4.5 Real-Time operating and control system:

Operating System Basics

An operating system (OS) is a system software that acts as an intermediary between hardware and user applications. It manages hardware resources, provides a user interface, and facilitates the execution of application programs. Key functions of an OS include process management, memory management, file system management, I/O device management, and ensuring system security and protection. Examples of operating systems include Windows, Linux, macOS, and real-time operating systems (RTOS) like VxWorks.
The OS can be classified into different types based on functionality:
Batch Operating Systems: Execute batches of jobs without user interaction.
Time-Sharing Operating Systems: Allow multiple users to interact with the system simultaneously.
Real-Time Operating Systems (RTOS): Ensure tasks are executed within specific timing constraints.
Distributed Operating Systems: Enable resource sharing across multiple systems connected via a network.

Task

A task is a basic unit of work in a computing environment, often representing a program or a process to be executed. In embedded or real-time systems, tasks are defined to perform specific operations, such as reading sensor data or controlling actuators. Tasks can be classified into:
Periodic Tasks: Execute at regular intervals.
Aperiodic Tasks: Execute sporadically, triggered by events.
Critical Tasks: Must meet strict timing deadlines to ensure system reliability.

Process

A process is an executing instance of a program, consisting of the program code, data, and execution context. The OS allocates resources like CPU time, memory, and I/O devices to processes. Each process has a lifecycle with distinct states:
New: Process is being created.
Ready: Process is waiting for CPU allocation.
Running: Process is executing on the CPU.
Blocked/Waiting: Process is waiting for an I/O operation or event to complete.
Terminated: Process execution is completed.
Processes are isolated, and inter-process communication (IPC) mechanisms, such as shared memory, message passing, or pipes, are used for collaboration between processes.

Threads

Threads are lightweight processes that share the same address space but have independent execution flows. A process may consist of multiple threads, enabling concurrent execution. Threads are categorized into:
User-Level Threads: Managed by user-level libraries without kernel intervention.
Kernel-Level Threads: Managed directly by the OS kernel, offering better integration but higher overhead.
Threads improve performance by enabling multitasking within a single process, such as updating a GUI while processing background tasks.

Multiprocessing

Multiprocessing involves the use of two or more CPUs (or cores) in a single computer system to execute processes simultaneously. It is a form of parallel processing aimed at improving performance and reliability. Multiprocessing systems can be:
Symmetric (SMP): All processors share the same memory and perform identical tasks.
Asymmetric (AMP): Processors are specialized for specific tasks, with one acting

as the master.
Multiprocessing enables load balancing, fault tolerance, and efficient utilization of system resources.

Multitasking

Multitasking allows a CPU to handle multiple tasks or processes seemingly simultaneously by rapidly switching between them. This is achieved through context switching, where the OS saves the state of one task and loads the state of another. Multitasking can be:
Preemptive: The OS forcibly interrupts a task to switch to another.
Cooperative: Tasks voluntarily yield control to the OS.
Multitasking enhances system responsiveness and resource utilization.

Task Scheduling

Task scheduling determines the order and timing of task execution to optimize system performance. Common scheduling algorithms include:
First-Come-First-Serve (FCFS): Tasks are executed in the order they arrive.
Shortest Job Next (SJN): Executes the shortest task first, minimizing average wait time.
Round Robin (RR): Allocates a fixed time slice to each task in a cyclic manner.
Priority Scheduling: Tasks with higher priority are executed before lower-priority ones.
Real-Time Scheduling: Focuses on meeting deadlines, with algorithms like Rate-Monotonic Scheduling (RMS) and Earliest Deadline First (EDF).

Task Synchronization

Task synchronization ensures that multiple tasks access shared resources safely without conflicts. Common synchronization techniques include:
Semaphores: Counting mechanisms to control access to a shared resource.
Mutexes: Mutual exclusion locks that allow only one task to access a resource at a time.
Monitors: High-level synchronization constructs combining mutexes and condition variables.
Barriers: Ensure all tasks reach a specific point before proceeding.

Device Drivers

Device drivers are specialized software that act as intermediaries between the operating system and hardware devices. They enable the OS and applications to communicate with hardware without needing to know the device's underlying details. A driver abstracts the hardware's functionality and presents a standardized interface to the OS.
Key Functions of Device Drivers are:
Device Initialization: The driver sets up the hardware by configuring registers, enabling power, and preparing it for communication.
Data Transfer: The driver handles data exchange between the hardware and memory using methods like programmed I/O, interrupt-driven I/O, or DMA (Direct Memory Access).
Interrupt Handling: When a device generates an interrupt (e.g., indicating task completion), the driver processes it, ensuring the system responds appropriately.
Error Handling: Drivers detect and handle hardware faults or communication errors, logging errors for diagnostics.
Control Commands: Drivers allow software to send specific commands to the hardware, such as changing modes or settings.
Types of Device Drivers are:
Character Drivers: Manage character-oriented devices like keyboards and serial ports, processing data one character at a time.
Block Drivers: Handle devices like hard drives, organizing data into fixed-size blocks for storage and retrieval.
Network Drivers: Facilitate communication over networks by implementing

protocols like TCP/IP.
Virtual Device Drivers: Simulate hardware functionality in software, often used in virtual machines.
Driver Development
Driver development requires understanding the hardware's architecture and operating system internals. Drivers are typically written in low-level languages like C or assembly and operate in kernel mode, granting them direct access to hardware but requiring careful error handling to avoid system crashes.

Open-loop control System overview

An open-loop control system is a type of control system that operates based solely on predefined inputs without monitoring the output. It assumes a predictable relationship between input and output but does not adjust its behavior based on actual performance.
Characteristics of Open-Loop Control Systems:
No Feedback: The system lacks a feedback loop to monitor or correct output.
Simple Design: The absence of feedback makes the system easier to design and implement.
Low Cost: Fewer components are needed, making it cost-effective.
Limited Accuracy: Open-loop systems cannot compensate for disturbances or changes in system dynamics.
Examples:
A toaster operates for a fixed time based on the userâ ™s input, regardless of the actual toasting level.
Traffic lights operate on a predefined timer without sensing real-time traffic conditions.
Mathematical Representation
Open-loop systems can be described by a transfer function, G(s), representing the system's response to an input X(s): Y(s)=G(s)â‹…X(s) Where Y(s) is the output.

Close-Loop control System overview

A close-loop control system uses feedback to compare the actual output with the desired reference input. The system adjusts its input based on the error signal (difference between reference and actual output) to achieve the desired performance.

Characteristics of Close-Loop Control Systems
Feedback: Continuous monitoring of output to ensure accuracy.
Higher Accuracy: Able to adapt to disturbances and maintain performance.
Complex Design: Requires additional components like sensors and controllers.
Stability Challenges: Feedback loops must be carefully designed to avoid instability.
Examples
A thermostat adjusts the heating or cooling based on the difference between the room temperature and the desired setpoint.
Cruise control in cars maintains a set speed by adjusting throttle input based on road conditions.
Mathematical Representation
Close-loop systems are represented as: T(s)= G(s)/(1+G(s)H(s)) Where:
T(s) is the overall transfer function.
G(s) is the plant's transfer function.
H(s) is the feedback path's transfer function.

Components of a Close-Loop System
Sensor: Measures the output and sends feedback.
Controller: Computes the error and determines corrective actions.
Actuator: Implements the controller's output to adjust the system.
Plant: The system being controlled.

Control:
Control engineering focuses on designing systems to achieve desired outputs by

manipulating inputs based on feedback or predefined rules. Control systems are categorized as:
Continuous-Time Control: Operates on analog signals and is described using differential equations.
Discrete-Time Control: Processes sampled signals and uses difference equations.
Types of Control Strategies
Proportional-Integral-Derivative (PID) Control:
PID controllers combine three control actions:
Proportional (P): Produces an output proportional to the error.
Integral (I): Accounts for accumulated error over time.
Derivative (D): Predicts future error based on its rate of change. The control action is described as:
u(t)=Kp e(t)+Ki âˆ«e(t)dt+Kd de(t)/dt
â ‹Where:
u(t): Control output.
e(t): Error signal.
K p,Ki,Kd: Proportional, integral, and derivative gains.
Feedforward Control: Uses knowledge of the system to preemptively adjust inputs without relying on feedback.
Adaptive Control: Adjusts control parameters dynamically to handle system variations or disturbances.
Applications:
Control systems are vital in industries such as:
Automotive: Anti-lock braking systems and engine control units.
Aerospace: Flight stabilization and autopilot systems.
Manufacturing: Robotic arm movement and conveyor belt operations.