7.任务同步

2025年10月14日 23:46

一、什么是任务同步

- 前面我们学习了很多种通信机制，如**队列、信号量、事件组，任务通知**等等。我们发现他们都**有一个相同的作用**就是用于**任务同步**，那么什么是任务同步呢
- 在FreeRTOS中，**任务同步是指多个任务之间通过特定的机制，来调节执行节奏，确定任务按照预期的顺序执行或者协调任务之间工作的过程。**
- **任务同步**的**目的**是解决多任务并发时 **资源竞争、执行顺序依赖、事件通知**等问题，**避免因任务执行顺序的错乱导致数据错误、逻辑异常或者系统崩溃**

二、为什么需要任务同步

- 多任务系统中，**任务的调度由系统内核（任务的优先级和时间片）确定**，执行顺序不是确定的，接下来我将举几个具体的例子来说明一下**任务同步的必要性**
  - 开始时数据缓冲区为空，任务A要向缓冲区中写入数据，任务B要从缓冲区中获取数据，如果任务B在任务A之前执行，会得到无用的数据
  - 任务1要检测传感器数据，任务2要处理传感器数据，任务2必须在任务1之后执行
  - 多任务同时操作一个硬件时（如串口，GPIO），可能导致操作冲突（比如同时通过串口发送数据导致数据错乱），这时就需要同步机制让任务“有秩序”地执行，这就是任务同步的意义

三、任务同步的三个核心目标

- **1.确保执行顺序**：让任务**按逻辑依赖关系执行**（如先存数据，再读数据）
- **2.确保事件通知**：一个任务的**状态变化（如“数据存完”“按键按下”）能准确通知到其他任务**
- **3.确保资源安全（间接）**：虽然同步不直接保护资源，但**合理的同步能保证资源在未保护好的情况下被访问，间接减少资源竞争风险**

四、任务同步的核心机制

- FreeRTOS中提供了多种任务同步的机制，每种机制都适用于不同的场景，核心是**“等待--唤醒”逻辑协调任务**：**当条件不满足时，任务进入阻塞状态（释放CPU），当条件满足时，任务被唤醒（重新参与调度）**

1、信号量（Semaphore）

- 信号量时最常用的同步工具，**本质是一个计数器**，通过**对计数器的“获取”和“释放”实现同步**
  - **核心机制**
    - xSemaphoreTake（）：**获取信号量（计数值-1）；如果计数值为0，那么任务进入阻塞等待状态**，直到有任务释放信号量了，任务才会回到就绪态继续执行
    - xSemaphoreGive（）：**释放信号量（计数值+1）；如果有任务在等待，会唤醒其中一个任务（通常是优先级最高的）**
  - **分类及应用场景**
    - **二值信号量（Binary xSemaphore）**：计数值**只能是0和1**，适用于“事件通知”或“互斥”（互斥操作更推荐互斥锁）
      - **同步逻辑**
        - 1、**等待任务先获取信号量**：若信号量为0，则进入阻塞
        - 2、**通知任务完成操作后释放信号量**：将信号量重置为1
        - 3、信**号量被释放后，阻塞等待任务会被唤醒**，同时“获取令牌”，继续执行
      - **适用场景**
        - 单一任务通知另一任务（一对一）
    - **计数信号量（Counting xSemaphore）**：计数值可以是**0到最大值**；适用于“资源池管理”（如有限数据缓冲区，连接数）

- **同步逻辑**
  - 1、**初始化设置计数最大值**
  - 2、**通知任务释放信号时，计数值+1**
  - 3、**等待任务获取信号量，计数值-1，若计数为0，任务阻塞**
- **适用场景**
  - 1、**多任务通知同一任务**（如3个传感器任务，任一任务完成后通知数据处理任务）
  - 2、**一个任务通知多个任务**
  - 3、**资源管理池**（对资源数进行控制，每一个获取资源时，信号量-1）

2、互斥锁（Mutex）

互斥锁是一个**特殊的二值信号量**，专门用于保护共享资源（避免多任务同时访问），解决“临界区”问题

- **与二值信号量的区别**
  - **互斥锁支持优先级继承**，当低优先级任务获得互斥锁时，此时如果**有高优先级的任务去获取互斥锁**，此时**互斥锁就会将低优先级任务的优先级临时提升到与高优先级任务的优先级相同，避免了此时被中优先级任务抢占从而发生优先级反转问题**（高优先级任务因中优先级任务的发生而阻塞）
- **核心操作**
  - xSemaphoreTake（）：**获取互斥量**（同信号量，计时器从1->0）
  - xSemaphoreGive（）：**释放互斥量**（计数器从0->1）

3、事件组（Event Group）

事件标志组用于多任务同步，允许任务等待事件中的一个或多个发生

- **本质**
  - 一个**32位无符号整数（EventBits_t）**，每一位代表一个事件（Bit0~Bit31）
- **核心操作**
  - xEventGroupWaitBits（）：等待**指定事件标志位被置位**，可以选者全部都满足唤醒或者是任一事件执行唤醒，并可以设置超时时间
  - xEventGroupSetBits（）：设置**指定标志位为1**（触发事件），唤醒等待该事件的任务
  - xEventGroupClearBits（）：**清除指定的标志位**
- **同步逻辑**
  - 1、**等待任务执行要等待的掩码**（如0x31，bit0h和bit1）和**等待方式**（AND=两个事件都满足，OR=任一一个事件满足）
  - 2、**其他任务或中断设置标志位**
  - 3、当**事件标志位满足等待条件**，就**唤醒所有满足条件的任务**，可选择**“自动清除事件位”**和**“手动清除”**
- **适用场景**
  - **单个任务需要等待多个任务**，如电机启动需要满足按下按键+电压检测正常

4、消息队列（Queue）

上面我们说了信号量可以用于任务同步，我们知道信号量是特殊的队列，所以消息队列不仅可以用于任务通信，还可以用于任务同步，通过”**发送消息--等待消息**“的逻辑协调任务执行

- **核心逻辑**
  - 任务A向消息队列发送消息，任务B从消息队列中接收消息。如果队列为空，那么任务B会进入阻塞等待；如果队列已满，那么任务A将会进入阻塞
- **同步场景**
  - 任务A生成数据后通过队列发送给任务B，只有任务B接收到数据后才能处理，天然实现了**“生产--消费”**的同步

5.任务通知（Task Notification）

FreeRTOS中的**轻量级通信同步机制**，通过**直接向任务发送通知实现同步，效率高于信号量**（无需额外分配内存）

- **核心机制**
  - 每一个任务都有一个**32位的通知值（ulNotifiedValue）**和**通知状态**，支持多种通知方式：
    - **发送通知并唤醒任务（覆盖/递增/设置Bits等）**

- 任务通过ulTaskNotifyTake发送通知（）和xTaskNotifyWait（）等待通知，等待某位，实现事件组的操作
- 任务可以通过ulTaskNotifyTake（）发送通知和xTaskNotifyGive（）等待通知，实现信号量的操作

- **同步逻辑**
  - 1、**通知任务**调用xTaskNotify（）或xTaskNotifyGive（），向目标任务**发送“通知值”或“通知计数”**
  - 2、**目标任务**调用xTaskNotifyWait（）或ulTaskNotifyTake（），等待通知，若未收到通知就阻塞
- **适用场景**
  - 替代了**简单的二值信号量或计数信号量（一对一同步）**，尤其适合资源受限的MCU，能减少内存占用
- **局限性**
  - 仅支持**一对一**（一个任务通知另一个任务），不支持“多对一”或“多事件组合等待”

五、任务同步VS任务互斥

维度	任务同步（Synchronization）	任务互斥（Mutual Exclusion）
核心目标	解决“任务何时执行”的问题（协调执行顺序）	解决“任务用什么资源”的问题（保护共享资源）
核心场景	生产者 - 消费者（先存后读）、多事件等待	多个任务访问同一个串口 / 变量 / 外设
关键机制	信号量、事件组、任务通知	互斥锁（Mutex）、临界区（Critical Section）
典型问题	任务执行顺序错误（读空缓冲区）	资源竞争（变量值被覆盖、串口数据错乱）

- **例子**
  - 用**二值信号量**实现**生产者存放数据**后通知**消费者读取数据**--这是**任务同步**
  - 用**互斥锁**确保**生产者和消费者不会同时操作数据缓冲区**--这是**任务互斥**
  - 实现项目中，二者往往结合使用

六、任务同步的关键原则

- **1、最小阻塞时间**：任务等待同步条件时，应**合理的设置任务阻塞时间**，且**持有同步对象的时间应尽可能短**
- **2、避免死锁**：例如，任务1持有互斥锁1等待互斥锁2，任务2持有互斥锁2的等待互斥锁1，此时就会导致任务锁死，**注意锁的获取顺序**
- **3、优先级合理**：**高优先级任务不应该长时间等待低优先级任务**，必要时可以使用互斥锁的优先级继承
- **4、选择合适的机制**：**简单使用二值信号量，多事件用事件标志组，共享资源用互斥锁，轻量场景用任务通知**

七、总结

- 任务同步是FreeRTOS中多**任务协作的核心**，通过信号量，事件组，消息队列，任务通知等机制，解决了**任务间执行顺序、资源竞争、事件依赖等问题**。所以任务同步功能是FreeRTOS中必不可少的的功能。