**Unit Learning Outcomes (ULOs):**

1. **Assess programming paradigms:** Understand and select suitable programming paradigms based on problem types.
2. **Design and implement programs:** Write efficient programs using different paradigms.
3. **Apply theoretical concepts:** Analyze and critique the design of programs based on theoretical principles.

**Key Concepts:**

1. **Real-time Systems**
   * Must satisfy strict timing constraints to function correctly (e.g., anti-skid braking systems, traffic lights).
   * **Hard real-time systems:** Strict time-bound systems where failure can lead to serious consequences.
   * **Soft real-time systems:** Less strict timing; typical computer systems.
2. **Computer Architecture:**
   * **Von Neumann architecture:** Uses the same memory for data and instructions.
   * **Harvard architecture:** Separate memory for data and instructions.
3. **Operating Systems:**
   * Provides resource management (memory, CPU scheduling, file systems, etc.).
   * **Process:** A program in execution, needs resources like CPU time and memory.
   * **Thread:** Smallest unit of a process, allows multiple tasks at once within a process.
4. **CPU Scheduling:**
   * Decides which process should use the CPU at a given time.
   * **Non-preemptive:** Process runs until completed.
   * **Preemptive:** CPU is shared, switching tasks after a fixed time (e.g., Round-Robin).
5. **Interrupts and Polling:**
   * **Interrupts:** Hardware signals the CPU when an event occurs.
   * **Polling:** CPU checks periodically for events, less efficient.
6. **Context Switching:**
   * When switching between processes, the system saves the current process’s state and restores another process’s state.

Detailed explanation:  
**Key Concepts and Terms:**

**1. Real-Time Systems:**

* **Definition:** A system that must deliver its output within a strictly defined time frame, or risk failure. The system’s performance is judged not only by the correctness of the result but also by the time it takes to deliver it.
* **Examples of Real-Time Systems:**
  + **Hard Real-Time Systems:** Systems with strict timing constraints, such as anti-lock braking systems (ABS) in cars, where failure to act within a specific timeframe could lead to catastrophic failure.
  + **Soft Real-Time Systems:** Systems that aim to meet deadlines but can tolerate some level of delay, such as video streaming services or online games.

**2. Computer Architecture:**

* **Von Neumann Architecture:**
  + Uses a single memory space for both data and instructions.
  + Simple design where the CPU fetches instructions and data from the same memory.
  + Commonly used in general-purpose computers (like PCs and laptops).
* **Harvard Architecture:**
  + Separates memory spaces for data and instructions.
  + Faster and more efficient because instructions and data can be fetched simultaneously.
  + Often used in embedded systems like microcontrollers.

**3. Operating Systems (OS):**

* **Definition:** Software that manages hardware and software resources, providing services for application programs. Key functionalities include memory management, CPU scheduling, file system management, and I/O handling.
* **Types of OS:**
  + **Single-Tasking:** Can only run one task at a time.
  + **Multi-Tasking:** Allows multiple tasks to run concurrently, sharing CPU time.
  + **Distributed OS:** Manages multiple computers working together as a single system.
  + **Embedded OS:** Designed for systems with limited resources, typically found in devices like washing machines or medical equipment.
  + **Real-Time OS (RTOS):** Special OS designed for systems with strict timing constraints.

**4. Processes and Threads:**

* **Process:** A program in execution that has its own memory space and resources. A process can perform tasks independently.
* **Thread:** A lightweight process. Multiple threads can run within a process, sharing the same memory space but having their own stack and registers. This enables multitasking within a single process, allowing more efficient use of system resources.
* **Example of Threads Usage:**
  + A web server can handle multiple client requests simultaneously by assigning a thread to each client while sharing resources like memory and cache.

**5. CPU Scheduling:**

* **Definition:** The method by which the OS determines which process or thread should execute at any given time.
* **Key Terms:**
  + **Arrival Time:** The time when a process enters the system.
  + **Completion Time:** When the process finishes execution.
  + **Burst Time:** The time required by a process to complete its execution.
  + **Waiting Time:** The time a process spends waiting in the ready queue before it gets CPU time.
* **Scheduling Algorithms:**
  + **First-Come, First-Served (FCFS):** Processes are executed in the order they arrive. Simple but can lead to long waiting times for processes arriving later.
  + **Round-Robin:** Each process gets a fixed time slice to execute. If a process doesn’t complete in that time, it is moved to the back of the queue.
  + **Priority Scheduling:** Processes with higher priority are executed first, but low-priority processes may starve if high-priority processes keep arriving.

**6. Interrupts vs Polling:**

* **Interrupts:** The CPU is interrupted when a device needs attention, and it temporarily stops what it's doing to handle the device request. More efficient as it doesn’t waste CPU time.
* **Polling:** The CPU continuously checks whether a device needs attention. Less efficient as it consumes CPU resources even when there’s nothing to do.
* **Context Switching:** When the OS switches the CPU from one process to another, it must save the current state (context) of the process and load the saved state of the new process. This process adds overhead but is necessary for multitasking.

**Advanced Real-Time Concepts:**

* **Interrupt Service Routine (ISR):** A function that handles interrupts when they occur, pausing the current task, and executing the appropriate actions for the interrupting event. ISRs need to be quick and efficient to avoid delaying other tasks.
* **Critical Sections and Atomic Access:** A part of the program that must be executed without interruptions to avoid data corruption, especially in multithreaded environments.

A circuit board with many different colored wires

Description automatically generated