Introduction to OS

Purpose of a Computer System

The main purpose of a computer system is to run programs that we can execute. These programs are the tasks that we want the computer to perform, like opening a browser, editing a document, or playing a game.

Key Tasks of a Computer System:

1. **Storing an Executable Program:** When you save a program (like a game or application), it's stored on your computer's hard drive or SSD.
2. **Loading into Main Memory:** When you want to use that program, the computer loads it from the hard drive into the RAM.
3. **Setting Up CPU:** The CPU (the brain of the computer) needs to be set up to start running the program.
4. **Managing Multiple Tasks:** Sometimes, you might want to run several programs at once (like listening to music while browsing the web). The computer needs to manage these programs so they can share resources and communicate if needed.

How Does the Operating System Help?

- **RAM Management:** The operating system decides how to manage the RAM, figuring out where each program should go.
- **Finding Files on Disk:** It knows where everything is stored on your hard drive and can find it quickly.
- **Loading Programs:** It decides where in the RAM to load each program so it can run smoothly.
- **Tracking Running Programs:** It keeps track of all the programs running on your computer, making sure they don't interfere with each other.

What is an Operating System?

An operating system (OS) is like a middleman between you (the user) and the computer's hardware. It's a program that helps you run other programs and use your computer efficiently.

Operating System Goals:

1. **User Convenience:** It makes your computer easy to use.
2. **Problem Solving:** It helps you run programs that solve problems or perform tasks.
3. **Efficiency:** It ensures that the computer uses its resources (like processors, memories, timers, disks, keyboard, network interfaces, printers, memory, etc.) in the best way possible.

Top-Down View:

From a user’s perspective, the OS is what makes the computer easy to use, helping you run programs and interact with the hardware.

Bottom-Up View:

From a technical perspective, the OS is a resource manager. It allocates resources (like CPU, memory) when needed and makes sure they are used fairly and securely.

Resource Management:

- **Shareable Resources:** These are resources that can be used by multiple programs at the same time without conflict.

  Examples:

  - **Disk Storage:** Multiple programs can read and write data to the disk without interfering with each other.
  - **Network Interfaces:** Different programs can send and receive data over the network simultaneously.

- **Non-Shareable Resources:** These resources can typically be used by only one program at a time to avoid conflicts.

  Examples:

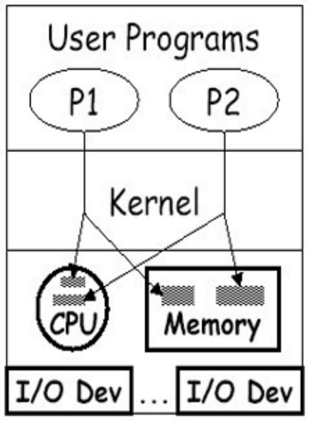
  - **CPU:** Only one program can use the CPU at a time, but the OS quickly switches between programs to give the appearance of multitasking.
  - **Printer:** Typically, only one program can send a print job at a time to avoid jumbled prints.

Resource Multiplexing:

- **Time Multiplexing:** Resources are shared over time, meaning different programs take turns using them. For example, CPU scheduling allows multiple programs to run by giving each one a small amount of time to use the CPU. Timers, Printer and Keyboard also follow time multiplexing.
- **Space Multiplexing:** Resources are divided so different programs can use them simultaneously. For example, RAM is split among multiple running programs.

Kernel, Bootstrap Loader, Booting:

- **Kernel:** The core part of the OS that manages all resources and communicates with the hardware.
- **Bootstrap Loader:** A small program that loads the OS into the memory when the computer starts.
- **Booting:** The process of starting up the computer and loading the operating system.



Early Systems in Operating Systems

No Operating Systems:

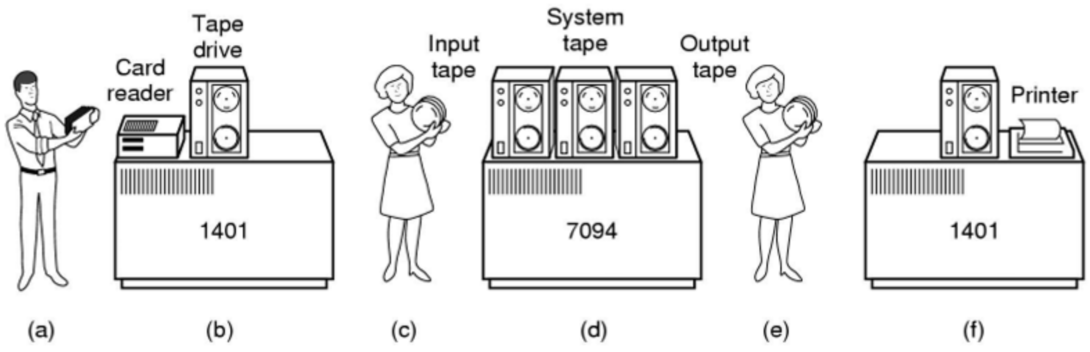
In the early days, computers didn't have operating systems. Users had to manually enter programs using binary switches, which was very slow and complicated. For example, adding two numbers required direct manipulation of registers (temporary storage locations in the CPU) and memory locations.

Problems with Early Systems:

- **Idle Time:** Computers often sat idle while programmers set things up. This was a waste of the machine's potential, especially since these early computers were expensive investments.
- **Solution:** At first, companies hired specialized operators to set up the machine, but even they were much slower than the computer itself.

Introduction of Batch Processing:

To address the inefficiencies, batch processing systems were developed. Instead of having the computer idle while waiting for a new program, jobs were grouped together into batches and processed one after the other.



How Batch Systems Worked:

- **Preparation:** A user would prepare a job that included the program, data, and instructions on how to run it. This job was submitted to the computer on punch cards or tapes.
- **Processing:** The jobs were then grouped by operators based on their needs (e.g., all FORTRAN programs) and fed into the computer in batches.
- **Monitor Program:** A special program called the **"monitor"** would manage the execution of each job in the batch. The monitor resided in the computer's main memory and would read jobs one by one, load them into memory, and start them.
- **Output:** After running the program, the system would generate output, which included the program's result and a memory dump for debugging purposes. The memory dump showed the contents of memory and registers at the end of the program, helping programmers find any errors.

Challenges of Early Batch Systems:

CPU Idle Time: Even with batch processing, the CPU often sat idle while waiting for slower input/output (I/O) devices like tape drives to catch up. This inefficiency was because mechanical I/O devices were much slower than electronic components.

Beyond Batch Processing Systems

Multiprogramming:

Definition: Multiprogramming is a method where the computer runs multiple jobs at the same time. However, the CPU can only execute one instruction at a time, so it rapidly switches between jobs.

Purpose: The main goal of multiprogramming is to keep the CPU busy by organizing jobs so that there's always something for the CPU to do.

How It Works:

- The operating system keeps several jobs in memory at once.
- The CPU executes one job until it needs to wait for something (like an input/output (I/O) operation).
- Instead of sitting idle, the CPU switches to another job that is ready to run.

Hardware Requirements for Multiprogramming:

- **I/O Interrupts:** The system uses interrupts to handle input/output operations efficiently. For example, the OS can request an I/O operation and then go back to computing until the I/O is finished.
- **Direct Memory Access (DMA):** This allows data to be transferred directly between memory and an I/O device without CPU intervention, freeing up the CPU for other tasks.
- **Memory Management:** The system must manage memory carefully to keep multiple jobs ready to run. This includes memory protection to ensure that one job doesn't interfere with another's data.

Performance Criteria in Multiprogramming:

Turnaround Time: This is the time taken from when a job starts to when it completes. The goal is to minimize this time to make the system more efficient.

Time Sharing Systems (TSS):

Definition: Timesharing allows multiple users to use the computer simultaneously by sharing the CPU's time among them.

Purpose: Timesharing increases CPU utilization and gives each user a sense that they have their own dedicated computer, even though they are sharing the system with others.

How Time Sharing Works:

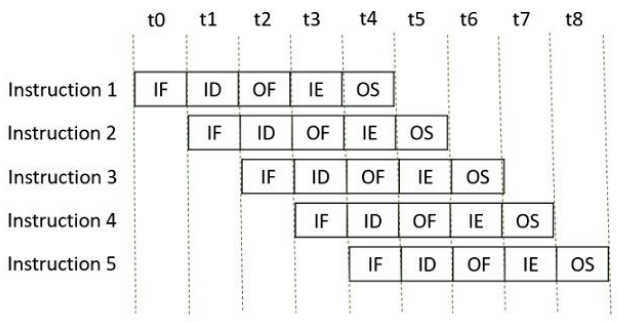
- **Resource Sharing:** Time sharing is achieved by sharing the CPU and other resources among different jobs.
- **Interactive System:** In a time-sharing system, multiple users can interact with their programs simultaneously. Each user's program is given a small slice of the CPU's time, creating the illusion that each user has their own processor.
- **Swapping Jobs:** To ensure a reasonable response time, jobs might need to be swapped in and out of main memory.

Examples of Time Sharing Systems:

Operating Systems: UNIX, Linux, Windows NT Server, and Windows 2000 Server are examples of time-sharing systems.

Performance in Time Sharing:

- **Response Time:** This is the time it takes from starting a job to getting the first output. The aim is to reduce response time to keep users satisfied.



Instruction Pipelining of Five Instructions

- **Utilization Challenges:** While time-sharing systems aim for better user response times, they may sacrifice some CPU efficiency. For example, if a user is slow in entering data (like typing on a keyboard), the CPU might still have to wait, although it can switch to other jobs during this time.

Benefits of Multiprogramming and Timesharing:

- **Increased CPU Utilization:** By ensuring the CPU always has something to execute, multiprogramming and timesharing reduce the amount of time the CPU sits idle.
- **Efficient Use of Memory:** The operating system keeps multiple jobs in memory, ready to be executed when the CPU is free.

Challenges:

- **Limited Memory:** Usually, the number of jobs that can be kept in memory is much smaller than the number of jobs waiting in the job pool.
- **Job Scheduling:** The OS must decide which jobs to bring into memory and which job to execute next.

Real-Time Systems:

A real-time system is designed to respond to inputs and produce outputs within a specific time frame. If the system fails to do so, it could lead to system failure. These systems are critical in environments where timing is crucial.

Types of Real-Time Systems:

1. **Hard Real-Time Systems:**

   Definition: These systems must meet strict deadlines. All tasks must be completed within a fixed time, and any delay can cause the system to fail.

   Characteristics:

   - **Strict Timing:** All delays in the system must be controlled and predictable (bounded).
   - **Limited or No Secondary Storage:** These systems often avoid using hard drives or other forms of secondary storage because accessing data from these can be slow. Instead, they use short-term memory (RAM) or read-only memory (ROM).
   - **Minimal Features:** Advanced features found in general-purpose operating systems (like multitasking) are often absent to ensure that the system can meet its timing requirements.

   Examples: Systems used in medical devices like pacemakers, industrial robots, and aviation control systems.

2. **Soft Real-Time Systems:**

   Definition: In these systems, meeting deadlines is important but not as strict as in hard real-time systems. A critical task is given priority over others, but slight delays are acceptable.

   Characteristics:

   - **Priority Handling:** Critical tasks are prioritized over other tasks and keep this priority until they are completed.
   - **Bounded Delays:** The delays in executing tasks are kept within acceptable limits, but they don’t have to be as tightly controlled as in hard real-time systems.
   - **Flexibility:** Soft real-time systems can coexist with other types of systems, such as time-sharing systems, without conflict.

   - **Examples:** Streaming media applications, online gaming, and mobile phone systems.

Personal vs. Distributed Computing

Personal Computing:

- **Definition:** Personal computing refers to the use of a single computer by an individual for tasks like word processing, browsing the internet, gaming, or running applications. These tasks are typically executed on a single machine, often referred to as a personal computer (PC).
- **Characteristics:**
  - **Single User:** Usually, personal computers are designed for use by one person at a time.
  - **Local Resources:** The computing power, storage, and software are all contained within the individual machine.
  - **Examples:** Laptops, desktops, and tablets used by individuals.

Distributed Computing:

- **Definition:** Distributed computing involves a network of computers working together to solve a common problem or perform a task. The workload is divided among multiple machines, each contributing to the overall computation.
- **Characteristics:**
  - **Multiple Machines:** Unlike personal computing, distributed computing involves several computers that may be spread across different locations.
  - **Resource Sharing:** Resources such as processing power, storage, and data are shared across the network of computers.
  - **Examples:** Cloud computing services (like Google Cloud or AWS), distributed databases, and scientific computing projects that use the combined power of many computers to solve complex problems.