

Structure of Operating System

Unit-1

Operating System/01CE1401



Department of Computer Engineering
Prof. Shilpa Singhal



Table of Content

- Structure of Operating System
 - Monolithic Kernel
 - Micro-Kernel
 - Layered System
 - Modular System
 - Virtual Machines
- System Calls
- Interrupts
 - Hardware Interrupt
 - Software Interrupt

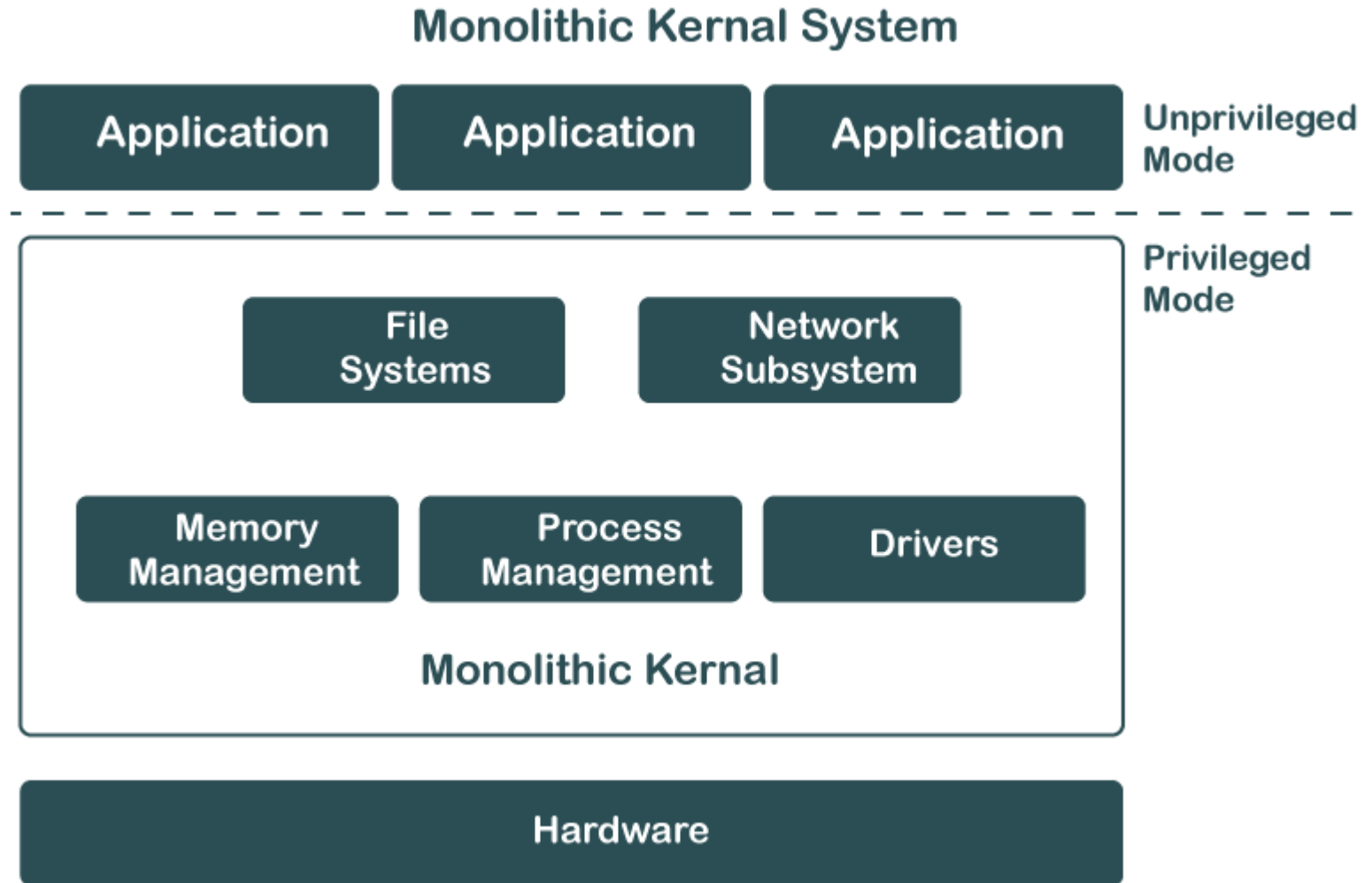
1.Monolithic Kernel



The monolithic operating system controls all aspects of the operating system's operation, including file management, memory management, device management, and operational operations.

The core of an operating system for computers is called the **kernel (OS)**. All other System components are provided with fundamental services by the kernel. The operating system and the hardware use it as their main interface. When an operating system is built into a single piece of hardware, such as a keyboard or mouse, the kernel can directly access all of its resources.

Monolithic Kernel



Advantages of Monolithic Structure:

- Because layering is unnecessary and the kernel alone is responsible for managing all operations, it is easy to design and execute.
- Due to the fact that functions like memory management, file management, process scheduling, etc., are implemented in the same address area, the monolithic kernel runs rather quickly when compared to other systems. Utilizing the same address speeds up and reduces the time required for address allocation for new processes.

Disadvantages of Monolithic Structure:

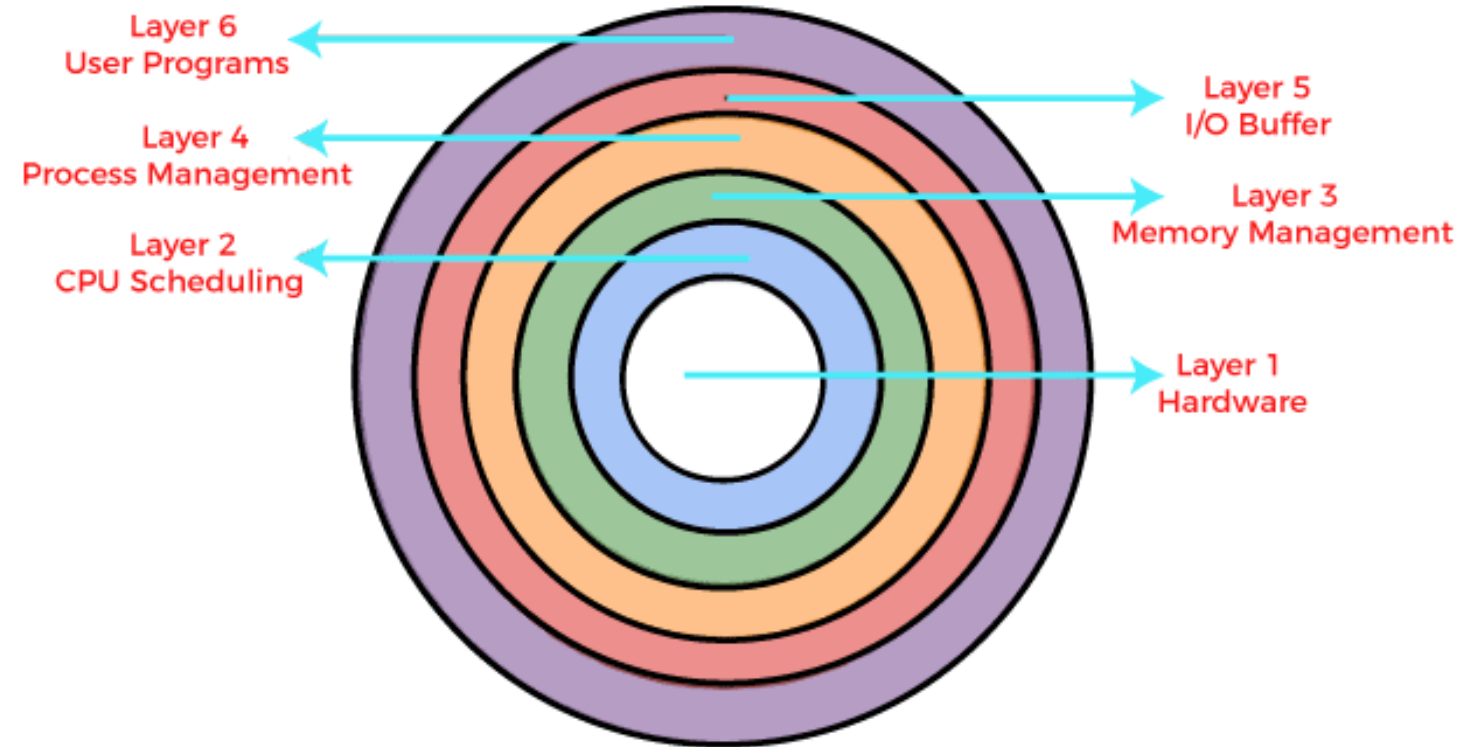
- The monolithic kernel's services are interconnected in address space and have an impact on one another, so if any of them malfunctions, the entire system does as well.
- It is not adaptable. Therefore, launching a new service is difficult.

2.Layered System



- The layered structure approach breaks up the operating system into different layers and retains much more control on the system. The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers only.
- All the layers can be defined separately and interact with each other as required. Also, it is easier to create, maintain and update the system if it is done in the form of layers. Change in one layer specification does not affect the rest of the layers.
- A key problem in layered system is deciding on the ordering of the layers.(This is critical because of the requirement that each layer can ultimately only use services provided by layers below it so the ordering cannot have any cycles.)

Layered System



Advantages of Layered Structure:

- Work duties are separated since each layer has its own functionality, and there is some amount of abstraction.
- Debugging is simpler because the lower layers are examined first, followed by the top layers.

Disadvantages of Layered Structure:

- Performance is compromised in layered structures due to layering.
- Construction of the layers requires careful design because upper layers only make use of lower layers' capabilities.

3. Micro-Kernel System

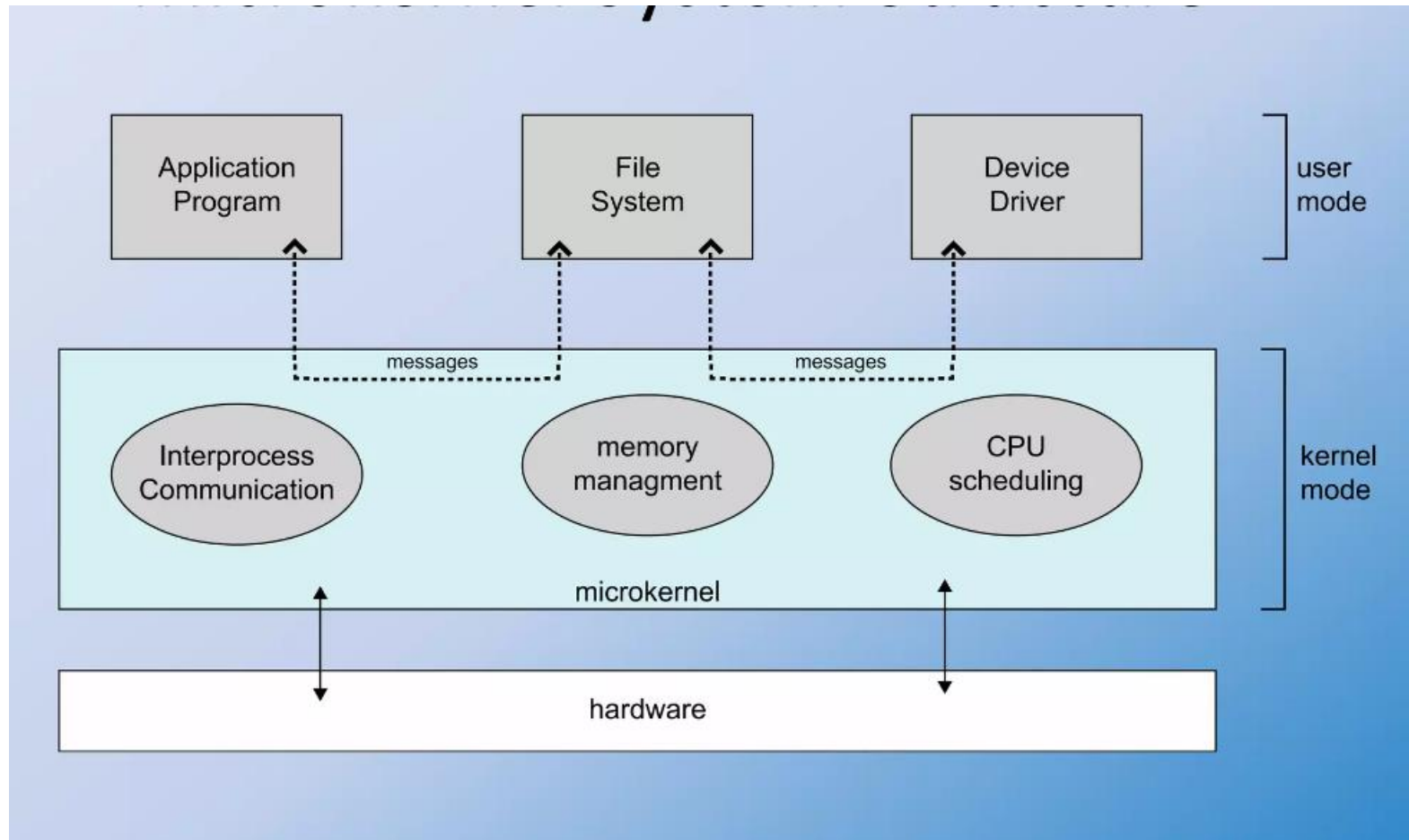
Kernel is made as small as possible only the most important services are put inside the kernel and rest of the OS service are present in the system application program.

The user can easily interact with those not-so important services within the system applications kernel is solely responsible for the three most important services of operating system namely:

- Inter-Process communication
- Memory management
- CPU scheduling

- Microkernel and system applications can interact with each other by message passing as and when required.
- This is extremely advantageous architecture since burden of kernel is reduced and less crucial services are accessible to the user and hence security is improved too. It is being highly adopted in the present-day systems.
- MacOSX, Eclipse IDE is a good example of Microkernel Architecture.

Micro-Kernel System



MicroKernel vs Monolithic

S. No.	Parameters	Microkernel	Monolithic kernel
1.	Address Space	In microkernel, user services and kernel services are kept in separate address space.	In monolithic kernel, both user services and kernel services are kept in the same address space.
2.	Design and Implementation	OS is complex to design.	OS is easy to design and implement.
3.	Size	Microkernel are smaller in size.	Monolithic kernel is larger than microkernel.
4.	Functionality	Easier to add new functionalities.	Difficult to add new functionalities.
5.	Coding	To design a microkernel, more code is required.	Less code when compared to microkernel

S. No.	Parameters	Microkernel	Monolithic kernel
6.	Failure	Failure of one component does not effect the working of micro kernel.	Failure of one component in a monolithic kernel leads to the failure of the entire system.
7.	Processing Speed	Execution speed is low.	Execution speed is high.
8.	Extend	It is easy to extend Microkernel.	It is not easy to extend monolithic kernel.
9.	Communication	To implement IPC messaging queues are used by the communication microkernels.	Signals and Sockets are utilized to implement IPC in monolithic kernels.
10.	Debugging	Debugging is simple.	Debugging is difficult.

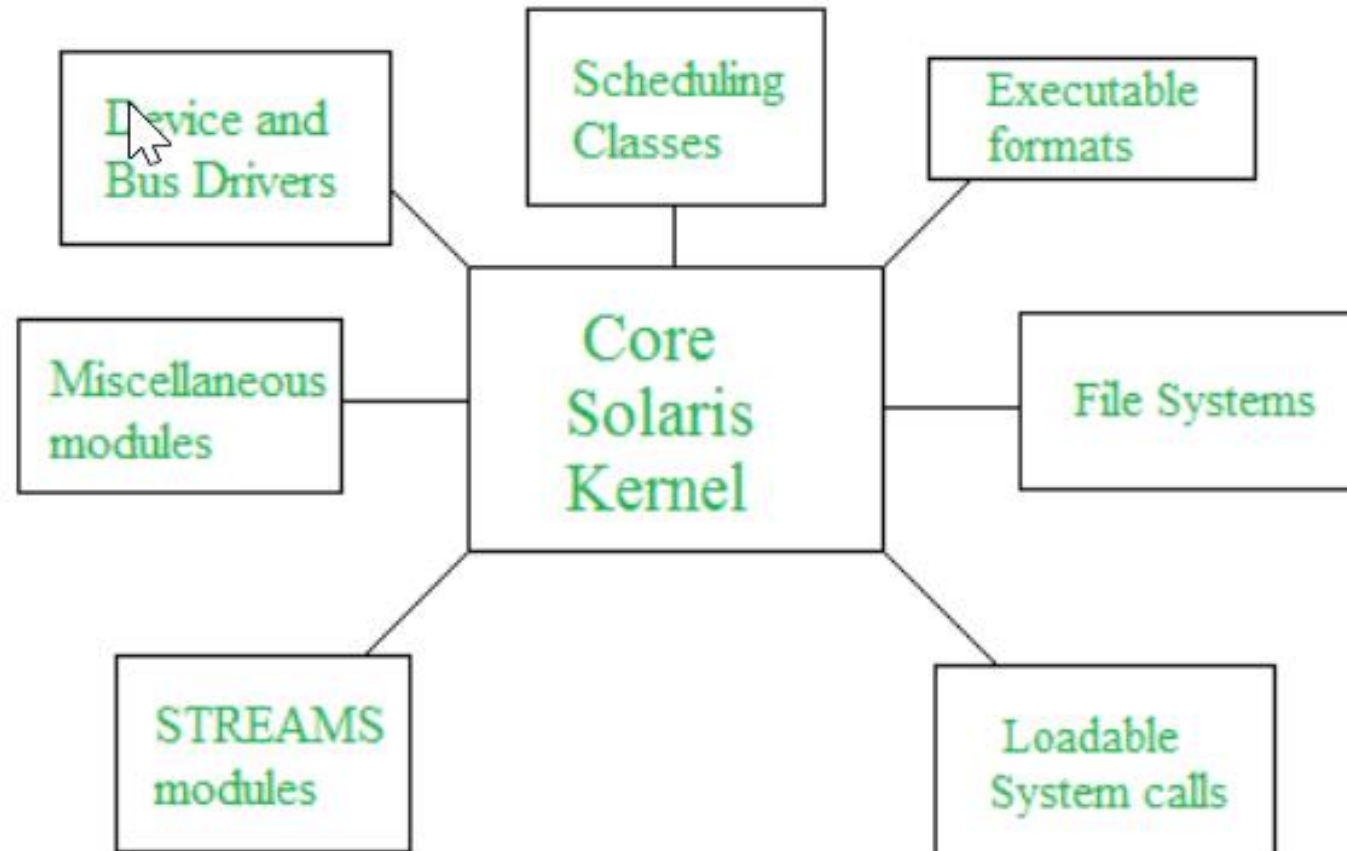
S. No.	Parameters	Microkernel	Monolithic kernel
11.	Maintain	It is simple to maintain.	Extra time and resources are needed for maintenance.
12.	Message passing and Context switching	Message forwarding and context switching are required by the microkernel.	Message passing and context switching are not required while the kernel is working.
13.	Services	The kernel only offers IPC and low-level device management services.	The Kernel contains all of the operating system's services.
14.	Example	Example : Mac OS X.	Example : Microsoft Windows 95.

4. Modular System



It is considered as the best approach for an OS. It involves designing of a modular kernel. The kernel has only a set of core components and other services are added as dynamically loadable modules to the kernel either during runtime or boot time. It resembles layered structure due to the fact that each kernel has defined and protected interfaces, but it is more flexible than a layered structure as a module can call any other module.

Modular System



5. Virtual Machine



Marwadi
University
Marwadi Chandarana Group



Based on our needs, a virtual machine abstracts the hardware of our personal computer, including the CPU, disc drives, RAM, and NIC (Network Interface Card), into a variety of different execution contexts, giving us the impression that each execution environment is a different computer. An illustration of it is a virtual box.

Virtual Machine



- A virtual machine (VM) is an efficient, isolated duplicate of a real machine
- Enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS
- The individual user was quite conscious of sharing a particular machine with a number of other users. For example, the user would have to specifically request certain system resources when they were needed
- Virtual box and VMware is example of virtual machine

Virtual Machine



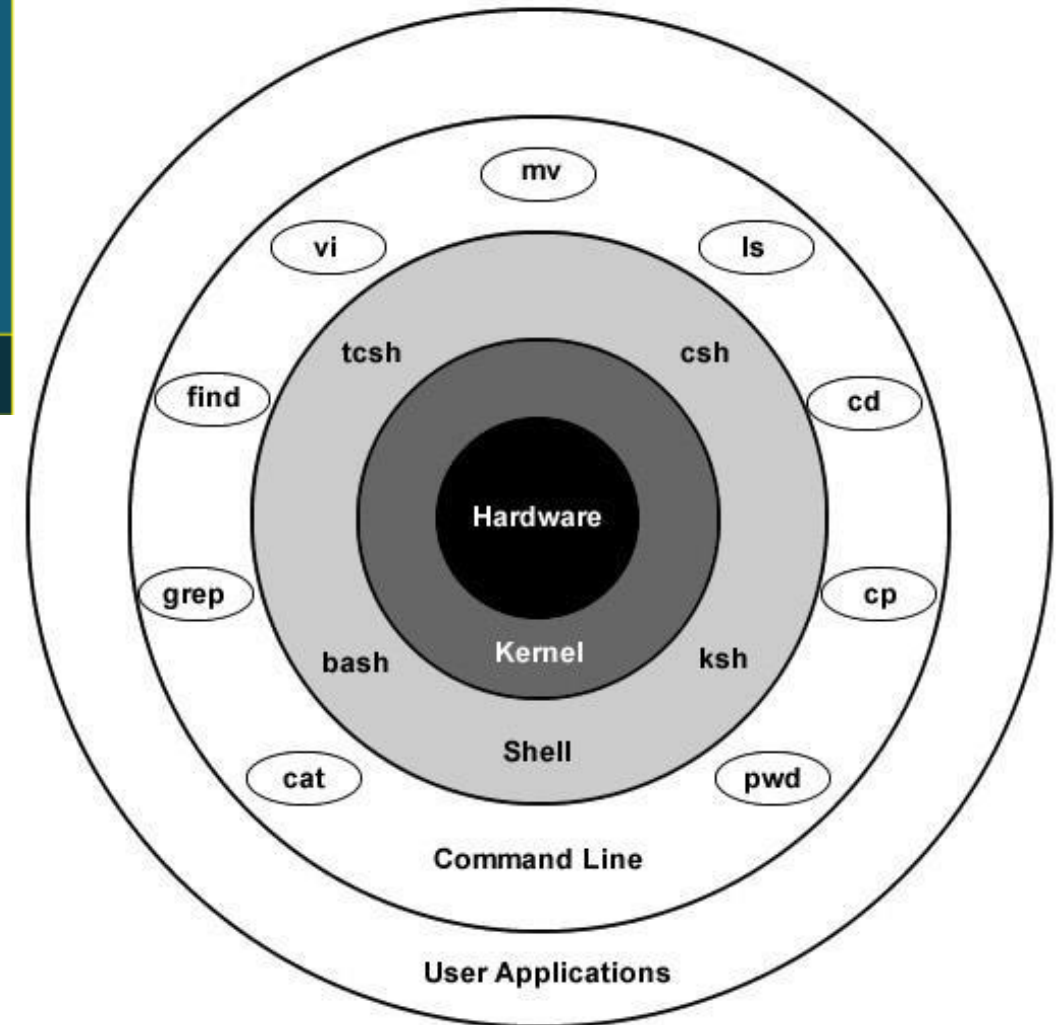
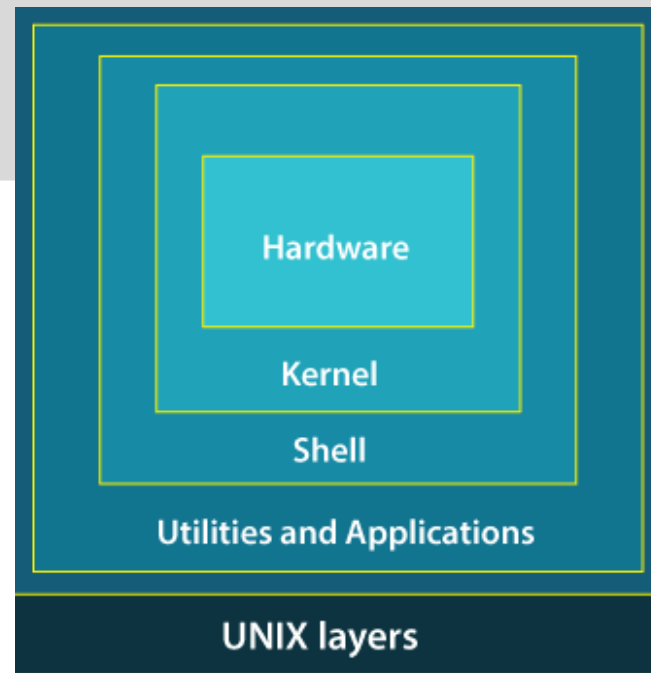
Apparent Situation

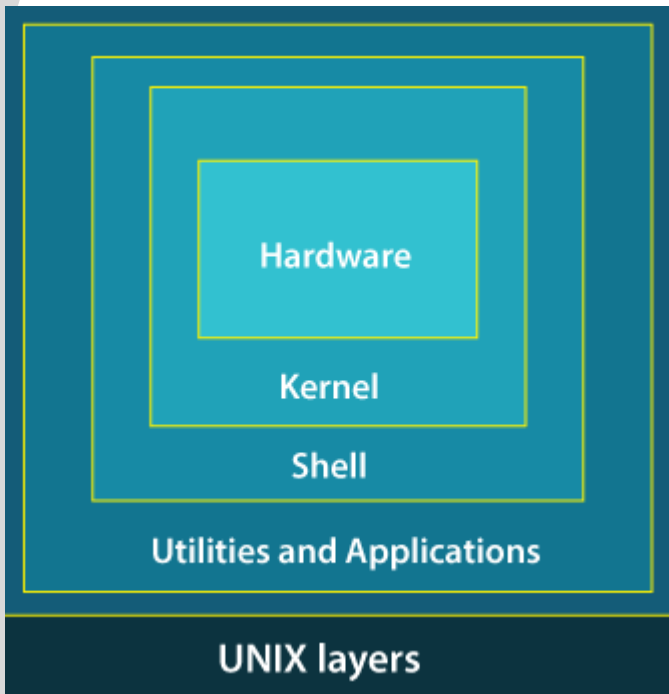
One-User OS
Dedicated Machine

Real Situation

One-User OS	One-User OS	One-User OS
VM		
Shared Machine		

UNIX SYSTEM STRUCTURE





Layer-1: Hardware -

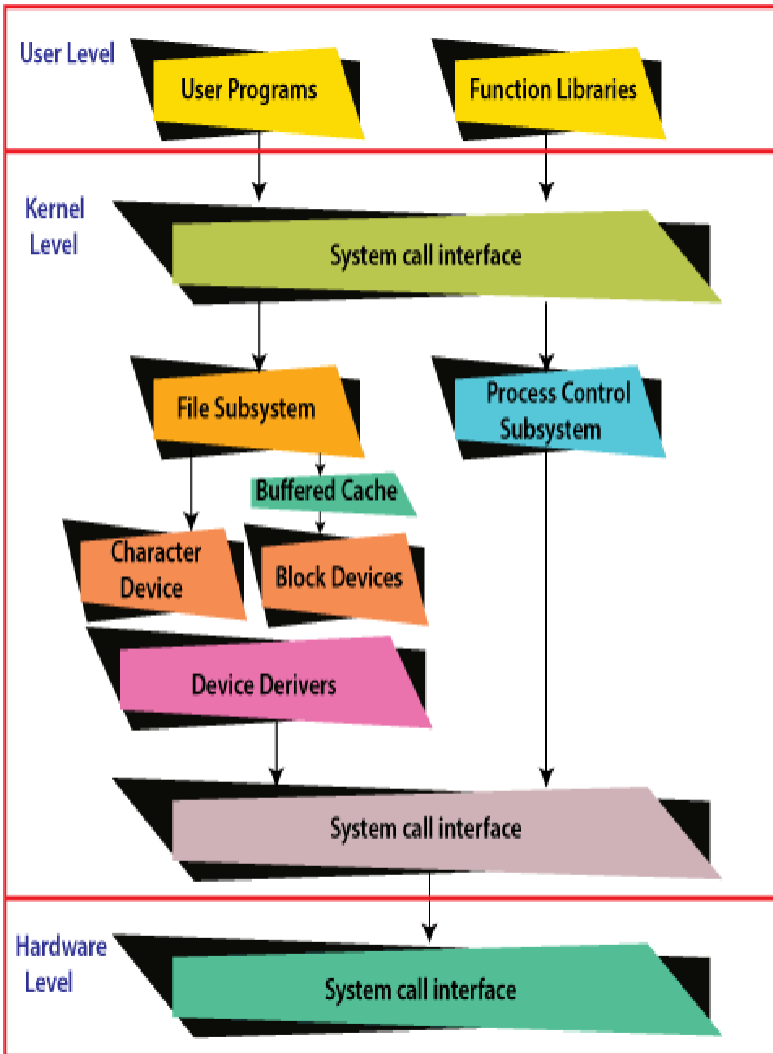
This layer of UNIX consists of all hardware-related information in the UNIX environment.

UNIX SYSTEM STRUCTURE



Layer-2: Kernel -

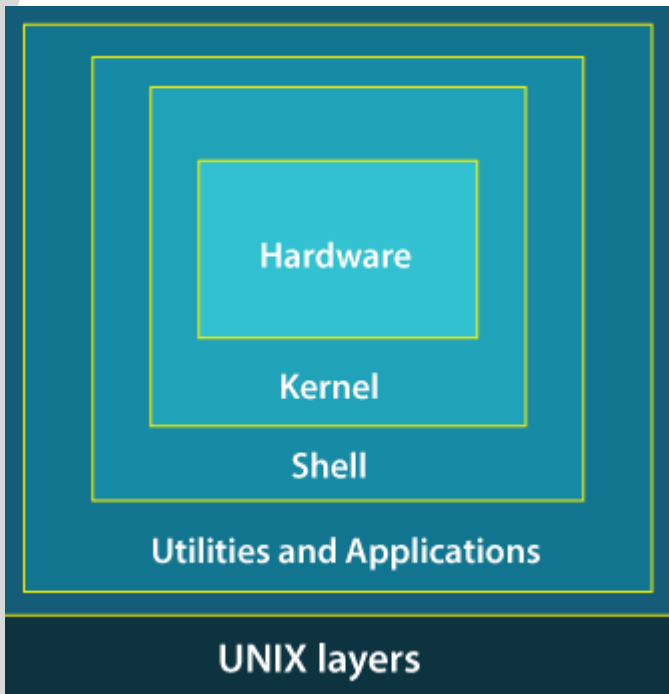
- The core of the operating system that's liable for maintaining the full functionality is named the kernel.
- The kernel of UNIX runs on the particular machine hardware and interacts with the hardware effectively.
- The kernel is the heart of the operating system.
- The kernel ingests user input via the shell and accesses the hardware to perform things like memory allocation and file storage.
- It also works as a device manager and performs valuable functions for the processes which require access to the peripheral devices connected to the computer.
- The kernel controls these devices through device drivers.
- The kernel also manages the memory. Processes are executed programs that have owner's humans or systems who initiate their execution.
- The system must provide all processes with access to an adequate amount of memory, and a few processes require a lot of it.
- To make effective use of main memory and to allocate a sufficient amount of memory to every process. It uses essential techniques like paging, swapping, and virtual storage.



Kernel Architecture

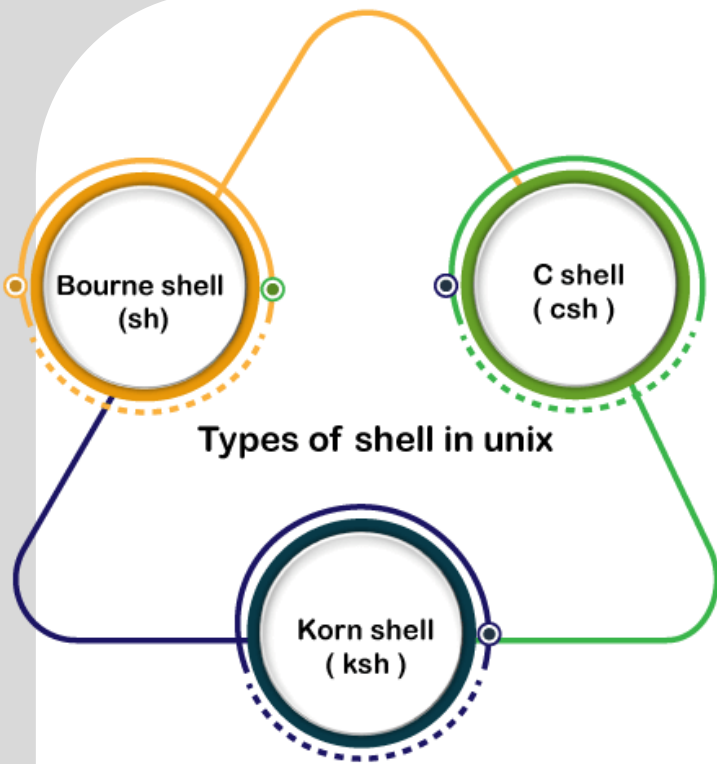
UNIX SYSTEM STRUCTURE

Layer-3: The Shell -



- The Shell is an interpreter that interprets the command submitted by the user at the terminal, and calls the program you simply want
- Shell are the Soul of the operating system..
- It also keeps a history of the list of the commands you have typed in.
- If you need to repeat a command you typed it, use the cursor keys to scroll up and down the list or type history for a list of previous commands.
- There are various commands like cat, mv, cat, grep, id, wc, and many more.

UNIX SYSTEM STRUCTURE

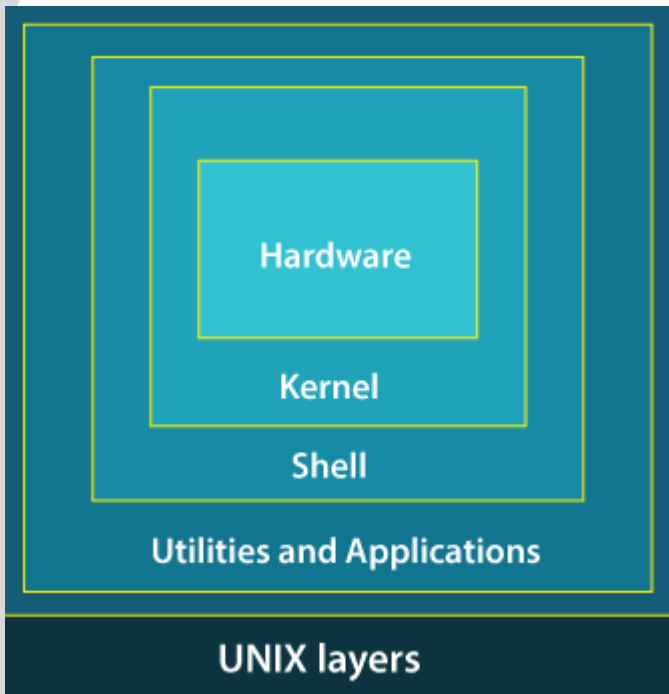


Type of Shell:

Bourne Shell: This Shell is simply called the Shell. It was the first Shell for UNIX OS. It is still the most widely available Shell on a UNIX system.

C Shell: The C shell is another popular shell commonly available on a UNIX system. The C shell was developed by the University of California at Berkeley and removed some of the shortcomings of the Bourne shell.

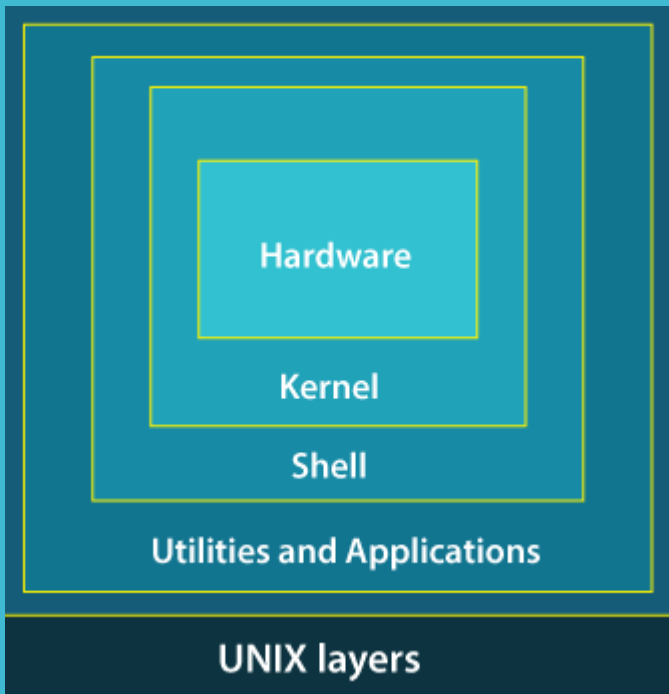
Korn Shell: This Shell was created by David Korn to address the Bourne Shell's user-interaction issues and to deal with the shortcomings of the C shell's scripting quirks.



Layer-4: Application Programs Layer -

It is the outermost layer that executes the given external applications. UNIX distributions typically come with several useful applications programs as standard.

For Example: emacs editor, StarOffice, xv image viewer, g++ compiler etc.



One example of how the shell and kernel work together is copying a file.

- If you want to copy a file named "file1" and name the copy "file2", you would enter "cp file1 file2" at the command line.
- The shell will search for the program "cp" then tell the kernel to run that program on "file 1" and name the output "file 2".
- When the copying is finished, the shell returns you to the prompt and awaits more commands.



Linux vs Unix



Linux	Unix
The source code of Linux is freely available to its users	The source code of Unix is not freely available general public
It has graphical user interface along with command line interface	It only has command line interface
Linux OS is portable, flexible, and can be executed in different hard drives	Unix OS is not portable
Different versions of Linux OS are Ubuntu, Linux Mint, RedHat Enterprise Linux, Solaris, etc.	Different version of Unix are AIS, HP-UX, BSD, Iris, etc.
The file systems supported by Linux are as follows: xfs, ramfs, vfat, cramfsm, ext3, ext4, ext2, ext1, ufs, autofs, devpts, ntfs	The file systems supported by Unix are as follows: zfs, js, hfx, gps, xfs, vxfs

System Call



Marwadi
University
Marwadi Chandarana Group



System call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.

A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel.

System call **provides** the services of the operating system to the user programs via Application Program Interface(API).

It provides an interface between a process and operating system to allow user-level processes to request services of the operating system.

System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Services Provided by System Calls :

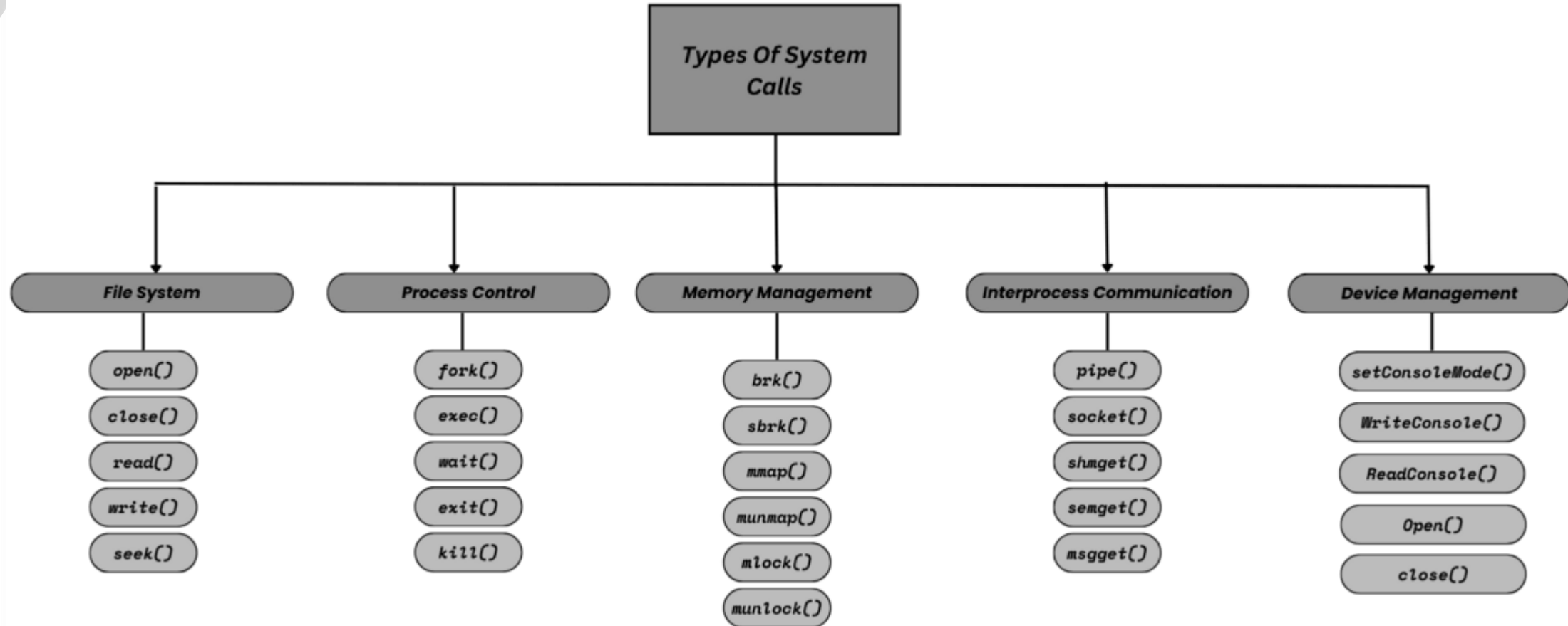


Marwadi
University
Marwadi Chandarana Group



- Process creation and management
- Main memory management
- File Access, Directory and File system management Device handling(I/O)
- Protection
- Networking, etc.

Types of System Calls :



Types of System Calls :



File Management System Calls:

- **open():** Opens a file for reading or writing. A file could be of any type like text file, audio file etc.
- **read():** Reads data from a file. Just after the file is opened through open() system call, then if some process want to read the data from a file, then it will make a read() system call.
- **write():** Writes data to a file. Whenever the user makes any kind of modification in a file and saves it, that's when this is called.
- **close():** Closes a previously opened file.
- **seek():** Moves the file pointer within a file. This call is typically made when we the user tries to read the data from a specific position in a file. For example, read from line – 47. Then the file pointer will move from line 1 or wherever it was previously to line-47.

Types of System Calls :



Process Control System Calls:

- **fork():** Creates a new process (child) by duplicating the current process (parent). This call is made when a process makes a copy of itself and the parent process is halted temporarily until the child process finishes its execution.
- **exec():** Loads and runs a new program in the current process and replaces the current process with a new process. All the data such as stack, register, heap memory everything is replaced by a new process and this is known as overlay. For example, when you execute a java byte code using command – java “filename”. Then in the background, exec() call will be made to execute the java file and JVM will also be executed.
- **wait():** The primary purpose of this call is to ensure that the parent process doesn't proceed further with its execution until all its child processes have finished their execution. This call is made when one or more child processes are forked.
- **exit():** It simply terminates the current process.
- **kill():** This call sends a signal to a specific process and has various purpose including – requesting it to quit voluntarily, or force quit, or reload configuration.

Types of System Calls :



fork() vs exec():

Features	Fork()	Exec()
Definition	It is a command that allows a process to copy itself.	It is a command that makes a new process by replacing the existing process.
Address Space	The parent and child processes are in separate address spaces in the fork().	The child address space replaces the parent address space in the exec().
Parent Process	There is a child process and a parent process after calling the fork() function.	There is only a child process after calling the exec().
Result	The fork() makes a child's process equal to the parent's process.	The exec() makes a child process and replaces it with the parent process.

Types of System Calls :



Memory Management System Calls:

- **brk():** Changes the data segment size for a process in HEAP Memory. It takes an address as argument to define the end of the heap and explicitly sets the size of HEAP.
- **sbrk():** This call is also for memory management in heap, it also takes an argument as an integer (+ve or -ve) specifying whether to increase or decrease the size respectively.
- **mmap(): Memory Map** – it basically maps a file or device into main memory and further into a process's address space for performing operations. And any changes made in the content of a file will be reflected in the actual file.
- **munmap():** Unmaps a memory-mapped file from a process's address space and out of main memory
- **mlock() and unlock():** memory lock defines a mechanism through which certain pages stay in memory and are not swapped out to the swap space in the disk. This could be done to avoid page faults. Memory unlock is the opposite of lock, it releases the lock previously acquired on pages.

Types of System Calls :



Inter Process Communication (IPC) System Calls:

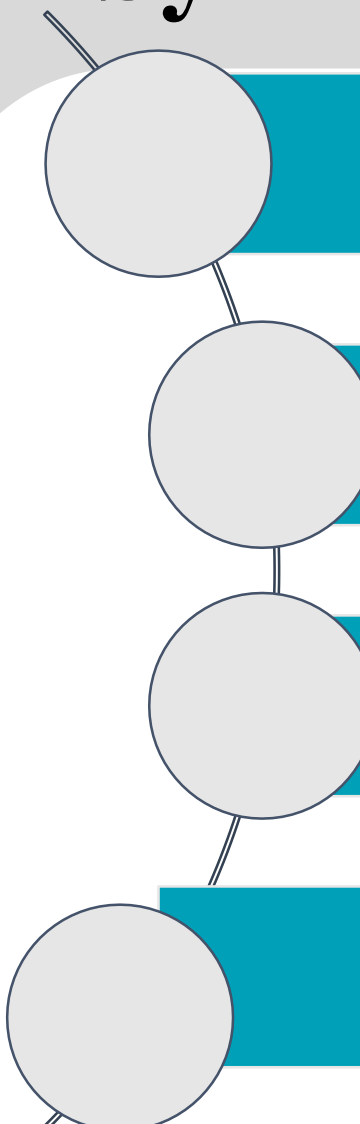
- **pipe():** Creates a unidirectional communication channel between processes. For example, a parent process may communicate to its child process through a pipe making a parent process as input source of its child process.
- **socket():** Creates a network socket for communication. Processes in same or other networks can communicate through this socket, provided that they have necessary network permissions granted.
- **shmget():** It is short for – ‘shared-memory-get’. It allows one or more processes to share a portion of memory and achieve interprocess communication.
- **semget():** It is short for – ‘semaphore-get’. This call typically manages the coordination of multiple processes while accessing a shared resource that is, the critical section.
- **msgget():** It is short for – ‘message-get’. IPC mechanism has one of the fundamental concept called – ‘message queue’ which is a queue data structure inside memory through which various processes communicate with each other. This message queue is allocated through this call allowing other processes a structured way of communication for data exchange purpose.

Types of System Calls :

Device Management System Calls:

- **SetConsoleMode():** This call is made to set the mode of console (input or output). It allows a process to control various console modes. In windows, it is used to control the behaviour of command line.
- **WriteConsole():** It allows us to write data on console screen.
- **ReadConsole():** It allows us to read data from console screen (if any arguments are provided).
- **open():** This call is made whenever a device or a file is opened. A unique file descriptor is created to maintain the control access to the opened file or device.
- **close():** This call is made when the system or the user closes the file or device.

System Calls: Interrupts



An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention

The processor responds by suspending its current activities, saving its state, and executing a function needed by interrupt

This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.

There are two types of interrupts: hardware interrupts and software interrupts.

System Calls: Interrupts



Hardware interrupt

pressing a key on the keyboard or moving the mouse triggers hardware interrupts that cause the processor to read the keystroke or mouse position.

Software interrupt

errors or events occurring during program execution that are cannot be handled within the program itself.

For example, if the processor's arithmetic logic unit is commanded to divide a number by zero.

Hardware vs Software Interrupt



SR.NO.	Hardware Interrupt	Software Interrupt
1	Hardware interrupt is an interrupt generated from an external device or hardware.	Software interrupt is the interrupt that is generated by any internal system of the computer.
2	It do not increment the program counter.	It increment the program counter.
3	Hardware interrupt can be invoked with some external device such as request to start an I/O or occurrence of a hardware failure.	Software interrupt can be invoked with the help of INT instruction.
4	It has lowest priority than software interrupts	It has highest priority among all interrupts.
5	Hardware interrupt is triggered by external hardware and is considered one of the ways to communicate with the outside peripherals, hardware.	Software interrupt is triggered by software and considered one of the ways to communicate with kernel or to trigger system calls, especially during error or exception handling.
6	It is an asynchronous event.	It is synchronous event.
7	Hardware interrupts can be classified into two types they are: 1. Maskable Interrupt. 2. Non Maskable Interrupt.	Software interrupts can be classified into two types they are: 1. Normal Interrupts. 2. Exception
8	Keystroke depressions and mouse movements are examples of hardware interrupt.	All system calls are examples of software interrupts



Thank You