OPERATING SYSTEM

Subject Code:- 4ITRC2

LAB ASSIGNMENT

BE IIND YEAR (IT-A)

Semester-4

JAN 2025 - MAY 2025

SUBMITTED TO:-

Er. Ms. Jasneet K. Monga

SUBMITTED BY:-

Vishesh Jain
Enrol. No-DE23627
Roll No-23I4077
Information Technology-'A'

Department of Information Technology

Institute of Engineering and Technology,

Devi Ahilya Vishwavidyalaya, Indore (M. P.)

TABLE OF CONTENT

LAB ASSIGNMENT -1

1. INTRODUCTION(LINUX &UBUNTU)

Overveiw Of Linux:-

- Linux refers to a family of operating systems modeled off of Unix
    - Can perform many of the same functions as Windows or OS X
    - Built in a collaborative, open-source environment -
- Anyone may use, modify, or distribute the Linux kernel -
- Anyone can develop software to run on the Linux kernel -
- Many programmers collaborate to develop or improve Linux programs -
- Many Linux operating systems and add-on programs are free

What is Linux Operaring System:-

Linux is a free and open-source family of operating systems that is resilient and flexible. In 1991, an individual by the name as Linus Torvalds constructed it. The system's source code is accessible to everyone for anyone to look at and change, making it cool that anyone can see how the system works. People from all across the world are urged to work together and keep developing Linux due to its openness.

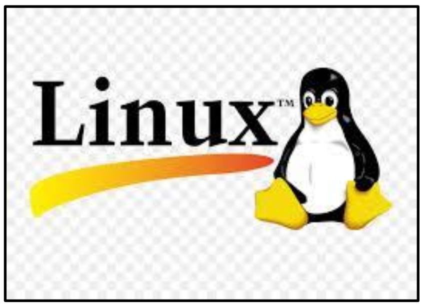


Figure 1:- Logo Of Linux

Developed by Linus Torvalds in 1991, the Linux operating system is a powerful and flexible open-source software platform. It acts as the basis for a variety of devices, such embedded systems, cell phones, servers, and personal computers. Linux, that's well-known for its reliability, safety, and flexibility, allows users to customize and improve their environment to suit specific needs. With an extensive and active community supporting it, Linux is an appealing choice for people as well as companies due to its wealth of resources and constant developments.

History Of Linux:-

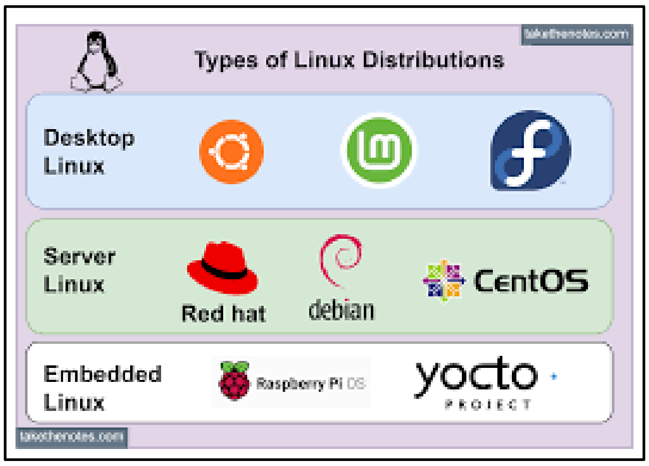
Linus Torvalds designed the free and open-source Linux operating system kernel in 1991. Torvalds set out to develop a free and flexible system for personal computers, drawing ideas from the UNIX operating system and the MINIX operating system. Teamwork in development was encouraged with the initial release of the Linux kernel, which attracted developers and enthusiasts globally quickly. Various open-source software packages integrated with the Linux kernel created fully operational operating systems, occasionally referred to as Linux distributions.
Over the years, Linux has become known as a key component of modern computing, powering everything from servers and personal computers to supercomputers and smartphones. Due to its flexibility, durability, and strong community support, developers, businesses, and educational institutions frequently opt for it.

Different Types Of Linux:-

There are many different flavors (OSs) built off the Linux kernel
- **Ubuntu**: Most popular flavor. It is free and isthe most user-friendly.
- **Mint**: A popular variation of Linux

-**Red Hat:** Designed by a company that developsspecialized flavors for government and big business
- **Fedora:** An open-source, free version of RedHat. Used frequently as a test bed for Red Hat programs.

• These flavors are similar at the basic level,but can have very different interfaces and specialized commands.



Overveiw Of Ubuntu:-

Ubuntu is a Linux-based operating system. It is designed for computers, smartphones, and network servers. The system is developed by a UK based company called Canonical Ltd. All the principles used to develop the Ubuntu software are based on the principles of Open Source software development**.**
Ubuntu is based on Debian Linux. Debian is a large Linux project with active community participation. Ian Murdock, a young computer expert, started the Debian Linux project in 1993. Debian is a popular Linux-based operating system among programmers and system administrators.

Using Debian Linux was not simple for everyday use by ordinary people. Ubuntu has made several changes to the Debian Linux for public utility.
Ubuntu mainly uses the GNU General Public License. Several operating systems were again developed based on Ubuntu. Linux Mint, Zorin OS, Elementary OS, Peppermint OS and many more operating systems built on the Ubuntu platform.

What is Ubuntu Operaring System:-

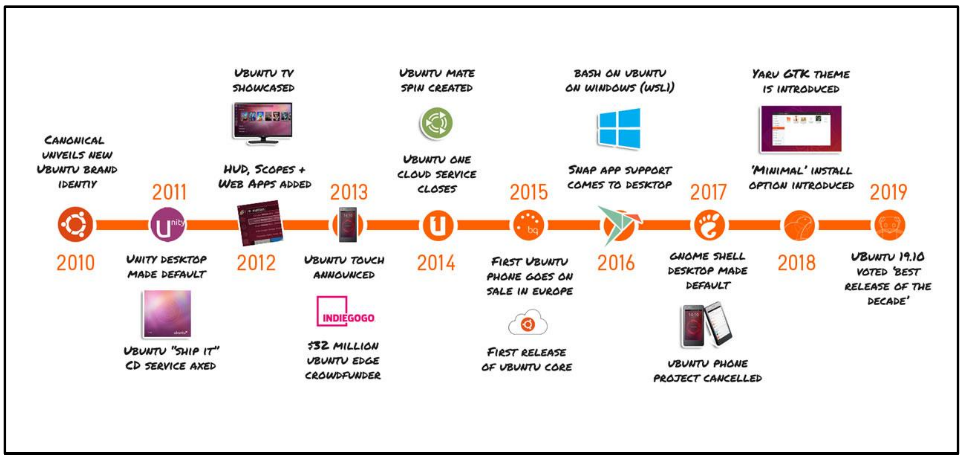
Ubuntu (pronounced oo-BOON-too) is a free, open source operating system (OS) based on Debian Linux. It was first released in 2004 when Mark Shuttleworth and a small team of Debian developers founded Canonical and then launched the Ubuntu project. Canonical released its first official version of the OS -- Ubuntu 4.10 -- in October 2004. The word *ubuntu* comes from the southern African Nguni languages and translates as "humanity to others."

As the manager of Ubuntu, Canonical is responsible for releasing a new Ubuntu version every six months. Canonical also provides hosting servers for Ubuntu Community, allowing people worldwide to contribute to testing software bugs, answer questions, and give technical support for free.

History Of Ubuntu:-

Here’s a concise history of Ubuntu in key points:

1. **2004 - Launch**: Ubuntu was launched by Mark Shuttleworth and his company, Canonical Ltd., based on Debian Linux.
2. **Goal**: To create a user-friendly Linux distribution for everyone, focusing on ease of use and accessibility.
3. **First Release (Ubuntu 4.10)**: Released in October 2004 with the codename "Warty Warthog."
4. **Regular Releases**: Ubuntu adopted a regular release cycle, with new versions every six months.
5. **GNOME Desktop**: Initially used the GNOME desktop environment to provide a simple, intuitive user interface.
6. **Canonical's Role**: Canonical Ltd., the company behind Ubuntu, provided support, commercial services, and development.
7. **Ubuntu LTS (Long-Term Support)**: In 2006, Ubuntu introduced LTS versions, offering 5 years of support for stability in enterprise environments.
8. **Popularity**: Ubuntu became one of the most widely used Linux distributions due to its user-friendly nature and extensive community support.
9. **2010 - Unity Interface**: Ubuntu introduced the Unity desktop environment (later replaced by GNOME in 2017).
10. **Cloud & IoT**: Ubuntu grew beyond desktop use, with versions focused on cloud computing, IoT devices, and servers.
11. **Snap Packages**: Ubuntu introduced Snap packages in 2016 for easier software distribution across different Linux distributions.
12. **Current Status**: Ubuntu remains a major Linux distribution with regular updates, widespread use, and a large, active community.


Canonical unveils new Ubuntu brand identity
2010
2011
Unity desktop made default
Ubuntu "Ship it" CD service axed
Ubuntu TV showcased
HUD, Scopes + Web Apps added
2012
2013
Ubuntu Touch announced
$32 million Ubuntu Edge crowdfunder
INDIEGOGO
Ubuntu MATE spin created
Ubuntu One cloud service closes
2014
2015
First Ubuntu phone goes on sale in Europe
First release of Ubuntu Core
Bash on Ubuntu on Windows (WSL)
Snap app support comes to desktop
2016
2017
GNOME Shell desktop made default
Ubuntu phone project cancelled
Yaru GTK theme is introduced
'Minimal' install option introduced
2018
2019
Ubuntu 19.10 voted 'best release of the decade'

Different Versions Of Ubuntu:-

Ubuntu releases a new version every six months, with Long-Term Support (LTS) versions released every two years. LTS releases are supported for **5 years** with security updates and maintenance. These are typically more stable and are preferred for production environments.

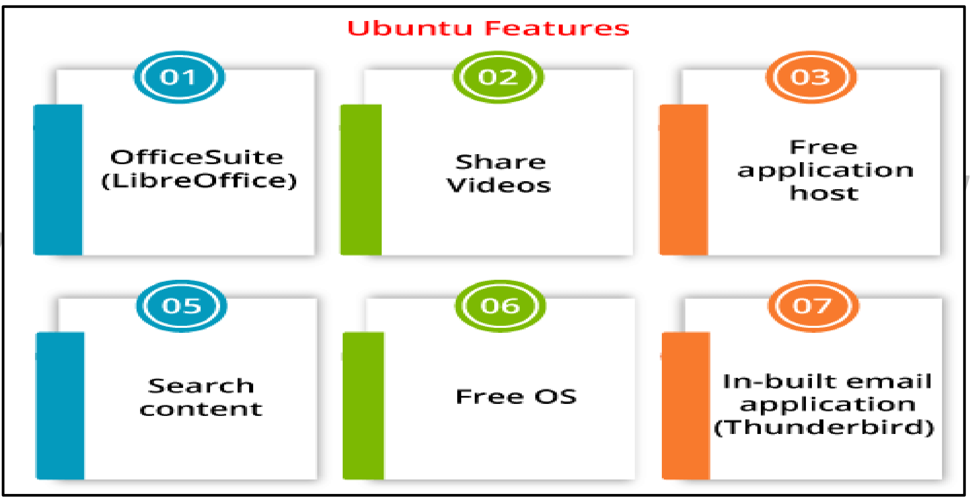
Here’s a list of the main versions of Ubuntu:

1. **Ubuntu 4.10 ("Warty Warthog") -** October 2004
2. **Ubuntu 6.06 LTS ("Dapper Drake") -** June 2006
3. **Ubuntu 8.04 LTS ("Hardy Heron") -** April 2008
4. **Ubuntu 10.04 LTS ("Lucid Lynx") -** April 2010
5. **Ubuntu 12.04 LTS ("Precise Pangolin") -** April 2012
6. **Ubuntu 14.04 LTS ("Trusty Tahr") -** April 2014
7. **Ubuntu 16.04 LTS ("Xenial Xerus") -** April 2016
8. **Ubuntu 18.04 LTS ("Bionic Beaver") -** April 2018
9. **Ubuntu 20.04 LTS ("Focal Fossa") -** April 2020
10. **Ubuntu 22.04 LTS ("Jammy Jellyfish") -** April 2022

CHOOSE THE PERFECT UBUNTU FLAVOR FOR YOU!
EDUBUNTU
KUBUNTU
LUBUNTU
UBUNTU BUDGIE
UBUNTU CINNAMON
UBUNTU KYLIN
UBUNTU MATE
UBUNTU STUDIO
UBUNTU UNITY
XUBUNTU

2. FEATURES OF UBUNTU

- The desktop release of Ubuntu supports every basic software on Windows like VLC, Chrome, Firefox, etc.
- It supports *OfficeSuite* which is known as *LibreOffice*.
- Ubuntu contains an in-built email application which is known as *Thunderbird* which provides the user access to email like Hotmail, Gmail, exchange, etc.Also, there are many applications for managing videos and they permit the users for sharing videos as well.
- The free application's host is also available for users for viewing and editing photos.
- It is easy to search content in Ubuntu using the smart searching feature.
- The best aspect is that it is a free OS and is backed by a large open-source community.



Some More Features of Ubuntu Includes:-

1. **Office software**
2. **An open-source operating system**
3. **Web browsing**
4. **Email**
5. **Photos**
6. **Videos**
7. **Gaming**
8. **A whole world of apps**
9. **Backed by Canonical**
10. **No Antivirus**
11. **Hardware autoconfiguration**
12. **Software Repositories**
13. **Multiple desktops**
14. **ssh client**

1. Office Software

In Ubuntu, we have a software called **LibreOffice,** via which we can create professional documents, spreadsheets, and presentations. **LibreOffice** is an open-source office suite that is compatible with Microsoft Office. That means we can open and modify files such as **Word documents, PowerPoint,** and **Excel spreadsheets** and share them with other people easily and quickly. Google docs can also be used directly from our desktop.

2. An Open-Source Operating System:-

In Ubuntu, our code is openly shared during the development cycle. We're transparent about our plans for future releases, so as a developer, hardware manufacturer, or OEM, we can start developing Ubuntu applications and systems right now.

3. Email

Thunderbird, Mozilla's famous email applications is included with Ubuntu, so we'll have quick access to our email from our desktop. Email works regardless of the email service we use, such as **Microsoft Exchange, Hotmail, Gmail, POP 3,** or **IMAP.**

4. Web Browsing

Ubuntu and **Firefox,** both famed for their speed and security, make browsing the web a pleasure once more. Ubuntu now supports Chrome and other browsers, which we can get via the Ubuntu Software Centre.

5. Photos

Ubuntu has a plethora of free apps to let you enjoy, edit, manage and share the photos-whatever camera you use to take photos. With excellent support for cameras and phones, we won't require any additional drivers to get started.

In Ubuntu, we can easily and quickly import, edit, organize and view our photos using **Shotwell.** We can also share our favourite photos on any of the famous websites and social media platforms.

Tools like **Gimp** and **Krita,** both accessible in the Ubuntu Software centre ad we can use these tools to edit images or create professional illustrations and designs.

6. Videos

On Ubuntu, we can watch HD videos in our browser or with the default Movie Player, **VLC,** and **OpenShot** from the Snap Store. Use **Shotcut** or **kdenlive** to edit our videos, then watch them in Movie Player.

7. Gaming

In Ubuntu, from **Sudoku to first-person shooters,** we have a number of games that will keep us engaged for hours. There are thousands of games, including titles from the Unity and Steam platforms. Choice from critically acclaimed titles like **Dota2, Kerbal Space Program, Counter-Strike: Global Offensive,** and **Borderlands: The Pre-Sequel.**

8. A Whole World of Apps

Thousands of apps are available for download on Ubuntu. Most of them are free to download and install with just a few clicks. For example, **VLC player, Firefox, Chromium, Telegram, PyCharm, Skype, Spotify, Atom, Slack, etc.**

9. Backed by Canonical

Canonical is a multinational software company that offers commercial, design, and engineering support to the project of Ubuntu. Hundreds of laptops and workstations have been pre-installed with Ubuntu by Ubuntu's hardware enablement team throughout the world.

10. No Antivirus

In the Windows environment, security practices are extremely contradictory. Most of the same companies which write Windows software also make millions of dollars providing hogging applications that safeguard Windows apps from security issues. Although Ubuntu is not malware protected, it is as secure as it needs to be for most users right out of the box, even without the addition of any expensive antivirus scanners.

11. Hardware Autoconfiguration

Another feature of Ubuntu is hardware autoconfiguration. Most hardware drivers are already included in Ubuntu. Anybody who has installed a Windows generic version of Windows (i.e., one that has not been pre-configured by a PC vendor to work with specific hardware) understands how convenient it is not to spend hours looking for drivers after the operating system has been installed.

12. Software Repositories

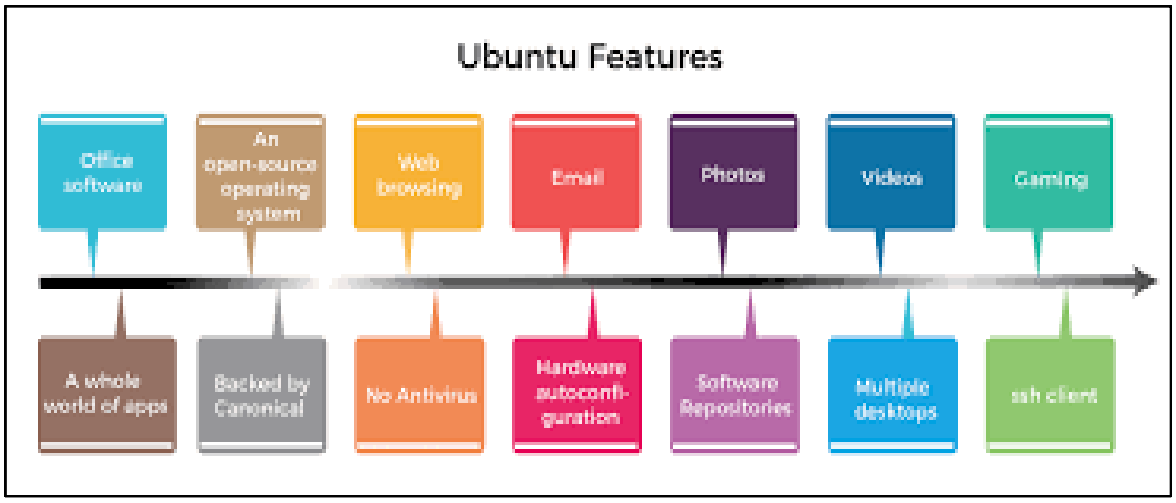
It's a tremendous advantage to install a number of applications from the repositories of Ubuntu in some clicks. Apart from the fact that the software is free and safer than .exe packages, which are downloaded from random websites, installing programs from a centralized location is far more convenient.

13. Multiple Desktops

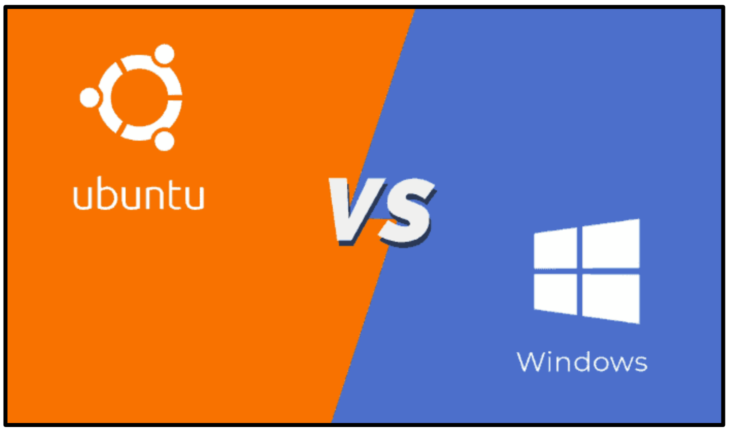
The virtual desktops are similar to tabbed web browsing- we do not understand how beneficial they are until we use them. There are various third-party tools for achieving the same capability on Windows, but few of them perform properly with Vista and higher, in our experience.

14. ssh Client

Having a ssh client embedded into the operating system is a significant advantage for us. There are several ssh clients for Windows, such as Putty, but none of them come pre-installed in Windows, and even the finest of them isn't as functional as gnome-terminal.



3. UBUNTU v/s WINDOWS OS



Here's a more detailed comparison between **Windows OS** and **Ubuntu**:

1. Licensing and Cost

- **Windows OS**:
  - **Proprietary Software**: Windows is developed by Microsoft and is a proprietary operating system, meaning its source code is not publicly available.
  - **Cost**: Windows requires users to purchase a license for each installation. The cost varies depending on the edition (e.g., Home, Pro, Enterprise).
  - **Updates**: While security updates are generally free, major updates (e.g., new versions of Windows) may also require a separate purchase or subscription (e.g., Windows 10 or Windows 11).
- **Ubuntu**:
  - **Open Source**: Ubuntu is an open-source operating system developed by Canonical Ltd. Its source code is freely available for anyone to view, modify, or distribute.
  - **Cost**: Ubuntu is completely free to download, use, and modify. There are no licensing fees.
  - **Updates**: Security updates and bug fixes are freely available. LTS (Long Term Support) versions receive five years of updates, while regular releases are supported for 9 months

.

2. User Interface

- **Windows OS**:
  - **Graphical User Interface (GUI)**: Windows is known for its user-friendly GUI, with a well-established taskbar, Start Menu, and window management system. The interface has remained relat consistent, with periodic refinements.

    - **Customizability**: While you can personalize certain aspects (e.g., themes, taskbar icons), Windows’ core interface remains largely the same and doesn't offer deep customization out-of-the-box.
    - **File Explorer**: Windows provides a graphical file management system called File Explorer, which is designed to be easy to use for accessing files, folders, and drives.
- **Ubuntu**:
    - **Graphical User Interface (GUI)**: Ubuntu traditionally used the **GNOME desktop environment** (though other desktop environments like **KDE** or **XFCE** are also available). The interface is more minimalist compared to Windows, and it emphasizes efficiency and simplicity.
    - **Customizability**: Ubuntu is highly customizable. Users can tweak the look and feel, change themes, install different desktop environments, and even design their own interface using tools like **GNOME Tweaks**.
    - **File Manager**: Ubuntu uses the **Nautilus** file manager (in GNOME), which is also easy to use, but it can feel a bit different from Windows' File Explorer in terms of layout and features.

3. Software Compatibility

- **Windows OS**:
    - **Wide Software Support**: Windows is the most widely used operating system and supports a vast number of software applications, including most commercial software like Microsoft Office, Adobe products (Photoshop, Illustrator), and popular games.
    - **Gaming**: Windows is the dominant platform for gaming, with full support for the latest games, DirectX, and high-performance gaming hardware.
    - **Legacy Support**: Many legacy applications are built specifically for Windows, making it a go-to for businesses and users with older software.
- **Ubuntu**:
    - **Open-Source and Third-Party Software**: Ubuntu primarily focuses on open-source software and has a rich collection of apps available via the **Ubuntu Software Center**. Many common applications (e.g., Firefox, LibreOffice, GIMP) are natively supported.
    - **Gaming**: While Ubuntu supports some games via platforms like **Steam** (which has a growing library of Linux games), it doesn’t natively support most high-performance games built for Windows. Some games can be played via **Wine** or **Proton**, tools that allow Windows applications to run on Linux.

- **Compatibility Layers**: Many Windows-only programs can be run on Ubuntu through compatibility layers like **Wine** or virtualization tools like **VirtualBox**.

4. System Requirements

- **Windows OS**:
  - **Higher System Demands**: Windows typically has higher system requirements. It needs a more powerful processor, more RAM (typically at least 4GB for Windows 10 or 11), and larger storage. Windows 11, for example, has strict hardware requirements like TPM 2.0.
  - **Performance**: Windows can run slower on older hardware, especially if there's a heavy load due to background processes or updates.
- **Ubuntu**:
  - **Lightweight**: Ubuntu is generally more lightweight than Windows. The minimal system requirements are lower (Ubuntu recommends 2GB RAM and 25GB of storage for a modern system). It can even run on older machines or those with limited resources.
  - **Performance**: Ubuntu tends to perform better on older hardware or lower-spec machines compared to Windows, especially if you opt for lightweight desktop environments like **Xfce** or **LXQt**.

5. Security

- **Windows OS**:
  - **Security Features**: Windows includes a firewall, Windows Defender Antivirus, and security features like BitLocker and Windows Sandbox for isolating potentially harmful software.
  - **Vulnerabilities**: Windows has historically been a target for malware, viruses, and ransomware, primarily due to its widespread use. While Microsoft has significantly improved security over the years, the large user base makes it a prime target.
  - **Frequent Updates**: Security patches are frequent, but users sometimes delay them, leaving systems exposed.
- **Ubuntu**:
  - **Built-in Security**: Ubuntu is known for being a secure system by default. It's less likely to be targeted by malware and viruses because it's not as widely used on desktops (although it's still a target for specific threats).
  - **User Permissions**: Ubuntu uses a **sudo** model for administrative access, ensuring that users must explicitly authenticate for system changes.

    - **Regular Updates**: Ubuntu's update process includes security patches that are often delivered faster than Windows. It also has security tools like **AppArmor** for application sandboxing.

6. Updates and Patches

- **Windows OS**:
    - **Automatic Updates**: Windows offers automatic updates, but they are sometimes forced on users, especially with major updates. This can lead to frustration as updates often require system reboots or can affect performance.
    - **Frequent Patches**: Windows receives monthly patches for security vulnerabilities and feature updates.
- **Ubuntu**:
    - **Package Management**: Ubuntu updates its software packages through its package manager (**APT**). Users can also install updates through the Software Center. Updates are easy to manage and are available frequently.
    - **LTS Versions**: Ubuntu releases Long Term Support (LTS) versions that receive updates for five years. These versions are more stable and have fewer major updates compared to regular releases.

7. Support

- **Windows OS**:
    - **Official Support**: Windows provides official support via Microsoft's website, forums, and customer service. Windows users are generally directed to paid support for in-depth issues.
    - **Third-Party Support**: Many hardware and software companies provide specific support for Windows due to its large market share.
- **Ubuntu**:
    - **Community Support**: Ubuntu has a large and active community that provides extensive support through forums, online documentation, and discussion groups. If you face any issue, you can often find a solution through the community.
    - **Commercial Support**: Canonical Ltd. Offers paid support services for businesses using Ubuntu in production environments.

8. Customization

- **Windows OS**:
    - **Limited Customization**: Windows allows some level of customization (themes, taskbar, desktop settings), but it is relatively limited compared to open-source platforms like Linux.

- **Registry Tweaks**: Advanced users can tweak Windows via the registry, but this can be risky and complex.

- **Ubuntu**:
  - **Highly Customizable**: Ubuntu offers extensive customization options. You can switch desktop environments (GNOME, KDE, XFCE), change themes, icons, and even alter system behavior through configuration files.

Table Of Comparision Between UBUNTU and WINDOWS

Feature	Windows	Ubuntu
Operating System	Proprietary, closed-source operating system developed by Microsoft	An open-source operating system based on the Linux kernel.
User Interface	GUI (Graphical User Interface) with a familiar Windows desktop experience	GUI with various desktop environments (e.g., GNOME, Unity)
Software	Large selection of commercial and proprietary software available	Vast collection of free and open-source software available
Compatibility	Widely supported by software developers and hardware manufacturers	Growing support, but some software and hardware may not be compatible
Updates	Regular updates and patches released by Microsoft	Frequent updates and patches through the Ubuntu package manager
Customization	Limited customization options for appearance and functionality	High level of customization through various themes and extensions
Gaming	Extensive support for mainstream games and popular gaming platforms	Growing support, with some popular games and platforms available
System Resources	Typically requires more system resources (RAM, CPU)	Generally lighter and more efficient, suitable for older hardware
Support	Extensive documentation, user forums, and paid technical support options	Active community support, forums, and official documentation
Cost	Commercial licenses with different editions and price points	Free to use and distribute, although commercial support is available

…………………………………………………………….

<u>LAB ASSIGNMENT -2</u>

1. OUTPUTS OF LINUX COMMANDS

Here's a detailed theoretical explanation of each command, including the full command syntax and usage examples.

1. pwd (Print Working Directory)

- **Command**: pwd
- **Explanation**: The pwd command stands for "Print Working Directory." It displays the full path of the current working directory you are in. This is particularly useful to confirm the location within the filesystem.
- **Full Command**: pwd
- **Example Output**:
- /home/user/Documents

2. cd (Change Directory)

- **Command**: cd /path/to/directory
- **Explanation**: The cd command is used to change the current working directory. You provide the directory path as an argument, and the terminal moves to that directory.
- **Full Command**: cd /home/user/Downloads
- **Example Output**:
- (No output, but the current working directory is now `/home/user/Downloads`.)

3. ls (List)

- **Command**: ls
- **Explanation**: The ls command lists the files and directories in the current directory. It can be combined with options like -l for long format, -a to show hidden files, and -h for human-readable sizes.
- **Full Command**:
- ls
- **Example Output**:
- file1.txt file2.txt directory1 directory2

4. mkdir (Make Directory)

- **Command**: mkdir new_directory

- **Explanation**: The mkdir command creates a new directory with the specified name. You can create multiple directories at once by providing multiple names or use the -p flag to create parent directories as needed.
- **Full Command**:
- mkdir new_directory
- **Example Output**:
- (No output, but a new directory `new_directory` is created.)

5. rm (Remove)

- **Command**: rm file_name
- **Explanation**: The rm command is used to remove files or directories. To delete a directory, you can use rm -r directory_name. Be cautious when using rm as it permanently deletes files without asking for confirmation (unless -i is used).
- **Full Command**:
- rm file1.txt
- **Example Output**:
- (No output, but the file `file1.txt` is deleted.)

6. touch (Create an Empty File or Update Timestamps)

- **Command**: touch newfile.txt
- **Explanation**: The touch command is used to create an empty file if the specified file does not exist, or it updates the file's access and modification times if the file already exists.
- **Full Command**:
- touch newfile.txt
- **Example Output**:
- (No output, but the file `newfile.txt` is created if it doesn't exist.)

7. hostname

- **Command**: hostname
- **Explanation**: The hostname command displays the name of the current system, which is typically the identifier of the machine on a network.
- **Full Command**:
- hostname
- **Example Output**:
- Mycomputer

8. cat (Concatenate and Display File Contents)

- **Command**: cat file_name
- **Explanation**: The cat command is used to display the contents of a file or concatenate multiple files into a single output. It's often used to quickly view small text files.
- **Full Command**:
- cat file1.txt
- **Example Output**:
- This is the content of file1.txt.

9. chmod (Change File Permissions)

- **Command**: chmod mode file_name
- **Explanation**: The chmod command is used to change the file permissions (read, write, execute) for the owner, group, and others. The mode can be specified either in numeric or symbolic format.
  - Numeric format: r=4, w=2, x=1 (combined as a 3-digit number)
  - Symbolic format: r for read, w for write, and x for execute
- **Full Command**:
- chmod 755 file1.txt
- **Example Output**:
- (No output, but the file permissions are changed.)

10. echo (Print Text to Terminal)

- **Command**: echo "text"
- **Explanation**: The echo command prints the specified text to the terminal. It is often used in scripts or to display messages or variables.
- **Full Command**:
- echo "Hello, World!"
- **Example Output**:
- Hello, World!

11. grep (Search Text in Files)

- **Command**: grep "pattern" file_name
- **Explanation**: The grep command searches for a specified pattern in a file and prints the matching lines. It supports regular expressions and various options, such as -i for case-insensitive search.
- **Full Command**:
- grep "hello" file1.txt

- **Example Output**:
- hello there

12. fgrep (Fixed-String Search)

- **Command**: fgrep "string" file_name
- **Explanation**: Similar to grep, the fgrep command searches for fixed strings without interpreting special characters or regular expressions.
- **Full Command**:
- fgrep "hello" file1.txt
- **Example Output**:
- hello there

13. mv (Move or Rename Files)

- **Command**: mv source_file target_file
- **Explanation**: The mv command is used to move files or directories from one location to another or to rename a file or directory. If the destination is a directory, the file is moved; if it's a file, the source is renamed.
- **Full Command**:
- mv file1.txt file2.txt
- **Example Output**:
- (No output, but `file1.txt` is renamed to `file2.txt`.)

14. cp (Copy Files)

- **Command**: cp source_file target_file
- **Explanation**: The cp command is used to copy a file or directory from one location to another. You can use the -r flag to copy directories recursively.
- **Full Command**:
- cp file1.txt file2.txt
- **Example Output**:
- (No output, but `file1.txt` is copied to `file2.txt`.)

15. more (View File Contents Page by Page)

- **Command**: more file_name
- **Explanation**: The more command displays the contents of a file, one screen at a time. It allows you to scroll down through a file using the spacebar.
- **Full Command**:
- more file1.txt

- **Example Output**:
- This is the content of file1.txt.
- -- More –

16. less (View File Contents with Better Navigation)

- **Command**: less file_name
- **Explanation**: The less command is similar to more but offers more features, such as backward scrolling and better navigation. It’s more versatile and user-friendly.
- **Full Command**:
- less file1.txt
- **Example Output**:
- This is the content of file1.txt.
- (Press space to scroll down)

17. wc (Word Count)

- **Command**: wc file_name
- **Explanation**: The wc command is used to count the number of lines, words, and characters in a file. It has options like -l for lines, -w for words, and -c for characters.
- **Full Command**:
- wc file1.txt
- **Example Output**:
- 10 25 150 file1.txt

18. awk (Pattern Scanning and Processing Language)

- **Command**: awk 'pattern {action}' file_name
- **Explanation**: awk is a powerful text processing tool that allows you to search for patterns in files and perform actions on them. It is commonly used to extract specific columns or process structured text files.
- **Full Command**:
- awk '{print $1}' file1.txt
- **Example Output**:
- First_column_value

19. sed (Stream Editor)

- **Command**: sed 's/old/new/g' file_name

- **Explanation**: The sed command is used for stream editing, such as replacing text in a file. The example command replaces all occurrences of the string "old" with "new".
- **Full Command**:
- sed 's/old/new/g' file1.txt
- **Example Output**:
- (No output, but the file's content is modified.)

20. tail (View End of a File)

- **Command**: tail file_name
- **Explanation**: The tail command displays the last 10 lines of a file. It's useful for viewing log files or large files where you only care about the most recent data. Use -f to continuously monitor the file as it changes.
- **Full Command**:
- tail file1.txt
- **Example Output**:
- Last line of the file
- Another last line

These commands are helpful in managing files, viewing content, and performing various operations on a Linux or Unix-like system.

....

2.QUESTIONS BASED ON COMMANDS

The detailed answers with commands and explanations for each of your questions:

---

1. How to navigate to a Specific Directory?

- **Command**: cd /path/to/directory
- **Explanation**: The cd (change directory) command is used to navigate to a specific directory. You need to specify the full path or relative path to the directory you want to access.
- **Example Command**:
- cd /home/user/Documents
  This command changes the current directory to /home/user/Documents.

---

2. How to see detailed information about files and directories using ls?

- **Command**: ls -l
- **Explanation**: The ls command lists the files and directories, but when used with the -l option (long listing format), it displays detailed information such as file permissions, owner, size, and last modification date.
- **Example Command**:
- ls -l
  Output Explanation:
  -rwxr-xr-x 1 user group 1234 Mar 1 12:34 file.txt
  This shows file permissions, number of links, owner, group, size, modification time, and file name.

---

3. How to create multiple directories in Linux using mkdir command?

- **Command**: mkdir dir1 dir2 dir3
- **Explanation**: The mkdir (make directory) command allows you to create multiple directories at once by specifying them in a space-separated list.
- **Example Command**:
- mkdir dir1 dir2 dir3
  This command will create three directories: dir1, dir2, and dir3.

---

4. How to remove multiple files at once with rm?

- **Command**: rm file1.txt file2.txt file3.txt
- **Explanation**: The rm (remove) command can be used to delete multiple files at once by listing them in a space-separated format.
- **Example Command**:
- rm file1.txt file2.txt file3.txt

This command will remove file1.txt, file2.txt, and file3.txt from the directory.

---

5. Can rm be used to delete directories?

- **Command**: rm -r directory_name
- **Explanation**: Yes, rm can delete directories, but you must use the -r (recursive) option to delete a directory and its contents.
- **Example Command**:
- rm -r dir1
  This deletes the dir1 directory and all files or subdirectories it contains.

---

6. How Do You Copy Files and Directories in Linux?

- **Command**: cp source_file target_file
- **Explanation**: The cp command is used to copy files. To copy directories, use the -r (recursive) option.
- **Example Command**:
- cp file1.txt file2.txt # Copy a file
- cp -r dir1 dir2   # Copy a directory
  The first command copies file1.txt to file2.txt, while the second command copies the dir1 directory to dir2.

---

7. How to Rename a file in Linux Using mv Command

- **Command**: mv old_file_name new_file_name
- **Explanation**: The mv command is used for both moving files and renaming them. To rename a file, simply specify the old file name and the new file name.
- **Example Command**:
- mv file1.txt file2.txt
  This renames file1.txt to file2.txt.

---

8. How to Move Multiple files in Linux Using mv Command

- **Command**: mv file1.txt file2.txt target_directory
- **Explanation**: The mv command can move multiple files to a different directory by specifying the files and the target directory.
- **Example Command**:
- mv file1.txt file2.txt dir1
  This command moves both file1.txt and file2.txt to the dir1 directory.

---

9. How to Create Multiple Empty Files by Using Touch Command in Linux

- **Command**: touch file1.txt file2.txt file3.txt
- **Explanation**: The touch command is used to create empty files. You can specify multiple files in the same command.

- **Example Command**:
- touch file1.txt file2.txt file3.txt

This command creates three empty files: file1.txt, file2.txt, and file3.txt.

---

10. How to View the Content of Multiple Files in Linux

- **Command**: cat file1.txt file2.txt
- **Explanation**: The cat command is used to view the contents of files. To view multiple files, list them after the command.
- **Example Command**:
- cat file1.txt file2.txt
  This will display the content of both file1.txt and file2.txt sequentially.

---

11. How to Create a file and add content in Linux Using cat Command

- **Command**: cat > file_name
- **Explanation**: The cat command can also be used to create a file and add content to it. After running the command, type the content and press Ctrl+D to save the file.
- **Example Command**:
- cat > file1.txt
  Type the content you want, then press Ctrl+D to save it to file1.txt.

---

12. How to Append the Contents of One File to the End of Another File using cat command

- **Command**: cat file1.txt >> file2.txt
- **Explanation**: The >> operator appends the contents of file1.txt to the end of file2.txt.
- **Example Command**:
- cat file1.txt >> file2.txt
  This appends the content of file1.txt to file2.txt.

---

13. How to use cat command if the file has a lot of content and can’t fit in the terminal

- **Command**: cat file_name | less
- **Explanation**: Use the less command in combination with cat to view files with large content that doesn't fit on the screen.
- **Example Command**:
- cat file1.txt | less
  This allows you to scroll through the content of file1.txt using the less command.

---

14. How to Merge Contents of Multiple Files Using cat Command

- **Command**: cat file1.txt file2.txt > merged_file.txt
- **Explanation**: You can merge the contents of multiple files into one file using the cat command and the > operator to redirect the output to a new file.
- **Example Command**:
- cat file1.txt file2.txt > merged_file.txt
  This merges the content of file1.txt and file2.txt into a new file merged_file.txt.

---

15. How to use cat Command to Append to an Existing File

- **Command**: cat file1.txt >> file2.txt
- **Explanation**: The >> operator appends the content of file1.txt to the end of file2.txt.
- **Example Command**:
- cat file1.txt >> file2.txt

---

16. What is "chmod 777", "chmod 755" and "chmod +x" or "chmod a+x"?

- **Command**: chmod 777 file_name, chmod 755 file_name, chmod +x file_name
- **Explanation**:
  - chmod 777: Grants full permissions (read, write, execute) to the owner, group, and others.
  - chmod 755: Grants read, write, and execute permissions to the owner, and read and execute permissions to the group and others.
  - chmod +x: Adds execute permission to the file for the user, group, or others.
- **Example Command**:
- chmod 777 file1.txt
- chmod 755 script.sh
- chmod +x script.sh

---

17. How to find the number of lines that matches the given string/pattern

- **Command**: grep -c "pattern" file_name
- **Explanation**: The -c option in grep counts the number of lines that match the given pattern.
- **Example Command**:
- grep -c "hello" file1.txt
  This will return the number of lines in file1.txt containing the string "hello."

---

18. How to display the files that contain the given string/pattern

- **Command**: grep -l "pattern" *
- **Explanation**: The -l (lowercase L) option shows only the names of files that contain the specified pattern.
- **Example Command**:
- grep -l "hello" *
  This lists the files that contain the string "hello".

---

19. How to show the line number of file with the line matched

- **Command**: grep -n "pattern" file_name
- **Explanation**: The -n option in grep shows the line number along with the matching line.
- **Example Command**:
- grep -n "hello" file1.txt
  This displays the line number along with each matching line in file1.txt.

---

20. How to match the lines that start with a string using grep

- **Command**: grep "^pattern" file_name
- **Explanation**: The ^ symbol is used in grep to match lines that start with a specified pattern.
- **Example Command**:
- grep "^hello" file1.txt
  This matches lines that start with the string "hello".

---

21. Can the 'sort' command be used to sort files in descending order by default?

- **Answer**: No, the sort command sorts files in ascending order by default. To sort in descending order, use the -r option.

---

22. How can I sort a file based on a specific column using the 'sort' command?

- **Command**: sort -k column_number file_name
- **Explanation**: The -k option is used with sort to specify the column on which the sorting should be performed.
- **Example Command**:
- sort -k 2 file1.txt
  This sorts file1.txt based on the second column.

……………………………

LAB ASSIGNMENT -3
SHELL SCRIPTING

Here's the solution of the shell scripts with each question.

---

1. To find Largest of Three Numbers:
Solution:
```
#!/bin/bash
echo "Enter three numbers: "
read num1 num2 num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]; then
echo "$num1 is the largest."
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]; then
echo "$num2 is the largest."
else
echo "$num3 is the largest."
fi
```

---

2. To find a year is leap year or not:
Solution:
```
#!/bin/bash
echo "Enter a year: "
read year
if (( $year % 400 == 0 )) || (( $year % 4 == 0 && $year % 100
!= 0 )); then
echo "$year is a leap year."
else
echo "$year is not a leap year."
fi
```

---

3. To input angles of a triangle and find out whether it is valid triangle or not:
Solution:
```
#!/bin/bash
echo "Enter three angles of a triangle: "
read angle1 angle2 angle3
sum=$((angle1 + angle2 + angle3))
if [ $sum -eq 180 ]; then
echo "It is a valid triangle."
else
echo "It is not a valid triangle."
fi
```

---

4. To check whether a character is alphabet, digit or special character:
Solution:

```
#!/bin/bash
echo "Enter a character: "
read char
if [[ $char =~ [a-zA-Z] ]]; then
echo "$char is an alphabet."
elif [[ $char =~ [0-9] ]]; then
echo "$char is a digit."
else
echo "$char is a special character."
fi
```

---

5. <u>To calculate profit or loss:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Enter cost price: "
read cost_price
echo "Enter selling price: "
read selling_price
if (( selling_price > cost_price )); then
profit=$((selling_price - cost_price))
echo "Profit: $profit"
elif (( selling_price < cost_price )); then
loss=$((cost_price - selling_price))
echo "Loss: $loss"
else
echo "No profit, no loss."
fi
```

---

6. <u>To print all even and odd numbers from 1 to 10:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Even numbers from 1 to 10:"
for i in {1..10}; do
if (( i % 2 == 0 )); then
echo $i
fi
done
echo "Odd numbers from 1 to 10:"
for i in {1..10}; do
if (( i % 2 != 0 )); then
echo $i
fi
done
```

---

7. <u>To print table of a given number:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Enter a number: "
```

```
read num
echo "Table of $num:"
for i in {1..10}; do
result=$((num * i))
echo "$num * $i = $result"
done
```

---

8. <u>To find factorial of a given integer:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Enter a number: "
read num
fact=1
for (( i=1; i<=num; i++ )); do
fact=$((fact * i))
done
echo "Factorial of $num is $fact"
```

---

9. <u>To print sum of all even numbers from 1 to 10:</u>
<u>Solution:</u>

```
#!/bin/bash
sum=0
for i in {1..10}; do
if (( i % 2 == 0 )); then
sum=$((sum + i))
fi
done
echo "Sum of even numbers from 1 to 10 is $sum"
```

---

10. <u>To print sum of digit of any number:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Enter a number: "
read num
sum=0
while [ $num -gt 0 ]; do
digit=$((num % 10))
sum=$((sum + digit))
num=$((num / 10))
done
echo "Sum of digits is $sum"
```

---

11. <u>To make a basic calculator which performs addition, subtraction, multiplication, division:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Enter first number: "
```

```
read num1
echo "Enter second number: "
read num2
echo "Choose operation (+, -, *, /): "
read op
case $op in
+) result=$((num1 + num2)) ;;
-) result=$((num1 - num2)) ;;
\*) result=$((num1 * num2)) ;;
/) result=$((num1 / num2)) ;;
*) echo "Invalid operation" ;;
esac
echo "Result: $result"
```

---

12. <u>To print days of a week:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "Enter a number (1-7): "
read day
case $day in
1) echo "Sunday" ;;
2) echo "Monday" ;;
3) echo "Tuesday" ;;
4) echo "Wednesday" ;;
5) echo "Thursday" ;;
6) echo "Friday" ;;
7) echo "Saturday" ;;
*) echo "Invalid input" ;;
esac
```

---

13. <u>To print starting 4 months having 31 days:</u>
<u>Solution:</u>

```
#!/bin/bash
echo "The first 4 months with 31 days are:"
echo "January, March, May, July"
```

---

14. <u>Using functions:</u>
<u>a. To find given number is Armstrong number or not:</u>
<u>Solution:</u>

```
#!/bin/bash
armstrong_number() {
num=$1
sum=0
temp=$num
while [ $num -gt 0 ]; do
digit=$((num % 10))
sum=$((sum + digit * digit * digit))
num=$((num / 10))
```

```
done
if [ $sum -eq $temp ]; then
echo "$temp is an Armstrong number."
else
echo "$temp is not an Armstrong number."
fi
echo "Enter a number: "
read num
armstrong_number $num
```

---

b. <u>To find whether a number is palindrome or not:</u>
<u>Solution:</u>

```
#!/bin/bash
palindrome() {
num=$1
rev=0
temp=$num
while [ $num -gt 0 ]; do
digit=$((num % 10))
rev=$((rev * 10 + digit))
num=$((num / 10))
done
if [ $rev -eq $temp ]; then
echo "$temp is a palindrome."
else
echo "$temp is not a palindrome."
fi
}
echo "Enter a number: "
read num
palindrome $num
```

---

c. <u>To print Fibonacci series upto n terms:</u>
<u>Solution:</u>

```
#!/bin/bash
fibonacci() {
n=$1
a=0
b=1
echo "Fibonacci series:"
for (( i=0; i<n; i++ )); do
echo -n "$a "
fn=$((a + b))
a=$b
b=$fn
done
echo
}
```

```
echo "Enter the number of terms: "
read n
fibonacci $n
```

---

d. <u>To find given number is prime or composite:</u>
<u>Solution</u>:

```
#!/bin/bash
prime_or_composite() {
num=$1
if [ $num -le 1 ]; then
echo "$num is neither prime nor composite."
return
fi
for (( i=2; i*i<=num; i++ )); do
if [ $((num % i)) -eq 0 ]; then
echo "$num is a composite number."
return
fi
done
echo "$num is a prime number."
}
echo "Enter a number: "
read num
prime_or_composite $num
```

---

e. <u>To convert a given decimal number to binary equivalent:</u>
<u>Solution:</u>

```
#!/bin/bash
decimal_to_binary() {
num=$1
binary=""
while [ $num -gt 0 ]; do
binary=$((num % 2))$binary
num=$((num / 2))
done
echo "Binary equivalent: $binary"
}
echo "Enter a decimal number: "
read num
decimal_to_binary $num
```

…………………………………

LAB ASSIGNMENT -4

STUDY OF SYSTEM CALLS

Comprehensive Study of Different Categories of Linux System Calls
Linux system calls provide an interface for user-space applications to interact with the kernel. System calls are categorized into various types based on their functionality. Below is a detailed exploration of different categories of Linux system calls, their purpose, and example usage for each.

---

1. Process Management System Calls

Process management system calls are responsible for the creation, scheduling, and termination of processes in the system. These system calls allow for process control in a multitasking environment.

- **fork()**
  - **Purpose**: Creates a new process by duplicating the calling process.
  - **Behavior**: The child process created by `fork()` is identical to the parent process, except for the returned value. In the parent process, **`fork()`** returns the child process' PID, while in the child process, `fork()` returns 0.
  - **Example**:

```
#include <stdio.h>
#include <unistd.h>
int main() {
pid_t pid = fork();
if (pid == 0) {
printf("This is the child process\n");
} else {
printf("This is the parent process with child PID:
%d\n", pid);
}
return 0;
}
```

- **exec()**
  - **Purpose**: Replaces the current process image with a new one, i.e., runs a different program.
  - **Behavior**: It does not return if successful, as the calling process is replaced with a new program. It can be used with variants like `execv()`,`execp()`,`execl()`, etc.
  - **Example**:

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

```
int main() {
    printf("Before exec() call\n");
    execlp("/bin/ls", "ls", "-l", NULL);    //
Executes the 'ls' command
    printf("This will not be printed\n");  // This
won't be printed
    return 0;
}
```

- **wait()**
  - **Purpose**: Pauses the execution of the calling process until one of its child processes finishes execution.
  - **Behavior**: It returns the process ID of the terminated child process.
  - **Example**:

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
    pid_t pid = fork();
    if (pid == 0) {
        // Child process
        printf("Child process\n");
    } else {
        // Parent process
        wait(NULL);  // Wait for child process to
terminate
        printf("Parent process after child
terminates\n");
    }
    return 0;
}
```

- **exit()**
  - **Purpose**: Terminates the calling process and returns an exit status to the parent process.
  - **Behavior**: It terminates the calling process immediately and returns the provided exit status.
  - **Example**:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("Process is exiting\n");
    exit(0);   // Normal termination with exit
status 0
}
```

2. File Management System Calls

File management system calls provide an interface for applications to perform operations on files such as opening, reading, writing, and closing files.

- **open()**
  - **Purpose**: Opens a file descriptor for reading, writing, or both.
  - **Behavior**: It returns a non-negative integer (the file descriptor) on success or -1 on failure.
  - **Example**:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main() {
    int fd = open("example.txt", O_RDWR | O_CREAT,
S_IRUSR | S_IWUSR);
    if (fd == -1) {
        perror("Failed to open file");
    } else {
        printf("File opened successfully with file
descriptor: %d\n", fd);
        close(fd);  // Always close after using
    }
    return 0;
}
```

- **read()**
  - **Purpose**: Reads data from a file descriptor into a buffer.
  - **Behavior**: It returns the number of bytes read, or -1 on error.
  - **Example**:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
    int fd = open("example.txt", O_RDONLY);
    char buffer[128];
    if (fd != -1) {
        ssize_t  bytes_read  =  read(fd,  buffer,
sizeof(buffer));
        if (bytes_read > 0) {
            write(STDOUT_FILENO,          buffer,
bytes_read);  // Print to stdout
        }
        close(fd);
    }
return 0;
```

- **write()**
  - **Purpose**: Writes data from a buffer to a file descriptor.
  - **Behavior**: It returns the number of bytes written, or -1 on error.
  - **Example**:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
    int  fd  =  open("example.txt",  O_WRONLY  |
O_CREAT, S_IRUSR | S_IWUSR);
    if (fd != -1) {
        const char *msg = "Hello, world!\n";
        write(fd, msg, strlen(msg));
        close(fd);
    }
    return 0;
}
```

- **close()**
  - **Purpose**: Closes an open file descriptor.
  - **Behavior**: It returns 0 on success, or -1 on failure.
  - **Example**:

```
#include <unistd.h>
int main() {
    int fd = open("example.txt", O_RDONLY);
    if (fd != -1) {
        close(fd);  // Close the file descriptor
    }
    return 0;
}
```

---

3. Device Management System Calls

Device management system calls enable interaction with hardware devices (e.g., reading from or writing to a device).

- **read()**
  - Same as in File Management, `read()` can be used for device interaction when the device is represented as a file (e.g., `/dev/tty`).
- **write()**
  - Same as in File Management, `write()` is used to send data to devices like printers or serial ports.
- **ioctl()**
  - **Purpose**: Controls device parameters or configuration by passing commands to the kernel.

- **Behavior**: It is often used to set device options or interact with device drivers.
- **Example**:

```
#include <stdio.h>
#include <sys/ioctl.h>
#include <unistd.h>
int main() {
    int fd = open("/dev/tty", O_RDWR);
    if (fd != -1) {
        int result = ioctl(fd, TIOCGWINSZ, NULL);
// Example for terminal window size
        close(fd);
    }
    return 0;
}
```

- **select()**
  - **Purpose**: Monitors multiple file descriptors to see if they are ready for reading, writing, or have an exception.
  - **Example**:

```
#include <stdio.h>
#include <sys/select.h>
#include <unistd.h>
int main() {
    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(STDIN_FILENO, &read_fds);
    struct timeval timeout = { 5, 0 };  // Timeout
of 5 seconds
    int activity = select(STDIN_FILENO + 1,
&read_fds, NULL, NULL, &timeout);
    if (activity > 0) {
        printf("Input available\n");
    }
    return 0;
}
```

---

4. Network Management System Calls

Network management system calls provide mechanisms for network communication.

- **socket()**
  - **Purpose**: Creates a new communication endpoint for network communication.

- **Example**:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
int main() {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
// Create a TCP socket
    if (sockfd == -1) {
        perror("Socket creation failed");
    }
    return 0;
}
```

- **connect()**
  - **Purpose**: Establishes a connection to a remote server.
  - **Example**:

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    connect(sockfd,                    (struct
sockaddr*)&server_addr, sizeof(server_addr));
    printf("Connected to server\n");
    return 0;
}
```

- **send()**
  - **Purpose**: Sends data over a socket.
  - **Example**:

```
#include <stdio.h>
#include <sys/socket.h>
int main() {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    char msg[] = "Hello, Server!";
    send(sockfd, msg, sizeof(msg), 0);   // Send
message to server
    return 0;
}
```

- **recv()**
  - **Purpose**: Receives data from a socket.

- **Example**:

```
#include <stdio.h>
#include <sys/socket.h>
int main() {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    char buffer[128];
    recv(sockfd, buffer, sizeof(buffer), 0);   //
Receive message from server
    printf("Received message: %s\n", buffer);
    return 0;
}
```

---

5. <u>System Information Management System Calls</u>

System information management system calls provide functions for retrieving information about the system and processes.

- **getpid()**
  - **Purpose**: Returns the process ID (PID) of the calling process.
  - **Example**:

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Process ID: %d\n", getpid());
    return 0;
}
```

- **getuid()**
  - **Purpose**: Returns the real user ID of the calling process.
  - **Example**:

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("User ID: %d\n", getuid());
    return 0;
}
```

- **gethostname()**
  - **Purpose**: Returns the hostname of the system.
  - **Example**:

```
#include <stdio.h>
#include <unistd.h>
int main() {
    char hostname[128];
    gethostname(hostname, sizeof(hostname));
printf("Hostname: %s\n", hostname);
    return 0;
}
```

- **sysinfo()**
  - **Purpose**: Provides information about the system's memory, load averages, and other statistics.
  - **Example**:

```
#include <stdio.h>
#include <sys/sysinfo.h>
int main() {
    struct sysinfo info;
    sysinfo(&info);
    printf("Total RAM: %ld KB\n", info.totalram /
1024);
    return 0;
  }
```

....

LAB ASSIGNMENT -5
STUDY OF SYSTEM CALLS

Here are the C programs for the three common CPU scheduling algorithms: **First Come First Serve (FCFS)**, **Shortest Job First (SJF)**, and **Round Robin Scheduling**.

1. First Come First Serve (FCFS) Scheduling Algorithm

The **FCFS** algorithm schedules processes based on their arrival order.

C++ Code for FCFS Scheduling:

```
#include <iostream>
#include <vector>
using namespace std;

void findWaitingTime(vector<int>& processes, int n, vector<int>& bt,
vector<int>& wt) {
    wt[0] = 0; // Waiting time for the first process is always 0
    for (int i = 1; i < n; i++) {
        wt[i] = bt[i - 1] + wt[i - 1];
    }
}
void findTurnAroundTime(vector<int>& processes, int n, vector<int>& bt,
vector<int>& wt, vector<int>& tat) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}

void findAverageTime(vector<int>& processes, int n, vector<int>& bt) {
    vector<int> wt(n), tat(n);
    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);
    int total_wt = 0, total_tat = 0;
    cout << "Process\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        cout << processes[i] << "\t\t" << bt[i] << "\t\t" << wt[i] << "\t\t" << tat[i] <<
endl;
    }
    cout << "\nAverage Waiting Time: " << (float)total_wt / n << endl;
    cout << "Average Turnaround Time: " << (float)total_tat / n << endl;
}
int main() {
    vector<int> processes = {1, 2, 3};
    int n = processes.size();
    vector<int> bt = {5, 3, 8};  // Burst Times of the processes

    findAverageTime(processes, n, bt);
    return 0;
}
```

Example Output for FCFS Scheduling:

Process	Burst Time	Waiting Time	Turnaround Time
1	5	0	5
2	3	5	8
3	8	8	16

```
Average Waiting Time: 4.33
Average Turnaround Time: 9.67
```

---

2. Shortest Job First (SJF) Scheduling Algorithm

The **SJF** algorithm schedules the process with the shortest burst time first. It assumes that all processes are available at time 0.

C++ Code for SJF Scheduling:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void findWaitingTime(vector<int>& processes, int n, vector<int>& bt,
vector<int>& wt) {
    wt[0] = 0;  // Waiting time for the first process is always 0
    for (int i = 1; i < n; i++) {
        wt[i] = bt[i - 1] + wt[i - 1];
    }
}
void findTurnAroundTime(vector<int>& processes, int n, vector<int>& bt,
vector<int>& wt, vector<int>& tat) {
    for (int i = 0; i < n; i++) {
      tat[i] = bt[i] + wt[i];
    }}
void findAverageTime(vector<int>& processes, int n, vector<int>& bt) {
    vector<int> wt(n), tat(n);
    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);

    int total_wt = 0, total_tat = 0;
    cout << "Process\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
  cout << processes[i] << "\t\t" << bt[i] << "\t\t" << wt[i] << "\t\t" << tat[i] << endl;
    }
    cout << "\nAverage Waiting Time: " << (float)total_wt / n << endl;
    cout << "Average Turnaround Time: " << (float)total_tat / n << endl;
}
void sjf(vector<int>& processes, int n, vector<int>& bt) {
    // Sort the burst time in ascending order
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (bt[i] > bt[j]) {
                swap(bt[i], bt[j]);
                swap(processes[i], processes[j]);
            } }    }
    findAverageTime(processes, n, bt);
}
int main() {
    vector<int> processes = {1, 2, 3};
    int n = processes.size();
    vector<int> bt = {6, 8, 7};  // Burst Times of the processes
  sjf(processes, n, bt);
    return 0;
}
```

Example Output for SJF Scheduling:

Process	Burst Time	Waiting Time	Turnaround Time
1	6	0	6
3	7	6	13
2	8	13	21

```
Average Waiting Time: 6.33
Average Turnaround Time: 13.33
```

3. Round Robin (RR) Scheduling Algorithm

In **Round Robin** scheduling, each process is given a fixed time slice or quantum to execute. Once a process's quantum expires, it is moved to the back of the queue.

C++ Code for Round Robin Scheduling:

```
#include <iostream>
#include <vector>
using namespace std;

void findWaitingTime(vector<int>& processes, int n, vector<int>& bt,
vector<int>& wt, int quantum) {
    vector<int> rem_bt(bt);
    int t = 0;  // Time counter
    while (true) {
        bool done = true;
        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                done = false;
                if (rem_bt[i] > quantum) {
                    t += quantum;
                    rem_bt[i] -= quantum;
                } else {
                    t += rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
        if (done) break;
    }
}
void findTurnAroundTime(vector<int>& processes, int n, vector<int>& bt,
vector<int>& wt, vector<int>& tat) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}
void findAverageTime(vector<int>& processes, int n, vector<int>& bt, int
quantum) {
    vector<int> wt(n), tat(n);
    findWaitingTime(processes, n, bt, wt, quantum);
    findTurnAroundTime(processes, n, bt, wt, tat);
    int total_wt = 0, total_tat = 0;
    cout << "Process\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
 cout << processes[i] << "\t\t" << bt[i] << "\t\t" << wt[i] << "\t\t" << tat[i] << endl;
    }
    cout << "\nAverage Waiting Time: " << (float)total_wt / n << endl;
    cout << "Average Turnaround Time: " << (float)total_tat / n << endl;
}
int main() {
    vector<int> processes = {1, 2, 3};
    int n = processes.size();
    vector<int> bt = {10, 5, 8};  // Burst Times of the processes
    int quantum = 4;  // Time Quantum
    findAverageTime(processes, n, bt, quantum);
    return 0;
}
```

Example Output for Round Robin Scheduling:

Process	Burst Time	Waiting Time	Turnaround Time
1	10	4	14
2	5	6	11
3	8	6	14

```
Average Waiting Time: 5.33
Average Turnaround Time: 13.00
```

These programs demonstrate the implementation and output for each of the scheduling algorithms. You can compile and run these C++ programs using any C++ compiler to observe the scheduling and time calculations for each algorithm.

……………….END……………….