#college-studies #computer IT

Introduction of Operating System

1. An Operating System acts as on interface between users and computer hardware components.
2. The purpose of an Operating System is to provide an environment in which users can execute programs conveniently and efficiently.
3. An Operating System is software that manages computer hardware. The hardware must provide appropriate mechanisms to ensure correct operation of computer system and to prevent users programs from interfering with he proper operation of the system.
4. An Operating System core typically includes programs to manage these resources, such as a traffic controllers, a scheduler, memory management, I/O programs, and system utilities.

What is Program ?

- Program is set of Instructions.

What is Software ?

- Software is a set of Programs designed for specific tasks. There are basically two types of software.

System Software

- Software designed for controlling and managing the operation of computer systems is known as system software.
- Examples: Operating systems, device drivers etc.

Application Software

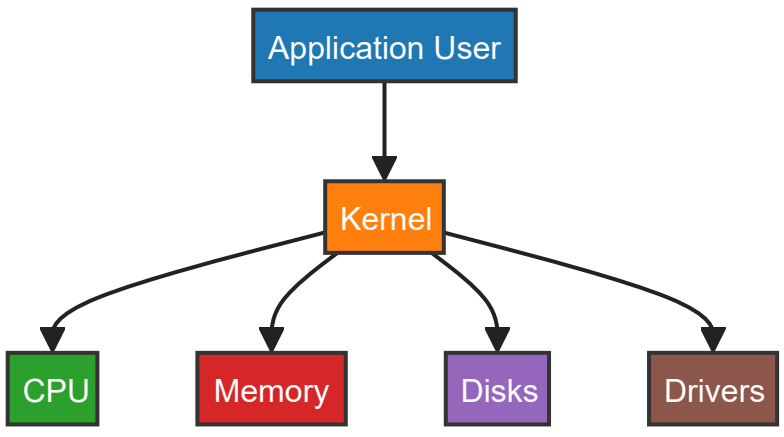
- Software designed for end-users to be used for specific purpose.
- Examples: MS-Office, Chrome, VS code, Canva, Whatsapp etc

History of Operating System

Generation	Duration	Technologies	Types of O.S
First	1940s-50s	Vacuum Tubes	-
Second	1950s-60s	transistors	Batch Systems
Third	1960s-80s	Intergrated Circuits	Multi-Programming
Forth	1980s-	-	PC

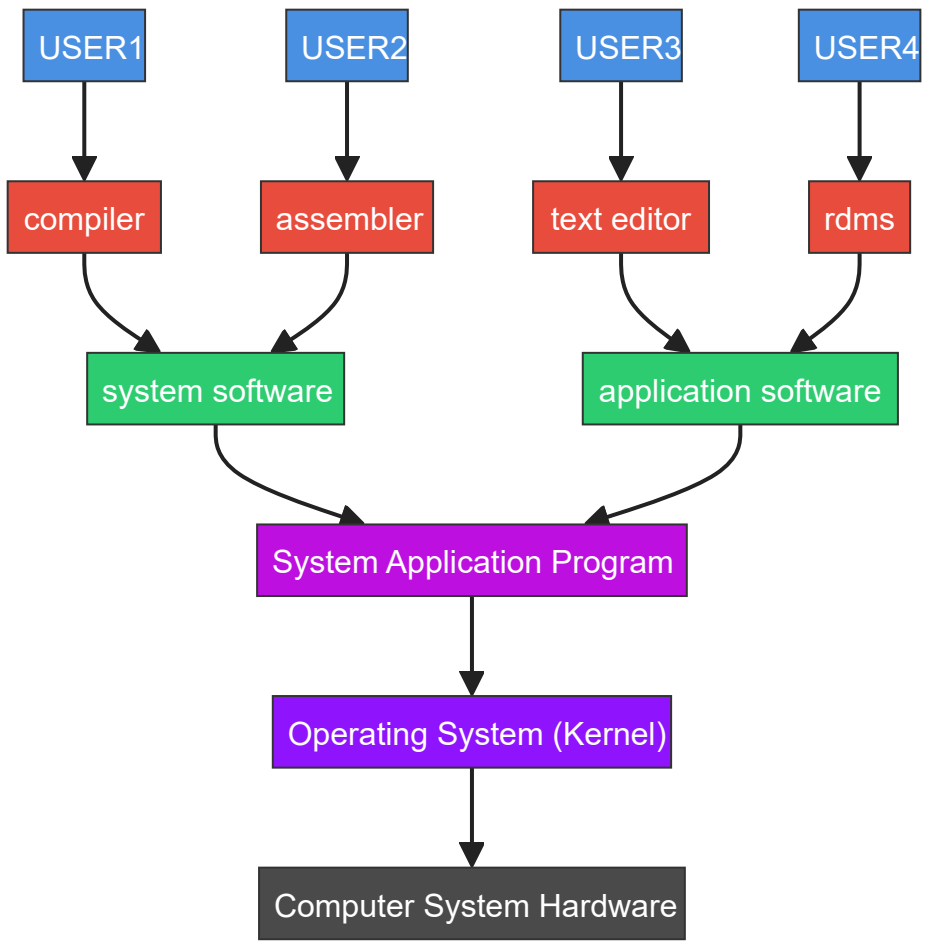
- An operating system is an interface between the user and computer system. These are numbers of operating systems, e.g. windows.
- Nowadays, windows and other operating system provide graphical user interface.
- Along with operating systems, microprocessors and operating systems have also evolved. This example of command-based operating systems which provides character-based user interfaces like CUI, GUI.

Kernel



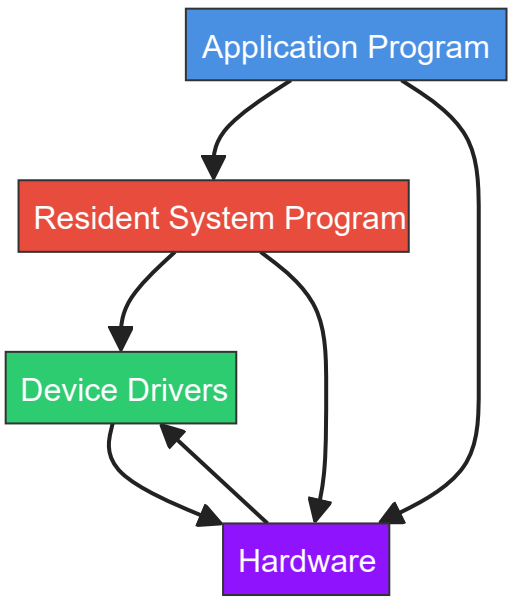
- The kernel is a computer program, at the core of operating system & generally it has complete control over the system
- Kernel prevents processes from conflicts.
- It act as a bridge between software applications and hardware components.
- The kernel manages system resources like memory, CPU and devices. Insuring everything is working smoothly and efficiently.
- It handles tasks like running different programs accessing files & connecting to device like printer, scanner etc.

Structure of Operating System



- It is the simplest structure of operating system and can be used only for small and limited system.
- It is not well-defined structure.
- Operating system acts as intermediate between user and computer system.

Simple Operating System Structure



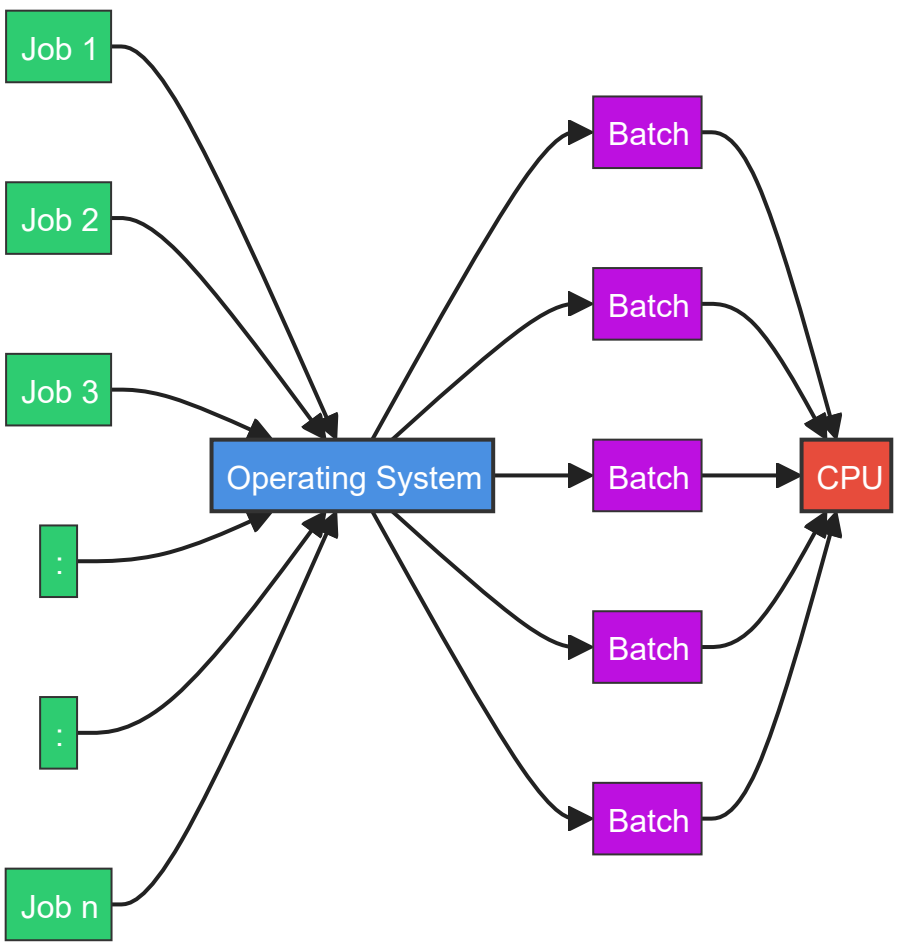
- This is a simple and not well-defined structure of operating system.

Types of Operating System

---

1. Batch Operating System

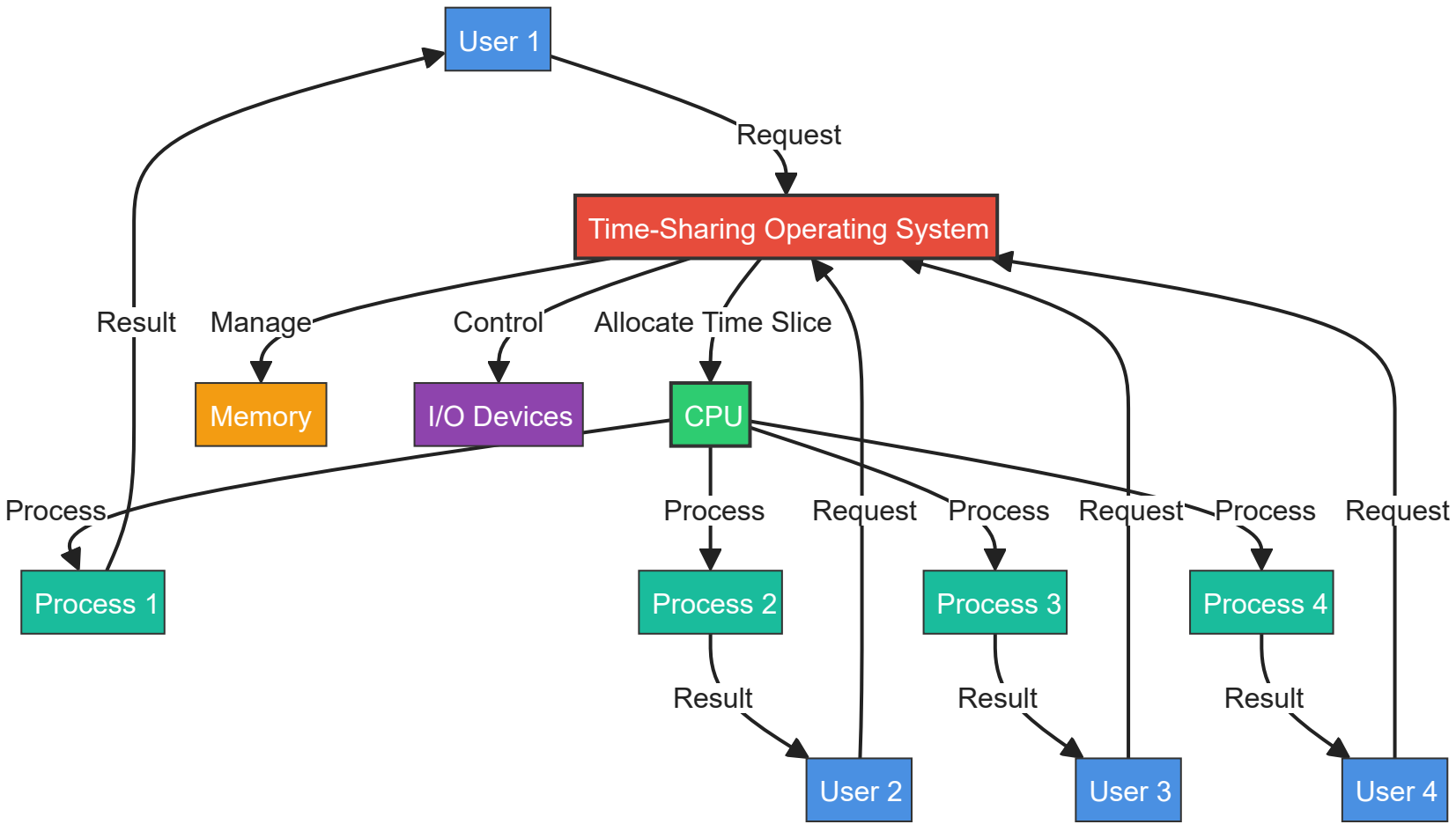
- Jobs are processed in batches without user interaction, jobs with similar needs are batched together and run through the system.
- Example: Early IBM mainframe operating system.



---

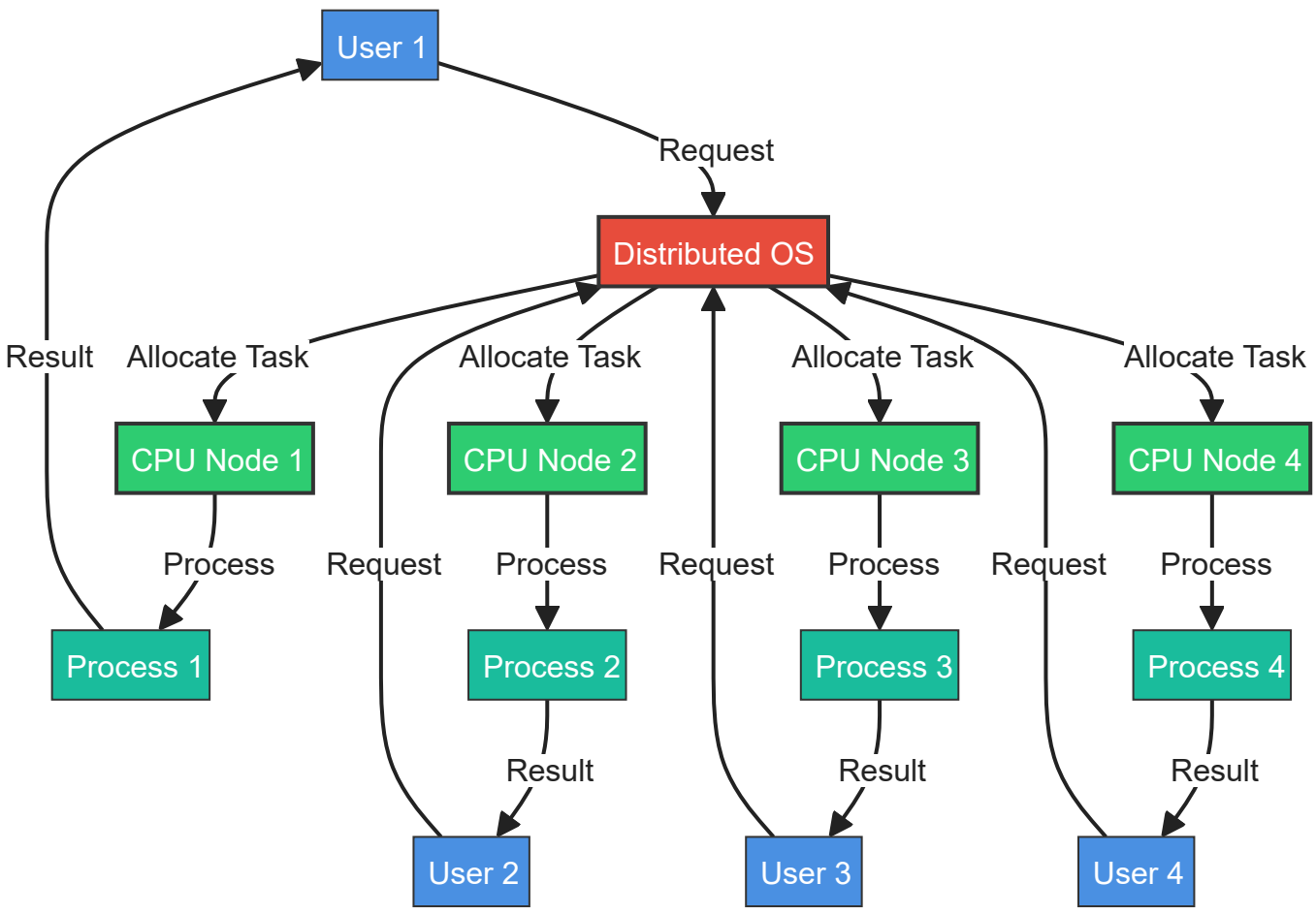
2. Time-Sharing Operating System

- Multiple Users can access the system concurrently. Time-sharing divides system time into small intervals and switches between tasks quickly, giving the illusion of simultaneous execution.
- Example: UNIX, Mulitics.



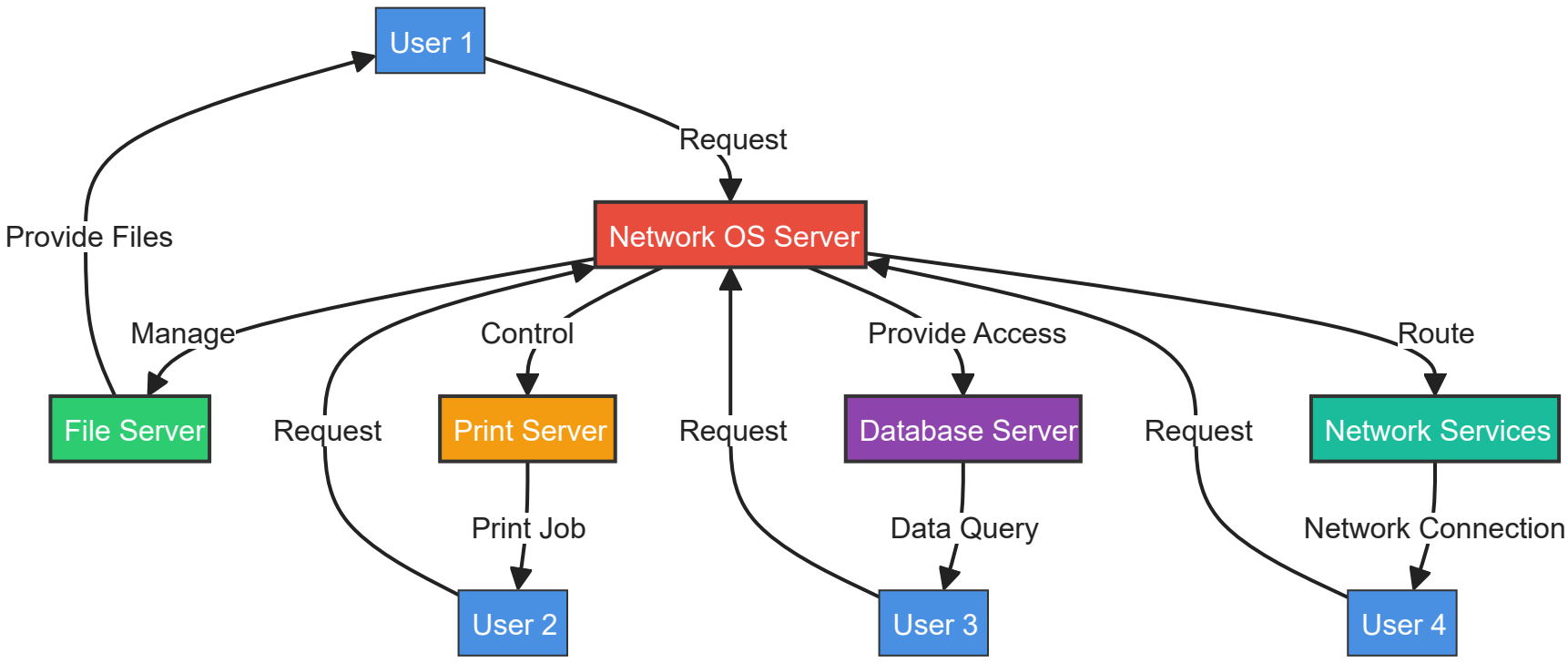
3. Distributed Operating System

- Manages a group of independent computers and makes them appear to be a single computer. It distributes computations among different machines to improve performance and efficiency.
- Example: Apache Hadoop, Amoeba.



4. Network Operating System

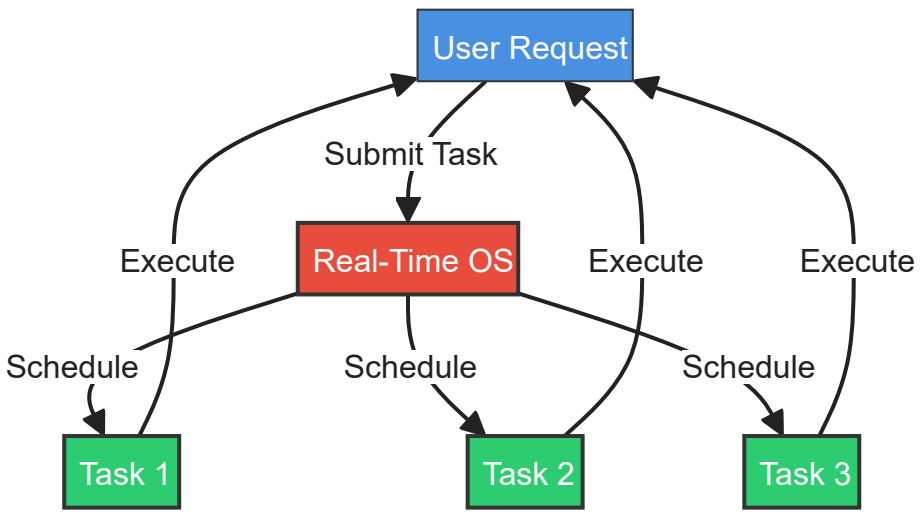
- Provides features for computers connected on network to communicate and share resources. Manages data, users, groups, security and applications over a network.
- Example; Novell Netware, Microsoft Windows Server.



---

5. Real-Time Operating System

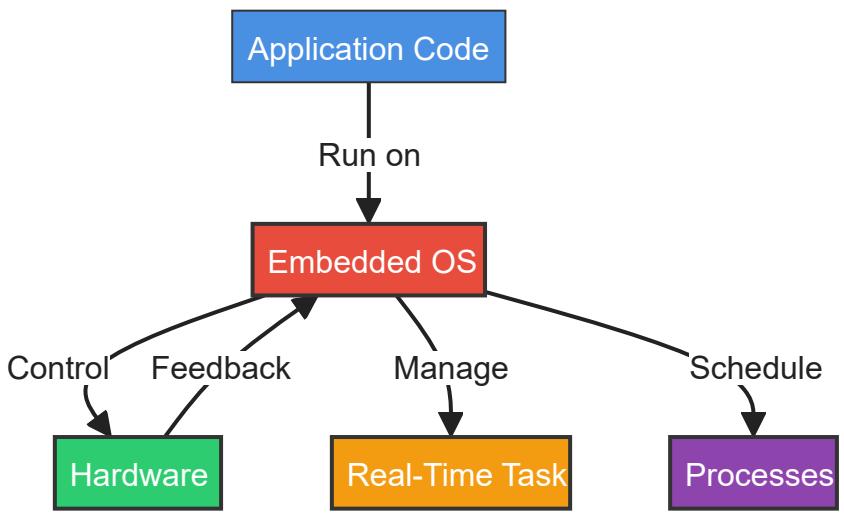
- Designed to process data as it comes in, typically without buffer delays, used in environments where time constraints are critical.
- Types:
  1. Hard Real-Time Systems:
     - Strict timing constraints, missing a deadline can result in system failure.
  2. Soft Real-Time Systems:
     - More flexible, deadlines can occasionally be missed without catastrophic consequences.
- Examples: Vxworks, Free RTOS.



---

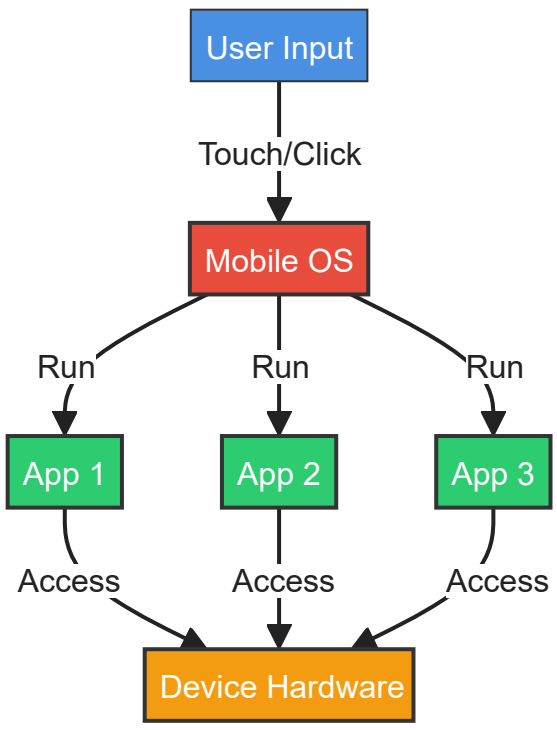
6. Embedded Operating System

- Designed to operate on small machines like PDAs with limited resources. They are typically specialized for specific tasks.
- Example: Embedded Linux, embedded windows.



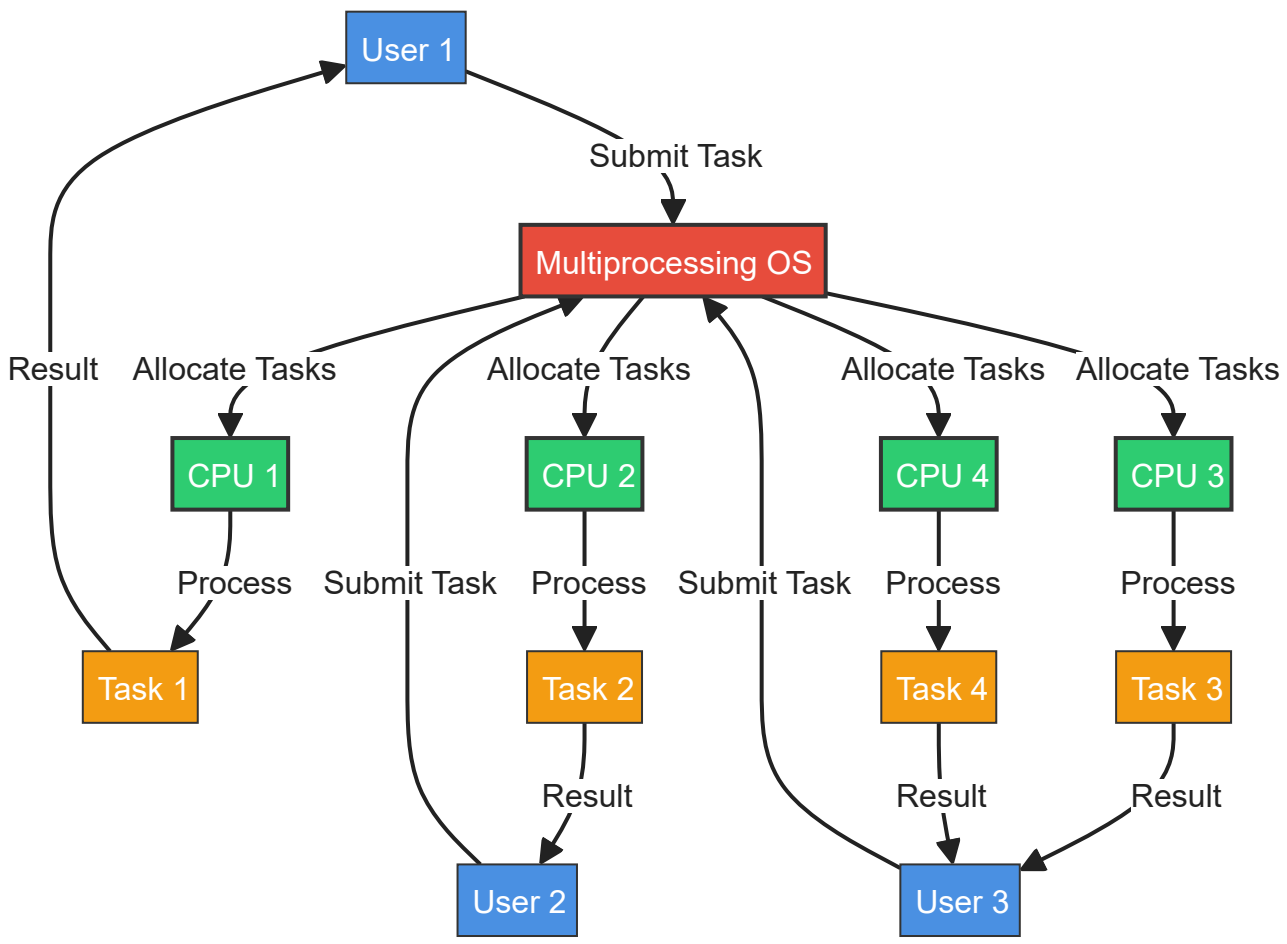
7. Mobile Operating System

- Designed specifically for mobile devices such as smartphones and tablets. These OSs are optimized for mobile specific features like touch interfaces and low power consumption.
- Example: Android, iOS, Linux.



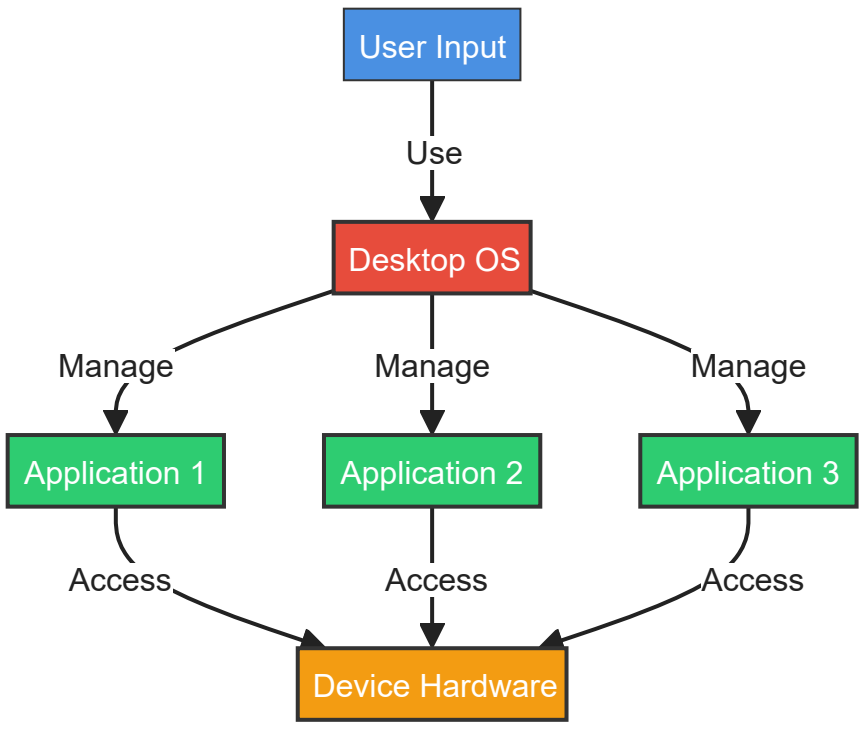
8. Multiprocessing Operating System

- Supports the use of more than one processor at the same time. These systems can execute multiple processes simultaneously.
- Example: Linux, UNIX.



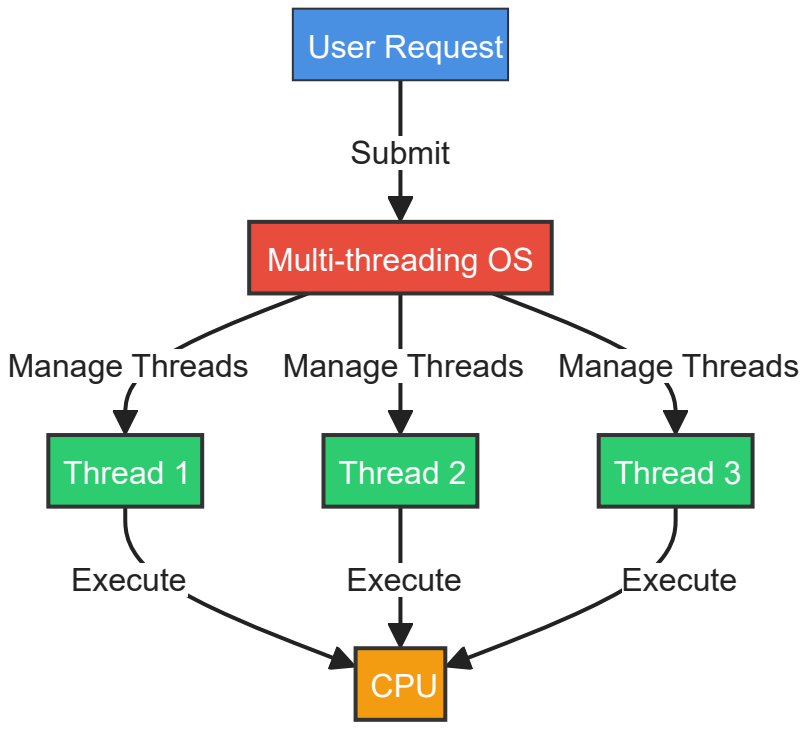
9. Desktop Operating System

- Intended for use on personal computers, these OSs support a wide range of applications and peripherals.
- Example: Microsoft Windows, macOS, Linux distributions.

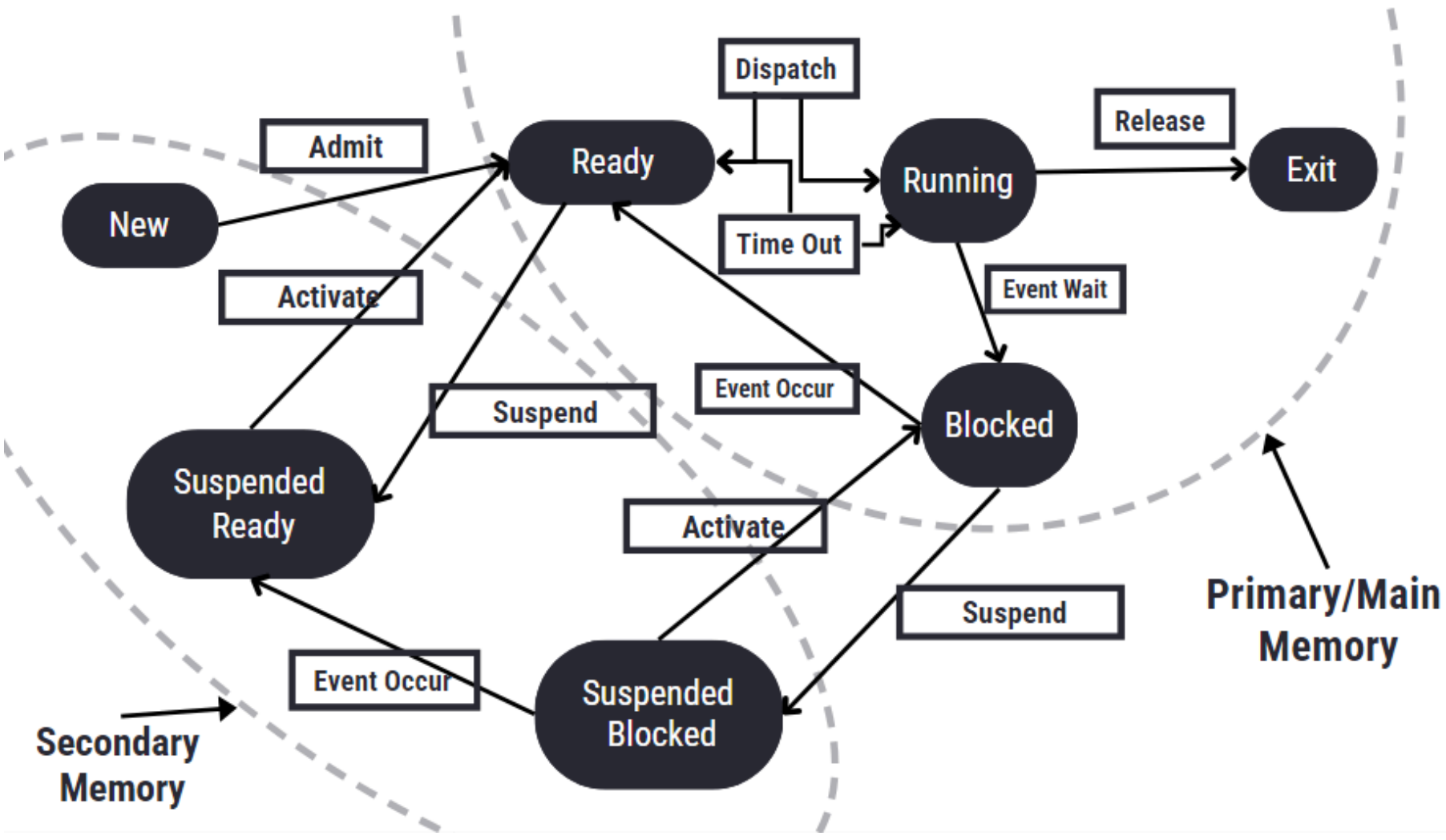


10. Multi-threading Operating System

- Allows different parts of a single programs to run concurrently, multi-threading improves the performance of applications by splitting tasks into smaller threads.
- Examples: Modern versions of windows, Linux.



Process state transition diagram



Scheduling

- Process is a program in execution, one process can have multiple threads(light weight processes) scheduling is important in many computer environments.
- One of the most important areas of scheduling is which program will work on the CPU. This is handled by the operating system. There are many different ways in which we can choose to configure program.

Process Schedulers

- Are the fundamental components of operating system, responsible for deciding the order in which processes are executed by the CPU.
- In simple terms they manage how the CPU allocates it's time among the multiple processes. That are competing for it's attention.

Process Scheduling

- Is the activity of process manager. That Handles the removal of running process from the CPU and the selection of another process based on a particular strategy.
- Process scheduling is an essential part of multi-programming operating system, such O.S allow more than one process to be loaded into executable memory at that time and the loaded process shares the CPU using time multi-plexin.

Categories

- **Primitive**
  - In this case operating system assigns resources to process for a pre-determined period. The process switches from running state to ready state or from waiting state to ready state during resource allocation this switching happens

because CPU may give other process priority and substitute the currently active process for the higher priority process.

- **Non-primitive**
  - In this case process's resource can't be taken before the process has finished running. When a running process finishes and transition to a wait state resources are utilized.

Types of Schedulers

Long-Term Schedulers

- **Purpose**: Decides which processes are admitted into the system for execution.
- **Function**: Controls the admission of new processes from a queue into the system's main memory (RAM). It helps in managing the process mix and balancing the load in the system.
- **Example**: When you start a new application on your computer, the long-term scheduler decides when and if that application should be loaded into memory for execution.

Medium-Term Schedulers

- **Purpose**: Manages processes that are in the main memory and can be swapped in and out of the main memory to the disk.
- **Function**: Controls which processes should be kept in memory and which should be temporarily moved to secondary storage (disk) to free up memory. It helps in balancing between processes that are running and those that are suspended.
- **Example**: If your computer is running low on RAM, the medium-term scheduler might move some processes to disk storage to free up memory for other processes.

Short-Term Schedulers

- **Purpose**: Decides which of the processes in memory should be executed next by the CPU.
- **Function**: Handles the process switching at a much faster rate compared to the other schedulers. It ensures that the CPU is always busy by selecting processes that are ready to run and allocating CPU time to them.
- **Example**: When you have multiple applications open, the short-term scheduler decides which application's process gets CPU time next.

Some Other Types of Schedulers

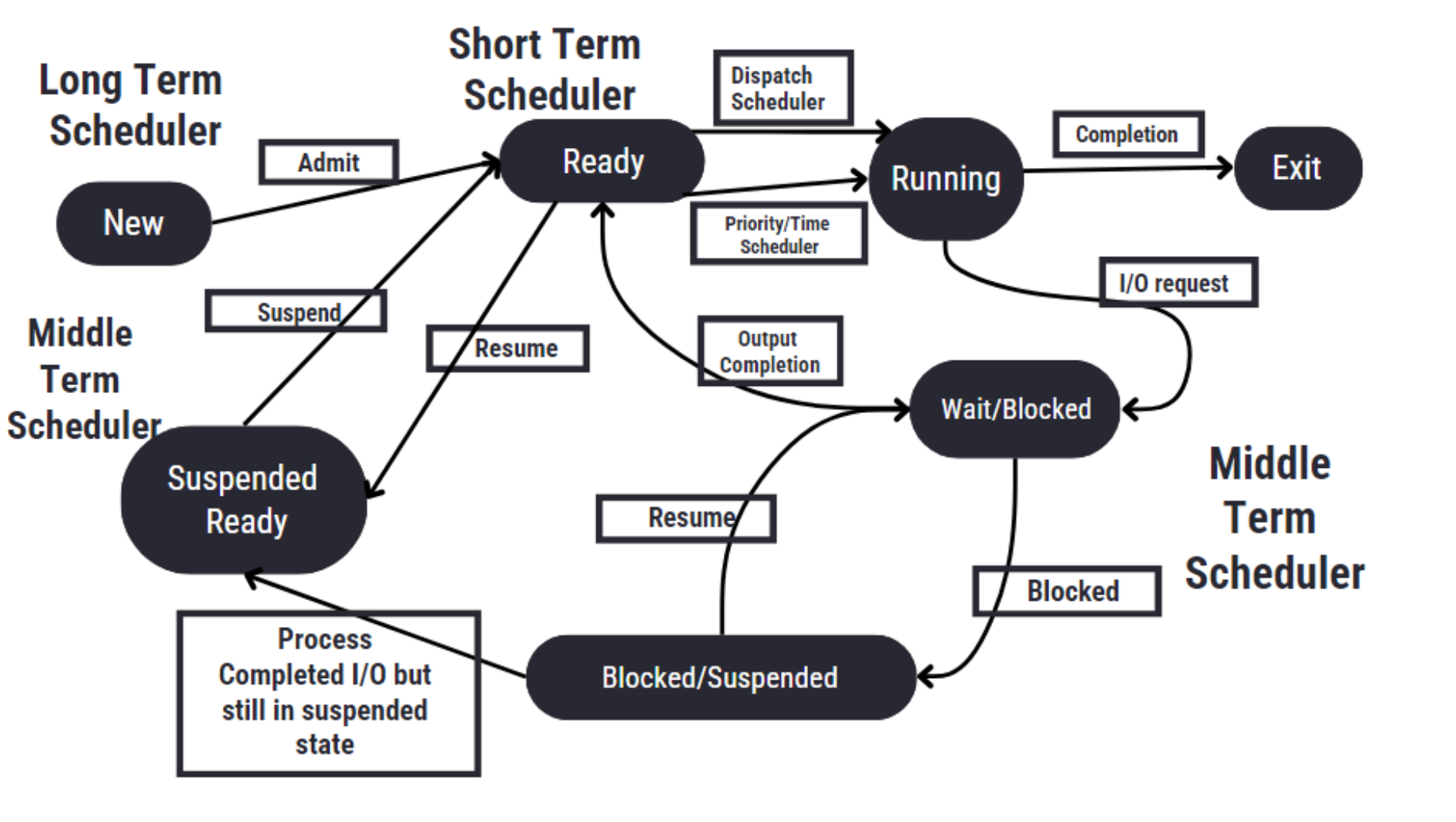
I/O Schedulers

- **Purpose**: Manages the order of I/O operations (such as reading from or writing to disk).
- **Function**: Optimizes the performance of I/O operations by deciding the order in which I/O requests should be processed. It helps in reducing waiting times and improving system efficiency.
- **Example**: If multiple programs are requesting to read from a disk, the I/O scheduler decides the sequence in which these requests are handled to minimize delays and maximize throughput.

Real-Time Schedulers

- **Purpose**: Ensures that real-time tasks are completed within specified time constraints.
- **Function**: Manages tasks that require timely and predictable execution, often used in systems where missing deadlines could lead to significant issues (e.g., embedded systems, control systems).
- **Example**: In an autonomous vehicle, the real-time scheduler ensures that the system's response to sensor inputs and control commands happens within strict time limits to maintain safe operation.

Diagram



Criteria of Scheduling

1. CPU Utilization

- **Description**: Measures the percentage of time the CPU is actively working on tasks.
- **Goal**: Maximize CPU utilization to ensure the CPU is being used effectively and not idle.

2. Throughput

- **Description**: The number of processes completed per unit of time.
- **Goal**: Maximize throughput to increase the number of processes finished in a given time frame.

3. Turnaround Time

- **Description**: The total time taken from the submission of a process to its completion.
- **Goal**: Minimize turnaround time to improve the overall efficiency and responsiveness of the system.

---

4. Waiting Time

- **Description**: The total time a process spends waiting in the ready queue before being executed.
- **Goal**: Minimize waiting time to reduce the time processes spend waiting for CPU access.

---

5. Response Time

- **Description**: The time from the submission of a request until the first response is produced (not the output).
- **Goal**: Minimize response time to improve the interactive experience for users.

---

6. Fairness

- **Description**: Ensures that each process receives a fair share of the CPU and other resources.
- **Goal**: Provide equitable access to resources for all processes, preventing starvation or unfair delays.

---

7. Priority

- **Description**: The ability to prioritize processes based on their importance or urgency.
- **Goal**: Ensure that high-priority processes receive more immediate attention and resources.

---

8. Predictability

- **Description**: The ability to predict the execution time of processes based on the scheduling algorithm.
- **Goal**: Achieve predictable performance to ensure that processes meet their deadlines, especially in real-time systems.

---

9. Overhead

- **Description**: The extra time and resources spent on managing the scheduling process, such as context switching.
- **Goal**: Minimize scheduling overhead to ensure efficient use of system resources.

---

10. Scalability

- **Description**: The ability of the scheduling algorithm to handle an increasing number of processes without significant performance degradation.

- **Goal**: Ensure the scheduler remains effective as the number of processes or the system size grows.

---

11. Resource Utilization

- **Description**: Efficient use of various system resources (CPU, memory, I/O).
- **Goal**: Maximize the utilization of all available resources to improve overall system performance.

Concurrency - Introduction

Concurrency is the ability of an operating system to execute multiple tasks simultaneously, allowing for efficient utilization of resources and improved performance. In today's computing environment, with multicore processors and high-speed networking, concurrent processing has become increasingly important for operating systems to meet user demands.

Definition of Concurrent Processes

Concurrent processing, also known as concurrent execution, refers to the ability of an operating system to execute multiple tasks or processes at the same time. It involves the parallel execution of tasks, with the operating system managing and coordinating these tasks to ensure they do not interfere with each other.

Concurrency is typically achieved through techniques such as process scheduling, multithreading, and parallel processing. It is a critical technology in modern systems, enabling them to provide the performance, scalability, and responsiveness required in today’s computing environments.

Importance of Concurrent Processing in Modern Operating Systems

---

Concurrent processing plays a significant role in modern operating systems due to the following reasons:

1. **Improved Performance**: With the advent of multicore processors, modern operating systems can execute multiple threads or processes simultaneously, leading to improved system performance. Concurrent processing allows the operating system to make optimal use of available resources, thereby maximizing system throughput.
2. **Resource Utilization**: Concurrent processing allows for better utilization of system resources, such as CPU, memory, and I/O devices. By executing multiple threads or processes concurrently, the operating system can utilize idle resources effectively, leading to enhanced resource utilization.
3. **Enhanced Responsiveness**: Concurrent processing improves system responsiveness by enabling the operating system to handle multiple tasks simultaneously. This is especially important in interactive applications, where quick responses to user inputs are critical.

Concurrency Processing in Operating Systems

Concurrency is the ability of an operating system to execute multiple tasks seemingly simultaneously, optimizing resource use and boosting performance. This is essential in modern computing due to multicore processors and high-speed networking.

Types of Concurrency Processing:

- **Process Scheduling:** This fundamental form sequentially executes multiple processes, with each receiving a time slice before the next takes over. Imagine this as a juggling act, where the operating system rapidly switches between processes to give the illusion of simultaneous execution.
- **Multithreading:** A process is divided into multiple threads that share the same memory space and resources. This division of labor within a process leads to enhanced performance and efficiency. Consider it similar to a team working on different aspects of a project concurrently.
- **Parallel Processing:** Multiple processors or cores execute tasks simultaneously, distributing the workload for significant performance gains. This is analogous to having multiple workers tackling different parts of a task simultaneously, speeding up completion.
- **Distributed Processing:** Tasks are distributed across multiple interconnected computers or nodes, enhancing scalability and fault tolerance. This is similar to a large project being divided into smaller tasks, each handled by a different team, leading to faster completion and resilience to individual team failures.

Independent Processes

Independent processes, as the name suggests, operate without sharing resources or memory. They interact through inter-process communication mechanisms, ensuring a high level of isolation and security.

Think of them as individuals working on separate projects in their own workspaces. They don't interfere with each other's work or share resources.

Dependent Processes

In contrast to independent processes, **dependent processes** rely on shared resources or data. This interdependency, while potentially increasing efficiency, introduces the risk of data inconsistency and race conditions if not managed correctly.

Imagine multiple chefs sharing the same kitchen and ingredients. While they can potentially prepare a meal faster by collaborating, they need to coordinate their actions to avoid conflicts and ensure the dish is prepared correctly.

Introduction to Process Synchronization

Process synchronization is crucial in managing dependent processes. It ensures that processes access shared resources in a controlled manner, preventing conflicts and maintaining data consistency.

This coordination is typically achieved using mechanisms like semaphores, critical sections, and monitors. These tools act like traffic signals, ensuring that only one process can access a shared resource at a time, thus preventing accidents and ensuring smooth operation.

To illustrate, consider a shared bank account accessed by multiple users (processes). Without synchronization, if two users try to withdraw money simultaneously, it could lead to an incorrect balance. Process synchronization mechanisms prevent such conflicts by ensuring that only one user can access and modify the account balance at any given time.