HOMEWORK 4

Q-1. Explain the role of the CPU in an operating system. How does the OS manage CPU resources to ensure efficient processing?

The **Central Processing Unit (CPU)** is the heart of a computer system, responsible for executing instructions, performing calculations, and managing data flow. However, with multiple programs and tasks running simultaneously, the CPU cannot manage everything on its own. This is where the **Operating System (OS)** plays a crucial role in managing CPU resources to ensure smooth and efficient operation.

The OS uses **process scheduling** to decide which task should access the CPU and when. Since the CPU can typically handle only one task per core at a time, the OS uses algorithms like **Round Robin**, **First-come First-Served**, or **Priority Scheduling** to allocate CPU time fairly and efficiently among processes. These techniques help maintain system responsiveness and prevent any single process from dominating the CPU.

To allow **multitasking**, the OS performs **context switching** - saving the state of one process and loading another. This rapid switching allows multiple applications to run seemingly at the same time. Each process gets a fixed **time slice**, ensuring all processes get regular access to CPU time.

The OS also handles **interrupts**, which are signals indicating that immediate attention is required, such as input from a keyboard or a request from another device. The CPU temporarily halts its current task to address these high-priority events.

In **multi-core** systems, the OS performs **load balancing**, distributing tasks across multiple CPUs or cores to maximize efficiency. Additionally, the OS **monitors CPU performance** in real-time, making adjustments as needed to optimize resource usage.

Overall, the OS acts as the CPU’s manager, coordinating access, improving responsiveness, and maintaining system stability.

Q-2. Compare and contrast primary and secondary storage. How does an operating system manage these different types of storage?

Comparison b/w primary and secondary storage:

Aspect	Primary Storage (RAM)	Secondary Storage (HDD/SSD)
Definition	Temporary memory for currently active data/programs	Permanent memory for long-term data storage
Volatility	Volatile (data lost whener off)	Non-volatile (*adta* retainden n)
Speed	Very fast access	Slower compared to primary storage
Capacity	Limited (typically in GBs)	Largar (typically in T .s)
OS management technique	Yes	Store files, OS, applications
OS Managment Technique	Memory allocation, paging, segmentation	File system management, space allocation
Data Handling	Yes (uses part of secondary as virtual RAM)	Yes (used for swap files and virtual memory)
Access Time	Temporary; flushed regularly	Persistent; structured
Access Time	Nanoseconds	Milliseconds (HDD) or microseconds (SSD)

An operating system manages **primary storage (RAM)** by allocating memory to active processes and ensuring efficient usage through techniques like **paging, segmentation, and virtual memory**. Virtual memory allows the system to use a portion of secondary storage as additional RAM when needed, enabling more processes to run simultaneously. The OS also performs **memory protection**, which isolates processes to prevent them from accessing each other's memory space, thus maintaining system stability and security.

For **secondary storage** (such as HDDs or SSDs), the OS is responsible for **file system management**, organizing data into files and directories, and keeping track of their locations. It handles storage allocation, determines how files are stored and retrieved, and uses **disk scheduling algorithms** to optimize read/write operations. The OS also enforces **access control** to protect files from unauthorized access and includes tools for **data integrity**, backup, and recovery. These combined functions ensure that both temporary and long-term data are managed effectively, securely, and efficiently.

Q-3. What is OpenStack and what are its key components? How does it differ from traditional virtualization platforms?

- **OpenStack** is an **open-source cloud computing platform** designed to control and manage large pools of compute, storage, and networking resources through a **dashboard or APIs**.
- It enables the creation and management of **public, private, and hybrid clouds**.
- Developed and maintained by a global community, it supports **Infrastructure as a Service (IaaS)**.

❖ **Key Components of OpenStack:**

1) **Nova** - Manages computer resources and virtual machines.
2) **Neutron** - Provides networking as a service, managing IPs, routers, firewalls
3) **Cinder** - Manages block storage volumes.
4) **Swift** - Offers object storage for scalable, redundant data storage.
5) **Glance** - Manages disk and server images.
6) **Keystone** - Handles authentication and identity management.

❖ **Differences from Traditional Virtualization Platforms:**

1) **Scope**: OpenStack offers full cloud management, not just virtualization.
2) **Multi-Component Architecture**: Uses modular services for compute, storage, and networking.
3) **Open Source**: Unlike many traditional platforms (e.g., VMware), OpenStack is free and community-driven.
4) **API-Driven**: Allows automated and programmable infrastructure provisioning.
5) **Scalability**: Designed for massive cloud environments, not just local virtual machines.

Q-4. Describe the architecture of the Windows operating system. What are some unique features of Windows that distinguish it from other operating systems?

The **Windows operating system architecture** is designed as a **hybrid structure**, combining aspects of both monolithic and microkernel designs. At its core is the **Windows Kernel**, which manages low-level tasks like memory management, process scheduling, and hardware abstraction. Above the kernel is the **Executive layer**, responsible for handling services such as security, input/output, and object management. The **Hardware Abstraction Layer (HAL)** sits below the kernel, enabling the OS to interact with various hardware platforms seamlessly.

User interaction occurs through the **Windows API** and **user-mode subsystems**, such as the **Win32 subsystem**, which provides the primary interface for applications. Additionally, Windows includes **system services**, **device drivers**, and **environment subsystems** for compatibility with legacy applications (e.g., DOS, POSIX).

Unique features that distinguish Windows from other operating systems include:

1. **Graphical User Interface (GUI)**: Windows offers a user-friendly and consistent GUI, making it accessible for both casual and professional users.
2. **Backward Compatibility**: Windows supports many legacy applications and drivers, ensuring continuity for long-term users.
3. **Active Directory**: A centralized domain management system for user authentication and network management in enterprise environments.
   Windows Registry: A hierarchical database for configuration settings, unique to Windows.
4. **Extensive Hardware Support**: Broad compatibility with a wide range of hardware devices.
5. **Integrated Security Features**: Built-in tools like Windows Defender, BitLocker, and User Account Control (UAC) enhance system security.
6. **Frequent Updates and Patching**: Regular system updates ensure improved functionality and protection.

These elements contribute to Windows dominance in personal and business computing environments.

Q-5. What are containers in the context of operating systems? How do they improve application deployment and management?

Containers are an OS-level virtualization method that allows multiple isolated user-space instances (containers) to run on a single host OS kernel. Each container operates as if it has its own separate operating system, but in reality, it shares the same kernel with the host, making containers much more lightweight than traditional virtual machines.

How Containers Improve Application Deployment and Management:

1. **Portability**: Containers encapsulate all dependencies, making applications run consistently across different environments (development, testing, production).
2. **Efficiency**: Containers use fewer resources than virtual machines since they don't require a full OS per instance, leading to faster startup times and lower overhead.

3. **Scalability**: Containers make it easy to scale applications horizontally. Tools like Kubernetes help orchestrate and manage large numbers of containers.
4. **Simplified Deployment**: Containers enable easy packaging and shipping of applications, reducing complexity in deployment pipelines.
5. **Version Control and Rollbacks**: Container images can be versioned, allowing easy rollback to previous application states when needed.

Overall, containers enhance the operating system's ability to manage, isolate, and scale applications, making deployment more efficient while maintaining strong control and security at the kernel level.