KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayanguda, Hyderabad – 500029

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB RECORD

` **OPERATING SYSTEMS**

B.Tech. II YEAR II SEM (RKR21)

ACADEMIC YEAR 2023-24

Kmit

An Autonomous Institution

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S. and Affliated to JNTUH)

NARAYANAGUDA, HYDERABAD - 500 029.

Certificate

This is to certify that the following is a Bonafide Record of the practical work done by ______________________________ bearing Roll No. ____________________ of __________ Branch of __________ year B.Tech Course in the ____________________ Laboratory during the Academic year ______ & ______ under our supervision.

Number of experiments conducted : ____________________

Signature of Staff Member Incharge

Signature of Head of the Dept.

Signature of Internal Examiner

Signature of External Examiner

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad
Department of Computer Science and Engineering

Daily Laboratory Assessment Sheet

Name of the Lab: Name of the Student:

Class: HT.No:

S.No.	Name of the Experiment	Date	Observation Marks (3M)	Record Marks (4M)	Viva Voice Marks (3M)	Total Marks (10M)	Signature of Faculty
	TOTAL						

Faculty Incharge

INDEX

Exp No:	List of Experiments	Page No
1	Identify the peripherals of a computer, components in a CPU and its functions. Draw the block diagram of the CPU along with the configuration of each peripheral. a) Disassemble and assemble the PC back to working condition. b) Install MS windows on the personal computer. c) Install Linux on the computer. This computer should have windows installed. The system should beconfigured as dual boot with both windows and Linux.	
2	a) Implement in c language the following Unix commands using system calls i) cat ii) ls iii) mv b) Write a C program to create child process and allow parent process to display “parent” and the child todisplay “child” on the screen	
3	Assume you have the following jobs to execute with one processor, with the jobs arriving in the order listed here: **i** **T(pi)** **0** **80** **1** **20** **2** **10** **3** **20** **4** **50** with the following values write a program to get the required output which is listed below a. Suppose a system uses FCFS scheduling. Create a Gantt chart illustrating the execution of these processes? b. What is the average turnaround time for the processes? c. What is the average wait time for the processes?	
4	a. Write a C program that illustrate communication between two unrelated process using named pipes b. Write a C program that receives a message from message queue and display them c. Write a C program to allow cooperating process to lock a resource for exclusive use (using semaphore) d. Write a C program that illustrate the suspending and resuming process using signal e. Write a C program that implements producer-Consumer system with two process using semaphore	

5

Consider the following snapshot of a system. P0, P1, P2, P3, P4 are the processes and A, B, C, D are the resourcetypes. The values in the table indicates the number of instances of a specific resource (for example: 3 3 2 1 underthe last column indicates that there are 3 A-type, 3 B-type, 2 C-type and 1 D-type resources are available after allocating the resources to all five processes). The numbers under allocation-column indicate that those number of resources are allocated to various processes mentioned in the first column. The numbers under Max- column indicate the maximum number of resources required by the processes. For example: in 1st row under allocation-column 2 0 0 1 indicate there are 2 A-type, 0 B-type, 0 C-type and 1 D- type resources are allocated to process P0. Whereas 4 2 1 2 under Max-column indicate that process P0‘s maximum requirement is 4 A- type, 2 B-type,1 C-type
and 2 D-type resources

Process	Allocation A B C D	Max A B C D	Available A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 12	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

.

Answer the following questions using banker’s algorithm by providing all intermediate steps

a. How many instances of resources are present in the system under each type of a resource?
b. Compute the Need matrix for the given snapshot of a system.
c. Verify whether the snapshot of the present system is in a safe state by demonstrating an order in which theprocesses may complete. If a request from process P1 arrives for (1,1,0,0), can the request be granted immediately?
d. If a request from process P4 arrives for (0,0,2,0), can the request be granted immediately?

6

Write a C program to simulate the following memory management technique: Paging

7

a. Write a C program that takes one or more file/directory names as command line input and reports following information

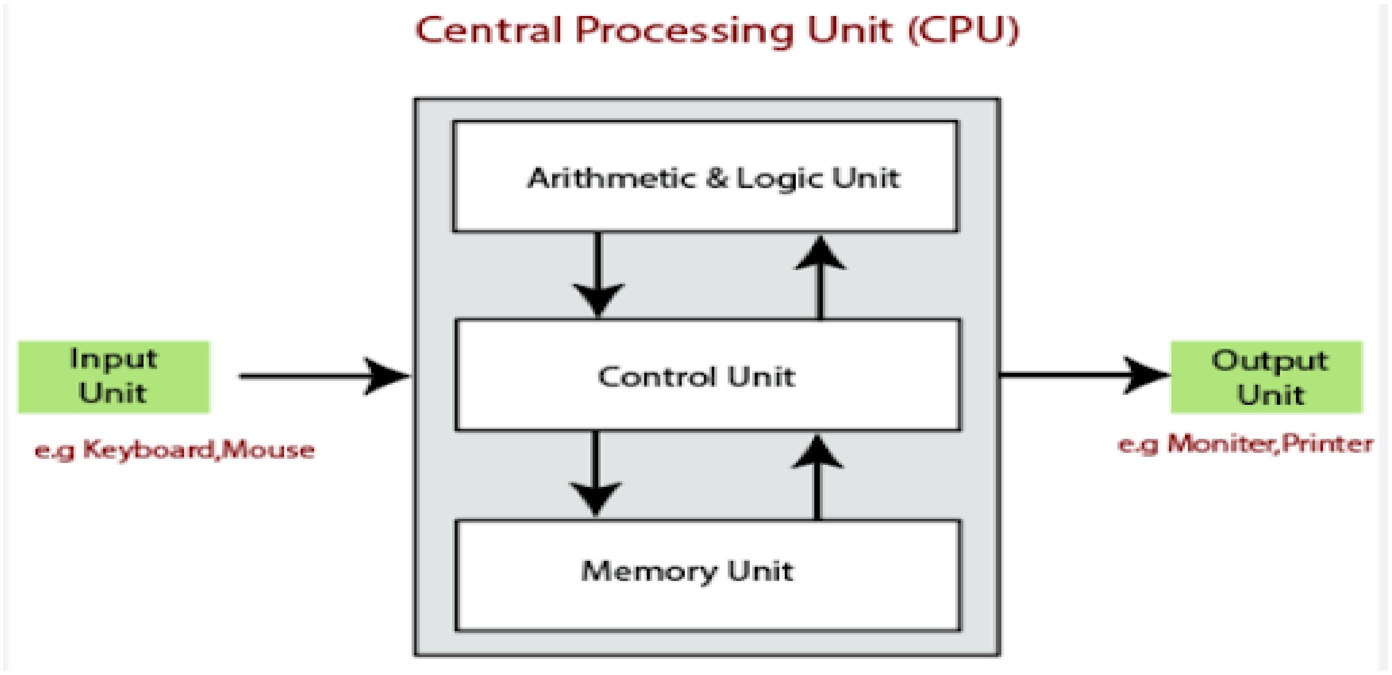
i) File Type ii) Number of Links iii) Time of last Access iv) Read, write and execute permissions.

b. Write a C program to list every file in directory, its inode number and file name

EXPERIMENT 1

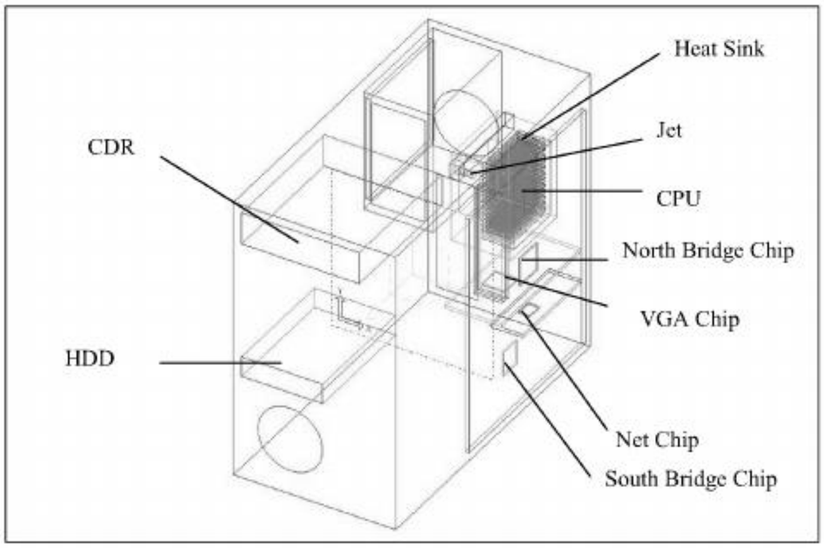
Q) Identify the peripherals of a computer, components in a CPU and its functions. Draw the block diagram of the CPU along with the configuration of each peripheral.

AIM: To Identify the peripherals of a computer, components in a CPU and its functions. Draw the block diagram of the CPU along with the configuration of each peripheral.



Peripherals of a computer:

1. Cabinet:



A computer cabinet, also known as a computer case, serves as a protective enclosure for essential computer components including the motherboard, CPU, RAM, and storage devices. In addition to safeguarding these components, it also facilitates efficient ventilation and cooling through strategically placed fans and vents. This ensures not only optimal performance but also extends the lifespan of the hardware by preventing overheating and maintaining stable operating temperatures.

2. Monitor:

A monitor is an essential electronic visual display used with computers, providing users with a graphical interface to interact with digital content. It serves as a primary output device, capable of displaying a wide range of graphical information, text, and images in various resolutions and color depths. Monitors come in various types, such as LCD, LED, and OLED, each offering unique features like high-definition clarity, wide viewing angles, and energy efficiency.

3. System board/Motherboard

The motherboard functions as the primary circuit board in a computer, playing a pivotal role as the central hub that interconnects and enables communication among all essential components such as the CPU, RAM, storage devices, and peripherals. Beyond connectivity, it houses critical components like the chipset, which manages data flow between these parts, and the BIOS/UEFI firmware, essential for initializing hardware during startup. Additionally, the motherboard features a variety of connectors and expansion slots that accommodate peripheral devices and allow for future hardware upgrades, ensuring versatility and longevity in computer configurations.

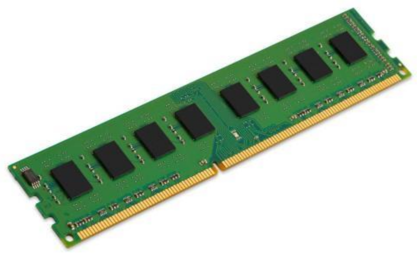
4. SMPS:

SMPS (Switched-Mode Power Supply) converts AC from the mains to DC for computers. It's efficient, compact, and includes components like transformers and rectifiers. SMPS regulates voltage and current and incorporates safety features.

5. CPU

The central processing unit (CPU) houses the core of every computer, known as the processor. This essential component is securely installed onto the motherboard, which serves as the main circuit board supporting the entire PC system. The CPU, often referred to as the brain of the computer, is responsible for executing instructions and performing calculations that drive the computer's functionality.

6. RAM

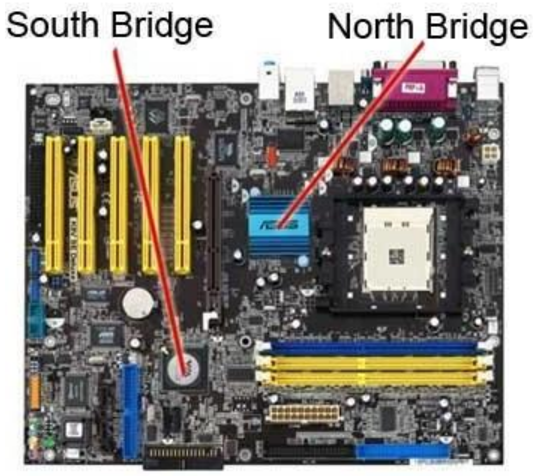


RAM, or Random Access Memory, is an essential component of a computer's architecture, playing a critical role in its performance and responsiveness. Unlike storage devices such as hard drives or SSDs, RAM is volatile memory, meaning it retains data only temporarily while the computer is powered on. This temporary storage capability allows the CPU to quickly access and manipulate data needed for active tasks, such as running applications, loading files, and processing multimedia content

7. CMOS Battery:

The CMOS battery is a small, round, silver battery located on the motherboard of a computer. It plays a crucial role in maintaining certain system settings and configurations, even when the computer is turned off. The primary purpose of the CMOS battery is to provide power to the CMOS (Complementary Metal-Oxide Semiconductor) memory chip.

8. North Bridge and South Bridge:



The Northbridge and Southbridge are chipset components historically found on computer motherboards. The Northbridge manages high-speed communication between the CPU, RAM, and PCIe slots, crucial for efficient data transfers and graphics performance. In contrast, the Southbridge handles lower-speed I/O interfaces such as USB, SATA, Ethernet, and legacy PCI connections, facilitating peripheral device connectivity. Modern advancements have integrated many functions of these bridges into CPUs or other chipset components, streamlining motherboard designs and improving overall system efficiency and performance.

9. BIOS Chip:

The BIOS chip, or firmware chip, on a motherboard stores essential instructions for initializing hardware and starting the computer. It holds configuration settings like boot sequence and system time, accessible through a setup utility during startup. This chip ensures proper interaction between hardware components and the operating system, crucial for system stability and compatibility. As technology advances, BIOS chips are evolving into UEFI (Unified Extensible Firmware Interface), offering enhanced features such as graphical interfaces and secure boot capabilities.

10. Keyboard:

A keyboard is a primary input device for computers, allowing users to input text and commands. It typically features alphanumeric keys, function keys, and special keys like Ctrl, Alt, and Esc. Modern keyboards often include multimedia controls and ergonomic designs for comfort. Keyboards connect to computers via USB or wireless technologies like Bluetooth. They are essential for navigating and interacting with software efficiently.

11. Mouse:

A mouse is a pointing device used with computers for navigation and interaction. It detects movement on surfaces via optical or laser sensors. Typically, it has left and right buttons and a scroll wheel for additional functionality. Modern mice may include extra buttons for customizable actions. Mice connect to computers through USB or wireless connections, enhancing productivity and ease of use.

12. Printer:

A printer is a peripheral device that produces hard copies of digital documents or images onto paper or other media. It uses inkjet, laser, or other technologies to transfer ink or toner onto the printing medium. Printers vary in size and capability, from compact home printers to large-scale office machines. They connect to computers via USB, Ethernet, or wireless networks for printing tasks. Printers are essential for both personal and business use, providing tangible copies of digital content.

13. Scanner :

A scanner is a device that captures images, documents, or objects and converts them into digital format for storage or manipulation on a computer. It uses sensors to capture optical information from the source material and then translates it into a digital image file.

1a) Disassemble and assemble the PC back to working condition.

AIM: Disassemble and assemble the PC back to working condition.

STEPS TO DISASSEMBLE THE PC:

1. **Prepare Motherboard:**
   - Install the CPU into the motherboard socket, aligning the notches and securing it with the retention arm.
2. **Install CPU Cooler:**
   - Apply thermal paste if necessary, then securely mount the CPU cooler onto the motherboard, ensuring it makes good contact with the CPU.
3. **Install RAM:**
   - Insert the RAM sticks into their slots on the motherboard, applying even pressure until they click into place.
4. **Install Storage Drives:**
   - Mount storage drives (HDDs, SSDs) into their drive bays and secure them with screws. Connect SATA or NVMe cables to the motherboard.
5. **Install Expansion Cards:**
   - Insert expansion cards (such as graphics cards) into their respective slots on the motherboard and secure them with screws or retention clips.
6. **Connect Power Supply:**
   - Connect power supply cables to the motherboard, ensuring all connections are secure and properly seated.
7. **Connect Peripherals:**
   - Reattach USB headers, audio connectors, or any other peripheral cables to the motherboard.
8. **Close the Case:**
   - Replace the side panel of the PC case and secure it with screws or latches.
9. **Final Checks:**
   - Double-check all connections and ensure components are properly seated.

STEPS TO DISASSEMBLE THE PC:

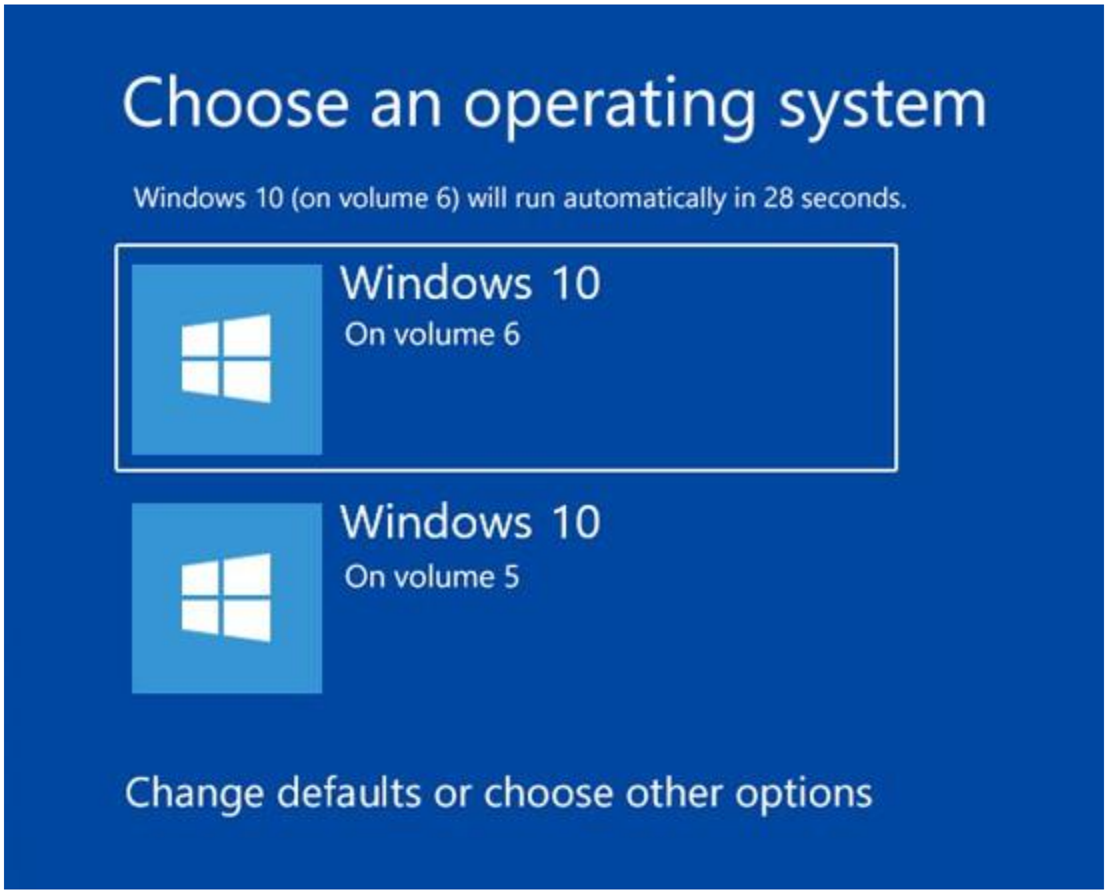
1. **Power Off and Prepare:**
   - Shut down the PC completely and disconnect all cables.
   - Ground yourself to discharge any static electricity.
2. **Open the Case:**
   - Remove the screws or release latches securing the side panel of the PC case.
3. **Disconnect Power Supply:**
   - Unplug all power supply cables from the motherboard, drives (HDD/SSD), and peripherals.
4. **Remove Expansion Cards:**
   - Release retention clips and gently remove any expansion cards (such as graphics cards) from their slots.
5. **Remove CPU Cooler:**
   - Unscrew the CPU cooler from the motherboard, disconnect any fan headers or clips, and carefully lift it away.
6. **Remove RAM:**
   - Press the retention clips on the RAM slots to release the RAM sticks, then carefully remove them from their slots.
7. **Disconnect Drives:**
   - Unscrew and disconnect SATA or power cables from storage drives (HDDs, SSDs) and optical drives.
8. **Disconnect Peripherals:**
   - Unplug USB headers, audio connectors, or any other peripheral cables connected to the motherboard.

1b) Install MS windows on the personal computer.

AIM: To Install MS Windows on Personal Computer

STEPS TO INSTALL MS WINDOWS:

1. Prepare Your Installation Media



Choose an operating system

Windows 10 (on volume 6) will run automatically in 28 seconds.

Ensure you have a bootable USB drive or DVD with the Windows installation files. You can create a bootable USB drive using the Windows Media Creation Tool available from the Microsoft website.

2. Back Up Important Data

Before proceeding, back up any important data from your current system to an external drive or cloud storage. Installing Windows will typically erase all data on the target drive.

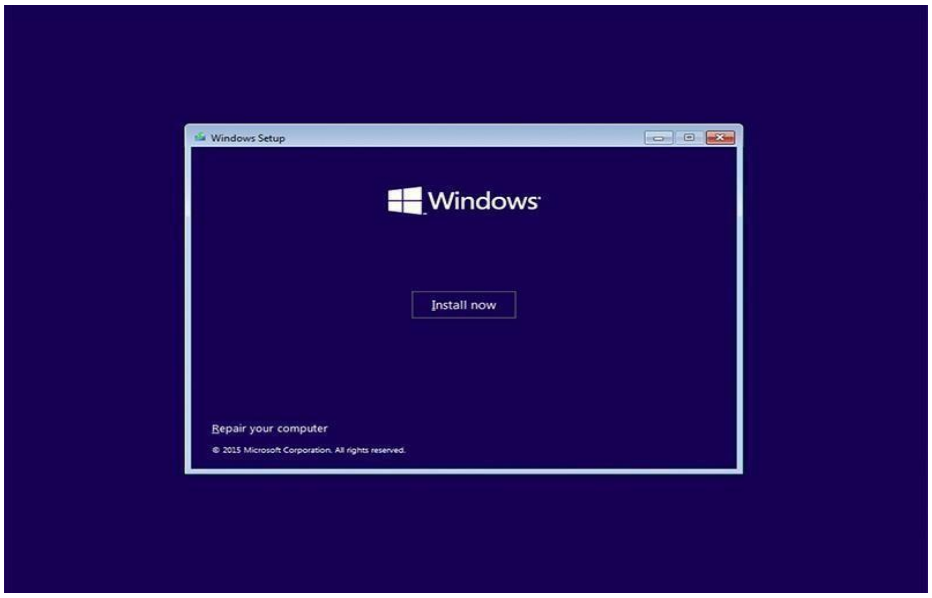
3. Insert Installation Media

Insert the bootable USB drive or DVD into your computer. Ensure it is properly connected and recognized by the system.

4. Boot from Installation Media

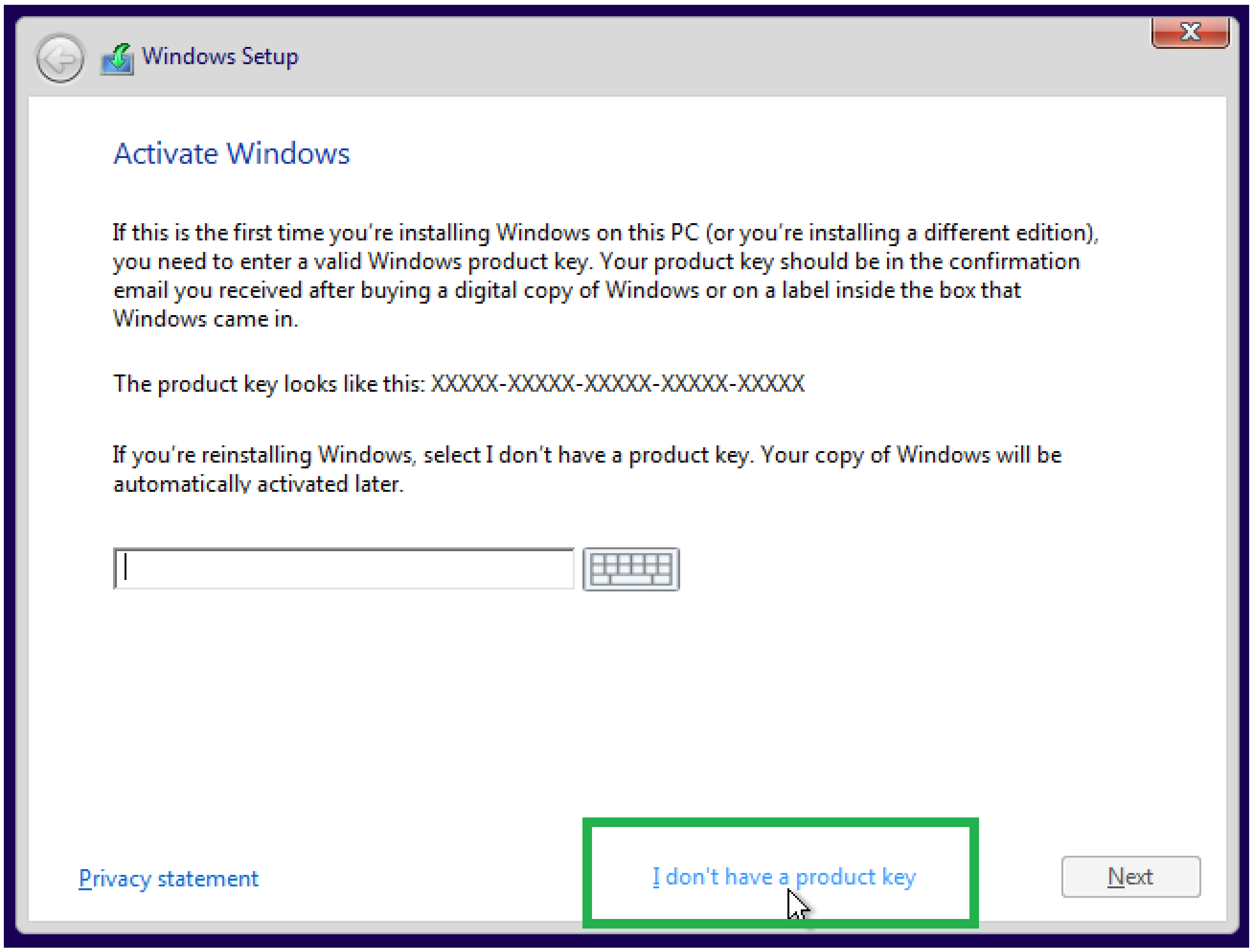
Restart your computer and enter the BIOS/UEFI setup by pressing the appropriate key during startup (usually F2, F12, Delete, or Esc). Change the boot order to prioritize the USB drive or DVD containing the Windows installation files. Save your changes and exit the BIOS/UEFI setup.

5. Start Windows Installation



The computer will boot from the installation media, and the Windows Setup screen will appear. Select your language, time and currency format, and keyboard or input method, then click "Next." Click "Install now" to begin the installation process.

6. Enter Product Key



Activate Windows

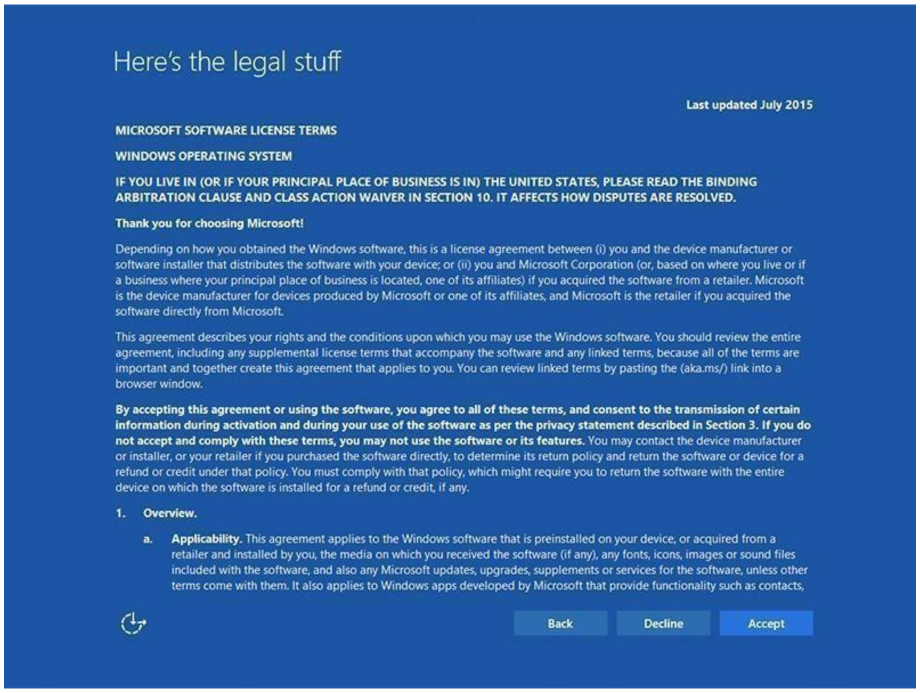
If this is the first time you're installing Windows on this PC (or you're installing a different edition), you need to enter a valid Windows product key. Your product key should be in the confirmation email you received after buying a digital copy of Windows or on a label inside the box that Windows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

If prompted, enter your Windows product key. You can skip this step if you don't have the key at the moment and enter it later after installation.

7. Accept License Terms



Here's the legal stuff

Last updated July 2015

MICROSOFT SOFTWARE LICENSE TERMS

WINDOWS OPERATING SYSTEM

IF YOU LIVE IN (OR IF YOUR PRINCIPAL PLACE OF BUSINESS IS IN) THE UNITED STATES, PLEASE READ THE BINDING ARBITRATION CLAUSE AND CLASS ACTION WAIVER IN SECTION 10. IT AFFECTS HOW DISPUTES ARE RESOLVED.

Thank you for choosing Microsoft!

Depending on how you obtained the Windows software, this is a license agreement between (i) you and the device manufacturer or software installer that distributes the software with your device; or (ii) you and Microsoft Corporation (or, based on where you live or if a business where your principal place of business is located, one of its affiliates) if you acquired the software from a retailer. Microsoft is the device manufacturer for devices produced by Microsoft or one of its affiliates, and Microsoft is the retailer if you acquired the software directly from Microsoft.

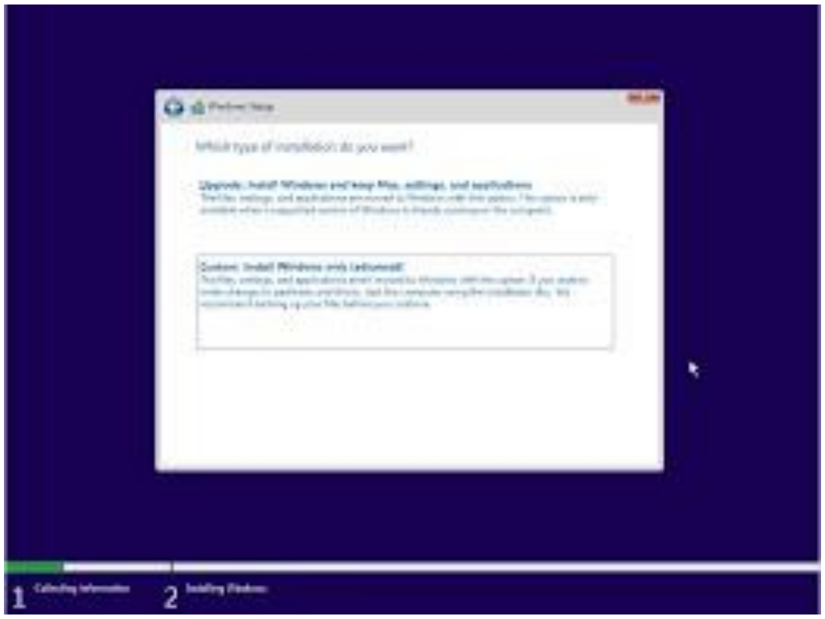
This agreement describes your rights and the conditions upon which you may use the Windows software. You should review the entire agreement, including any supplemental license terms that accompany the software and any linked terms, because all of the terms are important and together create this agreement that applies to you. You can review linked terms by pasting the (aka.ms/) link into a browser window.

By accepting this agreement or using the software, you agree to all of these terms, and consent to the transmission of certain information during activation and during your use of the software as per the privacy statement described in Section 3. If you do not accept and comply with these terms, you may not use the software or its features. You may contact the device manufacturer or installer, or your retailer if you purchased the software directly, to determine its return policy and return the software or device for a refund or credit under that policy. You must comply with that policy, which might require you to return the software with the entire device on which the software is installed for a refund or credit, if any.

1. **Overview.**

   a. **Applicability.** This agreement applies to the Windows software that is preinstalled on your device, or acquired from a retailer and installed by you, the media on which you received the software (if any), any fonts, icons, images or sound files included with the software, and also any Microsoft updates, upgrades, supplements or services for the software, unless other terms come with them. It also applies to Windows apps developed by Microsoft that provide functionality such as contacts, Read and accept the Microsoft Software License Terms by checking the box and clicking "Next."

8. Choose Installation Type



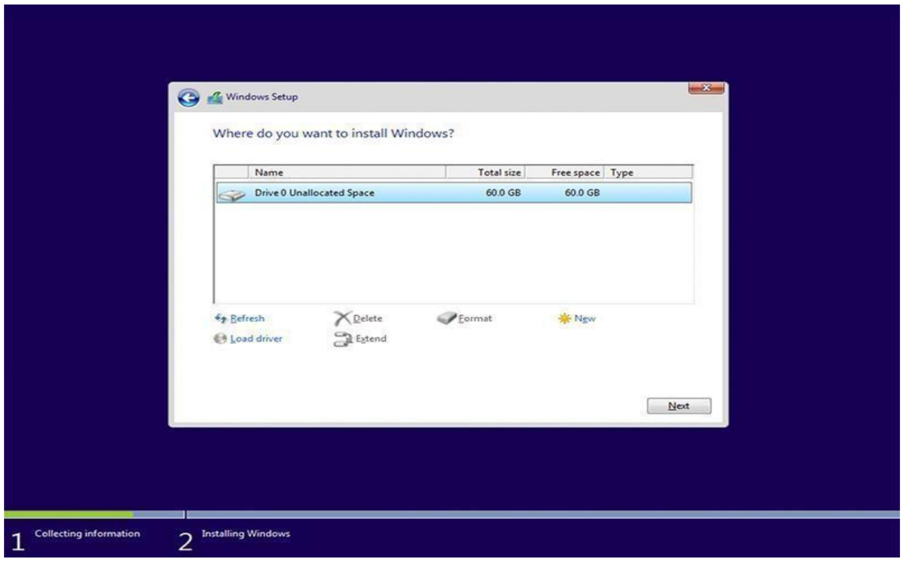
Which type of installation do you want?

Upgrade: Install Windows and keep files, settings, and applications
[illegible]

Custom: Install Windows only (advanced)
[illegible]

Select "Custom: Install Windows only (advanced)" to perform a clean installation.

9. Select Partition

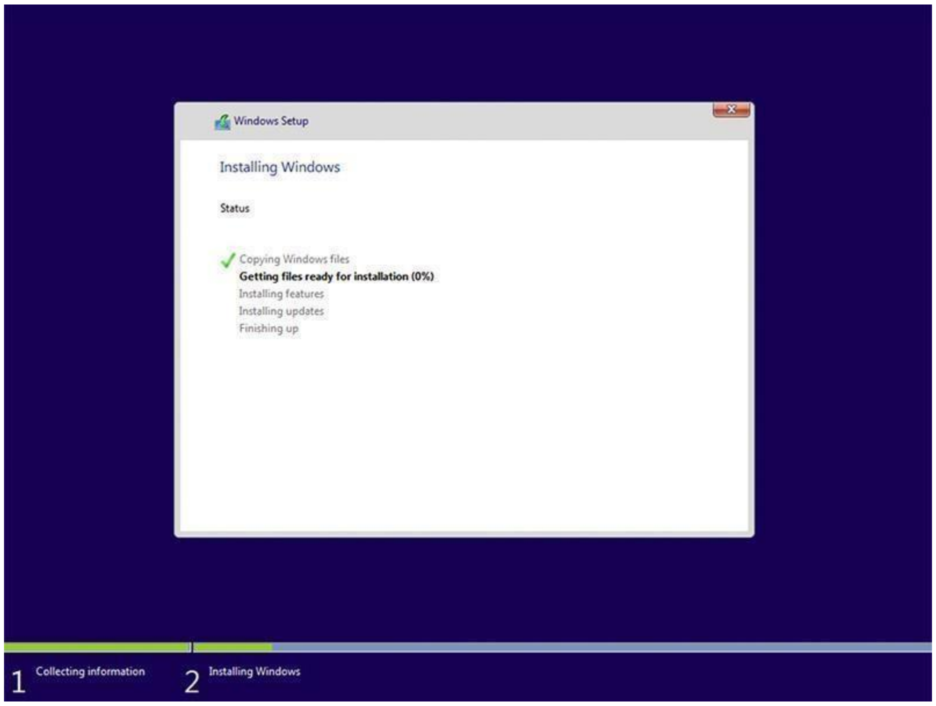


Where do you want to install Windows?

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

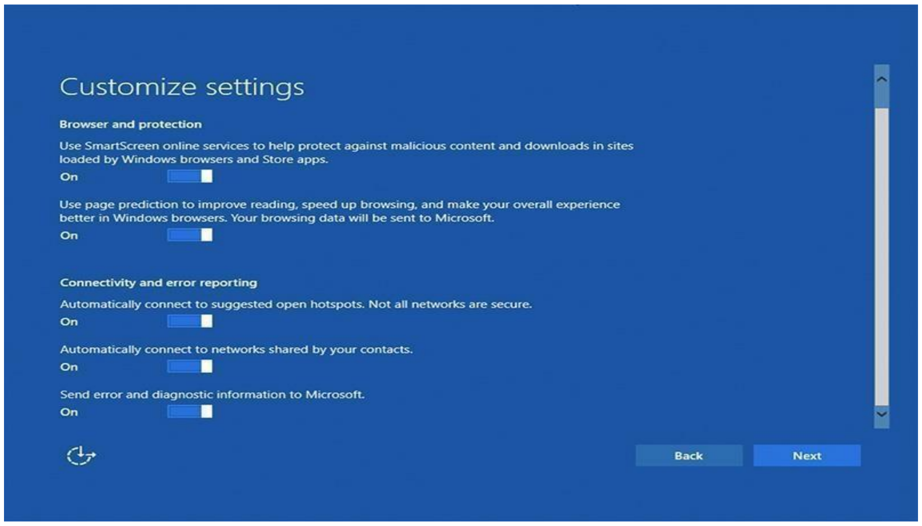
Choose the partition where you want to install Windows. If you want to create new partitions, delete existing ones, or format a partition, you can do so from this screen. Be careful, as this will erase all data on the selected partitions. Once you've selected the appropriate partition, click "Next."

10. Install Windows



Windows will begin copying files, expanding files, installing features, installing updates, and completing the installation process. This may take some time, and your computer will restart several times during this process.

11. Set Up Windows



Customize settings

Browser and protection

Use SmartScreen online services to help protect against malicious content and downloads in sites loaded by Windows browsers and Store apps.
On

Use page prediction to improve reading, speed up browsing, and make your overall experience better in Windows browsers. Your browsing data will be sent to Microsoft.
On

Connectivity and error reporting

Automatically connect to suggested open hotspots. Not all networks are secure.
On

Automatically connect to networks shared by your contacts.
On

Send error and diagnostic information to Microsoft.
On

After the final restart, Windows will guide you through the initial setup

12. Complete Setup

Once you complete the initial setup, Windows will finalize your settings and take you to the desktop.

13. Install Drivers and Updates

After installation, it is crucial to install the necessary drivers for your hardware.

1c) Install Linux on the computer. This computer should have windows installed. The system should be configured as dual boot with both windows and Linux.

AIM: To install Linux Operating System on the computer with Dual boot option along with Windows Operating System.

STEPS TO INSTALL LINUX OPERATING SYSTEM WITH DUAL BOOT

1. Prepare Your Installation Media

Ensure you have a bootable USB drive or DVD with the Linux distribution you want to install. You can create a bootable USB drive using tools like Rufus, Etcher, or the Linux distribution's own recommended tool.

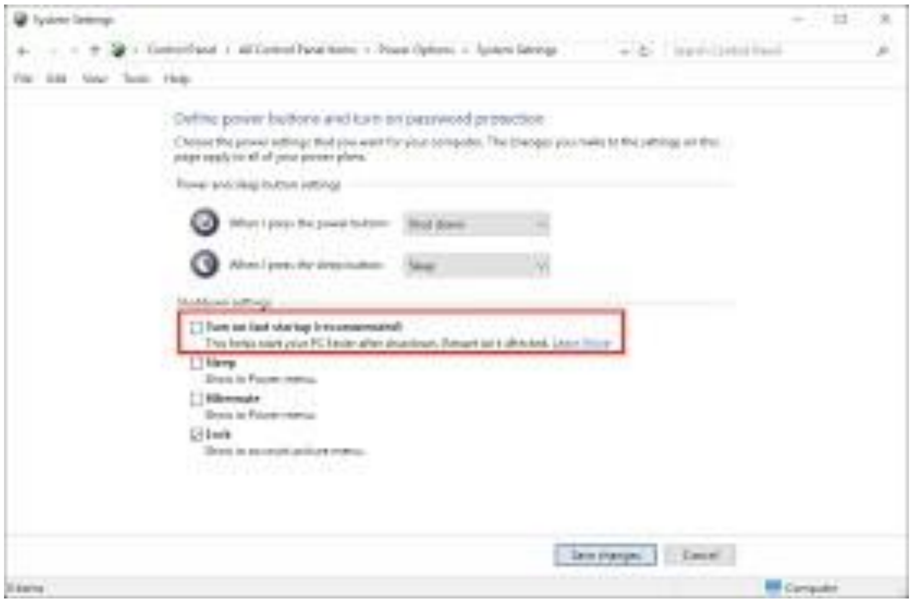
2. Back Up Important Data

Back up important data from your Windows system to an external drive or cloud storage to prevent data loss during the installation process.

3. Create Space for Linux Installation

i). Open Disk Management in Windows : Press `Win + X` and select "Disk Management."
ii). Shrink the Windows Partition : Right-click on the Windows partition (usually `C:`) and select "Shrink Volume." Enter the amount of space to shrink, which will be used for the Linux installation. Click "Shrink."

4. Disable Fast Startup and Secure Boot


System Settings

Control Panel > All Control Panel Items > Power Options > System Settings

Define power buttons and turn on password protection

Choose the power settings that you want for your computer. The changes you make to the settings on this page apply to all of your power plans.

Power and sleep button settings

When I press the power button: Shut down

When I press the sleep button: Sleep

Shutdown settings

- [ ] Turn on fast startup (recommended)
  This helps start your PC faster after shutdown. Restart isn't affected. Learn More
- [ ] Sleep
  Show in Power menu.
- [ ] Hibernate
  Show in Power menu.
- [x] Lock
  Show in account picture menu.

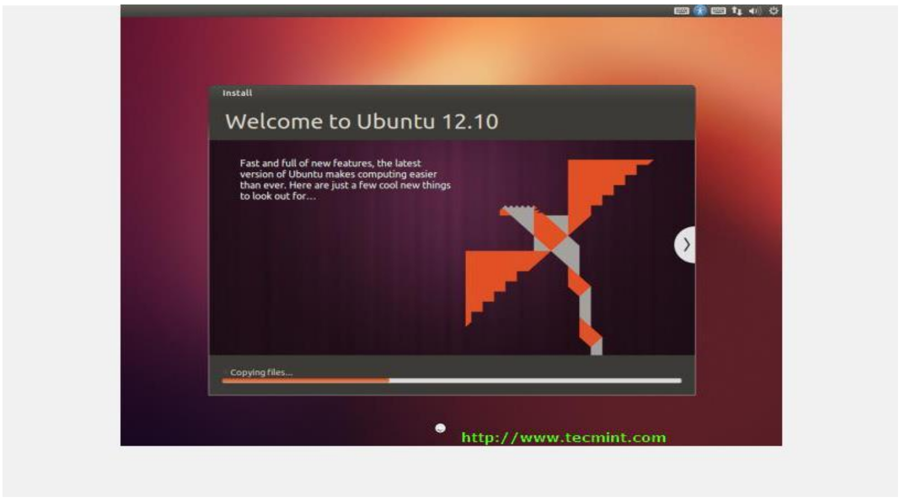
i). Disable Fast Startup: In Windows, go to Control Panel > Hardware and Sound > Power Options > Choose what the power buttons do > Change settings that are currently unavailable. Uncheck "Turn on fast startup" and save changes.

ii). Disable Secure Boot: Restart your computer and enter the BIOS/UEFI setup (usually by pressing `F2`, `F12`, `Delete`, or `Esc` during startup). Find the Secure Boot option and disable it. Save and exit BIOS/UEFI.

5. Boot from Linux Installation Media

i). Insert the bootable USB drive or DVD with the Linux installation files.
ii). Restart your computer and enter BIOS/UEFI setup again. Change boot order to prioritize USB drive or DVD
iii). Save changes and exit BIOS/UEFI.

6. Start Linux Installation


Install

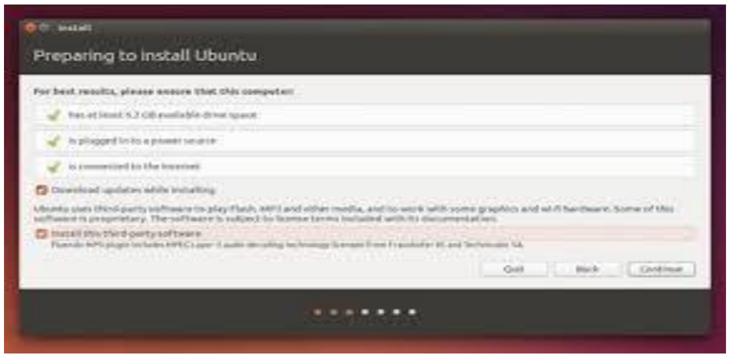
Welcome to Ubuntu 12.10

Fast and full of new features, the latest version of Ubuntu makes computing easier than ever. Here are just a few cool new things to look out for…

Copying files…

i). Boot from the installation media and start the Linux installation process.
ii). Select your language, time zone, and keyboard layout, then click "Next."

7. Choose Installation Type



Install

Preparing to install Ubuntu

For best results, please ensure that this computer:

✓ has at least 5.2 GB available drive space

✓ is plugged in to a power source

✓ is connected to the Internet

Download updates while installing

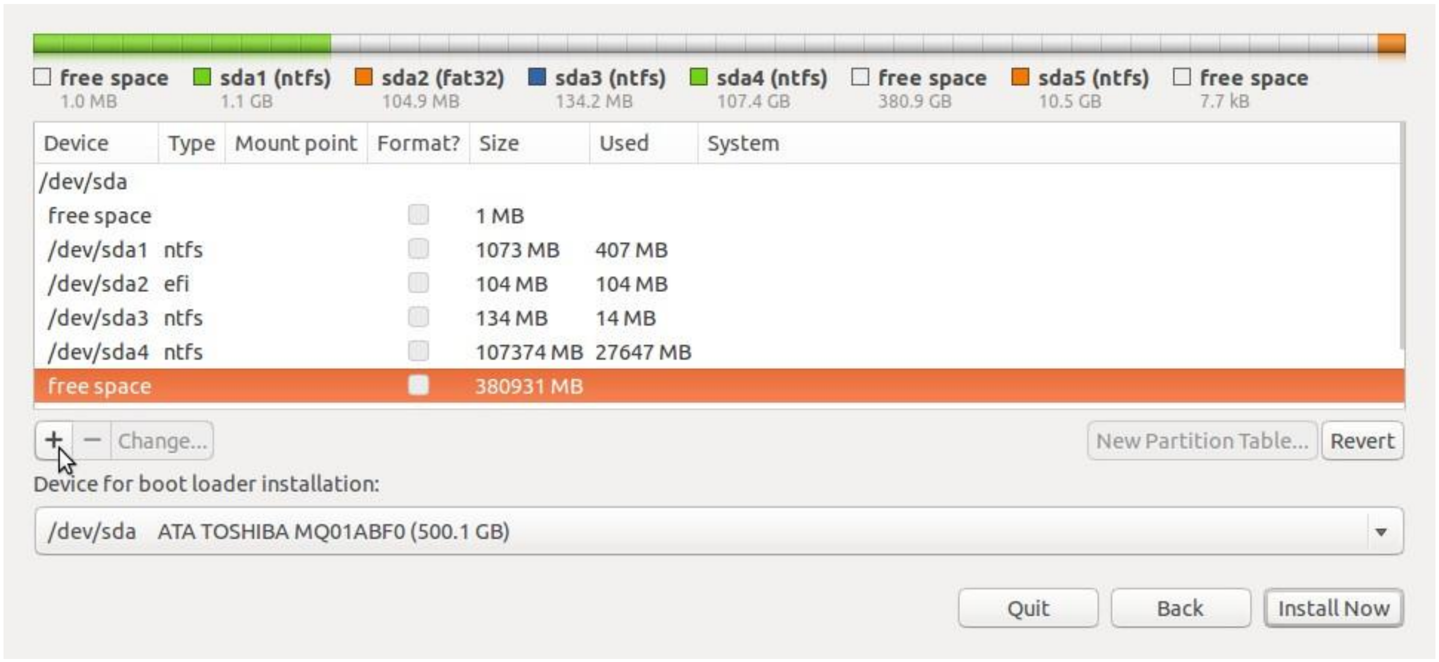
Ubuntu uses third-party software to play Flash, MP3 and other media, and to work with some graphics and wi-fi hardware. Some of this software is proprietary. The software is subject to license terms included with its documentation.

Install this third-party software

Fluendo MP3 plugin includes MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Technicolor SA.

i). When prompted, choose the option to install Linux alongside Windows. The installer will detect the existing Windows installation and provide an option for dual boot.
ii). If you don't see the option, select "Something else" to manually partition the drives.

8. Partition the Drive



☐ free space	🟩 sda1 (ntfs)	🟧 sda2 (fat32)	🟦 sda3 (ntfs)	🟩 sda4 (ntfs)	☐ free space	🟧 sda5 (ntfs)	☐ free space
1.0 MB	1.1 GB	104.9 MB	134.2 MB	107.4 GB	380.9 GB	10.5 GB	7.7 kB

Device	Type	Mount point	Format?	Size	Used	System
/dev/sda						
free space			☐	1 MB		
/dev/sda1	ntfs		☐	1073 MB	407 MB	
/dev/sda2	efi		☐	104 MB	104 MB	
/dev/sda3	ntfs		☐	134 MB	14 MB	
/dev/sda4	ntfs		☐	107374 MB	27647 MB	
free space			☐	380931 MB		

Device for boot loader installation:

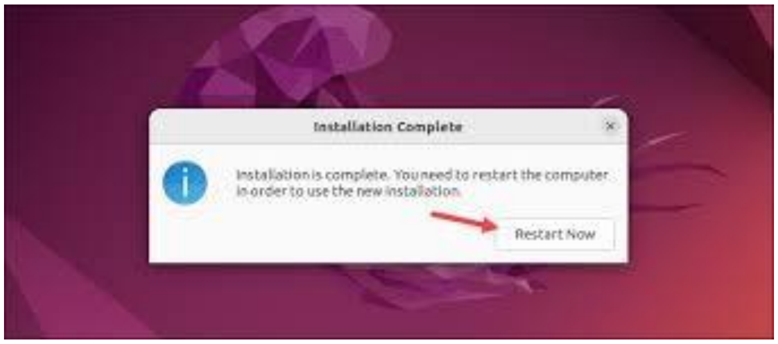
/dev/sda ATA TOSHIBA MQ01ABF0 (500.1 GB)

i). Create a Root Partition: Select the free space you created earlier and click "Add." Set the size, use Ext4 file system, and mount point as `/`.
ii). Create a Swap Partition: If desired, create a swap partition. The size of the swap partition is typically equal to the amount of RAM, but this can vary based on your needs.
iii). Create a Home Partition: Optionally, create a separate `/home` partition for user data.

9. Install the Bootloader

Ensure that the bootloader (GRUB) is installed on the same drive as your Windows installation. This is typically the default setting.

10. Complete the Installation



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

i). Follow the remaining prompts to complete the installation process.
ii). Set up your user account and password.
iii). The installer will copy files and configure the system. This may take some time.

11. Restart the Computer

Once the installation is complete, restart your computer. Remove the installation media when prompted.

12. Select the Operating System

Upon reboot, you will be presented with a GRUB menu, allowing you to choose between Windows and Linux.

EXPERIMENT 2

2 a) Implement in c language the following Unix commands using system calls i) cat ii) ls iii) mv

AIM: Implement the following UNIX commands functionality in C Programming Language using system calls
i) cat ii) ls iii) mv

i) cat command

DESCRIPTION:

The cat command in Unix-like operating systems is used to concatenate and display the contents of files. It reads one or more files and prints their content sequentially to the standard output (usually the terminal). Additionally, cat can be used to create, append to, or concatenate files. It is a versatile tool commonly used in shell scripts and for viewing file contents quickly.

SOURCE CODE:

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

int main(int argc, char *argv[3]) {
    int fd, i;
    char buf[2];
    fd = open(argv[1], O_RDONLY, 0777);
    if (fd == -argc)
    {
        printf("file open error");
    }
    else
    {
        while ((i = read(fd, buf, 1)) > 0)
        {
            printf("%c", buf[0]);
        }
        close(fd);
    }
    return 0;
}
```

2 ii) **ls command**

AIM: To Implement the following ls command functionality in C programming language using system calls

DESCRIPTION:

The ls command in Unix-like operating systems is used to list directory contents. It displays information about files and directories in the current directory by default, showing file names, permissions, size, and modification date. Options such as -l provide a detailed listing with additional information like owner, group, and file permissions, while -a includes hidden files starting with a dot.

SOURCE CODE:

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

int main(){
    DIR * dir = opendir(".");
    if(dir == NULL){
        printf("Error opening directory");
        return 1;
    }
    struct dirent * entry;
    while((entry = readdir(dir))!=NULL){
        printf("%s\n",entry->d_name);
    }
    closedir(dir);
    return 0;
}
```

2 iii) **mv command**

AIM: To Implement the following mv command functionality in C programming language using system calls

DESCRIPTION:

The mv command in Unix-like operating systems is used to move files or directories from one location to another. It can also be used to rename files or directories by specifying the current name followed by the desired new name. The syntax typically involves specifying the source file or directory and the destination path. mv preserves file attributes like permissions and timestamps during the move operation.

SOURCE CODE:

```
#include <sys/stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
    int i, fd1, fd2;
    char *file1, *file2, buf[2];
    file1 = argv[1];
    file2 = argv[2];
    printf("file1=%s file2=%s\n", file1, file2);
    fd1 = open(file1, O_RDONLY, 0777);
    fd2 = creat(file2, 0777);
    while (i = read(fd1, buf, 1) > 0)
        write(fd2, buf, 1);
    printf("%s is copied to %s\n", file1, file2);
    remove(file1);
    printf("%s is removed\n", file1);
    close(fd1);
    close(fd2);
}
```

2 b) Write a C program to create child process and allow parent process to display “parent” and the child to display “child” on the screen

AIM: To write a C program to create a Parent and child process and display the “child” and “parent” processes

DESCRIPTION:

The fork() function in Unix-like systems creates a new process by duplicating the existing process. After calling fork(), two processes run concurrently: the parent process continues from the point of fork() with the child process starting from the same state. The parent process receives the child's process ID (PID) as a return value, while the child process receives 0. This system call is crucial for multitasking and parallel processing in Unix-based operating systems.

SOURCE CODE :

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid;
    pid = fork();

    if (pid < 0) {
        printf("Fork failed\n");
        return 1;
    } else if (pid == 0) {
        printf("Child Process\n");
    } else {
        wait(NULL);
        printf("Parent Process\n");
    }
    return 0;
}
```

EXPERIMENT 3

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

Assume you have the following jobs to execute with one processor with the jobs arriving in the order listed here:

i	*T(pi)*
0	*80*
1	*20*
2	*10*
3	*20*
4	*50*

a. Suppose a system uses FCFS scheduling. Create a Gantt chart illustrating the execution of these processes?
b. What is the average turnaround time for the processes?
c. What is the average wait time for the processes?

DESCRIPTION:

FCFS (First-Come, First-Served) is a scheduling algorithm used in operating systems where processes are executed in the order they arrive. It operates on a non-preemptive basis, meaning once a process starts, it continues until completion or it voluntarily yields control. FCFS is straightforward to implement but can lead to longer average waiting times, particularly if shorter processes arrive after longer ones, causing potential delays for subsequent tasks.

- **Arrival Time**: This is the time at which a process enters the ready queue and requests CPU execution.

- **Burst Time**: Also known as execution time, this is the amount of time a process requires on the CPU to complete its execution.

- **Turnaround Time**: The total time taken for a process to complete execution, calculated as the difference between the completion time and the arrival time.

- **Waiting Time**: The total time a process spends waiting in the ready queue before it starts executing on the CPU, calculated as the difference between turnaround time and burst time.

- **Response Time**: The time taken from the submission of a request until the first response is produced, often considered in interactive systems to measure how quickly a process starts responding to input.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the burst time Step 4: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate
Waiting time (n) = waiting time (n-1) + Burst time (n- Turnaround time (n) = waiting time (n)+Burst time(n).

Step 6: Calculate
Average waiting time = Total waiting Time / Number of process Average Turnaround time = Total Turnaround Time / Number of process

Step7:stop

GANTT CHART:

SOURCE CODE:

```
#include<stdio.h>


int main(){
    int processes;
    printf("Enter the number of processes : ");
    scanf("%d",&processes);


    int burstTimes[processes],arrivalTimes[processes],ganttProcess[processes];
    int tempB,tempA,sortTemp;


    printf("\nEnter the burst times and arrival times of the processes \n");
    for(int i = 0; i<processes; i++){
        ganttProcess[i] = i;
        printf("Burst time and arrival time of process %d : ",i);
        scanf("%d%d",&tempB,&tempA);
    burstTimes[i] = tempB;
    arrivalTimes[i] = tempA;
}
for(int i = 0; i<processes;i++){
    for(int j = 0;j<processes-i-1;j++){
        if(arrivalTimes[j] > arrivalTimes[j+1]){
            sortTemp = arrivalTimes[j];
            arrivalTimes[j] = arrivalTimes[j+1];
            arrivalTimes[j+1] = sortTemp;
            sortTemp = burstTimes[j];
            burstTimes[j] = burstTimes[j+1];
            burstTimes[j+1] = sortTemp;
            sortTemp = ganttProcess[j];
            ganttProcess[j] = ganttProcess[j+1];
            ganttProcess[j+1] = sortTemp;


        }
    }
}
int turnAroundTime[processes],waitTime[processes];
int cumulativeTime = 0;
printf("\nGantt Chart for FCFS scheduling is : \n");
for(int i = 0;i<processes*2;i++){
    printf("--------");
}
printf("\n");
printf("|");
for(int i = 0;i<processes;i++){
    printf("\tP%d\t|",ganttProcess[i]);
}
printf("\n");
for(int i = 0;i<processes*2;i++){
    printf("--------");
    }
    printf("\n");
    printf("0\t\t");
    for(int i = 0;i<processes;i++){
        cumulativeTime += burstTimes[i];
        printf("%d\t\t",cumulativeTime);
        turnAroundTime[i]  = cumulativeTime-arrivalTimes[i];
        waitTime[i] = turnAroundTime[i]-burstTimes[i];
    }
    printf("\n\n");
    int sumTAT = 0,sumWT = 0;
    for(int i = 0;i<processes;i++){
        sumTAT += turnAroundTime[i];
        sumWT += waitTime[i];
    }
    printf("Average Turn Around Time for the processes is : %d\n",sumTAT/processes);
    printf("Average Waiting Time for the processes is : %d\n",sumWT/processes);
    return 0;
}
```

EXPERIMENT NO.4

4a) Write a C program that illustrate communication between two unrelated process using named pipes

AIM: To illustrate communication between two unrelated process using named pipes

DESCRIPTION:

Named pipes, also known as FIFOs, facilitate communication between two independent processes through a file-like interface. Using mkfifo(), a named pipe is created in the filesystem, allowing one process to write data to it and another process to read from it. This sequential data flow enables effective inter-process communication, handling synchronization seamlessly without requiring shared memory or other intricate mechanisms.

SOURCE CODE:

Namedpipe.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
int main()
{
    int res;
    res = mkfifo("fifo1", 0777);
    printf("named pipe created\n");
}
```

Sender.c

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

int main()
{
    int res;
    res = open("fifo1", O_WRONLY);
    if (res == -1) {
        printf("Error opening FIFO");
        return 1;
    }
    write(res, "Hello World", 12);
    printf("Sender Process %d sent the data\n", getpid());
    close(res);
    return 0;
}
```

Receiver.c

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
int main()
{
    int res, n;
    char buffer[100];
    res = open("fifo1", O_RDONLY);
    n = read(res, buffer, 100);
    printf("Reader process %d started\n", getpid());
    printf("Data received by receiver %d is: %s\n", getpid(), buffer);
}
return 0;
}
```

4b) Write a C program that receives a message from message queue and display them

AIM: To receive a message from message queue and display them

DESCRIPTION:

Message queue is a form of inter-process communication (IPC) mechanism where processes can send and receive messages asynchronously. They are managed by the operating system kernel and provide a structured way for processes to exchange data, enabling tasks like task scheduling, event notification, and synchronization between processes. Receiving messages from a message queue involves processes or threads actively fetching and processing queued messages, typically using blocking or non-blocking mechanisms.

SOURCE CODE:

Sender.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct msgbuf
{
   long mtype;
   char mtext[200];
};
int main()
{
   key_t key = 1234;
   int msg_id;
   struct msgbuf msg;
   msg_id = msgget(key, 0666 | IPC_CREAT);
   if (msg_id < 0)
   {
      perror("msgget");
      exit(EXIT_FAILURE);
   }
   msg.mtype = 1;
   strcpy(msg.mtext, "Hello, this message is from sender");
   if (msgsnd(msg_id, &msg, strlen(msg.mtext), 0) < 0)
   {
      perror("msgsnd");
      exit(EXIT_FAILURE);
   }
   printf("Sent: %s\n", msg.mtext);
   return 0;
}
```

Receiver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct msgbuf
{
    long mtype;
    char mtext[200];
};
int main()
{
    key_t key = 1234;
    int msg_id;
    struct msgbuf msg;
    msg_id = msgget(key, 0666 | IPC_CREAT);
    if (msg_id < 0)
    {
        perror("msgget");
        exit(EXIT_FAILURE);
    }
    if (msgrcv(msg_id, &msg, sizeof(msg.mtext), 0, 0) < 0)
    {
        perror("msgrcv");
        exit(EXIT_FAILURE);
    }
    printf("Received: %s\n", msg.mtext);
    if (msgctl(msg_id, IPC_RMID, NULL) < 0)
    {
        perror("msgctl");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

4c) Write a C program to allow cooperating process to lock a resource for exclusive use (using semaphore)

AIM: To allow cooperating process to lock a resource for exclusive use (using semaphore)

DESCRIPTION:

Semaphores in operating systems are synchronization mechanisms that manage access to shared resources among concurrent processes or threads, using integer counters and atomic operations. They are used to prevent race conditions and manage critical sections by allowing or blocking processes based on the semaphore's current value, modified by operations like wait (P) and signal (V). Semaphores play a crucial role in ensuring mutual exclusion, synchronization, and coordination in multi-process or multi-threaded environments, facilitating efficient resource management and concurrency control.

REQUIREMENTS:

Implementing a bank account transaction system.

1. The initial account balance is $1000.
2. Use POSIX semaphores to synchronize access to the shared account balance.
3. Implement two functions: deposit(int amount) and withdraw(int amount).
    - deposit(int amount): Increases the account balance by the specified amount and print the new balance
    - withdraw(int amount): Decreases the account balance by the specified amount if there are sufficient funds. If not, it prints an error message indicating insufficient funds.
    - The child process should perform a deposit & the parent process should attempt to withdraw.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <fcntl.h>
#include <sys/mman.h>

void deposit(int *balance , int amount){
   *balance += amount;
   printf("Depositing %d to account\n",amount);
   printf("New balance after deposit: %d\n",*balance);
}

void withdraw(int * balance , int amount){
   if(amount > *balance){
        printf("Insufficient funds\n");
        return;
    }
    *balance-=amount;
    printf("Withdrawing %d from account\n",amount);
    printf("New balance after withdrawal: %d\n",*balance);
}

int main()
{
    sem_t mutex;
    sem_init(&mutex, 0, 1);

    int shm_fd = shm_open("/shared_balance", O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Failed to create shared memory");
        exit(1);
    }

    if (ftruncate(shm_fd, sizeof(int)) == -1) {
        perror("Failed to set size of shared memory");
        exit(1);
    }

    int *initialBalance = mmap(0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (initialBalance == MAP_FAILED) {
        perror("Failed to map shared memory");
        exit(1);
    }
    *initialBalance = 1000;

    pid_t pid = fork();
    if(pid==0){
        sem_wait(&mutex);
        int amt = 500;
        deposit(initialBalance,amt);
        sem_post(&mutex);
    }
    else if (pid>0)
    {
        sem_wait(&mutex);
        int amt = 200;
        withdraw(initialBalance,amt);
        sem_post(&mutex);

    }
    else{
        printf("Error: fork() failed\n");
    }
    return 0;
}
```

4d) Write a C program that illustrate the suspending and resuming process using signal

AIM: To illustrate the suspending and resuming process using signal

DESCRIPTION :

Signals in operating systems are software interrupts used to notify processes of events or to request their attention for various reasons, such as hardware exceptions, user-defined events, or inter-process communication.

To suspend a process, we typically send it the `SIGSTOP` signal.

To resume a process, we send it the `SIGCONT` signal.

SOURCE CODE :

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/wait.h>

int main()
{
    pid_t pid = fork();
    if (pid == -1)
    {
        printf("Error : Fork() is failed ");
        exit(1);
    }
    if (pid == 0)
    {
        printf("Child process is running...\n");
    }
    else
    {
        printf("Parent process started\n");
        sleep(2);
        if (kill(pid, SIGSTOP) == -1)
        {
            perror("Failed to suspend child process");
            exit(EXIT_FAILURE);
        }

        printf("Parent has suspended the child process\n");
        sleep(2);
        if (kill(pid, SIGCONT) == -1)
        {
            perror("Failed to resume child process");
            exit(EXIT_FAILURE);
        }

        printf("Parent has resumed the child process\n");
        sleep(2);

        if (kill(pid, SIGKILL) == -1)
        {
            perror("Failed to terminate the child process");
        }
        if (wait(NULL) == -1)
        {
            printf("failed to wait for child..");
        }
    }

    return 0;
}
```

4e) Write a C program that implements producer-Consumer system with two process using semaphore

AIM: To implement producer-consumer system with two processes using semaphore

DESCRIPTION :

The producer-consumer problem is a classic synchronization issue in computer science where multiple processes (producers) generate data items and place them into a shared buffer, while other processes (consumers) retrieve and consume these items.

Semaphores: `sem_init()` initializes semaphores (`empty`, `full`, `mutex`). `sem_wait()` decrements a semaphore, `sem_post()` increments a semaphore, and `sem_destroy()` destroys a semaphore.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <fcntl.h>

#define SEM_PRODUCER "/semProducer"
#define SEM_CONSUMER "/semConsumer"
int main()
{
   sem_t *semProducer, *semConsumer;
   semProducer = sem_open(SEM_PRODUCER, O_CREAT, 0644, 1);
   semConsumer = sem_open(SEM_CONSUMER, O_CREAT, 0644, 0);
   if (semProducer == SEM_FAILED || semConsumer == SEM_FAILED)
   {
      perror("sem_open failed");
      exit(EXIT_FAILURE);
   }

   pid_t p;
   p = fork();

   if (p > 0)
   {
      for (int i = 0; i < 5; i++)
      {
         sem_wait(semProducer);
         printf("Producing Item\n");
         sem_post(semConsumer);
         sleep(2);
      }
      wait(NULL);
      sem_close(semProducer);
      sem_close(semConsumer);
      sem_unlink(SEM_PRODUCER);
      sem_unlink(SEM_CONSUMER);
```

```
    }
    else
    {
        for (int i = 0; i < 5; i++)
        {
            sem_wait(semConsumer);
            printf("Consuming item\n");
            sem_post(semProducer);
            sleep(2);
        }
    }
}
```

EXPERIMENT NO.5

AIM : Banker’s Algorithm

Consider the following snapshot of a system. P0, P1, P2, P3, P4 are the processes and A, B, C, D are the resource types. The values in the table indicate the number of instances of a specific resource (for example: 3 3 2 1 under the last column indicates that there are 3 A-type, 3 B-type, 2 C-type and 1 D-type resources available after allocating the resources to all five processes). The numbers under allocation-column indicate that those resources are allocated to various processes mentioned in the first column. The numbers under Max- column indicate the maximum number of resources required by the processes. For example: in the 1st row under allocation- column 2 0 0 1 indicate there are 2 A-type, 0 B-type, 0 C-type and 1 D- type resources allocated to process P0. Whereas 4 2 1 2 under Max-column indicate that process P0‘s maximum requirement is 4 A- type, 2 B-type, 1 C-type and 2 D-type resources.

Process	Allocation A B C D	Max A B C D	Available A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 12	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

Answer the following questions using banker's algorithm by providing all intermediate steps

a. How many instances of resources are present in the system under each type of a resource?

b. Compute the Need matrix for the given snapshot of a system.

c. Verify whether the snapshot of the present system is in a safe state demonstrating an order in which the processes may complete. If a request from process P1 arrives for (1,1,0,0), can the request be granted immediately?

d. If a request from process P4 arrives for (0,0,2,0), can the request be granted immediately?

DESCRIPTION :

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm used in operating systems to ensure safe allocation of resources to processes. It operates by simulating the allocation of resources and checks whether granting a request from a process could lead to deadlock or unsafe state, ensuring that resources are allocated in a way that prevents both deadlock and starvation.

SOURCE CODE :

```
#include <stdio.h>

#include <stdbool.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10


int allocation[MAX_PROCESSES][MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];
bool finish[MAX_PROCESSES];
int num_processes, num_resources;
int i, j;
void calculate_need_matrix() {

    for (i = 0; i < num_processes; i++) {
        for (j = 0; j < num_resources; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
}
}
bool is_safe_state(int work[], int safe_sequence[]) {
    int finish_count = 0;
    while (finish_count < num_processes) {
        bool found = false;
        for (i = 0; i < num_processes; i++) {
            if (!finish[i]) {
```

```
            bool possible = true;

            for (j = 0; j < num_resources; j++) {
                if (need[i][j] > work[j]) {
                    possible = false;
                    break;
                }

            }
            if (possible) {

                for (j = 0; j < num_resources; j++) {
                    work[j] += allocation[i][j];
                }

                finish[i] = true;
                safe_sequence[finish_count++] = i;
                found = true;
            }

        }

    }
    if (!found) {

        return false; }

  }
  return true;

}
 void handle_resource_request(int process_id, int request[]) {
  for (i = 0; i < num_resources; i++) {
    if (request[i] > available[i] || request[i] > need[process_id][i]) {
        printf("first one cannot be granted immediately.\n");
        return;

    }

  }

  for (i = 0; i < num_resources; i++) {
    available[i] -= request[i];
    allocation[process_id][i] += request[i];
    need[process_id][i] -= request[i];
  }

  int work[num_resources];

  for (i = 0; i < num_resources; i++) {
        work[i] = available[i];
    }
    int safe_sequence[num_processes];

    if (is_safe_state(work, safe_sequence)) {
        printf("Request cannot be granted immediately.\n");
        for (i = 0; i < num_resources; i++) {
            available[i] += request[i];
            allocation[process_id][i] -= request[i];
            need[process_id][i] += request[i];
        }

    } else {

        printf("process %d request can be granted immediately\n",process_id);

    }

}
int main() {

    printf("enter no.of Process : ");
    scanf("%d", &num_processes);
    printf("enter no.of Resources : ");

    scanf("%d", &num_resources);
    printf("enter Allocation matrix: ");
    for (i = 0; i < num_processes; i++) {
        for (j = 0; j < num_resources; j++) {
            scanf("%d", &allocation[i][j]);
        }

    }
    printf("enter maximum Matrix : ");
    for (i = 0; i < num_processes; i++) {
        for (j = 0; j < num_resources; j++) {
            scanf("%d", &max[i][j]);
        }

    }
    printf("enter available matrix : ");
    for (i = 0; i < num_resources; i++) {
        scanf("%d", &available[i]);

    }
    calculate_need_matrix();
    int total_instances[num_resources];
    for (i = 0; i < num_resources; i++) {
        total_instances[i] = available[i];
        for (j = 0; j < num_processes; j++) {
```

```
        total_instances[i] += allocation[j][i];
    }
}
printf("total instances of each resource type : ");
for (i = 0; i < num_resources; i++) {
    printf("%d ", total_instances[i]);

}

printf("\n");
int work[num_resources];

for (i = 0; i < num_resources; i++) {
    work[i] = available[i];
}

        int safe_seq = 1, want_continue=1;
        int choice;
        int iter = 0;
        do {

             int safe_sequence[num_processes];
            if (safe_seq == 1) {
                    if (iter == 0) {
                                if (is_safe_state(work, safe_sequence)) {

                                        printf("The system is in safe state and the safe sequence is : ");
                                        for (i = 0; i < num_processes; i++) {
                                            printf("%d ", safe_sequence[i]);

                                        }

                                        printf("\n");

                                    } else {

                                        printf("The system is not in a safe state.\n");
                                }
                    }

                    else {

                                if (is_safe_state(work, safe_sequence)) {
                                        printf("The system is not in a safe state.\n");
                                    } else {
                                        printf("The system is in safe state and the safe sequence is : ");
                                        for (i = 0; i < num_processes; i++) {
                                        printf("%d ", safe_sequence[i]);

                                        }
                                        printf("\n");

                                    }
                                }
                        }

                        if (want_continue != 1) {

                                printf("Want to continue enter 1 : ");
                                scanf("%d", &want_continue);
                                printf("\n");
                        }

                        if (want_continue != 1) {
                                break;
                        }

            printf("want to check for new Resource request enter 1 : ");
            scanf("%d", &choice);
            if (choice == 1) {
                int process_id;
                printf("which process wants to generate the request : ");
                scanf("%d", &process_id);
                int request[num_resources];
                printf("Enter resource request : ");
                for (i = 0; i < num_resources; i++) {
                scanf("%d", &request[i]);
                }
                handle_resource_request(process_id,  request);
            }
            safe_seq = 0;

            want_continue = 0;
            if (safe_seq != 1){
                        printf("want to check the safe sequence enter 1 : ");
                        scanf("%d", &safe_seq);
                        printf("\n");
                        }
                        iter += 1;
        } while (choice == 1);
        return 0;
    }
```

EXPERIMENT NO.6

Q) Write a C program to simulate the following memory management technique: Paging

AIM: To simulate the following memory management technique : Paging

DESCRIPTION:

Paging is a memory management technique where the physical memory is divided into fixed-size blocks called "frames," and the logical memory is divided into blocks of the same size called "pages." This program simulates a basic paging system by initializing a page table, loading pages into frames, and translating logical addresses to physical addresses. It demonstrates how logical addresses are mapped to physical addresses using the page table, and it handles page faults when a page is not in memory. This simple simulation illustrates the fundamental concepts of paging without delving into complex page replacement algorithms.

SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define PAGE_SIZE 1024

#define NUM_PAGES 32

#define MEMORY_SIZE (PAGE_SIZE * NUM_PAGES)

char physical_memory[MEMORY_SIZE];

void initialize_memory() {

    for (int i = 0; i < MEMORY_SIZE; ++i) {

        physical_memory[i] = 0;

    }

}

void page_fault(int page_number) {

    printf("Page fault occurred for page %d\n", page_number);

    printf("Loading page %d into physical memory...\n", page_number);

    for (int i = page_number * PAGE_SIZE; i < (page_number + 1) * PAGE_SIZE; ++i) {

        physical_memory[i] = page_number + 1;
    }
}
void access_memory(int address) {
    int page_number = address / PAGE_SIZE;
    int offset = address % PAGE_SIZE;
    if (physical_memory[page_number * PAGE_SIZE] == 0) {
        page_fault(page_number);
    }
    printf("Accessing memory location %d: Value = %d\n", address, physical_memory[address]);
}
int main() {
    initialize_memory();
    access_memory(4096);
    access_memory(2048);
    access_memory(8192);
    access_memory(10240);
    access_memory(3072);
    return 0;
}
```

EXPERIMENT NO.7

7a) Write a C program that takes one or more file/directory names as command line input and reports following information

i) File Type ii) Number of Links iii) Time of last Access iv) Read, write and execute permissions.

AIM : To write a C program that takes one or more file/directory names as command line input and reports the following information :
i)File Type ii) Number of Links iii) Time of last Access iv) Read, write and execute permissions

DESCRIPTION :

To achieve this objective in C, you'll utilize POSIX system calls and data structures to retrieve and analyze file metadata. POSIX, which stands for Portable Operating System Interface for Unix, comprises IEEE-defined standards that establish APIs, shell interfaces, and utility interfaces to ensure compatibility across Unix-like operating systems.

SOURCE CODE :

```
 #include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <unistd.h>

#include <time.h>

#include <pwd.h>

#include <grp.h>

#include <string.h>

#include <errno.h>

void print_file_info(char *filename) {

struct stat file_stat;

   if (stat(filename, &file_stat) == -1) {

       fprintf(stderr, "Error accessing %s: %s\n", filename, strerror(errno));

       return;

   }

   printf("File Type: ");
```

```
if (S_ISREG(file_stat.st_mode))
    printf("Regular File\n");
else if (S_ISDIR(file_stat.st_mode))
    printf("Directory\n");
else if (S_ISCHR(file_stat.st_mode))
    printf("Character Device\n");
else if (S_ISBLK(file_stat.st_mode))
    printf("Block Device\n");
else if (S_ISFIFO(file_stat.st_mode))
    printf("FIFO/Named Pipe\n");
else if (S_ISLNK(file_stat.st_mode))
    printf("Symbolic Link\n");
else if (S_ISSOCK(file_stat.st_mode))
    printf("Socket\n");
else
    printf("Unknown\n");
printf("Number of Links: %ld\n", (long) file_stat.st_nlink);
printf("Last Access Time: %s", ctime(&file_stat.st_atime));
printf("Permissions (rwx): ");
printf( (file_stat.st_mode & S_IRUSR) ? "r" : "-");
printf( (file_stat.st_mode & S_IWUSR) ? "w" : "-");
printf( (file_stat.st_mode & S_IXUSR) ? "x" : "-");
printf( (file_stat.st_mode & S_IRGRP) ? "r" : "-");
printf( (file_stat.st_mode & S_IWGRP) ? "w" : "-");
printf( (file_stat.st_mode & S_IXGRP) ? "x" : "-");
printf( (file_stat.st_mode & S_IROTH) ? "r" : "-");
printf( (file_stat.st_mode & S_IWOTH) ? "w" : "-");
printf( (file_stat.st_mode & S_IXOTH) ? "x\n" : "-\n");
```

```
    struct passwd *pw = getpwuid(file_stat.st_uid);

    struct group *gr = getgrgid(file_stat.st_gid);

    if (pw != NULL)

        printf("Owner: %s\n", pw->pw_name);

    if (gr != NULL)

        printf("Group: %s\n", gr->gr_name);

}

int main(int argc, char *argv[]) {

    if (argc < 2) {

        fprintf(stderr, "Usage: %s <file/directory names>\n", argv[0]);

        return 1;

    }

    for (int i = 1; i < argc; ++i) {

        printf("File Information for: %s\n", argv[i]);

        print_file_info(argv[i]);

        printf("\n");

    }
    return 0;

}
```

7b) Write a C program to list every file in directory, its inode number and file name

AIM : To write a C program that to list every file in directory, its inode number and file name

DESCRIPTION :

To traverse directories in C, utilize readdir() to iterate through each entry (struct dirent) within the directory. Employ snprintf() and stat() to gather file details (struct stat). Additionally, for error management, leverage perror() from the errno.h library to handle errors effectively.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <errno.h>


int main(int argc, char *argv[]) {
DIR *dir;
struct dirent *entry;
struct stat file_stat;

if (argc < 2) {
        dir = opendir(".");
}
else {
        dir = opendir(argv[1]);
}
if (dir == NULL) {
        perror("Error opening directory");
        return EXIT_FAILURE;
}

while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) {
        continue;
        }

        char path[1024];
        snprintf(path, sizeof(path), "%s/%s", argv[1], entry->d_name);
        if (stat(path, &file_stat) == -1) {
                perror("Error stat"); continue;
        }

        printf("Inode: %lu\t File: %s\n", file_stat.st_ino, entry→d_name);

    }

closedir(dir);

return EXIT_SUCCESS;

}
```