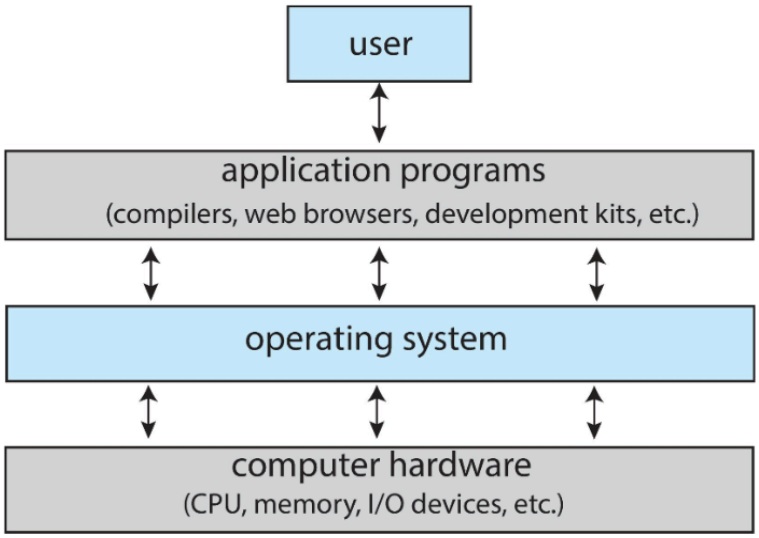
OS CHAP 1

1. What is an Operating System & Types of OS

An operating system (OS) is software that acts as an interface between computer hardware and user applications. It manages the resources and provides services for the efficient and secure execution of programs. The primary functions of an operating system include process management, memory management, file system management, device management, and user interface.



Roles of Operating System (OS) with Respect to Users and Hardware

For Users:

1. **Convenience and Usability:**
    - The OS provides a user-friendly interface, often through graphical user interfaces (GUIs) or command-line interfaces (CLIs), making it easier for users to interact wi0th the computer.
    - For mobile devices, specialized interfaces like touchscreens and voice recognition are designed for intuitive user experiences, optimizing for usability and battery life.
2. **Performance:**
    - The OS ensures good performance by efficiently managing hardware resources. Users expect smooth operation, quick response times, and the ability to run multiple applications simultaneously without noticeable delays.
3. **Resource Abstraction:**
    - The OS abstracts the complexities of hardware operations, presenting a simplified interface to the user. This allows users to perform complex tasks without needing to understand the underlying hardware processes.
4. **Multitasking and Resource Sharing:**
    - In shared systems like mainframes or servers, the OS facilitates multitasking and ensures that resources are distributed fairly among users, maintaining system stability and user satisfaction.

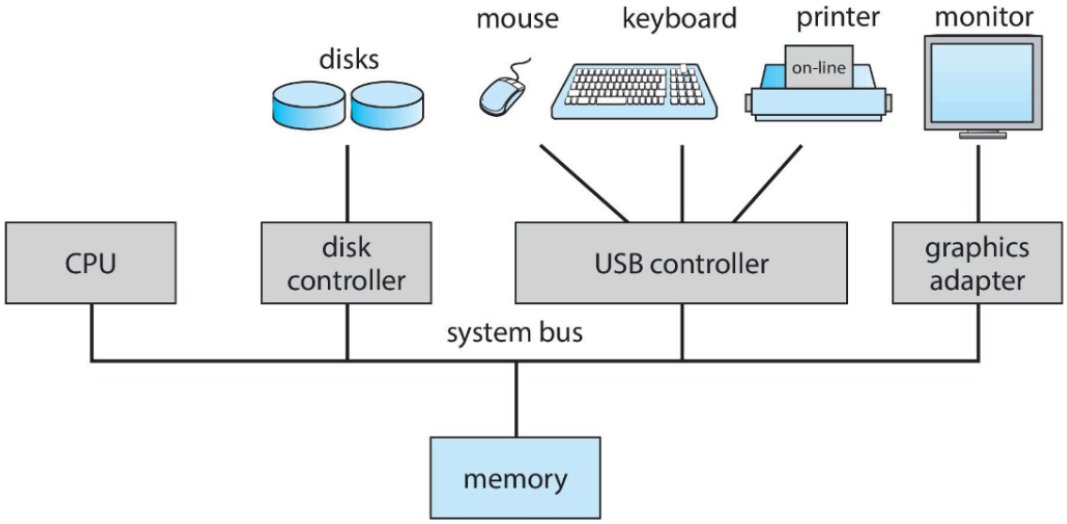
For Hardware:

1. **Resource Allocation:**
    - The OS acts as a resource allocator, ensuring efficient use of CPU, memory, storage, and input/output devices. It manages these resources to prevent conflicts and optimize performance.
2. **Control Program:**
    - The OS controls the execution of user programs and system processes, ensuring they run in an orderly manner. It handles process scheduling, context switching, and inter-process communication.
3. **Hardware Abstraction:**
    - The OS provides a layer of abstraction between the hardware and software applications, allowing applications to run on different hardware configurations without modification.
4. **Device Management:**
    - The OS manages device drivers and facilitates communication between hardware devices and software applications, ensuring devices operate correctly and efficiently.
5. **Security and Protection:**
    - The OS enforces security policies, protecting against unauthorized access and ensuring data integrity. It manages user permissions and isolates processes to prevent interference.
6. **Power Management:**
    - Particularly in mobile devices, the OS optimizes power consumption to extend battery life. It manages resource usage based on device activity and power-saving modes.
7. **Embedded Systems:**
    - In embedded systems, the OS runs with minimal user intervention, focusing on specific tasks. It ensures reliable and real-time performance for applications like automotive control systems or household appliances.

By balancing these roles, the OS bridges the gap between user expectations and hardware capabilities, ensuring smooth and efficient operation of computing devices.

Computer-system operation

- One or more CPUs, device controllers connect through a common bus providing access to shared memory.
- Concurrent execution of CPUs and devices competing for memory cycles.



Buffer: temporary storage
Concurrently: parallely

1. I/O devices and the CPU can execute concurrently.
2. Each device controller is in charge of a particular device type.
3. Each device controller has a local buffer.
4. Each device controller type has an operating system device driver to manage it.
5. CPU moves data from/to main memory to/from local buffers.
6. I/O is from the device to local buffer of controller.
7. Device controller informs CPU that it has finished its operation by causing an interrupt.

Differentiate between Traps, Exception and Interrupts.

- **Interrupts**: Asynchronous events usually produced by I/O devices which must be handled by the processor by interrupting execution of the currently running process
- **Traps**: Synchronous events produced by special instructions typically used to allow secure entry into operating system code
- **Exceptions**: Synchronous events usually associated with software requesting something the hardware can't perform i.e. illegal addressing, illegal op code, etc.

Type	Sync/Async	Source	Intentional?	Examples
Exception	Sync	Internal	No	Overflow, Divide by zero, Illegal memory address
Trap	Sync	Internal	Yes and No	System call, Page fault, Emulated instructions
Interrupt	Async	External	Yes	I/O device completion

1. What is an Interrupt?

An interrupt is a signal sent by hardware or software to the processor, indicating an event that needs immediate attention. Examples include hardware interrupts from devices like keyboards or software interrupts triggered by programs.

2. Interrupt Handling Process

When an interrupt occurs, the following steps generally take place:

a. Interrupt Signal Detection

The processor detects the interrupt signal from an external device or software.

b. Saving the Current State

To ensure the current execution context is not lost, the processor saves the current state, including the Program Counter (PC) and registers, to the stack.

c. Interrupt Vector Table Lookup

The processor uses an interrupt vector, which is an identifier for the interrupt, to look up the corresponding entry in the Interrupt Vector Table (IVT). The IVT contains addresses of the Interrupt Service Routines (ISRs).

3. Interrupt Vector Table (IVT)

The IVT is a table that holds pointers to ISRs for each interrupt type. When an interrupt occurs, the processor uses the interrupt vector to find the ISR address in the IVT.

4. Interrupt Service Routine (ISR)

The ISR is a specialized function designed to handle the specific interrupt. The steps involved are:

a. Execution of ISR

The processor jumps to the ISR address retrieved from the IVT and begins executing the ISR code.

b. Handling the Interrupt

The ISR performs the necessary actions to handle the interrupt. This may include interacting with hardware, reading/writing data, or performing specific tasks.

c. Clearing the Interrupt

After handling the interrupt, the ISR clears the interrupt flag, ensuring the processor knows the interrupt has been serviced.

5. Restoring the State

Once the ISR completes its tasks, the processor restores the saved state from the stack, allowing normal program execution to resume from where it was interrupted.

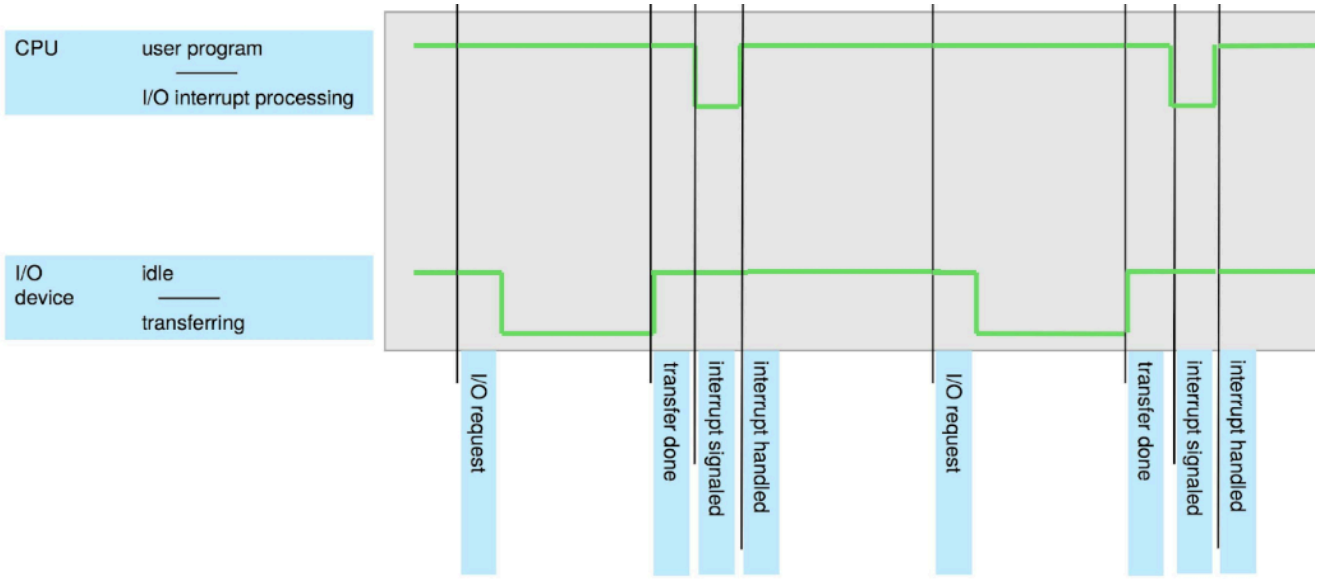
Summary

- **Interrupts**: Signals requiring immediate attention.
- **IVT**: Contains pointers to ISRs.
- **ISR**: Executes tasks related to the interrupt.
- **State Restoration**: Resumes normal execution.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

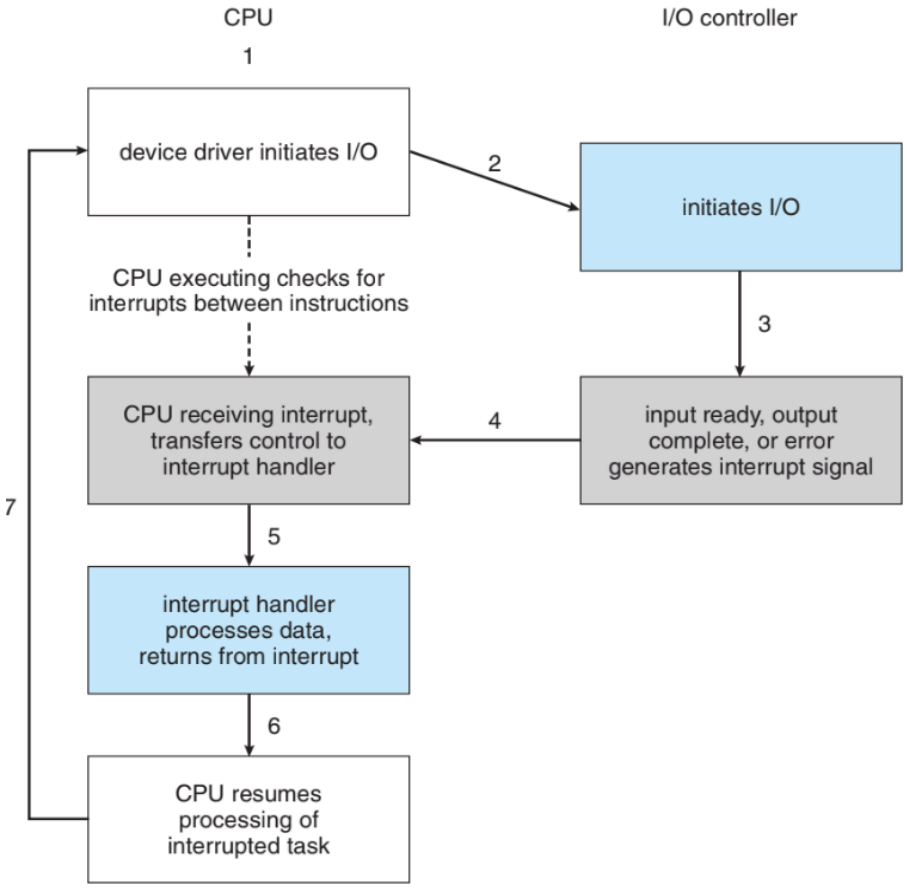
Figure 1.5 Intel processor event-vector table.

INTERRUPT TIMELINE:



Key Events and State Transitions:

1. **I/O Request**
    - **CPU**: The user program requests an I/O operation.
    - **I/O Device**: The device is idle, waiting for the request.
2. **Starting I/O Operation**
    - **CPU**: Initiates the I/O operation.
    - **I/O Device**: Begins the data transfer.
3. **CPU Continues Execution**
    - **CPU**: Resumes executing the user program while the I/O device handles the transfer.
4. **Transfer Completion**
    - **I/O Device**: Completes the data transfer and signals the CPU.
5. **Interrupt Signaled**
    - **CPU**: Detects the interrupt signal from the I/O device.
6. **ISR Execution**
    - **CPU**: Executes the Interrupt Service Routine (ISR) to handle the interrupt.
7. **Restoring State**
    - **CPU**: Resumes executing the user program from where it was interrupted.



STORAGE HIERARCHY:

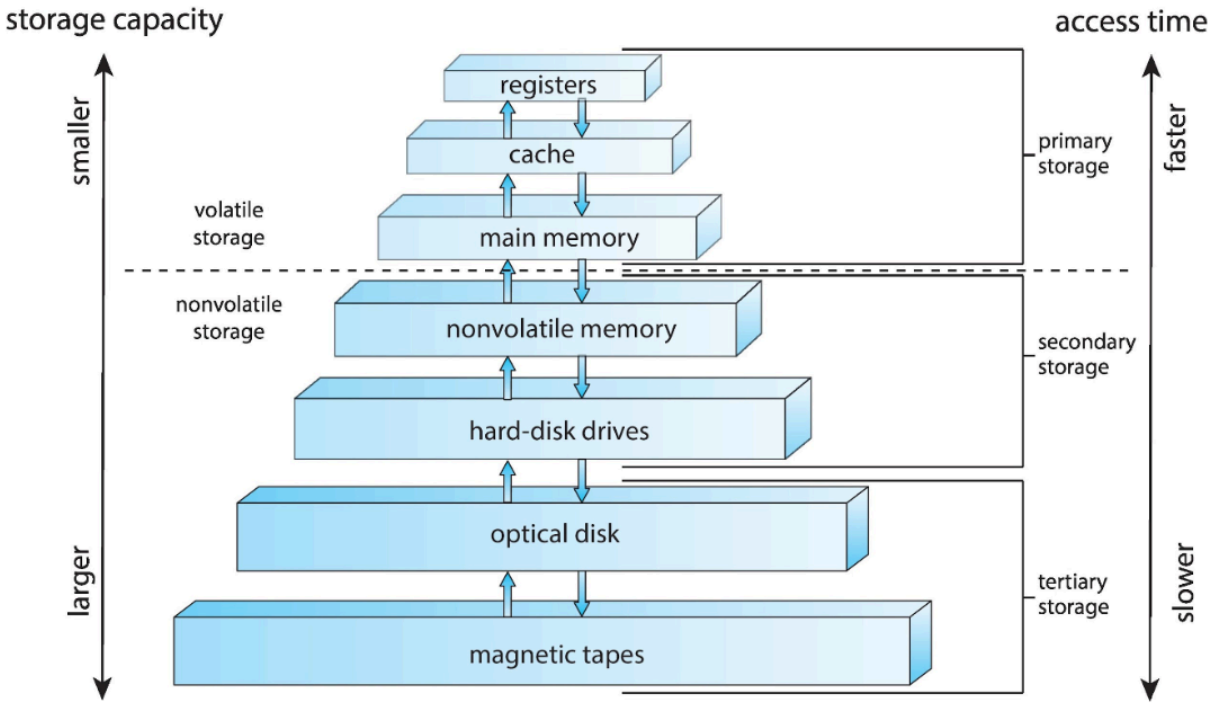
Storage systems are organized in hierarchy based on:

1. Speed
2. Cost
3. Volatility

Caching – copying information into faster storage system; main memory can be viewed as a cache for secondary storage

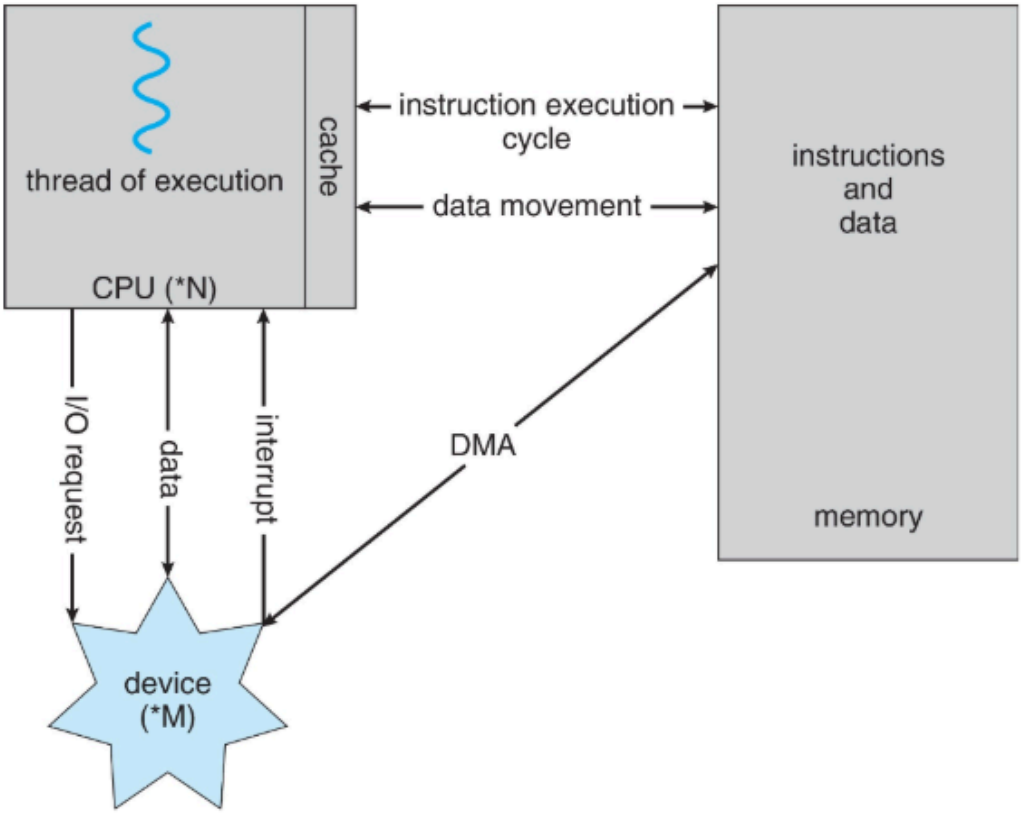
Device Driver for each device controller to manage I/O

Provides a uniform interface between controller and kernel.



DMA [DIRECT MEMORY ACCESS]:

- DMA is used for high-speed I/O devices that need to transfer data quickly.\
- The device controller transfers blocks of data directly to main memory without CPU intervention.
- This reduces the CPU's workload and speeds up data transfer.
- Only one interrupt is generated per block, rather than one per byte, which further enhances efficiency.



Multiprogramming and Multitasking in Modern Computers

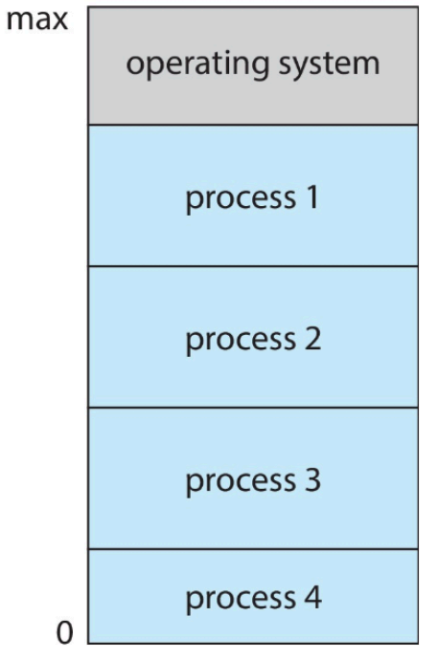
1. Multiprogramming (Batch System):

- **Single User Limitation:** One user cannot always keep the CPU and I/O devices busy.
- **Job Organization:** Multiprogramming organizes jobs (code and data) so that the CPU always has one to execute.
- **Memory Management:** A subset of the total jobs in the system is kept in memory.
- **Job Scheduling:** One job is selected and run via job scheduling.
- **OS Switching:** When a job has to wait (e.g., for I/O), the OS switches to another job.

2. Multitasking (Timesharing):

- **Interactive Computing:** The CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing.
- **Response Time:** The response time should be less than 1 second.
- **Processes:** Each user has at least one program executing in memory, called a process.
- **CPU Scheduling:** If several jobs are ready to run at the same time, CPU scheduling decides which job to run next.
- **Swapping:** If processes don't fit in memory, swapping moves them in and out to run.
- **Virtual Memory:** Allows execution of processes that are not completely in memory.

Memory layout of Multiprogrammed system:



Parameter	Multiprogramming	Multitasking
Definition	It lets multiple programs use the CPU at the same time.	Multitasking refers to the simultaneous execution of numerous programmes, processes, and threads with the certain timestamp.
Objective	It's useful for cutting down on CPU idle time and boosting throughput as much as possible.	It may be used to execute numerous tasks at once, significantly improving CPU and system throughput.
Mechanism	The context switching method is used.	Based on a time-sharing mechanism.
Time	Multiprogramming demands comparatively more time to complete tasks.	Multitasking allows you to complete tasks in less time.
Execution	In a multi-programmed system, when one job or process completes its execution or changes to an I/O task, the system temporarily suspends that process. It chooses a new process to execute from the process scheduling pool.	Multiple processes can run concurrently in this system by assigning the CPU for a fixed duration of time.
CPU Switching	The CPU shifts between processes swiftly in this environment.	The CPU shifts between the processes of several programs in a single-user environment.
CPU required	Only one CPU is needed in Multiprogramming to run the tasks.	Multiple CPUs are required for the task allocation.

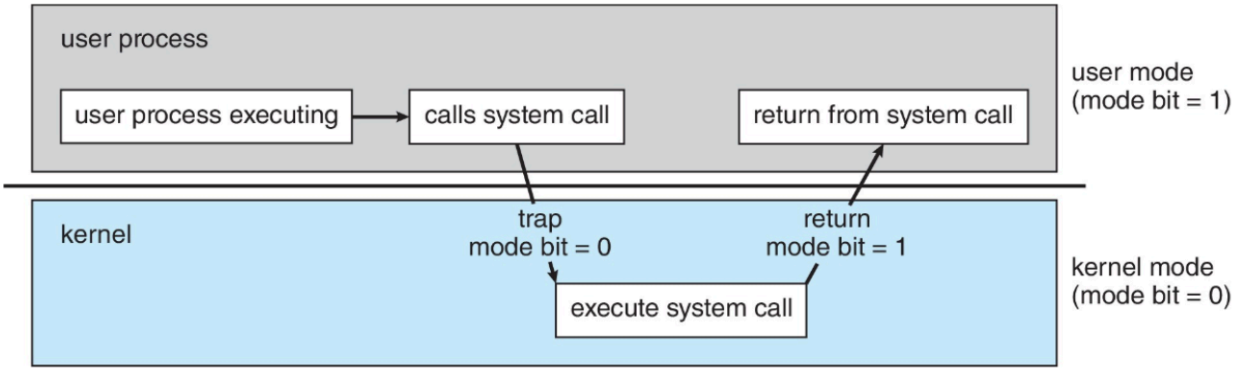
Multiprogramming vs. Multitasking

DUAL MODE OPERATION:

Privileged Instructions: Instructions that can only be executed in kernel mode, ensuring system protection.

Use of Dual-Mode Operation:

- **System Protection**: Prevents user applications from directly accessing or modifying kernel code and data, maintaining the integrity and security of the operating system.
- **Controlled Access**: Allows user applications to request services from the kernel in a controlled manner through system calls, ensuring proper system functionality.
- **Memory Protection**: Utilizes hardware features to prevent user applications from accessing kernel memory space, safeguarding critical system components.



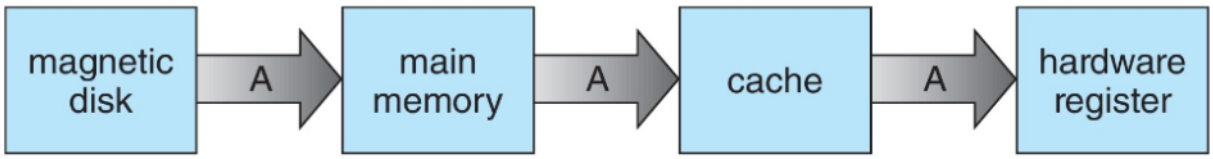
Timer in Operating Systems:

- Timer Interrupt: A timer is set to interrupt the computer after a specific time period to prevent infinite loops or a process from hogging resources.
- Counter: A counter is maintained and decremented by the physical clock. When the counter reaches zero, it generates an interrupt.
- Privileged Instruction: Setting the counter is a privileged instruction performed by the operating system.
- Regain Control: The interrupt allows the operating system to regain control, ensuring that no process exceeds its allotted time.

Use:

- **Prevents Resource Hogging:** Ensures that no single process can monopolize CPU time.
- **System Stability:** Helps maintain system stability and performance by enabling the OS to manage and allocate resources effectively.
- **Process Scheduling:** Facilitates fair process scheduling by allowing the OS to interrupt and switch between processes.

MIGRATION OF DATA "A" FROM DISK TO REGISTER:



COMPUTER SYSTEM ARCHITECTURE:

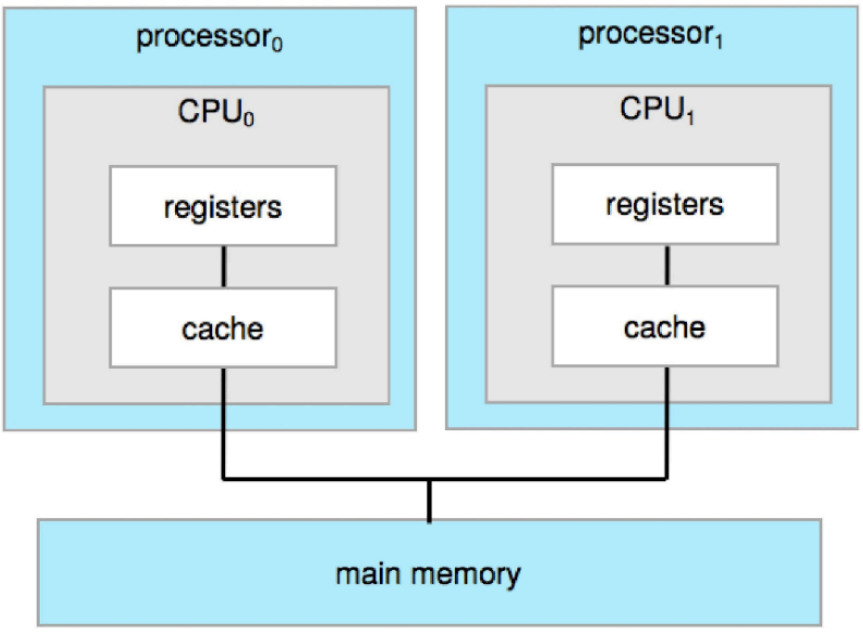
Advantages of Multiprocessor Systems:

1. **Increased Throughput**: More tasks can be processed simultaneously.
2. **Economy of Scale**: Costs per processor are reduced with more processors.
3. **Increased Reliability**: Fault tolerance through graceful degradation ensures system stability.

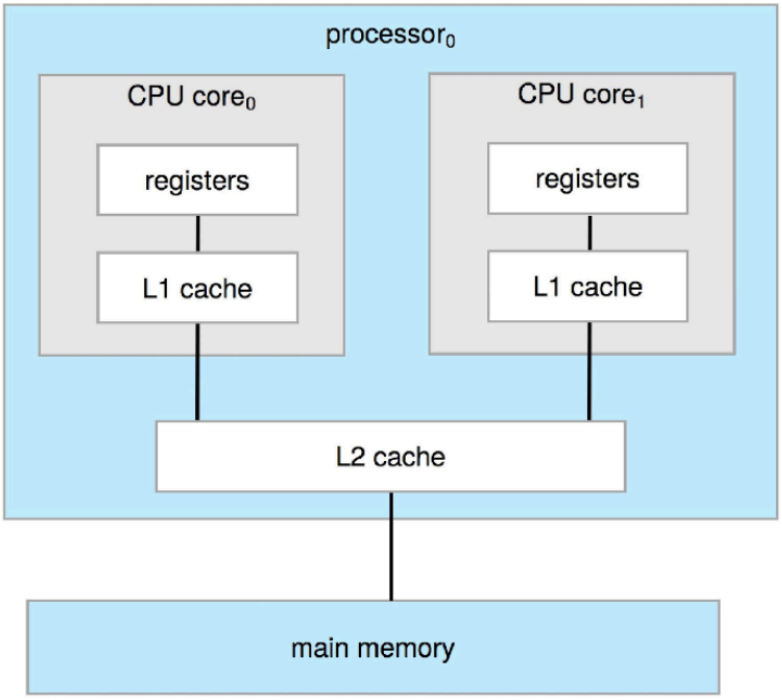
Types of Multiprocessing:

1. **Asymmetric Multiprocessing (AMP)**: Each processor is assigned a specific task, and one processor often controls the others.
2. **Symmetric Multiprocessing (SMP)**: All processors share tasks equally, with no one processor being in control.

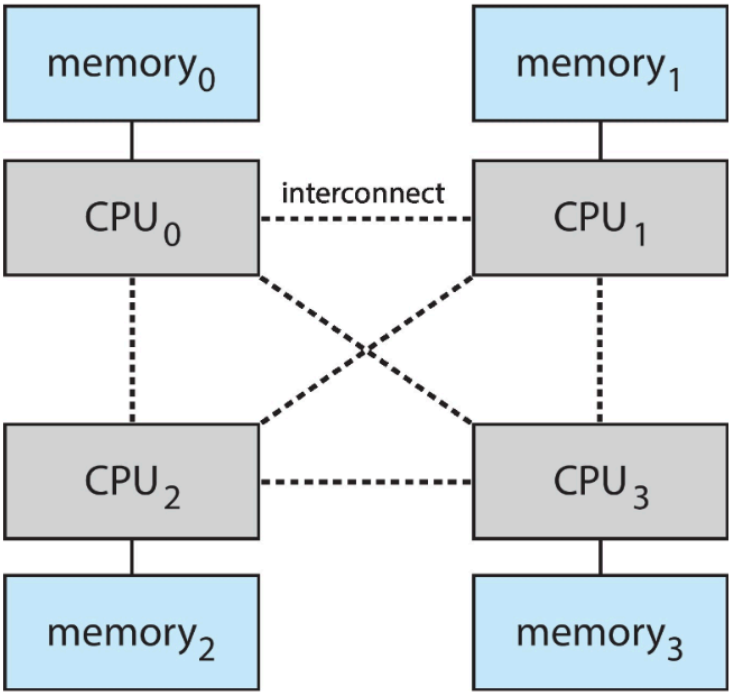
Symmetric Multiprocessing Architecture:



MULTICORE/DUAL CORE DESIGN:



NON UNIFORM MEMORY ACCESS SYSTEM:



Clustered Systems:

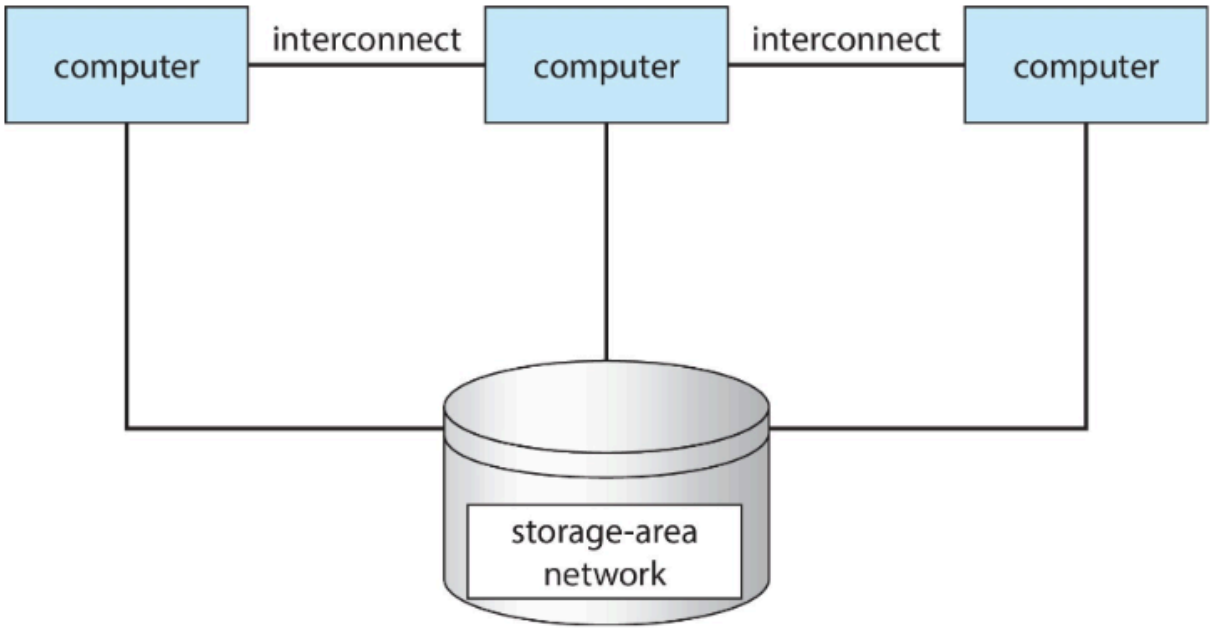
Clustered systems are similar to multiprocessor systems, but consist of multiple independent systems working together to provide higher availability and performance.

Key Characteristics:

- **Multiple Systems**: Multiple computers (nodes) work together, typically sharing storage via a Storage-Area Network (SAN).
- **High Availability**: Provides a high-availability service that can survive hardware or software failures.

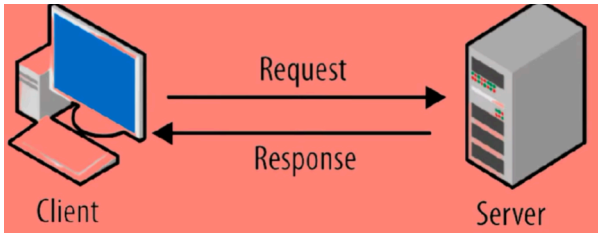
Clustering Types:

1. **Asymmetric Clustering**: One machine (node) is in hot-standby mode, ready to take over in case of a failure.
2. **Symmetric Clustering**: Multiple nodes run applications and monitor each other, providing a balanced workload and redundancy.



COMPUTING ENVIRONMENTS:

CLIENT-SERVER COMPUTING: way of communication between the client and server, which works as a system of requests and response.



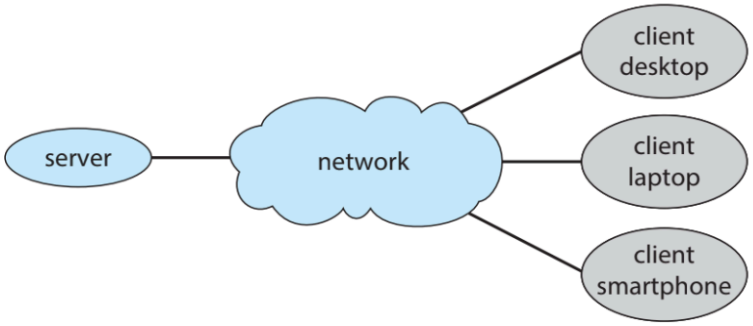


Figure 1.22 General structure of a client–server system.

PEER TO PEER COMPUTING:

In this model, clients and servers are not distinguished from one another. Instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service. Peer-to-peer systems offer an advantage over traditional client–server systems. In a client–server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.

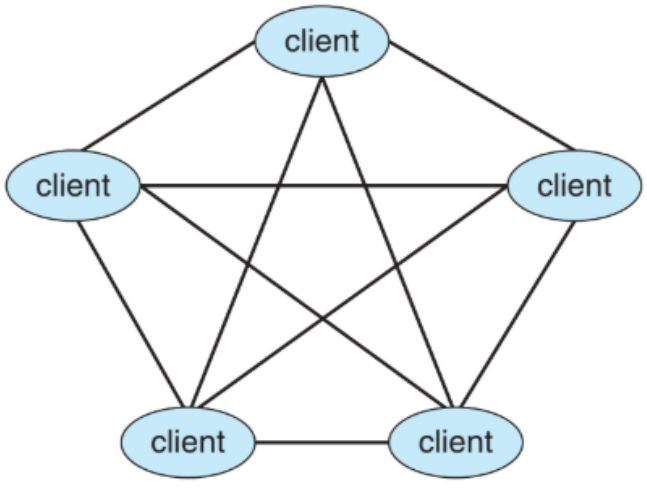


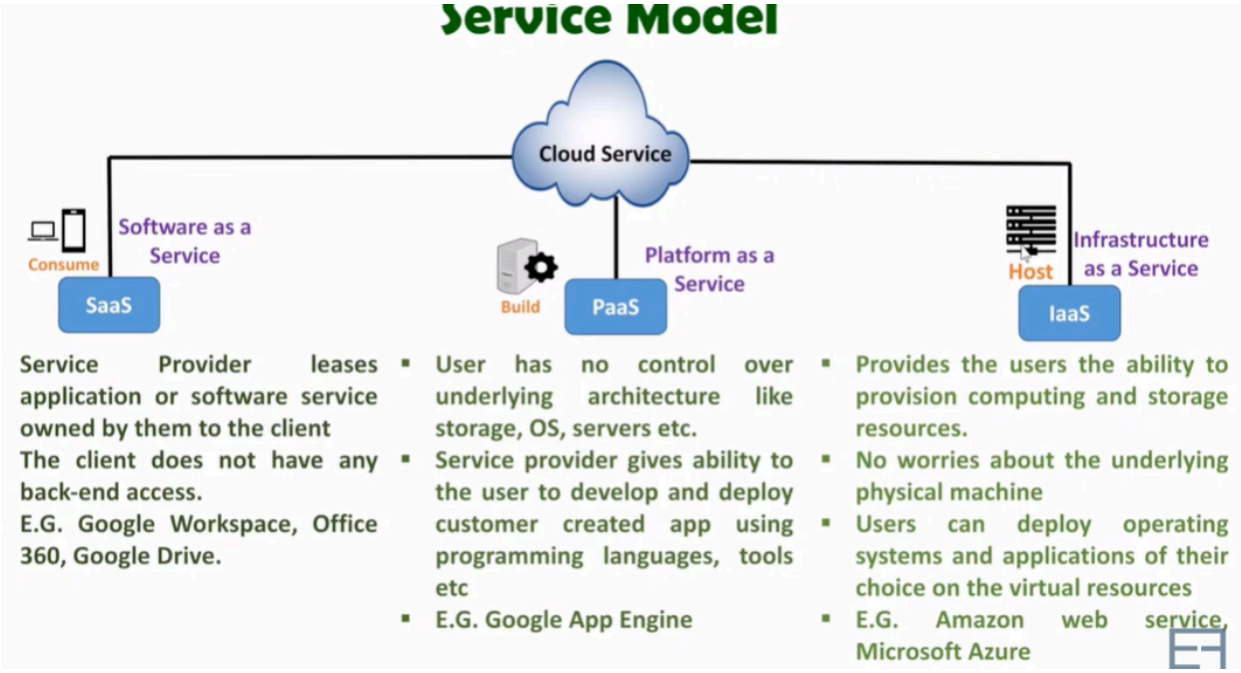
Figure 1.23 Peer-to-peer system with no centralized service.

Skype is another example of peer-to-peer computing. It allows clients to make voice calls and video calls and to send text messages over the Internet using a technology known as voice over IP (VoIP).

CLOUD COMPUTING:

Here are some key benefits of cloud computing:

- Scalability: Easily adjust resources as needed.
- Cost Efficiency: Pay only for what you use.
- Accessibility: Access your data and apps from anywhere with an internet connection.
- Backup and Recovery: Automatic backups and easier disaster recovery.
- Collaboration: Seamless collaboration with others on shared documents or projects.



Cloud Service Models:

1. **SaaS (Software as a Service):**
    - Provides software applications over the internet.
    - No need to install or maintain the software locally.
    - Examples: Google Workspace (formerly G Suite), Microsoft 365, Salesforce.
2. **PaaS (Platform as a Service):**
    - Provides a platform for developers to build, deploy, and manage applications.
    - No need to manage underlying infrastructure.
    - Examples: Microsoft Azure App Services, Google App Engine, AWS Elastic Beanstalk.

3. **IaaS (Infrastructure as a Service):**
    - Provides virtualized computing resources over the internet.
    - Users manage operating systems, applications, and middleware.
    - Examples: Amazon EC2, Microsoft Azure Virtual Machines, Google Compute Engine.

Cloud Deployment Models:

1. **Public Cloud:**
    - Services are delivered over the public internet and shared among multiple organizations.
    - Cost-effective and scalable.
    - Examples: AWS, Microsoft Azure, Google Cloud.
2. **Private Cloud:**
    - Services are dedicated to a single organization.
    - Offers greater control and security.
    - Examples: VMware vSphere, Microsoft Azure Stack.
3. **Hybrid Cloud:**
    - Combines both public and private clouds.
    - Allows data and applications to be shared between them.
    - Offers flexibility and optimized workload management.
    - Examples: IBM Hybrid Cloud, Microsoft Azure Hybrid Cloud.

REAL TIME EMBEDDED SYSTEMS: are specialized computing devices that run real-time operating systems to handle specific tasks within tight time constraints. From managing car engines and robotic arms to controlling home appliances and industrial systems, they are everywhere, making our lives more efficient.

Real-time operating systems (RTOS) in embedded systems ensure that tasks are executed within defined time limits, making them crucial for applications where timing is critical, like medical imaging systems and automobile fuel-injection systems.

WHY APPLICATIONS ARE OS SPECIFIC?

Operating systems have different file formats and system libraries, which means that software developed for one operating system may not be able to run on another without modification or additional software. For example, Windows uses the .exe file format for executables, while Linux uses the ELF format. Similarly, Windows has its own system libraries and dependencies that aren't present on Linux, and vice versa.

Software may also have dependencies on specific versions of system libraries, which may not be available on all operating systems. For example, an application developed on Windows using a

specific version of the .NET Framework may not be compatible with Linux, which doesn't support the .NET Framework out of the box.

Operating systems may have different hardware and architecture requirements.

OS-Specificity	Description
Binary Executable	A program that has been compiled and linked for a specific operating system, such as Windows or Linux. can run only on the target operating system.
API Calls	An application programming interface (API) is a set of functions or methods that allows programmers to interact with the operating system. API calls are specific to operating systems.
File System	Each operating system has its own file system, which organizes and stores files and directories. Applications may use specific paths or file naming conventions that are specific to the file system of a particular operating system.
Registry	The registry is a database that stores configuration settings and options for the operating system and applications. The structure of the registry is specific to each operating system.
User Interface	The user interface of an application may use widgets, menus, and other elements that are specific to a particular operating system. For example, the look and feel of a Windows application may be different from a Mac application.
Device Drivers	Device drivers are software components that allow the operating system to communicate with hardware devices. Each operating system may have different drivers for the same hardware, so device drivers can be specific to the operating system.

Chapter 1 : Operating Systems Questions and Answers

1.1 What are the 3 main purposes of an operating system?

1. **Resource Management:** Manages hardware resources like CPU, memory, and I/O devices efficiently.
2. **User Interface:** Provides an interface for users to interact with the computer (CLI or GUI).
3. **Program Execution:** Facilitates the execution of programs by handling process scheduling, memory allocation, etc.

1.2 When is it appropriate for the OS to "waste" resources, and why is it not really wasteful? When focusing on **user experience**, such as in **interactive systems** (e.g., GUI responsiveness), the OS may "waste" CPU cycles to reduce latency. It's not truly wasteful because the goal is to improve usability, even if hardware efficiency slightly decreases.

1.3 Main difficulty in writing an OS for a real-time environment?
Ensuring **predictable, time-bound responses** to events. The OS must guarantee that critical tasks meet strict deadlines, requiring precise scheduling and minimal delays.

1.4 Should the OS include applications like web browsers and mail programs?

- **Yes:** Provides a complete, user-friendly environment (e.g., Windows includes Edge).
- **No:** Increases OS bloat; violates modular design principles. Applications should be separate for flexibility and security (as in minimal Linux distros).

1.5 How does kernel mode vs. user mode provide security?

- **Kernel Mode:** Full access to system resources (privileged instructions).
- **User Mode:** Restricted access, preventing direct hardware manipulation.
  This separation protects the system from malicious or faulty user programs.

1.6 Which instructions should be privileged?

- **Privileged (for security reasons):**
  a. Set value of timer ✅
  c. Clear memory ✅
  e. Turn off interrupts ✅

f. Modify device-status table ✅
g. Switch from user to kernel mode ✅
h. Access I/O device ✅

- **Non-Privileged (safe for users):**
  b. Read the clock ❌
  d. Issue a trap instruction (since traps intentionally switch to kernel mode safely) ❌

1.7 Difficulties with OS protection via non-modifiable memory partitions:

1. **Inflexibility:** The OS cannot update itself, apply patches, or fix bugs dynamically.
2. **Inefficient Memory Use:** Wasted space if the partition is larger than needed, or limitations if it's too small.

1.8 Two possible uses of multiple CPU modes:

1. **Different Privilege Levels:** For complex systems (e.g., kernel, driver, user levels).

2. **Virtualization Support:** Separate modes for managing virtual machines securely.

1.9 How can timers compute the current time?
By setting a timer to generate periodic interrupts. The OS increments a **counter** with each interrupt, effectively tracking the passage of time relative to a known starting point.

1.10 Why are caches useful? What problems do they solve/cause?

- **Benefits:**
  1. **Speed:** Reduces data access time.
  2. **Efficiency:** Lowers CPU idle time waiting for data.
- **Problems:**
  1. **Consistency Issues:** Cached data may become outdated (cache coherence problems).
  2. **Complexity:** Requires sophisticated algorithms to manage cache hits/misses.

Why not make caches as large as the device?

Cost and performance trade-offs: Large caches are expensive and may not provide proportional speed gains due to diminishing returns.

1.11 Client–Server vs. Peer-to-Peer (P2P) Models:

- **Client–Server:** Centralized model where clients request services from a central server (e.g., web servers).
- **Peer-to-Peer (P2P):** Decentralized model where each node can act as both client and server (e.g., file-sharing networks like BitTorrent).

1.12 How do clustered systems differ from multiprocessor systems?

- **Clustered Systems:** Consist of multiple independent computers (nodes) connected via a network, working together.
- **Multiprocessor Systems:** Have multiple CPUs within a single system, sharing the same memory and I/O.

For cooperation in clusters:

- **Shared Storage (like SAN/NAS)**
- **Cluster Management Software** for load balancing and failover.
- **Heartbeat Mechanism** to detect node failures.

1.13 Two ways cluster software can manage database access:

1. **Shared-Disk Model:** Both nodes access the same disk.
    - *Pros:* High availability, data consistency.
    - *Cons:* Risk of data corruption without proper locking mechanisms.
2. **Shared-Nothing Model:** Each node has its own disk, data replication used.
    - *Pros:* Better scalability, no disk contention.
    - *Cons:* Complex replication management, potential data inconsistency during failovers.

1.14 Purpose of interrupts & difference from traps:

- **Interrupts:** Signal from hardware to CPU to handle events (e.g., I/O completion). • **Traps:** Software-generated interrupt, often due to errors or system calls.

Can traps be intentional? Yes, for **system calls** (e.g., accessing OS services).

1.15 How Linux kernel variables HZ and jiffies determine uptime:

- **HZ:** Number of clock ticks per second.
- **jiffies:** Counter incremented every tick.

Formula:
Uptime (in seconds) = jiffies / HZ

1.16 Direct Memory Access (DMA):

a. **CPU-DMA Coordination:** CPU sets up DMA by providing memory addresses and size, then signals the DMA controller.
b. **Completion Notification:** DMA sends an **interrupt** to CPU after transfer.
c. **Interference:** Minor delays due to bus contention, but CPU can execute other tasks simultaneously.

1.17 Secure OS without privileged mode:

- **Possible:** Through software-based controls, sandboxing, or language-level security. • **Not Possible:** Lacks hardware-enforced isolation, making it vulnerable to malicious code.

1.18 Why SMP systems have multi-level caches:

- **Local Cache:** Fast, reduces latency for each core.
- **Shared Cache:** Ensures data consistency across cores.
  This design balances **speed** and **data coherence**.

1.19 Rank storage systems (slowest → fastest):

Magnetic tapes < Optical disk < Hard-disk drives < Nonvolatile memory < Main memory < Cache < Registers

1.20 Data inconsistency in SMP systems:

If **Core A** updates a variable in its local cache but doesn't write it back to main memory, **Core B** might still read the old value from its cache, causing inconsistency.

1.21 Cache coherence problems:

a. **Single-Processor:** Issues with I/O devices modifying memory directly.
b. **Multiprocessor:** Inconsistent cached data across cores.
c. **Distributed Systems:** Replicated data becoming outdated across nodes.

1.22 Memory protection mechanism:

Use **base and limit registers** or **virtual memory** with page tables to restrict access outside a program's allocated space.

1.23 LAN vs. WAN:

a. **Campus Student Union:** LAN (localized, high-speed).
b. **Statewide University System:** WAN (geographically distributed).
c. **Neighborhood:** LAN (short distances).

1.24 Challenges in mobile OS design:

- Battery Life Optimization
- Limited Resources (CPU, RAM)
- Varied Hardware Compatibility
- Security in Untrusted Networks

1.25 Advantages of P2P over Client-Server:

- Decentralization (no single point of failure)
- Scalability

- Harder to manage security and data consistency.

1.26 Distributed applications suitable for P2P:

- File Sharing (e.g., BitTorrent)
- Blockchain Networks
- VoIP (e.g., Skype)

1.27 Open-Source OS: Pros & Cons

Advantages:

- Free to use
- Community-driven development
- High customizability

Disadvantages:

- Requires technical expertise
- Limited official support

Who benefits?

- **Developers:** Love flexibility.
- **Enterprises:** Appreciate cost savings.
- **Casual Users:** Might struggle with complexity.

1.6 explanation:

Privileged Instructions (Should be Restricted):

1. **(a) Set value of timer**
    - **Why Privileged?** Timers control process scheduling. If user programs could set them, they could manipulate CPU time unfairly or cause infinite loops.
2. **(c) Clear memory**
    - **Why Privileged?** Clearing memory can erase critical OS data or other processes' data, leading to crashes or security breaches.
3. **(e) Turn off interrupts**

- **Why Privileged?** Interrupts are essential for responding to hardware events. If disabled improperly, the system could hang, missing important signals (like I/O completions).

4. **(f) Modify entries in device-status table**
   - **Why Privileged?** This table tracks hardware device states. Unauthorized changes can cause device malfunctions or security vulnerabilities.
5. **(g) Switch from user to kernel mode**
   - **Why Privileged?** Allowing user programs to switch to kernel mode freely would grant them unrestricted access to all system resources, risking total control over the OS.
6. **(h) Access I/O device**
   - **Why Privileged?** Direct I/O access can lead to data corruption, unauthorized data leaks, or hardware damage. The OS controls access to maintain security and integrity.

Non-Privileged Instructions (Can Be Allowed in User Mode):

1. **(b) Read the clock**
   - **Why Non-Privileged?** Reading the system time doesn't modify any hardware or OS state. It's safe for user programs (like displaying timestamps).
2. **(d) Issue a trap instruction**
   - **Why Non-Privileged?** Traps are intentional system calls used to request OS services. They're designed for safe transitions from user to kernel mode under controlled conditions.

Key Takeaways:

- **Privileged = Can cause harm if misused (modifying hardware, OS state, etc.)**
- **Non-Privileged = Read-only or controlled requests (like system calls)**