SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT312 – EMBEDDED SYSTEM DESIGN

III YEAR/ VI SEMESTER

1

UNIT 4 :**Embedded Operating System and Modelling**

TOPIC 4.9 : Mutexes

Introduction to Mutexes

- Mutex stands for Mutual Exclusion. It's a synchronization primitive used to control access to shared resources in concurrent programming
- In embedded systems, where multiple tasks or threads execute concurrently, mutexes play a vital role in ensuring data integrity and preventing race conditions

Basic Concepts - Mutexes

- **Critical Section**: A section of code that accesses shared resources and must be protected from simultaneous access by multiple tasks

- **Race Condition**: A situation where the outcome of a program depends on the timing of uncontrollable events. It occurs when multiple tasks access shared resources concurrently without proper synchronization

- **Mutex**: A mutual exclusion object used to protect critical sections. It allows only one task at a time to access the protected resource.

Mutexes- Types

- **Binary Mutex**: Also known as a semaphore with count 1. It has only two states: locked and unlocked. Typically used to protect resources that can be accessed by only one task at a time

- **Counting Mutex**: A mutex that can be locked multiple times by the same task. It maintains a count of the number of times it has been locked and must be unlocked the same number of times before other tasks can acquire it.

Implementation of Mutexes

- **Spinlocks**: A simple form of mutex where a task repeatedly polls the mutex until it becomes available. Suitable for use in systems where task preemption is disabled or where the critical section is expected to be short.

- **Blocking Mutexes**: Tasks attempting to acquire a mutex block until it becomes available. This can be implemented using task suspension or context switching mechanisms.

- **Priority Inheritance**: A technique used to prevent priority inversion issues in real-time systems. When a lower-priority task holds a mutex required by a higher-priority task, the lower-priority task temporarily inherits the priority of the higher-priority task until it releases the mutex.

Thank you