FreeRTOS 学习之六：互斥量

前提：默认已经装好 MDK V5 和 STM32CubeMX，并安装了 STM32F1xx 系列的支持包。
硬件平台：STM32F1xx 系列。
目的：学习互斥量的使用。

多任务系统中存在一种潜在的风险。当一个任务在使用某个资源的过程中，即还没有完全结束对资源的访问时，便被切出运行态，使得资源处于非一致，不完整的状态。如果这个时候有另一个任务或者中断来访问这个资源，则会导致数据损坏或是其它相似的错误。

考虑如下情形，有两个任务都试图往一个 LCD 中写数据：

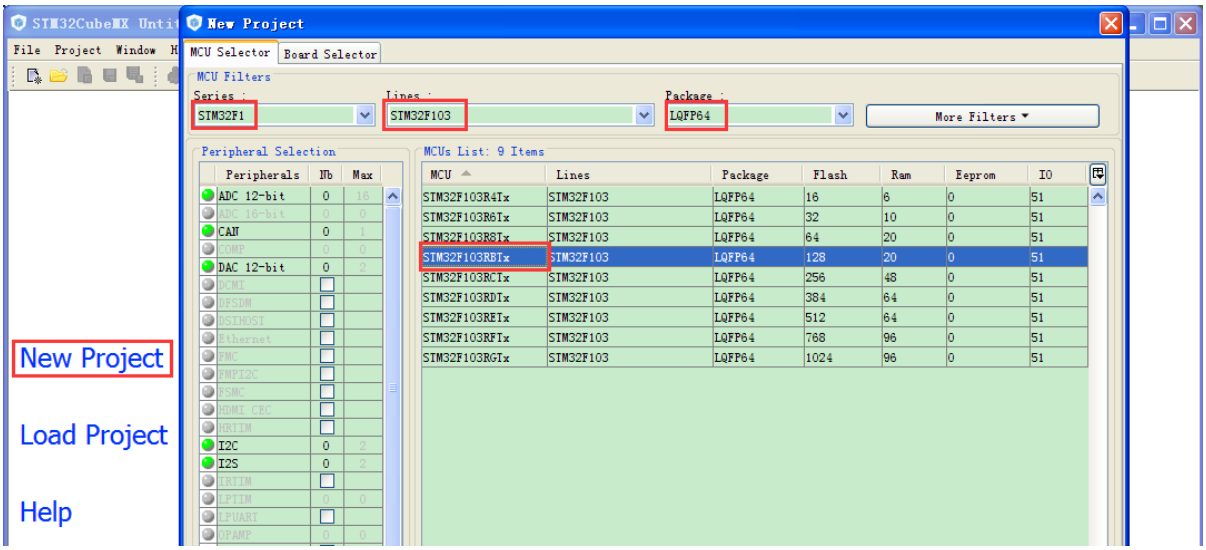
- 任务 A 运行，并往串口写字符串“Hello world”。
- 任务 A 被任务 B 抢占，但此时字符串才输出到“Hello w”。
- 任务 B 往串口写“Abort, Retry, Fail?”，然后进入阻塞态。
- 任务 A 从被抢占处继续执行，完成剩余的字符输出——“orld”。

现在串口接收端收到的是被破坏了的字符串“Hello wAbort, Retry, Fail?orld”。

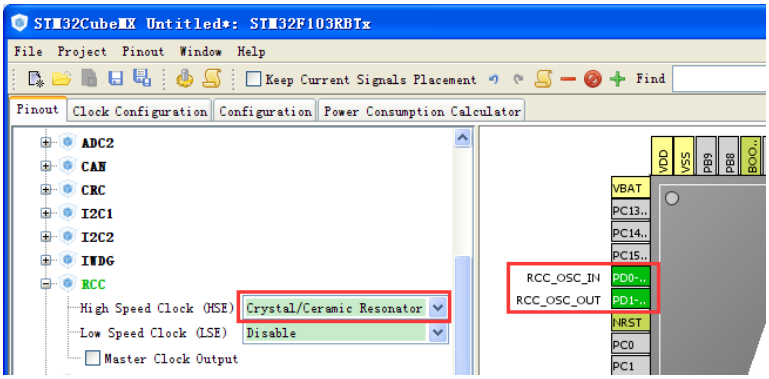
本文例子将再现上述问题，并用互斥量解决该问题。

本文例子使用 STM32CubeMX 配置创建两个任务，一个任务连续发送字符串“Hello world!”，另一个连续发送字符串“Abort, Retry, Fail?”。

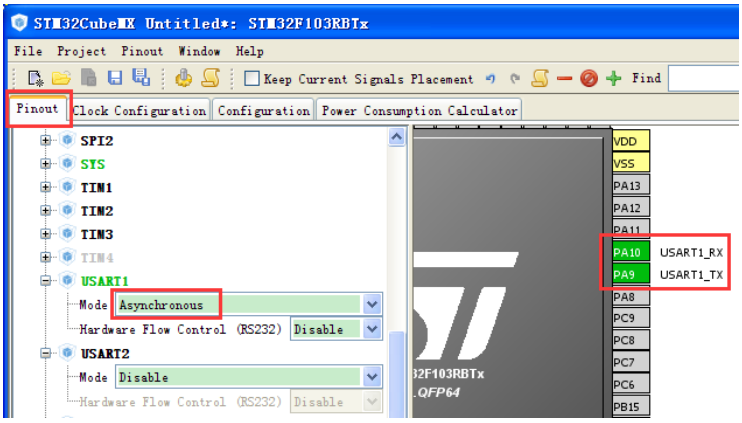
Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



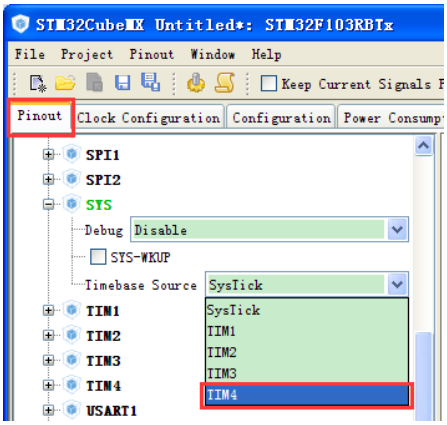
Step2.配置时钟引脚。



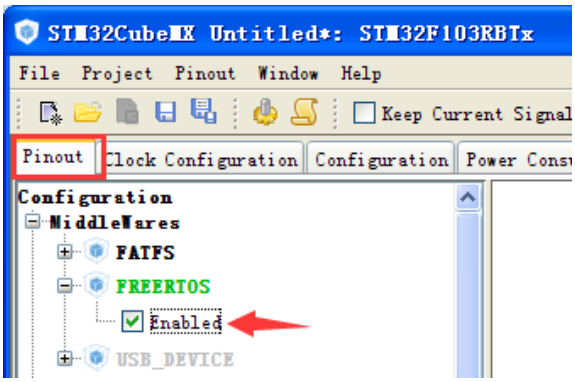
Step3.配置 USART1 为异步串口。



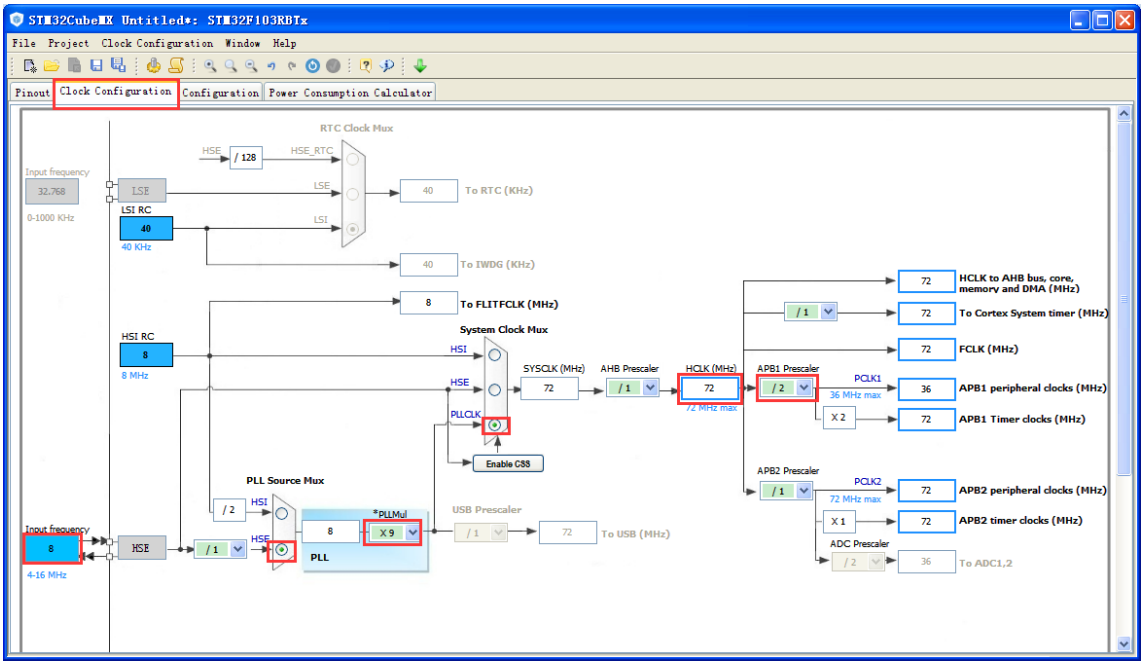
Step4.将系统时基源改为 TIM4。



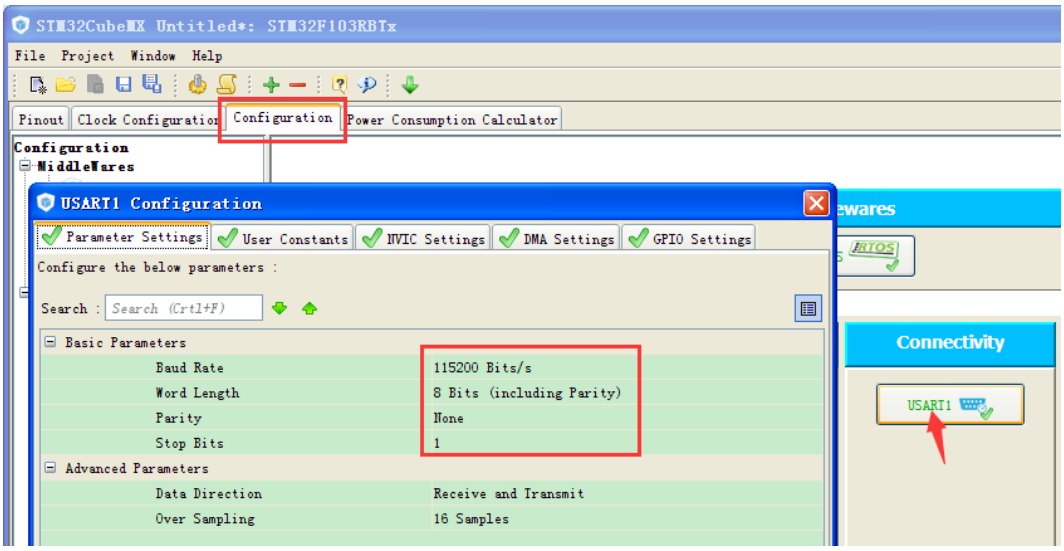
Step5.使能 FreeRTOS。



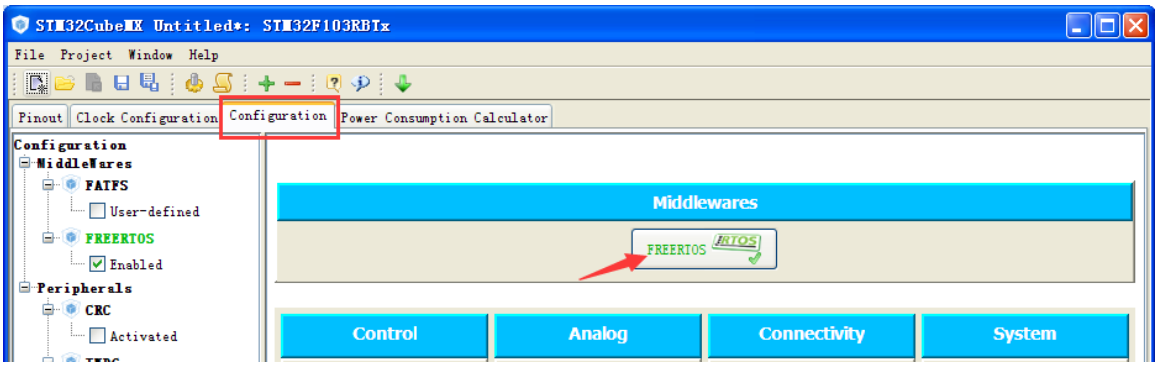
Step6.配置时钟树。8M 输入时，通过 PLL 得到 72M 内部时钟。



Step7.配置串口参数，使用默认值即可。

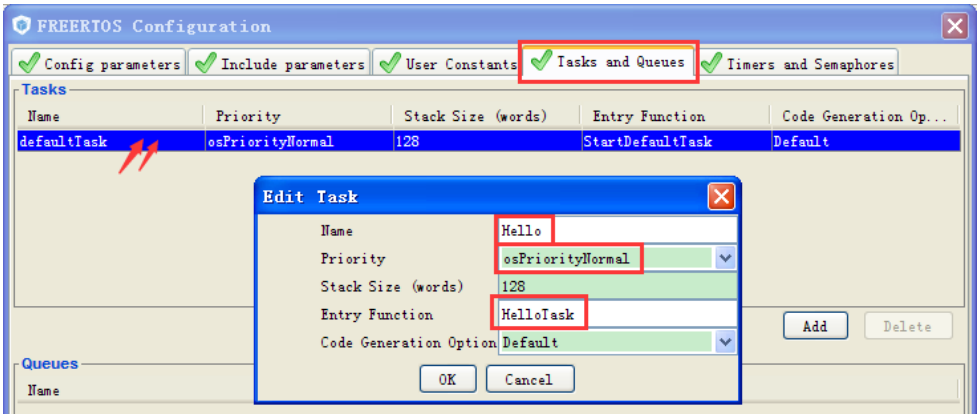


Step8.配置 FreeRTOS。

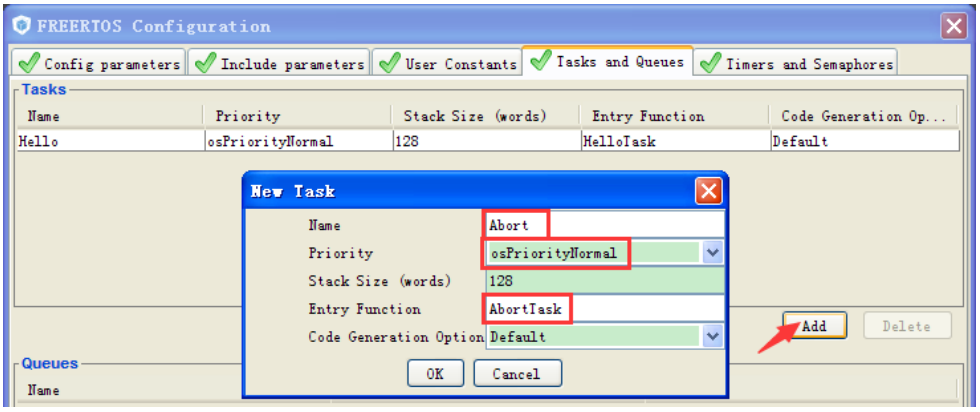


在 Tasks and Queues 选项卡，默认配置了一个名为 defaultTask 的任务，其优先级为普通，任务堆栈大小为 128 字，任务函数名为 StartDefaultTask。

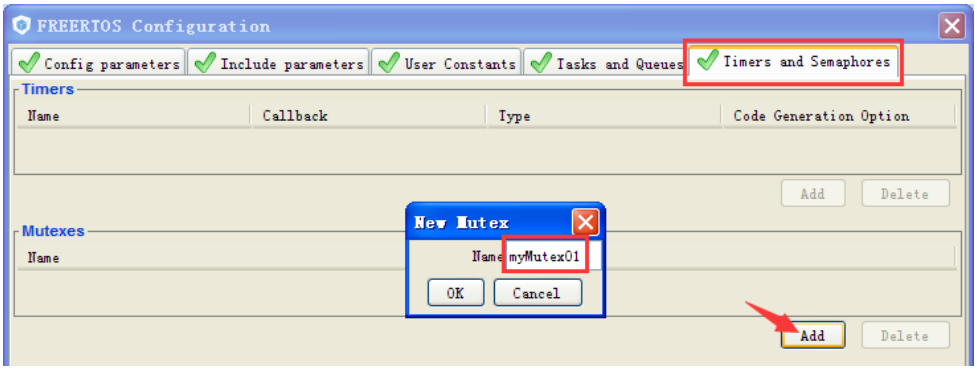
双击蓝色的地方，弹出对话框，将任务名修改为 Hello，将任务函数名修改为 HelloTask。



点击 Add 按钮，增加一个任务 Abort，优先级设置为 Normal，函数名为 AbortTask。

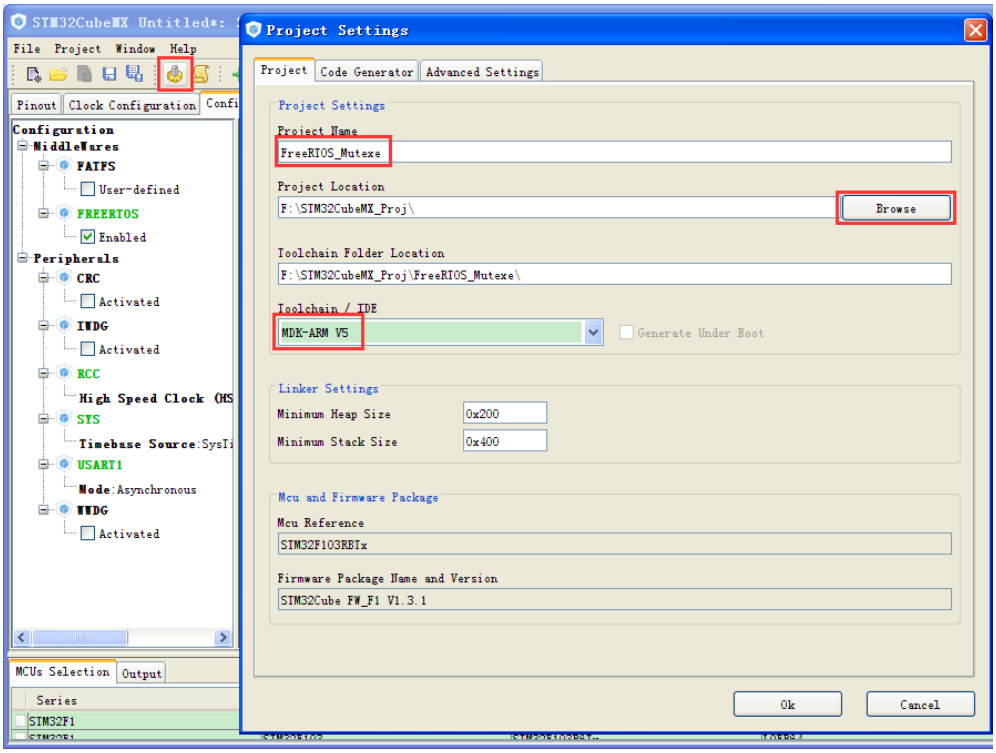


在 Timers and Semaphores 选项卡，添加互斥量 myMutex01。

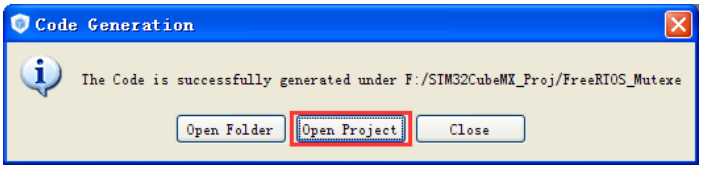


注：其他的都使用默认参数。

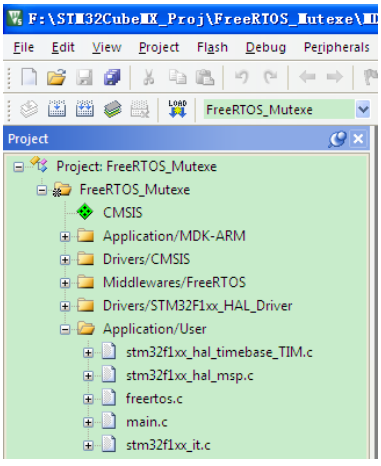
Step9.生成代码。



等完成后直接打开工程。



工程基本组织结构如下图，其中 Application/User 组中的文件是用户可以修改的，而其他组中的文件一般不进行修改。



Step10.分析程序结构。

在进入 main 函数之前，先定义了几个变量，声明了几个函数。

```
41  /* Private variables ---------------------
42  UART_HandleTypeDef huart1;
43
44  osThreadId HelloHandle;
45  osThreadId AbortHandle;
46  osMutexId myMutex01Handle;
47
48  /* USER CODE BEGIN PV */
49  /* Private variables ---------------------
50
51  /* USER CODE END PV */
52
53  /* Private function prototypes --------
54  void SystemClock_Config(void);
55  static void MX_GPIO_Init(void);
56  static void MX_USART1_UART_Init(void);
57  void HelloTask(void const * argument);
58  void AbortTask(void const * argument);
```

再看 main 函数。将 main 函数整理，删除很多注释之后，得到下图所示内容。

```
69  int main(void)
70  {
71    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
72    HAL_Init();
73    /* Configure the system clock */        ①
74    SystemClock_Config();
75    /* Initialize all configured peripherals */
76    MX_GPIO_Init();
77    MX_USART1_UART_Init();
78
79    /* Create the mutex(es) */
80    /* definition and creation of myMutex01 */          ②
81    osMutexDef(myMutex01);
82    myMutex01Handle = osMutexCreate(osMutex(myMutex01));
83    /* Create the thread(s) */
84    /* definition and creation of Hello */
85    osThreadDef(Hello, HelloTask,osPriorityNormal , 0, 128);
86    HelloHandle = osThreadCreate(osThread(Hello), NULL);
87    /* definition and creation of Abort */
88    osThreadDef(Abort, AbortTask, osPriorityNormal, 0, 128);
89    AbortHandle = osThreadCreate(osThread(Abort), NULL);
90    /* Start scheduler */
91    osKernelStart();      ③
92
93    /* We should never get here as control is now taken by the scheduler */
94
95    /* Infinite loop */
96    /* USER CODE BEGIN WHILE */
97    while (1)
98    {
99    }
100 }
```

其中第①部分，是硬件配置；第②部分，创建了一个互斥量和两个任务；第③部分，启动调度器。

Step11.添加代码。

先实现一个串口打印字符串的函数：

```
166 /* USER CODE BEGIN 4 */
167 void uart_send_str(const char *str)
168 {
169     uint16_t len;
170     len = strlen(str);
171     while(len--){
172         HAL_UART_Transmit(&huart1, (uint8_t *)str++, 1, 0xFFFF);
173         osDelay(5);
174     }
175 }
176 /* USER CODE END 4 */
```

要特别说明的是，因为 HAL_UART_Transmit()函数本身使用了硬件锁来避免串口共享冲突，所以在此使用了 osDelay()函数，目的是人为制造 UART 共享冲突，以模拟多任务系统中资源共享冲突的情况。

在此因为用到了 strlen()函数，所以要在 main 文件中包含头文件 string.h。
在两个任务函数中分别添加功能代码。

```
/* USER CODE END 4 */

/* HelloTask function */
void HelloTask(void const * argument)
{

  /* USER CODE BEGIN 5 */
    char *str = "\n\r Hello World! \n\r";
  /* Infinite loop */
  for(;;)
  {
      osDelay(333);
      uart_send_str(str);
  }
  /* USER CODE END 5 */
}
```

```
/* AbortTask function */
void AbortTask(void const * argument)
{
  /* USER CODE BEGIN AbortTask */
    char *str = "\n\r Abort, Retry, Fail? \n\r";
  /* Infinite loop */
  for(;;)
  {
      osDelay(100);
      uart_send_str(str);
  }
  /* USER CODE END AbortTask */
}
```

Step12.编译下载运行。通过串口调试助手接收 MCU 发送的数据，下图是一种可能的结果。

```
Abort, Retry, Fail?  Hello World!
Abort, Retry, Fail?
Abort, Retry,  FHaeilll?o  World!
Abort, Retry, Fail?
Abort, R eHterlyl,o  FWaoirll?d !

Abort, Retry, Fail?
Abo rHte,l lRoe tWroyr,l dF!a il?
Abort, Retry, Fail?

H eAlbloor tW,o rRledt!r y, Fail?
```

Step13.用互斥量解决共享冲突。
在实际应用中，比如使用 LCD 输出显示，就可能会遇到上述的共享冲突情况。
把串口打印字符串的函数函数改为：

```
/* USER CODE BEGIN 4 */
void uart_send_str(const char *str)
{
    uint16_t len;
    len = strlen(str);
    osMutexWait(myMutex01Handle, osWaitForever);
    while(len--){
        HAL_UART_Transmit(&huart1, (uint8_t *)str++, 1, 0xFFFF);
        osDelay(5);
    }
    osMutexRelease(myMutex01Handle);
}
/* USER CODE END 4 */
```

在使用串口输出前，先申请信号量，输出完字符串后，释放信号量。这样就解决了上面的冲突。下图是输出结果实例。

```
Abort, Retry, Fail?
Hello World!
Abort, Retry, Fail?
Abort, Retry, Fail?
Hello World!
Abort, Retry, Fail?
Abort, Retry, Fail?
Hello World!
Abort, Retry, Fail?
Abort, Retry, Fail?
Hello World!
Abort, Retry, Fail?
Abort, Retry, Fail?
Hello World!
Abort, Retry, Fail?
Abort, Retry, Fail?
Hello World!
Abort, Retry, Fail?
Abort, Retry, Fail?
Hello World!
```

S.D.Lu 于 深圳
2016 年 8 月