4.5 Prevention and Handling Strategies

1. **Overcoming Deadlock**
   Since detecting and recovering from a deadlock in an embedded system is very difficult, the best approach is prevention. This is done by breaking one of the four Coffman conditions.
   - **Break Circular Wait**: Enforce a strict lock ordering for all resources. If all tasks are required to lock Mutex 1 before Mutex 2, the deadlock scenario described above could never happen. This is the most common and effective deadlock prevention strategy
   - **Break Hold and Wait**: Request all required resources at once.
   - **Detection and Recovery**: Mechanisms like a wait-for graph can be used to detect cycles. If a deadlock is detected, the system can recover by aborting one of the tasks (termination) or forcibly taking a resource (preemption). However, this is rarely implemented on microcontrollers.

2. **Overcoming Priority Inversion and Starvation**
   - **Priority Inheritance**: This is the solution to priority inversion. If a high-priority task is blocked waiting for a mutex held by a low-priority task, the low-priority task will temporarily "inherit" the priority of the high-priority task. This allows the low-priority task to run quickly, finish its work, and release the mutex as soon as possible.
     Note: Mutexes in FreeRTOS already implement this mechanism automatically.
   - **Priority Ceiling**: Each resource is assigned a priority ceiling, which is the highest priority level of any task that can access it. When a task locks the resource, its priority is immediately raised to that ceiling level until it releases it.
   - **Aging**: To prevent starvation, the priority of a task that has been waiting for a long time can be gradually increased. This way, a low-priority task will eventually have a high enough priority to be executed.
   - **Fair Scheduling**: Using a scheduling algorithm (like round-robin for tasks of the same priority) to ensure all tasks get a fair share of CPU time.

Revision #1
Created 2025-08-28 16:21:34 UTC by NS
Updated 2025-08-28 16:26:05 UTC by NS