第10章　FreeRTOS 任务管理

对于初学者，特别是对于没有 RTOS 基础的同学来说，了解 FreeRTOS 的任务管理非常重要，了解任务管理的目的就是让初学者从裸机的，单任务编程过渡到带 OS 的，多任务编程上来。搞清楚了这一点，那么 FreeRTOS 学习就算入门了。

本章教程配套的例子含Cortex-M3内核的STM32F103和Cortex-M4内核的STM32F407以及F429。

10.1 单任务系统
10.2 多任务系统
10.3 FreeRTOS 的任务设置
10.4 FreeRTOS 的系统任务栈设置
10.5 FreeRTOS 的任务状态
10.6 FreeRTOS 启动
10.7 FreeRTOS 的任务创建
10.8 FreeRTOS 的任务删除
10.9 FreeRTOS 的任务挂起
10.10 FreeRTOS 的任务恢复
10.11 FreeRTOS 的任务恢复（中断方式）
10.12 FreeRTOS 的空闲任务
10.13 实验例程说明
10.14 实验例程说明（中断方式）
10.15 总结

10.1　单任务系统

学习多任务系统之前，我们先来回顾下单任务系统的编程框架，即裸机时的编程框架。裸机编程主要是采用超级循环（super-loops）系统，又称前后台系统。应用程序是一个无限的循环，循环中调用相应的函数完成相应的操作，这部分可以看做后台行为；中断服务程序处理异步事件，这部分可以看做是前台行为。后台也可以叫做任务级，前台也叫作中断级。

资料整理
火龙果软件
荐

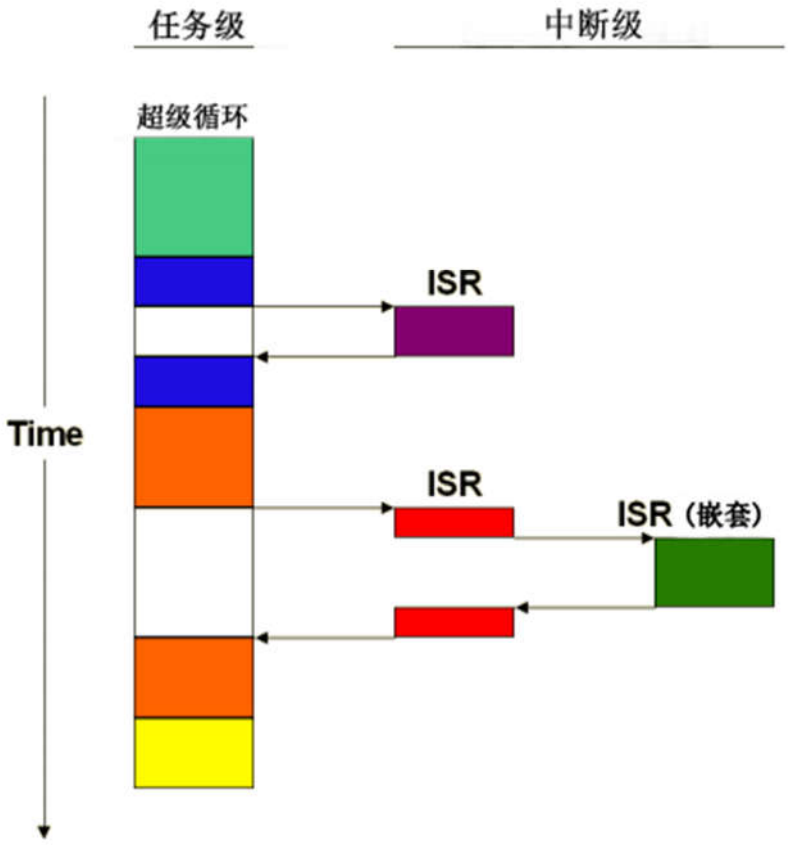


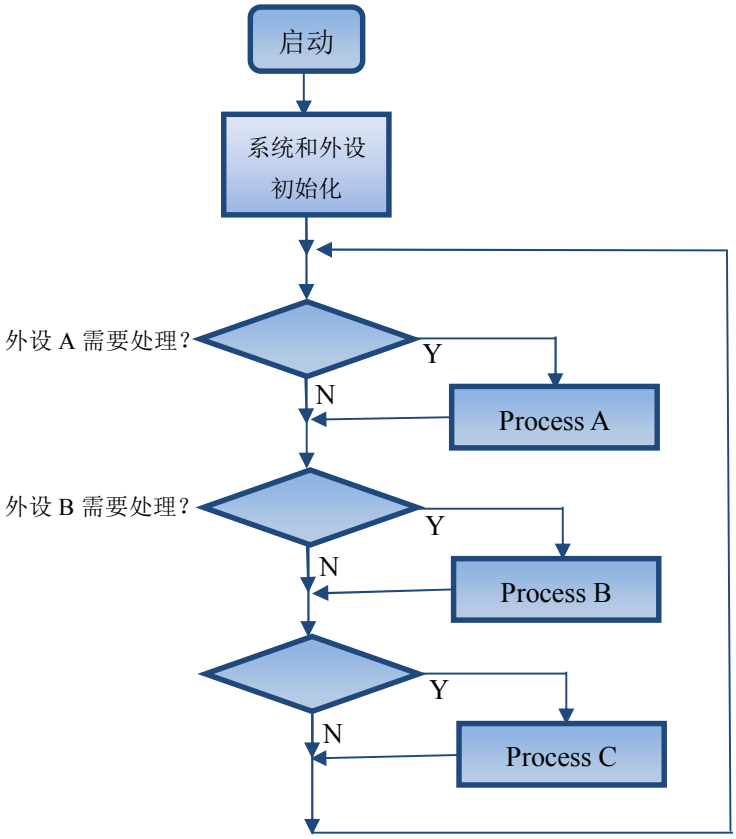
图 10.1 单任务系统

对于前后台系统的编程思路主要有以下两种方式：

10.1.1 查询方式

对于一些简单的应用，处理器可以查询数据或者消息是否就绪，就绪后进行处理，然后再等待，如此循环下去。对于简单的任务，这种方式简单易处理。但大多数情况下，需要处理多个接口数据或者消息，那就需要多次处理，如下面的流程图所示：

资料整理 火龙果软件 荐

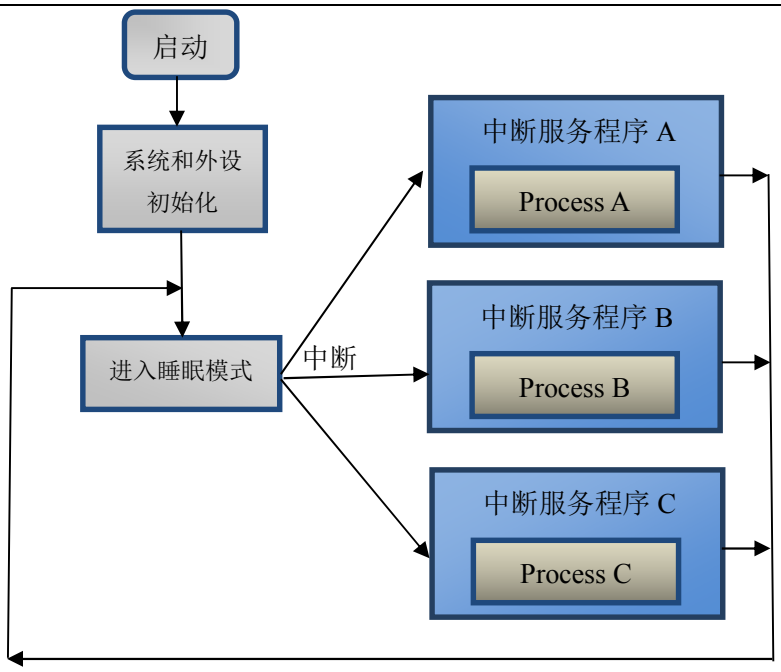


用查询方式处理简单的应用，效果比较好，但是随着工程的复杂，采用查询方式实现的工程就变得很难维护，同时，由于无法定义查询任务的优先级，这种查询方式会使得重要的接口消息得不到及时响应。比如程序一直在等待一个非紧急消息就绪，如果这个消息后面还有一个紧急的消息需要处理，那么就会使得紧急消息长时间得不到执行。

10.1.2 中断方式

对于查询方式无法有效执行紧急任务的情况，采用中断方式就有效地解决了这个问题，下面是中断方式简单的流程图：

资料整理
火龙果软件
荐



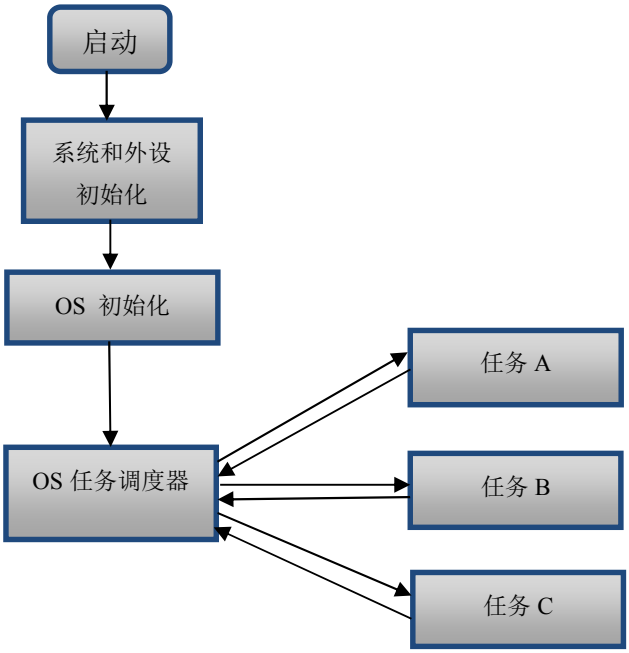
采用中断和查询结合的方式可以解决大部分裸机应用，但随着工程的复杂，裸机方式的缺点就暴露出来了：

- 必须在中断(ISR)内处理时间关键运算：
  - ISR 函数变得非常复杂，并且需要很长执行时间。
  - ISR 嵌套可能产生不可预测的执行时间和堆栈需求。
- 超级循环和 ISR 之间的数据交换是通过全局共享变量进行的：
  - 应用程序的程序员必须确保数据一致性。
- 超级循环可以与系统计时器轻松同步，但：
  - 如果系统需要多种不同的周期时间，则会很难实现。
  - 超过超级循环周期的耗时函数需要做拆分。
  - 增加软件开销，应用程序难以理解。
- 超级循环使得应用程序变得非常复杂，因此难以扩展：
  - 一个简单的更改就可能产生不可预测的副作用，对这种副作用进行分析非常耗时。
  - 超级循环概念的这些缺点可以通过使用实时操作系统（RTOS）来解决。

10.2 多任务系统

针对这些情况，使用多任务系统就可以解决这些问题了。下面是一个多任务系统的流程图：

资料整理 火龙果软件 荐



多任务系统或者说 RTOS 的实现，重点就在这个调度器上，而调度器的作用就是使用相关的调度算法来决定当前需要执行的任务。如上图所示的那样，创建了任务并完成 OS 初始化后，就可以通过调度器来决定任务 A，任务 B 和任务 C 的运行，从而实现多任务系统。另外需要初学者注意的是，这里所说的多任务系统同一时刻只能有一个任务可以运行，只是通过调度器的决策，看起来像所有任务同时运行一样。为了更好的说明这个问题，再举一个详细的运行例子，运行条件如下：

- 使用抢占式调度器。
- 1 个空闲任务，优先级最低。
- 2 个应用任务，一个高优先级和一个低优先级，优先级都比空闲任务优先级高。
- 中断服务程序，含 USB 中断，串口中断和系统滴答定时器中断。

下图 10.2 所示是任务的运行过程，其中横坐标是任务优先级由低到高排列，纵坐标是运行时间，时间刻度有小到大。

资料整理 火龙果软件 荐

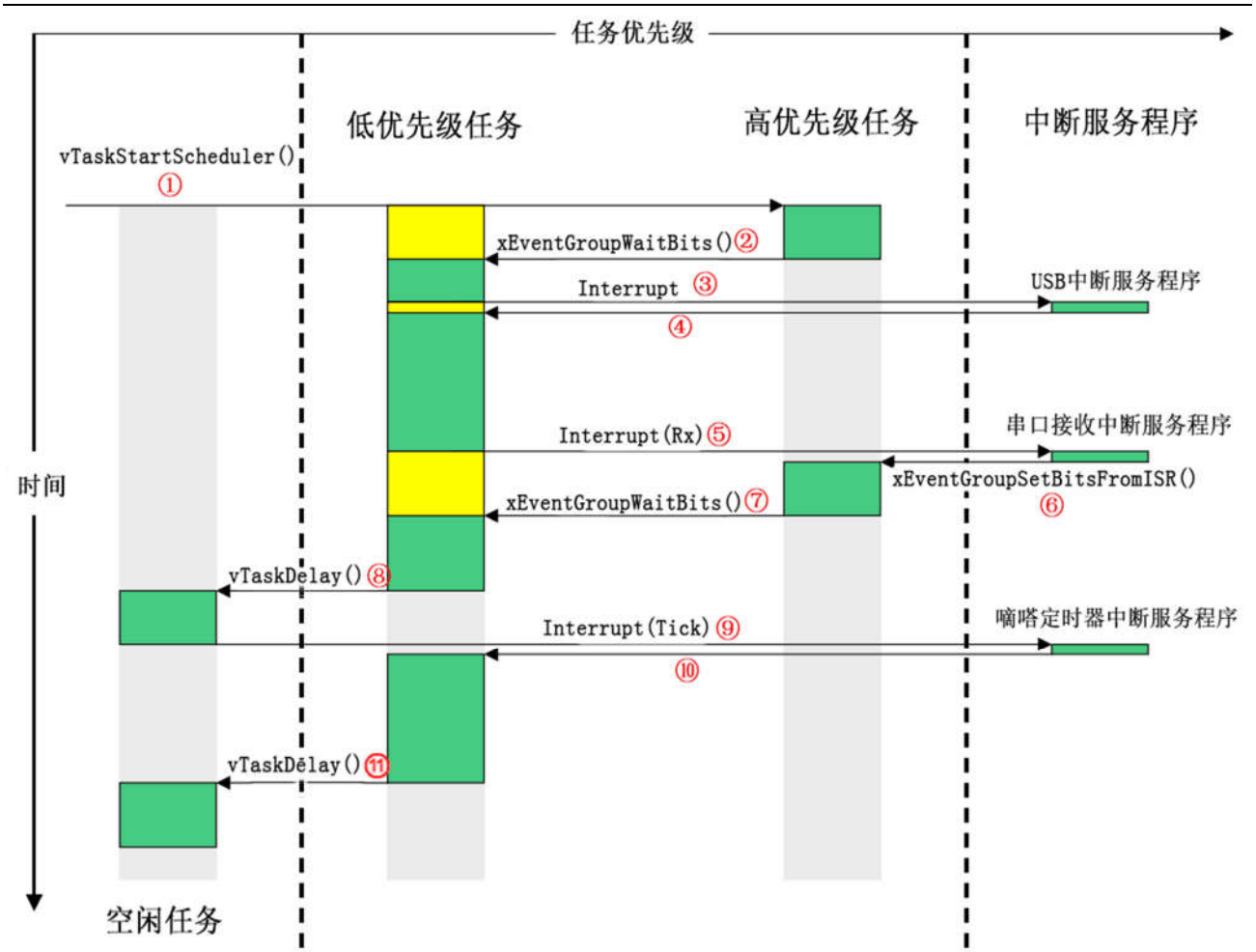


图 10.2 多任务系统运行过程

（1） 启动 RTOS，首先执行高优先级任务(vTaskStartScheduler)。

（2） 高优先级任务等待事件标志（xEventGroupWaitBits）被阻塞，低优先级任务得到执行。

（3） 低优先级任务执行的过程中产生 USB 中断，进入 USB 中断服务程序。

（4） 退出 USB 中断复位程序，回到低优先级任务继续执行。

（5） 低优先级任务执行过程中产生串口接收中断，进入串口接收中断服务程序。

（6） 退出串口接收中断复位程序，并发送事件标志设置消息(xEventGroupSetBitsFromISR)，被阻塞的高优先级任务就会重新进入就绪状态，这个时候高优先级任务和低优先级任务都在就绪态，抢占式调度器就会让高优先级的任务先执行，所以此时就会进入高优先级任务。

（7） 高优先级任务由于等待事件标志（xEventGroupWaitBits）会再次被阻塞，低优先级任务开始继续执行。

（8） 低优先级任务调用函数 vTaskDelay，低优先级任务被挂起，从而空闲任务得到执行。

（9） 空闲任务执行期间发生滴答定时器中断，进入滴答定时器中断服务程序。

（10） 退出滴答定时器中断，由于低优先级任务延时时间到，低优先级任务继续执行。

资料整理 火龙果软件 荐

（11）低优先级任务再次调用延迟函数 vTaskDelay，低优先级任务被挂起，从而切换到空闲任务。空闲任务得到执行。

通过上面实例的讲解，大家应该对多任务系统完整的运行过程有了一个全面的认识。随着教程后面对调度器，任务切换等知识点的讲解，大家会对这个运行过程有更深刻的理解。

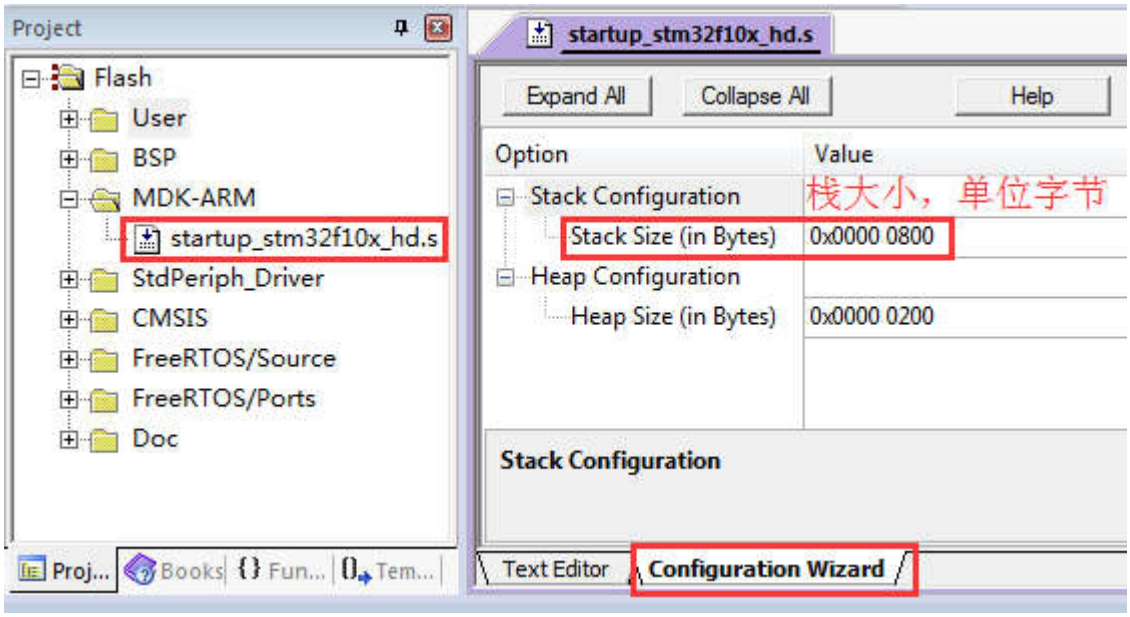
FreeRTOS 就是一款支持多任务运行的实时操作系统，具有时间片，抢占式和合作式三种调度方法。通过 FreeRTOS 实时操作系统可以将程序函数分成独立的任务，并为其提供合理的调度方式。

10.3 FreeRTOS 的任务栈设置

不管是裸机编程还是 RTOS 编程，栈的分配大小都非常重要。局部变量，函数调用时的现场保护和返回地址，函数的形参，进入中断函数前和中断嵌套等都需要栈空间，栈空间定义小了会造成系统崩溃。

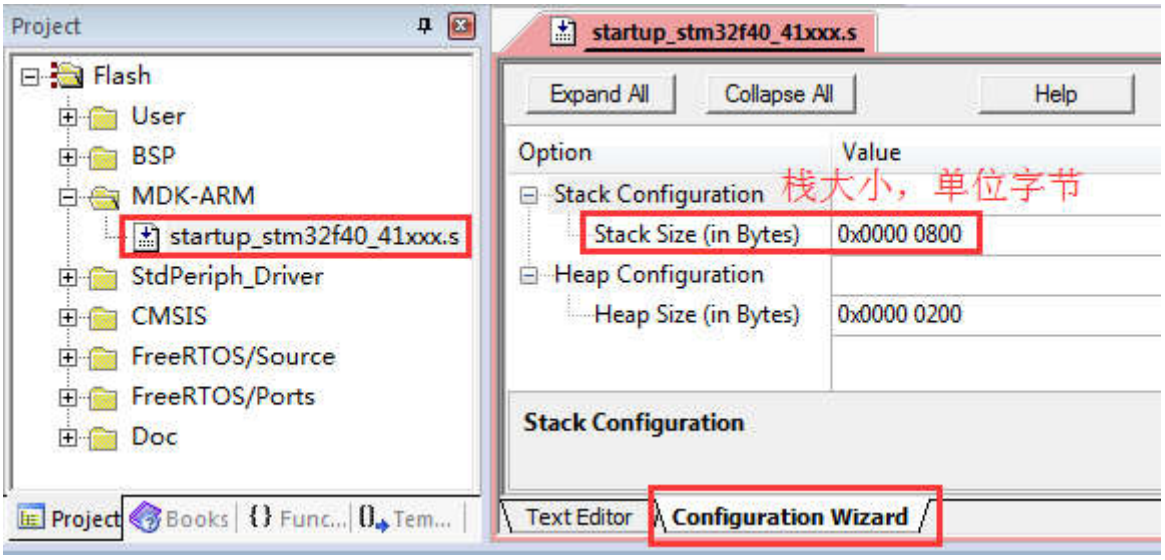
裸机的情况下，用户可以在这里配置栈大小：

◆ STM32F103 工程中栈大小的配置文件

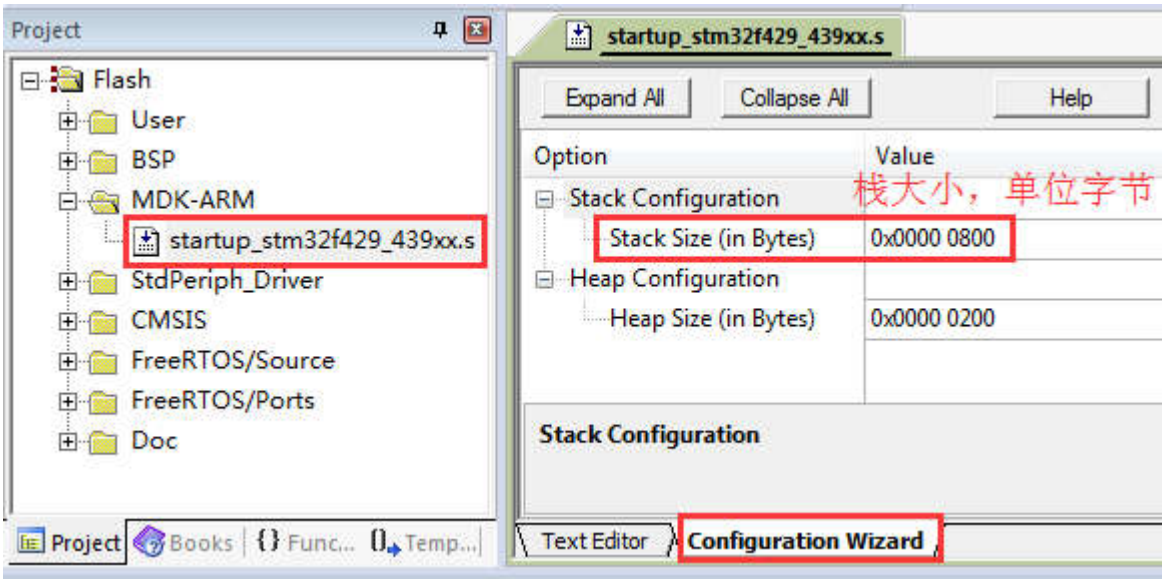


◆ STM32F407 工程中栈大小的配置文件

资料整理 火龙果软件 荐



◆ STM32F429 工程中栈大小的配置文件



不同于裸机编程，在 RTOS 下，每个任务都有自己的栈空间。对于 FreeRTOS 来说，任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE　　((size_t) (17 * 1024))

具体每个任务的栈大小是在创建 FreeRTOS 的任务时进行设置的：

```
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


    xTaskCreate( vTaskLED,            /* 任务函数  */
                 "vTaskLED",          /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
```


资料整理 火龙果软件 荐

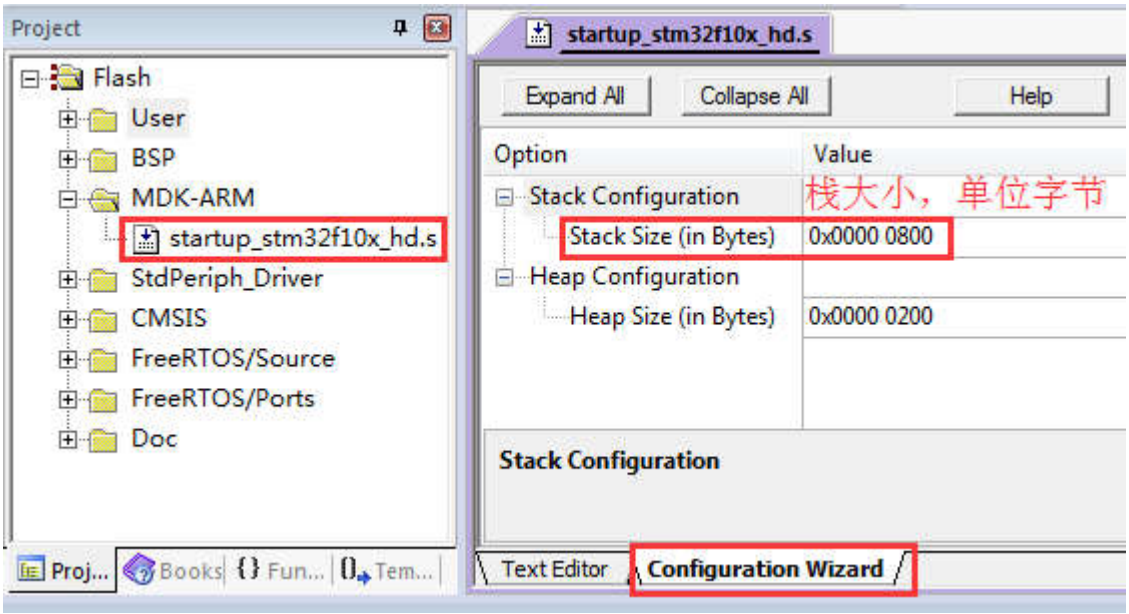
```
                NULL,                /* 任务参数  */
                2,                   /* 任务优先级*/
                &xHandleTaskLED ); /* 任务句柄  */

    xTaskCreate( vTaskMsgPro,            /* 任务函数  */
                 "vTaskMsgPro",          /* 任务名    */
                 512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                   /* 任务参数  */
                 3,                      /* 任务优先级*/
                 &xHandleTaskMsgPro );  /* 任务句柄  */


    xTaskCreate( vTaskStart,             /* 任务函数  */
                 "vTaskStart",           /* 任务名    */
                 512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                   /* 任务参数  */
                 4,                      /* 任务优先级*/
                 &xHandleTaskStart );   /* 任务句柄  */
}
```

实际应用中给任务开辟多大的堆栈空间合适呢，在第 11 章详细跟大家进行讲解。

10.4 FreeRTOS 的系统栈设置

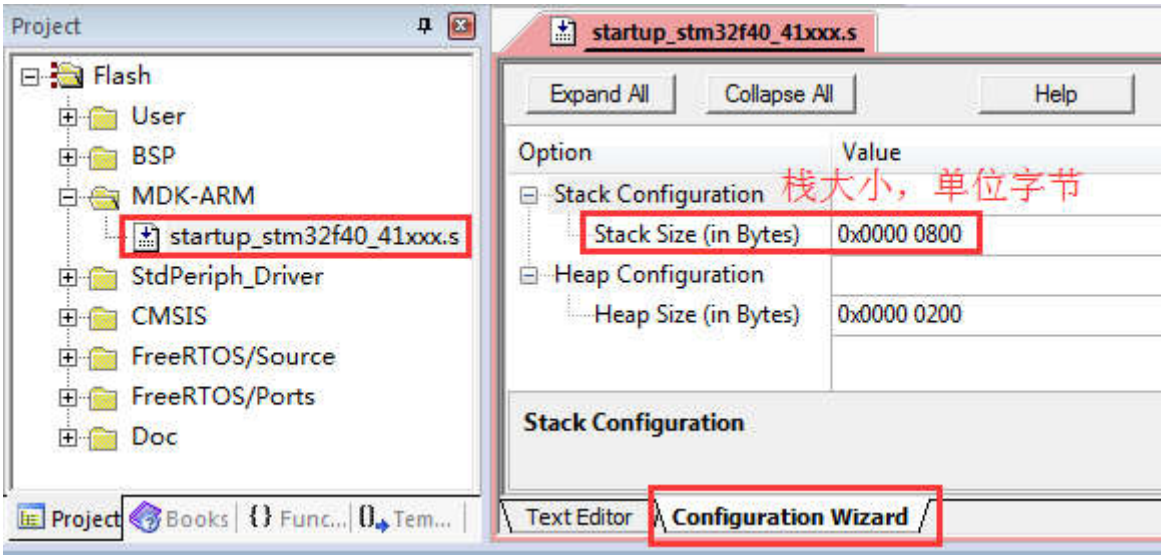
上面跟大家讲解了什么是任务栈，这里的系统栈又是什么呢？裸机的情况下，凡是用到栈空间的地方都是在这里配置的栈空间：

◆ STM32F103 工程中栈大小的配置文件

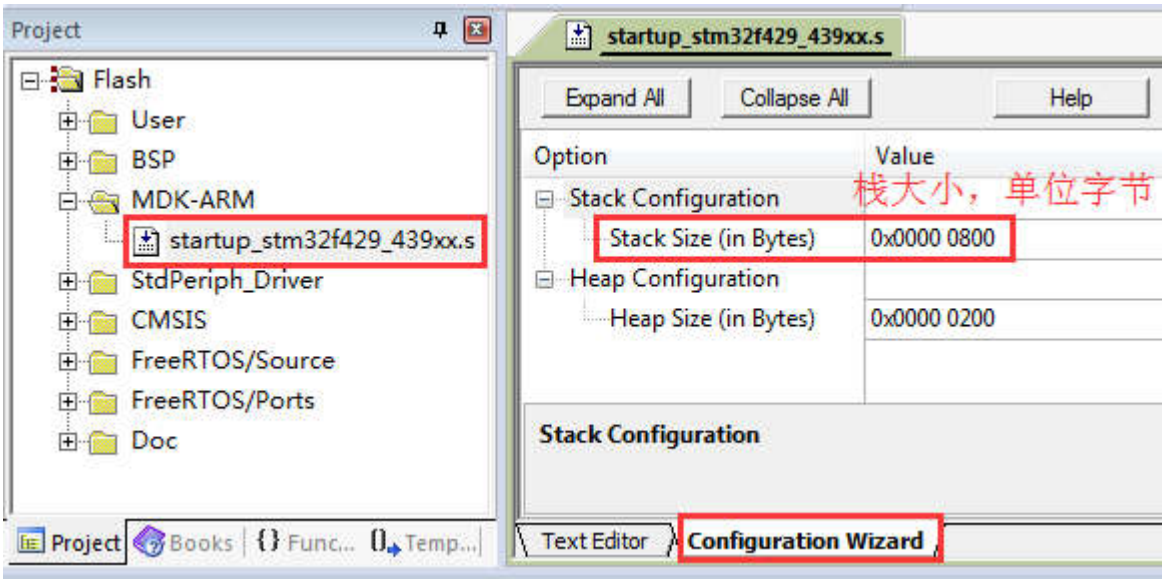


◆ STM32F407 工程中栈大小的配置文件

资料整理 火龙果软件 荐



◆ STM32F429 工程中栈大小的配置文件



在 RTOS 下，上面两个截图中设置的栈大小有了一个新的名字叫系统栈空间，而任务栈是不使用这里的空间的。任务栈不使用这里的栈空间，哪里使用这里的栈空间呢？答案就在中断函数和中断嵌套。

对于这个问题，简单的描述如下，更详细的内容待我们讲解 FreeRTOS 任务切换和双堆栈指针时再细说(此章节在后期 FreeRTOS 教程升级版本时再加入)。

◆ 由于 Cortex-M3 和 M4 内核具有双堆栈指针，MSP 主堆栈指针和 PSP 进程堆栈指针，或者叫 PSP 任务堆栈指针也是可以的。在 FreeRTOS 操作系统中，主堆栈指针 MSP 是给系统栈空间使用的，进程堆栈指针 PSP 是给任务栈使用的。也就是说，在 FreeRTOS 任务中，所有栈空间的使用都是通过 PSP 指针进行指向的。一旦进入了中断函数以及可能发生的中断嵌套都是用的 MSP 指针。这个知识点要记住它，当前可以不知道这是为什么，但是一定要记住。

◆ 实际应用中系统栈空间分配多大，主要是看可能发生的中断嵌套层数，下面我们就按照最坏执行情况进行考虑，所有的寄存器都需要入栈，此时分为两种情况：

资料整理 火龙果软件 荐

- **64 字节**

  对于 Cortex-M3 内核和未使用 FPU（浮点运算单元）功能的 Cortex-M4 内核在发生中断时需要将 16 个通用寄存器全部入栈，每个寄存器占用 4 个字节，也就是 16*4 = 64 字节的空间。可能发生几次中断嵌套就是要 64 乘以几即可。当然，这种是最坏执行情况，也就是所有的寄存器都入栈。

  （注：任务执行的过程中发生中断的话，有 8 个寄存器是自动入栈的，这个栈是任务栈，进入中断以后其余寄存器入栈以及发生中断嵌套都是用的系统栈）

- **200 字节**

  对于具有 FPU（浮点运算单元）功能的 Cortex-M4 内核，如果在任务中进行了浮点运算，那么在发生中断的时候除了 16 个通用寄存器需要入栈，还有 34 个浮点寄存器也是要入栈的，也就是 (16+34)*4 = 200 字节的空间。当然，这种是最坏执行情况，也就是所有的寄存器都入栈。

  （注：任务执行的过程中发送中断的话，有 8 个通用寄存器和 18 个浮点寄存器是自动入栈的，这个栈是任务栈，进入中断以后其余通用寄存器和浮点寄存器入栈以及发生中断嵌套都是用的系统栈）

10.5 FreeRTOS 的任务状态

FreeRTOS 的运行支持以下四种状态：

- **Running—运行态**

  当任务处于实际运行状态被称之为运行态，即 CPU 的使用权被这个任务占用。

- **Ready—就绪态**

  处于就绪态的任务是指那些能够运行（没有被阻塞和挂起），但是当前没有运行的任务，因为同优先级或更高优先级的任务正在运行。

- **Blocked—阻塞态**

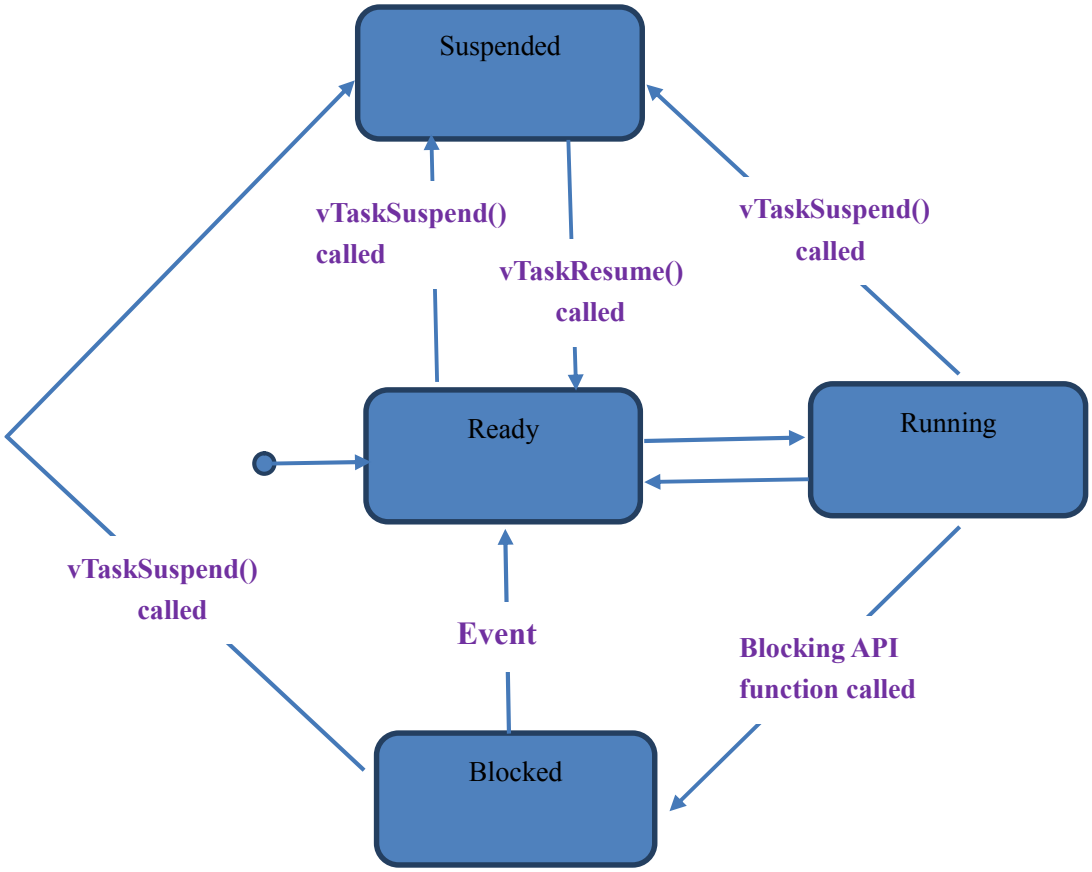
  由于等待信号量，消息队列，事件标志组等而处于的状态被称之为阻塞态，另外任务调用延迟函数也会处于阻塞态。

- **Suspended—挂起态**

  类似阻塞态，通过调用函数 vTaskSuspend()对指定任务进行挂起，挂起后这个任务将不被执行，只有调用函数 xTaskResume()才可以将这个任务从挂起态恢复。

下面是任务在各个状态之间切换的关系图，通过这个图，大家基本可以对任务的运行状态有了一个整体的认识。在后面章节的学习中会对各个状态有一个深入的认识。

资料整理 火龙果软件 荐



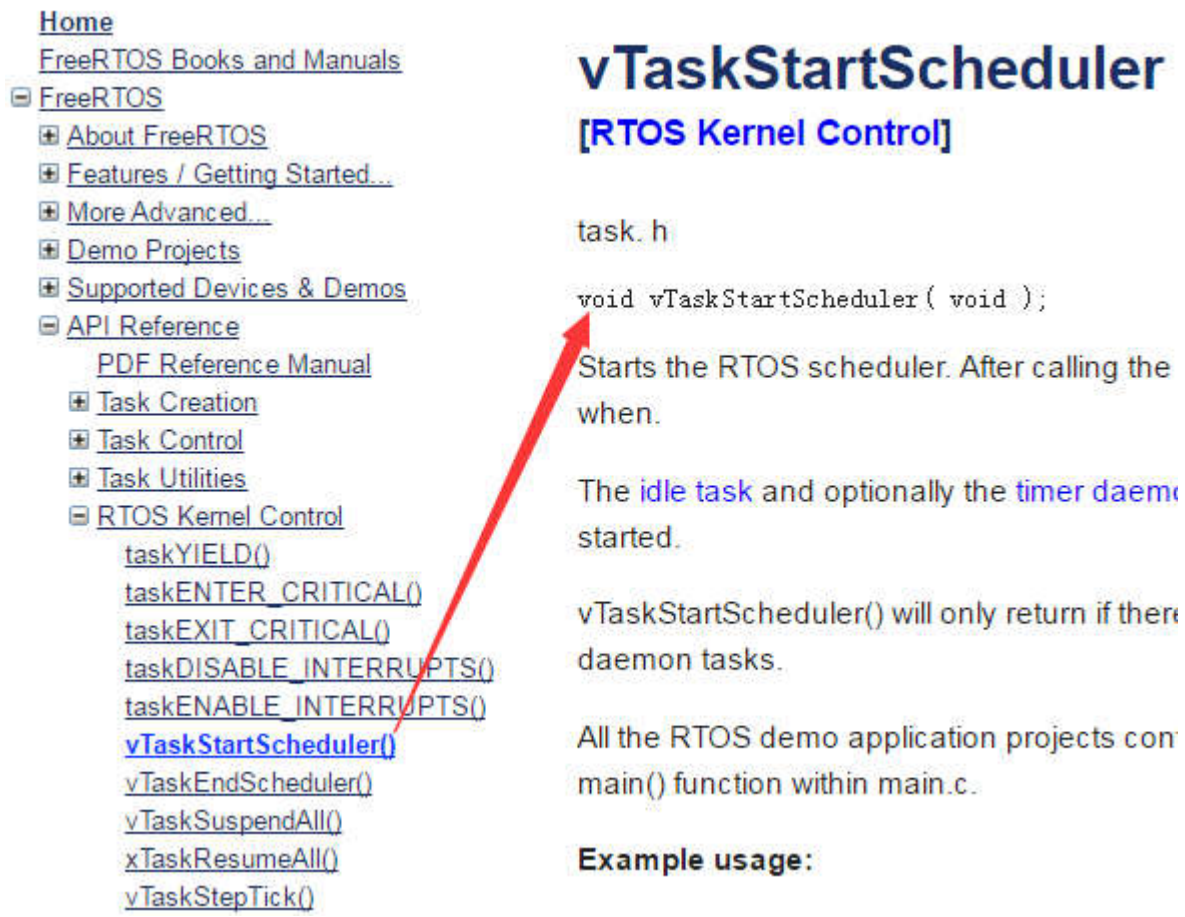
10.6 FreeRTOS 启动

使用如下函数即可启动 FreeRTOS：

- vTaskStartScheduler();

关于这个函数的讲解及其使用方法可以看 FreeRTOS 在线版手册：

资料整理
火龙果软件
荐



这里也对此函数进行下介绍。

函数原型：

```
void vTaskStartScheduler( void );
```

函数描述：

函数 vTaskStartScheduler 用于启动 FreeRTOS 调度器，即启动 FreeRTOS 的多任务执行。

使用这个函数要注意以下几个问题：

1. 空闲任务和可选的定时器任务是在调用这个函数后自动创建的。
2. 正常情况下这个函数是不会返回的，运行到这里极有可能是用于定时器任务或者空闲任务的 heap 空间不足造成创建失败，此时需要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
   #define configTOTAL_HEAP_SIZE　　((size_t) (17 * 1024))

使用举例：

```
/*
*****************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形      参: 无
*    返 回 值: 无
*****************************************************************************************
*/
int main(void)
{
```


资料整理 火龙果软件 荐

```
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是：
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
      2. 保证系统正常启动，不受别的中断影响。
      3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
      在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
      和 cpsie i 是等效的。
    */
    __set_PRIMASK(1);

    /* 硬件初始化 */
    bsp_Init();

    /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
          目中不要使用，因为这个功能比较影响系统实时性。
       2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
    */
    vSetupSysInfoTest();

    /* 创建任务 */
    AppTaskCreate();

   /* 启动调度，开始执行任务 */
   vTaskStartScheduler();

    /*
      如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
      heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
      #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
    */
    while(1);
}
```

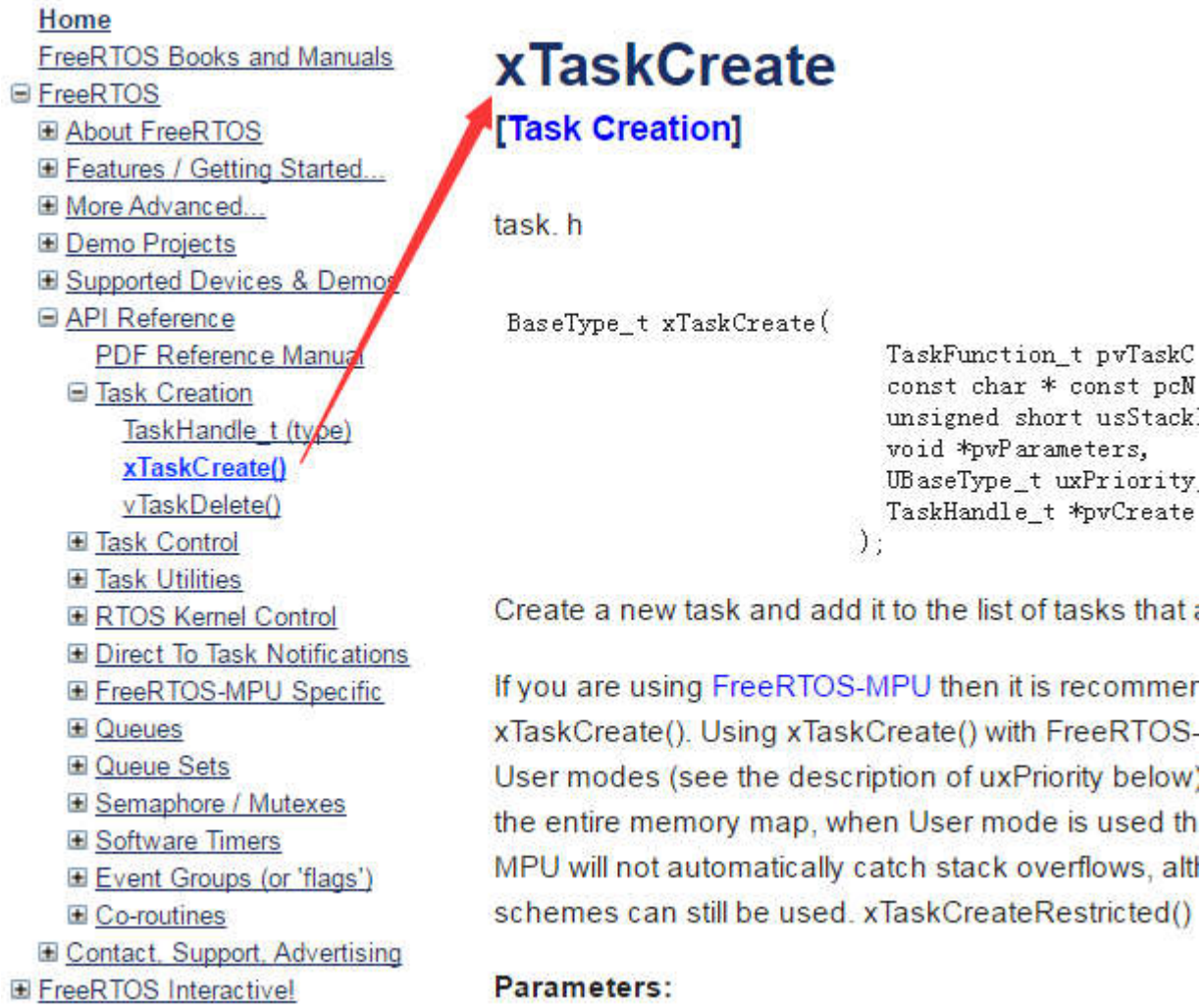
10.7 FreeRTOS 的任务创建

使用如下函数可以实现 FreeRTOS 的任务创建：

- xTaskCreate()

关于这个函数的讲解及其使用方法可以看 FreeRTOS 在线版手册：

资料整理 火龙果软件 荐



这里也对此函数进行下介绍。

函数原型：

```
BaseType_t xTaskCreate(
                        TaskFunction_t pvTaskCode,      /* 任务函数 */
                        const char * const pcName,      /* 任务名   */
                        unsigned short usStackDepth,    /* 任务栈大小，单位 word，也就是 4 字节 */
                        void *pvParameters,             /* 任务参数  */
                        UBaseType_t uxPriority,         /* 任务优先级  */
                        TaskHandle_t *pvCreatedTask     /* 任务句柄  */
                    );
```

函数描述：

函数 xTaskCreate 用于实现 FreeRTOS 操作系统的任务创建，并且还可以自定义任务栈的大小。

- 第 1 个参数填创建任务的函数名。
- 第 2 个参数是任务名，这个参数主要是用于调试目的，调试的时候方便看是哪个任务。
- 第 3 个参数是任务栈大小，单位 word，也就是 4 字节。
- 第 4 个参数是创建的任务函数的形参。
- 第 5 个参数是任务句柄，用于区分不同的任务。

使用举例：

资料整理 火龙果软件 荐

```
/* 任务句柄 */
static TaskHandle_t xHandleTaskStart = NULL;

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
        bsp_KeyScan();
       vTaskDelay(10);
    }
}

/*
*********************************************************************************************************
*    函 数 名: AppTaskCreate
*    功能说明: 创建应用任务
*    形     参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskStart,             /* 任务函数  */
                 "vTaskStart",           /* 任务名    */
                 512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                   /* 任务参数  */
                 4,                      /* 任务优先级*/
                 &xHandleTaskStart );    /* 任务句柄  */
}
```

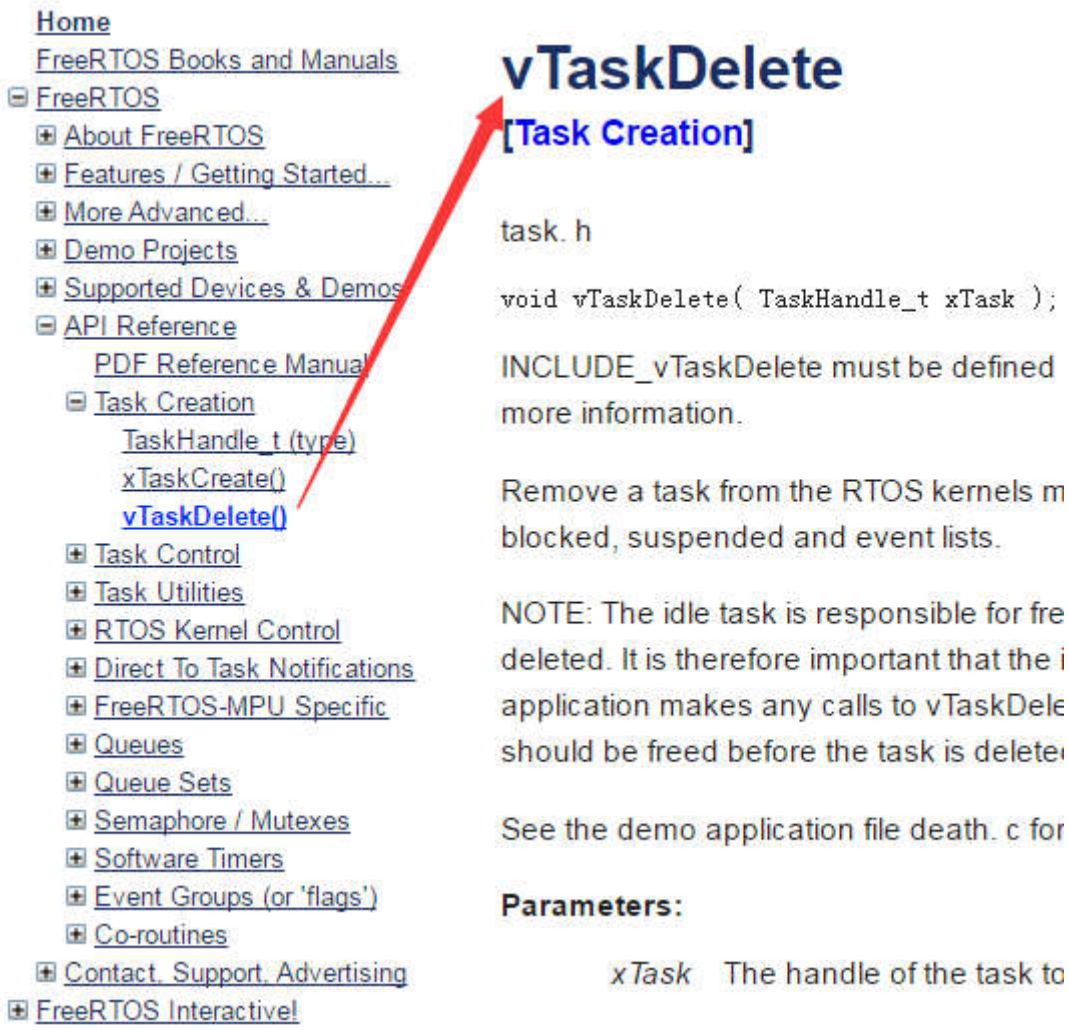
10.8 FreeRTOS 的任务删除

使用如下函数可以实现 FreeRTOS 的任务删除：

- vTaskDelete()

关于这个函数的讲解及其使用方法可以看 FreeRTOS 在线版手册：

资料整理 火龙果软件 荐



这里也对此函数进行下介绍。

函数原型：

```
void vTaskDelete(
    TaskHandle_t xTask ); /* 任务句柄 */
```

函数描述：

函数 vTaskDelete 用于实现 FreeRTOS 操作系统的任务删除。

- 第 1 个参数填要删除任务的句柄

使用这个函数要注意以下问题：

1. 使用此函数需要在 FreeRTOSConfig.h 配置文件中配置如下宏定义为 1

   #define INCLUDE_vTaskDelete 1

2. 如果用往此函数里面填的任务 ID 是 NULL，即数值 0 的话，那么删除的就是当前正在执行的任务，此任务被删除后，FreeRTOS 会切换到任务就绪列表里面下一个要执行的最高优先级任务。
3. 在 FreeRTOS 中，创建任务所需的内存需要在空闲任务中释放，如果用户在 FreeRTOS 中调用了这个函数的话，一定要让空闲任务有执行的机会，否则这块内存是无法释放的。另外，创建的这个任务在使用中申请了动态内存，这个内存不会因为此任务被删除而删除，这一点要注意，一定要在删除前将此内存释放。

资料整理 火龙果软件 荐

使用举例：

```
/* 任务句柄 */
static TaskHandle_t xHandleTaskStart = NULL;

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
        bsp_KeyScan();
       vTaskDelay(10);
    }
}

/*
*********************************************************************************************************
*    函 数 名: AppTaskDelete
*    功能说明: 任务删除
*    形     参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void AppTaskDelete (void)
{
    xTaskCreate( vTaskStart,              /* 任务函数  */
                 "vTaskStart",            /* 任务名    */
                 512,                     /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                    /* 任务参数  */
                 4,                       /* 任务优先级*/
                 &xHandleTaskStart );     /* 任务句柄  */

    /* 删除此任务 */
    if(xHandleTaskStart!= NULL)
    {
         vTaskDelete(xHandleTaskStart);
    }

}
```

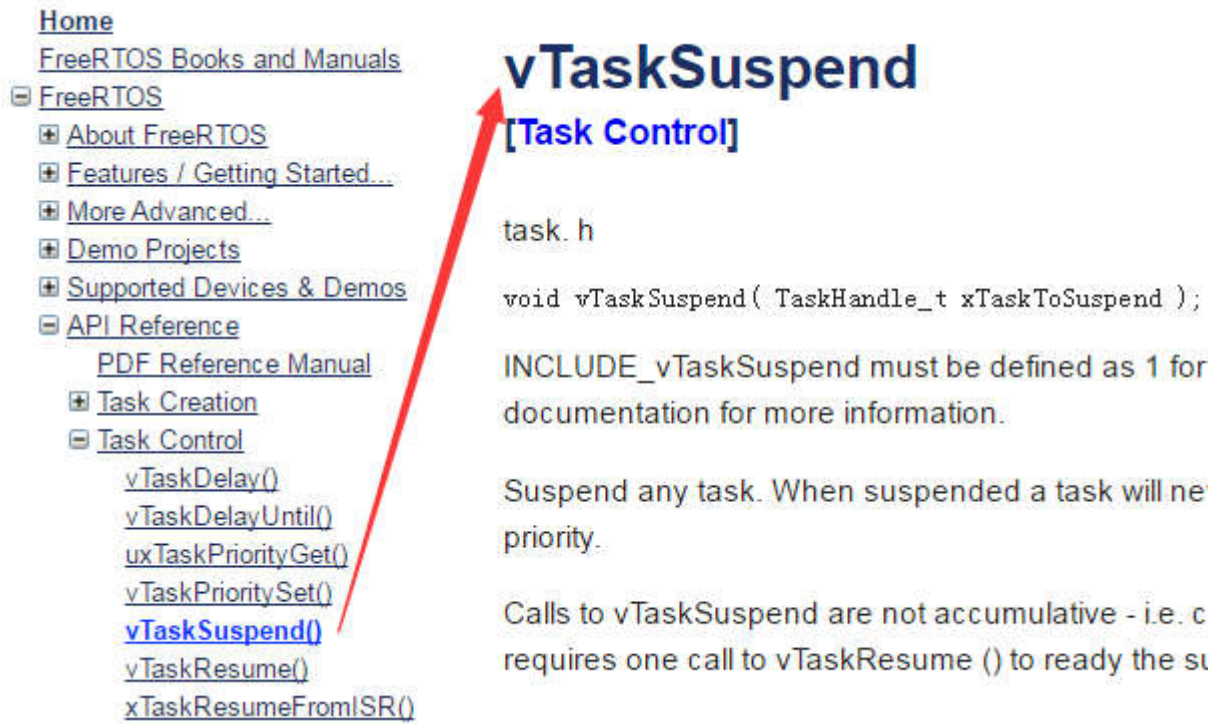
10.9 FreeRTOS 的任务挂起

使用如下函数可以实现 FreeRTOS 的任务挂起：

- xTaskSuspend()

关于这个函数的讲解及其使用方法可以看 FreeRTOS 在线版手册：

资料整理 火龙果软件 荐



这里也对此函数进行下介绍。

函数原型：

```
void vTaskSuspend(
    TaskHandle_t xTaskToSuspend); /* 任务句柄 */
```

函数描述：

函数 vTaskSuspend 用于实现 FreeRTOS 操作系统的任务挂起。

- 第 1 个参数填要挂起任务的句柄

使用这个函数要注意以下问题：

1. 使用此函数需要在 FreeRTOSConfig.h 配置文件中配置如下宏定义为 1

   #define INCLUDE_vTaskSuspend 1

2. 如果用往此函数里面填的任务 ID 是 NULL，即数值 0 的话，那么挂起的就是当前正在执行的任务，此任务被挂起后，FreeRTOS 会切换到任务就绪列表里面下一个要执行的高优先级任务。
3. 多次调用此函数的话，只需调用一次 vTaskResume 即可将任务从挂起态恢复。

使用举例：

```
/* 任务句柄 */
static TaskHandle_t xHandleTaskStart = NULL;

/*
*************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形      参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*************************************************************************************************
*/
```


资料整理 火龙果软件 荐

```
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
        bsp_KeyScan();
       vTaskDelay(10);
    }
}

/*
*****************************************************************************************************
*    函 数 名: AppTaskSuspend
*    功能说明: 任务挂起
*    形     参: 无
*    返 回 值: 无
*****************************************************************************************************
*/
static void AppTaskSuspend (void)
{
    xTaskCreate( vTaskStart,              /* 任务函数  */
                 "vTaskStart",            /* 任务名    */
                 512,                     /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                    /* 任务参数  */
                 4,                       /* 任务优先级*/
                 &xHandleTaskStart );     /* 任务句柄  */

    /* 挂起此任务 */
    if(xHandleTaskStart!= NULL)
    {
         vTaskSuspend(xHandleTaskStart);
    }

}
```

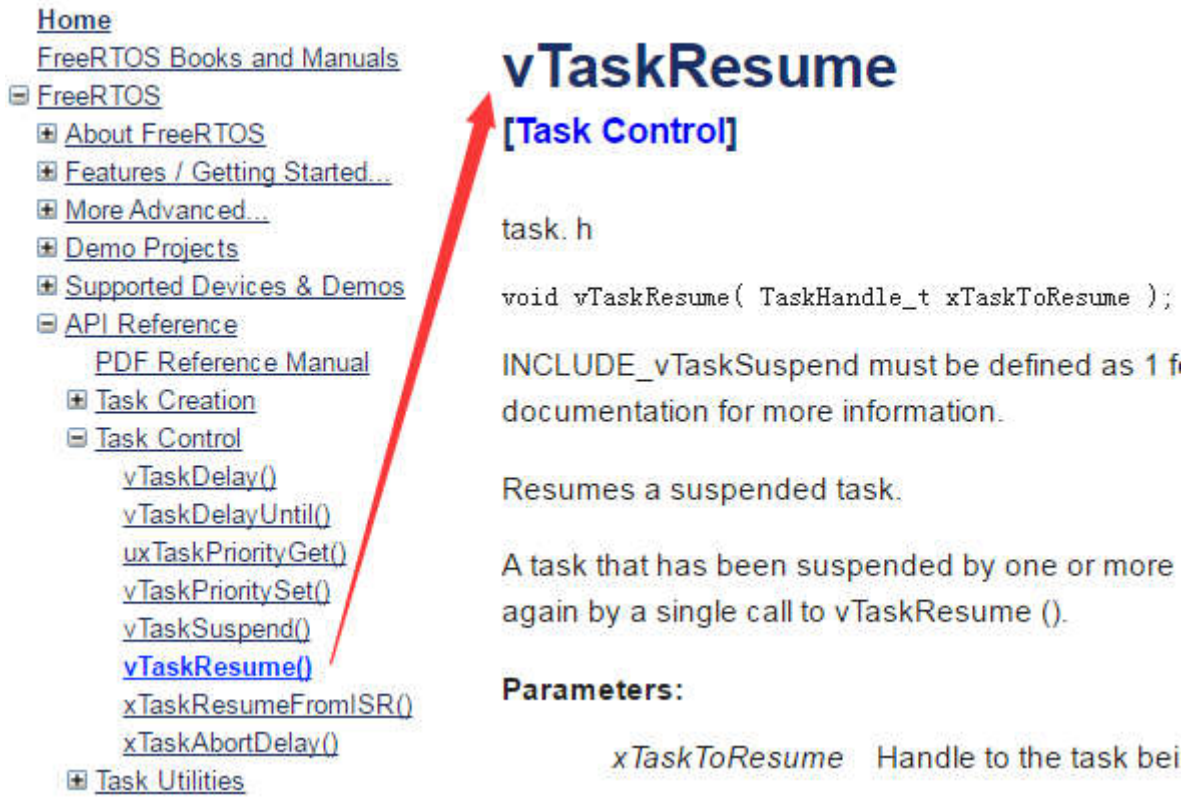
10.10 FreeRTOS 的任务恢复

使用如下函数可以实现 FreeRTOS 的任务恢复：

- xTaskResume()

关于这个函数的讲解及其使用方法可以看 FreeRTOS 在线版手册：

资料整理 火龙果软件 荐



这里也对此函数进行下介绍。

函数原型：

```
void vTaskResume(
    TaskHandle_t xTaskToResume); /* 任务句柄 */
```

函数描述：

函数 vTaskResume 用于实现 FreeRTOS 操作系统的任务挂起

- 第 1 个参数填要恢复任务的句柄

使用这个函数要注意以下问题：

1. 使用此函数需要在 FreeRTOSConfig.h 配置文件中配置如下宏定义为 1

   #define INCLUDE_vTaskSuspend 1

2. 多次调用函数 vTaskSuspend 的话，只需调用一次 vTaskResume 即可将任务从挂起态恢复。
3. 此函数是用于任务代码中调用的，故不可以在中断服务程序中调用此函数，中断服务程序中使用的 xTaskResumeFromISR()，以后缀 FromISR 结尾。

使用举例：

```
/* 任务句柄 */
static TaskHandle_t xHandleTaskStart = NULL;

/*
*****************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形      参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*    优 先 级: 4
```


资料整理 火龙果软件 荐

```
*****************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
        bsp_KeyScan();
       vTaskDelay(10);
    }
}

/*
*****************************************************************************************************
*    函 数 名: AppTaskResume
*    功能说明: 任务恢复
*    形     参: 无
*    返 回 值: 无
*****************************************************************************************************
*/
static void AppTaskResume (void)
{
    xTaskCreate( vTaskStart,              /* 任务函数  */
                 "vTaskStart",            /* 任务名    */
                 512,                     /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                    /* 任务参数  */
                 4,                       /* 任务优先级*/
                 &xHandleTaskStart );     /* 任务句柄  */

    /* 挂起此任务 */
    if(xHandleTaskStart!= NULL)
    {
        vTaskSuspend(xHandleTaskStart);
    }

    /* 恢复此任务 */
    if(xHandleTaskStart!= NULL)
    {
        vTaskResume(xHandleTaskStart);
    }
}
```

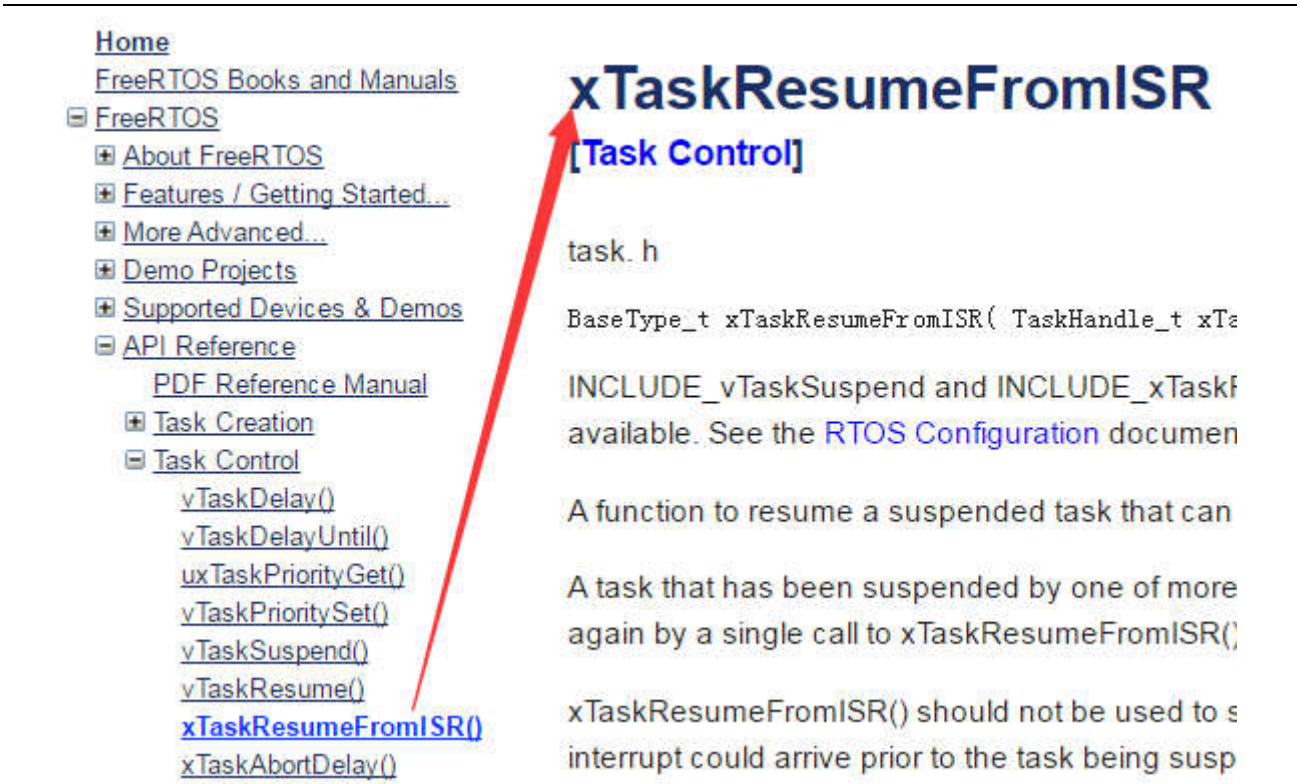
10.11 FreeRTOS 的任务恢复(中断方式)

使用如下函数可以实现 FreeRTOS 的任务恢复（中断方式）：

- xTaskResumeFromISR()

关于这个函数的讲解及其使用方法可以看 FreeRTOS 在线版手册：

资料整理 火龙果软件 荐



这里也对此函数进行下介绍。

函数原型：

```
void vTaskResumeFromISR(
    TaskHandle_t xTaskToResume); /* 任务句柄 */
```

函数描述：

函数 vTaskResumeFromISR 用于实现 FreeRTOS 操作系统的任务恢复。

- 第 1 个参数填要恢复任务的句柄

使用这个函数要注意以下问题：

1. 使用此函数需要在 FreeRTOSConfig.h 配置文件中配置如下宏定义为 1

   #define INCLUDE_xResumeFromISR 1

2. 多次调用函数 vTaskSuspend 的话，只需调用一次 vTaskResumeFromISR 即可将任务从挂起态恢复。
3. 如果用户打算采用这个函数实现中断与任务的同步，要注意一种情况，如果此函数的调用优先于函数 vTaskSuspend 被调用，那么此次同步会丢失，这种情况下建议使用信号量来实现同步。
4. 此函数是用于中断服务程序中调用的，故不可以在任务中使用此函数，任务中使用的是 vTaskResume。

使用举例：

```
/* 任务句柄 */
static TaskHandle_t xHandleTaskStart = NULL;

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
```


资料整理 火龙果软件 荐

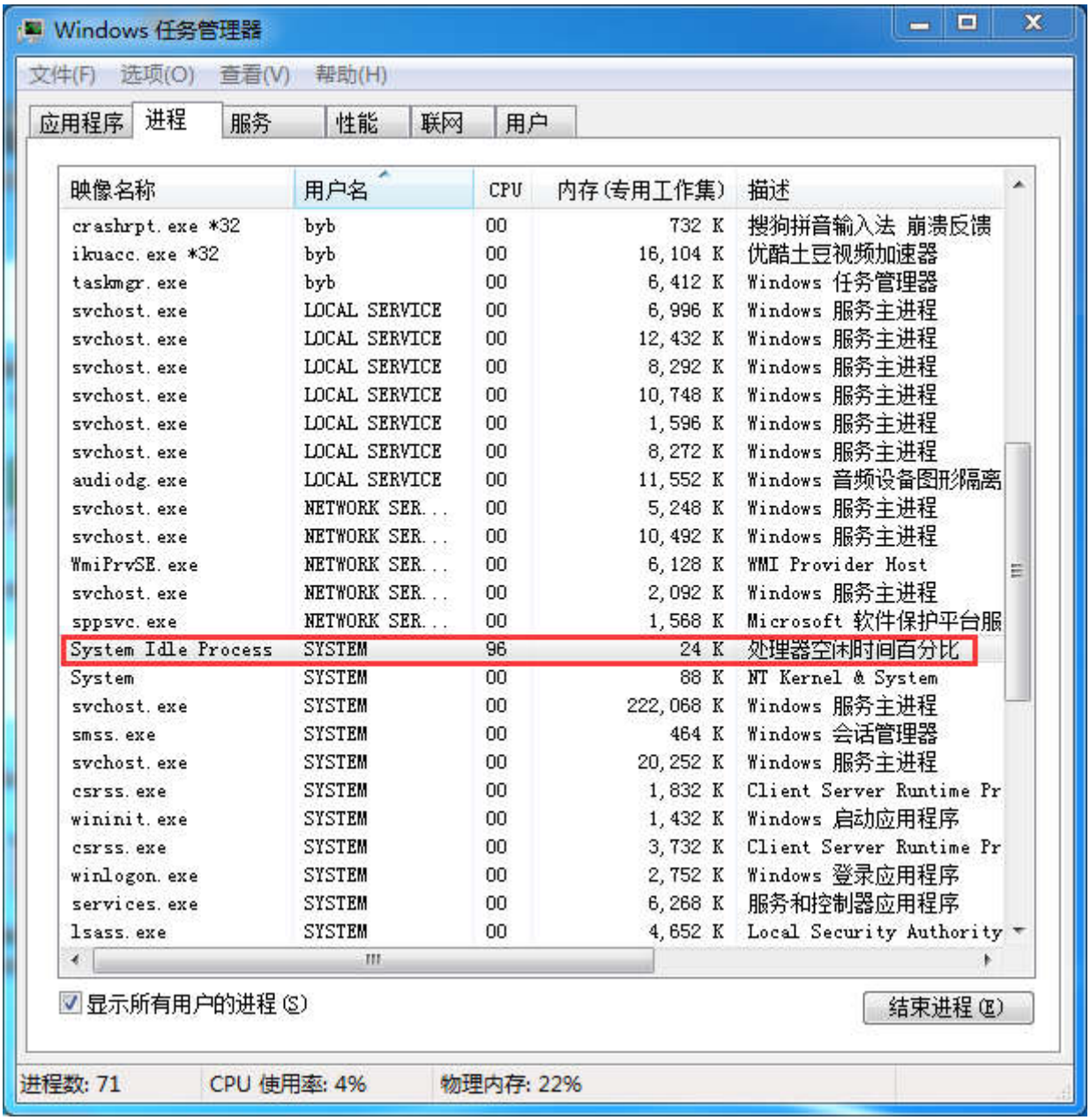
```
*   优 先 级: 4
*****************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
        bsp_KeyScan();
       vTaskDelay(10);
    }
}

/*
*****************************************************************************************************
*    函 数 名: AppTaskResume
*    功能说明: 任务恢复
*    形     参: 无
*    返 回 值: 无
*****************************************************************************************************
*/
static void AppTaskResume (void)
{
    xTaskCreate( vTaskStart,             /* 任务函数  */
                 "vTaskStart",           /* 任务名    */
                 512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                   /* 任务参数  */
                 4,                      /* 任务优先级*/
                 &xHandleTaskStart );    /* 任务句柄  */

    /* 挂起此任务 */
    if(xHandleTaskStart!= NULL)
    {
         vTaskSuspend(xHandleTaskStart);
    }
}

/*
*****************************************************************************************************
*    函 数 名: TIM2_IRQHandler
*    功能说明: 定时器中断。
*    形     参: 无
*    返 回 值: 无
*****************************************************************************************************
*/
static void TIM2_IRQHandler(void)
{
    BaseType_t xYieldRequired;

    /* 中断消息处理，此处省略 */
    ……

   /* 恢复挂起任务 */
    xYieldRequired = xTaskResumeFromISR(xHandleTaskStart);

     /* 退出中断后是否需要执行任务切换 */
    if( xYieldRequired == pdTRUE )
    {
```


资料整理 火龙果软件 荐

```
        portYIELD_FROM_ISR(xYieldRequired);
    }
}
```

10.12 FreeRTOS 的空闲任务

几乎所有的小型 RTOS 中都会有一个空闲任务，空闲任务属于系统任务，是必须要执行的，用户程序不能将其关闭。不光小型系统中有空闲任务，大型的系统里面也有的，比如 WIN7，下面的截图就是 WIN7 中的空闲进程。



空闲任务主要有以下几个作用：

- 用户不能让系统一直在执行各个应用任务，这样的话系统利用率就是 100%，系统就会一直超负荷运行，所以空闲任务很有必要。

资料整理 火龙果软件 荐

◆ 为了更好的实现低功耗，空闲任务也很有必要，用户可以在空闲任务中实现睡眠，停机等低功耗措施。在后面的章节中我们会用到空闲任务的钩子函数，再给大家进行详述。

10.13 实验例程说明

10.13.1 STM32F103 开发板实验

配套例子：

V4-303_FreeRTOS 实验_任务的创建，删除，挂起和恢复

实验目的：

1. 学习 FreeRTOS 的任务创建，删除，挂起和恢复。

实验内容：

1. K1 按键按下，串口打印任务执行情况（波特率 115200，数据位 8，奇偶校验位无，停止位 1）。
2. K2 按键按下，删除任务 vTaskLED。
3. K3 按键按下，重新创建任务 vTaskLED。
4. 摇杆 UP 按键按下，挂起任务 vTaskLED。
5. 摇杆 DOWN 键按下，恢复任务 vTaskLED。
6. 各个任务实现的功能如下：

   vTaskUserIF 任务　　：按键消息处理。

   vTaskLED 任务　　：LED 闪烁。

   vTaskMsgPro 任务　：消息处理，这里是用作 LED 闪烁。

   vTaskStart 任务　　：启动任务，也是最高优先级任务，这里实现按键扫描。

FreeRTOS 的配置：

FreeRTOSConfig.h 文件中的配置如下：

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
#endif

#define configUSE_PREEMPTION          1
#define configUSE_IDLE_HOOK           0
#define configUSE_TICK_HOOK           0
#define configCPU_CLOCK_HZ            ( ( unsigned long ) 72000000 )
#define configTICK_RATE_HZ            ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES          ( 5 )
#define configMINIMAL_STACK_SIZE      ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE         ( ( size_t ) ( 17 * 1024 ) )
#define configMAX_TASK_NAME_LEN       ( 16 )
#define configUSE_TRACE_FACILITY      1
#define configUSE_16_BIT_TICKS        0
```


资料整理 火龙果软件 荐

```
#define configIDLE_SHOULD_YIELD       1

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()     (ulHighFrequencyTimerTicks = 0ul)
#define portGET_RUN_TIME_COUNTER_VALUE()             ulHighFrequencyTimerTicks
//#define portALT_GET_RUN_TIME_COUNTER_VALUE           1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES            0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet          1
#define INCLUDE_uxTaskPriorityGet         1
#define INCLUDE_vTaskDelete               1
#define INCLUDE_vTaskCleanUpResources     0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
     /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
     #define configPRIO_BITS              __NVIC_PRIO_BITS
#else
     #define configPRIO_BITS              4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY             0x0f

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY         0x01
```

几个重要选项说明：

- #define configUSE_PREEMPTION　　1

  使能抢占式调度器

- #define configCPU_CLOCK_HZ　　((unsigned long) 72000000)

  系统主频 72MHz。

- #define configTICK_RATE_HZ　　((TickType_t) 1000)

  系统时钟节拍 1KHz，即 1ms。

- #define configMAX_PRIORITIES　　(5)

  定义可供用户使用的最大优先级数 如果这个定义的是 5 那么用户可以使用的优先级号是 0,1,2,3,4，不包含 5，对于这一点，初学者要特别的注意。

资料整理 火龙果软件 荐

- #define configTOTAL_HEAP_SIZE　　((size_t) (17 * 1024))

  定义堆大小，FreeRTOS 内核，用户动态内存申请，任务栈等都需要用这个空间。

- #define INCLUDE_vTaskSuspend　　1

  使用 FreeRTOS 的任务挂起函数 vTaskSuspend 和任务恢复函数 vTaskResume 必须配置此宏定义为 1。

- configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY　　0x01

  定义受 FreeRTOS 管理的最高优先级中断。简单的说就是允许用户在这个中断服务程序里面调用 FreeRTOS 的 API 的最高优先级。为了进一步说明这个宏定义的的作用，解释如下：

  - 使用 CM 内核的 MCU，官方强烈建议将 NVIC 的优先级分组配置为全抢占式优先级，全部配置为抢占式优先级的好处就是方便管理。
  - 对于 STM32 来说，设置 NVIC 的优先级分组为 4 时，NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4)就是全部配置为抢占式优先级。又因为 STM32 的优先级设置仅使用 CM 内核 8bit 中的高 4bit，即只能区分 2^4 = 16 种优先级。因此当优先级分组设置为 4 的时候可供用户选择抢占式优先级为 0 到 15，共 16 个优先级，配置为 0 表示最高优先级，配置为 15 表示最低优先级，不存在子优先级。
  - 这里配置 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 为 0x01 表示用户可以在抢占式优先级为 1 到 15 的中断里面调用 FreeRTOS 的 API 函数，抢占式优先级为 0 的中断里面是不允许调用的。

    更多关于这个参数说明请参看第 12 章。

FreeRTOS 任务调试信息（按 K1 按键，串口打印）：

上面截图中打印出来的任务状态字母 B, R, D, S 对应如下含义：

资料整理 火龙果软件 荐

#define tskBLOCKED_CHAR　　('B')　任务阻塞

#define tskREADY_CHAR　　('R')　任务就绪

#define tskDELETED_CHAR　　('D')　任务删除

#define tskSUSPENDED_CHAR　('S')　任务挂起

程序设计：

◆ **任务栈大小分配：**

vTaskUserIF 任务　　：2048 字节

vTaskLED 任务　　：2048 字节

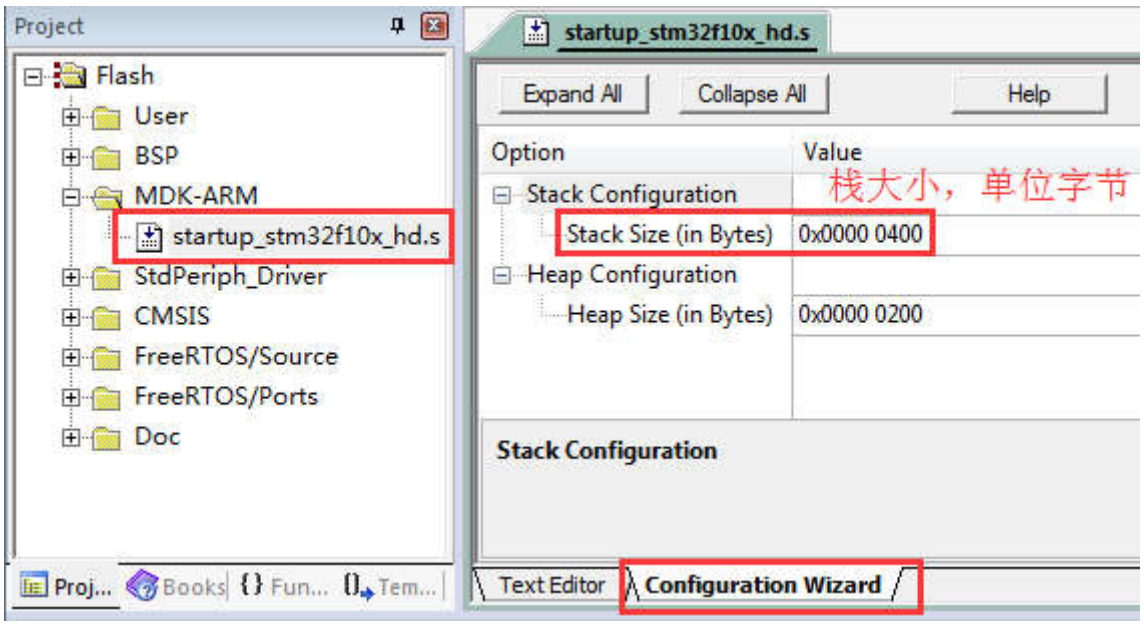
vTaskMsgPro 任务　：2048 字节

vTaskStart 任务　　：2048 字节

任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE　　((size_t) (17 * 1024))

◆ **系统栈大小分配：**



◆ **FreeROTS 初始化：**

```
/*
*********************************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
int main(void)
{
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是:
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
      2. 保证系统正常启动，不受别的中断影响。
```


资料整理 火龙果软件 荐

```
        3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
        在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
        和 cpsie i 是等效的。
    */
    __set_PRIMASK(1);

    /* 硬件初始化 */
    bsp_Init();

    /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
          目中不要使用，因为这个功能比较影响系统实时性。
       2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
    */
    vSetupSysInfoTest();

    /* 创建任务 */
    AppTaskCreate();

    /* 启动调度，开始执行任务 */
    vTaskStartScheduler();

    /*
      如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
      heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
      #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
    */
    while(1);
}
```

◆ **硬件外设初始化**

硬件外设的初始化是在 bsp.c 文件实现：

```
/*
*********************************************************************************************************
*    函 数 名: bsp_Init
*    功能说明: 初始化硬件设备。只需要调用一次。该函数配置 CPU 寄存器和外设的寄存器并初始化一些全局变量。
*              全局变量。
*    形    参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
void bsp_Init(void)
{
    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        系统时钟缺省配置为 72MHz，如果需要更改，可以修改 system_stm32f10x.c 文件
    */

    /* 优先级分组设置为 4，可配置 0-15 级抢占式优先级，0 级子优先级，即不存在子优先级。*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

    bsp_InitUart();    /* 初始化串口 */
    bsp_InitLed();     /* 初始 LED 指示灯端口 */
    bsp_InitKey();     /* 初始化按键 */
}
```

◆ **FreeRTOS 任务创建：**

资料整理 火龙果软件 荐

```
/*
*********************************************************************************************************
*    函 数 名: AppTaskCreate
*    功能说明: 创建应用任务
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


     xTaskCreate( vTaskLED,           /* 任务函数  */
                  "vTaskLED",         /* 任务名    */
                  512,                /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,               /* 任务参数  */
                  2,                  /* 任务优先级*/
                  &xHandleTaskLED ); /* 任务句柄  */

     xTaskCreate( vTaskMsgPro,            /* 任务函数  */
                  "vTaskMsgPro",          /* 任务名    */
                  512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,                   /* 任务参数  */
                  3,                      /* 任务优先级*/
                  &xHandleTaskMsgPro );  /* 任务句柄  */


     xTaskCreate( vTaskStart,             /* 任务函数  */
                  "vTaskStart",           /* 任务名    */
                  512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,                   /* 任务参数  */
                  4,                      /* 任务优先级*/
                  &xHandleTaskStart );   /* 任务句柄  */
}
```

◆ **四个 FreeRTOS 任务的实现：**

```
/*
*********************************************************************************************************
*    函 数 名: vTaskTaskUserIF
*    功能说明: 接口消息处理。
*    形      参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 1  (数值越小优先级越低，这个跟 uCOS 相反)
*********************************************************************************************************
*/
static void vTaskTaskUserIF(void *pvParameters)
{
     uint8_t ucKeyCode;
     uint8_t pcWriteBuffer[500];

    while(1)
    {
```


资料整理 火龙果软件 荐

```
        ucKeyCode = bsp_GetKey();

        if (ucKeyCode != KEY_NONE)
        {
            switch (ucKeyCode)
            {
                /* K1 键按下 打印任务执行情况 */
                case KEY_DOWN_K1:
                    printf("================================================\r\n");
                    printf("任务名      任务状态 优先级   剩余栈 任务序号\r\n");
                    vTaskList((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);

                    printf("\r\n 任务名       运行计数         使用率\r\n");
                    vTaskGetRunTimeStats((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);
                    break;

                /* K2 键按下 删除任务 vTaskLED */
                case KEY_DOWN_K2:
                    printf("K2 键按下，删除任务 vTaskLED\r\n");
                    if(xHandleTaskLED != NULL)
                    {
                        vTaskDelete(xHandleTaskLED);
                        xHandleTaskLED = NULL;
                    }
                    break;

                /* K3 键按下 重新创建任务 vTaskLED */
                case KEY_DOWN_K3:
                    printf("K3 键按下，重新创建任务 vTaskLED\r\n");
                    if(xHandleTaskLED == NULL)
                    {
                        xTaskCreate(    vTaskLED,           /* 任务函数  */
                                        "vTaskLED",         /* 任务名    */
                                        512,                /* stack 大小，单位 word，也就是 4 字节 */
                                        NULL,               /* 任务参数  */
                                        2,                  /* 任务优先级*/
                                        &xHandleTaskLED );  /* 任务句柄  */
                    }
                    break;

                /* 摇杆 UP 键按下，挂起任务 vTaskLED */
                case JOY_DOWN_U:
                    printf("摇杆 UP 键按下，挂起任务 vTaskLED\r\n");
                    vTaskSuspend(xHandleTaskLED);
                    break;

                /* 摇杆 DOWN 键按下，恢复任务 vTaskLED */
                case JOY_DOWN_D:
                    printf("摇杆 DOWN 键按下，恢复任务 vTaskLED\r\n");
                    vTaskResume(xHandleTaskLED);
                    break;

                /* 其他的键值不处理 */
                default:
                    break;
            }
```


资料整理 火龙果软件 荐

```
        }

        vTaskDelay(20);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskLED
*    功能说明: LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 2
*********************************************************************************************************
*/
static void vTaskLED(void *pvParameters)
{
    while(1)
    {
        bsp_LedToggle(2);
        vTaskDelay(200);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskMsgPro
*    功能说明: 消息处理，这里是用作 LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 3
*********************************************************************************************************
*/
static void vTaskMsgPro(void *pvParameters)
{
    while(1)
    {
        bsp_LedToggle(3);
        vTaskDelay(300);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
        /* 按键扫描 */
        bsp_KeyScan();
        vTaskDelay(10);
```


资料整理 火龙果软件 荐

```
    }
}
```

10.13.2 STM32F407 开发板实验

配套例子：

V5-303_FreeRTOS 实验_任务的创建，删除，挂起和恢复

实验目的：

1. 学习 FreeRTOS 的任务创建，删除，挂起和恢复。

实验内容：

1. K1 按键按下，串口打印任务执行情况（波特率 115200，数据位 8，奇偶校验位无，停止位 1）。
2. K2 按键按下，删除任务 vTaskLED。
3. K3 按键按下，重新创建任务 vTaskLED。
4. 摇杆 UP 按键按下，挂起任务 vTaskLED。
5. 摇杆 DOWN 键按下，恢复任务 vTaskLED。
6. 各个任务实现的功能如下：

   vTaskUserIF 任务　　：按键消息处理。

   vTaskLED 任务　　：LED 闪烁。

   vTaskMsgPro 任务　：消息处理，这里是用作 LED 闪烁。

   vTaskStart 任务　　：启动任务，也是最高优先级任务，这里实现按键扫描。

FreeRTOS 的配置：

FreeRTOSConfig.h 文件中的配置如下：

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;

/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
#endif

#define configUSE_PREEMPTION          1
#define configUSE_IDLE_HOOK           0
#define configUSE_TICK_HOOK           0
#define configCPU_CLOCK_HZ            ( ( unsigned long ) 168000000 )
#define configTICK_RATE_HZ            ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES          ( 5 )
#define configMINIMAL_STACK_SIZE      ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE         ( ( size_t ) ( 30 * 1024 ) )
#define configMAX_TASK_NAME_LEN       ( 16 )
#define configUSE_TRACE_FACILITY      1
#define configUSE_16_BIT_TICKS        0
```


资料整理 火龙果软件 荐

```
#define configIDLE_SHOULD_YIELD      1

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()     (ulHighFrequencyTimerTicks = 0ul)
#define portGET_RUN_TIME_COUNTER_VALUE()             ulHighFrequencyTimerTicks
//#define portALT_GET_RUN_TIME_COUNTER_VALUE           1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES             0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet      1
#define INCLUDE_uxTaskPriorityGet         1
#define INCLUDE_vTaskDelete               1
#define INCLUDE_vTaskCleanUpResources 0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
     /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
     #define configPRIO_BITS              __NVIC_PRIO_BITS
#else
     #define configPRIO_BITS              4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY             0x0f

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY    0x01
```

几个重要选项说明：

- #define configUSE_PREEMPTION　　1

  使能抢占式调度器

- #define configCPU_CLOCK_HZ　　((unsigned long) 168000000)

  系统主频 168MHz。

- #define configTICK_RATE_HZ　　((TickType_t) 1000)

  系统时钟节拍 1KHz，即 1ms。

- #define configMAX_PRIORITIES　　(5)

  定义可供用户使用的最大优先级数 如果这个定义的是 5 那么用户可以使用的优先级号是 0,1,2,3,4，不包含 5，对于这一点，初学者要特别的注意。

资料整理 火龙果软件 荐

- #define configTOTAL_HEAP_SIZE　　((size_t) (30 * 1024))

  定义堆大小，FreeRTOS 内核，用户动态内存申请，任务栈等都需要用这个空间。

- #define INCLUDE_vTaskSuspend　　1

  使用 FreeRTOS 的任务挂起函数 vTaskSuspend 和任务恢复函数 vTaskResume 必须配置此宏定义为 1。

- configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY　　0x01

  定义受 FreeRTOS 管理的最高优先级中断。简单的说就是允许用户在这个中断服务程序里面调用 FreeRTOS 的 API 的最高优先级。为了进一步说明这个宏定义的的作用，解释如下：

  - 使用 CM 内核的 MCU，官方强烈建议将 NVIC 的优先级分组配置为全抢占式优先级，全部配置为抢占式优先级的好处就是方便管理。
  - 对于 STM32 来说，设置 NVIC 的优先级分组为 4 时，NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4)就是全部配置为抢占式优先级。又因为 STM32 的优先级设置仅使用 CM 内核 8bit 中的高 4bit，即只能区分 2^4 = 16 种优先级。因此当优先级分组设置为 4 的时候可供用户选择抢占式优先级为 0 到 15，共 16 个优先级，配置为 0 表示最高优先级，配置为 15 表示最低优先级，不存在子优先级。
  - 这里配置 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 为 0x01 表示用户可以在抢占式优先级为 1 到 15 的中断里面调用 FreeRTOS 的 API 函数，抢占式优先级为 0 的中断里面是不允许调用的。

    更多关于这个参数说明请参看第 12 章。

FreeRTOS 任务调试信息（按 K1 按键，串口打印）：

资料整理 火龙果软件 荐

上面截图中打印出来的任务状态字母 B, R, D, S 对应如下含义：

#define tskBLOCKED_CHAR　　('B')　任务阻塞

#define tskREADY_CHAR　　('R')　任务就绪

#define tskDELETED_CHAR　　('D')　任务删除

#define tskSUSPENDED_CHAR　('S')　任务挂起

程序设计：

◆ **任务栈大小分配：**

vTaskUserIF 任务　　：2048 字节

vTaskLED 任务　　：2048 字节

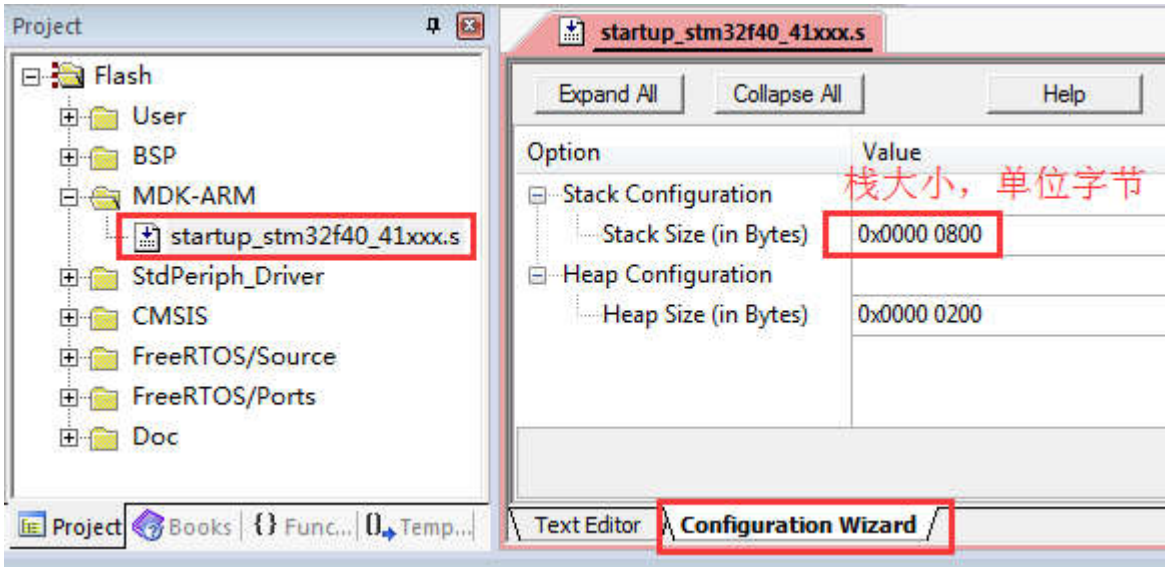
vTaskMsgPro 任务　：2048 字节

vTaskStart 任务　　：2048 字节

任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE　　((size_t) (30 * 1024))

◆ **系统栈大小分配：**



◆ **FreeROTS 初始化：**

```
/*
*********************************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
int main(void)
{
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是:
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
```


资料整理 火龙果软件 荐

```
        2. 保证系统正常启动，不受别的中断影响。
        3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
        在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
        和 cpsie i 是等效的。
      */
      __set_PRIMASK(1);

      /* 硬件初始化 */
      bsp_Init();

      /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
            目中不要使用，因为这个功能比较影响系统实时性。
         2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
      */
      vSetupSysInfoTest();

      /* 创建任务 */
      AppTaskCreate();

     /* 启动调度，开始执行任务 */
     vTaskStartScheduler();

      /*
        如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
        heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
        #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
      */
      while(1);
}
```

◆ **硬件外设初始化**

硬件外设的初始化是在 bsp.c 文件实现：

```
/*
*********************************************************************************************************
*    函 数 名: bsp_Init
*    功能说明: 初始化所有的硬件设备。该函数配置 CPU 寄存器和外设的寄存器并初始化一些全局变量。只需要调用一次
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
void bsp_Init(void)
{
    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        系统时钟缺省配置为 168MHz，如果需要更改，可以修改 system_stm32f4xx.c 文件
    */
    /* 优先级分组设置为 4，可配置 0-15 级抢占式优先级，0 级子优先级，即不存在子优先级。*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

    bsp_InitUart();    /* 初始化串口 */
    bsp_InitKey();     /* 初始化按键变量 */
    bsp_InitLed();     /* 初始 LED 指示灯端口 */
}
```

◆ **FreeRTOS 任务创建：**

```
/*
```


资料整理 火龙果软件 荐

```
*****************************************************************************************************
*    函 数 名: AppTaskCreate
*    功能说明: 创建应用任务
*    形     参: 无
*    返 回 值: 无
*****************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


     xTaskCreate( vTaskLED,           /* 任务函数  */
                  "vTaskLED",         /* 任务名    */
                  512,                /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,               /* 任务参数  */
                  2,                  /* 任务优先级*/
                  &xHandleTaskLED ); /* 任务句柄  */

     xTaskCreate( vTaskMsgPro,            /* 任务函数  */
                  "vTaskMsgPro",          /* 任务名    */
                  512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,                   /* 任务参数  */
                  3,                      /* 任务优先级*/
                  &xHandleTaskMsgPro );  /* 任务句柄  */


     xTaskCreate( vTaskStart,             /* 任务函数  */
                  "vTaskStart",           /* 任务名    */
                  512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,                   /* 任务参数  */
                  4,                      /* 任务优先级*/
                  &xHandleTaskStart );   /* 任务句柄  */
}
```

◆ **四个 FreeRTOS 任务的实现：**

```
/*
*****************************************************************************************************
*    函 数 名: vTaskTaskUserIF
*    功能说明: 接口消息处理。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 1  (数值越小优先级越低，这个跟 uCOS 相反)
*****************************************************************************************************
*/
static void vTaskTaskUserIF(void *pvParameters)
{
     uint8_t ucKeyCode;
     uint8_t pcWriteBuffer[500];

    while(1)
    {
         ucKeyCode = bsp_GetKey();
```


资料整理 火龙果软件 荐

```
if (ucKeyCode != KEY_NONE)
{
    switch (ucKeyCode)
    {
        /* K1 键按下 打印任务执行情况 */
        case KEY_DOWN_K1:
            printf("==============================================\r\n");
            printf("任务名      任务状态 优先级   剩余栈 任务序号\r\n");
            vTaskList((char *)&pcWriteBuffer);
            printf("%s\r\n", pcWriteBuffer);

            printf("\r\n 任务名       运行计数         使用率\r\n");
            vTaskGetRunTimeStats((char *)&pcWriteBuffer);
            printf("%s\r\n", pcWriteBuffer);
            break;

        /* K2 键按下 删除任务 vTaskLED */
        case KEY_DOWN_K2:
            printf("K2 键按下，删除任务 vTaskLED\r\n");
            if(xHandleTaskLED != NULL)
            {
                vTaskDelete(xHandleTaskLED);
                xHandleTaskLED = NULL;
            }
            break;

        /* K3 键按下 重新创建任务 vTaskLED */
        case KEY_DOWN_K3:
            printf("K3 键按下，重新创建任务 vTaskLED\r\n");
            if(xHandleTaskLED == NULL)
            {
                xTaskCreate(    vTaskLED,           /* 任务函数  */
                                "vTaskLED",         /* 任务名    */
                                512,                /* stack 大小，单位 word，也就是 4 字节 */
                                NULL,               /* 任务参数  */
                                2,                  /* 任务优先级*/
                                &xHandleTaskLED );  /* 任务句柄  */
            }
            break;

        /* 摇杆 UP 键按下，挂起任务 vTaskLED */
        case JOY_DOWN_U:
            printf("摇杆 UP 键按下，挂起任务 vTaskLED\r\n");
            vTaskSuspend(xHandleTaskLED);
            break;

        /* 摇杆 DOWN 键按下，恢复任务 vTaskLED */
        case JOY_DOWN_D:
            printf("摇杆 DOWN 键按下，恢复任务 vTaskLED\r\n");
            vTaskResume(xHandleTaskLED);
            break;

        /* 其他的键值不处理 */
        default:
            break;
    }
}
```


资料整理 火龙果软件 荐

```
        vTaskDelay(20);
    }
}

/*
*********************************************************************************************
*    函 数 名: vTaskLED
*    功能说明: LED 闪烁
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 2
*********************************************************************************************
*/
static void vTaskLED(void *pvParameters)
{
    while(1)
    {
        bsp_LedToggle(2);
        vTaskDelay(200);
    }
}

/*
*********************************************************************************************
*    函 数 名: vTaskMsgPro
*    功能说明: 消息处理，这里是用作 LED 闪烁
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 3
*********************************************************************************************
*/
static void vTaskMsgPro(void *pvParameters)
{
    while(1)
    {
        bsp_LedToggle(3);
        vTaskDelay(300);
    }
}

/*
*********************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
        /* 按键扫描 */
        bsp_KeyScan();
        vTaskDelay(10);
    }
```


资料整理 火龙果软件 荐

```
}
```

10.13.3 STM32F429 开发板实验

配套例子：

V6-303_FreeRTOS 实验_任务的创建，删除，挂起和恢复

实验目的：

1. 学习 FreeRTOS 的任务创建，删除，挂起和恢复。

实验内容：

1. K1 按键按下，串口打印任务执行情况（波特率 115200，数据位 8，奇偶校验位无，停止位 1）。
2. K2 按键按下，删除任务 vTaskLED。
3. K3 按键按下，重新创建任务 vTaskLED。
4. 摇杆 UP 按键按下，挂起任务 vTaskLED。
5. 摇杆 DOWN 键按下，恢复任务 vTaskLED。
6. 各个任务实现的功能如下：

   vTaskUserIF 任务　　：按键消息处理。

   vTaskLED 任务　　：LED 闪烁。

   vTaskMsgPro 任务　：消息处理，这里是用作 LED 闪烁。

   vTaskStart 任务　　：启动任务，也是最高优先级任务，这里实现按键扫描。

FreeRTOS 的配置：

FreeRTOSConfig.h 文件中的配置如下：

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
#endif

#define configUSE_PREEMPTION          1
#define configUSE_IDLE_HOOK           0
#define configUSE_TICK_HOOK           0
#define configCPU_CLOCK_HZ            ( ( unsigned long ) 168000000 )
#define configTICK_RATE_HZ            ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES          ( 5 )
#define configMINIMAL_STACK_SIZE      ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE         ( ( size_t ) ( 30 * 1024 ) )
#define configMAX_TASK_NAME_LEN       ( 16 )
#define configUSE_TRACE_FACILITY      1
#define configUSE_16_BIT_TICKS        0
#define configIDLE_SHOULD_YIELD       1

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()     (ulHighFrequencyTimerTicks = 0ul)
```


资料整理 火龙果软件 荐

```
#define portGET_RUN_TIME_COUNTER_VALUE()            ulHighFrequencyTimerTicks
//#define portALT_GET_RUN_TIME_COUNTER_VALUE          1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES            0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet          1
#define INCLUDE_uxTaskPriorityGet         1
#define INCLUDE_vTaskDelete               1
#define INCLUDE_vTaskCleanUpResources     0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
     /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
     #define configPRIO_BITS              __NVIC_PRIO_BITS
#else
     #define configPRIO_BITS              4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY             0x0f

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY         0x01
```

几个重要选项说明：

- #define configUSE_PREEMPTION　　1

  使能抢占式调度器

- #define configCPU_CLOCK_HZ　　((unsigned long) 168000000)

  系统主频 168MHz。

- #define configTICK_RATE_HZ　　((TickType_t) 1000)

  系统时钟节拍 1KHz，即 1ms。

- #define configMAX_PRIORITIES　　(5)

  定义可供用户使用的最大优先级数 如果这个定义的是 5 那么用户可以使用的优先级号是 0,1,2,3,4，不包含 5，对于这一点，初学者要特别的注意。

- #define configTOTAL_HEAP_SIZE　　((size_t) (30 * 1024))

  定义堆大小，FreeRTOS 内核，用户动态内存申请，任务栈等都需要用这个空间。

- #define INCLUDE_vTaskSuspend　　1

资料整理 火龙果软件 荐

使用 FreeRTOS 的任务挂起函数 vTaskSuspend 和任务恢复函数 vTaskResume 必须配置此宏定义为 1。

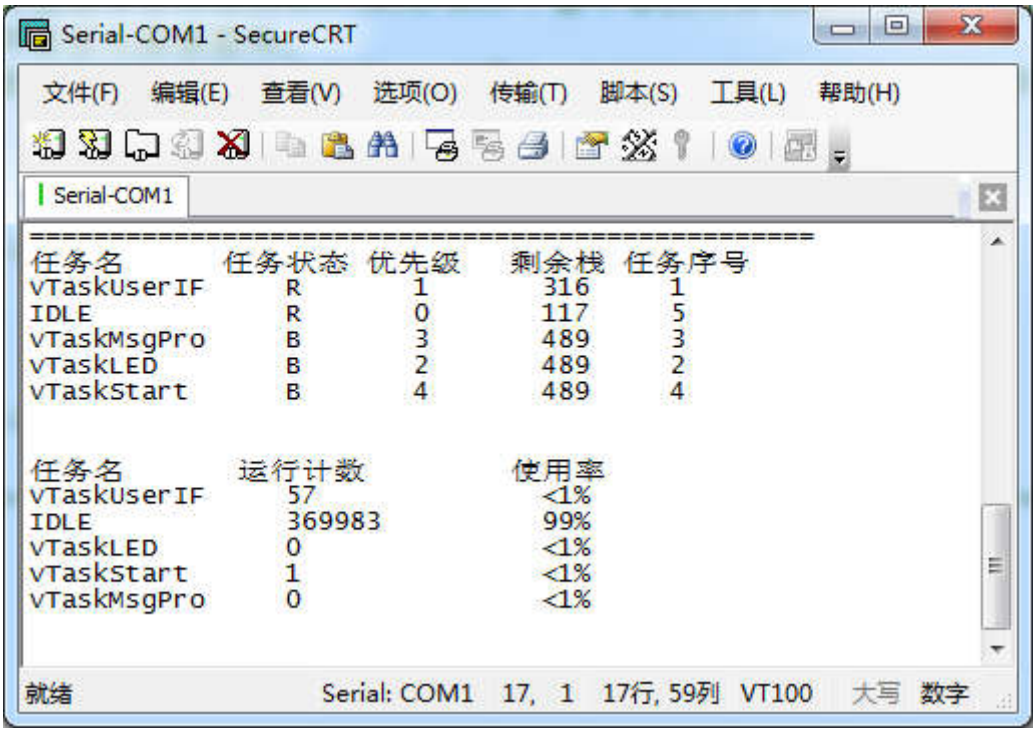
◆ configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY　　0x01

定义受 FreeRTOS 管理的最高优先级中断。简单的说就是允许用户在这个中断服务程序里面调用 FreeRTOS 的 API 的最高优先级。为了进一步说明这个宏定义的的作用，解释如下：

- 使用 CM 内核的 MCU，官方强烈建议将 NVIC 的优先级分组配置为全抢占式优先级，全部配置为抢占式优先级的好处就是方便管理。
- 对于 STM32 来说，设置 NVIC 的优先级分组为 4 时，NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4)就是全部配置为抢占式优先级。又因为 STM32 的优先级设置仅使用 CM 内核 8bit 中的高 4bit，即只能区分 2^4 = 16 种优先级。因此当优先级分组设置为 4 的时候可供用户选择抢占式优先级为 0 到 15，共 16 个优先级，配置为 0 表示最高优先级，配置为 15 表示最低优先级，不存在子优先级。
- 这里配置 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 为 0x01 表示用户可以在抢占式优先级为 1 到 15 的中断里面调用 FreeRTOS 的 API 函数，抢占式优先级为 0 的中断里面是不允许调用的。

  更多关于这个参数说明请参看第 12 章。

FreeRTOS 任务调试信息（按 K1 按键，串口打印）：



上面截图中打印出来的任务状态字母 B, R, D, S 对应如下含义：

#define tskBLOCKED_CHAR　　('B')　任务阻塞

#define tskREADY_CHAR　　('R')　任务就绪

资料整理 火龙果软件 荐

#define tskDELETED_CHAR　　('D')　任务删除

#define tskSUSPENDED_CHAR　('S')　任务挂起

程序设计：

◆ **任务栈大小分配：**

vTaskUserIF 任务　：2048 字节

vTaskLED 任务　：2048 字节

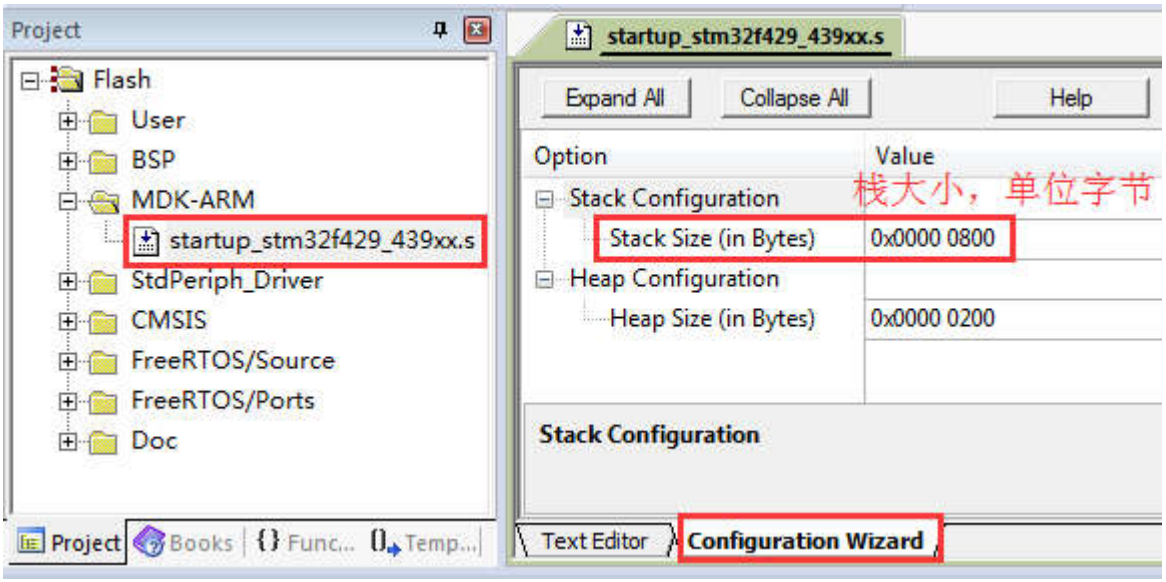
vTaskMsgPro 任务　：2048 字节

vTaskStart 任务　：2048 字节

任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE　　((size_t) (30 * 1024))

◆ **系统栈大小分配：**



◆ **FreeROTS 初始化：**

```
/*
*********************************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形    参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
int main(void)
{
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是:
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
      2. 保证系统正常启动，不受别的中断影响。
      3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
      在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
      和 cpsie i 是等效的。
    */
```


资料整理 火龙果软件 荐

```
    __set_PRIMASK(1);

    /* 硬件初始化 */
    bsp_Init();

    /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
          目中不要使用，因为这个功能比较影响系统实时性。
       2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
    */
    vSetupSysInfoTest();

    /* 创建任务 */
    AppTaskCreate();

   /* 启动调度，开始执行任务 */
   vTaskStartScheduler();

    /*
      如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
      heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
      #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
    */
    while(1);
}
```

◆ **硬件外设初始化**

硬件外设的初始化是在 bsp.c 文件实现：

```
/*
*********************************************************************************************************
*    函 数 名: bsp_Init
*    功能说明: 初始化所有的硬件设备。该函数配置 CPU 寄存器和外设的寄存器并初始化一些全局变量。只需要调用一次
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
void bsp_Init(void)
{
    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        系统时钟缺省配置为 168MHz，如果需要更改，可以修改 system_stm32f4xx.c 文件
    */
    /* 优先级分组设置为 4，可配置 0-15 级抢占式优先级，0 级子优先级，即不存在子优先级。*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

    SystemCoreClockUpdate();    /* 根据 PLL 配置更新系统时钟频率变量 SystemCoreClock */

    bsp_InitUart();    /* 初始化串口 */
    bsp_InitKey();     /* 初始化按键变量 */
    bsp_InitExtIO();   /* FMC 总线上扩展了 32 位输出 IO，操作 LED 等外设必须初始化 */
    bsp_InitLed();     /* 初始 LED 指示灯端口 */
}
```

◆ **FreeRTOS 任务创建：**

```
/*
*********************************************************************************************************
*    函 数 名: AppTaskCreate
```


资料整理 火龙果软件 荐

```
*    功能说明: 创建应用任务
*    形      参: 无
*    返 回 值: 无
*****************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


     xTaskCreate( vTaskLED,           /* 任务函数  */
                  "vTaskLED",         /* 任务名    */
                  512,                /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,               /* 任务参数  */
                  2,                  /* 任务优先级*/
                  &xHandleTaskLED ); /* 任务句柄  */

     xTaskCreate( vTaskMsgPro,            /* 任务函数  */
                  "vTaskMsgPro",          /* 任务名    */
                  512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,                   /* 任务参数  */
                  3,                      /* 任务优先级*/
                  &xHandleTaskMsgPro );  /* 任务句柄  */


     xTaskCreate( vTaskStart,             /* 任务函数  */
                  "vTaskStart",           /* 任务名    */
                  512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                  NULL,                   /* 任务参数  */
                  4,                      /* 任务优先级*/
                  &xHandleTaskStart );   /* 任务句柄  */
}
```

◆ **四个 FreeRTOS 任务的实现：**

```
/*
*****************************************************************************************************
*    函 数 名: vTaskTaskUserIF
*    功能说明: 接口消息处理。
*    形      参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 1  (数值越小优先级越低，这个跟 uCOS 相反)
*****************************************************************************************************
*/
static void vTaskTaskUserIF(void *pvParameters)
{
     uint8_t ucKeyCode;
     uint8_t pcWriteBuffer[500];

    while(1)
    {
         ucKeyCode = bsp_GetKey();

         if (ucKeyCode != KEY_NONE)
```


资料整理 火龙果软件 荐

```
        {
            switch (ucKeyCode)
            {
                /* K1 键按下 打印任务执行情况 */
                case KEY_DOWN_K1:
                    printf("==================================================\r\n");
                    printf("任务名      任务状态 优先级   剩余栈 任务序号\r\n");
                    vTaskList((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);

                    printf("\r\n 任务名       运行计数         使用率\r\n");
                    vTaskGetRunTimeStats((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);
                    break;

                /* K2 键按下 删除任务 vTaskLED */
                case KEY_DOWN_K2:
                    printf("K2 键按下，删除任务 vTaskLED\r\n");
                    if(xHandleTaskLED != NULL)
                    {
                        vTaskDelete(xHandleTaskLED);
                        xHandleTaskLED = NULL;
                    }
                    break;

                /* K3 键按下 重新创建任务 vTaskLED */
                case KEY_DOWN_K3:
                    printf("K3 键按下，重新创建任务 vTaskLED\r\n");
                    if(xHandleTaskLED == NULL)
                    {
                        xTaskCreate(    vTaskLED,           /* 任务函数  */
                                        "vTaskLED",         /* 任务名    */
                                        512,                /* stack 大小，单位 word，也就是 4 字节 */
                                        NULL,               /* 任务参数  */
                                        2,                  /* 任务优先级*/
                                        &xHandleTaskLED );  /* 任务句柄  */
                    }
                    break;

                /* 摇杆 UP 键按下，挂起任务 vTaskLED */
                case JOY_DOWN_U:
                    printf("摇杆 UP 键按下，挂起任务 vTaskLED\r\n");
                    vTaskSuspend(xHandleTaskLED);
                    break;

                /* 摇杆 DOWN 键按下，恢复任务 vTaskLED */
                case JOY_DOWN_D:
                    printf("摇杆 DOWN 键按下，恢复任务 vTaskLED\r\n");
                    vTaskResume(xHandleTaskLED);
                    break;

                /* 其他的键值不处理 */
                default:
                    break;
            }
        }

        vTaskDelay(20);
```


资料整理 火龙果软件 荐

```
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskLED
*    功能说明: LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 2
*********************************************************************************************************
*/
static void vTaskLED(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(2);
        vTaskDelay(200);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskMsgPro
*    功能说明: 消息处理，这里是用作 LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 3
*********************************************************************************************************
*/
static void vTaskMsgPro(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(3);
        vTaskDelay(300);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
         bsp_KeyScan();
        vTaskDelay(10);
    }
}
```


资料整理 火龙果软件 荐

10.14 实验例程说明（中断方式）

10.14.1 STM32F103 开发板实验

配套例子：

V4-304_FreeRTOS 实验_任务恢复（中断方式）

实验目的：

1. 学习 FreeRTOS 的任务恢复（中断方式）。
2. 任务中的任务恢复函数调用的 vTaskResume，而中断里面的任务恢复函数调用 xTaskResumeFromISR，中断中调用的函数是以 FromISR 结尾的，实际应用中切不可混淆。

实验内容：

1. K1 按键按下，串口打印任务执行情况（波特率 115200，数据位 8，奇偶校验位无，停止位 1）。
2. K2 按键按下，删除任务 vTaskLED。
3. K3 按键按下，重新创建任务 vTaskLED。
4. 摇杆 UP 按键按下，挂起任务 vTaskLED。
5. 摇杆 DOWN 键按下，恢复任务 vTaskLED。
6. 摇杆 OK 键按下，启动单次定时器中断，50ms 后在定时器中断将任务 vTaskLED 恢复。
7. 各个任务实现的功能如下：

   vTaskUserIF 任务　　：按键消息处理。

   vTaskLED 任务　　：LED 闪烁。

   vTaskMsgPro 任务　：消息处理，这里是用作 LED 闪烁。

   vTaskStart 任务　　：启动任务，也是最高优先级任务，这里实现按键扫描。

FreeRTOS 的配置：

FreeRTOSConfig.h 文件中的配置如下：

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
#endif

#define configUSE_PREEMPTION          1
#define configUSE_IDLE_HOOK           0
#define configUSE_TICK_HOOK           0
#define configCPU_CLOCK_HZ            ( ( unsigned long ) 72000000 )
#define configTICK_RATE_HZ            ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES          ( 5 )
#define configMINIMAL_STACK_SIZE      ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE         ( ( size_t ) ( 17 * 1024 ) )
#define configMAX_TASK_NAME_LEN       ( 16 )
#define configUSE_TRACE_FACILITY      1
#define configUSE_16_BIT_TICKS        0
```


资料整理 火龙果软件 荐

```
#define configIDLE_SHOULD_YIELD       1

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()     (ulHighFrequencyTimerTicks = 0ul)
#define portGET_RUN_TIME_COUNTER_VALUE()             ulHighFrequencyTimerTicks
//#define portALT_GET_RUN_TIME_COUNTER_VALUE           1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES            0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet          1
#define INCLUDE_uxTaskPriorityGet         1
#define INCLUDE_vTaskDelete               1
#define INCLUDE_vTaskCleanUpResources     0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_xTaskResumeFromISR        1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
     /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
     #define configPRIO_BITS              __NVIC_PRIO_BITS
#else
     #define configPRIO_BITS              4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY             0x0f

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY         0x01
```

几个重要选项说明：

- #define configUSE_PREEMPTION　　1

  使能抢占式调度器

- #define configCPU_CLOCK_HZ　　((unsigned long) 72000000)

  系统主频 72MHz。

- #define configTICK_RATE_HZ　　((TickType_t) 1000)

  系统时钟节拍 1KHz，即 1ms。

- #define configMAX_PRIORITIES　　(5)

  定义可供用户使用的最大优先级数 如果这个定义的是 5 那么用户可以使用的优先级号是 0,1,2,3,4，

资料整理 火龙果软件 荐

不包含 5，对于这一点，初学者要特别的注意。

- #define configTOTAL_HEAP_SIZE　　((size_t) (17 * 1024))

  定义堆大小，FreeRTOS 内核，用户动态内存申请，任务栈等都需要用这个空间。

- #define INCLUDE_vTaskSuspend　　1

  使用 FreeRTOS 的任务挂起函数 vTaskSuspend 和任务恢复函数 vTaskResume 必须配置此宏定义为 1。

- #define INCLUDE_xTaskResumeFromISR　1

  默认情况下，用户只需配置了宏定义 INCLUDE_vTaskSuspend 为 1 就可以使用函数 xTaskResumeFromISR ,因为宏定义 INCLUDE_xTaskResumeFromISR 已经在 FreeRTOS.h 文件里面默认配置为 1。这里我们为说明问题也将这个宏定义写出来了。

- configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY　　0x01

  定义受 FreeRTOS 管理的最高优先级中断。简单的说就是允许用户在这个中断服务程序里面调用 FreeRTOS 的 API 的最高优先级。为了进一步说明这个宏定义的的作用，解释如下：

  - 使用 CM 内核的 MCU，官方强烈建议将 NVIC 的优先级分组配置为全抢占式优先级，全部配置为抢占式优先级的好处就是方便管理。
  - 对于 STM32 来说，设置 NVIC 的优先级分组为 4 时，NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4)就是全部配置为抢占式优先级。又因为 STM32 的优先级设置仅使用 CM 内核 8bit 中的高 4bit，即只能区分 2^4 = 16 种优先级。因此当优先级分组设置为 4 的时候可供用户选择抢占式优先级为 0 到 15，共 16 个优先级，配置为 0 表示最高优先级，配置为 15 表示最低优先级，不存在子优先级。
  - 这里配置 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 为 0x01 表示用户可以在抢占式优先级为 1 到 15 的中断里面调用 FreeRTOS 的 API 函数，抢占式优先级为 0 的中断里面是不允许调用的。

    更多关于这个参数说明请参看第 12 章。

FreeRTOS 任务调试信息（按 K1 按键，串口打印）：

资料整理 火龙果软件 荐

上面截图中打印出来的任务状态字母 B, R, D, S 对应如下含义：

#define tskBLOCKED_CHAR ('B') 任务阻塞

#define tskREADY_CHAR ('R') 任务就绪

#define tskDELETED_CHAR ('D') 任务删除

#define tskSUSPENDED_CHAR ('S') 任务挂起

程序设计：

- **任务栈大小分配：**

vTaskUserIF 任务 ：2048 字节

vTaskLED 任务 ：2048 字节

vTaskMsgPro 任务 ：2048 字节

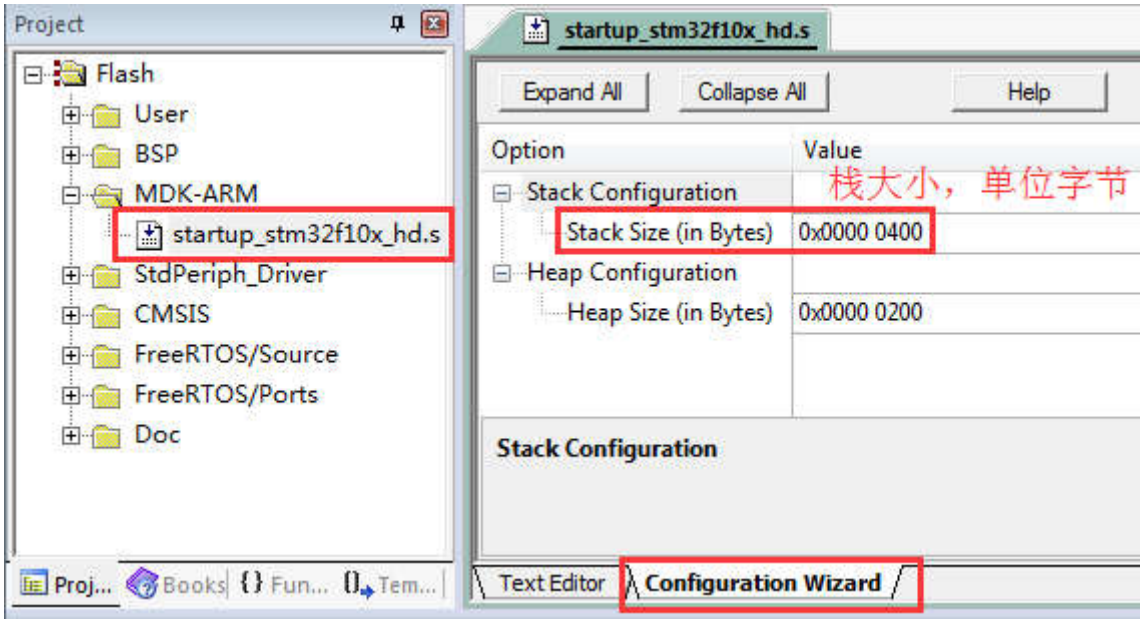
vTaskStart 任务 ：2048 字节

任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE ((size_t) (17 * 1024))

- **系统栈大小分配：**

资料整理 火龙果软件 荐



◆ **FreeROTS 初始化：**

```
/*
*********************************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
int main(void)
{
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是:
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
      2. 保证系统正常启动，不受别的中断影响。
      3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
      在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
      和 cpsie i 是等效的。
    */
    __set_PRIMASK(1);

    /* 硬件初始化 */
    bsp_Init();

    /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
          目中不要使用，因为这个功能比较影响系统实时性。
       2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
    */
    vSetupSysInfoTest();

    /* 创建任务 */
    AppTaskCreate();

   /* 启动调度，开始执行任务 */
   vTaskStartScheduler();
```


资料整理 火龙果软件 荐

```
    /*
      如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
      heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
      #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
    */
    while(1);
}
```

◆ 硬件外设初始化

硬件外设的初始化是在 bsp.c 文件实现：

```
/*
*********************************************************************************************************
*    函 数 名: bsp_Init
*    功能说明: 初始化硬件设备。只需要调用一次。该函数配置 CPU 寄存器和外设的寄存器并初始化一些全局变量。
*              全局变量。
*    形     参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
void bsp_Init(void)
{
    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        系统时钟缺省配置为 72MHz，如果需要更改，可以修改 system_stm32f10x.c 文件
    */

    /* 优先级分组设置为 4，可配置 0-15 级抢占式优先级，0 级子优先级，即不存在子优先级。*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

    bsp_InitUart();    /* 初始化串口 */
    bsp_InitLed();     /* 初始 LED 指示灯端口 */
    bsp_InitKey();     /* 初始化按键 */

    bsp_InitHardTimer(); /* 初始化 TIM2 定时器 */
}
```

◆ FreeRTOS 任务创建：

```
/*
*********************************************************************************************************
*    函 数 名: AppTaskCreate
*    功能说明: 创建应用任务
*    形     参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


    xTaskCreate( vTaskLED,            /* 任务函数  */
```


资料整理 火龙果软件 荐

```
                "vTaskLED",          /* 任务名     */
                512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                NULL,                /* 任务参数   */
                2,                   /* 任务优先级*/
                &xHandleTaskLED ); /* 任务句柄   */

    xTaskCreate( vTaskMsgPro,            /* 任务函数   */
                "vTaskMsgPro",           /* 任务名     */
                512,                     /* 任务栈大小，单位 word，也就是 4 字节 */
                NULL,                    /* 任务参数   */
                3,                       /* 任务优先级*/
                &xHandleTaskMsgPro );  /* 任务句柄   */


    xTaskCreate( vTaskStart,             /* 任务函数   */
                "vTaskStart",            /* 任务名     */
                512,                     /* 任务栈大小，单位 word，也就是 4 字节 */
                NULL,                    /* 任务参数   */
                4,                       /* 任务优先级*/
                &xHandleTaskStart );   /* 任务句柄   */
}
```

◆ **四个 FreeRTOS 任务的实现：**

```
/*
*********************************************************************************************************
*    函 数 名: vTaskTaskUserIF
*    功能说明: 接口消息处理。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 1  (数值越小优先级越低，这个跟 uCOS 相反)
*********************************************************************************************************
*/
static void vTaskTaskUserIF(void *pvParameters)
{
    uint8_t ucKeyCode;
    uint8_t pcWriteBuffer[500];

    while(1)
    {
        ucKeyCode = bsp_GetKey();

        if (ucKeyCode != KEY_NONE)
        {
            switch (ucKeyCode)
            {
                /* K1 键按下 打印任务执行情况 */
                case KEY_DOWN_K1:
                    printf("================================================\r\n");
                    printf("任务名      任务状态 优先级   剩余栈 任务序号\r\n");
                    vTaskList((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);

                    printf("\r\n 任务名       运行计数         使用率\r\n");
                    vTaskGetRunTimeStats((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);
                    break;

                /* K2 键按下 删除任务 vTaskLED */
```


资料整理 火龙果软件 荐

```
                case KEY_DOWN_K2:
                    printf("K2 键按下，删除任务 vTaskLED\r\n");
                    if(xHandleTaskLED != NULL)
                    {
                        vTaskDelete(xHandleTaskLED);
                        xHandleTaskLED = NULL;
                    }
                    break;

                /* K3 键按下 重新创建任务 vTaskLED */
                case KEY_DOWN_K3:
                    printf("K3 键按下，重新创建任务 vTaskLED\r\n");
                    if(xHandleTaskLED == NULL)
                    {
                        xTaskCreate(    vTaskLED,           /* 任务函数  */
                                        "vTaskLED",         /* 任务名    */
                                        512,                /* stack 大小，单位 word，也就是 4 字节 */
                                        NULL,               /* 任务参数  */
                                        2,                  /* 任务优先级*/
                                        &xHandleTaskLED );  /* 任务句柄  */
                    }
                    break;

                /* 摇杆 UP 键按下，挂起任务 vTaskLED */
                case JOY_DOWN_U:
                    printf("摇杆 UP 键按下，挂起任务 vTaskLED\r\n");
                    vTaskSuspend(xHandleTaskLED);
                    break;

                /* 摇杆 DOWN 键按下，恢复任务 vTaskLED */
                case JOY_DOWN_D:
                    printf("摇杆 DOWN 键按下，恢复任务 vTaskLED\r\n");
                    vTaskResume(xHandleTaskLED);
                    break;

                /* 摇杆 OK 键按下，启动单次定时器中断，50ms 后在定时器中断将任务 vTaskLED 恢复 */
                case JOY_DOWN_OK:
                   printf("摇杆 OK 键按下,启动单次定时器中断,50ms 后在定时器中断将任务 vTaskLED 恢复\r\n");
                   bsp_StartHardTimer(1 ,50000, (void *)TIM_CallBack1);
                   break;

                /* 其他的键值不处理 */
                default:
                    break;
            }
        }

        vTaskDelay(20);
    }
}

/*
*****************************************************************************************************
*    函 数 名: vTaskLED
*    功能说明: LED 闪烁
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 2
```


资料整理 火龙果软件 荐

```
*****************************************************************************************************
*/
static void vTaskLED(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(2);
        vTaskDelay(200);
    }
}

/*
*****************************************************************************************************
*    函 数 名: vTaskMsgPro
*    功能说明: 消息处理，这里是用作 LED 闪烁
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 3
*****************************************************************************************************
*/
static void vTaskMsgPro(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(3);
        vTaskDelay(300);
    }
}

/*
*****************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*****************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
         bsp_KeyScan();
        vTaskDelay(10);
    }
}
```

◆ **定时器中断回调函数中将任务从挂起状态恢复：**

定时器中断的初始化和中断函数在 bsp_timer.c 文件中实现，这个不是教程的重点，故不作介绍。 这里主要关心任务恢复函数 xTaskResumeFromISR 在中断服务程序中的使用方法。

```
/*
*****************************************************************************************************
*    函 数 名: TIM_CallBack1
*    功能说明: 定时器中断的回调函数，此函数被 bsp_StartHardTimer 所调用。
*    形     参: 无
*    返 回 值: 无
```


资料整理 火龙果软件 荐

```
*****************************************************************************************************
*/
static void TIM_CallBack1(void)
{
    BaseType_t xYieldRequired;

    /* 恢复挂起任务 */
     xYieldRequired = xTaskResumeFromISR(xHandleTaskLED);

     /* 退出中断后是否需要执行任务切换 */
     if( xYieldRequired == pdTRUE )
     {
         portYIELD_FROM_ISR(xYieldRequired);
     }
}
```

10.14.2 STM32F407 开发板实验

配套例子：

V5-304_FreeRTOS 实验_任务恢复（中断方式）

实验目的：

1. 学习 FreeRTOS 的任务恢复（中断方式）。
2. 任务中的任务恢复函数调用的 vTaskResume，而中断里面的任务恢复函数调用 xTaskResumeFromISR，中断中调用的函数是以 FromISR 结尾的，实际应用中切不可混淆。

实验内容：

1. K1 按键按下，串口打印任务执行情况（波特率 115200，数据位 8，奇偶校验位无，停止位 1）。
2. K2 按键按下，删除任务 vTaskLED。
3. K3 按键按下，重新创建任务 vTaskLED。
4. 摇杆 UP 按键按下，挂起任务 vTaskLED。
5. 摇杆 DOWN 键按下，恢复任务 vTaskLED。
6. 摇杆 OK 键按下，启动单次定时器中断，50ms 后在定时器中断将任务 vTaskLED 恢复。
7. 各个任务实现的功能如下：

   vTaskUserIF 任务　　：按键消息处理。

   vTaskLED 任务　　：LED 闪烁。

   vTaskMsgPro 任务　：消息处理，这里是用作 LED 闪烁。

   vTaskStart 任务　　：启动任务，也是最高优先级任务，这里实现按键扫描。

FreeRTOS 的配置：

FreeRTOSConfig.h 文件中的配置如下：

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
```


资料整理 火龙果软件 荐

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
#endif

#define configUSE_PREEMPTION         1
#define configUSE_IDLE_HOOK          0
#define configUSE_TICK_HOOK          0
#define configCPU_CLOCK_HZ           ( ( unsigned long ) 168000000 )
#define configTICK_RATE_HZ           ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES         ( 5 )
#define configMINIMAL_STACK_SIZE     ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 30 * 1024 ) )
#define configMAX_TASK_NAME_LEN      ( 16 )
#define configUSE_TRACE_FACILITY      1
#define configUSE_16_BIT_TICKS       0
#define configIDLE_SHOULD_YIELD      1

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()     (ulHighFrequencyTimerTicks = 0ul)
#define portGET_RUN_TIME_COUNTER_VALUE()             ulHighFrequencyTimerTicks
//#define portALT_GET_RUN_TIME_COUNTER_VALUE           1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES             0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet          1
#define INCLUDE_uxTaskPriorityGet         1
#define INCLUDE_vTaskDelete               1
#define INCLUDE_vTaskCleanUpResources     0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_xTaskResumeFromISR        1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
     /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
     #define configPRIO_BITS              __NVIC_PRIO_BITS
#else
     #define configPRIO_BITS              4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY             0x0f

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
```


资料整理 火龙果软件 荐

```
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY    0x01
```

几个重要选项说明：

- #define configUSE_PREEMPTION 1

  使能抢占式调度器

- #define configCPU_CLOCK_HZ ((unsigned long) 168000000)

  系统主频 168MHz。

- #define configTICK_RATE_HZ ((TickType_t) 1000)

  系统时钟节拍 1KHz，即 1ms。

- #define configMAX_PRIORITIES (5)

  定义可供用户使用的最大优先级数 如果这个定义的是 5 那么用户可以使用的优先级号是 0,1,2,3,4，不包含 5，对于这一点，初学者要特别的注意。

- #define configTOTAL_HEAP_SIZE ((size_t) (30 * 1024))

  定义堆大小，FreeRTOS 内核，用户动态内存申请，任务栈等都需要用这个空间。

- #define INCLUDE_vTaskSuspend 1

  使用 FreeRTOS 的任务挂起函数 vTaskSuspend 和任务恢复函数 vTaskResume 必须配置此宏定义为 1。

- #define INCLUDE_xTaskResumeFromISR 1

  默认情况下，用户只需配置了宏定义 INCLUDE_vTaskSuspend 为 1 就可以使用函数 xTaskResumeFromISR ,因为宏定义 INCLUDE_xTaskResumeFromISR 已经在 FreeRTOS.h 文件里面默认配置为 1。这里我们为说明问题也将这个宏定义写出来了。

- configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 0x01

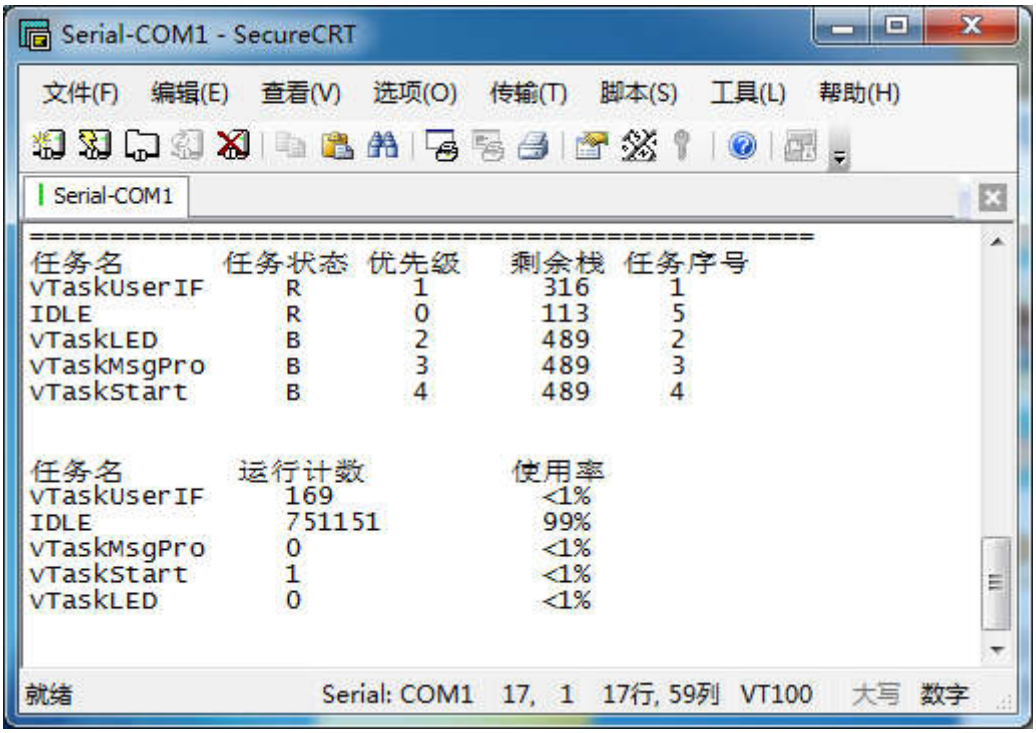
  定义受 FreeRTOS 管理的最高优先级中断。简单的说就是允许用户在这个中断服务程序里面调用 FreeRTOS 的 API 的最高优先级。为了进一步说明这个宏定义的的作用，解释如下：

  - 使用 CM 内核的 MCU，官方强烈建议将 NVIC 的优先级分组配置为全抢占式优先级，全部配置为抢占式优先级的好处就是方便管理。
  - 对于 STM32 来说，设置 NVIC 的优先级分组为 4 时，NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4)就是全部配置为抢占式优先级。又因为 STM32 的优先级设置仅使用 CM 内核 8bit 中的高 4bit，即只能区分 2^4 = 16 种优先级。因此当优先级分组设置为 4 的时候可供用户选择抢占式优先级为 0 到 15，共 16 个优先级，配置为 0 表示最高优先级，配置为 15 表示最低优先级，不存在子优先级。
  - 这里配置 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 为 0x01 表示用户可以在抢占式优先级为 1 到 15 的中断里面调用 FreeRTOS 的 API 函数，抢占式优先级为 0 的中断里面是不允许调用的。

    更多关于这个参数说明请参看第 12 章。

资料整理 火龙果软件 荐

FreeRTOS 任务调试信息（按 K1 按键，串口打印）：



上面截图中打印出来的任务状态字母 B, R, D, S 对应如下含义：

#define tskBLOCKED_CHAR （'B'） 任务阻塞

#define tskREADY_CHAR （'R'） 任务就绪

#define tskDELETED_CHAR （'D'） 任务删除

#define tskSUSPENDED_CHAR （'S'） 任务挂起

程序设计：

- **任务栈大小分配：**

vTaskUserIF 任务 ：2048 字节

vTaskLED 任务 ：2048 字节

vTaskMsgPro 任务 ：2048 字节

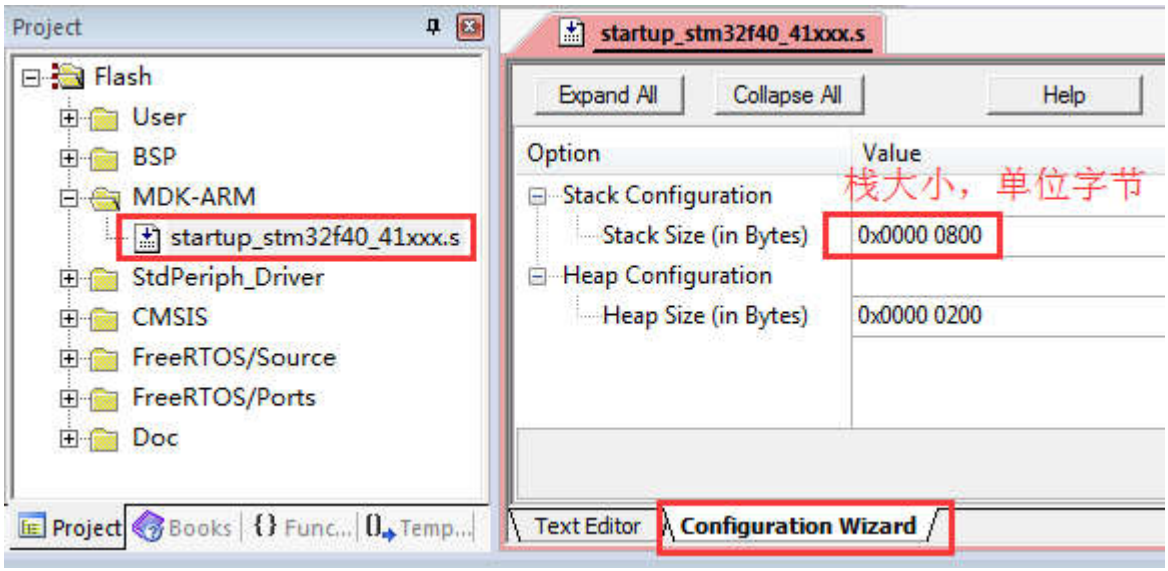
vTaskStart 任务 ：2048 字节

任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE （（size_t）（30 * 1024））

- **系统栈大小分配：**

资料整理 火龙果软件 荐



◆ **FreeROTS 初始化：**

```
/*
*********************************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
int main(void)
{
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是:
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
      2. 保证系统正常启动，不受别的中断影响。
      3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
      在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
      和 cpsie i 是等效的。
    */
    __set_PRIMASK(1);

    /* 硬件初始化 */
    bsp_Init();

    /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
          目中不要使用，因为这个功能比较影响系统实时性。
       2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
    */
    vSetupSysInfoTest();

    /* 创建任务 */
    AppTaskCreate();

   /* 启动调度，开始执行任务 */
   vTaskStartScheduler();

    /*
      如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
```


资料整理 火龙果软件 荐

```
        heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
        #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
    */
    while(1);
}
```

◆ 硬件外设初始化

硬件外设的初始化是在 bsp.c 文件实现：

```
/*
*********************************************************************************************************
*    函 数 名: bsp_Init
*    功能说明: 初始化所有的硬件设备。该函数配置 CPU 寄存器和外设的寄存器并初始化一些全局变量。只需要调用一次
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
void bsp_Init(void)
{
    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        系统时钟缺省配置为 168MHz，如果需要更改，可以修改 system_stm32f4xx.c 文件
    */
    /* 优先级分组设置为 4，可配置 0-15 级抢占式优先级，0 级子优先级，即不存在子优先级。*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

    bsp_InitUart();    /* 初始化串口 */
    bsp_InitKey();     /* 初始化按键变量 */
    bsp_InitLed();     /* 初始 LED 指示灯端口 */

    bsp_InitHardTimer(); /* 初始化 TIM2 定时器 */
}
```

◆ FreeRTOS 任务创建：

```
/*
*********************************************************************************************************
*    函 数 名: AppTaskCreate
*    功能说明: 创建应用任务
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


    xTaskCreate( vTaskLED,            /* 任务函数  */
                 "vTaskLED",          /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 2,                   /* 任务优先级*/
```


资料整理 火龙果软件 荐

```
                &xHandleTaskLED ); /* 任务句柄  */

    xTaskCreate( vTaskMsgPro,            /* 任务函数  */
                 "vTaskMsgPro",          /* 任务名    */
                 512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                   /* 任务参数  */
                 3,                      /* 任务优先级*/
                 &xHandleTaskMsgPro );  /* 任务句柄  */


    xTaskCreate( vTaskStart,             /* 任务函数  */
                 "vTaskStart",           /* 任务名    */
                 512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                   /* 任务参数  */
                 4,                      /* 任务优先级*/
                 &xHandleTaskStart );   /* 任务句柄  */
}
```

◆ **四个 FreeRTOS 任务的实现：**

```
/*
*********************************************************************************************************
*    函 数 名: vTaskTaskUserIF
*    功能说明: 接口消息处理。
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 1  (数值越小优先级越低，这个跟 uCOS 相反)
*********************************************************************************************************
*/
static void vTaskTaskUserIF(void *pvParameters)
{
    uint8_t ucKeyCode;
    uint8_t pcWriteBuffer[500];

    while(1)
    {
        ucKeyCode = bsp_GetKey();

        if (ucKeyCode != KEY_NONE)
        {
            switch (ucKeyCode)
            {
                /* K1 键按下 打印任务执行情况 */
                case KEY_DOWN_K1:
                    printf("================================================\r\n");
                    printf("任务名      任务状态 优先级   剩余栈 任务序号\r\n");
                    vTaskList((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);

                    printf("\r\n 任务名       运行计数         使用率\r\n");
                    vTaskGetRunTimeStats((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);
                    break;

                /* K2 键按下 删除任务 vTaskLED */
                case KEY_DOWN_K2:
                    printf("K2 键按下，删除任务 vTaskLED\r\n");
                    if(xHandleTaskLED != NULL)
                    {
```


资料整理 火龙果软件 荐

```
                        vTaskDelete(xHandleTaskLED);
                        xHandleTaskLED = NULL;
                    }
                    break;

                /* K3 键按下 重新创建任务 vTaskLED */
                case KEY_DOWN_K3:
                    printf("K3 键按下，重新创建任务 vTaskLED\r\n");
                    if(xHandleTaskLED == NULL)
                    {
                        xTaskCreate(    vTaskLED,           /* 任务函数  */
                                        "vTaskLED",         /* 任务名    */
                                        512,                /* stack 大小，单位 word，也就是 4 字节 */
                                        NULL,               /* 任务参数  */
                                        2,                  /* 任务优先级*/
                                        &xHandleTaskLED );  /* 任务句柄  */
                    }
                    break;

                /* 摇杆 UP 键按下，挂起任务 vTaskLED */
                case JOY_DOWN_U:
                    printf("摇杆 UP 键按下，挂起任务 vTaskLED\r\n");
                    vTaskSuspend(xHandleTaskLED);
                    break;

                /* 摇杆 DOWN 键按下，恢复任务 vTaskLED */
                case JOY_DOWN_D:
                    printf("摇杆 DOWN 键按下，恢复任务 vTaskLED\r\n");
                    vTaskResume(xHandleTaskLED);
                    break;

                /* 摇杆 OK 键按下，启动单次定时器中断，50ms 后在定时器中断将任务 vTaskLED 恢复 */
                case JOY_DOWN_OK:
                   printf("摇杆 OK 键按下,启动单次定时器中断,50ms 后在定时器中断将任务 vTaskLED 恢复\r\n");
                   bsp_StartHardTimer(1 ,50000, (void *)TIM_CallBack1);
                   break;

                /* 其他的键值不处理 */
                default:
                    break;
            }
        }

        vTaskDelay(20);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskLED
*    功能说明: LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 2
*********************************************************************************************************
*/
static void vTaskLED(void *pvParameters)
{
```


资料整理 火龙果软件 荐

```
    while(1)
    {
         bsp_LedToggle(2);
        vTaskDelay(200);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskMsgPro
*    功能说明: 消息处理，这里是用作 LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 3
*********************************************************************************************************
*/
static void vTaskMsgPro(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(3);
        vTaskDelay(300);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
         bsp_KeyScan();
        vTaskDelay(10);
    }
}
```

◆ **定时器中断回调函数中将任务从挂起状态恢复：**

定时器中断的初始化和中断函数在 bsp_timer.c 文件中实现，这个不是教程的重点，故不作介绍。 这里主要关心任务恢复函数 xTaskResumeFromISR 在中断服务程序中的使用方法。

```
/*
*********************************************************************************************************
*    函 数 名: TIM_CallBack1
*    功能说明: 定时器中断的回调函数，此函数被 bsp_StartHardTimer 所调用。
*    形    参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void TIM_CallBack1(void)
{
```


资料整理 火龙果软件 荐

```
    BaseType_t xYieldRequired;

    /* 恢复挂起任务 */
    xYieldRequired = xTaskResumeFromISR(xHandleTaskLED);

     /* 退出中断后是否需要执行任务切换 */
    if( xYieldRequired == pdTRUE )
    {
        portYIELD_FROM_ISR(xYieldRequired);
    }
}
```

10.14.3 STM32F429 开发板实验

配套例子：

V6-304_FreeRTOS 实验_任务恢复（中断方式）

实验目的：

1. 学习 FreeRTOS 的任务恢复（中断方式）。
2. 任务中的任务恢复函数调用的 vTaskResume，而中断里面的任务恢复函数调用 xTaskResumeFromISR，中断中调用的函数是以 FromISR 结尾的，实际应用中切不可混淆。

实验内容：

1. K1 按键按下，串口打印任务执行情况（波特率 115200，数据位 8，奇偶校验位无，停止位 1）。
2. K2 按键按下，删除任务 vTaskLED。
3. K3 按键按下，重新创建任务 vTaskLED。
4. 摇杆 UP 按键按下，挂起任务 vTaskLED。
5. 摇杆 DOWN 键按下，恢复任务 vTaskLED。
6. 摇杆 OK 键按下，启动单次定时器中断，50ms 后在定时器中断将任务 vTaskLED 恢复。
7. 各个任务实现的功能如下：

   vTaskUserIF 任务　　：按键消息处理。

   vTaskLED 任务　　：LED 闪烁。

   vTaskMsgPro 任务　：消息处理，这里是用作 LED 闪烁。

   vTaskStart 任务　　：启动任务，也是最高优先级任务，这里实现按键扫描。

FreeRTOS 的配置：

FreeRTOSConfig.h 文件中的配置如下：

```
/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
 #include <stdint.h>
 extern volatile uint32_t ulHighFrequencyTimerTicks;
#endif

#define configUSE_PREEMPTION          1
#define configUSE_IDLE_HOOK           0
```


资料整理 火龙果软件 荐

```
#define configUSE_TICK_HOOK          0
#define configCPU_CLOCK_HZ           ( ( unsigned long ) 168000000 )
#define configTICK_RATE_HZ           ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES         ( 5 )
#define configMINIMAL_STACK_SIZE     ( ( unsigned short ) 128 )
#define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 30 * 1024 ) )
#define configMAX_TASK_NAME_LEN      ( 16 )
#define configUSE_TRACE_FACILITY      1
#define configUSE_16_BIT_TICKS       0
#define configIDLE_SHOULD_YIELD      1

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()     (ulHighFrequencyTimerTicks = 0ul)
#define portGET_RUN_TIME_COUNTER_VALUE()             ulHighFrequencyTimerTicks
//#define portALT_GET_RUN_TIME_COUNTER_VALUE           1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES            0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet          1
#define INCLUDE_uxTaskPriorityGet         1
#define INCLUDE_vTaskDelete               1
#define INCLUDE_vTaskCleanUpResources     0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_xTaskResumeFromISR        1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
     /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
     #define configPRIO_BITS              __NVIC_PRIO_BITS
#else
     #define configPRIO_BITS              4        /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY             0x0f

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY         0x01
```

几个重要选项说明：

- #define configUSE_PREEMPTION　　1

  使能抢占式调度器

- #define configCPU_CLOCK_HZ　　((unsigned long) 168000000)

资料整理 火龙果软件 荐

系统主频 168MHz。

- #define configTICK_RATE_HZ　　((TickType_t) 1000)

  系统时钟节拍 1KHz，即 1ms。

- #define configMAX_PRIORITIES　　(5)

  定义可供用户使用的最大优先级数 如果这个定义的是 5 那么用户可以使用的优先级号是 0,1,2,3,4，不包含 5，对于这一点，初学者要特别的注意。

- #define configTOTAL_HEAP_SIZE　　((size_t) (30 * 1024))

  定义堆大小，FreeRTOS 内核，用户动态内存申请，任务栈等都需要用这个空间。

- #define INCLUDE_vTaskSuspend　　1

  使用 FreeRTOS 的任务挂起函数 vTaskSuspend 和任务恢复函数 vTaskResume 必须配置此宏定义为 1。

- #define INCLUDE_xTaskResumeFromISR　1

  默认情况下，用户只需配置了宏定义 INCLUDE_vTaskSuspend 为 1 就可以使用函数 xTaskResumeFromISR ,因为宏定义 INCLUDE_xTaskResumeFromISR 已经在 FreeRTOS.h 文件里面默认配置为 1。这里我们为说明问题也将这个宏定义写出来了。

- configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY　　0x01

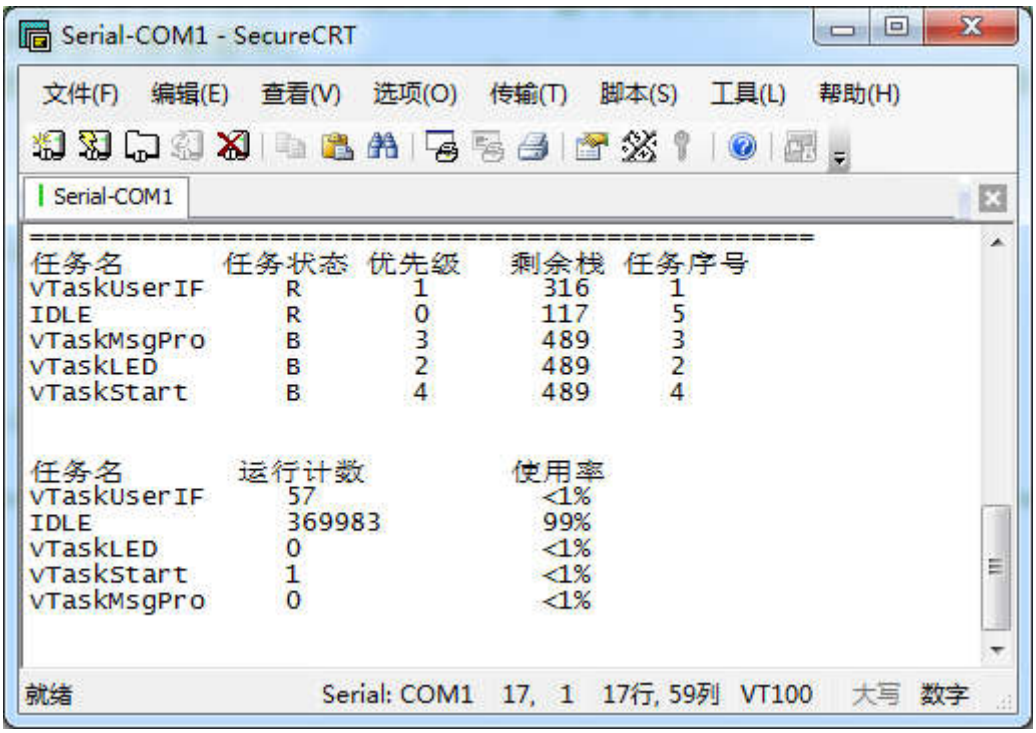
  定义受 FreeRTOS 管理的最高优先级中断。简单的说就是允许用户在这个中断服务程序里面调用 FreeRTOS 的 API 的最高优先级。为了进一步说明这个宏定义的的作用，解释如下：

  - 使用 CM 内核的 MCU，官方强烈建议将 NVIC 的优先级分组配置为全抢占式优先级，全部配置为抢占式优先级的好处就是方便管理。
  - 对于 STM32 来说，设置 NVIC 的优先级分组为 4 时，NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4)就是全部配置为抢占式优先级。又因为 STM32 的优先级设置仅使用 CM 内核 8bit 中的高 4bit，即只能区分 2^4 = 16 种优先级。因此当优先级分组设置为 4 的时候可供用户选择抢占式优先级为 0 到 15，共 16 个优先级，配置为 0 表示最高优先级，配置为 15 表示最低优先级，不存在子优先级。
  - 这里配置 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 为 0x01 表示用户可以在抢占式优先级为 1 到 15 的中断里面调用 FreeRTOS 的 API 函数，抢占式优先级为 0 的中断里面是不允许调用的。

    更多关于这个参数说明请参看第 12 章。

FreeRTOS 任务调试信息（按 K1 按键，串口打印）：

资料整理 火龙果软件 荐



上面截图中打印出来的任务状态字母 B, R, D, S 对应如下含义：

#define tskBLOCKED_CHAR ('B') 任务阻塞

#define tskREADY_CHAR ('R') 任务就绪

#define tskDELETED_CHAR ('D') 任务删除

#define tskSUSPENDED_CHAR ('S') 任务挂起

程序设计：

- **任务栈大小分配：**

vTaskUserIF 任务 ：2048 字节

vTaskLED 任务 ：2048 字节

vTaskMsgPro 任务 ：2048 字节

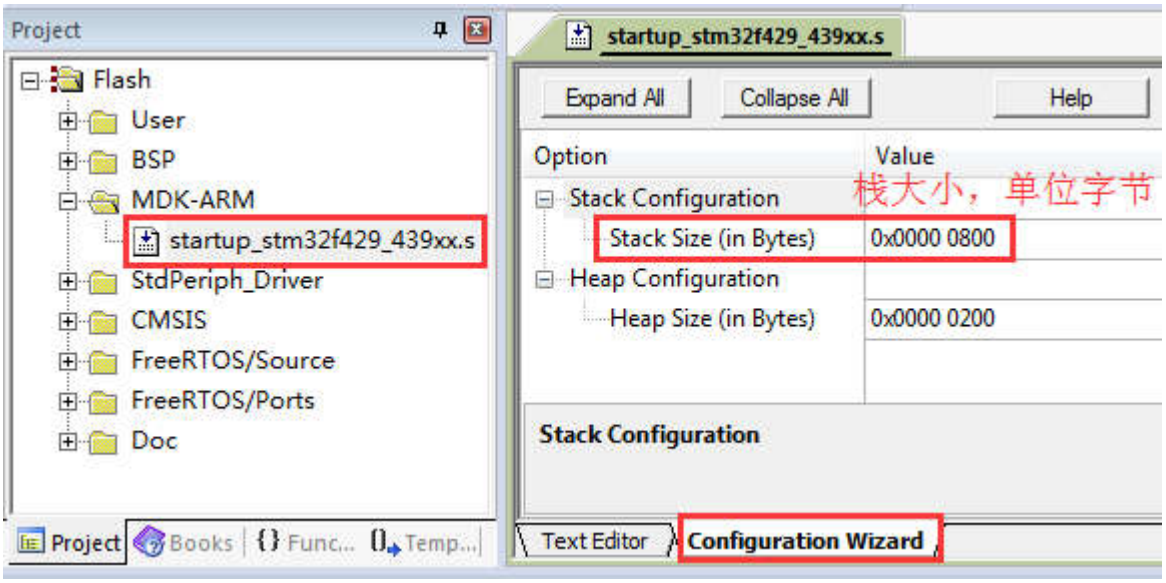
vTaskStart 任务 ：2048 字节

任务栈空间是在任务创建的时候从 FreeRTOSConfig.h 文件中定义的 heap 空间中申请的

#define configTOTAL_HEAP_SIZE ((size_t) (30 * 1024))

- **系统栈大小分配：**

资料整理 火龙果软件 荐



◆ **FreeROTS 初始化：**

```
/*
*********************************************************************************************************
*    函 数 名: main
*    功能说明: 标准 c 程序入口。
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
int main(void)
{
    /*
      在启动调度前，为了防止初始化 STM32 外设时有中断服务程序执行，这里禁止全局中断(除了 NMI 和 HardFault)。
      这样做的好处是:
      1. 防止执行的中断服务程序中有 FreeRTOS 的 API 函数。
      2. 保证系统正常启动，不受别的中断影响。
      3. 关于是否关闭全局中断，大家根据自己的实际情况设置即可。
      在移植文件 port.c 中的函数 prvStartFirstTask 中会重新开启全局中断。通过指令 cpsie i 开启，__set_PRIMASK(1)
      和 cpsie i 是等效的。
    */
    __set_PRIMASK(1);

    /* 硬件初始化 */
    bsp_Init();

    /* 1. 初始化一个定时器中断，精度高于滴答定时器中断，这样才可以获得准确的系统信息 仅供调试目的，实际项
          目中不要使用，因为这个功能比较影响系统实时性。
       2. 为了正确获取 FreeRTOS 的调试信息，可以考虑将上面的关闭中断指令__set_PRIMASK(1); 注释掉。
    */
    vSetupSysInfoTest();

    /* 创建任务 */
    AppTaskCreate();

   /* 启动调度，开始执行任务 */
   vTaskStartScheduler();

    /*
      如果系统正常启动是不会运行到这里的，运行到这里极有可能是用于定时器任务或者空闲任务的
```


资料整理 火龙果软件 荐

```
        heap 空间不足造成创建失败，此要加大 FreeRTOSConfig.h 文件中定义的 heap 大小：
        #define configTOTAL_HEAP_SIZE        ( ( size_t ) ( 17 * 1024 ) )
    */
    while(1);
}
```

◆ 硬件外设初始化

硬件外设的初始化是在 bsp.c 文件实现：

```
/*
*********************************************************************************************************
*    函 数 名: bsp_Init
*    功能说明: 初始化所有的硬件设备。该函数配置 CPU 寄存器和外设的寄存器并初始化一些全局变量。只需要调用一次
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
void bsp_Init(void)
{
    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        系统时钟缺省配置为 168MHz，如果需要更改，可以修改 system_stm32f4xx.c 文件
    */
    /* 优先级分组设置为 4，可配置 0-15 级抢占式优先级，0 级子优先级，即不存在子优先级。*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

    SystemCoreClockUpdate();    /* 根据 PLL 配置更新系统时钟频率变量 SystemCoreClock */

    bsp_InitUart();    /* 初始化串口 */
    bsp_InitKey();     /* 初始化按键变量 */
    bsp_InitExtIO();   /* FMC 总线上扩展了 32 位输出 IO，操作 LED 等外设必须初始化 */
    bsp_InitLed();     /* 初始 LED 指示灯端口 */

    bsp_InitHardTimer(); /* 初始化 TIM2 定时器 */
}
```

◆ FreeRTOS 任务创建：

```
/*
*********************************************************************************************************
*    函 数 名: AppTaskCreate
*    功能说明: 创建应用任务
*    形      参: 无
*    返 回 值: 无
*********************************************************************************************************
*/
static void AppTaskCreate (void)
{
    xTaskCreate( vTaskTaskUserIF,     /* 任务函数  */
                 "vTaskUserIF",       /* 任务名    */
                 512,                 /* 任务栈大小，单位 word，也就是 4 字节 */
                 NULL,                /* 任务参数  */
                 1,                   /* 任务优先级*/
                 &xHandleTaskUserIF );  /* 任务句柄  */


    xTaskCreate( vTaskLED,            /* 任务函数  */
                 "vTaskLED",          /* 任务名    */
```


资料整理 火龙果软件 荐

```
                512,                /* 任务栈大小，单位 word，也就是 4 字节 */
                NULL,               /* 任务参数  */
                2,                  /* 任务优先级*/
                &xHandleTaskLED ); /* 任务句柄  */

    xTaskCreate( vTaskMsgPro,           /* 任务函数  */
                "vTaskMsgPro",          /* 任务名    */
                512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                NULL,                   /* 任务参数  */
                3,                      /* 任务优先级*/
                &xHandleTaskMsgPro );  /* 任务句柄  */


    xTaskCreate( vTaskStart,            /* 任务函数  */
                "vTaskStart",           /* 任务名    */
                512,                    /* 任务栈大小，单位 word，也就是 4 字节 */
                NULL,                   /* 任务参数  */
                4,                      /* 任务优先级*/
                &xHandleTaskStart );   /* 任务句柄  */
}
```

◆ **四个 FreeRTOS 任务的实现：**

```
/*
*********************************************************************************************************
*    函 数 名: vTaskTaskUserIF
*    功能说明: 接口消息处理。
*    形      参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 1  (数值越小优先级越低，这个跟 uCOS 相反)
*********************************************************************************************************
*/
static void vTaskTaskUserIF(void *pvParameters)
{
    uint8_t ucKeyCode;
    uint8_t pcWriteBuffer[500];

    while(1)
    {
        ucKeyCode = bsp_GetKey();

        if (ucKeyCode != KEY_NONE)
        {
            switch (ucKeyCode)
            {
                /* K1 键按下 打印任务执行情况 */
                case KEY_DOWN_K1:
                    printf("================================================\r\n");
                    printf("任务名      任务状态 优先级   剩余栈 任务序号\r\n");
                    vTaskList((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);

                    printf("\r\n 任务名       运行计数         使用率\r\n");
                    vTaskGetRunTimeStats((char *)&pcWriteBuffer);
                    printf("%s\r\n", pcWriteBuffer);
                    break;

                /* K2 键按下 删除任务 vTaskLED */
                case KEY_DOWN_K2:
```


资料整理 火龙果软件 荐

```
                    printf("K2 键按下，删除任务 vTaskLED\r\n");
                    if(xHandleTaskLED != NULL)
                    {
                        vTaskDelete(xHandleTaskLED);
                        xHandleTaskLED = NULL;
                    }
                    break;

                /* K3 键按下 重新创建任务 vTaskLED */
                case KEY_DOWN_K3:
                    printf("K3 键按下，重新创建任务 vTaskLED\r\n");
                    if(xHandleTaskLED == NULL)
                    {
                        xTaskCreate(    vTaskLED,           /* 任务函数  */
                                        "vTaskLED",         /* 任务名    */
                                        512,                /* stack 大小，单位 word，也就是 4 字节 */
                                        NULL,               /* 任务参数  */
                                        2,                  /* 任务优先级*/
                                        &xHandleTaskLED );  /* 任务句柄  */
                    }
                    break;

                /* 摇杆 UP 键按下，挂起任务 vTaskLED */
                case JOY_DOWN_U:
                    printf("摇杆 UP 键按下，挂起任务 vTaskLED\r\n");
                    vTaskSuspend(xHandleTaskLED);
                    break;

                /* 摇杆 DOWN 键按下，恢复任务 vTaskLED */
                case JOY_DOWN_D:
                    printf("摇杆 DOWN 键按下，恢复任务 vTaskLED\r\n");
                    vTaskResume(xHandleTaskLED);
                    break;

                /* 摇杆 OK 键按下，启动单次定时器中断，50ms 后在定时器中断将任务 vTaskLED 恢复 */
                case JOY_DOWN_OK:
                   printf("摇杆 OK 键按下,启动单次定时器中断,50ms 后在定时器中断将任务 vTaskLED 恢复\r\n");
                   bsp_StartHardTimer(1 ,50000, (void *)TIM_CallBack1);
                   break;

                /* 其他的键值不处理 */
                default:
                    break;
            }
        }

        vTaskDelay(20);
    }
}

/*
*****************************************************************************************************
*    函 数 名: vTaskLED
*    功能说明: LED 闪烁
*    形     参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 2
*****************************************************************************************************
```


资料整理 火龙果软件 荐

```
*/
static void vTaskLED(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(2);
        vTaskDelay(200);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskMsgPro
*    功能说明: 消息处理，这里是用作 LED 闪烁
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 3
*********************************************************************************************************
*/
static void vTaskMsgPro(void *pvParameters)
{
    while(1)
    {
         bsp_LedToggle(3);
        vTaskDelay(300);
    }
}

/*
*********************************************************************************************************
*    函 数 名: vTaskStart
*    功能说明: 启动任务，也就是最高优先级任务，这里用作按键扫描。
*    形    参: pvParameters 是在创建该任务时传递的形参
*    返 回 值: 无
*   优 先 级: 4
*********************************************************************************************************
*/
static void vTaskStart(void *pvParameters)
{
    while(1)
    {
         /* 按键扫描 */
         bsp_KeyScan();
        vTaskDelay(10);
    }
}
```

◆ **定时器中断回调函数中将任务从挂起状态恢复：**

定时器中断的初始化和中断函数在 bsp_timer.c 文件中实现，这个不是教程的重点，故不作介绍。 这里主要关心任务恢复函数 xTaskResumeFromISR 在中断服务程序中的使用方法。

```
/*
*********************************************************************************************************
*    函 数 名: TIM_CallBack1
*    功能说明: 定时器中断的回调函数，此函数被 bsp_StartHardTimer 所调用。
*    形    参: 无
*    返 回 值: 无
*********************************************************************************************************
```


资料整理 火龙果软件 荐

```
*/
static void TIM_CallBack1(void)
{
    BaseType_t xYieldRequired;

    /* 恢复挂起任务 */
     xYieldRequired = xTaskResumeFromISR(xHandleTaskLED);

      /* 退出中断后是否需要执行任务切换 */
     if( xYieldRequired == pdTRUE )
     {
         portYIELD_FROM_ISR(xYieldRequired);
     }
}
```

10.15 总结

本章节主要为大家讲解了 FreeRTOS 的任务管理，此章节比较重要，望初学者用心掌握。

资料整理 火龙果软件 荐