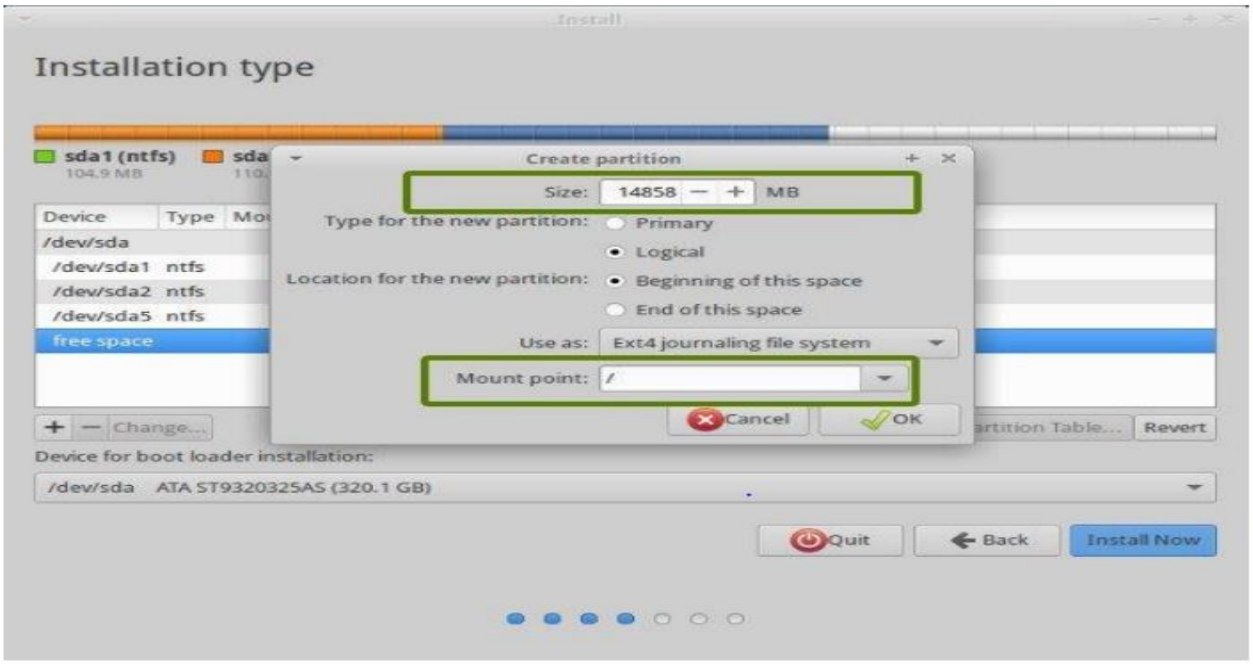
🟩 sda1 (ntfs) 104.9 MB | 🟧 sda2 (ntfs) 110.3 GB | 🟦 sda5 (ntfs) 104.9 GB | ☐ free space 104.9 GB

| Device | Type | Mount point | Format? | Size | Used | System |
|---|---|---|---|---|---|---|
| /dev/sda | | | | | | |
| /dev/sda1 | ntfs | | ☐ | 104 MB | 31 MB | Windows 8 (loader) |
| /dev/sda2 | ntfs | | ☐ | 110251 MB | unknown | Windows 8 (loader) |
| /dev/sda5 | ntfs | | ☐ | 104855 MB | unknown | |
| free space | | | ☐ | 104858 MB | | |

Device for boot loader installation:

/dev/sda ATA ST9320325AS (320.1 GB)

Here, choose the size of root directory, choose ext4file system, and mount point as / (i.e. root):


Install

Installation type

Create partition

Size: 14858 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

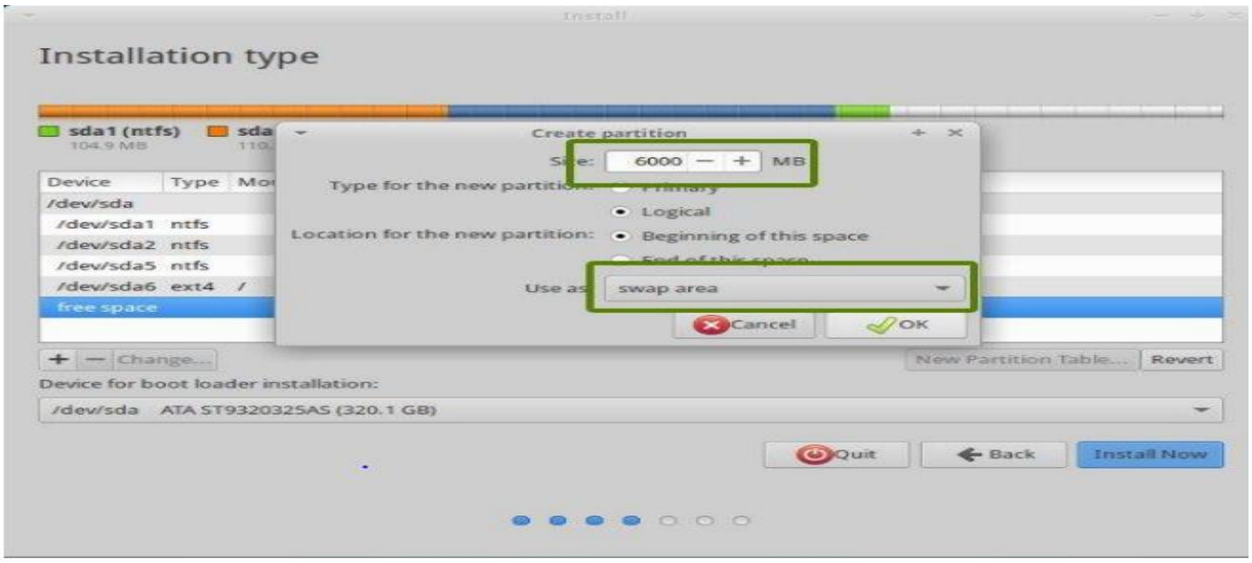
Mount point: /

| Device | Type |
|---|---|
| /dev/sda | |
| /dev/sda1 | ntfs |
| /dev/sda2 | ntfs |
| /dev/sda5 | ntfs |
| free space | |

Device for boot loader installation:

/dev/sda ATA ST9320325AS (320.1 GB)

Next step is to create swap partition. It is advised by many that Swap should be double of your system's RAM size. You can choose the swap size accordingly.


Installation type

🟩 sda1 (ntfs) 104.9 MB 🟧 sda 110.

| Device | Type | Mo |
|---|---|---|
| /dev/sda | | |
| /dev/sda1 | ntfs | |
| /dev/sda2 | ntfs | |
| /dev/sda5 | ntfs | |
| /dev/sda6 | ext4 | / |
| free space | | |

Create partition

Size: 6000 MB

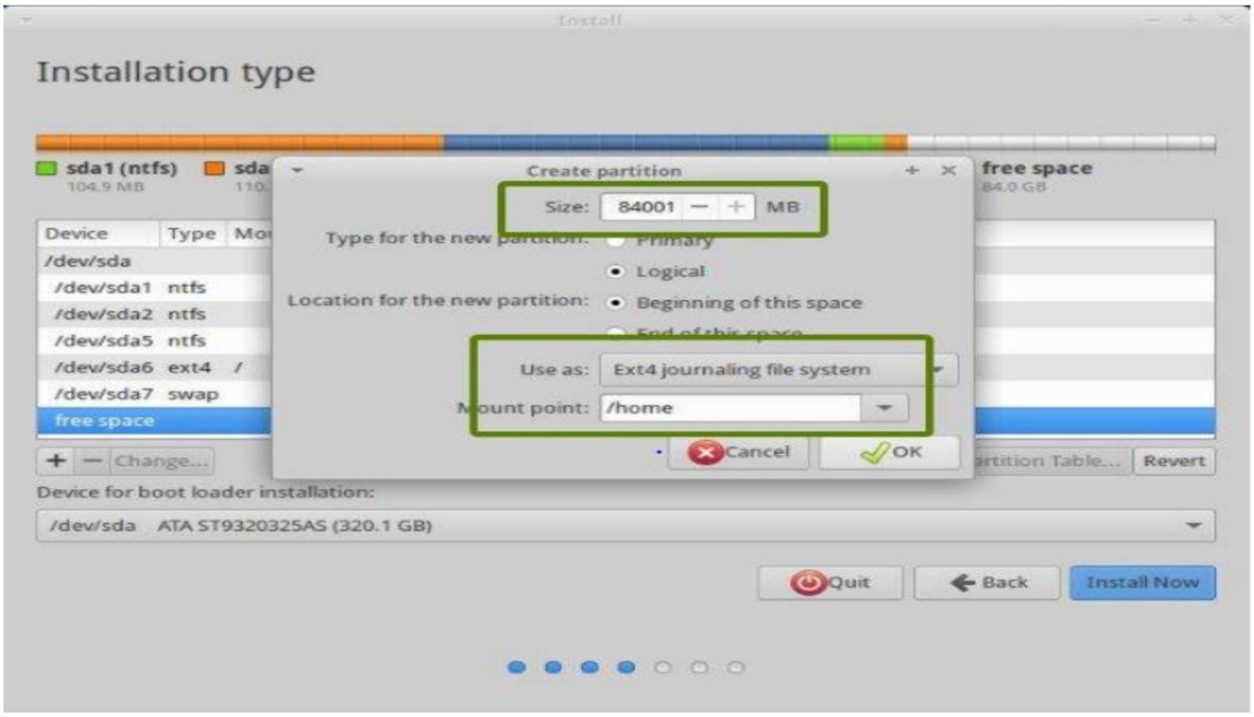
Type for the new partition: Primary [✓ Logical]

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

Use as: swap area

Device for boot loader installation: /dev/sda ATA ST9320325AS (320.1 GB)

The next step is to create Home. Try to allocate the maximum size to Home because this is where you'll be downloading and keeping the files.


Installation type

🟩 sda1 (ntfs) 104.9 MB 🟧 sda 110. free space 84.0 GB

| Device | Type | Mo |
|---|---|---|
| /dev/sda | | |
| /dev/sda1 | ntfs | |
| /dev/sda2 | ntfs | |
| /dev/sda5 | ntfs | |
| /dev/sda6 | ext4 | / |
| /dev/sda7 | swap | |
| free space | | |

Create partition

Size: 84001 MB

Type for the new partition: Primary [✓ Logical]

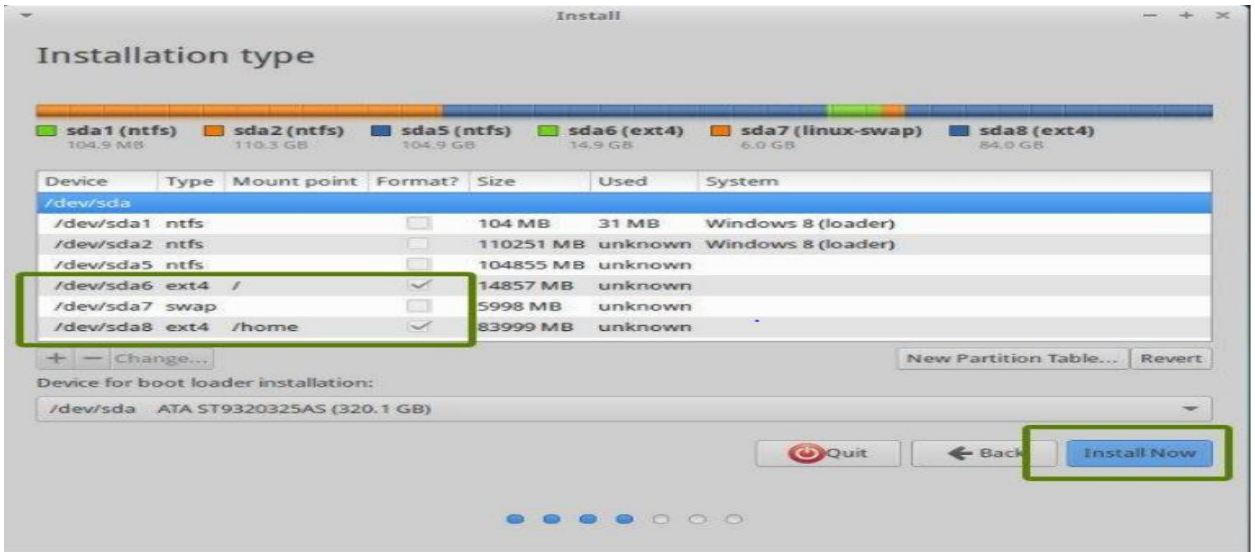
Location for the new partition: [✓ Beginning of this space] End of this space

Use as: Ext4 journaling file system

Mount point: /home

Device for boot loader installation: /dev/sda ATA ST9320325AS (320.1 GB)

Once you have created Root, Swap and Home partitions, click on Install Now button.


Installation type

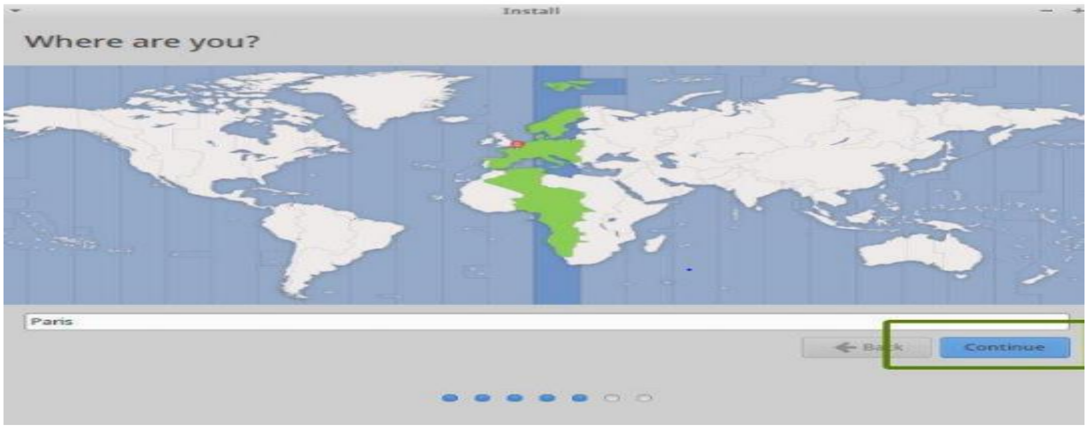
🟩 sda1 (ntfs) 104.9 MB 🟧 sda2 (ntfs) 110.3 GB 🟦 sda5 (ntfs) 104.9 GB 🟩 sda6 (ext4) 14.9 GB 🟧 sda7 (linux-swap) 6.0 GB 🟦 sda8 (ext4) 84.0 GB

| Device | Type | Mount point | Format? | Size | Used | System |
|---|---|---|---|---|---|---|
| /dev/sda | | | | | | |
| /dev/sda1 | ntfs | | | 104 MB | 31 MB | Windows 8 (loader) |
| /dev/sda2 | ntfs | | | 110251 MB | unknown | Windows 8 (loader) |
| /dev/sda5 | ntfs | | | 104855 MB | unknown | |
| /dev/sda6 | ext4 | / | ✓ | 14857 MB | unknown | |
| /dev/sda7 | swap | | | 5998 MB | unknown | |
| /dev/sda8 | ext4 | /home | ✓ | 83999 MB | unknown | |

Device for boot loader installation: /dev/sda ATA ST9320325AS (320.1 GB)

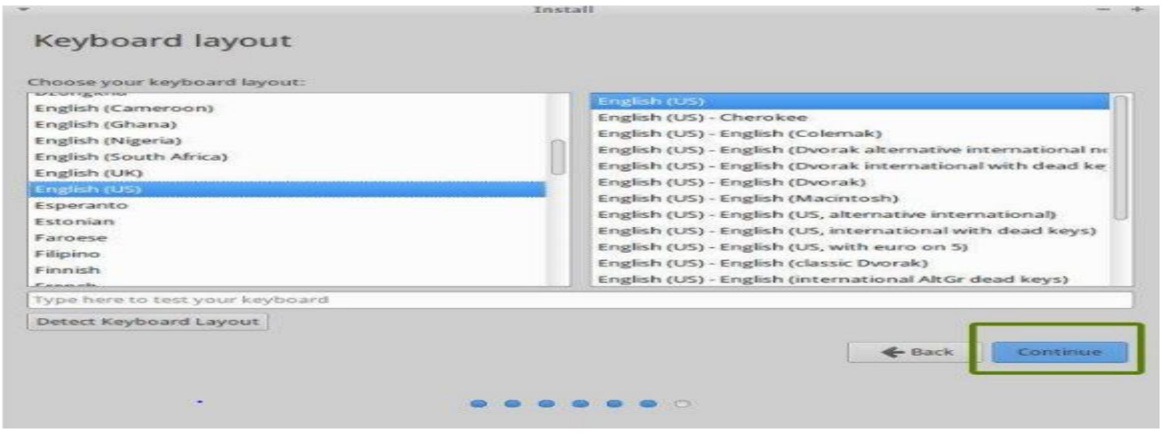
**Step 6: Follow the trivial instructions**

If you successfully created the partitions as mentioned above, you will be taken through a number of screens to select options like keyboard layout, login credentials etc.



Where are you?

Paris



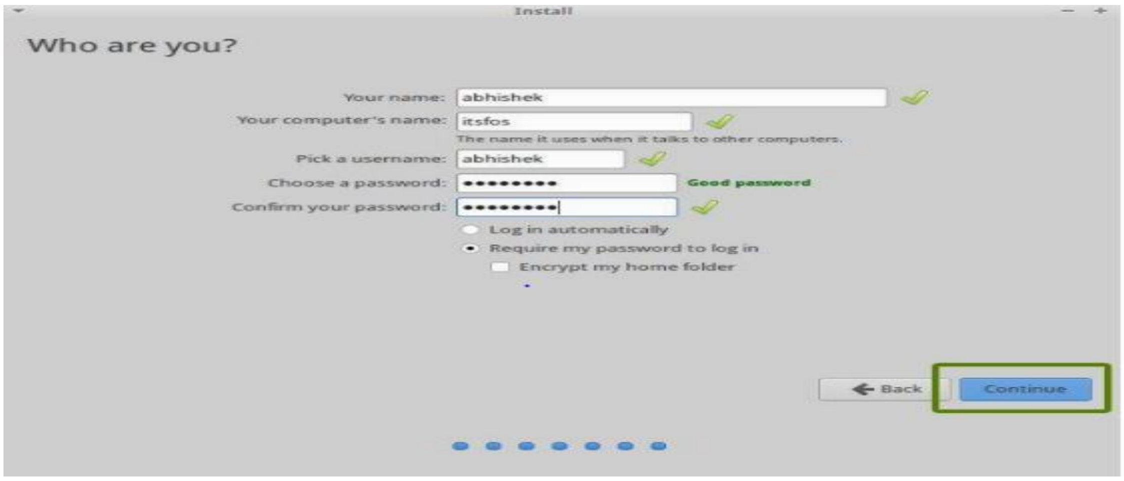
Keyboard layout

Choose your keyboard layout:

English (Cameroon)
English (Ghana)
English (Nigeria)
English (South Africa)
English (UK)
English (US)
Esperanto
Estonian
Faroese
Filipino
Finnish

English (US)
English (US) - Cherokee
English (US) - English (Colemak)
English (US) - English (Dvorak alternative international n
English (US) - English (Dvorak international with dead ke
English (US) - English (Dvorak)
English (US) - English (Macintosh)
English (US) - English (US, alternative international)
English (US) - English (US, international with dead keys)
English (US) - English (US, with euro on 5)
English (US) - English (classic Dvorak)
English (US) - English (international AltGr dead keys)

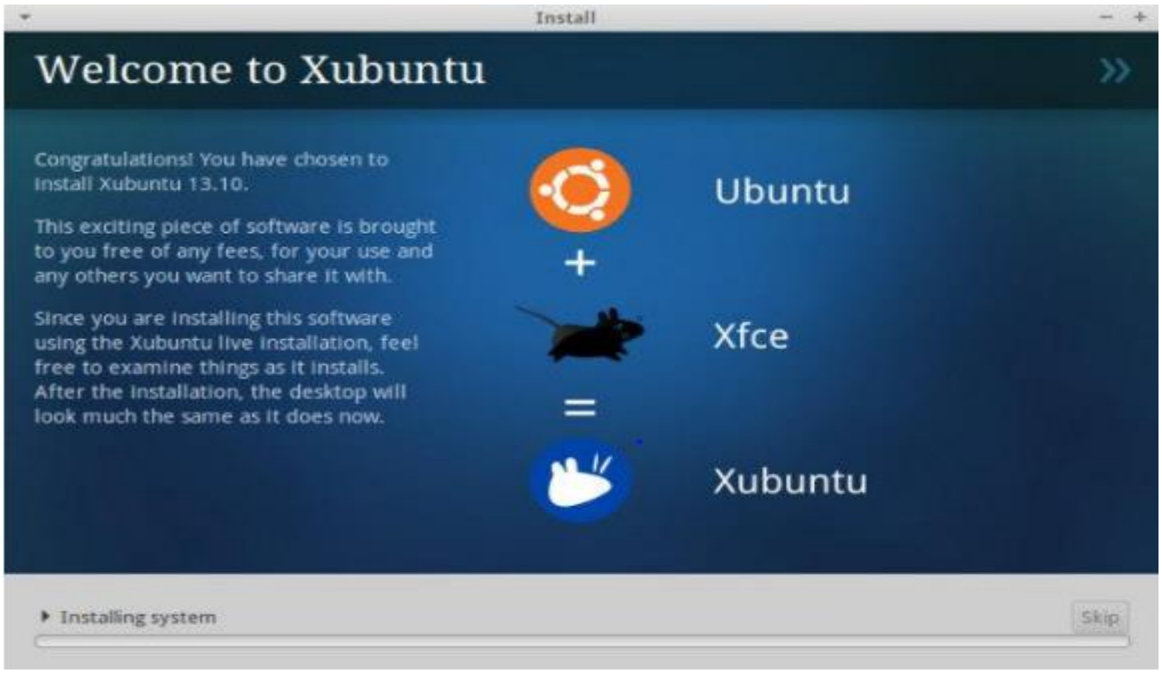
Type here to test your keyboard



Who are you?

Your name: abhishek
Your computer's name: itsfos
The name it uses when it talks to other computers.
Pick a username: abhishek
Choose a password: ●●●●●●●●● Good password
Confirm your password: ●●●●●●●●●

Log in automatically
[✓ Require my password to log in]
Encrypt my home folder


Install

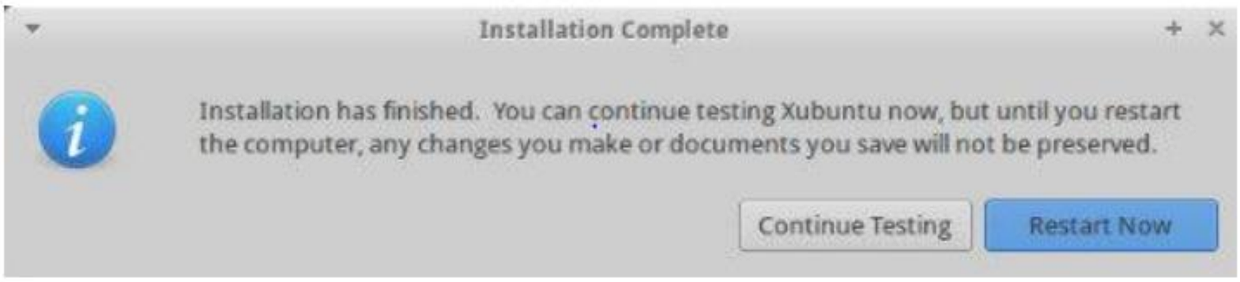
Welcome to Xubuntu

Congratulations! You have chosen to install Xubuntu 13.10.

This exciting piece of software is brought to you free of any fees, for your use and any others you want to share it with.

Since you are installing this software using the Xubuntu live installation, feel free to examine things as it installs. After the installation, the desktop will look much the same as it does now.

Once the installation is over, you will be presented with the option to keep trying live version or to restart the system.


Installation Complete

Installation has finished. You can continue testing Xubuntu now, but until you restart the computer, any changes you make or documents you save will not be preserved.

On next boot, you will see the option of Ubuntu in the grub screen.

**Macintosh:** Recent versions of the Macintosh operating system, including the Mac OS X, follow the secure architecture of UNIX. Systems developed by Apple are efficient and easy to use, but can only function on Apple branded hardware.

### When OS start its work?

Although operating systems differ, many of their basic functions are similar. Most users today have a computer with a hard disk drive. When the computer is turned on, the operating system will be loaded from the hard drive into the computer's memory, thus making it available for use. The process of loading the operating system into memory is called bootstrapping, or booting the system. The word booting is used because, figuratively speaking, the operating system pulls itself up by its own bootstraps.

When you turn on the power to a computer, the first program that runs is usually a set of instructions kept in the computer's read-only memory (ROM). This code examines the system hardware to make sure everything is functioning properly. This power-on self test (POST) checks the CPU, memory, and basic input-output systems (BIOS) for errors and stores the result in a special memory location. Once the POST has successfully completed, the software

loaded in ROM (sometimes called the BIOS or firmware) will begin to activate the computer's disk drives. In most modern computers, when the computer activates the hard disk drive, it finds the first piece of the operating system: the bootstrap loader.
The bootstrap loader is a small program that has a single function: It loads the operating system into memory and allows it to begin operation. In the most basic form, the bootstrap loader sets up the small driver programs that interface with and control the various hardware subsystems of the computer. It sets up the divisions of memory that hold the operating system, user information and applications. Then it turns control of the computer over to the operating system.

## EXERCISE

## Discuss on the following portions

1. History of Operating system
2. Operating system design issues
3. Principles of operating systems

# Chapter Two: Process Management

## Process Description

Modern computers work in a multitasking environment in which they execute multiple programs simultaneously. These programs cooperate with each other and share the same resource, such as memory and CPU. An operating system manages all processes to facilitate proper utilization of these resources.

### Process Scheduling:

When more than one process is running, the operating system must decide which one should first run. The part of the operating system concerned with this decision is called the **scheduler**, and algorithm it uses is called the **scheduling algorithm**.

In multitasking and uniprocessor system scheduling is needed because more than one process is in ready and waiting state. A certain scheduling algorithm is used to get all the processes to run correctly. The processes should be in such a manner that no processes must be made to wait for a long time.

### Scheduling Algorithms

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

The following are some scheduling algorithms:

- First Come First Served (FCFS) Scheduling.
- Shortest Job First (SJF) Scheduling.
- Shortest Remaining Time (SRT) Scheduling.
- Round Robin Scheduling.
- Priority Scheduling.

## Exercise

**Why is process management important?**

## Experiment 2:-CPU SCHEDULING ALGORITHMS

### 2.1 ROUND ROBIN SCHEDULING

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

**PROBLEM DESCRIPTION:**

CPU scheduler will decide which process should be given the CPU for its execution.

For this it use different algorithm to choose among the process. One among that algorithm is Round robin algorithm.

In this algorithm we are assigning some time slice .The process is allocated according to the time slice, if the process service time is less than the time slice then process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue.

If the CPU burst of the currently running process is longer than time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed and the process will be put at the tail of the ready queue.

**ALGORITHM**:

Step 1: Initialize all the structure elements

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

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

i) The waiting time for first instance of a process is calculated as: a[i].waittime=count + a[i].arrivt

ii) The waiting time for the rest of the instances of the process is calculated as:

a) If the time quantum is greater than the remaining burst time then waiting time is calculated as: a[i].waittime=count + tq

b) Else if the time quantum is greater than the remaining burst time then waiting time is calculated as: a[i].waittime=count - remaining burst time

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

Step 5: Print the results of the step 4.

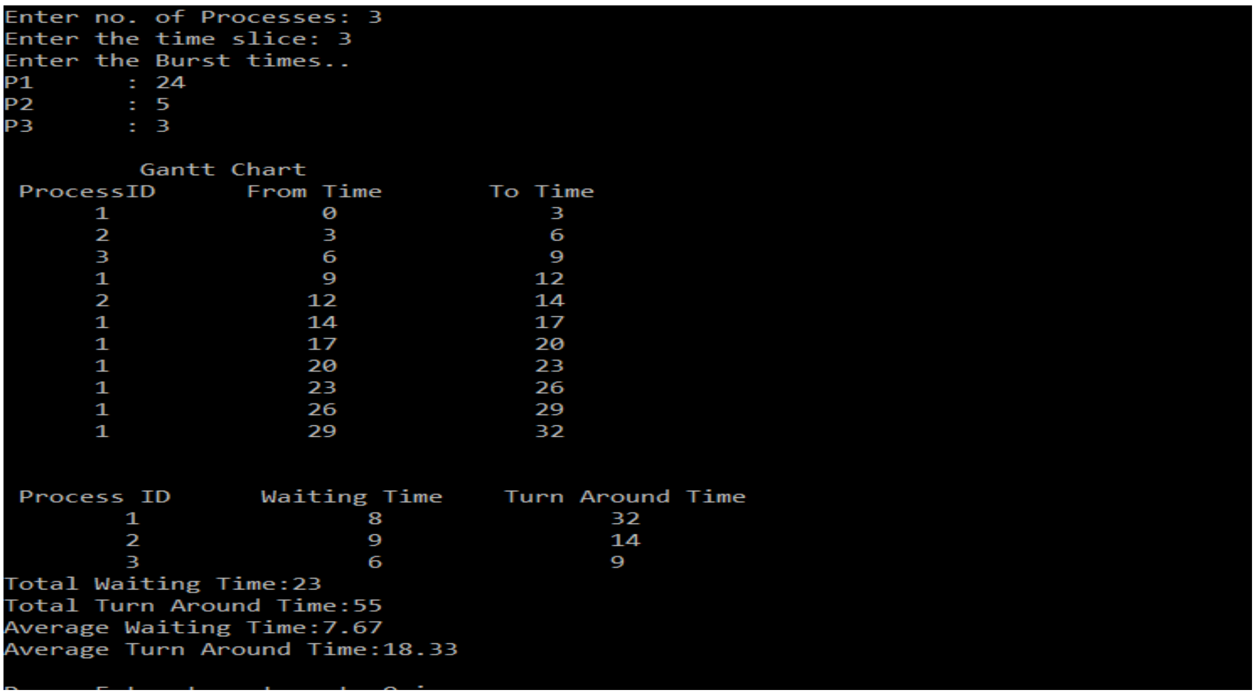
**PROGRAM CODING:**

```
#include<stdio.h>

int main()

{ int i,tbt=0,nop,ts=0,flag[20], rem[20];

int from,wt[20],tt[20],b[20], twt=0,ttt=0;

int dur;

float awt,att;

printf("Enter no. of Processes: ");

scanf("%d",&nop);

printf("Enter the time slice: ");
```

```
scanf("%d",&ts);
printf("Enter the Burst times..\n");
for(i=0;i<nop;i++)
{ wt[i]=tt[i]=0;
printf("P%d\t: ",i+1);
scanf("%d",&b[i]);
rem[i]=b[i];
tbt+=b[i];
flag[i]=0;
}
from=0;
i=0;
printf("\n\t Gantt Chart");
printf("\n ProcessID\tFrom Time\tTo Time\n");
while(from<tbt)
{
if(!flag[i])
{
if(rem[i]<=ts)
{
dur=rem[i];
flag[i]=1;
tt[i]=dur+from;
wt[i]=tt[i]-b[i];
}
else
dur=ts;
printf("%7d%15d%15d\n",i+1, from,from+dur);
rem[i] -= dur;
from += dur;
}
i=(i+1)%nop;
}
```

```
for(i=0;i<nop;i++)
{
twt+=wt[i];
ttt+=tt[i];
}
printf("\n\n Process ID \t Waiting Time \t Turn Around Time");
for(i=0;i<nop;i++)
{
printf("\n\t%d\t\t%d\t\t%d",i+1,wt[i],tt[i]);
}
awt=(float)twt/(float)nop;
att=(float)ttt/(float)nop;
printf("\nTotal Waiting Time:%d",twt);
printf("\nTotal Turn Around Time:%d",ttt);
printf("\nAverage Waiting Time:%.2f",awt);
printf("\nAverage Turn Around Time:%.2f\n",att);
}
```

**OUTPUT**:



```
Enter no. of Processes: 3
Enter the time slice: 3
Enter the Burst times..
P1      : 24
P2      : 5
P3      : 3

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


 Process ID      Waiting Time    Turn Around Time
        1              8               32
        2              9               14
        3              6               9
Total Waiting Time:23
Total Turn Around Time:55
Average Waiting Time:7.67
Average Turn Around Time:18.33
```

**Exercise**

1. **Write the above program using other programming languages i.e. C++, JAVA,C# and others.**
2. **What makes this scheduling algorithm different from other scheduling algorithms?**

## 2.2 SHORTEST JOB FIRST.

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

**PROBLEM DESCRIPTION:**

CPU scheduler will decide which process should be given the CPU for its execution.

For this it uses different algorithm to choose among the process. One among that algorithm is SJF algorithm.

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

ALGORITHM:

Step 1: Get the number of process.

Step 2: Get the id and service time for each process.

Step 3: Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.

Step 4: Calculate the total time and waiting time of remaining process.

Step 5: Waiting time of one process is the total time of the previous process.

Step 6: Total time of process is calculated by adding the waiting time and service time of each process.

Step 7: Total waiting time calculated by adding the waiting time of each process.

Step 8: Total turnaround time calculated by adding all total time of each process.

Step 9: Calculate average waiting time by dividing the total waiting time by total number of process.

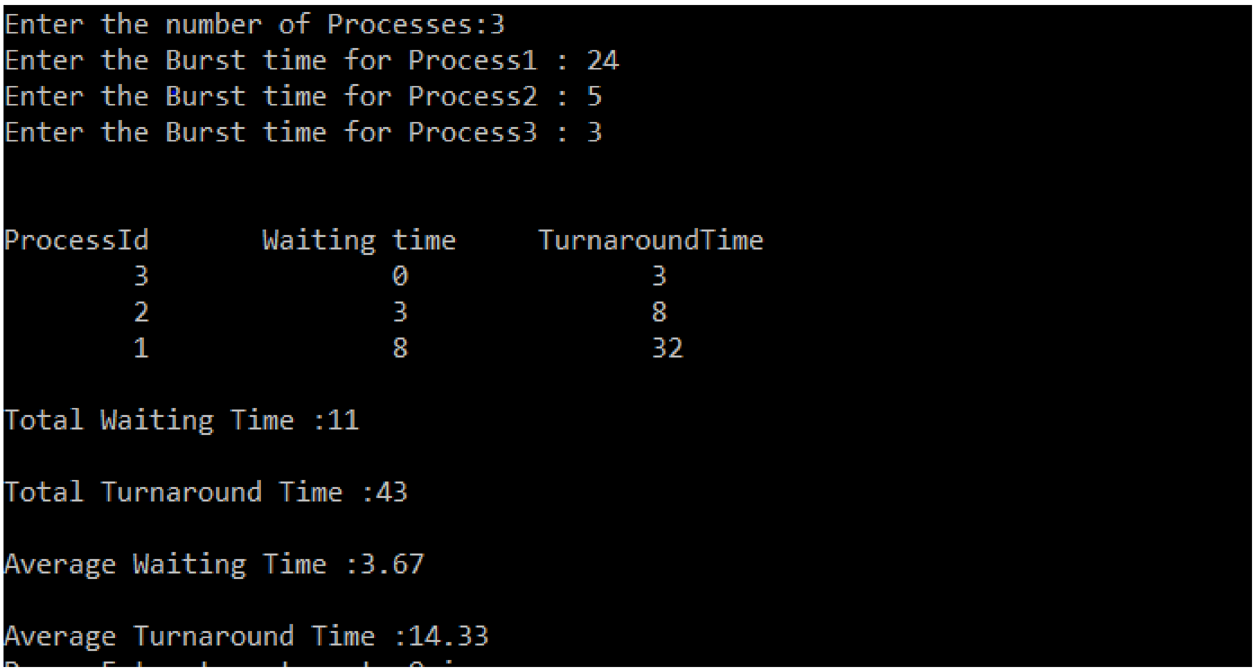
Step 10: Calculate average turnaround time by dividing the total waiting time by total number of process.

Step 11: Display the result.

**PROGRAM CODING:**

```
#include<stdio.h>

int main ()

{

int n,w[100],tot[100],i,j,awt,atot; float avwt,avtot;

struct

{

int p,bt; }sjf[10],temp;

printf("Enter the number of Processes:"); scanf("%d",&n);

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

**OUTUT**



```
Enter the number of Processes:3
Enter the Burst time for Process1 : 24
Enter the Burst time for Process2 : 5
Enter the Burst time for Process3 : 3


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

Total Waiting Time :11

Total Turnaround Time :43

Average Waiting Time :3.67

Average Turnaround Time :14.33
```

## Exercise

1. **Write the above program using other programming languages i.e. C++, JAVA, C# and others.**
2. **What is the important features of this algorithm?**
3. **Does this algorithm important for accurate CPU utilization?**

## 2.3 FIRST COME FIRST SERVE WITHOUT ARRIVAL TIME

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

**PROBLEM DESCRIPTION:**

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

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

**ALGORITHM**:

Step 1: Create the number of process.

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

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

Step 4: Calculate the Total time and processing time for the remaining processes.

Step 5: Waiting time of one process is the Total time of the previous process.

Step 6: Total time of process is calculated by adding Waiting time and

Service time.

Step 7: Total waiting time is calculated by adding the waiting time for lack process.
Step 8: Total turnaround time is calculated by adding all total time of each process.
Step 9: Calculate Average waiting time by dividing the total waiting time by total number of process.

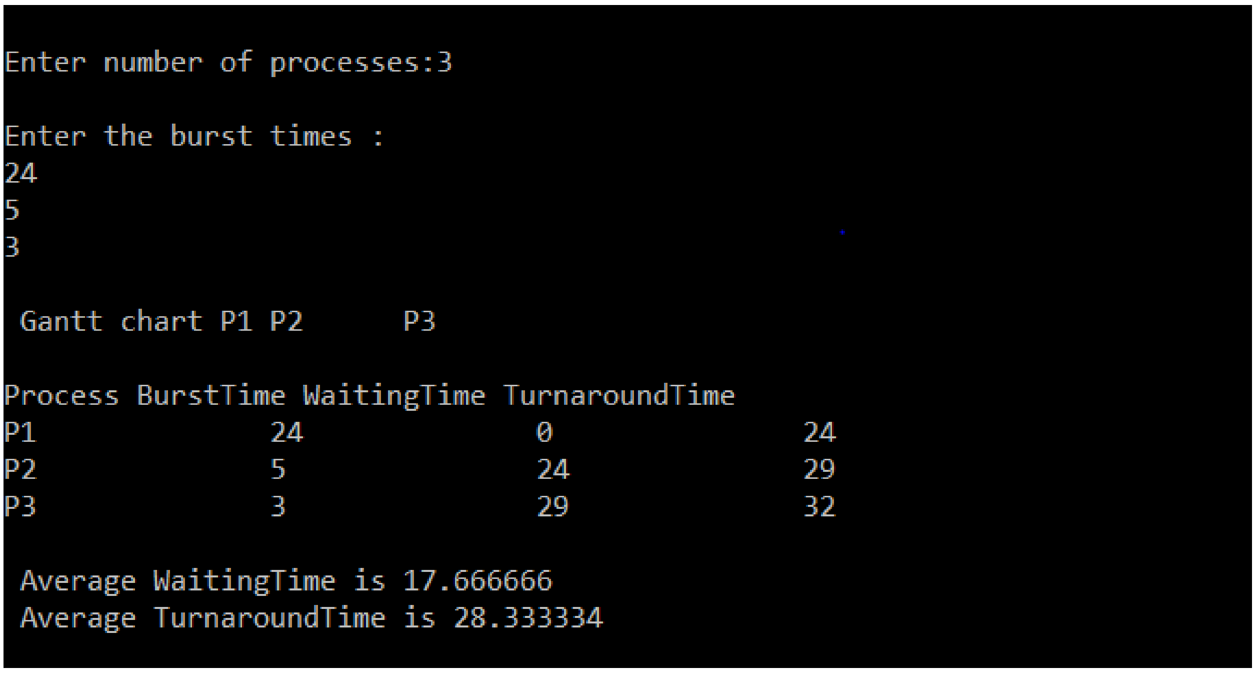
Step 10: Calculate Average turnaround time by dividing the total time by the number of process.

Step 11: Display the result.

**PROGRAM CODING:**

```
#include<stdio.h> int

int main()

{

int n,b[10],t=0,i,w=0,r=0,a=0; float

avg,avg1;

printf("\nEnter number of processes:");

scanf("%d",&n);

printf("\nEnter the burst times : \n");

for(i=1;i<=n;i++) scanf("%d",&b[i]);

printf("\n Gantt chart ");
for(i=1;i<=n;i++)
printf("P%d\t",i);
printf("\n\nProcess BurstTime WaitingTime TurnaroundTime\n");
for(i=1;i<=n;i++)
{
t=t+w;
r=r+b[i];
printf("P%d\t\t%d\t\t%d\t\t%d\t\t\n",i,b[i],w,r);
w=w+b[i];
a=a+r;
}
avg=(float)t/n;
avg1=(float)a/n;
printf("\n Average WaitingTime is %f",avg); printf("\n
Average TurnaroundTime is %f\n",avg1); return(0);
}
```

**OUTUT**



```
Enter number of processes:3

Enter the burst times :
24
5
3

 Gantt chart P1 P2      P3

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

 Average WaitingTime is 17.666666
 Average TurnaroundTime is 28.333334
```

## Exercise

1. **Write the above program using other programming languages i.e. C++, JAVA,C# and others.**
2. **Write a program for first come first serve with arrival time.**

## 2.4 PRIORITY SCHEDULING

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

**PROBLEM DESCRIPTION:**

CPU scheduler will decide which process should be given the CPU for its execution.

For this it uses different algorithm to choose among the process. One among that algorithm is Priority algorithm.

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

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

**ALGORITHM**:

Step 1: Get the number of process

Step 2: Get the id and service time for each process.

Step 3: Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.

Step 4: Calculate the total time and waiting time of remaining process.

Step 5: Waiting time of one process is the total time of the previous process.

Step 6: Total time of process is calculated by adding the waiting time and service time of each process.

Step 7: Total waiting time calculated by adding the waiting time of each process.

Step 8: Total turnaround time calculated by adding all total time of each process.

Step 9: Calculate average waiting time by dividing the total waiting time by total number of process.

Step 10: Calculate average turnaround time by dividing the total waiting time by total number of process.

Step 11: Display the result.

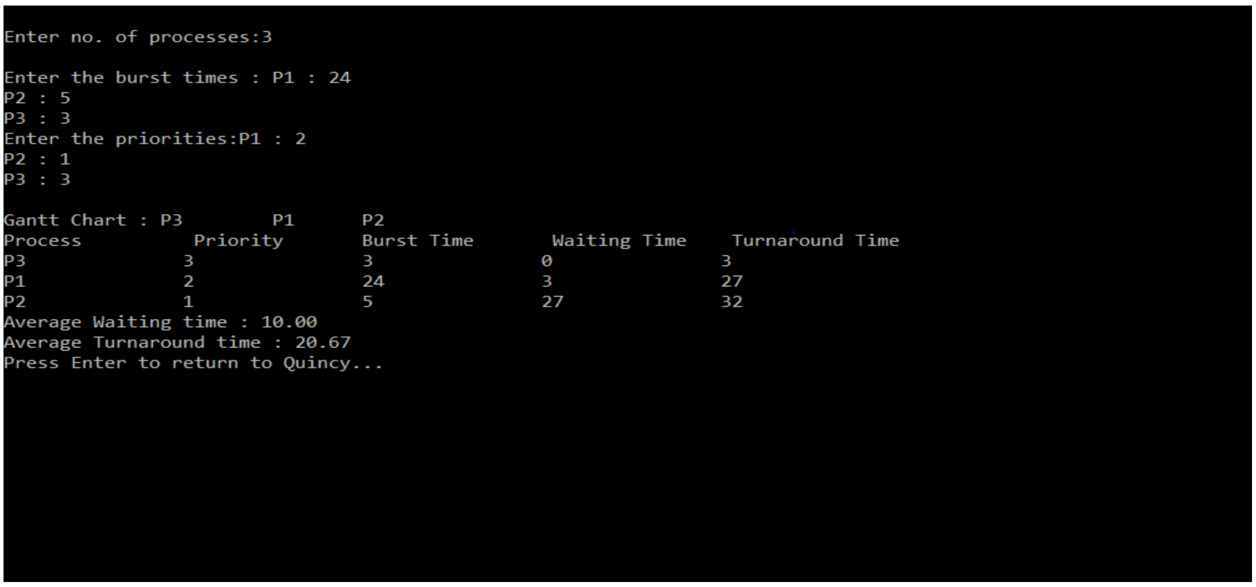
PROGRAM CODING:

```
#include<stdio.h>

int main()

{

int n,temp=0,w[20],b[20], p[20],

t2[20],j,t1,d[20],i,

te=0,b1[20],t3=0;

float t,r;

w[1]=0;

printf("\nEnter no. of processes:");

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

```
printf("\nEnter the burst times : ");
for(i=1;i<=n;i++)
{
printf("P%d : ",i);
scanf("%d",&b[i]); d[i]=i;
}
printf("Enter the priorities:");
for(i=1;i<=n;i++)
{
printf("P%d : ",i);
scanf("%d",&p[i]);     }
for(i=1;i<=n;i++)
for(j=i+1;j<=n;j++)
if(p[i]<p[j])
{
temp=p[i];
t1=d[i];
te=b[i];
p[i]=p[j];
d[i]=d[j];
b[i]=b[j];
p[j]=temp;
d[j]=t1;
b[j]=te;    }
printf("\nGantt Chart : ");
for(i=1;i<=n;i++) printf("P%d\t",d[i]);
printf("\nProcess \t Priority\tBurst Time\t Waiting Time\t Turnaround Time");
for(i=1;i<=n;i++)
{     t=d[i];
w[i+1]=w[i]+b[i];
t2[i]=b[i]+w[i];
t3+=t2[i];
printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d",d[i],p[i],b[i],w[i],t2[i]);      }
```

```
temp=0;
for(i=1;i<=n;i++)
temp+=w[i];
t=(float)temp/n;
r=(float)t3/n;
printf("\nAverage Waiting time : %.2f",t);
printf("\nAverage Turnaround time : %.2f",r);
}
```

**<u>OUTUT</u>**



```
Enter no. of processes:3

Enter the burst times : P1 : 24
P2 : 5
P3 : 3
Enter the priorities:P1 : 2
P2 : 1
P3 : 3

Gantt Chart : P3        P1      P2
Process          Priority       Burst Time       Waiting Time    Turnaround Time
P3               3              3                0               3
P1               2              24               3               27
P2               1              5                27              32
Average Waiting time : 10.00
Average Turnaround time : 20.67
Press Enter to return to Quincy...
```

**Exercise**

1. **Write the above program using other programming languages i.e. C++, JAVA,C# and others.**
2. **What is the important features of this scheduling algorithms?**

# Chapter 3

# Memory Management

## Introduction

- Memory is one of the most important resources of the computer system that is used to store data and programs temporarily.

- Part of the operating system that manages memory is called the Memory Manager (MM).

- The main functions of the Memory Manager (MM) are the following:

- Keeping track of which part of memory are in used and which parts are free allocating
- and deallocating memory to processes
- Managing swapping between memory and disk when memory is not big enough to hold all the processes

## EXPERIMENT 3.1:- MVT (Multiprogramming with a Variable number of Tasks) and MFT (Multiprogramming with a Fixed number of Tasks) memory management

**AIM:** Write a C program to simulate the MVT and MFT memory management techniques

**DESCRIPTION**

MFT (Multiprogramming with a fixed number of Tasks) is one of the old memory management techniques in which 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. MVT (Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning 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.

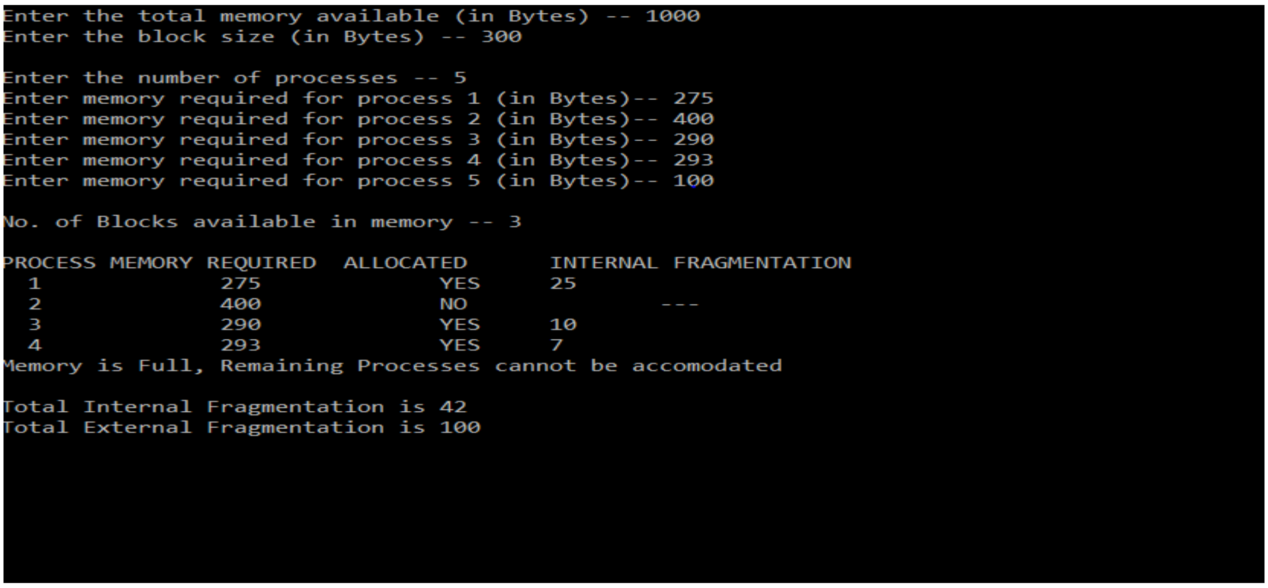
**PROGRAM**

**MFT MEMORY MANAGEMENT TECHNIQUE**

```
#include<stdio.h>

#include<conio.h>

int main()

{

int  ms, bs, nob, ef,n, mp[10],tif=0;

 int i,p=0;

 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\tMEMORY REQUIRED\t ALLOCATED\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();
    }
```

**OUTPUT**



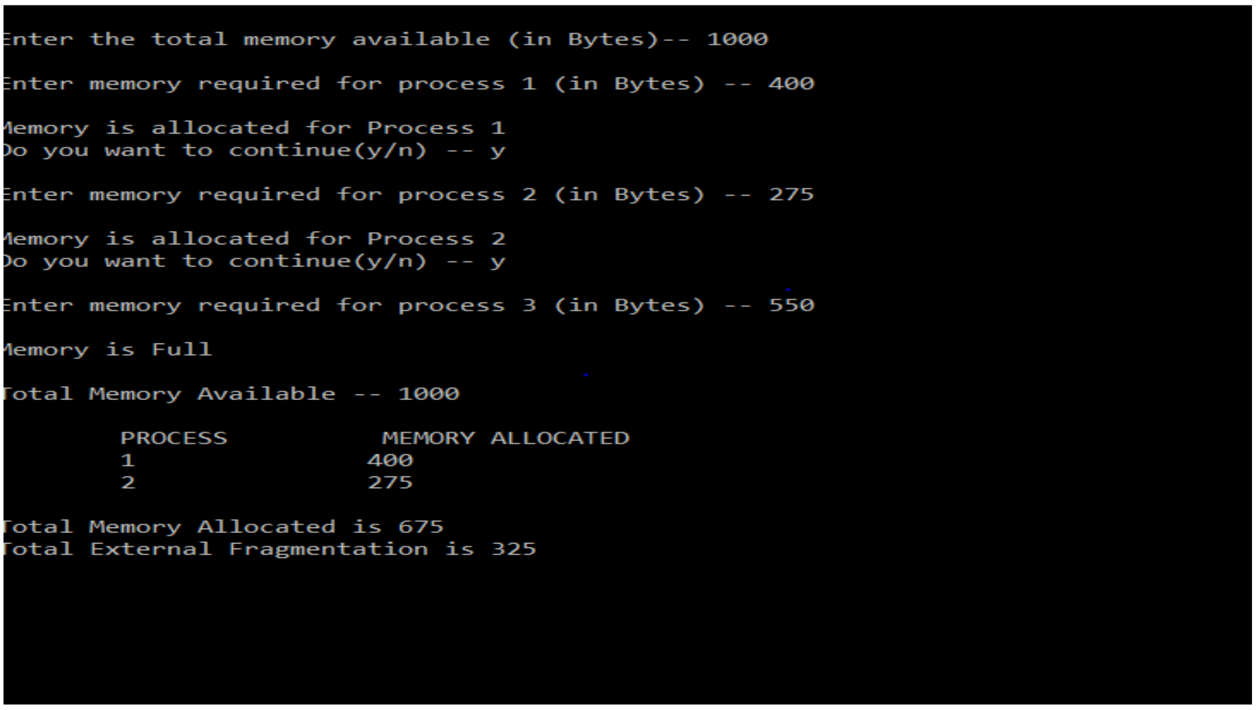
```
Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes) -- 300

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

PROCESS MEMORY REQUIRED  ALLOCATED      INTERNAL  FRAGMENTATION
  1            275            YES      25
  2            400            NO                ---
  3            290            YES      10
  4            293            YES      7
Memory is Full, Remaining Processes cannot be accomodated

Total Internal Fragmentation is 42
Total External Fragmentation is 100
```

**Exercise**

1. **Write the above program using other programming languages i.e. C++,JAVA,C# and others.**
2. **What is the drawbacks of this memory allocation technique?**

**MVT MEMORY MANAGEMENT TECHNIQUE**

```
#include<stdio.h>

#include<conio.h>
int main()
{
 int  ms,mp[10],i, temp,n=0;

 char ch = 'y';
 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();
}
```



```
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
```

**Exercise**

1. **Write the above program using other programming languages i.e. C++,JAVA,C# and others.**
2. **What is the difference from MFT memory management technique**
3. **What is the important features of this technique from fixed allocation?**

## Experiment 3.2:-Contiguous 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**

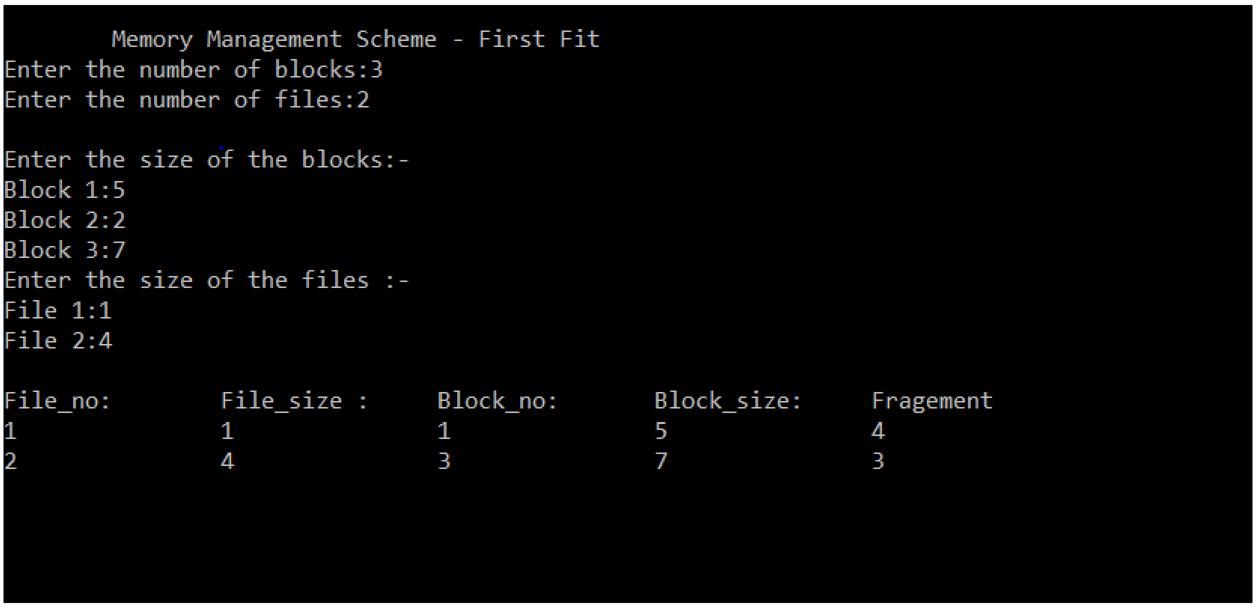
One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When 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**

a) **WORST-FIT**

```
#include<stdio.h>

#include<conio.h>

#define max 25

int main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp;

        static int bf[max],ff[max];

        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();
                                             }
```



```
        Memory Management Scheme - First Fit
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

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               1               5               4
2               4               3               7               3
```

**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

**b) BEST-FIT**

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

#define max 25
int main()

{    int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

         static int bf[max],ff[max];
        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",&nb[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();}
```

```
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

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

**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

c) **FIRST-FIT**

```
#include<stdio.h>
#include<conio.h>
#define max 25
int main()
{  int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit"); printf("\nEnter the
number of blocks:"); scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)  {
printf("Block %d:",i);
scanf("%d",&b[i]);

}

printf("Enter the size of the files :-\n");

for(i=1;i<=nf;i++) {

        printf("File %d:",i);
```

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

        for(i=1;i<=nf;i++)
        {   for(j=1;j<=nb;j++)  {
        if(bf[j]!=1)      //if bf[j] is not allocated
                          {   temp=b[j]-f[i];
                          if(temp>=0)
                          if(highest<temp)  {
                          ff[i]=j;
                          highest=temp;  }}}
                          frag[i]=highest;
                          bf[ff[i]]=1;
                          highest=0;  }
                          printf("\nFile_no:\tFile_size
                          :\tBlock_no:\tBlock_size:\tFragement"); for(i=1;i<=nf;i++)
                          printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
                          getch(); }
```

```
                          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

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

## Exercise

1. **Write the above program using other programming languages i.e. C++, JAVA,C# and others.**
2. **Discuss the importance of Contiguous memory allocation techniques**

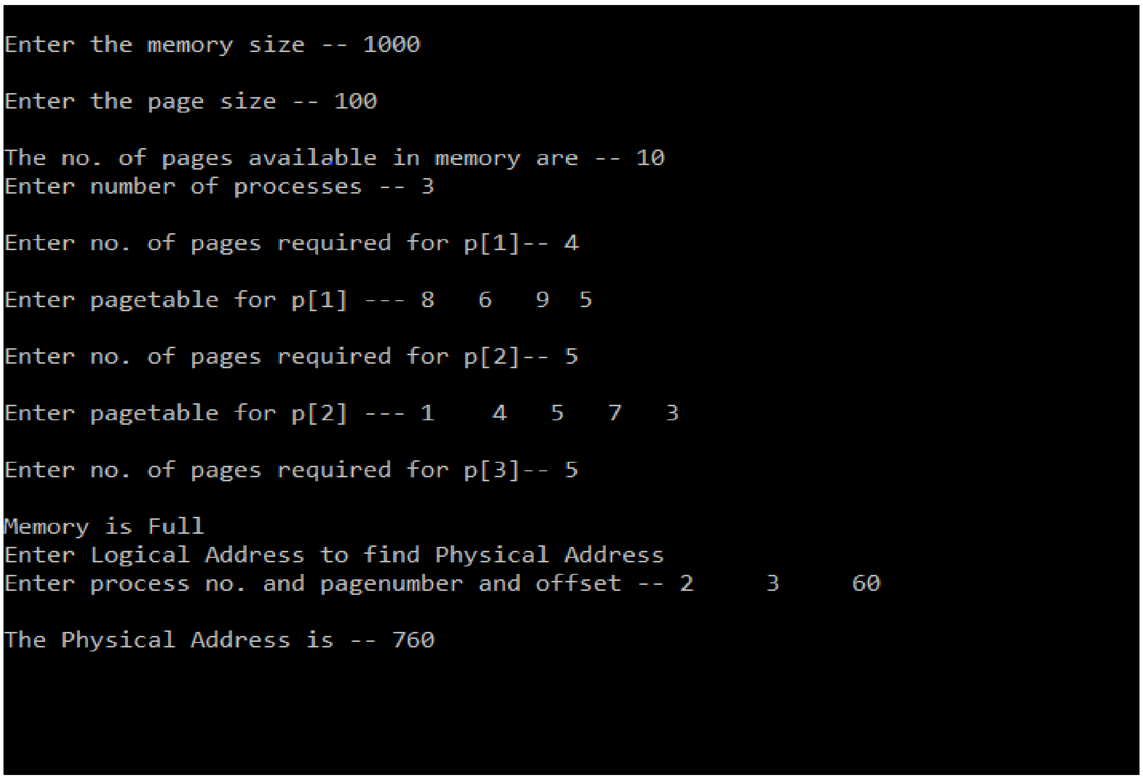
## Experiment 3.3:-Paging technique of memory management

**AIM:** Write a C program to simulate paging technique of memory management.

**DESCRIPTION:** In computer operating systems, paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages. Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous. The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source

```
#include<stdio.h>

#include<conio.h>

int main()

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

int s[10], fno[10][20];

printf("\nEnter the memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);

nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);

printf("\nEnter number of processes -- ");

scanf("%d",&np);

rempages = nop;

for(i=1;i<=np;i++)   {

                printf("\nEnter no. of pages required for p[%d]-- ",i);

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

                if(s[i] >rempages)  {

                printf("\nMemory is Full");  break;}

                rempages = rempages - s[i];

                printf("\nEnter pagetable for p[%d] --- ",i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);

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

**OUTPUT**



```
Enter the memory size -- 1000

Enter the page size -- 100

The no. of pages available in memory are -- 10
Enter number of processes -- 3

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

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

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

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

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

Memory is Full
Enter Logical Address to find Physical Address
Enter process no. and pagenumber and offset -- 2     3     60

The Physical Address is -- 760
```

**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

**Experiment 3.4 page replacement algorithms**

**Aim**

Write a C program to simulate page replacement algorithms

a) FIFO b) LRU c) LFU

**DESCRIPTION**

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A **FIFO** replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. **LRU** replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. **Least frequently used (LFU)** page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.

**PROGRAM**

**a) FIFO PAGE REPLACEMENT ALGORITHM**

```
#include<stdio.h>
#include<conio.h>
int main()
{  int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
              {
```

```
if(m[k]==rs[i])  break;

}
if(k==f)
{ m[count++]=rs[i];
pf++;
  }
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f)
count=0;
 }
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
   }
```

```
Enter the length of reference string – 20
Enter the reference string --    7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter no. of frames-   3
Page Replacement Process is
    7   -1    -1        PF No. 1
    7    0    -1           PFNo2
    7    0     1           PFNo3
    2    0     1           PFNo4
    2    0     1
    2    3     1           PFNo5
    2    3     0           PFNo6
    4    3     0           PFNo7
    4    2     0           PFNo8
    4    2     3           PFNo9
    0    2     3           PF No. 10
    0    2     3
```

```
     0   2    3
     0   1    3            PF No. 11
     0   1    2            PF No. 12
     0   1    2
     0   1    2
     7   1    2            PF No. 13
     7   0    2            PF No. 14
     7   0    1            PF No. 15

The number of Page Faults using FIFO are 15
```

**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

## b) LRU PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h>
#include<conio.h>
int main()
{ int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1; clrscr();
        printf("Enter the length of reference string -- ");
        scanf("%d",&n);
        printf("Enter the reference string -- ");
        for(i=0;i<n;i++)  {
        scanf("%d",&rs[i]);
        flag[i]=0;   }
        printf("Enter the number of frames -- ");
        scanf("%d",&f);
        for(i=0;i<f;i++)
        {  count[i]=0;
        m[i]=-1;
        } printf("\nThe Page Replacement process is -- \n");
        for(i=0;i<n;i++)
        {  for(j=0;j<f;j++)
                {
                if(m[j]==rs[i])
{
flag[i]=1;
count[j]=next;
next++;
}}
if(flag[i]==0)
{  if(i<f)
{    m[i]=rs[i];
count[i]=next;
            next++;

}
else{
min=0;
for(j=1;j<f;j++)
if(count[min] > count[j])
min=j;
m[min]=rs[i];
count[min]=next;
next++;
}  pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" , pf);
printf("\n");}
printf("\nThe number of page faults using LRU are %d",pf);
getch();  }
```

```
Enter the length of reference string – 20

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

Enter the number of frames – 3

The Page Replacement process is --

7    -1    -1    PF No -1
7    0     -1    PF No -2
7    0     1     PF No- 3
2    0     1     PF No- 4
2    0     1
2    0     3     PF No- 5
2    0     3
4    0     3     PF No- 6
4    0     2     PF No- 7
4    3     2     PF No– 8
0    3     2     PF No– 9
0    3     2
0    3     2
1    3     2     PF No-10
1    3     2
1    0     2     PF No-11
1    0     2
1    0     7     PF No-12
1    0     7
1    0     7
The number of page faults using LRU are 12
```

## Exercise

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

## c) LFU PAGE REPLACEMENT ALGORITHM

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

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

```
Enter number of page references – 10
Enter the reference string --              1 2 3 4 5 2 5 2 5 143
Enter the available no. of frames -- 3

    The Page Replacement Process is –


    1          -1          -1                       PF No.1
    1          2           -1                       PF No.2
    1          2           3                        PF No.3
    4          2           3                        PF No.4
    5          2           3                        PF No.5
    5          2           3
    5          2           3
    5          2           1                        PF No.6
    5          2           4                        PF No.7
    5          2           3                        PF No.8
    Total number of page faults –8
```

**Exercise**

4. **Discuss in detail Page replacement algorithms**
5. **Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

# Chapter 4

# File System Management

## Overview of File System

A file is a collection of related information recorded on the secondary storage. For example, file containing student information, file containing employee information, files containing C source code and so on. A file is thus the smallest allotment of logical secondary storage, that is any information to be stored on the secondary storage need to be written on to a file and the file is to be stored. Information in files could be program code or data in numeric, alphanumeric, alphabetic or binary form either formatted or in free form. A file is therefore a collection of records if it is a data file or a collection of bits / bytes / lines if it is code.

You used several files to store information on your computer as the information in them can later be retrieved and used. It is important that the files are managed properly so that they can be located and accessed easily without consuming time.

To help you to organize and manage the files and directories on your computer, an os uses an application called a file manager. A file manager enables you to create, delete, copy, move, rename and view files and create and manage directories (folders). The way in which an os organizes, names, protects, accesses, and uses files is called a file management system.

**Files are required for the following reasons.**

- store data permanently (even after the termination of the process that creates them)
- store large data that is beyond the size of a process's address space
- enable multiple processes to access data simultaneously as in the case of databases

Part of the operating system that deals with files is known as the file system. Users and designers of file systems have different views of files.

- ✓ From the users' point of view, the most important issue is the interface (naming, protection, and operations allowed.)
- ✓ From the designers' point of view, the choice between the use of linked lists versus bit maps and the number of tracks per block has more importance.

**Files and Directories – Interface**

**Files**

**File Naming**

Name is used to identify a file with an abstract way. A file name consists of strings. The maximum allowed length of the string and case sensitivity of names vary from one operating system to the other. Most operating systems have some characters, called file extension, as part of the file name following a period (.). In some operating systems (such as Microsoft Windows), the file extension is used to associate the file with a program while in others, it has no special use. Some characters, e.g. /, \ >, have special meaning in the file system so that they cannot be used as part of the file name.

**File Access**

- Sequential access: the bytes or records of the file are read from beginning to end sequentially
- Random access: the bytes or records of the file can be read out of order

**File Attributes**

A file has a number of attributes. The available attributes of files differ from one operating system to the other. The most common attributes are:

- Attributes for ownership, access permission, and password
- Flags (bits) for archive, hidden, system, read only, temporary, etc.
- File size
- Time of creation, last access, and last modification

**File Operations**

In order to provide the basic store and retrieval operations of a file and other properties of files outlined above, operating systems have some common file operations. Some of them are: Create, delete, open, close, read, write, append, seek, get attributes, set attributes, and rename.

**Directories**

To keep track of files, file systems normally have directories (folders). In most operating systems directories themselves are files.

**Single Level Directory Systems**

There is only one directory (the root directory) in the system. The advantage of this scheme is its implementation simplicity and ease of locating files. The main disadvantage is the need to provide distinct name for each file. This kind of directory organization was used in early operating systems.

**Two Level Directory Systems**

In a multi user system, it is more convenient to have a directory for each user under the root directory and a system directory to store system level files. If only a file name is provided, it refers to a file in the users' own directory, but if a file in another user's directory is needed, the user's directory name has to be supplied along with the file name.

**Hierarchical Directory Systems**

To give users the ability to organize their files arbitrarily, it is important to have a file system with a general tree structure. The user is allowed to create directory structures of arbitrary levels deep.

**Directory Operations**

The type of operations on directories varies from one operating system to the other. The most common operations are: create, delete, opendir, closedir, and rename.

**Files and Directories – Implementation**

**Implementing Files:-t**he most important issue of file implementation is to keep track of which disk blocks go to which file. There are several alternative ways of allocating disk space for files.

**Contiguous Allocation**

A file is stored in consecutive disk blocks.

Advantage:

- Simple to implement: what we keep track of is the first block of a file and the number of blocks used by the file.
- High performance: when accessing a file, the only delay is while searching the first block of the file.

Disadvantage:

- The file size must be specified at the time of file creation.
- The disk would eventually have many holes (unused blocks) as files are deleted. Thus the disk may be full while there are holes at many locations which are small to contain a file of specified length.

**Linked List Allocation**

Files are implemented as a linked list of disk blocks. The first word of a block is used to point to the next block and the rest of the block is used to store data.

Advantage:

- Disk usage is effective; there is no fear of losing disk blocks to holes.
- It is enough to store only the address of the first block of a file in the directory.

Disadvantage:

- Random access of a file is slow since file access always begins at the first block.
- The amount of data stored in a block is not a power of two as some space in each block is allocated for the pointer to the next block. Since many programs read/write data with size of a power of two, it may be necessary to read two consecutive blocks and concatenate the data.

## EXPERIMENT 4:- File allocation strategies

**Aim**

Write a C program to simulate the following file allocation strategies.

a) Sequential b) Linked c) ) Indexed

**DESCRIPTION**

A file is a collection of data, usually stored on disk. As a logical entity, a file enables to divide data into meaningful groups. As a physical entity, a file should be considered in terms of its organization. The term "file organization" refers to the way in which data is stored in a file and, consequently, the method(s) by which it can be accessed

### 4.1.1 SEQUENTIAL FILE ALLOCATION

In this file organization, the records of the file are stored one after another both physically and logically. That is, record with sequence number 16 is located just after the 15th record. A record of a sequential file can only be accessed by reading all the previous records.

### 4.1.2 LINKED FILE ALLOCATION

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.

### 4.1.3 INDEXED FILE ALLOCATION

Indexed file allocation strategy brings all the pointers together into one location: an index block. Each file has its own index block, which is an array of disk-block addresses. The $i^{th}$ entry in the index block points to the $i^{th}$ block of the file. The directory contains the address of the index block. To find and read the $i^{th}$ block, the pointer in the $i^{th}$ index-block entry is used.

**PROGRAM**

### 4.1.1 SEQUENTIAL FILE ALLOCATION

```
#include<stdio.h>
#include<conio.h>
struct fileTable
{
char name[20];
int sb, nob;
}ft[30];
int main()
{
int i, j, n;
char s[20];
printf("Enter no of files   :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d  :",i+1);
scanf("%s",ft[i].name);
printf("Enter starting block of file %d");
scanf("%d",&ft[i].sb);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob); }
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;



if(i==n)
printf("\nFile Not Found");
else{
printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");
printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob); for(j=0;j<ft[i].nob;j++)
printf("%d, ",ft[i].sb+j);
} getch();
}
```

```
Enter no of files: 3
Enter file name 1        : A
Enter starting block of file 1:85

Enter no of blocks in file 1     :6
Enter file name 2        : B
Enter starting block of file 2:102

Enter no of blocks in file 2     :4
Enter file name 3        : C

Enter starting block of file 3:60

Enter no of blocks in file 3     :4

Enter the file name to be searched -- B

FILE           START          NO OF
NAME           BLOCK          BLOCKS

B                 102              4
```

**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

### 4.1.2 LINKED FILE ALLOCATION

```
        #include<stdio.h>
        #include<conio.h>
        struct fileTable
        {
        char name[20];
        int nob;
        struct block *sb;
        }
        struct block
        {  int bno;
        struct block *next;
        };
        int  main()   {
           int i, j, n;
      char s[20];
struct block *temp;
printf("Enter no of files:");
scanf("%d",&n);
for(i=0;i<n;i++)  {

printf("\nEnter filename%d:"i+1);
scanf("%s",ft[i].name);

printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
ft[i].sb=(struct block*)malloc(sizeof(struct block));
temp = ft[i].sb;
printf("Enter the blocks of the file:");
scanf("%d",&temp->bno);
temp->next=NULL;
for(j=1;j<ft[i].nob;j++)
{    temp->next = (struct block*)malloc(sizeof(struct block));
temp = temp->next;
scanf("%d",&temp->bno);
}temp->next = NULL; }
printf("\nEnter the file name to be searched --  ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)
printf("\nFile Not Found");
else  {
printf("\nFILE NAME NO OF BLOCKS  BLOCKS OCCUPIED");
printf("\n%s\t\t%d\t",ft[i].name,ft[i].nob);

temp=ft[i].sb;
for(j=0;j<ft[i].nob;j++)
{
printf("%d □ ",temp->bno);
temp = temp->next;}}
getch();
  }
```

```
Enter no of files   : 2
Enter file 1        : A
Enter no of blocks in file 1      : 4
Enter the blocks of the file 1          : 12 23 9 4
Enter file 2        : G
Enter no of blocks in file 2      : 5
Enter the blocks of the file 2          : 88 77 66 55 44
Enter the file to be searched  : G
                     NO OF
FILE NAME            BLOCKS                BLOCKS OCCUPIED

G                         5            88 □ 77□ 66□ 55□ 44
```

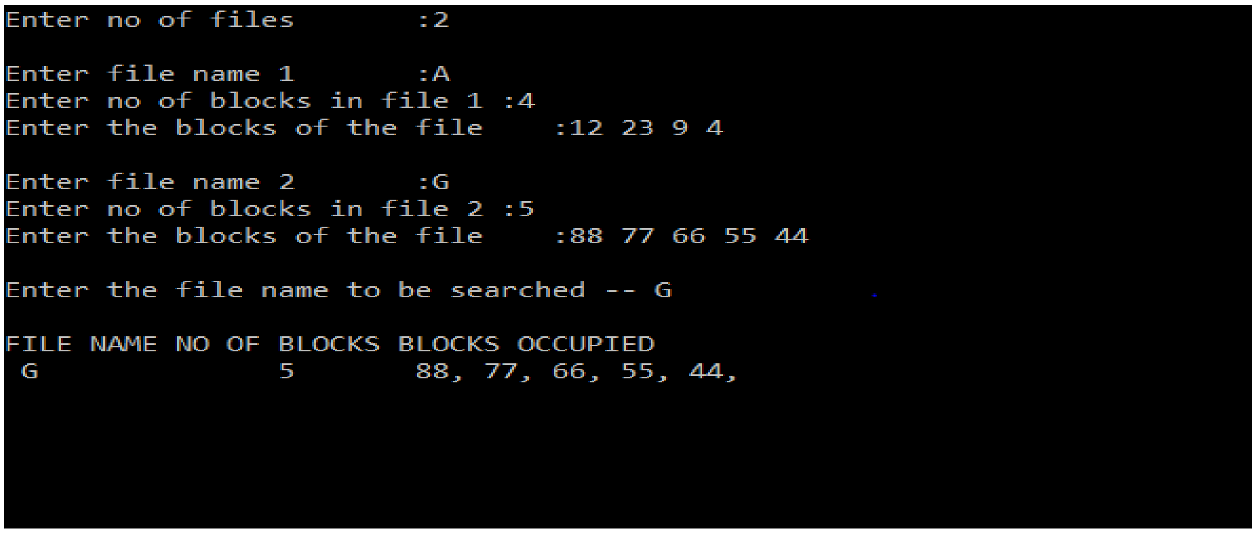
**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

### 4.1.3 INDEXED FILE ALLOCATION

```
#include<stdio.h>
#include<conio.h>
struct fileTable
{
char name[20];
int nob, blocks[30];
}ft[30];
int main()
{
int i, j, n;
char s[20];
printf("Enter no of files          :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d  :",i+1);
scanf("%s",ft[i].name);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
printf("Enter the blocks of the file      :");
for(j=0;j<ft[i].nob;j++)
scanf("%d",&ft[i].blocks[j]);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)

printf("\nFile Not Found");

else

{

printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED"); printf("\n
%s\t\t%d\t",ft[i].name,ft[i].nob); for(j=0;j<ft[i].nob;j++)

printf("%d, ",ft[i].blocks[j]);

}

getch();

}
```

**<u>OUTUT</u>**



```
Enter no of files       :2

Enter file name 1       :A
Enter no of blocks in file 1 :4
Enter the blocks of the file    :12 23 9 4

Enter file name 2       :G
Enter no of blocks in file 2 :5
Enter the blocks of the file    :88 77 66 55 44

Enter the file name to be searched -- G

FILE NAME NO OF BLOCKS BLOCKS OCCUPIED
 G               5       88, 77, 66, 55, 44,
```

**Exercise**

1. Write the above program using other programming languages i.e. C++,JAVA,C# and others.
2. Discuss the difference between File allocation strategies

## Experiment 4.2:-FILE ORGANIZATION TECHNIQUES

**Aim**

Write a C program to simulate the following file organization techniques

a) Single level directory b) Two level directory c) Hierarchical

**DESCRIPTION**

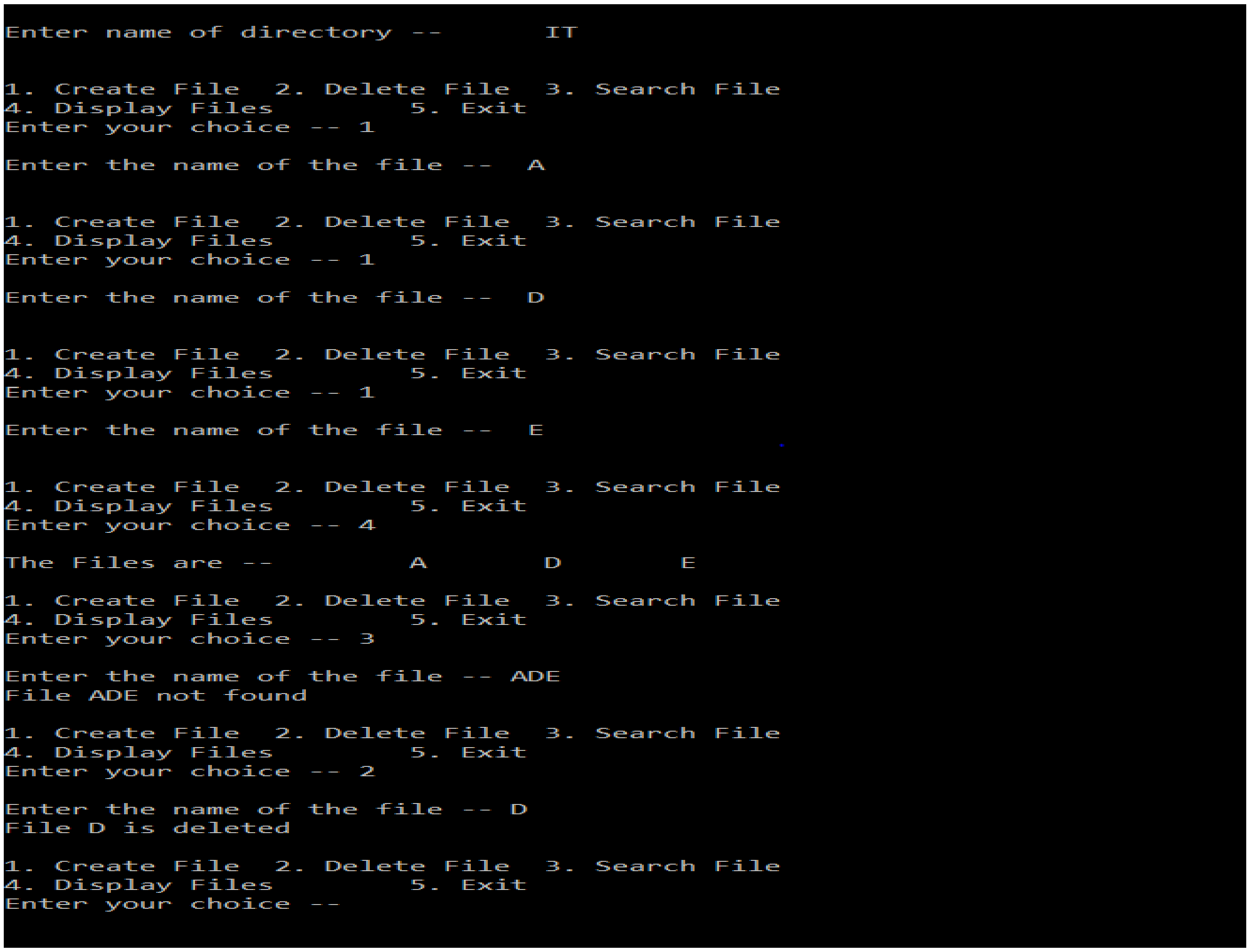
The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

**Experiment 4.2.1 Single level directory**

```
#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

int main()

{

int i,ch;

char f[30];

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 \n4. 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();
}
```



```
Enter name of directory --       IT

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 --   D


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

Enter the name of the file --   E


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

The Files are --          A        D        E

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

Enter the name of the file -- ADE
File ADE 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 -- D
File D is deleted

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

## Exercise

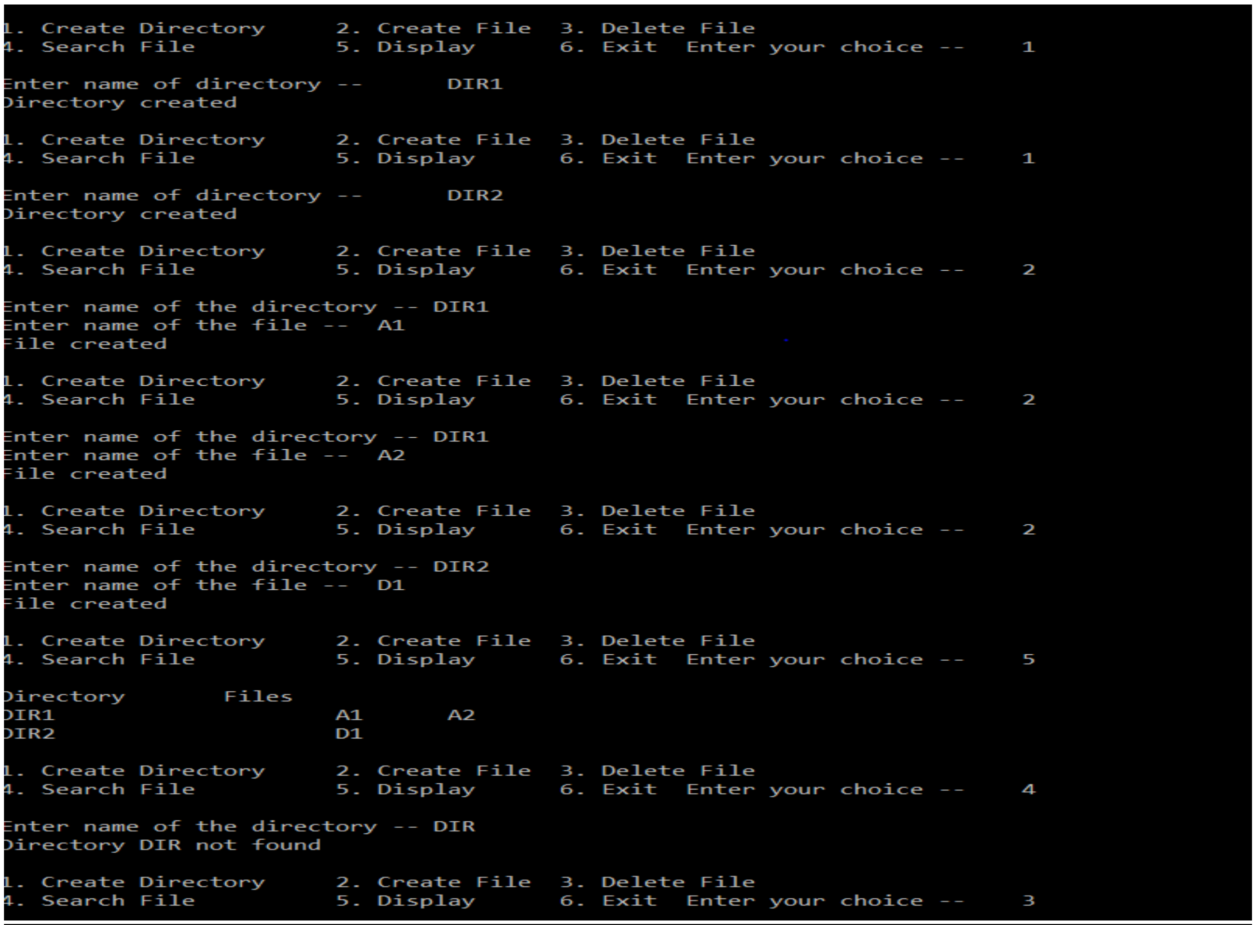
**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

**Experiment 4.2.2 Two level directory**

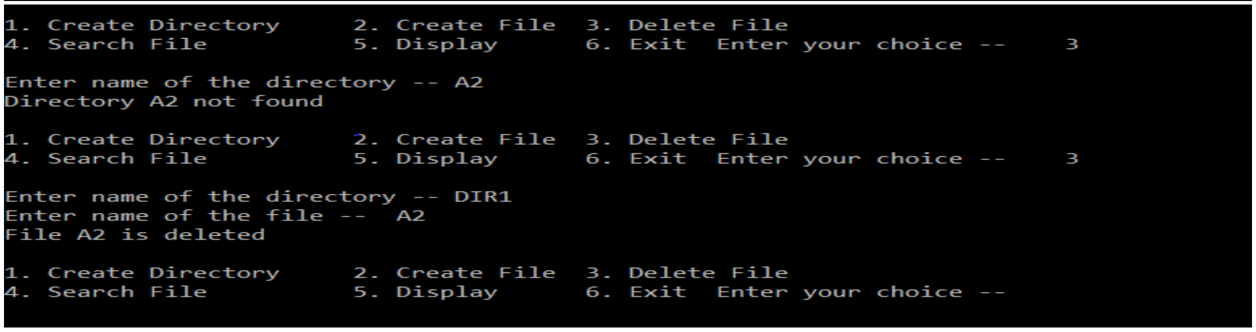
```
#include<stdio.h>
struct
{ char dname[10],fname[10][10];
int fcnt;
} dir[10];
int main()
{
int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File"); printf("\n4. Search
File\t\t5. Display\t6. Exit\t Enter your choice --          ");
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");
break;
}  if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
```

```
printf("Enter name of the file --  ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}}
printf("File %s not found",f);
goto jmp;
}}
printf("Directory %s not found",d);
jmp  : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file --  ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}}
printf("File %s not found",f);
goto jmp1;
}}
printf("Directory %s not found",d);
jmp1:  break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
```

```
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}}
break;
default:exit(0);
}}
getch();
}
```



```
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 --    2

Enter name of the directory -- DIR2
Enter name of the file --  D1
File created

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

Directory       Files
DIR1                    A1      A2
DIR2                    D1

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

Enter name of the directory -- DIR
Directory DIR not found

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



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

Enter name of the directory -- A2
Directory A2 not found

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

Enter name of the directory -- DIR1
Enter name of the file --  A2
File A2 is deleted

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

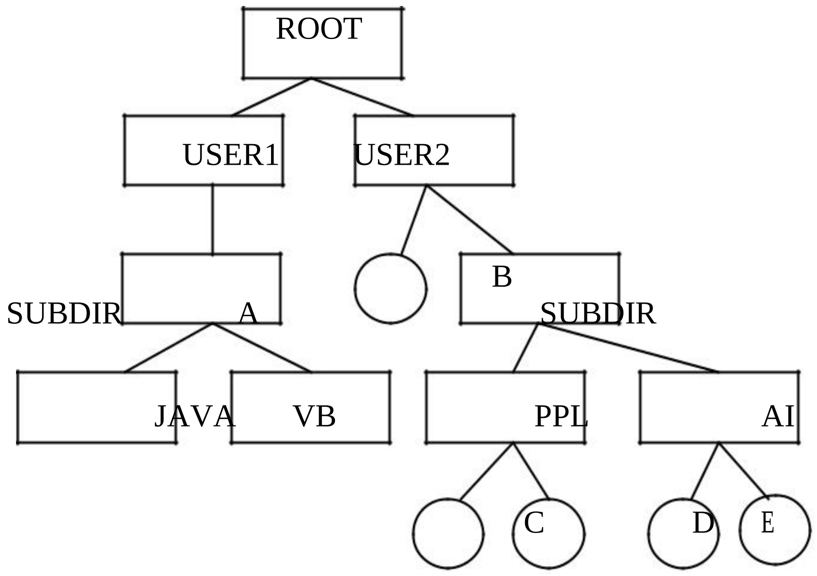
**Exercise**

**Write the above program using other programming languages i.e. C++,JAVA,C# and others.**

**Experiment 4.2.2 Hierarchical directory**

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

getch();

close graph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x) {
int i, gap;

if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) : ",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 for file :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1){
printf("No of sub directories/files(for %s):",(*root)->name);
scanf("%d",&(*root)>nc);
if((*root)->nc==0)
gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)>link[i]),lev+1,(*root)>name,lx+gap*i,lx+gap*i+gap,
lx+gap*i+gap/2);
}
else
(*root)->nc=0;
}}
display(node *root){
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root !=NULL)
{
for(i=0;i<root->nc;i++)
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
if(root->ftype==1)
bar3d(root->x-20,root->y-10,root->x+20,root>y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name);
for(i=0;i<root->nc;i++)
display(root->link[i]);
}}
```

```
Enter Name of dir/file (under root): ROOT
Enter 1 for Dir/2 for File:       1
No of subdirectories/files (for ROOT): 2
Enter Name of dir/file (under ROOT): USER1
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for USER1): 1
Enter Name of dir/file (under USER1): SUBDIR1
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for SUBDIR1): 2
Enter Name of dir/file (under USER1): JAVA
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for JAVA): 0
Enter Name of dir/file (under SUBDIR1): VB
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for VB): 0
Enter Name of dir/file (under ROOT): USER2
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for USER2): 2
Enter Name of dir/file (under ROOT): A
Enter 1 for Dir/2 for File: 2
Enter Name of dir/file (under USER2): SUBDIR2
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for SUBDIR2): 2
Enter Name of dir/file (under SUBDIR2): PPL
Enter 1 for Dir/2 for File: 1
No of subdirectories/files (for PPL): 2
Enter Name of dir/file (under PPL): B
Enter 1 for Dir/2 for File: 2
Enter Name of dir/file (under PPL): C
Enter 1 for Dir/2 for File: 2

Enter Name of dir/file(under SUBDIR): AI

Enter 1 for Dir/2 for File: 1

No of subdirectories/files(for AI): 2

Enter Name of dir/file(under AI): D

Enter 1 for Dir/2 for File: 2

Enter Name of dir/file(under AI): E

Enter 1 for Dir/2 for File: 2
```



## Exercise

1. Write the above program using other programming languages i.e. C++,JAVA,C# and others.
2. Discuss the difference between File organization strategies

# Chapter five

# Deadlocks

## Introduction

A deadlock is a condition where two or more users are waiting for data, locked by each other. Oracle automatically detects a deadlock and resolves them.

- Deadlock occurs when transactions executing at the same time lock each other out of data that they need to complete their logical units of work.
- Deadlock is a situation where a group of processes are all blocked and none of them can become unblocked until one of the other becomes unblocked.

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

Consider a case where two different processes want to be allocated on the same resource (say printer) at a particular time. If both the processes requests for the same resource then the system will come under the state of deadlock because a single resource can attend only one process at a time. In other words, a printer can print only one process (document) at a time.

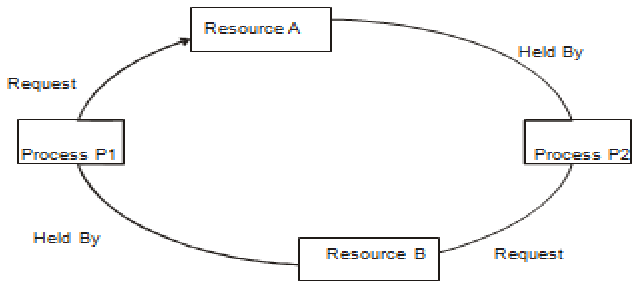


**Figure 1 Deadlock**

## Experiment 5:-Implementation of deadlock detection algorithm

**Aim:-**To write a C program to implement Deadlock Detection algorithm
ALGORITHM:
Step 1: Start the Program
Step 2: Obtain the required data through char and in data types.
Step 3: Enter the filename, index block.
Step 4: Print the file name index loop.
Step 5: File is allocated to the unused index blocks
Step 6: This is allocated to the unused linked allocation.
Step 7: Stop the execution

```
//Deadlock Detection algorithm implementation
#include <stdio.h>
#include <conio.h>
void main()
{
int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
clrscr();
printf("Enter total no of processes");
scanf("%d",&tp);
printf("Enter total no of resources");
scanf("%d",&tr);
printf("Enter claim (Max. Need) matrix\n");
for(i=1;i<=tp;i++)
{
printf("process %d:\n",i);
for(j=1;j<=tr;j++)
scanf("%d",&c[i][j]);
}
printf("Enter allocation matrix\n");
for(i=1;i<=tp;i++)
{
printf("process %d:\n",i);
for(j=1;j<=tr;j++)
scanf("%d",&p[i][j]);
}
printf("Enter resource vector (Total resources):\n");
for(i=1;i<=tr;i++)
{ scanf("%d",&r[i]);
}
printf("Enter availability vector (available resources):\n");
for(i=1;i<=tr;i++)
{
scanf("%d",&a[i]);
temp[i]=a[i];
}
for(i=1;i<=tp;i++)
{
sum=0;
for(j=1;j<=tr;j++)
{
sum+=p[i][j];
}
if(sum==0)
{
m[k]=i;
k++;
}}
for(i=1;i<=tp;i++)
{ for(l=1;l<k;l++)
if(i!=m[l])
{
flag=1;
for(j=1;j<=tr;j++)
if(c[i][j]<temp[j])
{
flag=0;
break;
}}
if(flag==1)
{
m[k]=i;
k++;
for(j=1;j<=tr;j++)
temp[j]+=p[i][j];
}}
printf("deadlock causing processes are:");
for(j=1;j<=tp;j++)
{
found=0;
for(i=1;i<k;i++)
{
if(j==m[i])
found=1;
}
if(found==0)
printf("%d\t",j);
}
getch();
}
```

**OUTPUT:**

```
Enter total no. of processes: 4
Enter total no. of resources: 5
Enter claim (Max. Need) matrix:
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter allocation matrix:
1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Enter resource vector (Total resources):
2 1 1 2 1
Enter availability vector (available resources):
0 0 0 0 1
Deadlock causing processes are: 2 3
```

## Exercise

1. **Write the above program using other programming languages i.e. C++,JAVA,C# and others.**
2. **What is the disadvantages of deadlock?**
3. **How and when deadlock occur?**

## References

1. https://studentsfocus.com/wp-content/uploads/anna_univ/CSE/4SEM/CS6413%20-%20OS%20Lab/3.Operating-Systems-Lab-1_2013_regulation.pdf ACCESSED ON [NOVEMBER 25,2020]
2. https://61faba30-a-62cb3a1a-s-sites.googlegroups.com/site/anandacademics/downloads/CS8461-OSRecord.pdf?attachauth=ANoY7cqWJ-U4Ae5vZP55XXrMTr_dBZ3VJBrML_PtcMSYGVJu0AS7Vy3sXSD-kcJUzrrR3kbPIGKph14JSmem2dVDkslA9KRWSNTT9bHPxeHahQDmDQ2txLW0ahuCekHyGQ-vaGyD7trVgPsS6IiOB3e-JtLnq7CWGP475VZ3j7U2opgybbDTh7nLgEIcfS_rsDD4hh_HjT69Bh-yUJ-fj412fb3IT0ByDg6eD4cBo32qZekB4UUT1go%3D&attredirects=0 ACCESSED ON[NOVEMBER 20,2020]
3. ANNA UNIVERSITY CHENNAI REGULATION -2013 CS 6413 – OPERATING SYSTEMS LABORATORY
4. https://management.ind.in/forum/operating-system-lab-manual-anna-university-235041.html ACCESSED ON[NOVEMBER 15,2020]
5. https://drive.google.com/file/d/16H5mLBNg9y5UVp-Zy0ZkEZ8HH8eT8VQW/view ACCESSED ON[NOVEMBER 25,2020]
6. https://www.guru99.com/operating-system-tutorial.html ACCESSED ON[NOVEMBER 25,2020]
7. DESIGN OF OPERATING SYSTEMS LABORATORY MANUAL BY MANAL ELSIR ELHADI MUSTAFA B.Sc. Hon. (Computer Engineering, 1999) A Thesis Submitted For Partial Fulfillment of Degree of Master Of Computer Architecture and Networking Department of Electrical and Electronics Engineering Faculty of Engineering, University of Khartoum Sudan (July 2003)
8. https://www.studocu.com/row/document/comsats-university-islamabad/operating-systems/tutorial-work/os-lab-manual/6110517/view ACCESSED ON[NOVEMBER 10,2020]
9. https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/ ACCESSED ON[DECEMBER 22,2020]
10. https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/7_Deadlocks.html ACCESSED ON[DECEMBER 22,2020]
11. https://www.studytonight.com/operating-system/deadlocks ACCESSED ON[DECEMBER 22,2020]
12. https://www.guru99.com/deadlock-in-operating-system.html ACCESSED ON[DECEMBER 22,2020]
13. https://peda.net/kenya/css/subjects/computer-studies/form-three/driac2/data-processing/fom ACCESSED ON[DECEMBER 22,2020]
14. https://www.guru99.com/os-memory-management.html ACCESSED ON[DECEMBER 22,2020]
15. https://www.includehelp.com/operating-systems/contiguous-and-non-contiguous-memory-allocation.aspx ACCESSED ON[DECEMBER 22,2020]
16. http://www2.latech.edu/~box/os/ch08.pdf ACCESSED ON[DECEMBER 22,2020]
17. https://www.geeksforgeeks.org/difference-between-contiguous-and-noncontiguous-memory-allocation/ ACCESSED ON[DECEMBER 22,2020]
18. https://www.gatevidyalay.com/tag/memory-management-in-os/ ACCESSED ON[DECEMBER 22,2020]
19. https://www.studytonight.com/operating-system/cpu-scheduling ACCESSED ON[DECEMBER 22,2020]
20. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/ ACCESSED ON[DECEMBER 22,2020]