**Define Monolithic and Layered System with suitable examples.**

**A Monolithic System** is an operating system structure where all the core components (such as file management, process management, memory management, etc.) are tightly coupled together.

**Key Characteristics of a Monolithic System**

* **Tight Coupling**: Components are highly interdependent, making changes to one part potentially impact others.
* **No Information Hiding**: Each part of the system can access all other parts.
* **Faster Performance**: Accessing between components, tasks can be completed quicker.

**Example**:  
In early versions of Unix and Linux, the operating system was built as one large program where all main functions worked together in a single structure. This meant that if one part of the system had a problem or crashed, it could bring down the entire OS.

A diagram of a structure

Description automatically generated

**Layered System:**

**Definition**:  
A Layered System organizes the components of an operating system into hierarchical layers, where each layer performs specific functions. Each layer communicates only with the layer directly above or below it, creating a clear structure and separation of concerns.

**Key Features**

* **Clear Separation**: Each layer has specific tasks and does not interfere with other layers unless necessary.
* **Information Hiding**: Each layer hides its internal workings from the other layers.
* **Slower Execution**: Because tasks go through multiple layers, the system might run slower.

**Example in Operating Systems**:

A diagram of a colorful circle

Description automatically generated

## **Batch Operating System**

**Batch Operating System**  
A Batch Operating System is an operating system designed to manage and execute a large number of jobs without direct user interaction.  
Jobs with similar requirements are grouped into batches by an operator.  
Once the jobs are grouped, the system processes them one by one, in sequence.

A diagram of a operating system

Description automatically generated

**Advantages of Batch Operating System**

* **Group tasks together**: Instead of processing each user’s task individually, the system groups jobs from different users and processes them all at once.
* **No idle time**: The system keeps working continuously. Once one batch of tasks is completed, the next batch starts immediately.

**Disadvantages of Batch Operating System**

* **Difficult to identify issues**: If something goes wrong during processing, it can be hard to pinpoint which specific job caused the problem, because all jobs are processed together in a batch.
* **Single job failure affects others**: If a job fails in a batch, all other jobs in the same batch have to wait for the issue to be fixed. This can cause unpredictable delays for all tasks in the batch.

**Examples of Batch Operating Systems**:

* Payroll Systems
* Bank Statements
* etc.

## **Real Time Operating System**

**A Real-Time Operating System (RTOS)**  
A Real-Time Operating System (RTOS) is a type of software that is designed to handle tasks that must be completed within strict time limits. Unlike regular operating systems, which can handle many tasks at once without worrying about exact timing, an RTOS makes sure that important tasks are finished exactly on time. This is especially important in systems like factory machines, airplanes, or cars, where missing a deadline could cause serious problems. The main goal of an RTOS is to ensure that critical tasks happen when they are supposed to, without delays, to keep everything running smoothly.

**Characteristics of RTOS:**

* **Time-Sensitive Tasks**: An RTOS ensures critical tasks, like controlling machines or medical devices, are done on time to avoid problems.
* **Predictability and Reliability**: It guarantees that tasks are completed on schedule, which is crucial for systems like air traffic control or heart pacemakers, where delays can have serious consequences.

**Applications of RTOS:**

* Radar systems for defense
* Missile guidance systems
* Online stock trading platforms
* Air traffic control systems
* Medical devices
* Autopilot systems for travel simulators

**Advantages of RTOS:**

* **Error-Free Operation**: RTOS is designed to be reliable and minimize errors, which is essential for systems that must work without fail.
* **Faster Task Switching**: RTOS can quickly switch between tasks, often in microseconds, which is important for responding fast to changes.

**Disadvantages of RTOS:**

* **Complex Design and Development**: Building an RTOS is difficult and costly.
* **Specialized Drivers**: RTOS needs special device drivers and signals to respond quickly to external events, making it harder to set up and maintain.

**Time-Sharing Operating System (TSOS)**

A **Time-Sharing Operating System (TSOS)** allows many users to work on the same computer at nearly the same time. Each user gets a brief, set amount of CPU time—called a **time slice**—which makes it feel like they have the whole computer to themselves, even though many others are using it too.

A diagram of a computer system

Description automatically generated

### How It Works

Imagine a library with just one computer, and each person in line gets 10 seconds to check their emails. The computer quickly moves to the next person after each time slice. By moving fast enough, everyone feels they’re making progress without having to wait long. This is the basic idea of a time-sharing OS.

**Main Components**

* **CPU Scheduling**: CPU scheduling quickly switches between tasks to ensure each one gets a fair share of CPU time, creating a balanced and responsive system for all users.
* **Memory Management**: Memory management allocates separate memory spaces to each user and program, preventing overlap and enhancing security.
* **Virtual Memory**: If there’s not enough memory for everyone’s programs, OS uses storage to extend RAM, which moves parts of programs in and out of the main memory as needed.

**Pros of Time-Sharing OS**

* **Multi-User Access**: Multiple users can interact with the system at the same time, sharing resources without needing separate computers.
* **Fair Access**: Every user gets equal CPU time, creating a sense of equal access.

**Cons of Time-Sharing OS**

* **Complexity**: More users and tasks mean higher chances of errors or failures.
* **Security Risks**: With many users, there’s a greater chance of security breach of each user’s information.

**How operating is as a resource manager? Explain.**

**How an Operating System (OS) Functions as a Resource Manager**

An **Operating System (OS)** serves as a **resource manager** by efficiently distributing and managing a computer's hardware and software resources. This ensures that multiple processes and applications can run simultaneously without interfering with each other.

* **CPU Scheduling/Management:**  
  The OS allocates CPU time to multiple processes through scheduling algorithms (e.g., round-robin, priority scheduling). This approach ensures each process has access to CPU resources, preventing any single process from occupying the CPU for too long.

* **Memory Management:**  
  The OS manages the allocation and deallocation of memory to different processes. The OS ensures that each program or process gets its own section of memory, preventing conflicts. When a program finishes, the OS releases the memory so it can be used by other programs, helping to avoid issues like **memory leaks**. It handles tasks like paging, segmentation, and virtual memory, which help in efficient memory utilization.
* **Device Management:**  
  The OS controls **input/output devices** like keyboards, mice, monitors, and printers. It acts as an intermediary between applications and hardware by providing device drivers and standard protocols. This allows applications to interact with hardware without needing to know specific device details
* **File Management:**  
  The OS organizes data storage in the form of files and directories, providing a structured file system. It enables users and applications to store, retrieve, and manage data efficiently, while also handling tasks like access permissions and file locking to prevent data corruption.

**What is system call? Discuss process of handling system calls briefly.**

A **system call** is a mechanism that allows a program running in user mode (a restricted environment) to request services or resources from the operating system kernel. Since programs in user mode cannot directly access hardware or perform certain low-level operations, system calls provide an interface through which they can interact with hardware and access necessary system resources.

**A diagram of a call center

Description automatically generated**

**System Call Process:**

1. Push parameters into the stack (1-3)
2. Calls library procedure (4)
3. Pass parameters in registers (5)
4. Switch from user mode to kernel mode and start to execute (6)
5. Check the system call number and then dispatch to the correct system call handler (7)
6. Run system call handlers (8)
7. Once the system call handler completes its work, control returns to the library procedure (9)
8. Control returns to the user program so it can continue running as normal.
9. Increment SP before call to finish the job

**Operating system is broker between computer System and User". Justify the statement?**

**Key functions of an OS that justify its role as a broker:**

* **User Interface**:  
  The OS provides both graphical (GUI) and command-line (CLI) interfaces, allowing users to interact with the system without needing technical knowledge of hardware.
* **Resource Management**:
  + **CPU Scheduling**: The OS allocates CPU time to different processes, ensuring each process gets fair access to the CPU.
  + **Memory Management**: The OS manages allocation and deallocation of memory to processes, to avoid conflicts.
  + **Device Management**: The OS manages input/output devices, such as keyboards, mice, printers, and storage devices. It provides a standard way for applications to access these devices, so users don’t have to manage hardware directly.
  + **File Management**: The OS organizes files and directories, making it easy for users to store and access data.
* **Security**:  
  The OS implements various security mechanisms to protect the system from unauthorized access and malicious attacks. This includes user authentication (passwords, biometrics), and encryption. The OS ensures that resources are accessible only to authorized users or applications, thus preventing data breaches and system compromises.
* **Error Handling**:  
  The OS is designed to detect, handle, and recover from errors. These errors could be hardware failures, software crashes, or resource conflicts. The OS provides meaningful error messages and takes corrective actions, such as terminating faulty processes. In critical cases, it may shut down processes or even the entire system to avoid further damage.

**What are two modes of OS? Discuss different OS structures briefly.**

**Two Modes of an Operating System:**

1. **User Mode:**
   * This is the mode in which most applications run.
   * In user mode, processes have limited access to system resources.
   * They cannot directly access hardware or critical system components.
   * Any requests for system services are made through system calls, which are then handled by the OS kernel, which operates in kernel mode.
2. **Kernel Mode:**
   * This is the privileged mode where the operating system operates.
   * The kernel has complete access to system hardware and resources.
   * It handles system-level tasks like memory management, process scheduling, device management, and file system operations.
   * It can execute special instructions that are restricted in user mode.

**Different OS Structures**

1. **Monolithic Kernel**  
   A Monolithic System is an operating system structure where all the core components (such as file management, process management, memory management, etc.) are tightly coupled together.  
   **Pros**: High performance due to close integration.  
   **Cons**: Hard to maintain, and a single error can affect the whole system.  
   **Example**: Linux (early versions)
2. **Layered Architecture**  
   A Layered System organizes the components of an operating system into hierarchical layers, where each layer performs specific functions. Each layer communicates only with the layer directly above or below it, creating a clear structure and separation of concerns.  
   **Pros**: Easier to debug and maintain, as each layer can be modified independently.  
   **Cons**: Lower performance due to layer-to-layer communication.  
   **Example**: MINIX
3. **Microkernel**
4. **Client-Server Structure**
5. **Hybrid Structure**

**Types Of Operating System**

**1. Batch Operating System:**

* Processes jobs in batches without user interaction.
* Jobs are collected and processed together in a sequence.
* Minimizes CPU idle time.
* Example: IBM 7090.

**2. Real-time Operating System (RTOS):**

* Guarantees execution of tasks within specific time frames.
* Used in systems where timing is critical (e.g., robotics, flight control).
* Ensures predictable behavior and timely response.
* Examples: QNX, VxWorks.

**3. Time-Sharing Operating System:**

* Allows multiple users to access the computer simultaneously.
* Divides CPU time into time slices for each user.
* Provides the illusion of simultaneous execution.
* Example: Unix.

**4. Server Operating System:**

* Runs on servers to manage and deliver resources to other computers.
* Supports multiple users and services like file sharing and web hosting.
* Examples: Linux, Windows Server, macOS Server.

**5. Multiprocessor Operating System:**

* Supports systems with multiple CPUs.
* Manages parallel processing for efficient task handling.
* Enhances performance and reliability.
* Examples: High-performance computing systems.

**6. Distributed Operating System:**

* Manages a group of independent computers as a single system.
* Shares resources like processing power and storage over a network.
* Ensures efficient resource allocation and coordination.
* Examples: Android, cloud-based systems.

**Differentiate between time sharing and real time system.**

A table with text on it

Description automatically generated

A table of informational text

Description automatically generated with medium confidence

**Why some process requires high priority? What would happen if all processes have some the priority? Mention merits and demerits of assigning priority on process.**

**Why Some Processes Require High Priority**

Some processes require high priority because they are critical to the system's performance or functionality. These processes often include:

1. **Real-Time Processes**: Applications that require immediate processing, such as medical monitoring systems or air traffic control systems.
2. **System Processes**: The process that handles interrupts must have high priority to prevent system instability.

**What Happens if All Processes Have the Same Priority**

If all processes have the same priority, the scheduler will typically use a fair scheduling algorithm, such as round-robin, to allocate CPU time to each process. This can lead to a situation where no process gets enough CPU time to complete its task in a timely manner, especially if there are many processes competing for resources.

**Pros**: High-priority tasks get CPU access faster, enhancing system efficiency.

**Cons**: Low-priority processes may wait indefinitely if higher-priority tasks keep arriving.

Deadlock is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources.