title: 从0开始学FreeRTOS-(创建任务)-2 top: false cover: false toc: true mathjax: false date: 2019-08-31 19:53:46 password: summary: tags:

- FreeRTOS
- RTOS
- 操作系统 categories: 操作系统

补充

开始今天的内容之前，先补充一下上篇文章从单片机到操作系统-1的一点点遗漏的知识点。

```
BaseType_t xTaskCreate(       TaskFunction_t pvTaskCode,
                              const char * const pcName,
                              uint16_t usStackDepth,
                              void *pvParameters,
                              UBaseType_t uxPriority,
                              TaskHandle_t *pvCreatedTask
                          );
```

创建任务中的堆栈大小问题，在task.h中有这样子的描述：

```
/**
* @param usStackDepth The size of the task stack specified as the number of
variables the stack * can hold - not the number of bytes.  For example, if the
stack is 16 bits wide and
* usStackDepth is defined as 100, 200 byteswill be allocated for stack storage.
*/
```

当任务创建时，内核会分为每个任务分配属于任务自己的唯一堆栈。usStackDepth 值用于告诉内核为它应该分配多大的栈空间。

这个值指定的是栈空间可以保存多少个字(word)，而不是多少个字节(byte)。

文档也有说明，如果是16位宽度的话，假如usStackDepth = 100；那么就是200个字节（byte）。

当然，我用的是stm32，32位宽度的， usStackDepth=100；那么就是400个字节（byte）。

好啦，补充完毕。下面正式开始我们今天的主题。

我自己学的是应用层的东西，很多底层的东西我也不懂，水平有限，出错了还请多多包涵。

其实我自己写文章的时候也去跟着火哥的书看着底层的东西啦，但是本身自己也是不懂，不敢乱写。所以，这个《从单片机到操作系统》系列的文章，我会讲一点底层，更多的是应用层，主要是用的方面。

按照一般的写代码的习惯，在main函数里面各类初始化完毕了，并且创建任务成功了，那么，可以开启任务调度了。

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);//设置系统中断优先级分组4
    Delay_Init();                       //延时函数初始化
    Uart_Init(115200);                  //初始化串口
    LED_Init();                     //初始化LED
    KEY_Init();
    //创建开始任务
    xTaskCreate((TaskFunction_t )start_task,            //任务函数
                (const char*    )"start_task",          //任务名称
                (uint16_t       )START_STK_SIZE,        //任务堆栈大小
                (void*          )NULL,                  //传递给任务函数的参数
                (UBaseType_t    )START_TASK_PRIO,       //任务优先级
                (TaskHandle_t*  )&StartTask_Handler);   //任务句柄
    vTaskStartScheduler();          //开启任务调度
}
```

来大概看看分析一下创建任务的过程，虽然说会用就行，但是也是要知道了解一下的。

注意：下面说的创建任务均为xTaskCreate（动态创建）而非静态创建。

```
pxStack = ( StackType_t * ) pvPortMalloc( ( ( ( size_t ) usStackDepth ) * sizeof(
StackType_t ) ) );
 /*lint !e961 MISRA exception as the casts are only redundant for some ports. */
            if( pxStack != NULL )
            {
                /* Allocate space for the TCB. */
                pxNewTCB = ( TCB_t * ) pvPortMalloc( sizeof( TCB_t ) );
                /*lint !e961 MISRA exception as the casts are only redundant for
some paths. */
                if( pxNewTCB != NULL )
                {
                   /* Store the stack location in the TCB. */
                   pxNewTCB->pxStack = pxStack;
               }
               else
               {
                   /* The stack cannot be used as the TCB was not created.  Free
                   it again. */
                   vPortFree( pxStack );
               }
            }
            else
            {
                pxNewTCB = NULL;
```

```
            }
        }
```

首先是利用pvPortMalloc给任务的堆栈分配空间，if(pxStack != NULL)如果内存申请成功，就接着给任务控制块申请内存。pxNewTCB = (TCB_t *) pvPortMalloc(sizeof(TCB_t));同样是使用pvPortMalloc();如果任务控制块内存申请失败则释放 之前已经申请成功的任务堆栈的内存vPortFree(pxStack);

然后就初始化任务相关的东西，并且将新初始化的任务控制块添加到列表中prvAddNewTaskToReadyList(pxNewTCB);

最后返回任务的状态，如果是成功了就是pdPASS，假如失败了就是返回errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;

```
prvInitialiseNewTask(    pxTaskCode,
                         pcName,
                         ( uint32_t ) usStackDepth,
                         pvParameters,
                         uxPriority,
                         pxCreatedTask,
                         pxNewTCB,
                         NULL );
            prvAddNewTaskToReadyList( pxNewTCB );
            xReturn = pdPASS;
        }
        else
        {
            xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
        }
        return xReturn;
    }
// 相关宏定义
#define pdPASS            ( pdTRUE )
#define pdTRUE            ( ( BaseType_t ) 1 )
/* FreeRTOS error definitions. */
#define errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY    ( -1 )
```

具体的static void prvInitialiseNewTask()实现请参考FreeRTOS的tasks.c文件的767行代码。具体的static void prvAddNewTaskToReadyList(TCB_t *pxNewTCB)实现请参考FreeRTOS的tasks.c文件的963行代码。

因为这些是tasks.c中的静态的函数，仅供xTaskCreate创建任务内部调用的，我们无需理会这些函数的实现过程，当然如果需要请自行了解。

创建完任务就开启任务调度了：

```
vTaskStartScheduler();          //开启任务调度
```

在任务调度里面，会创建一个空闲任务（我们将的都是动态创建任务，静态创建其实一样的）

```
xReturn = xTaskCreate(    prvIdleTask,
                          "IDLE", configMINIMAL_STACK_SIZE,
                          ( void * ) NULL,
                          ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ),
                          &xIdleTaskHandle );
/*lint !e961 MISRA exception, justified as it is not a redundant explicit cast to
all supported compilers. */
    }
相关宏定义：
#define tskIDLE_PRIORITY            ( ( UBaseType_t ) 0U )
#ifndef portPRIVILEGE_BIT
    #define portPRIVILEGE_BIT ( ( UBaseType_t ) 0x00 )
#endif
#define configUSE_TIMERS                        1
 //为1时启用软件定时器
```

从上面的代码我们可以看出，空闲任务的优先级是tskIDLE_PRIORITY为0，也就是说空闲任务的优先级最低。当CPU没事干的时候才执行空闲任务，以待随时切换优先级更高的任务。

如果使用了软件定时器的话，我们还需要创建定时器任务，创建的函数是：

```
#if ( configUSE_TIMERS == 1 )
    BaseType_t xTimerCreateTimerTask( void )
```

然后还要把中断关一下

```
portDISABLE_INTERRUPTS();
```

至于为什么关中断，也有说明：

```
/* Interrupts are turned off here, toensure a tick does not occur
before or during the call toxPortStartScheduler().  The stacks of
the created tasks contain a status wordwith interrupts switched on
so interrupts will automatically getre-enabled when the first task
starts to run. */
```

中断在这里被关闭，以确保不会发生滴答在调用xPortStartScheduler（）之前或期间。堆栈创建的任务包含一个打开中断的状态字因此中断将在第一个任务时自动重新启用开始运行。

那么如何打开中断呢？？？？这是个很重要的问题

别担心，我们在SVC中断服务函数里面就会打开中断的

看代码：

```
__asm void vPortSVCHandler( void )
{
        PRESERVE8
        ldr    r3, =pxCurrentTCB  /* Restore the context. */
        ldrr1, [r3]                          /* UsepxCurrentTCBConst to get the
pxCurrentTCB address. */
        ldrr0, [r1]                          /* Thefirst item in pxCurrentTCB
is the task top of stack. */
        ldmiar0!, {r4-r11}             /* Pop theregisters that are not
automatically saved on exception entry and the criticalnesting count. */
        msrpsp, r0                                    /*Restore the task stack
pointer. */
        isb
        movr0, #0
        msr  basepri, r0
        orrr14, #0xd
        bxr14
}
```

```
msr  basepri, r0
```

就是它把中断打开的。看不懂没所谓，我也不懂汇编，看得懂知道就好啦。

```
xSchedulerRunning = pdTRUE;
```

任务调度开始运行

```
/* If configGENERATE_RUN_TIME_STATS isdefined then the following
macro must be defined to configure thetimer/counter used to generate
the run time counter time base. */
portCONFIGURE_TIMER_FOR_RUN_TIME_STATS();
```

如果`configGENERATE_RUN_TIME_STATS`使用时间统计功能，这个宏为`1`，那么用户必须实现一个宏`portCONFIGURE_TIMER_FOR_RUN_TIME_STATS();`用来配置一个定时器或者计数器。

来到我们的重点了，开启任务调度，那么任务到这了就不会返回了。

```
if( xPortStartScheduler() != pdFALSE )
{
   /*Should not reach here as if the scheduler is running the
    functionwill not return. */
 }
```

然后就能开启第一个任务了，感觉好难是吧，我一开始也是觉得的，但是写了这篇文章，觉得还行吧，也不算太难，可能也是在查看代码跟别人的书籍吧，写东西其实还是蛮好的，能加深理解，写过文章的人就知道，懂了不一定能写出来，所以，我还是很希望朋友们能投稿的。杰杰随时欢迎。。。

开始任务就按照套路模板添加自己的代码就好啦，很简单的。

先创建任务：

```
 xTaskCreate((TaskFunction_t )led0_task,
             (const char*    )"led0_task",
             (uint16_t       )LED0_STK_SIZE,
             (void*          )NULL,
             (UBaseType_t    )LED0_TASK_PRIO,
             (TaskHandle_t*  )&LED0Task_Handler);
   //创建LED1任务
  xTaskCreate((TaskFunction_t )led1_task,
             (const char*    )"led1_task",
             (uint16_t       )LED1_STK_SIZE,
             (void*          )NULL,
             (UBaseType_t    )LED1_TASK_PRIO,
             (TaskHandle_t*  )&LED1Task_Handler);
```

创建完任务就开启任务调度：

```
1vTaskStartScheduler();          //开启任务调度
```

然后具体实现任务函数：

```
//LED0任务函数
void led0_task(void *pvParameters)
{
   while(1)
    {
       LED0=~LED0;
       vTaskDelay(500);
    }
}

//LED1任务函数
void led1_task(void *pvParameters)
{
   while(1)
    {
       LED1=0;
       vTaskDelay(200);
       LED1=1;
```

```
        vTaskDelay(800);
    }
}
```

好啦，今天的介绍到这了为止，后面还会持续更新，敬请期待哦~

欢迎大家一起来讨论操作系统的知识

我们的群号是：783234154

喜欢就关注我吧！

相关代码可以在公众号后台获取。