SUBJECT NAME: Unix and Shell Programming

SUBJECT CODE: BCAC602

Modile-M1

Introduction to Unix; Discuss about POSIX; Discuss about Linux and most popular distributions of Linux; Compare Between the Unix and Linux; Unix system Architecture; Discuss about the Unix kernel and system call interface; Unix directory structure.

MULTICS:

Multics (**Multi**plexed **I**nformation and **C**omputing **S**ervice) was a mainframe timesharing operating system that began at **MIT(Massachusetts Institute of Technology)** as a research project in 1965. It was an important influence on operating system development.

The **history of Unix** dates back to the mid-1960s when the Massachusetts Institute of
Technology, **AT&T Bell Labs**, and General Electric were jointly developing an experimental time sharing operating system called Multics for the GE-645 mainframe. **Multics introduced many innovations, but had many problems.**

What is UNIX?

Unix was originally spelt “Unics”. **UNICS** stands for **UNiplexed Information and Computing System,** is a popular operating system developed at Bell Labs in the early 1970s. The name was intended as a pun on an earlier system called “**Multics**” (**Multiplexed Information and Computing Service)**.

UNIX stands for **Uniplexed Information and Computing Service**, which was originally spelled “**Unics”. UNICS** stands **for Uniplexed Information and Computing System**.

The UNIX operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

- UNIX was originally **developed in 1969** by a group of **AT&T** employees **Ken Thompson, Dennis Ritchie, Douglas McIlroy**, and **Joe Ossanna at Bell Labs.**

- There are various Unix variants available in the market. **Solaris UNIX, AIX, HP UNIX and BSD** are a few examples. **Linux** is also a **flavor of UNIX** which is **freely** available.

- A "flavor of Unix" refers to any operating system based on the original AT&T Unix, differing in design, features, hardware, and licensing (proprietary vs. free); popular examples include **Linux**, **macOS**, **AIX** (IBM), **HP-UX** (HP), and **Solaris** (Oracle), all sharing core Unix concepts like multi-user/tasking, hierarchical files, and powerful command-line interfaces.

- Several people can use a Unix computer at the same time; hence Unix is called a **multiuser system.**

- A user can **also run multiple programs** at the same time; hence UNIX is a **multitasking** environment.

AIX is an abbreviation of "**Advanced Interactive EXecutive**". It is a progression sequence of proprietary **UNIX** operating systems designed, created and sold by IBM for a number of its computer platforms.

The **Berkeley Software Distribution** (**BSD**) was an operating system based on **Research Unix**, developed and distributed by the **Computer Systems Research Group (CSRG**) at the University of California, Berkeley. Today, "**BSD**" often refers to its descendants, such as FreeBSD, OpenBSD, **NetBSD(Network Stack based BSD).**

NetBSD(Network Stack based BSD) is a free, fast, secure, and highly portable Unix-like Open Source operating system. It is available for a wide range of platforms, from large-scale servers and powerful desktop systems to handheld and embedded devices.

The **Portable Operating System Interface** (**POSIX**) is a family of standards specified by the IEEE Computer Society for **maintaining compatibility** between operating **systems**.

UNIX Initial release date: 3 November 1971

Developer: Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe
Ossanna at Bell Labs

License: Varies; some versions are **proprietary**(used to describe a product that is made and sold by a particular company whose name, or a name that it owns, is on the product), others are free/open-source software.

Written in: C, Assembly language

Note:
Linux began in **1991** as a personal project by Finnish (**Republic of Finland**) **student**

Linus Torvalds: to create a **new** free operating system kernel.

Discuss about Linux and most popular distributions of Linux:

Linux is an open-source operating system kernel, but a "distribution" (or "distro") bundles it with essential software, creating a complete OS for specific needs, with popular choices like **Ubuntu** (user-friendly, beginners), **Linux Mint** (Windows-like, easy transition), **Debian** (stable, foundational), **Fedora** (cutting-edge, Red Hat backed), **Arch Linux** (DIY, highly customizable), and **Kali Linux** (cybersecurity focus), each differing in package managers, default software, and target audience.

What is Linux?

- **Kernel:** At its core, Linux is just the kernel, managing hardware and system resources.
- **Open-Source:** Its open-source nature allows anyone to modify and distribute it, leading to many variations.
- **Operating System:** A full Linux OS combines the kernel with libraries, utilities, a graphical interface (like GNOME or KDE), and a package manager.

Popular Linux Distributions (Distros)

- **Ubuntu:** Extremely popular for desktops, known for user-friendliness, large community support, and LTS (Long-Term Support) versions.
- **Linux Mint:** Based on Ubuntu, it offers a familiar Windows-like interface (Cinnamon desktop) and comes with useful apps pre-installed, ideal for new users.
- **Debian:** A foundational, community-driven distro known for its stability and being the base for many others, including Ubuntu and Mint.
- **Fedora:** Backed by Red Hat, it offers newer software and technologies, serving as a testbed for enterprise features.
- **Arch Linux:** A rolling-release distro that's highly minimalist and customizable, requiring users to build their system from the ground up, best for advanced users.

- **Kali Linux:** Specialized for penetration testing and digital forensics, packed with security tools.
- **Red Hat Enterprise Linux (RHEL) & CentOS/Alma Linux:** RHEL is a leading commercial, stable enterprise server OS; CentOS Stream and Alma Linux provide community-supported, RHEL-compatible alternatives.

Key Differences Between Distros

- **Package Manager:** (e.g., APT for Debian/Ubuntu, DNF for Fedora, Pacman for Arch).
- **Desktop Environment:** (e.g., GNOME, KDE Plasma, XFCE, Cinnamon).
- **Philosophy:** From beginner-friendly (Ubuntu) to highly technical (Arch) or server-focused (RHEL).
- **Software Selection & Updates:** Different repositories and update cycles

Linux	Unix
Linux was developed in the 1990s by Linus Torvalds as a free and open-source alternative to Unix.	Unix was developed in the 1970s at Bell Labs
Linux is Open Source, and many programmers work together online and contribute to its development.	Unix was developed by AT&T Labs, different commercial vendors, and non-profit organizations.
Linux, on the other hand, is open-source software and can be used freely without any licensing fees.	Unix is a proprietary operating system, meaning

	that it requires a license to use.
Linux Kernel is Lightweight and modular	Unix kernel is Monolithic and complex
On the other hand, Linux is widely used on both enterprise and personal computers.	Unix is typically found on enterprise-level servers and workstations and is less commonly used on personal computers.
Linux has a large and active community of developers and users who contribute to its development and provide support.	While Unix also has a community, it is generally smaller and more focused on enterprise-level users.
It is an open-source operating system which is freely accessible to everyone.	It is an operating system which can only be utilized by its copywriters.
Threat recognition and solution is very fast because Linux is mainly community driven. So, if any Linux client poses any sort of threat, a team of qualified developers starts working to resolve this threat.	Unix clients require longer hold up time, to get the best possible bug-fixing, and a patch.
File system supports - Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, FAT32, NTFS	File system supports - jfs, gpfs, hfs, hfs+, ufs, xfs, zfs

Linux provides two GUIs, KDE and Gnome. But there are many other options. For example, LXDE, Xfce, Unity, Mate, and so on.	Initially, Unix was a command-based OS, however later a GUI was created called Common Desktop Environment. Most distributions now ship with Gnome.
It is used everywhere from servers, PCs, smartphones, tablets to mainframes.	It is used on servers, workstations, and PCs.
The default interface is bash (Bourne Again Shell). Anybody can use Linux whether a home client, developer or a student.	It initially used Bourne shell. But it is also compatible with other GUIs. Developed mainly for servers, workstations, and mainframes.
The source is accessible to the public.	The source is not accessible to the public.
Some Linux versions are Ubuntu, Debian GNU, Arch Linux, etc.	Some Unix versions are SunOS, Solaris, SCO UNIX, AIX, HP/UX, ULTRIX, etc.

The UNIX Architecture:

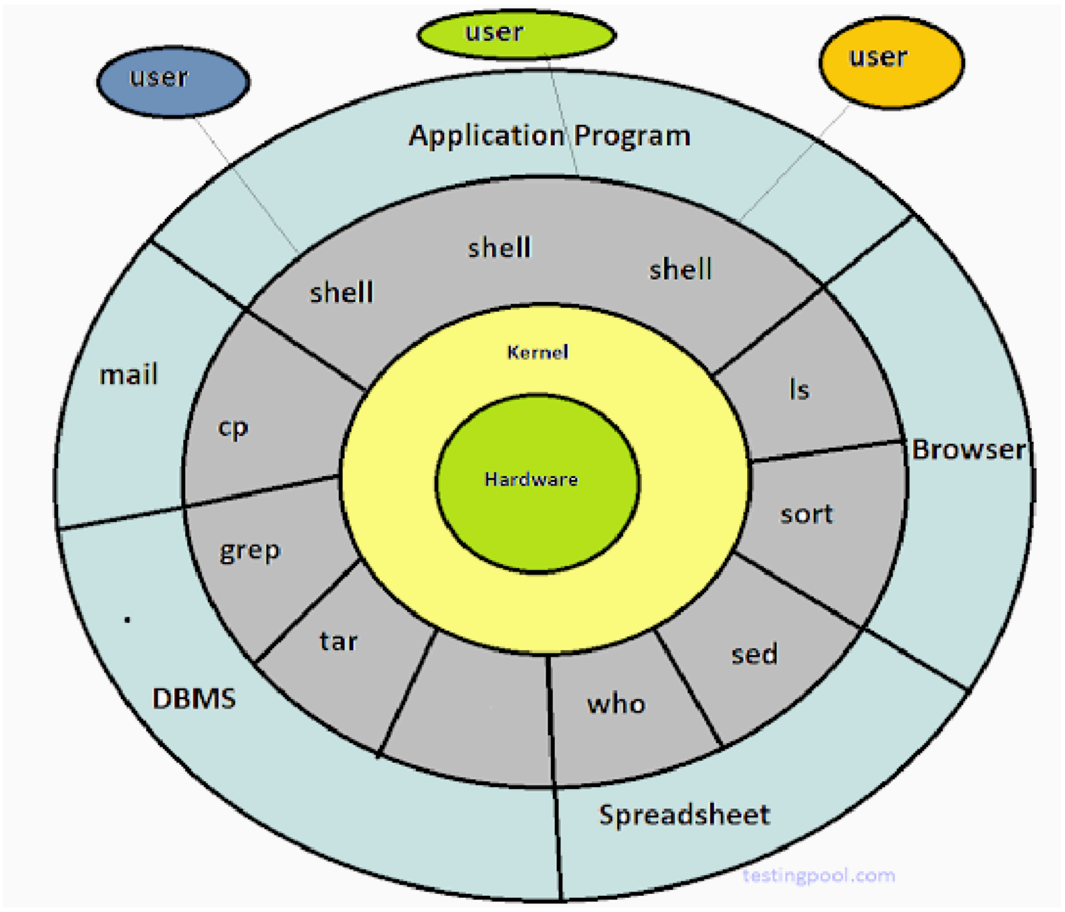
A **Unix architecture** is a computer operating system system architecture that embodies the Unix philosophy. It may adhere to standards such as **the Single UNIX Specification (SUS)** or similar

POSIX IEEE standard. No single published standard describes all Unix architecture computer operating systems - this is in part a legacy of the Unix wars.

The **Unix wars** were the struggles between vendors of the Unix computer operating system in the late 1980s and early 1990s to set the standard for Unix thenceforth (i.e. without being argued).

The architecture of UNIX is basically divided into **four main layers**-

- Layer-1: **Hardwar**
- Layer-2: **Kernel**
- Layer-3**: Shell**
- Layer-4: **Users and Application Program**



The main concept that unites all the versions of UNIX is the following **four basics** –

- 1.**Kernel**
- 2.**Shell**
- 3.**Command and utilities**
- 4.**The File and Process**

1.Kernel: T**he** kernel is the **core (heart)** of the operating system- a collection of routines mostly **written in C**. These routines communicate with the **hardware directly**. It is the part of the UNIX operating system that is **loaded into memory** when the system is **booted**.

The **System calls** are the functions used in the **kernel itself**. **System calls** are used to create files, to provide the basic I/O services, to access the system clock etc.

It interacts with the hardware and most of the tasks **like memory management, task scheduling and file management, network management** for the operating system.

Example of System calls are (i) **fork ()**-create a new process (ii) **nice ()**-change priorities of process (iii) **open ()**-open for reading or writing file (iv) **wait ()**-wait for child process to stop or terminate.

2.Shell: It is the interface between the user and the kernel. When we enter a command, shell check whether the command is valid then execute the result according to the command.

Shell is a **command interpreter** that is communicates between the user and the kernel.

The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the UNIX variants.

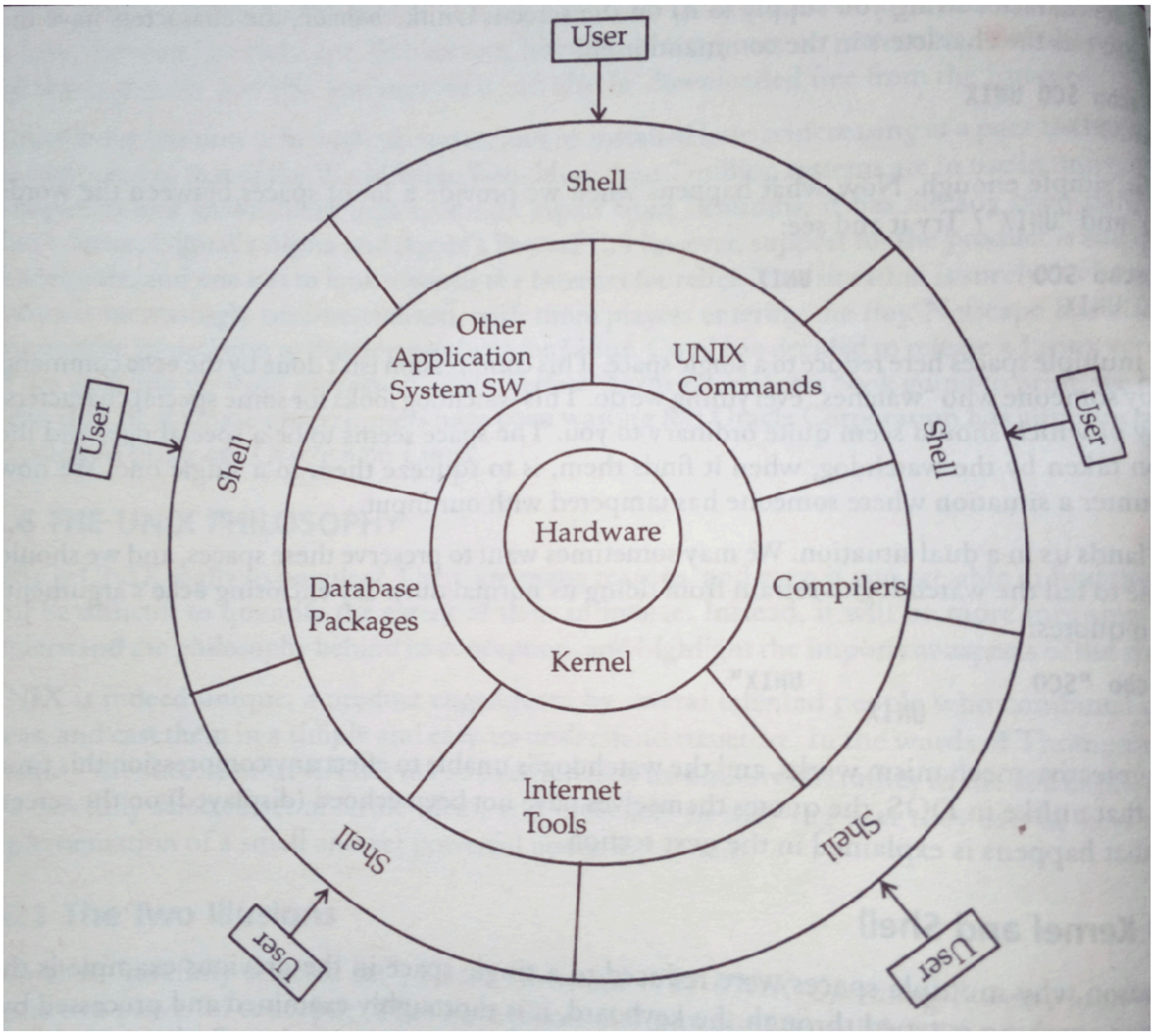
sh is a command language interpreter that executes commands read from a command line string, the standard input, or a specified file. The Bourne shell was developed in 1977 by Stephen Bourne at AT&T's Bell Labs in 1977. It was the default shell of **Unix** Version 7.

3.Commands and Utilities − There are various commands and utilities which you can make use of in your day-to-day activities. **cp**, **mv**, **ls**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.

4.The Files and Process –

File: All the data of UNIX is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **file system**.

Process: The second entities are the **process**, which is name given to a file when it is **executed** as a program. It is simply the **"time image"** of an executable file. Like files, processes also belong to a special hierarchical tree structure. We also treat processes as living organism which have parent, children and grandchildren and are born and die. UNIX provides the tools that allow us to control processes move them between foreground and background and even kill them.



Types of shells:

nameShell	**Developed by**	**Prompt**	**Interpreter name**
Bourne Shell	Stephen Bourne	$	sh
Bash Shell	Stephen Bourne	$	bash
Korn Shell	David Korn	$	ksh
Z shell	Paul	$	zsh
C shell	Bill Joy	%	csh

The advanced version of Bourne shell is Bash shell. Bash means Bourne again shell.

Default Shell name	**Flavor name**
Bash Shell	Linux
Bourne Shell	Sco-Unix, Solaries, HP-UX
Korn Shell	IBM-AIX(Advanced Interactive eXecutive)
C Shell	IRIX (EYE-ricks) developed Silicon Graphics.

Booting process in Unix system(i)turn on power(ii)Boot hardware(BIOS)(iii)Memory resident code(iv)boot block (v)kernel starts (vi)init getty Login Shell .(vii)setup single/multi user mode(viii)Runs the startup scripts.

Booting process in Linux system (i)System start up(BIOS) (ii)MBR(Master Boot record)-Boot loader (iii)Boot loader-GRUB(Grand Unified Boot Loader) (iv)Kernel (v)init (vi)run level program.

How the shell created?

```
Fork   fork-exec     fork-exec
Init---- getty------- Login ---- Shell.
```

Sequence of executing commands by shell:

(i)parsing(ii)Evaluation of variable(iii)command substitution (iv)wild card interpretation(v)path evaluation

Kernel VS Shell:

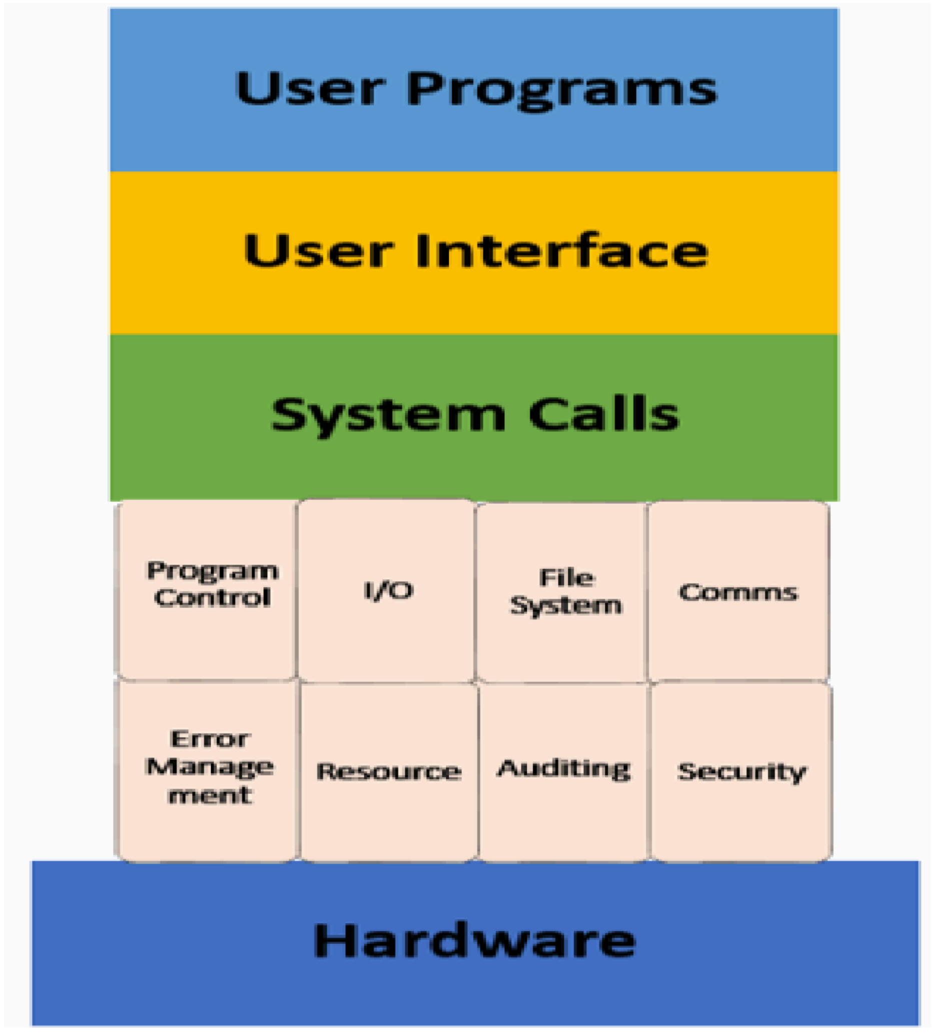
(i) The **Kernel** interacts with machine hardware, and the **Shell** interacts with users.
(ii) A system has only one kernel and a system have more than one shell.
(iii) (iii) The **kernel** is the core of the Unix OS system and collection of routines, which are written in C.
(iv) **Shell** works like an outer part, which interact with users and convey all information to kernel.
(v) **Kernel** can communicate directly with the **hardware,** and the **shell** can communicate directly in the **user.**

System calls:

What is System Call in Operating System?

A **system call** is a mechanism that provides the **interface between a process and the operating system**. It is a programmatic method in which a computer program requests a service from the **kernel of the OS.**

System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.



How System Call Works?

Here are steps for System Call:

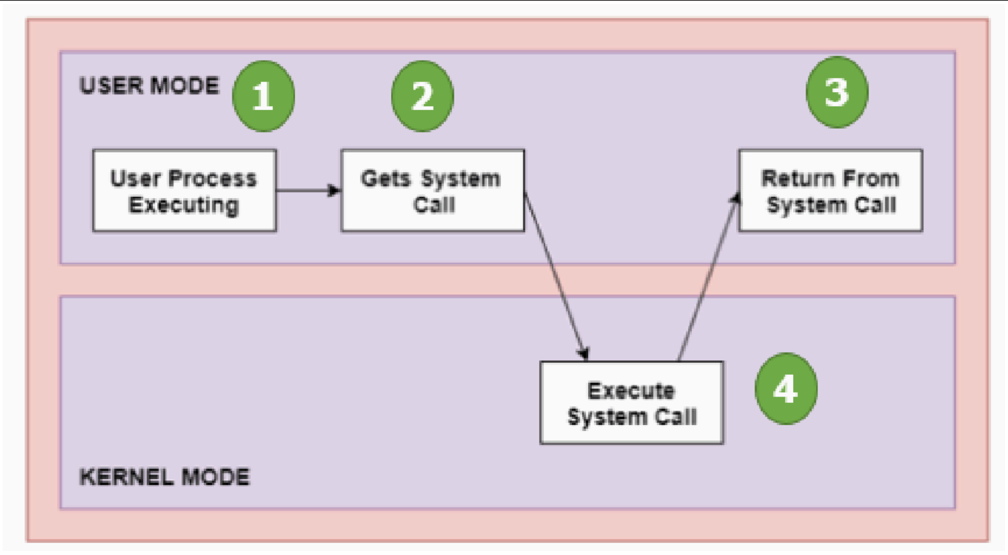


Fig: Architecture of the System Call

As you can see in the above-given diagram.

Step 1) The processes executed in the user mode till the time a system call interrupts it.

Step 2) After that, the system call is executed in the kernel-mode on a priority basis.

Step 3) Once system call execution is over, control returns to the user mode.,

Step 4) The execution of user processes resumed in Kernel mode.

Why do you need System Calls in OS?

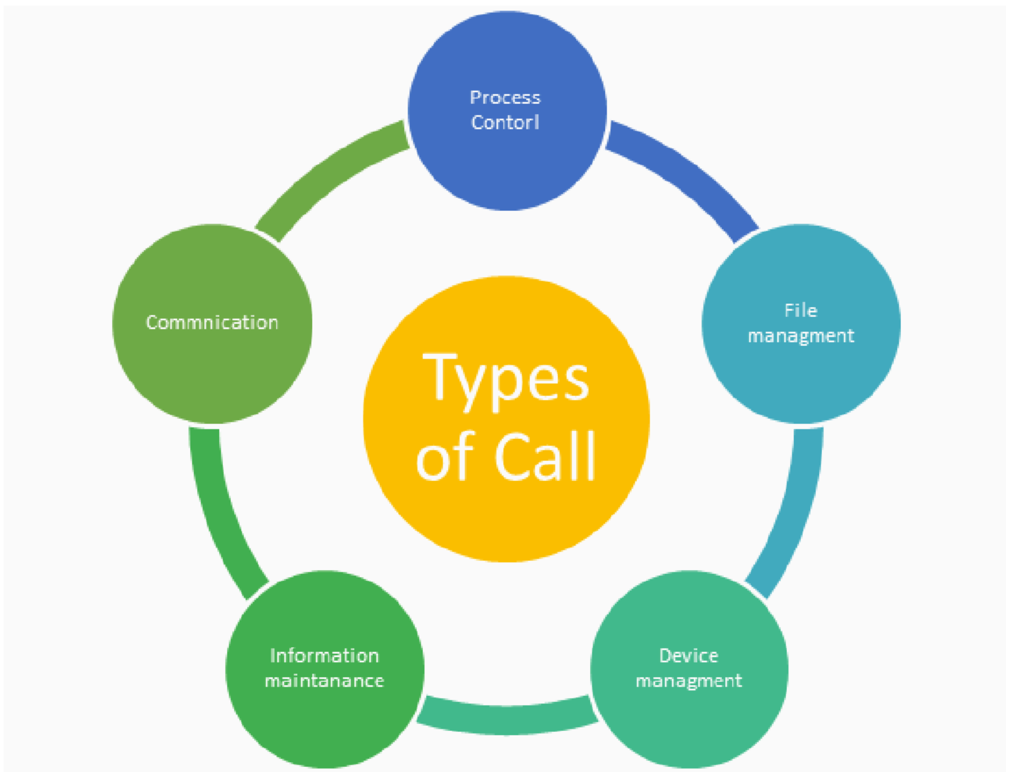
Following are situations which need system calls in OS:

- **Reading and writing** from files demand system calls.
- If a file system wants **to create or delete** files, system calls are required.
- System calls are used for the **creation and management of new processes.**
- **Network connections** need system calls for **sending and receiving packets.**
- **Access to hardware** devices like **scanner, printer**, need a system call.

Types of System calls:

Here are the **five types** of system calls used in OS:

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communications



Process Control
This system calls perform the task of **process creation, process termination**, etc.
Functions:
- End and Abort
- Load and Execute
- Create Process and Terminate Process
- Wait and Signed Event
- Allocate and free memory

File Management
File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.
Functions:

- Create a file
- Delete file
- Open and close file
- Read, write, and reposition
- Get and set file attributes

Device Management
Device management does the job of device manipulation like reading from device buffers,writing into device buffers, etc.

Functions
- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

Information Maintenance
It handles information and its transfer between the OS and the user program.
Functions:
- Get or set time and date
- Get process and device attributes

Communication:
These types of system calls are specially used for interprocess communications.
Functions:
- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

Rules for passing Parameters for System Call:
Here are general common rules for passing parameters to the System Call:
- Parameters should be pushed on or popped off the stack by the operating system.
- Parameters can be passed in registers.
- When there are more parameters than registers, it should be stored in a block, and the block address should be passed as a parameter to a register.

Important System Calls Used in OS:
(i)fork() (ii) exec() (iii) wait() (iv)exit() (v) kill (vi)Open() (vi) Read() (vii) write() (viii) close() (iX) getpid() (x alarm() (xi) sleep()

Q2. What are the tasks performed by kernel?

Answer: There are following main tasks performed by kernel:-

- Memory Management
- Process Management
- File System Management
- Device (Disk) Management
- Scheduling
- Network Management
- Device Driver Management(Hardware)
- Security

E**xplain block diagram of system kernel with diagram:**

Figure 2.1 gives a block diagram of the kernel, showing various modules and their relationships to each other. In particular, it shows the **file subsystem on the left** and the **process control subsystem on the right**, the two major component of the kernel.

The diagram serves as a useful logical view of the kernel, although in practice the kernel deviates from the model because some modules interact with the internal operations of others.

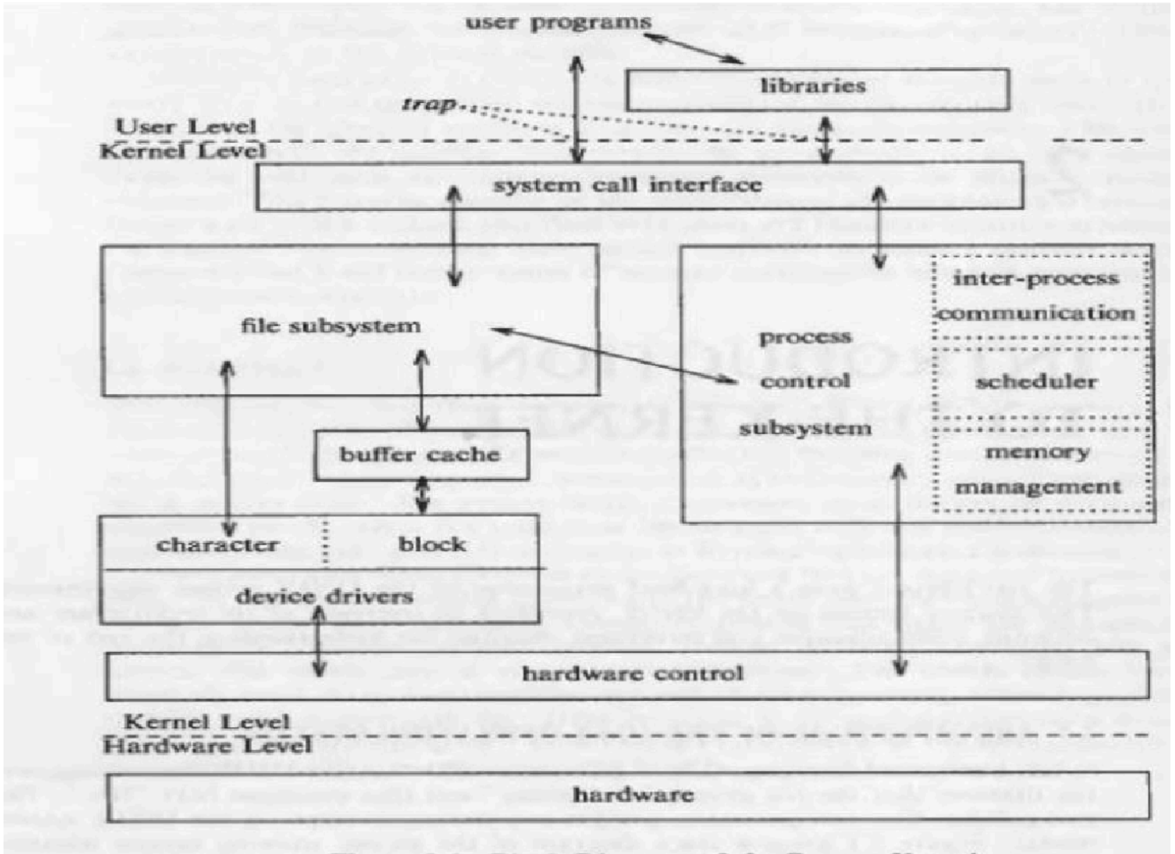


Figure 2.1. Block Diagram of the System Kernel

Figure 2.1 **shows three levels**: **user, kernel**, and **hardware**. System calls look like ordinary function calls in C programs, and libraries map these function calls to the primitives needed to enter the operating system.

Assembly language programs may invoke **system calls directly** without a system call library, however. Programs frequently use other libraries such as the standard I/O library to provide a more sophisticated use of the system calls. The libraries are linked with the programs at compile time.

The figure partitions the set of system calls into those that interact with the file subsystem and those that interact with the process control subsystem. The **file subsystem** manages files, **allocating file space**, administering **free space**, controlling **access to files**, and **retrieving data** for users.

Processes interact with the file subsystem via a specific set of system calls, such as **open** (to open a file for reading or writing), **close, read, write,** stat (query the attributes of a file).

The file subsystem accesses file data using a buffering mechanism that regulates data flow between **the kernel and secondary storage devices**.

Device drivers are the kernel modules that control the operation of peripheral devices. Block **I/O devices** are random access storage devices alternatively; their device drivers make them appear to be random access storage devices to the rest of the system.

The process control subsystem is responsible for **process synchronization**, **inter process communication, memory management**, and **process scheduling**.

The memory management module controls the allocation of memory.

The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel pre-empts them when their recent run time exceeds a time quantum.

POSIX (Portable Operating System Interface)

- **POSIX** defines **the application programming interface (API**), along with **command line shells** and utility interfaces, for software compatibility with variants of Unix and other operating systems.

- **POSIX (Portable Operating System Interface) documentation is divided into two parts:**
- **(i)POSIX.1 and (ii) POSIX.2**

POSIX.1: Core Services (**incorporates Standard ANSI C**) (IEEE Std 1003.1-1988)

o Process Creation and Control

o Signals

- Floating Point Exceptions
- Segmentation / Memory Violations
- Illegal Instructions
- Bus Errors
- Timers

o File and Directory Operations

o Pipes

- C Library (Standard C)
- I/O Port Interface and Control
- Process Triggers

POSIX.1b: Real-time extensions (IEEE Std 1003.1b-1993, later appearing as librt—the
Realtime Extensions library)
- Priority Scheduling
- Real-Time Signals
- Clocks and Timers
- Semaphores
- Message Passing Shared Memory
- Asynchronous and Synchronous I/O
- Memory Locking Interface

POSIX.1c: Threads extensions (IEEE Std 1003.1c-1995)
- Thread Creation, Control, and Cleanup
- Thread Scheduling
- Thread Synchronization
- Signal Handling

POSIX.2: **Shell and Utilities** (IEEE Std 1003.2-1992)
- **Command Interpreter**
- **Utility Programs**

Types of Utility Programs:
Many operating systems provide different types of utility programs to resolve common issues of
software and hardware.

Two types of utility programs are:
- **built-in** (Disk scanner, Disk defragmenter, File viewer) and
- **stand-alone utility** (antivirus, Winzip, WinRAR, Google Chrome).

Unix directory structure.

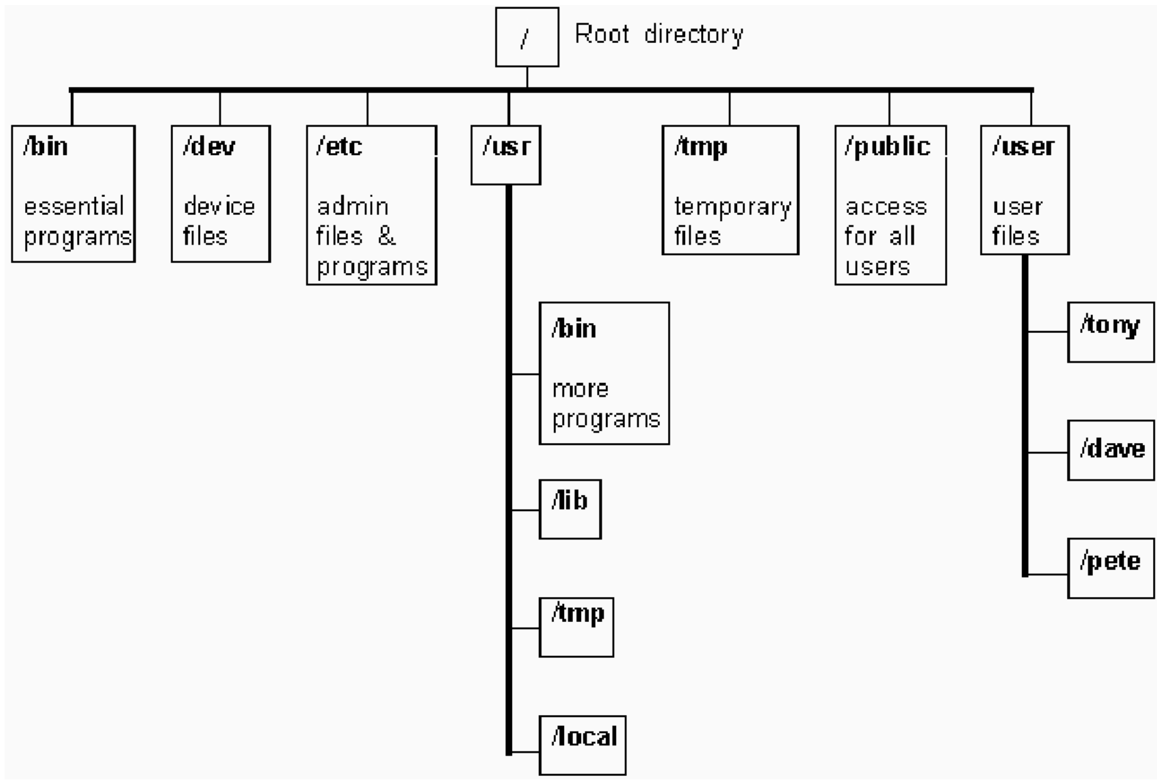
The Unix directory structure is a **hierarchical, tree-like filesystem** that starts from a single top-level directory called the **root directory**, denoted by a single forward slash (/). In Unix, all data, including physical devices and directories themselves, is treated as a file.

This structure is largely governed by the Filesystem Hierarchy Standard (FHS), which defines the purpose of the primary directories to ensure consistency across different Unix-like systems.

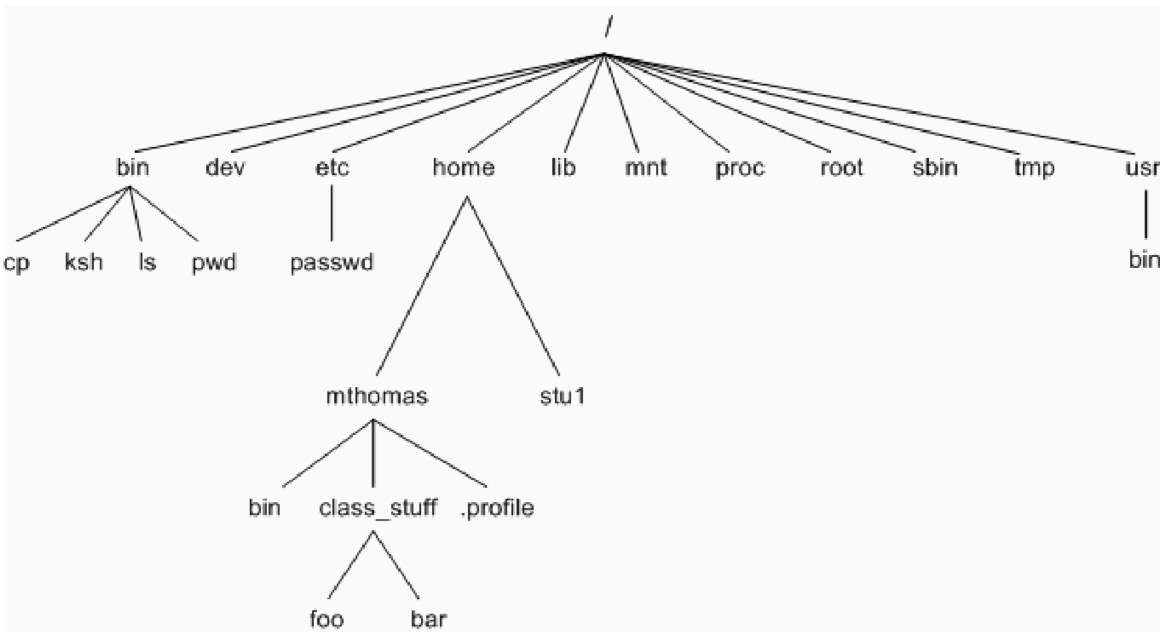
File System and parent-child relationship:

The Unix File system: -

All files in Unix are related to one another. It is a collection of all related files (ordinary, directory and device files) organized in a multi-level hierarchical structure known as a directory tree. The top of the file system is called root, and it is represented by a “ / “ (front slash).



Directories and files have parent-child relationship.



Key Directories in the Unix Filesystem

- **/ (Root)**: The base of the entire file system hierarchy. All other directories and files branch out from here. It is the only directory that does not have a parent.

- **/bin**: Contains essential **user binaries** (executable programs) that are needed for the system to boot and run in minimal mode, such as ls, cp, and mv.

- **/sbin**: Contains **system binaries**, primarily for system administration tasks, that usually require root privileges to execute, like init or iptables.

- **/etc**: Stores host-specific **system-wide configuration files** and system databases. These are typically editable text files used by system services and applications.

- **/lib:** Contains system libraries, and some critical files such as kernel modules or device drivers. contains all library files in a binary form.

- **/dev**: Short for "devices". This directory contains **special files that represent physical hardware devices** attached to the system, such as hard drives (/dev/sda1) or the random number generator (/dev/random).

- **/home**: Contains the **personal home directories for all non-root users**. For a user named "john", their home directory would typically be /home/john.

- **/root**: The **home directory for the root user** (the system administrator). It is separate from the /home directory for security and system integrity reasons.

- **/usr**: Stands for "Unix System Resources" (or historically, "user" home directories). This is a secondary hierarchy for read-only user data and contains the majority of user utilities and applications, including /usr/bin (more user binaries) and /usr/lib (libraries).

- **/var**: Short for "variable". This location is for **files whose content is expected to change frequently** or grow, such as system **log files** (/var/log), mail spools, and databases.

- **/var/log:** Contains system log files

- **/tmp**: A place for **temporary files** created by various programs. Data in this directory is usually cleared upon system reboot.

- **/proc**: A **virtual filesystem** that provides an interface to the kernel and running processes. It contains real-time system information, such as CPU and memory usage statistics.

- **/mnt** and **/media**: Common mount points for temporarily mounting external filesystems and removable media like USB drives and DVDs.

Key Concepts

- **Everything is a File**: Unix treats directories, regular files, and hardware devices as different types of files.

- **Pathnames**: Files are located using pathnames.
  - **Absolute pathnames** start with / (e.g., /home/user/file.txt) and define the path from the root directory.
  - **Relative pathnames** define the path from the current working directory. Shortcuts include . for the current directory and .. for the parent directory.

- **Tilde (~)**: A convenient shortcut character that the shell interprets as the current user's home directory.

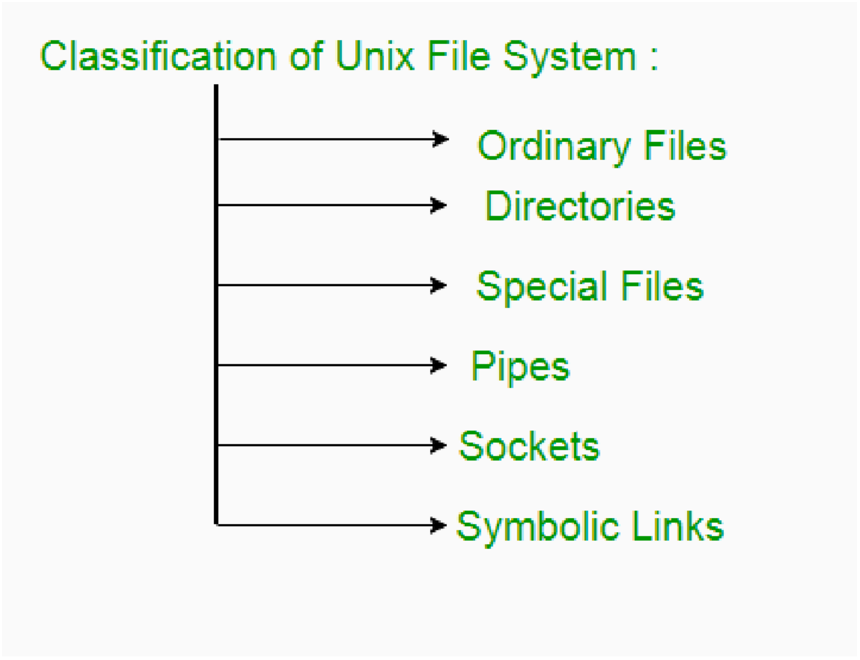
HOME variable:

What is Home variable in Linux?

HOME contains the path to the **home** directory of the current user. This **variable** can be used by applications to associate configuration files and such like with the user running it.

For example: /home/student1, /home/student2

Types of Unix files – The UNIX files system contains several different types of files:



1. Ordinary (Regular) files – An ordinary file is a file on the system that contains data (text, or program instructions).

An ordinary file itself is divided into two parts
(a) Text file (printable characters),
(b) Binary file (printable and nonprintable characters that cover the entire ASCII range 0-255).
2. Directory files- Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.

A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory.

Each entry has two components.
(i) The Filename
(ii) A unique identification number for the file or directory (called the **inode number**).

3. Device/Special Files – Used to represent a real physical device such as a printer, tape drive or terminal used for Input/output (I/O) operations. They appear in a file system just like an ordinary file or a directory.

4. Pipes – UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another.

A Unix pipe provides a one-way flow of data. **The output or result of the first command sequence is used as the input to the second command sequence**. To make a pipe, put a **vertical bar (|)** on the command line between two commands.

For example: **who | wc –l**

5.Sockets – A UNIX socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server application framework.

6. Symbolic Link – Symbolic link is used for referencing some other file of the file system. Symbolic link is also known as **soft link**. It contains a text form of the path to the file it references **Symbolic links** are much like Windows **shortcuts**. They are like an alias that points to the real object in the file system. If we delete the source file or move it to a different location, symbolic file will not function properly.