阿熊的FreeRTOS教程系列！

哈喽大家好！我是你们的老朋友阿熊！STM32教程系列更新完结已经有一段时间了，视频反馈还是不错的，从今天开始我们将会更新我们的FreeRTOS的教程

由于东西真的太多了，也纠结了很久要不要讲这个系列，毕竟难度真的很大，怕在难以做到那么通俗易懂，经过一段时间的考虑，还是决定好了给大家做一个入门级的讲解使用，由于FreeRTOS的内容真的很多，作为还是学生的我使用的也相对较少，操作系统层面的东西，我会用最大的能力去让大家理解，主要讲述主要功能，学完以后保证大伙可以理解80%以上的FreeRTOS的使用场景，好了废话不多说，开始我们的课程吧！

第十二章：中断讲解

如果你已经将前面11章的内容全部吸收掌握，那么恭喜你，你已经基本上会使用FreeRTOS的绝大多数功能了，然后本章内容是为了给大家填一下前面的坑，以及给大家补充一些关于中断方面的内容

填坑：

前面我们给大家讲过很多中断的函数，但是没有带着大家去使用，甚至我连那个中断中切换任务的函数也是没有给大家讲的，然后现在给大家填坑

任务切换函数：

portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);

portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

这两个都可以完成相关的任务切换，传入的参数应该不陌生了，就是任务是否需要切换，然后两个函数最大的区别是，前者是使用汇编进行编写的，然后后者是C语言版本

好了，函数已经介绍给大家了，然后坑已经给大家填上了，然后我们现在正常开始讲课

壹：中断的优先级

任何中断的优先级都大于任务

在我们的操作系统，中断同样是具有优先级的，并且我们也可以设置它的优先级，但是他的优先级并不是从0~15，默认情况下它是从5~15，0~4这5个中断优先级不是FreeRTOS控制的（5是取决于configMAX_SYSCALL_INTERRUPT_PRIORITY），因为我们的那个操作系统的并不是万能的，所以我们通常情况下需要弄一些中断凌驾在我们的操作系统之上，并且任何中断都会大于任务，可以打断任务，哪怕是最高优先级的任务

中断中必须使用相关中断函数

我们的操作系统中很多的函数它都是有两套的，一套是在正常任务中使用，一套是在中断中使用，两套函数的功能几乎是一样的，但是他们必须在自己合适的位置使用

中断运行时间越少越好

由于我们的中断会凌驾于所有的任务之上，如果它的运行时间过长，哪怕是我们的定时器，都会不再准确，所以我们的中断中一般是不会进行很复杂的操作，大多时候都是改变一些标志位

判断任务是否需要切换

我们所有关于中断的函数，它都有一个参数，也就是判断任务是否需要切换，然后前面也给大家提到过很多次，但是一直也没有教过大家如何去使用，然后这里就给大家放一段示例代码

```
void XXX_ISR()
{
    int i;
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    for (i = 0; i < N; i++)
    {
        xQueueSendToBackFromISR(..., &xHigherPriorityTaskWoken); /*
被多次调用 */
    }
    /* 最后再决定是否进行任务切换 */
    if (xHigherPriorityTaskWoken == pdTRUE)
    {
        /* 任务切换 */
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
    }
}
```

一般不进行中断的任务切换

多数情况下，我们是不会直接进行任务切换的，而是进行一些相关标志位的改变，直接切换的话就可能会有很多，不可控的因素导致我们的任务出现一些小问题

贰：中断屏蔽（临界区，保护区）

我们前面说过，操作系统中的中断是大于任何的任务，可是我们的中断又不确定什么时候会发生这样的话，就可能会导致我们的中断，将我们的任务进行打断，虽然他绝大多数时候都会跳转回来，但是这样就有可能会影响到他的时间精确性啊，或者一些其他的意外，所以我们的操作系统中他就提供了屏蔽中断的一些函数供我们使用

任务中的中断屏蔽函数：

屏蔽中断：

```
taskENTER_CRITICAL();
//优先级低于、等于configMAX_SYSCALL_INTERRUPT_PRIORITY的中断将会被屏蔽
//优先级高于configMAX_SYSCALL_INTERRUPT_PRIORITY的中断不受影响
```

恢复中断：

```
taskEXIT_CRITICAL();
//恢复中断使能
```

中断中的中断屏蔽函数：

同样的在我们的，相对较低极的中断中，他也可能会被更高级的中断所打断，这样的话就可能导致我们的中断反应不及时，所以我们就需要使用中断中的屏蔽函数，将我们更高级的中断给屏蔽，防止干扰我们低级的中断

屏蔽中断：

```
UBaseType_t taskENTER_CRITICAL_FROM_ISR();
//返回当前状态
```

恢复中断：

```
taskEXIT_CRITICAL_FROM_ISR(UBaseType_t  uxSavedInterruptStatus );
//传入屏蔽时候的状态
```

叁：中断的相关使用示例

实验：任务中屏蔽中断

设置两个中断，中断1中断优先级，在0~4之间，中断2的优先级在5~15之间,按下按键屏蔽中断后观察现象

现象：

按下按键以后中断2停止运行，中断1正常运行

分析：

处于0-4优先级的中断凌驾于操作系统之上

完结！撒花！