OPERATING SYSTEM (ENCT254)

INTRODUCTION

Chapter-1 Getting Started..

Er. Sagar Panta
Asst. Lecturer, Dept. of Elx. & Computer Engineering
National College of Engineering
Email:- sagar@nce.edu.np

Introduction to Operating systems

- Operating system is a system software that acts as an interface between computer hardware and Computer Users and controls the execution of all kinds of programs
- It provides an environment to the user so that, the user can perform its task in convenient and efficient way
- The OS performs basic tasks such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices.
- It is responsible for managing the hardware components of the system and allows users to interact with the computer without needing to understand the complexities of the hardware.

Introduction to Operating systems

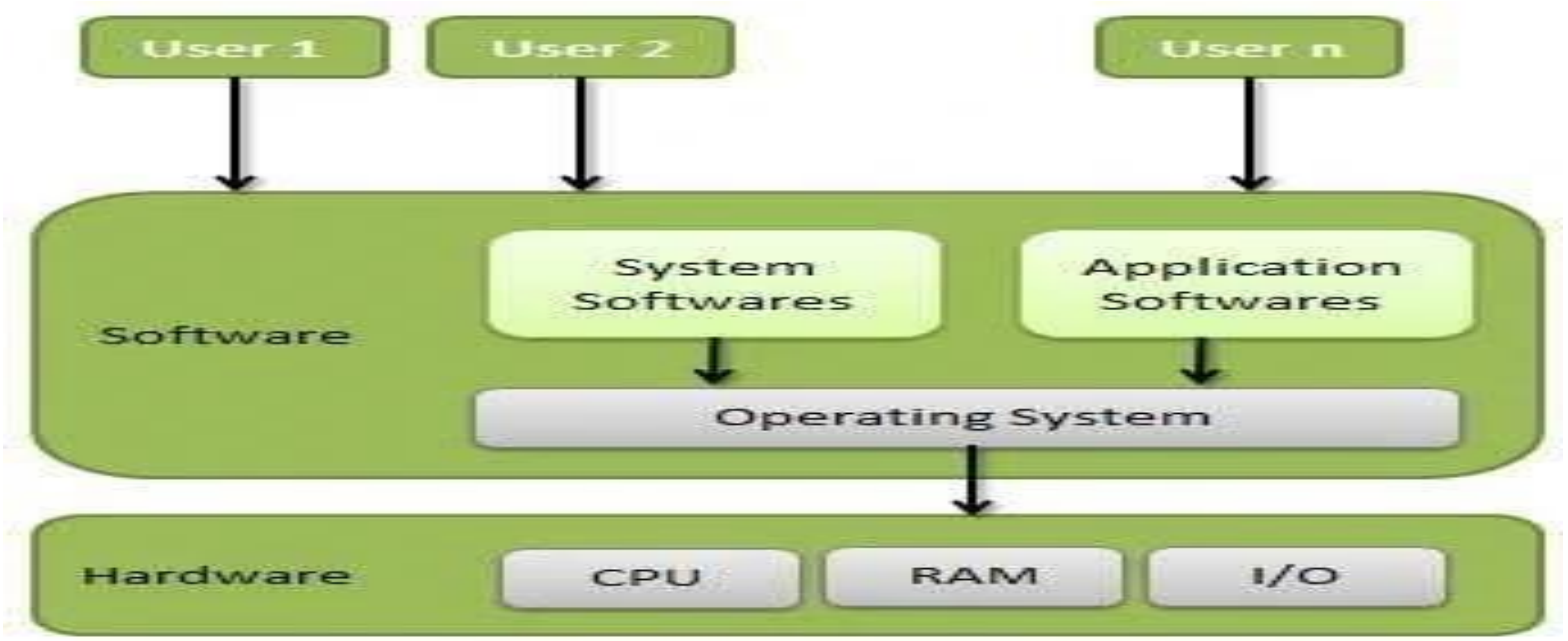
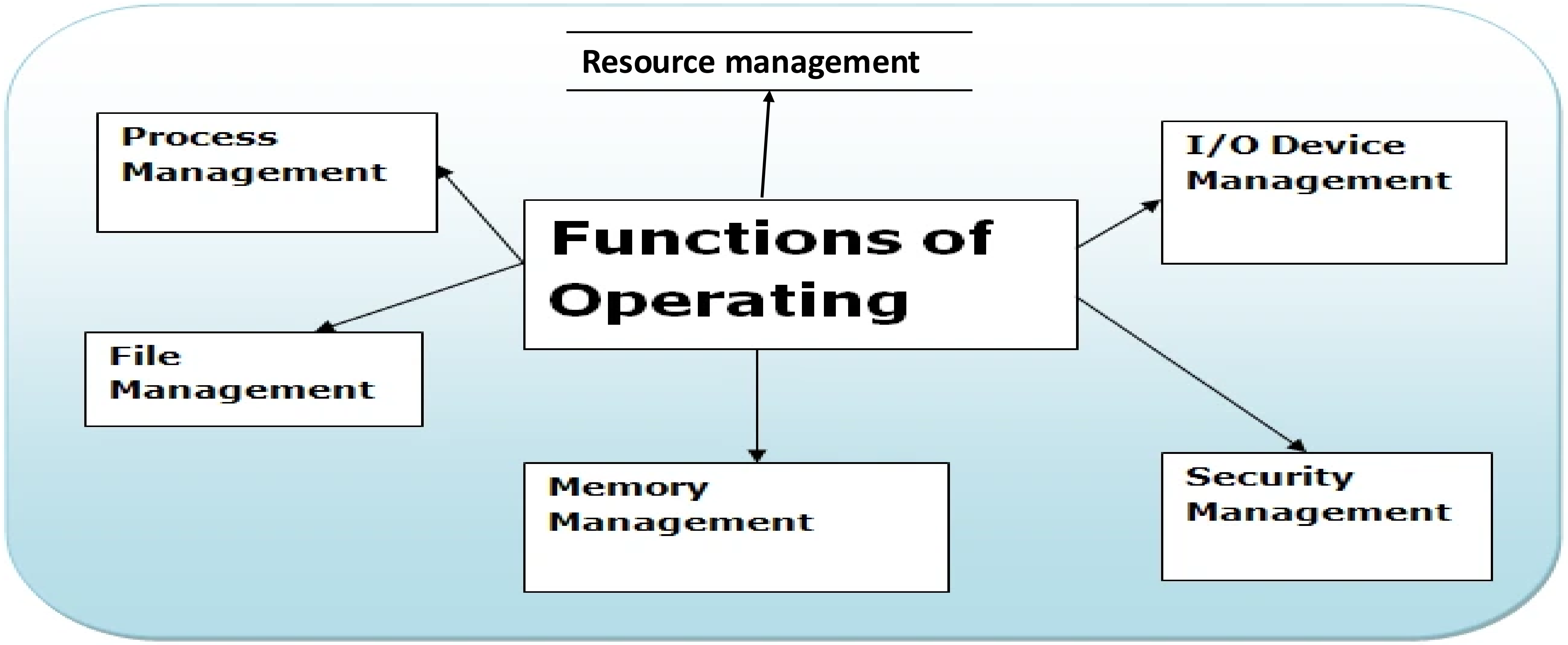


Fig: Operating System Overview

Functions of an OS



OS as an Extended Machine

- OS hides hardware complexity and gives easy-to-use features (e.g., virtual memory instead of physical addresses).
- As an extended machine, the operating system hides the complexity of hardware and provides users with a virtual machine that is easier to interact with.
- Instead of managing registers, memory addresses, or disk sectors directly, users and application developers interact with higher-level abstractions provided by the OS, such as files, virtual memory, and processes.
- This simplifies application development and user operations.
- Virtual machine encompasses:
  - Providing execution environment
    - Process creation, management, control and termination
    - File manipulation, I/O operation
    - Interrupt handling, language support
  - Error detection and handling
  - Protection and security

OS as a Resource Manager

- OS distributes CPU, memory, files, and devices to programs fairly and efficiently.
- As a resource manager, the OS manages the allocation and deallocation of system resources like CPU time, memory space, disk space, and I/O devices.
- It ensures efficient utilization of resources, maintains fairness among processes, avoids deadlocks, and ensures that system integrity is preserved.
- The OS schedules processes, manages memory hierarchies, handles I/O buffering, and enforces protection and security mechanisms.

History of operating system

- GENERATIONS

1945-1955	: Vacuum Tubes and Plug Boards
1955-1965	: Transistors and Batch Systems
1956-1980	: IC’s and Multiprogramming
1980-Present	: Personal Computers

First Generation (1940s–1950s):

- No OS existed.
- Programmers interacted directly with the hardware using switches and plugboards.

Second Generation (1950s–early 1960s):

- Batch processing systems emerged.
- Jobs were prepared on punch cards and submitted for execution without user interaction.
- Input/output was slow and sequential.

Third Generation (mid-1960s–1970s):

- Introduction of multiprogramming and time-sharing.
- Systems like IBM's OS/360 and MULTICS emerged.
- OS features such as memory management and file systems matured.

Fourth Generation (1980s–1990s):

- Personal computers became widespread.
- User-friendly interfaces were developed.
- Operating systems like MS-DOS, Windows, and Mac OS became popular.
- Networking support started integrating into OSs.

Modern Era (2000s–present):

- Rise of mobile operating systems (Android, iOS), distributed systems, real-time OS, embedded systems, cloud computing, virtualization, and advanced security features.
- Modern OSs are multitasking, multi-user, and provide strong GUI and networking capabilities.

Assignment:

- For **OS history** please refer to “Tanenbaum Modern OS Book”[Page No:-33-45]

Types of operating system: Mainframe

- At the high end are the operating systems for the mainframes, those room-sized computers still found in major corporate data centers.
- These computers distinguish themselves from personal computers in terms of their I/O capacity.
- A mainframe with 1000 disks and thousands of gigabytes of data is not unusual: a personal computer with these specifications would be odd indeed.
- Mainframes are also making something of a comeback as high-end Web servers, servers for large-scale electronic commerce sites, and servers for business-to-business transactions.

Types of operating system: Mainframe[Cont’d]

- The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need prodigious amounts of I/O.
- They typically offer three kinds of services: batch, transaction processing, and timesharing.
    - A batch system is one that processes routine jobs without any interactive user present. Claims processing in an insurance company or sales reporting for a chain of stores is typically done in batch mode.
    - Transaction processing systems handle large numbers of small requests, for example, check processing at a bank or airline reservations. Each unit of work is small, but the system must handle hundreds or thousands per second.
    - Timesharing systems allow multiple remote users to run jobs on the computer at once, such as querying a big database.
- These functions are closely related: mainframe operating systems often perform all of them.
- An example mainframe operating system is OS/390, a descendantof OS/360.

Types of operating system: Server

- One level down are the server operating systems.
- They run on servers, which are either very large personal computers, workstations, or even mainframes.
- They serve multiple users at once over a network and allow the users to share hardware and software resources.
- Servers can provide print service, file service, or Web service.
- Internet providers run many server machines to support their customers and Web sites use servers to store the Web pages and handle the incoming requests.
- Typical server operating systems are UNIX and Windows 2000. Linux is also gaining ground for servers.

Types of operating system: Personal

- The next category is the personal computer operating system.
- Their job is to provide a good interface to a single user.
- They are widely used for word processing, spreadsheets, and Internet access.
- Common examples are Windows 98, Windows 2000, the Macintosh operating system, and Linux.
- Personal computer operating systems are so widely known that probably little introduction is needed.
- In fact, many people are not even aware that other kinds exist.

Types of operating system: Smartphone and Handheld

- Mobile Os’s are optimised for **touch input**, **limited power**, **small screens**, and **wireless connectivity**.
- They include features for **sensor access**, **battery saving**, and **app sandboxing**.
- They have ability to support **mobile apps**, **push notifications**, and **location services**, while ensuring **security** and **resource efficiency**.
- **Use Case:** Smartphones, tablets, PDAs
- **Examples:** Android, iOS

Types of operating system: IOT and Embedded

- Embedded systems run on the computers that control devices that are not generally thought of as computers, such as TV sets, microwave ovens, and mobile telephones.
- These often have some characteristics of real-time systems but also have size, memory, and power restrictions that make them special. Examples of such operating systems are PalmOS and Windows CE (Consumer Electronics).
- These OSs run on special-purpose hardware with limited resources.
- **Embedded OSs** are designed for efficiency, reliability, and real-time performance.
- They often reside in firmware and perform dedicated tasks.
- **IoT OSs** extend this by enabling networked communication, remote control, and sensor integration.

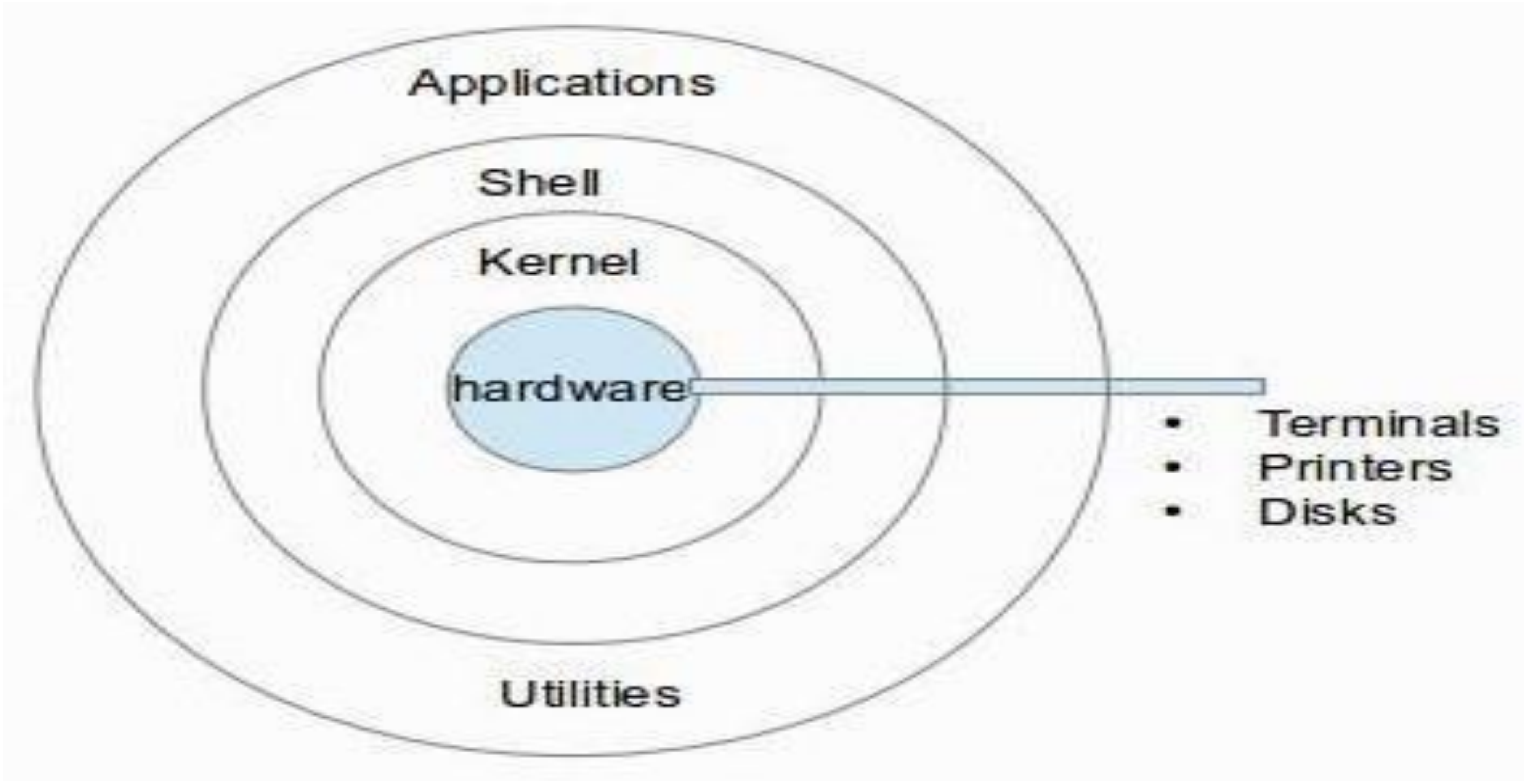
Types of operating system: Real-time

- These systems are characterized by having time as a key parameter.
- For example, in industrial process control systems, real-time computers have to collect data about the production process and use it to control machines in the factory.
- Often there are hard deadlines that must be met. For example, if a car is moving down an assembly line, certain actions must take place at certain instants of time, if a welding robot welds too early or too late, the car will be ruined.
- If the action absolutely must occur at a certain moment (or within a certain range), we have a hard real-time system.
- Another kind of real-time system is a soft real-time system, in which missing an occasional deadline is acceptable. Digital audio or multimedia systems fall in this category.
- VxWorks and QNX are well-known real-time operating systems.

Types of operating system: Smart-card

- The smallest operating systems run on smart cards, which are credit card-sized devices containing a CPU chip.
- They have very severe processing power and memory constraints.
- Some of them can handle only a single function, such as electronic payments, but others can handle multiple functions on the same smart card. Often these are proprietary systems.
- Some smart cards are Java oriented. What this means is that the ROM on the smart card holds an interpreter for the Java Virtual Machine (JVM). Java applets (small programs) are downloaded to the card and are interpreted by the JVM interpreter. Some of these cards can handle multiple Java applets at the same time, leading to multiprogramming and the need to schedule them.
- Resource management and protection also become an issue when two or more applets are present at the same time. These issues must behandled by the (usually extremely primitive) operating system present on the card.
- **Examples:** Java Card OS, MULTOS

Operating System Components



Kernel

- Kernel is the central module of an OS
- It is the part of the operating system that loads first, and it remains in main memory
- Because it stays in memory, it should be as small as possible while still providing all the essential services required by other parts of the OS and applications
- Then the kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system

Kernel[Cont’d]

- **Typically, the kernel is responsible for:-**
  - Memory management
  - Process and task management
  - Disk management.

- The kernel connects the system hardware to the application software.

- Every operating system has a kernel.

- For example, the Linux kernel is used numerous operating systems including Linux, FreeBSD, Android and others.

Shell

- Acts as an interface between the user and the kernel.

- Can be command-line (e.g., bash, zsh) or graphical (e.g., GNOME shell, Windows Explorer).

- When a user gives his command for performing any operation, then the request will go to the shell parts, the shell parts is also called as the interpreter which translate the human program into the machine language and then the request will be transferred to the kernel that means shell is just as the interpreter of the commands which converts the request of the user into the machine language.

Utilities

- Utilities are system software programs that provide specialized services and tools to users or the OS itself, often used for maintenance, monitoring, and performance tuning.
- E.g: File managers, text editors, and graphical interfaces.
- **Common Examples**:
  - **File Management Utilities** (e.g., cp, mv, rm): Support user-level file operations.
  - **Disk Utilities**: Tools like fsck (file system check), defragmentation tools.
  - **Process Management**: Commands like top, ps, kill offer information or control over processes.
  - **Backup and Compression**: Tools like tar, gzip, rsync.
  - **Security Utilities**: Firewall configuration tools, user access management, audit trails.
- **Role in OS**:
  - Extend OS functionality.
  - Bridge between users and kernel-level functions.
  - Provide scripting capability via shell to automate repetitive tasks.

Applications

- Application programs are user-developed software that runs in user mode and uses OS services to perform specific tasks.
- **Examples**: Word processors, media players, compilers, browsers.
- **Relationship with OS**:
  - Operate using system calls and APIs provided by OS.
  - Depend on OS for memory, I/O, and process management.

Note: While not strictly a component of the OS, application programs rely heavily on OS services.

System calls

- System calls provide an interface between user programs and operating system(kernel).

- System call is a programmatic way in which a computer program request a services from the kernel of the operating system.

- The kernel system can only be accessed using system calls. System calls are required for any programs that use resources.

Why do you need system calls in Operating System?

- It is must require when a file system wants to create or delete a file.
- Network connections require the system calls to sending and receiving data packets.
- If you want to read or write a file, you need to system calls.
- If you want to access hardware devices, including a printer, scanner, you need a system call.
- System calls are used to create and manage new processes.

Services Provided by System Calls:

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

User mode

- In an operating system, **User Mode** and **Kernel Mode** define how programs access system resources.
- They differ mainly in their level of access.
- **Kernel Mode** has full control over hardware and system operations, while **User Mode** has limited access to protect the system. This separation ensures system stability and security.
- User Mode is a restricted mode where regular applications run. Programs here cannot directly access hardware or system memory. Instead, they must request services from the operating system.
- **Real-World Example:** Think of User Mode as a visitor at a museum. You can look at the exhibits but cannot touch or move anything. If you need something, you ask the staff.
- **Key Points:**
  - ➢ Limited access to hardware and memory.
  - ➢ Applications run here.
  - ➢ If a program crashes, it doesn't affect the whole system.

Kernel mode

- Kernel Mode is where the operating system core and essential drivers operate. Programs running in this mode have complete access to the system's memory and hardware.
- **Real-World Example:** Kernel Mode is like the museum staff. They can go behind the scenes, fix issues, and control everything because they have full access.
- **Key Points:**
  - ➢ Full access to hardware and memory.
  - ➢ Only trusted system processes run here.
  - ➢ Errors in this mode can crash the entire system.

User Mode vs. Kernel Mode

Feature	User Mode	Kernel Mode
Access level	Limited	Full
Example Programs	Games, Browsers	Device drivers, OS Kernels
System Stability	More stable(isolated crashes)	Less stable(Critical crashes)
Direct Hardware Access	No	Yes

Types of OS Kernel

- Monolithic
- Micro
- Nano
- Layered
- Hybrid
- Exo-kernel

Monolithic

- The monolithic operating system is a very basic operating system in which file management, memory management, device management, and process management are directly controlled within the kernel. The kernel can access all the resources present in the system. In monolithic systems, each component of the operating system is contained within the kernel.
- The monolithic operating system is also known as the monolithic kernel. This is an old operating system used to perform small tasks like batch processing and time-sharing tasks in banks.
- The monolithic kernel acts as a virtual machine that controls all hardware parts.
- It is simple to design and implement because all operations are managed by kernel only, and layering is not needed.
- As services such as memory management, file management, process scheduling, etc., are implemented in the same address space, the execution of the monolithic kernel is relatively fast as compared to normal systems. Using the same address saves time for address allocation for new processes and makes it faster.

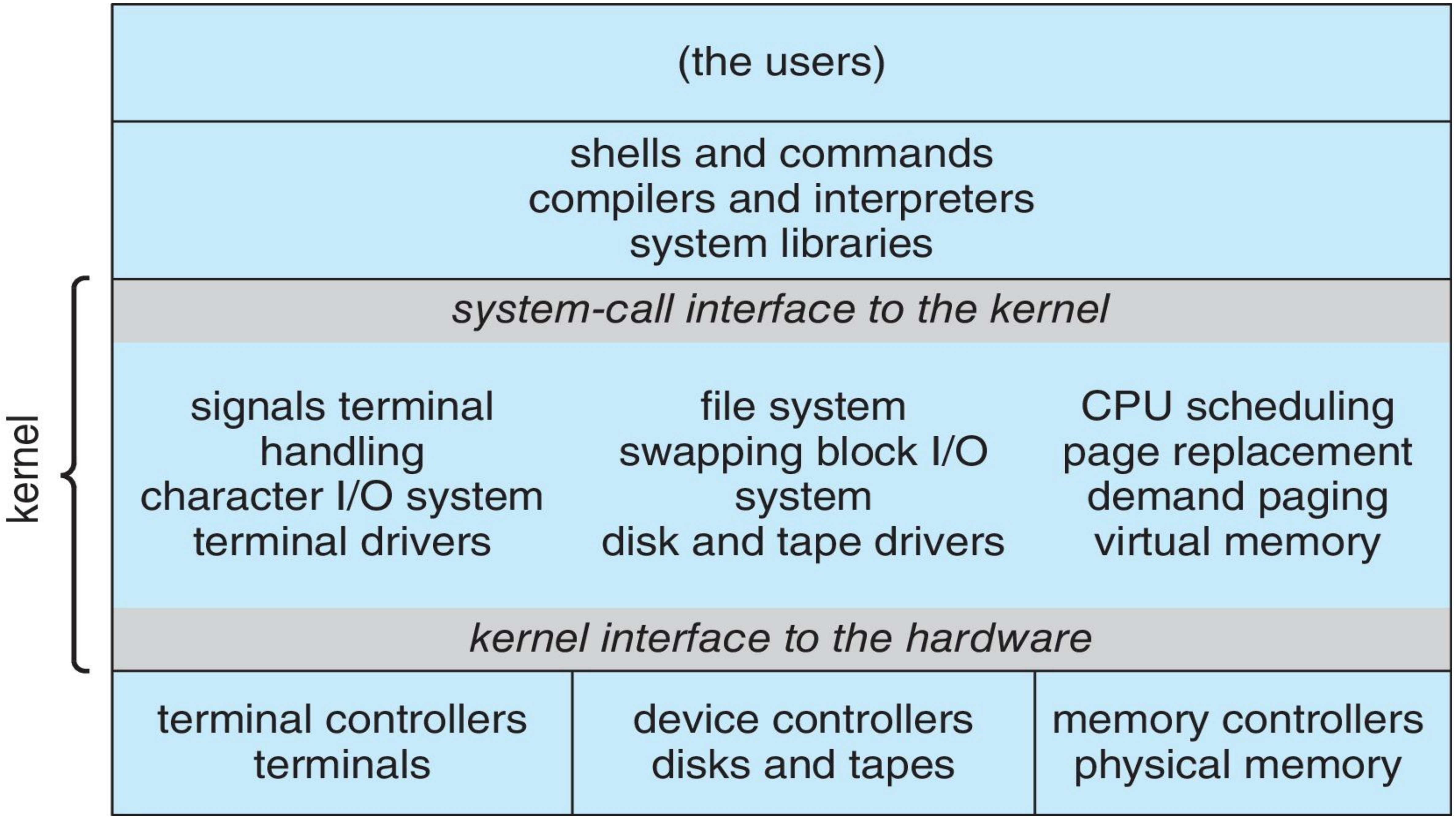


Fig: Monolithic Kernel

Advantages

- Execution of processes is faster.
- Since it is a single piece of software, its sources and compiled form are smaller.

Disadvantages

- If a new service is to be added, the whole operating system needs to be modified.
- This type of Kernel is not portable. For each different architecture, the Kernel needs to be rewritten.
- If a service generates an error, the whole system may crash.
- Its size is larger and hence difficult to manage.

Micro

- This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs.

- This results in a smaller kernel called the micro-kernel.

- Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified.

- Thus, it is more secure and reliable as if a service fails, then rest of the operating system remains untouched.

- Mac OS is an example of this type of OS.

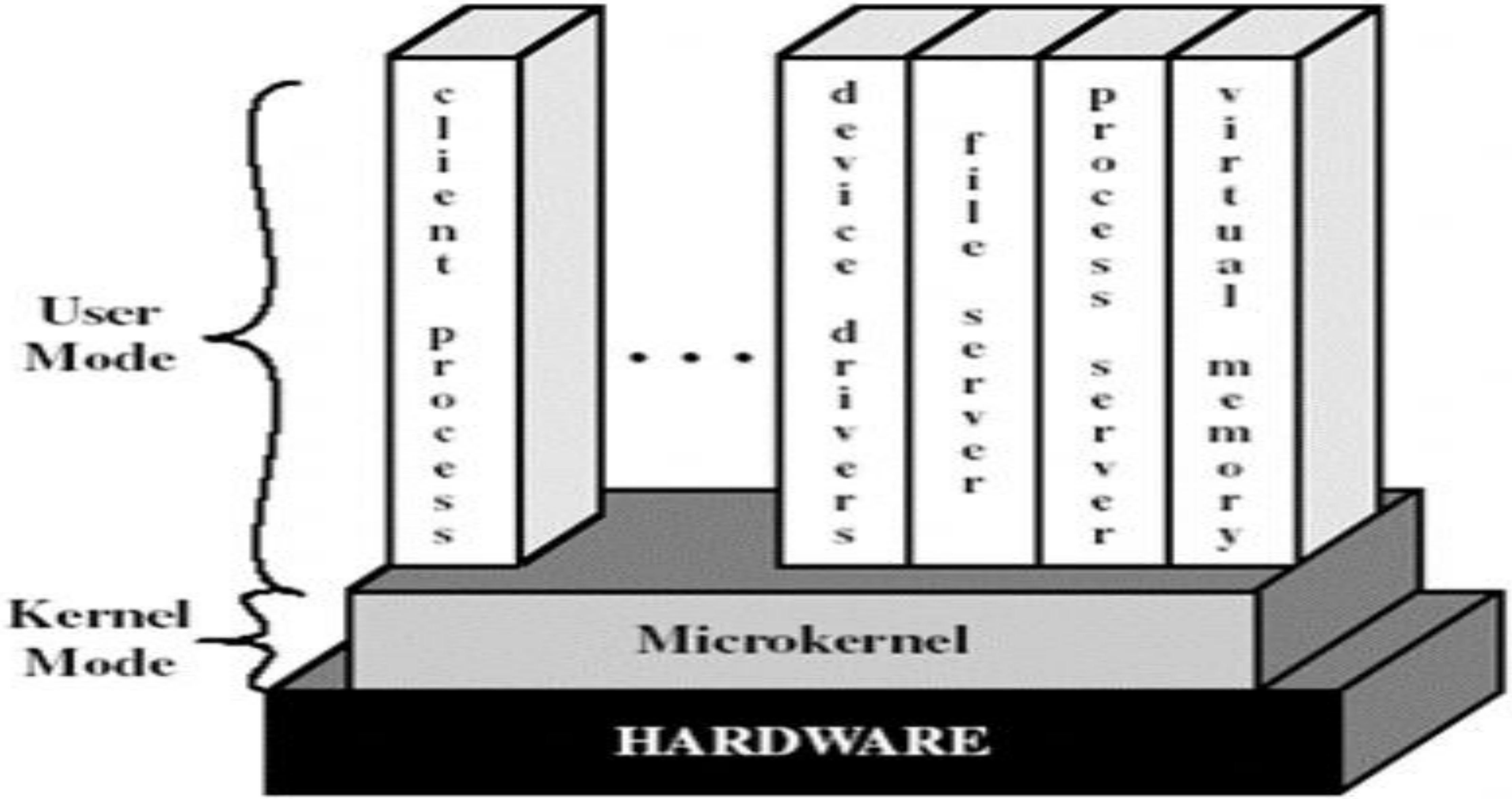


Fig: Micro-kernel Structure

Advantages

- Easily manageable
- New services can be easily added without modifying the existing OS.
- If a kernel process crashes, a system crash can be prevented by restarting the error-caused services. Hence it is more stable.

Disadvantages

- System performance is reduced due to the increased requirement of software for interfacing.
- Process management in microkernel is complicated.

Monolithic kernel	Microkernel.
In a monolithic kernel, user services and kernel services are kept in the same address space.	In a microkernel, user services and kernel services are kept in separate address space.
The monolithic kernel is larger in size.	The microkernel is smaller in size.
Execution speed is faster in the case of a monolithic kernel.	Execution speed is slower in the case of a microkernel.
The monolithic kernel is hard to extend.	The microkernel is easily extendable.
The whole system will crash if one component fails.	If one component fails, it does not affect the working of the microkernel.
Fewer lines of code need to be written for a monolithic kernel.	More lines of code need to be written for a microkernel.
Debugging and management are complex in the case of a monolithic kernel.	Debugging and management are more straightforward as the kernel is smaller in size.
Monolithic OS is easier to design and implement.	Microkernels are complex to design.
Examples of this OS include Unix, Linux, OS/360, OpenVMS, Multics, AIX, BSD etc.	Examples of this OS include QNX, L4Linux, Mac OS X, Symbian, Singularity etc.

Nano

- In **Nanokernel**, the complete code of the Kernel is very small.
- This means that the code getting executed in the privileged mode of hardware is very small.
- In Nanokernel, the term nano defines the support for a nanosecond clock resolution.
- **Example:** EROS, etc.

Advantages

- It can provide hardware abstraction even with a very small size.

Disadvantages

- It lacks the system services.

Layered

- In this type of structure, OS is divided into layers or levels. The hardware is on the bottom layer (layer 0), while the user interface is on the top layer (layer N).

- These layers are arranged in a hierarchical way in which the top-level layers use the functionalities of their lower-level levels.

- It was designed to improve existing structures such as the Monolithic (UNIX) and Simple structures (MS-DOS).

- For example, this tiered method is used in the Windows NT operating system.

Layered Structure of the THE OS

- A layered design was first used in THE operating system. Its six layers are as follows:

layer 5:	user programs
layer 4:	buffering for input and output
layer 3:	Process management
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware

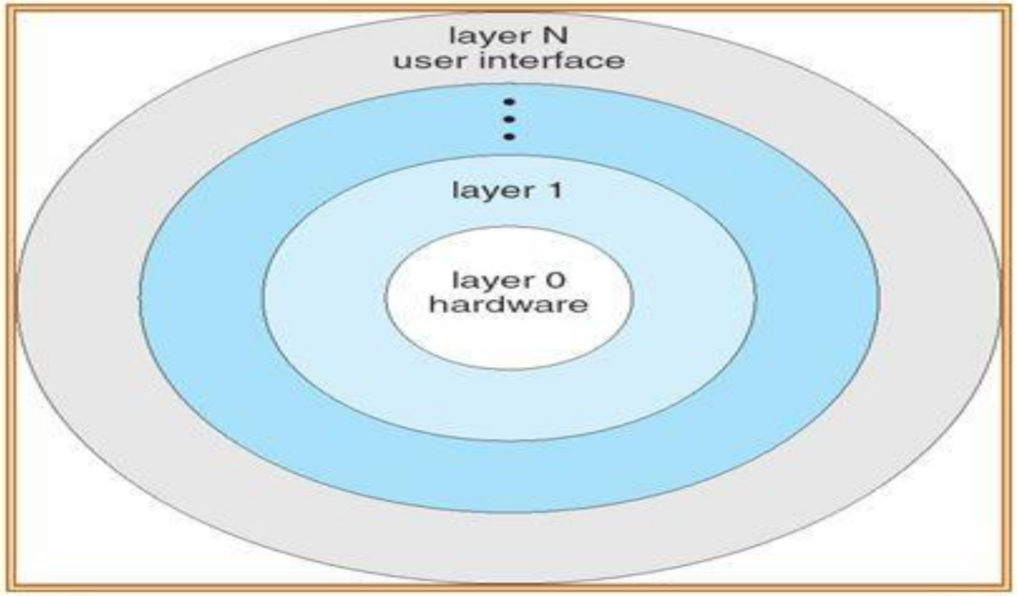


Fig: Layered kernel structure

Key characteristics of Layered Structure

- Each layer is responsible for a specific set of tasks. This makes it easier to understand, develop, and maintain the operating system.
- Layers are typically arranged in a hierarchy. This means that each layer can only use the services provided by the layers below it.
- Layers are independent of each other. This means that a change to one layer should not affect the other layers.

Advantages:

- **Modularity:** This architecture promotes modularity because each layer only does its assigned duties.
- **Easy debugging:** It is relatively simple to debug because the layers are discrete. If a mistake happens in the CPU scheduling layer, the developer can only debug that layer, as opposed to a Monolithic system where all services are present at the same time.
- **Easy update:** A change made to one layer will have no effect on the other layers.
- **No direct access to hardware:** The hardware layer is the design's innermost layer. So, unlike the Simple system, where the user has direct access to the hardware, a user can use hardware services but not directly modify or access it.
- **Abstraction:** Every layer is focused on its own set of tasks. As a result, the other layers' functions and implementations are abstract.

Disadvantages:

- In Layered Structure, layering causes degradation in performance.
- It takes careful planning to construct the layers since higher layers only utilize the functions of lower layers.
- There can be some performance overhead associated with the communication between layers. This is because each layer must pass data to the layer above it.

Hybrid

- Hybrid Kernels are a combination of both **Monolithic kernels and Microkernels**.
- It combines the speed of Monolithic Kernel with the modularity of Microkernels.
- It is similar to a microkernel, but it also includes some additional code in kernel space to enhance the system performance.
- Hybrid Kernel allows to run some services like network stack in kernel space, but it still allows kernel code like device drivers to run as servers in user space.
- **Example:** Windows NT, BeOS, Netware, etc.

Advantages

- No requirements for a reboot for testing
- Third-party technologies can be integrated rapidly.

Disadvantages

- Possibility of more bugs due to more interfaces to pass-through
- It is challenging for the administrators to maintain the modules.

Exo-kernel

- In **Exokernel**, resource protection is separated from the management part, which allows us to perform application-specific customization.
- It follows the end-to-end principle.
- It has the fewest hardware abstractions possible and allocates the physical resources to applications.

Advantages

- It provides better support for application control.
- Improved performance of applications.
- Applications are allowed to have their optimized memory management system

Disadvantages

- The design of exokernel interfaces is complex.

SERVICES	WINDOWS	UNIX	
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()	
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()	
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()	
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()	
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()	
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()	

Shell commands

- A command-line interface that allows users to interact with the OS.
- Instructions given to the shell to perform file and process-related operations.
- **Types of Shells**: Bourne Shell (sh), C Shell (csh), Korn Shell (ksh), Bourne Again Shell (bash)
- **Common Shell Commands:**
  - **File Operations**: ls, cd, pwd, touch, mkdir, rm
  - **Process Management**: ps, kill, jobs, bg, fg, nice
  - **Text Processing**: cat, more, less, head, tail, grep, cut, sort, uniq
  - **I/O Redirection and Pipes**: >, >>, <, |
  - **Permission Management**: chmod, chown, umask
  - **System Information**: whoami, date, uptime, hostname

Shell Programming

- Writing scripts using shell commands, control flow constructs (loops, if-else), variables, and functions to automate tasks.
- **Shell script**: a file containing a series of shell commands.

Shell Script Basics:

- Start with *#!/bin/bash*
- Variables: *name=”Ram"* → Access: *$name*
- Input: *read variable*
- Output: *echo "Hello $variable"*

POSIX Standard

- **POSIX (Portable Operating System Interface)** is a set of IEEE standards designed to provide compatibility between UNIX-like operating systems.
- It defines standard APIs for system calls, utilities, and shell scripts.
- POSIX ensures that software written on one compliant OS can run on another with little or no modification.
- It covers:
  - ➢ Process creation and control
  - ➢ File and directory operations
  - ➢ Threads and signals
  - ➢ Input/output mechanisms
  - ➢ Standard command-line utilities
- POSIX compliance is essential for writing portable and interoperable applications.
- Linux, macOS, and many versions of UNIX are POSIX-compliant.

Bootloader

- A **bootloader** is a small program that **loads the operating system** (or its kernel) into memory and starts its execution.
- It is the **first software** that runs after a computer is powered on or restarted.
- Bootloader is typically stored in the **boot sector** of a storage device (e.g., MBR of a hard drive or EFI system partition for UEFI).
- In MBR-based systems, it resides in the **first 512 bytes** of the disk.

Booting Process Overview

- **BIOS/UEFI Execution**: When the system powers on, the BIOS/UEFI firmware performs **POST** (Power-On Self-Test) and identifies bootable devices.

- **Load Bootloader**: BIOS/UEFI loads the bootloader into RAM from the boot device.

- **Bootloader Execution**:
  - Bootloader **loads the OS kernel** into memory.
  - Transfers control to the kernel to continue the system boot-up.

Functions of Bootloader

- **Initialize hardware minimally** (keyboard, display, etc.)
- **Identify and load OS kernel**
- **Provide options to boot different OSes** (multiboot systems)
- **Pass parameters to the kernel**
- Support for loading **initramfs/initrd** (temporary file system)

Types of Bootloaders

- **GRUB (GNU GRUB Bootloader)** – Used in most Linux distributions
- **LILO (LInux LOader)** – Older bootloader, now mostly replaced
- **Windows Boot Manager** – Used in Windows OS
- **Syslinux / PXELINUX** – Used in lightweight or network boot environments

MBR/GPT

- **MBR (Master Boot Record),** Traditional partitioning method with a bootloader and a partition table. Located at the first sector (512 bytes) of the storage device. Supports up to 2 TB and four primary partitions.

- **GPT (GUID Partition Table),** Modern partitioning method supporting larger disks and more partitions. Includes redundancy and CRC for integrity.

UEFI and legacy boot

UEFI(Unified Extensible Firmware Interface),

- Replaces legacy BIOS.
- Provides a faster, secure boot process with support for larger disks and GUI configuration.

Legacy boot,

- Traditional BIOS-based boot process.
- Still supported for backward compatibility but lacks features like secure boot and large disk support.