MULTICS:

Multics (**Mult**iplexed **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.**

UNIX :

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.
- 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**.

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.

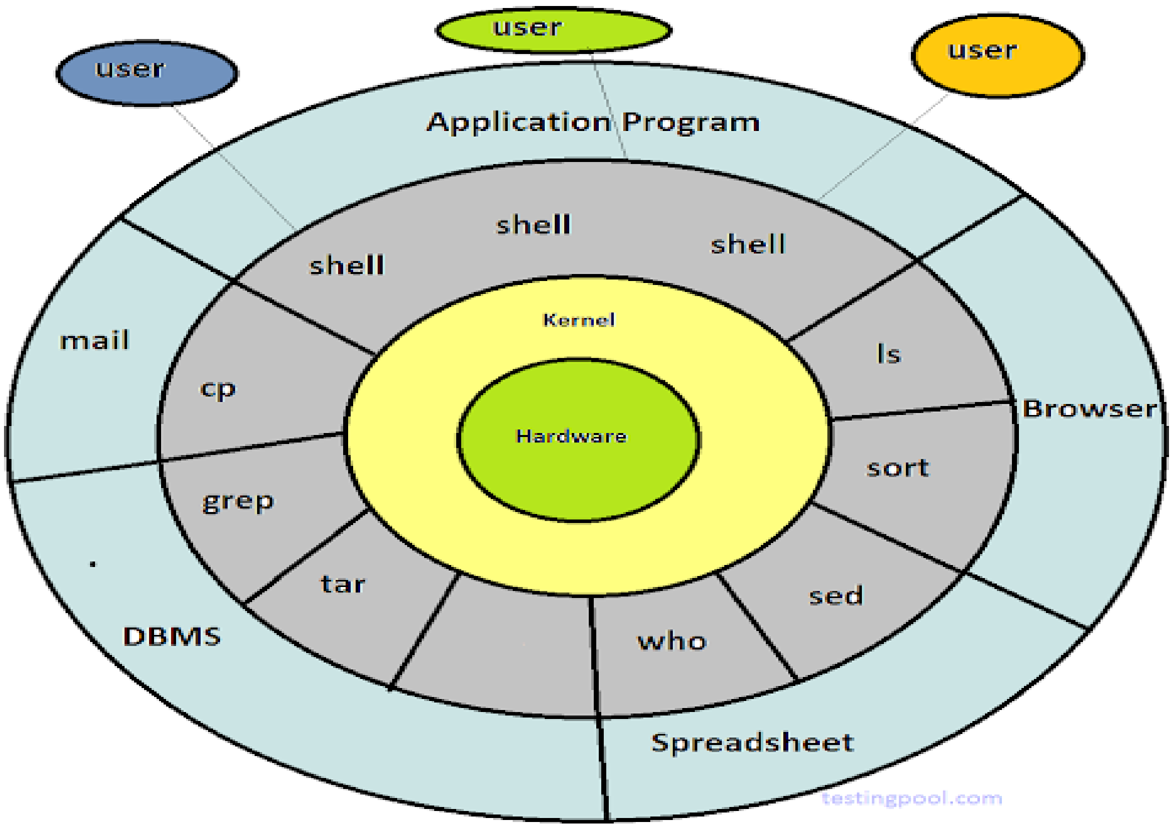
The UNIX Architecture:

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**

Kernel: **The** 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.

Shell: It is actually 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.

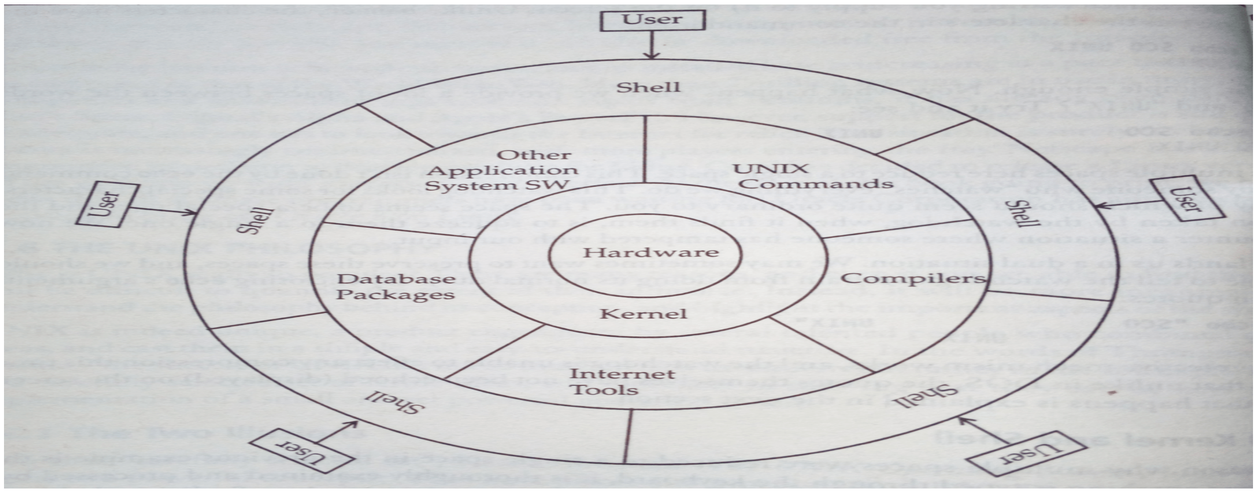
sh is a command language interpreter that executes commands read from a command line string, the standard input, or a specified file. The Bourne shell was developed in 1977 by Stephen Bourne at AT&T's Bell Labs in 1977. It was the default shell of **Unix** Version 7.

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.

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 is created?

$$\begin{tikzcd} \text{Init} \arrow[r, dash, "\text{Fork}"] & \text{getty} \arrow[r, dash, "\text{fork-exec}"] & \text{Login} \arrow[r, dash, "\text{fork-exec}"] & \text{Shell.} \end{tikzcd}$$

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) The **kernel** is the core of the Unix OS system and collection of routines, which are written in C.

Shell works like an outer part, which interact with users and convey all information to kernel.

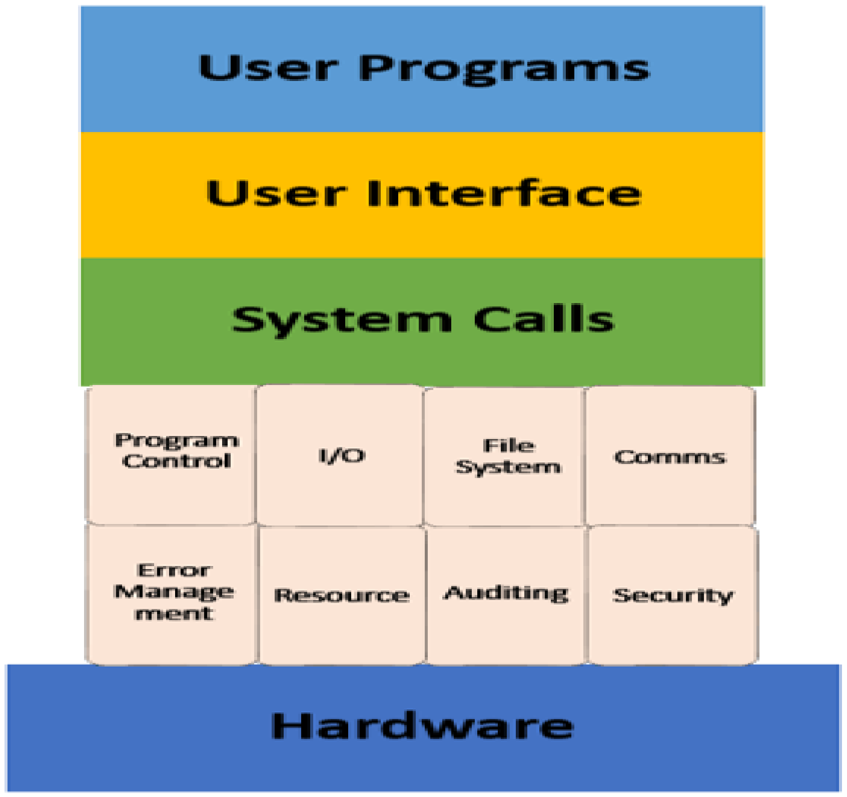
(iv) **Kernel** can communicate directly with the **hardware** and the **shell** can communicate directly in the **user.**

<u>**System calls:**</u>

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:

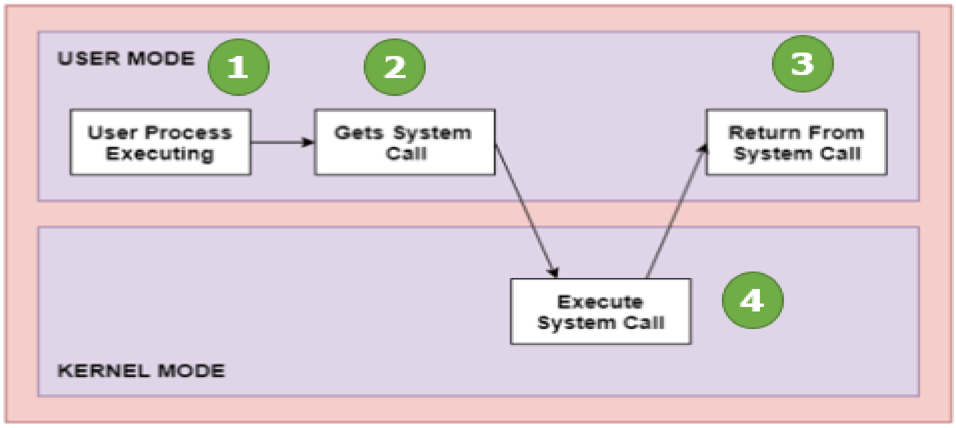


Fig: Architecture of the System Call

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:

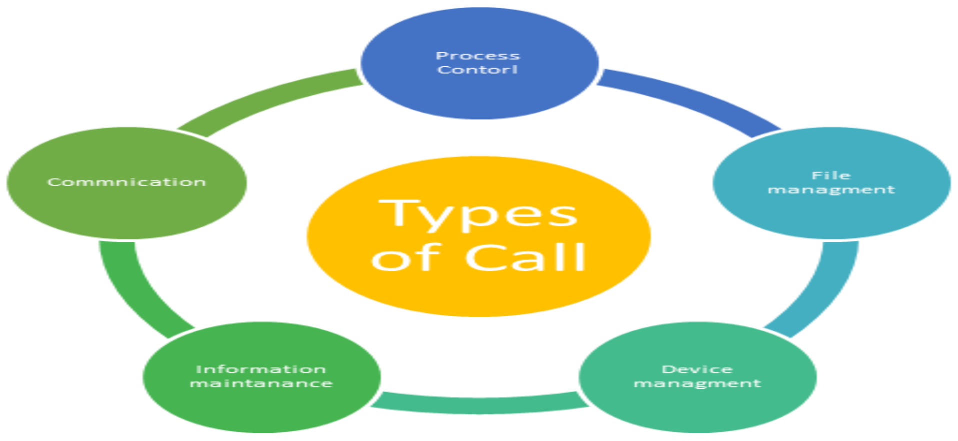
- **Reading and writing** from files demand system calls.
- If a file system wants **to create or delete** files, system calls are required.
- System calls are used for the **creation and management of new processes.**
- **Network connections** need system calls for **sending and receiving packets.**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

Explain 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.

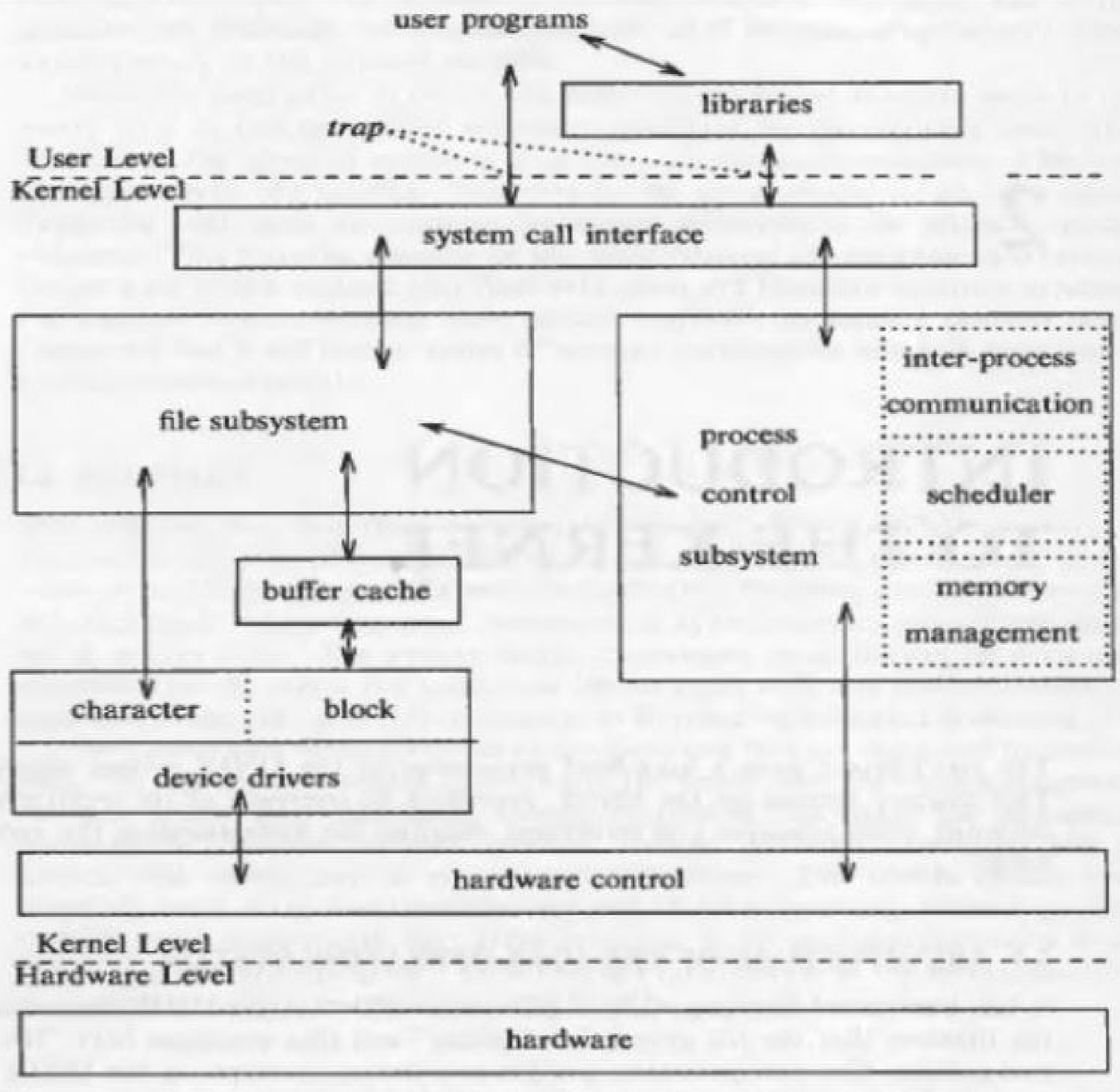


Figure 2.1. Block Diagram of the System Kernel

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**, **interprocess 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 preempts them when their recent run time exceeds a time quantum.

Features and Advantages of UNIX Operating System:

Advantages/principles/Features of Unix/Linux Operating system:

(i)File and Process :

File: UNIX uses a **hierarchical file structure** in its design that **starts** with **a root** directory**---** signified by the **forward slash (/).** The root is followed by its subdirectories, as in an inverted tree, and ends with the file.

It allows for **easy maintenance** and **efficient implementation.**

Process: A **process** is a **program in execution** in memory or in other words, an **instance** of a program in memory. Any program **executed creates a process**. A program can be a command, a shell script, or any binary executable or any application. However, not all commands end up in creating process, there are some exceptions.

(ii) Multi user system: A multi-user Operating system allows more than one user to share the same computer system at the same time.

UNIX is a multi-user system designed to support a group of users simultaneously. The system allows for the sharing of processing power and peripheral resources, white at the same time providing excellent security features.

(iii)Multi tasking system: More than one program can be run at a time. The main concept of multitasking is maximum utilizing CPU resources

it means when a active task in in process, there can be a simultaneous background process working too. Unix handles these active and background threads efficiently and manages the system resources in a fair-share manner.

(iv)Multi programming system: the technique of utilizing several programs concurrently in a single computer system via multiprocessing.

Software Development Tools:

UNIX offers an excellent variety of tools for software development for all phases, from program editing to maintenance of software

Advantages of using Multi programming:(i) High speed utilizations (ii) Main memory utilizations (iii) allocation of a computer system and its resources to more than one concurrent application.

(v)Programming facility: **UNIX** o/s provides shell. Shell works like a **programming** language. It provides commands and keywords. It is an **interpreter** based language.

UNIX shell:

UNIX has a simple user interface called the shell that has the power to provide the services that the user wants. It protects the user from having to know the intricate hardware details.

Pipes and Filters:

UNIX has facilities called Pipes and Filters which permit the user to create complex programs from simple programs.

(vi)Pattern matching : Pattern matching is the process of checking whether a specific sequence of characters/tokens/data exists among the given data. Regular programming languages make use of regular expressions (regex) for **pattern matching**.

(vii) More secures than windows.

UNIX/LINUX has given **two levels of securities**.

System level Security: Its controlled by **system Administrator**.
File level Security: Its controlled by **owner** of the file.

(viii) Communication: The main concept of communication facility **exchanging of information** or files form one user account to other user account.

(ix)Portability: Portability means Independent of hardware & processors.

(x)Documentation:

A **man page** (short for **manual page**) is a form of **software documentation** usually found on a Unix or Unix-like operating system. Topics covered include computer programs (including **library and system calls**), commands, formal standards and conventions.

By default, man typically uses a terminal pager commands or program such as more or less to display its output.

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)
    - Process Creation and Control
    - Signals
        - Floating Point Exceptions
        - Segmentation / Memory Violations
        - Illegal Instructions
        - Bus Errors
        - Timers
    - File and Directory Operations
    - 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).

Single UNIX Specification (SUS:

The **Single UNIX Specification** (**SUS**) is the collective name of a family of standards for computer operating systems, compliance with which is required to qualify for using the "UNIX" trademark. The core specifications of the SUS are developed and maintained by the Austin Group, which is a joint working group of IEEE, ISO JTC 1 SC22 and The Open Group. If an operating system is submitted to The Open Group for certification, and passes conformance tests, then it is deemed to be compliant with a UNIX standard such as UNIX 98 or UNIX 03.

Very few BSD and Linux-based operating systems are submitted for compliance with the Single UNIX Specification, although system developers generally aim for compliance with POSIX standards, which form the core of the Single UNIX Specification.

<u>Internal and external commands:</u>

What is the difference between internal and external commands?

UNIX commands are classified into two types:

Internal commands are commands that **are already loaded** in the system. They can be executed any time and are independent. On the other hand, **external commands** are loaded when the user requests for them**. Internal commands don't require a separate process** to execute them. **External commands** will have an **individual process**.

Internal commands are **a part of the shell** while **external commands require a Path**. If the files for the command are **not present** in the path, the external command **won't execute**.

- The commands that are **directly executed** by the shell are known as internal commands. No separate process is there to run these commands.
- The commands that are executed by the kernel are knows as external commands. Each command has its **unique process id.**

How to find out whether a command is internal or external?
type command:

```
$ type cd

#cd is a shell builtin internal type command

$ type cat

#cat is /bin/cat external type command
```

For the **internal commands,** the type command will clearly say its **shell built-in**, however for the **external commands**, it gives the **path of the command from where it is executed..**

Example:

- **Internal Commands** - : mkdir, cd, cp, mv, r m, fg , pwd, kill, history, read, logout, return , exit, set, help
- **External Commands** - : ls, cat ,head, tail , more , grep.