Lecture
6

# Embedded Systems

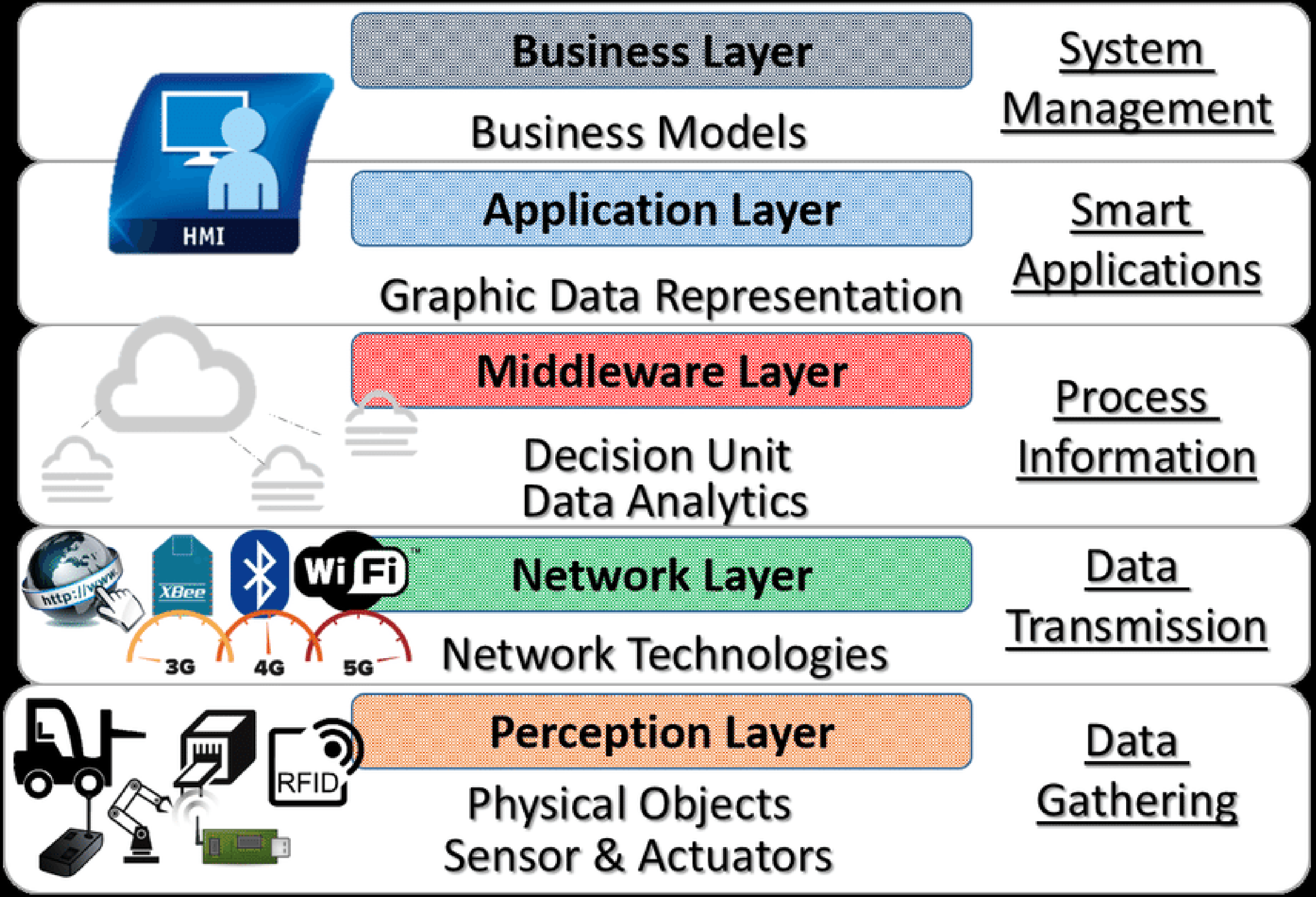
# RTOS II

Associated Prof. Wafaa Shalash

Lect. 6

## Class Rules

- Be in class on time,
- Listen to instructions and explanations.
- Talk to your classmates only when there is an activity.
- Use appropriate and professional language.
- Keep your mobile silent.



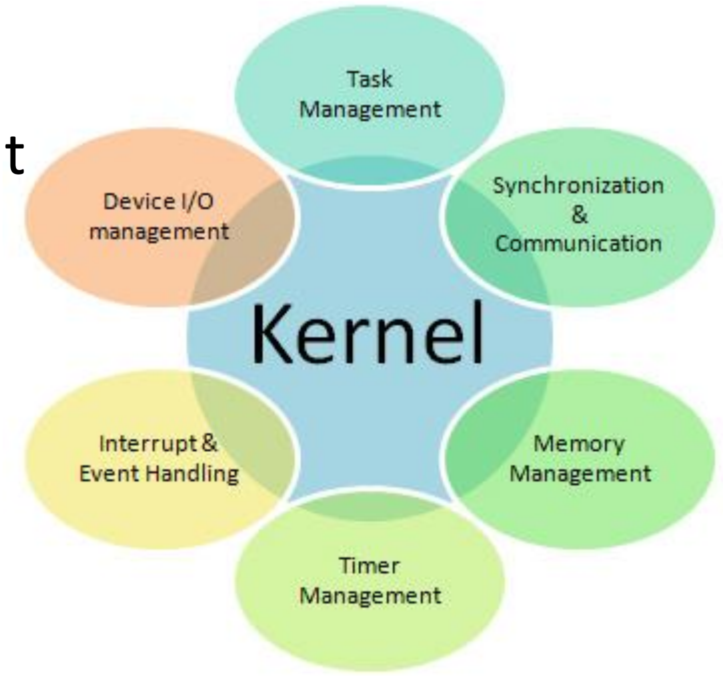
Business Layer
Business Models
System Management
HMI
Application Layer
Graphic Data Representation
Smart Applications
Middleware Layer
Decision Unit
Data Analytics
Process Information
http://www
XBee
WiFi
Network Layer
3G
4G
5G
Network Technologies
Data Transmission
Perception Layer
RFID
Physical Objects
Sensor & Actuators
Data Gathering

## Lecture Topics

- RTOS kernelKey
- Functions of the RTOS Kernel
- What is RTOS task? Scheduler
-

## RTOS kernel

An **RTOS kernel** is the core part of a Real-Time Operating System that manages the hardware resources of the system and facilitates multitasking by managing task scheduling, memory allocation, inter-task communication, and synchronization.
The RTOS kernel is responsible for ensuring tasks are executed within their defined time constraints (real-time performance). It typically provides the following functions:



# Key Functions of the RTOS Kernel:

**1. Task Scheduling**:

1. It schedules tasks based on priorities (preemptive or non-preemptive) and time-slices, ensuring critical tasks get executed within deadlines.

**2. Inter-Task Communication (IPC)**:

1. The kernel supports various methods like queues, semaphores, and mailboxes to allow tasks to communicate with each other in a synchronized way.

**3. Memory Management**:

1. It manages memory allocation for tasks and system resources, ensuring each task gets its required memory without causing conflicts.

**4. Synchronization**:

1. The kernel synchronizes tasks by providing mechanisms like mutexes and semaphores, ensuring data consistency when multiple tasks access shared resources.

**5. Interrupt Management**:

1. The kernel handles hardware interrupts and ensures the system responds to events in real-time, delegating execution of critical tasks based on interrupt priority.

**6. Time Management**:

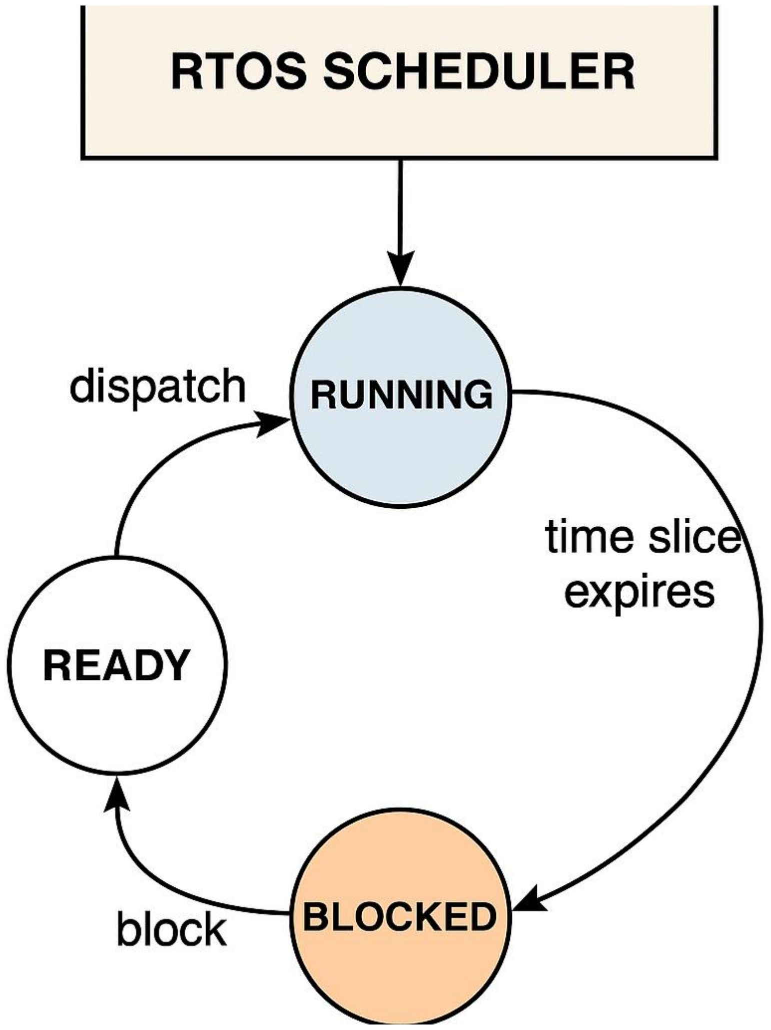
1. The kernel tracks system time and manages the execution of time-sensitive tasks, ensuring they run at specific intervals or within deadlines.

# Types of RTOS Kernels:

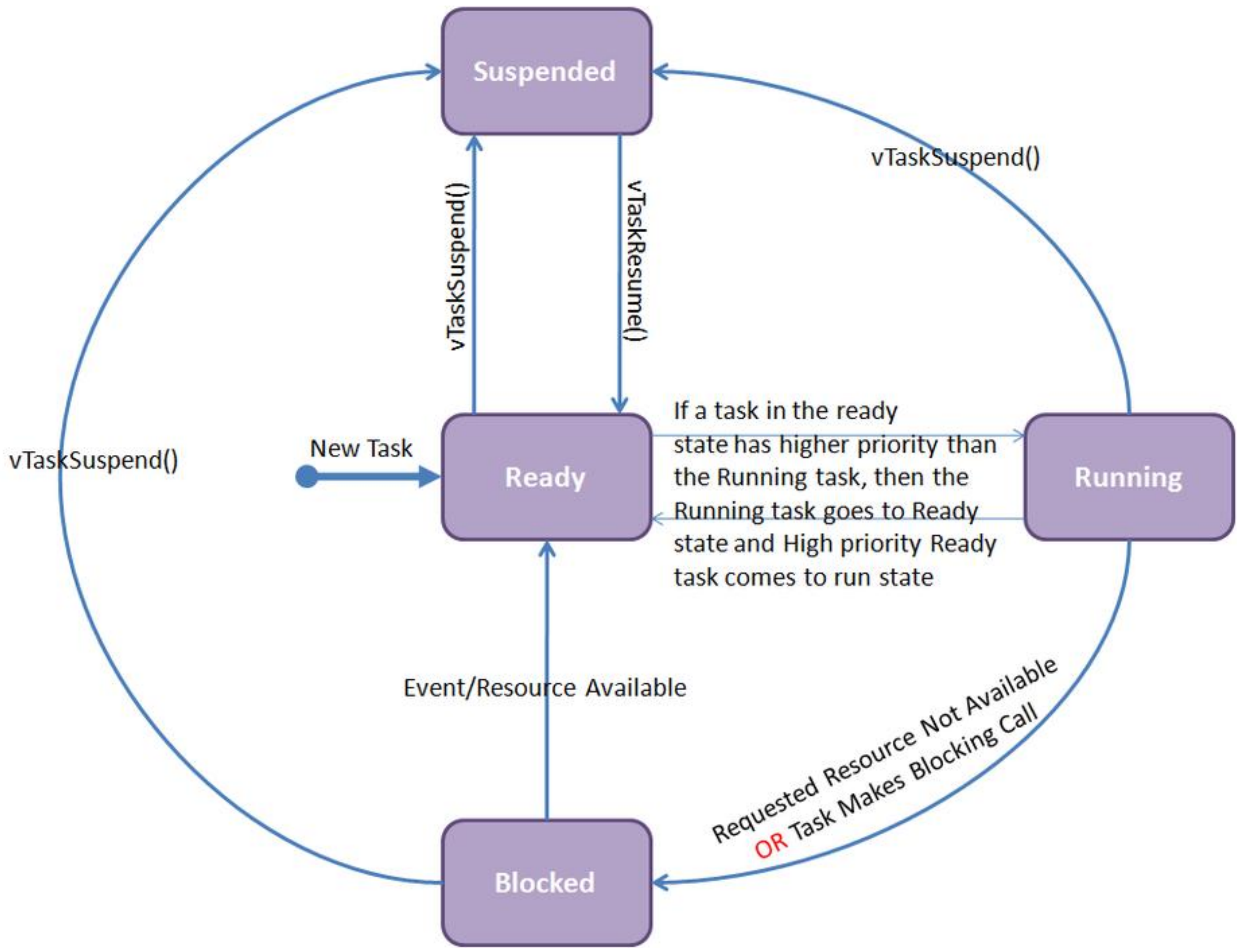
1. **Monolithic Kernel**:
   1. In a monolithic kernel, the entire kernel works in a single address space, offering fast communication and system calls, but potentially less stability and security if a bug occurs.
2. **Microkernel**:
   1. A microkernel keeps minimal services running in the kernel and moves other services (e.g., device drivers) into user space. It offers better stability and modularity but may incur some overhead due to the inter-process communication between user space and kernel space.
3. **Hybrid Kernel**:
   1. A hybrid kernel combines aspects of both monolithic and microkernels, trying to balance speed and modularity.

# What is RTOS task?

- An **RTOS task** refers to a **unit of work** that the RTOS schedules for execution. It is essentially a program or a thread that is executed within the operating system in response to a specific event or condition. In a Real-Time Operating System (RTOS), tasks are typically designed to perform specific, time-sensitive functions.



# Task Status


Suspended
vTaskSuspend()
vTaskSuspend()
vTaskResume()
vTaskSuspend()
New Task
Ready
Running
If a task in the ready state has higher priority than the Running task, then the Running task goes to Ready state and High priority Ready task comes to run state
Event/Resource Available
Requested Resource Not Available OR Task Makes Blocking Call
Blocked

## Key Characteristics of RTOS Tasks:

1. **Task States:**
   - Tasks in an RTOS can have different states such as:
     - **Ready**: The task is ready to be executed but waiting for CPU time.
     - **Running**: The task is currently being executed.
     - **Blocked/Waiting**: The task is waiting for an event (e.g., data from a sensor, or completion of another task).
     - **Suspended**: The task is stopped temporarily, often by the kernel or by the task itself.
2. **Task Priority**:
   - RTOS tasks can have **priorities** to determine which tasks are executed first. Higher-priority tasks can preempt lower-priority ones.
   - Tasks are scheduled based on these priorities and can be **preemptive** (task with the highest priority can interrupt lower-priority tasks) or **non-preemptive** (tasks continue to run until they yield the CPU).
3. **Multitasking**:
   - RTOS tasks enable **multitasking** where multiple tasks are running concurrently. The RTOS kernel manages the switching between tasks using scheduling algorithms (such as round-robin, priority-based, etc.).

# Key Characteristics of RTOS Tasks:

4. **Task Creation and Management**:
    - RTOS tasks are typically created using specific API calls like xTaskCreate() in FreeRTOS. These tasks are given names, priorities, stack sizes, and functions they will execute.
    - Tasks may also be suspended or resumed programmatically using functions like vTaskSuspend() or vTaskResume().
5. Synchronization and Communication:
    - Tasks often need to synchronize with each other to ensure data integrity, particularly when multiple tasks share resources. RTOS provides synchronization mechanisms like semaphores, mutexes, and **message queues** to achieve this.
6. **Time Constraints**:
    - RTOS tasks are often **time-constrained** to meet real-time deadlines. These constraints could involve processing data from sensors at regular intervals, or executing a task within a certain period to maintain system stability.

## Key RTOS Task Properties

**1. Task Priority**

•Determines the order in which tasks are executed.

•Higher-priority tasks **preempt** lower-priority ones (in preemptive scheduling).

•Example: In FreeRTOS, priority ranges from 0 (lowest) to configMAX_PRIORITIES-1 (highest).

**2. Task State**

•A task can be in one of these states:

- •**Ready** – Waiting to be executed (eligible for CPU time).
- •**Running** – Currently executing on the CPU.
- •**Blocked/Waiting** – Waiting for an event (e.g., delay, semaphore, message).
- •**Suspended** – Manually paused (not scheduled until resumed).

## Key RTOS Task Properties (cont.)

**3. Task Stack Size**

•Each task has its own **stack memory** for local variables and function calls.

•Must be carefully allocated to avoid **stack overflow**.

•Example: xTaskCreate(taskFunc, "Task1", 256, NULL, 2, &handle) → 256 bytes stack.

**4. Task Entry Function**

•The function where the task begins execution.

•Typically runs in an infinite loop (while(1)).

•Example:

```
void vTaskFunction(void *pvParameters) {
        while(1)
                { // Task code here
                 }
}
```

## Key RTOS Task Properties (cont.)

- **5. Task Control Block (TCB)**
- A data structure maintained by the RTOS to store task information:
  - Current state.
  - Priority.
  - Stack pointer.
  - Delays or event waits.
- **6. Task Scheduling Policy**
- **Preemptive Scheduling** – Higher-priority tasks immediately take CPU.
- **Time-Sliced (Round Robin)** – Tasks of equal priority share CPU time.
- **Cooperative Scheduling** – Tasks yield CPU voluntarily.

# Key RTOS Task Properties (cont.)

**9. Task Notification & Synchronization**

- Tasks can communicate via:
  - **Semaphores** (xSemaphoreTake/Give).
  - **Mutexes** (for resource locking).
  - **Message Queues** (xQueueSend/Receive).
  - **Event Groups** (xEventGroupSet/Wait).

**10. Task Deletion & Cleanup**

- Tasks can be dynamically deleted (vTaskDelete()).
- Some RTOS require manual cleanup (e.g., freeing memory).

# Key RTOS Task Properties (cont.)

- **7. Task Parameters**
- Allows passing data to a task during creation.
- Example:

```
xTaskCreate(taskFunc, "Task1", 128, (void*)&config, 1, NULL);
```

## 8. Task Delay & Timeouts

- Tasks can **sleep** (vTaskDelay()) or wait for events with timeouts.
- Example:

```
vTaskDelay(100); // Delay for 100 RTOS ticks
 xQueueReceive(queue, &data, pdMS_TO_TICKS(200)); // Wait max 200ms
```

## Example Task Properties in FreeRTOS

```
void vSensorTask(void *pvParameters) {
   uint32_t sensor_id = *(uint32_t*)pvParameters;
   while(1) {
      // Read sensor data
      vTaskDelay(pdMS_TO_TICKS(100)); // 100ms delay
   }
}
void main() {
   uint32_t sensor_id = 5;
   xTaskCreate(
      vSensorTask,       // Task function
      "Sensor Task",     // Task name
      256,               // Stack size (bytes)
      &sensor_id,        // Task parameter
      3,                 // Priority (higher = more urgent)
      NULL               // Task handle (optional)
   );
   vTaskStartScheduler();  // Start RTOS
}
```

# Summary Table of RTOS Task Properties

| Property | Description |
|---|---|
| **Priority** | Determines execution order (higher = runs first) |
| **State** | Ready, Running, Blocked, Suspended |
| **Stack Size** | Memory reserved for task execution |
| **Entry Function** | The function where the task starts |
| **TCB** | OS-managed metadata for task control |
| **Scheduling** | Preemptive, Round Robin, or Cooperative |
| **Parameters** | Input data passed at task creation |
| **Delay/Timeout** | Pausing or waiting for events |
| **Sync Mechanisms** | Semaphores, Mutexes, Queues, Events |
| **Deletion** | Dynamically removing a task |