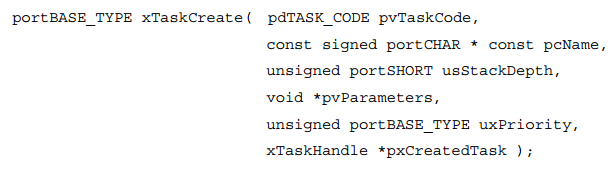
FreeRTOS学习笔记

使用注意：FreeRTOS与ucOS相比，需要修改启动文件。

1. 任务创建

包含FreeRTOS.h //这个是必须要包含的// 和task.h。

调用API函数：xTaskCreate



pvTaskCode：函数指针(函数名)。

pcName：描述性任务名，只是用于辅助调试。可以通过

FreeRTOSConfig.h第100行的

configMAX\_TASK\_NAME\_LEN来改变描述名长度。

usStackDepth：任务栈大小。建议为64的整数倍。

pvParameters：任务参数，void \* 类型。不用写NULL。

uxPriority：任务优先级。值越大优先级越高。同等优先级时，

按照os heart进行时间片轮转。

xTaskHandle：任务句柄。改变该任务优先级、删除任务等时才会

用到。不用写NULL。

1. 开始多任务

vTaskStartScheduler()

1. 如何多任务切换

调用API函数：vTaskDelay(unsigned int time)

Time为多少个os heart。

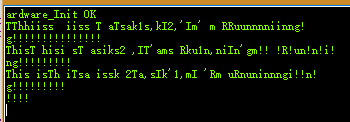
1. 时间片轮转例子

创建两个任务

xTaskCreate(USART1\_Task, "USART1", 256, (void \*)str1, 3, NULL);

xTaskCreate(USART2\_Task, "USART2", 256, (void \*)str2, 3, NULL);

相同优先级，波特率可以设置低一些。

然后就会看到

这就是时间片轮转。

1. 系统心跳频率

在FreeRTOSConfig.h第97行的configTICK\_RATE\_HZ，源码使用1000HZ，就是1ms，如果为200就是200HZ，5ms。

相关宏：portTICK\_RATE\_MS。

1. vTaskDelayUntil

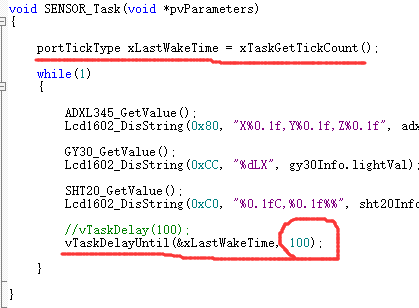
和vTaskDelay不同，这个函数不包括函数运行时间，而

vTaskDelayUntil包括。

比如，一个函数A执行时间为2ms，

使用vTaskDelay(10)，则再次执行这个函数最少需要等待12ms。

使用vTaskDelayUntil(10)，则这2ms包括在内，A以固定10ms周期运行。

使用方法：

红圈的地方就是固定心跳周期。

1. 空闲任务钩子函数

void vApplicationIdleHook(void)

{}

需使能FreeRTOSConfig.h中第126行的configUSE\_IDLE\_HOOK。

1. 消息队列---Queue

包含：#include "queue.h"

声明：QueueHandle\_t qMsg;

创建：qMsg = xQueueCreate(4, 2); //4个队列，每个队列2个成员

发送：xQueueSendToFront(qMsg, &Value, 1)//最后是超时时间

xQueueSendToBack(qMsg, &Value, 1)//最后是超时时间

xQueueSend(qMsg, &Value, 1)//最后是超时时间

成功返回pdPASS；失败返回errQUEUE\_FULL。

注：中断中的就是在函数名字后加上”FromISR”。

接收：xQueueReceive(qMsg, &value, 1)

xQueuePeek(qMsg, &value, 1) //不会删除读取的数据

成功返回pdPASS；失败返回errQUEUE\_EMPTY

注：中断中的就是在函数名字后加上”FromISR”。

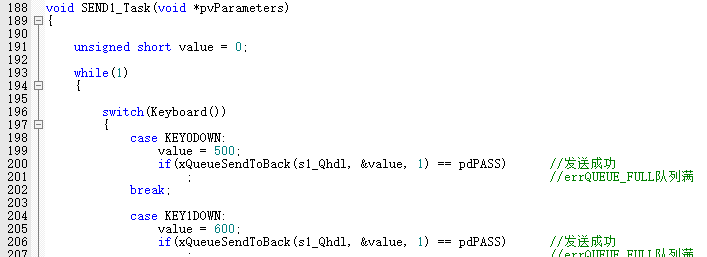
删除：vQueueDelete(qMsg);

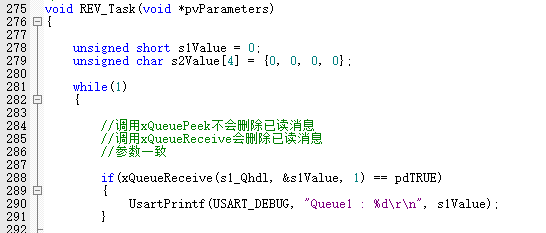
查询队列未读数据个数：

unsigned portBASE\_TYPE count = 0;

count = uxQueueMessagesWaiting(qMsg);

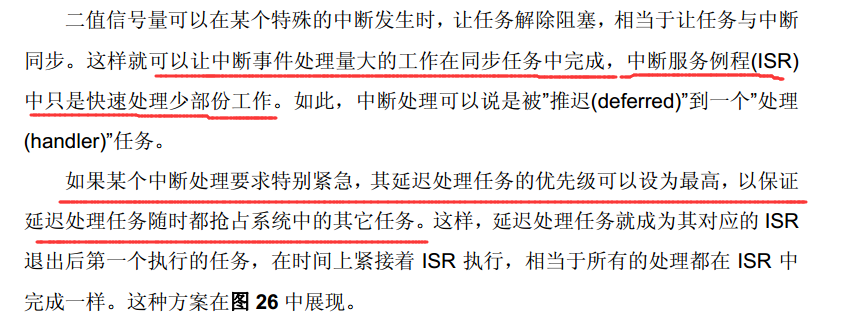
例程：





按键发送队列消息，接收任务接收到就显示。按键是两个，这里没有截出来。

1. 二值信号量



包含：#include "semphr.h"

声明：SemaphoreHandle\_t sSem;

创建：sSem = xSemaphoreCreateBinary(); //创建二值信号量(推荐)

vSemaphoreCreateBinary(sSem); //创建二值信号量

Give：portBASE\_TYPE xHigherPriorityTaskWoken = pdFALSE;

xSemaphoreGiveFromISR(sSem, &xHigherPriorityTaskWoken);

if(xHigherPriorityTaskWoken == pdTRUE)

；

-----中断方法

xSemaphoreGive(sSem);

-----普通方法

Take：portBASE\_TYPE xHigherPriorityTaskWoken = pdFALSE;

xSemaphoreGiveFromISR(sSem, &xHigherPriorityTaskWoken);

if(xHigherPriorityTaskWoken == pdTRUE)

；

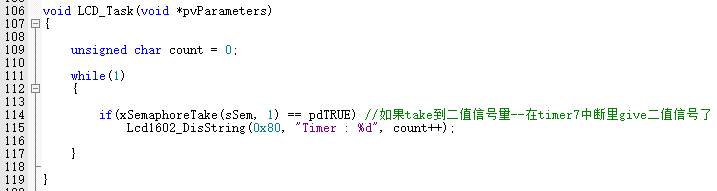
-----中断方法

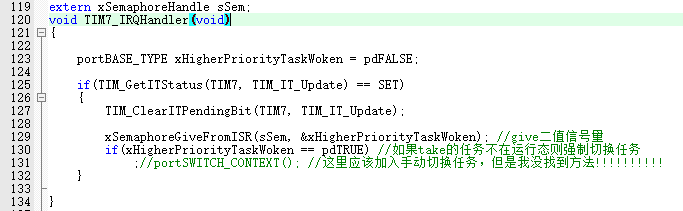
if(xSemaphoreTake(sSem, 1) == pdTRUE)

-----普通方法

删除：vSemaphoreDelete(sSem);

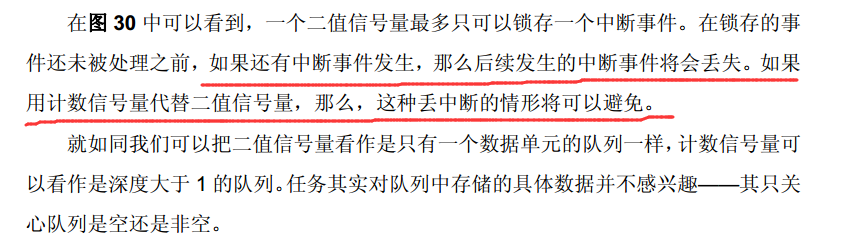
例程





Timer7每隔3sGive一次BinarySema，LCD任务循环Take BinarySema，这是中断延迟处理的例子。

1. 计数信号量



包含：#include "semphr.h"

声明：SemaphoreHandle\_t sSem;

创建：sSem = xSemaphoreCreateCounting(10, 0);

注：其余方法完全一样，只是创建是不同的。

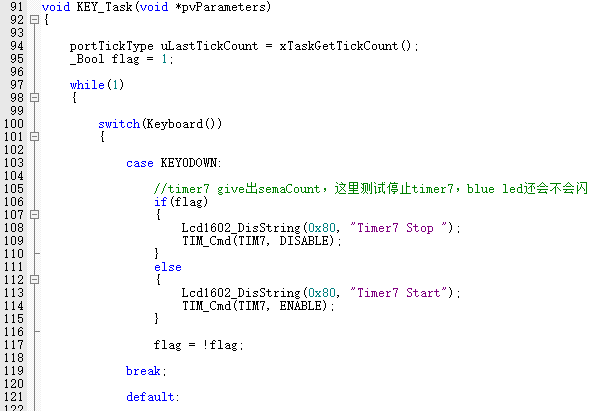
第一个参数：计数容量。

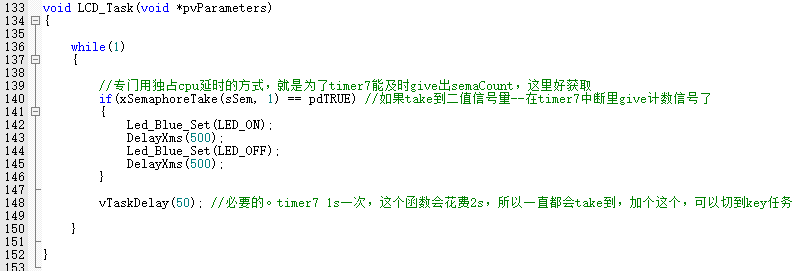
第二个参数：初值。

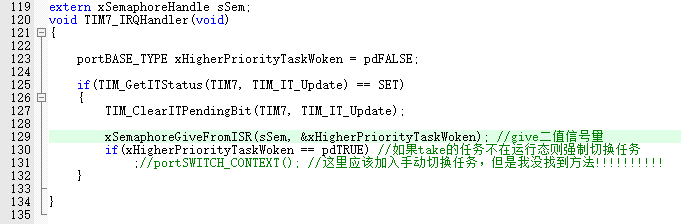
需要在FreeRTOS.h中第252行的

#define configUSE\_COUNTING\_SEMAPHORES 0。改为1才可使用。

例程

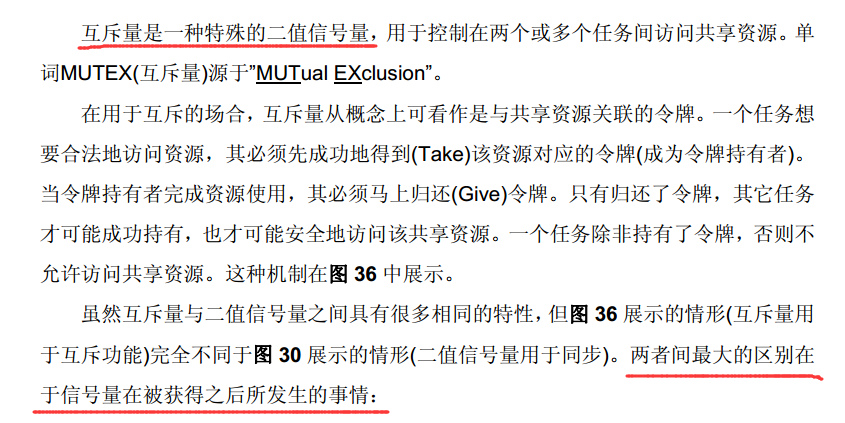






按键控制Timer7启停，Timer7干的事情就是每隔0.5s去Give一个CountingSema，LED任务就是Take到CountingSema，执行一次LED动作。当Timer7停止时，LED任务会吧CountingSema Take到为0为止。

1. 互斥信号量



包含：#include "semphr.h"

声明：SemaphoreHandle\_t sSem;

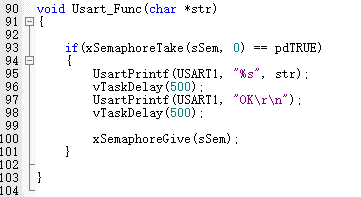
创建：sSem = xSemaphoreCreateMutex(); //创建互斥信号量

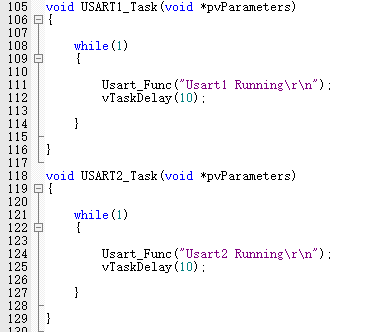
注：其余方法完全一样，只是创建是不同的。

需要在FreeRTOS.h中第244行的

#define configUSE\_MUTEXES 1。改为1才可使用。

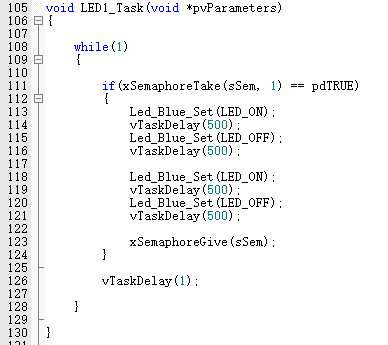
例程：

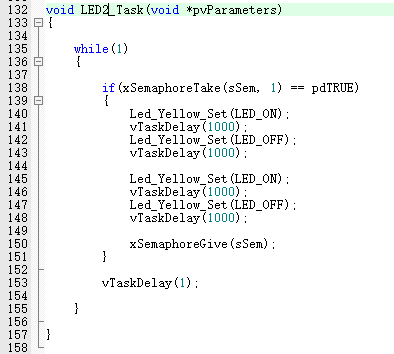
临界资源



USART1\_Task先调用Usart\_Func函数，里边先获取互斥量，然后打印一句，主动放弃cpu控制权，此时该运行USART2\_Task，但是它获取不到互斥量，阻塞，然后再执行USART1\_Task部分，执行完成，give互斥量，此时USART2\_Task即可打印出来。

例程2





1. 事件标志组-Event\_Groups

包含：#include "event\_groups.h"

声明：EventGroupHandle\_t eEvent; //事件标志组句柄

创建：eEvent = xEventGroupCreate();

设置：xEventGroupSetBits(eEvent, 0x01); //设置标志

等待：getBits = xEventGroupWaitBits(eEvent, 0x01, pdTRUE, pdTRUE, 1);

//句柄-等待标志-完成后是否清零标志-是否等待所有标志-超时时间

if((getBits & 0x01) == 0x01)

{

//…

}

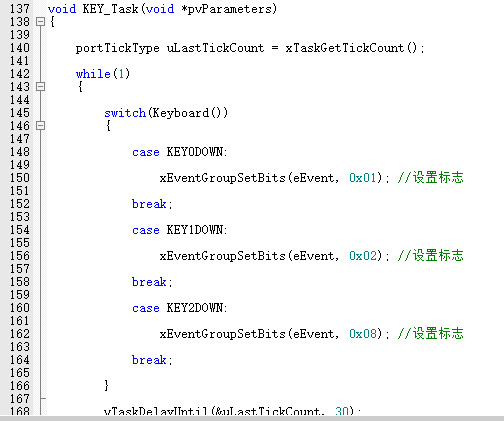
vTaskDelay(10);

删除：vEventGroupDelete(eEvent);

例程：







如果是等待bit1和bit3，pdTRUE、pdTRUE时(一定要判断if((getBits & 0x09) == 0x09))，先发生Bit1事件，getBits就会一直为0x01，然后发生Bit3事件，才会触发然后标志组内部吧对应事件清零；pdTRUE、pdFALSE时，发生任意一个事件后，触发然后标志组内部吧事件清零。

1. 软件定时器

包含：#include "timers.h"

声明：TimerHandle\_t t1\_Thdl, t2\_Thdl, t3\_Thdl;

创建：t1\_Thdl = xTimerCreate("Timer1", 200, pdTRUE, (void \*)1, (TimerCallbackFunction\_t)Timer\_Callback);

参数1：定时器名，调试用到。

参数2：定时周期。

参数3：pdTRUE-周期定时器；pdFALSE-单次定时器。

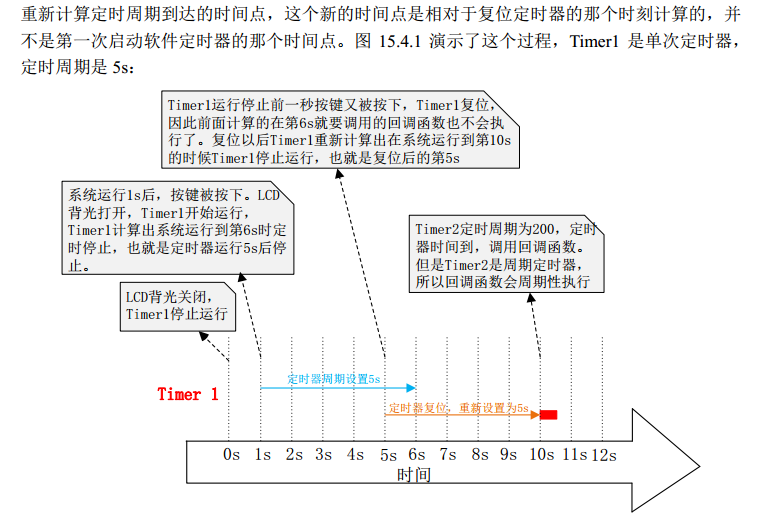
参数4：TimerID。允许多个定时器回调同一个函数，通过ID区分。

参数5：回调函数地址。

启停：xTimerStart(t1\_Thdl, 1); 句柄、等待时间。

xTimerStop(t1\_Thdl, 1);

复位：xTimerReset(t1\_Thdl, 1);



1. 内存管理

为了降低内存碎片的产生。FreeRTOS提供了一套内存分配与释放的方法。

分配：void \*pvPortMalloc(size\_t xWantedSize)

释放：void vPortFree(void \*pv)

使用方法同C库的malloc和free。

C库的内存分配方案的弊端

