1. Baremetal Programming

Definition

- Baremetal programming means writing software that runs directly on the microcontroller (MCU) or microprocessor (MPU) without any operating system.
- The application code directly interacts with the hardware registers, peripherals, and memory.
- You are the scheduler — you decide the exact order in which code runs.

How It Works

- The program starts from the `reset` vector after power-on.
- The compiler and linker generate a binary that includes all the code, which is loaded into flash memory.
- No OS kernel, no multitasking — just your code + hardware.
- Commonly used in resource-constrained devices (low RAM, low clock speeds).

Advantages

1. Maximum Performance — No OS overhead.
2. Minimal Resource Use — Can run in a few KB of flash/RAM.
3. Full Hardware Control — Direct access to registers without abstraction.

Disadvantages

1. No Built-In Multitasking — Must implement cooperative scheduling, interrupts, or state machines manually.
2. Difficult Maintenance — As the project grows, code becomes harder to organize.
   Portability Issues — Code is tightly coupled to specific hardware.

Examples

- 8051-based temperature controller.
- PIC microcontroller blinking LEDs.
- STM32 baremetal driver for UART or SPI.

2. RTOS-Based Programming

Definition

- RTOS (Real-Time Operating System) is a small OS optimized for deterministic, predictable timing and task scheduling.
- It allows multiple tasks (threads) to run concurrently with priority-based scheduling.

How It Works

- Your application is split into tasks.
- The RTOS kernel schedules which task runs at any time.
- Offers inter-task communication tools like semaphores, queues, and mutexes.
- Can run directly on bare-metal hardware (no MMU needed).

Advantages

1. Real-Time Capabilities — Can meet strict deadlines.
2. Multitasking Made Easy — No need to manually manage task switching.
3. Modular Development — Each task is independent, easier to debug.
4. Hardware Abstraction — Still close to hardware but with better organization.

Disadvantages

1. More Resources Needed — Requires extra RAM & flash for the RTOS kernel.
2. Slight Performance Overhead — Context switching consumes CPU cycles.
3. Learning Curve — Must understand RTOS APIs, scheduling, and synchronization.

Examples

- FreeRTOS running on STM32 for industrial automation.
- Drone flight controller (PX4 uses NuttX RTOS).
- ESP32 Wi-Fi control with FreeRTOS tasks.

3. OS-Based Programming

Definition

- Runs on top of a general-purpose operating system like Linux, Windows, or Android.
- The OS kernel manages hardware, multitasking, drivers, networking, and memory.
- Your program runs in user space, and hardware is accessed through OS APIs.

How It Works

- The CPU first boots into the OS kernel.
- The OS handles hardware via device drivers.
- Your application runs as a process, isolated from the kernel for safety.
- Often runs on MPUs with high RAM, MMU, and storage.

Advantages

1. Rich Features — Networking, GUI, file systems built-in.
2. Huge Ecosystem — Libraries, packages, and development tools.
3. Better Portability — Same program can run on different hardware if OS-compatible.

Disadvantages

1. High Resource Needs — Requires more RAM, storage, and CPU power.
2. Longer Boot Time — Not instant like baremetal or RTOS.
3. Non-Deterministic Timing — Hard to guarantee microsecond-level deadlines.

Examples

- Raspberry Pi running Linux for IoT gateway.
- Android-based infotainment system.
- Vision processing with AI on NVIDIA Jetson.