

FreeRTOS：任务状态和信息查询

原创

Hello xiǎo lǐ

已于 2023-05-15 20:56:04 修改

1060

收藏

5

版权

分类专栏：

FreeRTOS学习


 文章标签：

单片机

stm32

FreeRTOS

嵌入式



FreeRTOS学习 专栏收录该内容

1 订阅 12

目录

- 一、任务相关 API函数预览
- 二、任务相关API函数详解
 - 2.1uxTaskPriorityGet()
 - 2.2vTaskPrioritySet()
 - 2.3uxTaskGetSystemState() ※※※※※
 - 2.4vTaskGetInfo() ※※※※※
 - 2.5xTaskGetApplicationTaskTag()
 - 2.6xTaskGetCurrentTaskHandle()
 - 2.7xTaskGetHandle()
 - 2.8xTaskGetIdleTaskHandle()
 - 2.9uxTaskGetStackHighWaterMark()
 - 2.10eTaskGetState()
 - 2.11pcTaskGetName()
 - 2.12xTaskGetTickCount()
 - 2.13xTaskGetTickCountFromISR()
 - 2.14xTaskGetSchedulerState()
 - 2.15uxTaskGetNumberOfTask()
 - 2.16vTaskList()※※※※※
 - 2.17vTaskGetRunTimeStats()
 - 2.18vTaskSetApplicationTaskTag()
 - 2.19vTaskSetThreadLocalStoragePointer()
 - 2.20pvTaskGetThreadLocalStoragePointer()
- 三、函数应用
 - 3.1任务状态查询 API函数实验
 - 3.1.1实验要求
 - 3.1.2程序代码
 - 3.2任务运行时间信息统计实验
 - 3.2.1实验要求
 - 3.2.2程序代码

一、任务相关 API函数 预览

FreeRTOS 中提供了很多函数可以用来获取相应的任务信息，FreeRTOS中的任务信息查询函数列举如下：

函数名	描述
uxTaskPriorityGet()	获取某任务优先级
vTaskPrioritySet()	改变某任务优先级
uxTaskGetSystemState()	获取系统中所有任务状态信息
vTaskGetInfo()	获取某个任务信息
xTaskGetApplicationTaskTag()	获取某个任务的标签值
xTaskGetCurrentTaskHandle()	获取当前正在运行的任务的句柄
uxTaskGetStackHighWaterMark()	获取任务堆栈历史剩余最小值
eTaskGetState()	获取某任务状态
pcTaskGetName()	获取某任务名字
xTaskGetTickCount()	获取系统时间计数器值
xTaskGetTickCountFromISR()	在中断中获取时间计数器的值
xTaskGetSchedulerState()	获取任务调度器的状态
uxTaskGetNumberOfTask()	获取当前系统中存在的任务数量
vTaskList()	以表格形式输出当前系统中所有信息
vTaskGetRunTimeStats()	获取每个任务的运行时间
vTaskSetApplicationTaskTag()	设置任务标签值

CSDN @无敵XIAOLEI

这些 API函数在 FreeRTOS官网上都有，如图所示：



CSDN @无敵XIAOLEI

二、任务相关API函数详解

2.1uxTaskPriorityGet()

查询某个任务的优先级

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_uxTaskPriorityGet 1 // 必须置为1
3  /*****函数原型*****/
4  函数原型: UBaseType_t uxTaskPriorityGet(TaskHandle_t xTask)
5  传 入 值: xTask 要查找的任务的任务句柄
6  返 回 值: 获取到的对应的任务的优先级
```

2.2vTaskPrioritySet()

改变某个任务的优先级

```

1  /*****相关宏的配置*****/
2  #define INCLUDE_vTaskPrioritySet          必须置为1
3  /*****函数原型*****/
4  函数原型: void vTaskPrioritySet(TaskHandle_t xTask, UBaseType_t uxNewPriority)
5  传 入 值: xTask 要更改优先级的任务的任务句柄
6             uxNewPriority 任务要使用的新的优先级

```

2.3uxTaskGetSystemState() ※※※※

获取系统中所有任务的任务状态，每个任务的状态信息保存在一个TaskStatus_t类型的结构体里，这个结构体包含了任务句柄、任务名称、堆栈、优先级等信息

```

1  /*****相关宏的配置*****/
2  #define configUSE_TRACE_FACILITY          必须置为1
3  /*****函数原型*****/
4  函数原型: UBaseType_t uxTaskGetSystemState(TaskStatus_t *const pxTaskStatusArray,
5                                               const UBaseType_t uxArraySize,
6                                               uint32_t *const pulTotalRunTime)
7  传 入 值: pxTaskStatusArray 指向TaskStatus_t结构体类型的数组首地址
8             uxArraySize      保存任务状态数组的大小
9             pulTotalRunTime   若开启了系统运行时间统计，则用来保存系统总的运行时间
10 返 回 值: 统计到的任务状态的个数，也就是填写到数组pxTaskStatusArray中的个数

```

```

1  /****TaskStatus_t结构体****/
2  typedef struct xTASK_STATUS{
3      TaskHandle_t    xHandle;          //任务句柄
4      const char*      pcTaskName;      //任务名字
5      UBaseType_t      xTaskNumber;     //任务编号
6      eTaskState       eCurrentState;   //当前任务状态
7      UBaseType_t      uxCurrentPriority; //任务当前的优先级
8      UBaseType_t      uxBasePriority;   //任务基础优先级
9      uint32_t          ulRunTimeCounter; //任务运行的总时间
10     StackType_t*      pxStackBase;     //堆栈基地址
11     uint16_t          usStackHighWaterMark; //任务创建依赖任务堆栈剩余的最小值
12 }TaskStatus_t;

```

下面简要分析一下该函数的源码

```

1  UBaseType_t uxTaskGetSystemState( TaskStatus_t * const pxTaskStatusArray, const UBaseType_t uxArraySize, uint32_t * const pulTotalRunTime)
2  {
3      UBaseType_t uxTask = 0, uxQueue = configMAX_PRIORITIES;
4
5      vTaskSuspendAll();-----(1)
6      {
7          /* Is there a space in the array for each task in the system? */
8          if( uxArraySize >= uxCurrentNumberOfTasks )------(2)
9          {
10             /* Fill in an TaskStatus_t structure with information on each
11             task in the Ready state. */
12             do------(3)
13             {
14                 uxQueue--;
15                 uxTask += prvListTasksWithinSingleList( &(amp; pxTaskStatusArray[ uxTask ] ), &(amp; pxReadyTaskList) );
16             } while( uxQueue > ( UBaseType_t ) tskIDLE_PRIORITY ); /*lint !e961 MISRA exception as the cast is needed */
17
18             /* Fill in an TaskStatus_t structure with information on each
19             task in the Blocked state. */
20             uxTask += prvListTasksWithinSingleList( &(amp; pxTaskStatusArray[ uxTask ] ), ( List_t * ) pxDelayedTasks );
21             uxTask += prvListTasksWithinSingleList( &(amp; pxTaskStatusArray[ uxTask ] ), ( List_t * ) pxOverflownTasks );
22
23             #if( INCLUDE_vTaskDelete == 1 )------(6)
24             {
25                 /* Fill in an TaskStatus_t structure with information on
26                 each task that has been deleted but not yet cleaned up. */
27                 uxTask += prvListTasksWithinSingleList( &(amp; pxTaskStatusArray[ uxTask ] ), &xtasksWaiting );
28             }
29             }
30      }
31      vTaskResumeAll();
32      return uxTask;
33  }

```

```

29         }
30     #endif
31
32     #if ( INCLUDE_vTaskSuspend == 1 )----- (7)
33     {
34         /* Fill in an TaskStatus_t structure with information on
35         each task in the Suspended state. */
36         uxTask += prvListTasksWithinSingleList( &(amp; pxTaskStatusArray[ uxTask ] ), &xSuspendedTa:
37     }
38     #endif
39
40     #if ( configGENERATE_RUN_TIME_STATS == 1 )----- (8)
41     {
42         if( pulTotalRunTime != NULL )
43         {
44             #ifdef portALT_GET_RUN_TIME_COUNTER_VALUE
45                 portALT_GET_RUN_TIME_COUNTER_VALUE( ( *pulTotalRunTime ) );
46             #else
47                 *pulTotalRunTime = portGET_RUN_TIME_COUNTER_VALUE();
48             #endif
49         }
50     }
51     #else
52     {
53         if( pulTotalRunTime != NULL )
54         {
55             *pulTotalRunTime = 0;
56         }
57     }
58     #endif
59 }
60 else
61 {
62     mtCOVERAGE_TEST_MARKER();
63 }
64 }
65 ( void ) xTaskResumeAll();----- (9)
66
67 return uxTask;
68 }
69
70

```

(1)、挂起任务调度器，停止任务调度，以免任务打断，或任务状态信息改变。

(2)、判断定义的结构体数组大小是否比当前系统所有任务数量，即判断是否能够存储得下系统中所有任务的信息。

(3)、获取处于准备态任务的信息。uxQueue表示任务的优先级，按照高优先级到低优先级的顺序来获取任务信息。prvListTasksWithinSingleList()函数是用来获取单个任务的信息，在该任务中调用了vTaskGetInfo()函数,该函数返回值为同一优先级同一状态的任务数量。uxTask用以统计获得信息的任务个数。

(4)、获取处于阻塞态延时列表中任务的信息。

(5)、获取处于阻塞态延时溢出列表中任务信息，因为任务阻塞时可能会产生溢出，所以用这两个列表共同表示阻塞态任务。

(6)、获取未完全删除的任务信息。

(7)、获取处于挂起态任务的信息。

(8)、获取任务运行时间。本章节没有用到该信息

(9)、调度器解挂。

函数用法示例：

```

1 TaskStatus_t* StatusArray;
2 BaseType_t ArraySize ;
3 ArraySize = uxTaskGetNumberOfTasks();
4 StatusArray = pvPortMalloc(ArraySize * sizeof(TaskStatus_t)); //动态申请空间
5
6 printf("-----Get System State-----\r\n");
7 if (StatusArray != NULL)
8 {
9     ArraySize = uxTaskGetSystemState(StatusArray, ArraySize, &pulTotalRunTime);
10    //printf("TaskName\t\tPriority\tTaskNumber\t\runtime\r\n");
11

```

```

12     for (i=0; i<ArraySize; i++)
13     {
14
15         printf( "%d\t%s\t%#x\t%d\t%ld\t%ld\t%#x\t%ld\t\r\n",
16                 StatusArray[i].eCurrentState,
17                 StatusArray[i].pcTaskName,
18                 StatusArray[i].pxStackBase,
19                 StatusArray[i].usStackHighWaterMark,
20                 StatusArray[i].uxBasePriority,
21                 StatusArray[i].uxCurrentPriority,
22                 StatusArray[i].xHandle,
23                 StatusArray[i].xTaskNumber
24             );
25     }
26 }

```

2.4vTaskGetInfo() ※※※※※

获取某个指定的单个任务的任务信息

```

1  /*****相关宏的配置*****/
2  #define configUSE_TRACE_FACILITY          必须置为1
3  /*****函数原型*****/
4  函数原型: void vTaskGetInfo(TaskHandle_t xTask,
5                               TaskStatus_t* pxTaskStatus,
6                               BaseType_t xGetFreeStackSpace,
7                               eTaskState eState)
8  传 入 值: xTask 要查找的任务的任务句柄
9            pxTaskStatus 指向类型为TaskStatus_t结构体变量
10           xGetFreeStackSpace 堆栈剩余的历史最小值
11           eState 保存任务运行状态

```

函数输入参数有四个，TaskHandle_t xTask为所要查询任务的任务句柄；TaskStatus_t*pxTaskStatus为任务状态结构体，用于存放任务的状态，该结构体需要用户自定义；BaseType_t xGetFreeStackSpace为是否计算任务剩余最小堆栈大小，传入参数pdTRUE表示计算，但需要耗费一些时间；eTaskState eState为是否获取任务状态，输入参数eInvalid时表示获取任务状态，但同样也会消耗一些时间，否则用户指定任务状态，不会消耗时间。下面深入分析一下该函数源码：

```

1  void vTaskGetInfo( TaskHandle_t xTask, TaskStatus_t *pxTaskStatus, BaseType_t xGetFreeStackSpace, eTaskState eState )
2  {
3      TCB_t *pxTCB;
4
5      /* xTask is NULL then get the state of the calling task. */
6      pxTCB = prvGetTCBFromHandle( xTask );-----(1)
7
8      pxTaskStatus->xHandle = ( TaskHandle_t ) pxTCB;-----(2)
9      pxTaskStatus->pcTaskName = ( const char * ) &( pxTCB->pcTaskName [ 0 ] );-----(3)
10     pxTaskStatus->uxCurrentPriority = pxTCB->uxPriority;-----(4)
11     pxTaskStatus->pxStackBase = pxTCB->pxStack;-----(5)
12     pxTaskStatus->xTaskNumber = pxTCB->uxTCBNumber;-----(6)
13
14     #if ( INCLUDE_vTaskSuspend == 1 ) -----(7)
15     {
16         /* If the task is in the suspended list then there is a chance it is
17          actually just blocked indefinitely - so really it should be reported as
18          being in the Blocked state. */
19         if( pxTaskStatus->eCurrentState == eSuspended )
20         {
21             vTaskSuspendAll();
22             {
23                 if( listLIST_ITEM_CONTAINER( &( pxTCB->xEventListItem ) ) != NULL )
24                 {
25                     pxTaskStatus->eCurrentState = eBlocked;
26                 }
27             }
28             xTaskResumeAll();
29         }
30     }
31 }

```

```
31
32     #endif /* INCLUDE_vTaskSuspend */
33
34     #if ( configUSE_MUTEXES == 1 ) -----(8)
35     {
36         pxTaskStatus->uxBasePriority = pxTCB->uxBasePriority;
37     }
38     #else
39     {
40         pxTaskStatus->uxBasePriority = 0;
41     }
42     #endif
43
44     #if ( configGENERATE_RUN_TIME_STATS == 1 ) -----(9)
45     {
46         pxTaskStatus->ulRunTimeCounter = pxTCB->ulRunTimeCounter;
47     }
48     #else
49     {
50         pxTaskStatus->ulRunTimeCounter = 0;
51     }
52     #endif
53
54     /* Obtaining the task state is a little fiddly, so is only done if the value
55     of eState passed into this function is eInvalid - otherwise the state is
56     just set to whatever is passed in. */
57     if( eState != eInvalid ) -----(10)
58     {
59         pxTaskStatus->eCurrentState = eState;
60     }
61     else
62     {
63         pxTaskStatus->eCurrentState = eTaskGetState( xTask );
64     }
65
66     /* Obtaining the stack space takes some time, so the xGetFreeStackSpace
67     parameter is provided to allow it to be skipped. */
68     if( xGetFreeStackSpace != pdFALSE ) -----(11)
69     {
70         #if ( portSTACK_GROWTH > 0 )
71         {
72             pxTaskStatus->usStackHighWaterMark = prvTaskCheckFreeStackSpace( ( uint8_t * ) pxTCB->pxEndOfStack );
73         }
74         #else
75         {
76             pxTaskStatus->usStackHighWaterMark = prvTaskCheckFreeStackSpace( ( uint8_t * ) pxTCB->pxStack );
77         }
78         #endif
79     }
80     else
81     {
82         pxTaskStatus->usStackHighWaterMark = 0;
83     }
84 }
```

(1)、获取该任务的任务句柄

(2)、获取任务句柄，存放在任务状态结构体中

(3)、获取任务名的首地址，存放在任务状态结构体中

(4)、获取当前任务优先级，存放在任务状态结构体中

(5)、获取任务堆栈首地址，存放在任务状态结构体中

(6)、获取任务标号，存放在任务状态结构体中

(7)、这段宏定义完成的任务就是，如果任务处于挂起状态，那么有可能它是被无限期阻塞所导致的，所以此时判断是否是这种可能，如果是则将它状态反馈为阻塞态，而非挂起态。

(8)、这段条件编译是判断用户是否使用了互斥量，因为当用户使用了互斥量时，互斥信号量有可能会降低低优先级的任务优先级会拉高，这点在以后会讲述。所以这里如果使用了互斥量，那么就会获取其最初始的优先级大小，然后存放在任务状态结构体中

(9)、这段条件编译如果用户配置了任务事件信息统计，则会获取当前任务的运行时间。当配置configGENERATE_RUN_TIME_STATS == 1时需要用户自己再定义两个函数，具体配置会在后面的一个章节中讲述，这里只需要知道vTaskGetInfo()函数是可以获取任务运行时间的，不过本章没有用到。

- (10)、判断用户传入的状态参数是否是无效的，如果是则获取当前任务状态但会耗费一些时间，否则将用户传入的状态参数作为当前任务状态。
- (11)、是否获取任务历史最小剩余堆栈，如果传入参数pdTRUE，则会获取任务历史最小剩余堆栈同样会耗费一些时间，否则不获取。

总结一下vTaskGetInfo()函数可以获取的任务信息有：

- 1.任务句柄 xHandle
- 2.任务名 pcTaskName
- 3.任务当前优先级 uxCurrentPriority
- 4.任务堆栈基地址 pxStackBase
- 5.任务编号 xTaskNumber
- 6.当前任务状态 eCurrentState
- 7.任务初始优先级 uxBasePriority
- 8.任务运行时间 ulRunTimeCounter
- 9.任务历史剩余堆栈最小值 usStackHighWaterMark

函数用法示例：

```
1      TaskStatus_t  TaskStatus;
2
3      vTaskGetInfo(QueryTask_Handler,&TaskStatus,pdTRUE,eInvalid);
4
5      printf("-----Get Task Info-----\r\n");
6      printf("eCurrentState = %d\r\n",TaskStatus.eCurrentState);
7      printf("pcTaskName = %s\r\n",TaskStatus.pcTaskName);
8      printf("pxStackBase = %x\r\n",(int)TaskStatus.pxStackBase);
9      printf("usStackHighWaterMark = %d\r\n",TaskStatus.usStackHighWaterMark);
10     printf("uxBasePriority = %ld\r\n",TaskStatus.uxBasePriority);
11     printf("uxCurrentPriority = %ld\r\n",TaskStatus.uxCurrentPriority);
12     printf("xTaskNumber = %ld\r\n",TaskStatus.xTaskNumber);
13     printf("xHandle = %x\r\n",(void *)TaskStatus.xHandle);
14     printf("QueryTask_Handler= %x\r\n",(void *)QueryTask_Handler);
15     printf("-----Get Task Info-----\r\n");
```

2.5xTaskGetApplicationTaskTag()

获取某个任务的标签值

标签值的功能由用户自行决定， 此函数就是来获取这个标签值的。

要使用此函数话宏 configUSE_APPLICATION_TASK_TAG必须为1

```
1  /*****相关宏的配置*****/
2  #define configUSE_APPLICATION_TASK_TAG  必须置为1
3  /*****函数原型*****/
4  函数原型: void xTaskGetApplicationTaskTag(TaskHandle_t xTask)
5  传 入 值: xTask 要获取标签值的任务的任务句柄，若为NULL表示获取当前任务的标签值
6  返 回 值: 任务的标签值
```

2.6xTaskGetCurrentTaskHandle()

获取当前正在运行任务的任务句柄， 其实获取到的就是任务控制块

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_xTaskGetCurrentTaskHandle  必须置为1
3  /*****函数原型*****/
4  函数原型: TaskHandle_t xTaskGetCurrentTaskHandle(void)
5  返 回 值: 当前任务的任务句柄
```

2.7xTaskGetHandle()

根据任务名字查找某个任务的句柄

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_xTaskGetHandle  必须置为1
3  /*****函数原型*****/
4  函数原型: TaskHandle_t xTaskGetHandle(const char* pcNameToQuery)
5  传 入 值: pcNameToQuery 任务名
6  返 回 值: 任务名所对应的任务句柄；返回NULL表示没有对应的任务
```

2.8xTaskGetIdleTaskHandle()

获取空闲任务的句柄

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_xTaskGetIdleTaskHandle  必须置为1
3  /*****函数原型*****/
4  函数原型: TaskHandle_t xTaskGetIdleTaskHandle(void)
5  返 回 值: 空闲任务的任务句柄
```

2.9uxTaskGetStackHighWaterMark()

每个任务在创建的时候就确定了堆栈大小，此函数用于检查任务从创建好到现在的历史剩余最小值，这个值越小说明任务堆栈溢出的可能性就越大

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_uxTaskGetStackHighWaterMark  必须置为1
3  /*****函数原型*****/
4  函数原型: UBaseType_t uxTaskGetStackHighWaterMark(TaskHandle_t xTask)
5  传 入 值: xTask 要查询的任务的任务句柄，若为NULL表示查询自身任务的高水位线
6  返 回 值: 任务堆栈的高水位线值，即堆栈的历史剩余最小值
```

2.10eTaskGetState()

获取某个任务的状态，比如：运行态、阻塞态、挂起态、就绪态等

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_eTaskGetState  必须置为1
3  /*****函数原型*****/
4  函数原型: eTaskState eTaskGetState(TaskHandle_t xTask)
5  传 入 值: xTask 要查询的任务的任务句柄
6  返 回 值: 返回eTaskState枚举类型值
```

2.11pcTaskGetName()

根据任务句柄来获取该任务的任务名字

```
1  /*****函数原型*****/
2  函数原型: char* pcTaskGetName(TaskHandle_t xTaskToQuery)
3  传 入 值: xTaskToQuery 要查询的任务的任务句柄，若为NULL表示查询自身任务名
4  返 回 值: 返回任务所对应的任务名
```

2.12xTaskGetTickCount()

用于查询任务调度器从启动到现在时间计数器xTickCount的值。xTickCount是系统的时钟节拍值，并不是真实的时间值。每个滴答定时器中断xTickCount就会加一，中断周期取决于系统时钟节拍数

```
1  /*****函数原型*****/
2  函数原型: TickType_t xTaskGetTickCount(void)
3  返 回 值: 时间计数器xTickCount的值
```

2.13xTaskGetTickCountFromISR()

在中断服务函数中获取时间计数器值

```
1  /*****函数原型*****/
2  函数原型: TickType_t xTaskGetTickCountFromISR(void)
3  返 回 值: 时间计数器xTickCount的值
```

2.14xTaskGetSchedulerState()

获取任务调度器的状态，开启、关闭还是挂起

```
1  /*****相关宏的配置*****/
2  #define INCLUDE_xTaskGetSchedulerState  必须置为1
3  /*****函数原型*****/
4  函数原型: BaseType_t xTaskGetSchedulerState(void)
5  返 回 值: taskSCHEDULER_NOT_STARTED 调度器未启动
```


6		taskCHEDULER_RUNNING 调度器正在运行
7		taskCHEDULER_SUSPENDED 调度器挂起

2.15uxTaskGetNumberOfTask()

获取当前系统中存在的任务数量

```
1  /*****函数原型*****/
2  函数原型: UBaseType_t uxTaskGetNumberOfTask(void)
3  返回 值: 当前系统中存在的任务数量, 此值包含各种状态下的任务数
```

2.16vTaskList()****

很多时候, 我们并不需要这么多信息, 我们只需要几个常用的关键信息即可。所以FreeRTOS为我们提供了vTaskList(), 该函数不仅可以获取关键信息, 而且用法简单, 以表格的形式输出当前系统中所有任务的详细信息

```
1  /*****函数原型*****/
2  函数原型: void vTaskList(char *pcWriteBuffer)
3  传 入 值: pcWriteBuffer 保存任务状态信息表的存储区
```

该函数的函数声明很简单, 输入形参就一个字符指针。该指针需要用户自定义, 用以存放任务信息。函数源码这里就不分析了。我们直接给出其用法示例:

```
1  char InfoBuffer[200];
2  vTaskList(InfoBuffer);
3  printf("taskName\ttaskState\ttaskPrio\ttaskStack\ttaskNum\n");
4  printf("%s",InfoBuffer);
```

下图是其输出信息

Name	State	Priority	Stack	Num
query_task	R	2	0	6
start_task	R	1	86	1
IDLE	R	0	120	2
key_task	B	2	4	5
led_task	B	4	4	4
Tmr Svc	S	31	234	3

CSDN @无敵XIAOLEI

从上图可以看出, 该函数可以获取任务状态, 优先级, 剩余堆栈大小, 任务序号这四项信息。

2.17vTaskGetRunTimeStats()

获取每个任务的运行时间

```
1  /*****相关宏的配置*****/
2  #define configGENERATE_RUN_TIME_STATS 必须置为1
3  #define configUSE_STATS_FORMATTING_FUNCTIONS 必须置为1
4  /*****函数原型*****/
5  函数原型: void vTaskGetRunTimeStats(char *pcWriteBuffer)
6  传 入 值: pcWriteBuffer 保存任务时间信息的存储区
```

2.18vTaskSetApplicationTaskTag()

用于设置某个任务的标签值

```
1  /*****相关宏的配置*****/
2  #define configUSE_APPLICATION_TASK_TAG 必须置为1
3  /*****函数原型*****/
4  函数原型: void vTaskSetApplicationTaskTag(TaskHandle_t xTask,
5                                              TaskHookFunction_t pxHookFunction)
6  传 入 值: xTask 要设置标签值的任务的句柄
7              pxHookFunction 要设置的标签值
```

2.19vTaskSetThreadLocalStoragePointer()

设置线程本地存储指针的值, 每个任务都有自己的指针数组作为线程本地存储, 使用这些线程本地存储可以用来在任务控制块中存储一些只属于任务自己的应用信息

```
1  /*****相关宏的配置*****/
2  #define configNUM_THREAD_LOCAL_STORAGE_POINTERS 本地存储指针数组的大小
3  /*****函数原型*****/
4  函数原型: void vTaskSetThreadLocalStoragePointer(TaskHandle_t xTaskToSet,
5
6
7  BaseType_t xIndex,
8  void* pvValue)
9
10 传 入 值: xTaskToSet 要设置线程本地存储指针的任务的任务句柄
11
12          xIndex 要设置的线程本地存储指针数组的索引
13
14          pvValue 要存储的值
```

2.20pvTaskGetThreadLocalStoragePointer()

获取线程本地存储指针

```
1  /*****相关宏的配置*****/
2  #define configNUM_THREAD_LOCAL_STORAGE_POINTERS 本地存储指针数组的大小
3  /*****函数原型*****/
4  函数原型: void *pvTaskGetThreadLocalStoragePointer(TaskHandle_t xTaskToQuery,
5
6
7  BaseType_t xIndex )
8
9 传 入 值: xTaskToQuery 要获取线程本地存储指针的任务的任务句柄
10
11          xIndex 要获取的线程本地存储指针数组的索引
```

三、函数应用

3.1任务状态查询 API函数实验

3.1.1实验要求

本实验设计三个任务：start_task、led0_task 和 query_task ，这三个任务的功能如下：

start_task：用来创建其他 2个任务。

led0_task：控制LED0灯闪烁，提示系统正在运行。

query_task：任务状态和信息查询任务

实验需要一个按键 KEY_UP，这四个按键的功能如下：

KEY_UP: 控制程序的运行步骤。

3.1.2程序代码

任务优先级、堆栈大小和句柄等的设置如下：

```
1  #define START_TASK_PRIO 1 //任务优先级
2  #define START_STK_SIZE 128 //任务堆栈大小
3  TaskHandle_t StartTask_Handler; //任务句柄
4  void start_task(void *pvParameters); //任务函数
5  #define LED0_TASK_PRIO 2 //任务优先级
6  #define LED0_STK_SIZE 128 //任务堆栈大小
7  TaskHandle_t Led0Task_Handler; //任务句柄
8  void led0_task(void *pvParameters); //任务函数
9  #define QUERY_TASK_PRIO 3 //任务优先级
10 #define QUERY_STK_SIZE 256 //任务堆栈大小
11 TaskHandle_t QueryTask_Handler; //任务句柄
12 void query_task(void *pvParameters); //任务函数
13 char InfoBuffer[1000]; //保存信息的数组 保
```

main()函数

```
1  int main(void)
2  {
3  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4); //设置系统中断优先级分组 4
4  delay_init(); //延时函数初始化
5  uart_init(115200); //初始化串口
6  LED_Init(); //初始化 LED
7  KEY_Init(); //初始化按键
8  LCD_Init(); //初始化 LCD
9
10 POINT_COLOR = RED;
11 LCD_ShowString(30,10,200,16,16,"ATK STM32F103/407");
12 LCD_ShowString(30,30,200,16,16,"FreeRTOS Examp 11-1");
13 }
```

```
14 LCD_ShowString(30,50,200,16,16,"Task Info Query");
15 LCD_ShowString(30,70,200,16,16,"ATOM@ALIENTEK");
16 LCD_ShowString(30,90,200,16,16,"2016/11/25");
17
18 //创建开始任务
19 xTaskCreate((TaskFunction_t )start_task, //任务函数
20             (const char* )"start_task", //任务名称
21             (uint16_t )START_STK_SIZE, //任务堆栈大小
22             (void* )NULL, //传递给任务函数的参数
23             (UBaseType_t )START_TASK_PRIO, //任务优先级
24             (TaskHandle_t* )&StartTask_Handler); //任务句柄
25 vTaskStartScheduler(); //开启任务调度 开启任务调度
}
```

在 main函数中我们主要完成硬件的初始化，在硬件初始化完成后创建了任务 start_task()并且开启了 FreeRTOS的任务调度

开始任务任务函数

```
1 void start_task(void *pvParameters)
2 {
3     taskENTER_CRITICAL(); //进入临界区
4     //创建LED0任务
5     xTaskCreate((TaskFunction_t )led0_task,
6                 (const char* )"led0_task",
7                 (uint16_t )LED0_STK_SIZE,
8                 (void* )NULL,
9                 (UBaseType_t )LED0_TASK_PRIO,
10                 (TaskHandle_t* )&Led0Task_Handler);
11     //创建QUERY任务
12     xTaskCreate((TaskFunction_t )query_task,
13                 (const char* )"query_task",
14                 (uint16_t )QUERY_STK_SIZE,
15                 (void* )NULL,
16                 (UBaseType_t )QUERY_TASK_PRIO,
17                 (TaskHandle_t* )&QueryTask_Handler);
18     vTaskDelete(StartTask_Handler); //删除开始任务
19     taskEXIT_CRITICAL(); //退出临界区
20 }
```

任务函数

```
1 //Led0任务函数
2 void led0_task(void *pvParameters)
3 {
4
5     while(1)
6     {
7         LED0=~LED0;
8         vTaskDelay(500); //延时500ms, 也就是500个时钟节拍
9     }
10 }
11
12 //query任务函数
13 void query_task(void *pvParameters)
14 {
15     u32 TotalRunTime;
16     UBaseType_t ArraySize,x;
17     TaskStatus_t *StatusArray;
18
19     //第一步: 函数uxTaskGetSystemState()的使用
20     printf("/*****第一步: 函数uxTaskGetSystemState()的使用*****/\r\n");
21     ArraySize=uxTaskGetNumberOfTasks(); //获取系统任务数量
22     StatusArray=pvPortMalloc(ArraySize*sizeof(TaskStatus_t)); //申请内存
23     if(StatusArray!=NULL) //内存申请成功
24     {
25         ArraySize=uxTaskGetSystemState((TaskStatus_t* )StatusArray, //任务信息存储数组
26                                         (UBaseType_t )ArraySize, //任务信息存储数组大小
27                                         (uint32_t* )&TotalRunTime); //保存系统总的运行时间
28         printf("TaskName\t\tPriority\t\tTaskNumber\t\t\r\n");
29         for(x=0;x<ArraySize;x++)
```

```
100 vTaskList(InfoBuffer); //获取所有任务的信息
```

```
101     printf("%s\r\n",InfoBuffer);           //通过串口打印所有任务的信息
102     printf("/*****结束*****/\r\n");
103     while(1)
104     {
105         LED1=~LED1;
106         vTaskDelay(1000);           //延时1s，也就是1000个时钟节拍
107     }
108 }
109
```

3.2任务运行时间信息统计实验

3.2.1实验要求

本实验设计四个任务：start_task、task1_task、task2_task和RunTimeStats_task，这四个任务的任功能如下：

start_task：用来创建其他3个任务。

task1_task：应用任务1，控制LED0灯闪烁，并且刷新LCD屏幕上指定区域的颜色。

task2_task：应用任务2，控制LED1灯闪烁，并且刷新LCD屏幕上指定区域的颜色。

RunTimeStats_task：获取按键值，当KEY_UP键按下以后就调用函数vTaskGetRunTimeStats()获取任务的运行时间信息，并且将其通过串口输出到调试助手上。

实验需要一个按键KEY_UP，用来获取系统中任务运行时间信息。

3.2.2程序代码

任务优先级、堆栈大小和句柄等的设置如下：

```
1 //任务优先级
2 #define START_TASK_PRIO      1
3 //任务堆栈大小
4 #define START_STK_SIZE       128
5 //任务句柄
6 TaskHandle_t StartTask_Handler;
7 //任务函数
8 void start_task(void *pvParameters);
9
10 //任务优先级
11 #define TASK1_TASK_PRIO      2
12 //任务堆栈大小
13 #define TASK1_STK_SIZE       128
14 //任务句柄
15 TaskHandle_t Task1Task_Handler;
16 //任务函数
17 void task1_task(void *pvParameters);
18
19 //任务优先级
20 #define TASK2_TASK_PRIO      3
21 //任务堆栈大小
22 #define TASK2_STK_SIZE       128
23 //任务句柄
24 TaskHandle_t Task2Task_Handler;
25 //任务函数
26 void task2_task(void *pvParameters);
27
28 //任务优先级
29 #define RUNTIMESTATS_TASK_PRIO  4
30 //任务堆栈大小
31 #define RUNTIMESTATS_STK_SIZE   128
32 //任务句柄
33 TaskHandle_t RunTimeStats_Handler;
34 //任务函数
35 void RunTimeStats_task(void *pvParameters);
36
37 char RunTimeInfo[400];           //保存任务运行时间信息
38
```

main函数

```
1 int main(void)
2 {
3     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4); //设置系统中断优先级分组4
4     delay_init();           //延时函数初始化
5     uart_init(115200);       //初始化串口
6 }
```

```

6      LED_Init();                                //初始化LED
7      KEY_Init();                                //初始化按键
8      LCD_Init();                                //初始化LCD
9
10     POINT_COLOR = RED;
11     LCD_ShowString(30,10,200,16,16,"ATK STM32F103/407");
12     LCD_ShowString(30,30,200,16,16,"FreeRTOS Examp 11-2");
13     LCD_ShowString(30,50,200,16,16,"Get Run Time Stats");
14     LCD_ShowString(30,70,200,16,16,"ATOM@ALIENTEK");
15     LCD_ShowString(30,90,200,16,16,"2016/11/25");
16
17     //创建开始任务
18     xTaskCreate((TaskFunction_t)start_task,        //任务函数
19                (const char*) "start_task",        //任务名称
20                (uint16_t) START_STK_SIZE,          //任务堆栈大小
21                (void*) NULL,                       //传递给任务函数的参数
22                (UBaseType_t) START_TASK_PRIO,      //任务优先级
23                (TaskHandle_t*) &StartTask_Handler); //任务句柄
24     vTaskStartScheduler();                          //开启任务调度
25 }

```

开始任务任务函数

```

1 void start_task(void *pvParameters)
2 {
3     taskENTER_CRITICAL();        //进入临界区
4     //创建TASK1任务
5     xTaskCreate((TaskFunction_t)task1_task,
6                (const char*) "task1_task",
7                (uint16_t) TASK1_STK_SIZE,
8                (void*) NULL,
9                (UBaseType_t) TASK1_TASK_PRIO,
10                (TaskHandle_t*) &Task1Task_Handler);
11     //创建TASK2任务
12     xTaskCreate((TaskFunction_t)task2_task,
13                (const char*) "task2_task",
14                (uint16_t) TASK2_STK_SIZE,
15                (void*) NULL,
16                (UBaseType_t) TASK2_TASK_PRIO,
17                (TaskHandle_t*) &Task2Task_Handler);
18     //创建RunTimeStats任务
19     xTaskCreate((TaskFunction_t)RunTimeStats_task,
20                (const char*) "RunTimeStats_task",
21                (uint16_t) RUNTIMESTATS_STK_SIZE,
22                (void*) NULL,
23                (UBaseType_t) RUNTIMESTATS_TASK_PRIO,
24                (TaskHandle_t*) &RunTimeStats_Handler);
25     vTaskDelete(StartTask_Handler); //删除开始任务
26     taskEXIT_CRITICAL();            //退出临界区
27 }
28

```

任务函数

```

1 //task1任务函数
2 void task1_task(void *pvParameters)
3 {
4     u8 task1_num=0;
5
6     POINT_COLOR = BLACK;
7
8     LCD_DrawRectangle(5,110,115,314); //画一个矩形
9     LCD_DrawLine(5,130,115,130);      //画线
10    POINT_COLOR = BLUE;
11    LCD_ShowString(6,111,110,16,16,"Task1 Run:000");
12    while(1)
13    {
14        task1_num++; //任务执行次数加1 注意task1_num1加到255的时候会清零!!
15        LED0=!LED0;

```

```
16     LCD_Fill(6,131,114,313,lcd_discolor[task1_num%14]); //填充区域
17     LCD_ShowxNum(86,111,task1_num,3,16,0x80);           //显示任务执行次数
18     vTaskDelay(1000);                                     //延时1s, 也就是1000个时钟节拍
19 }
20 }
21
22 //task2任务函数
23 void task2_task(void *pvParameters)
24 {
25     u8 task2_num=0;
26
27     POINT_COLOR = BLACK;
28
29     LCD_DrawRectangle(125,110,234,314); //画一个矩形
30     LCD_DrawLine(125,130,234,130);      //画线
31     POINT_COLOR = BLUE;
32     LCD_ShowString(126,111,110,16,16,"Task2 Run:000");
33     while(1)
34     {
35         task2_num++; //任务2执行次数加1 注意task1_num2加到255的时候会清零!!
36         LED1=!LED1;
37         LCD_ShowxNum(206,111,task2_num,3,16,0x80); //显示任务执行次数
38         LCD_Fill(126,131,233,313,lcd_discolor[13-task2_num%14]); //填充区域
39         vTaskDelay(1000); //延时1s, 也就是1000个时钟节拍
40     }
41 }
42
43 //RunTimeStats任务
44 void RunTimeStats_task(void *pvParameters)
45 {
46     u8 key=0;
47     while(1)
48     {
49         key=KEY_Scan(0);
50         if(key==WKUP_PRES)
51         {
52             memset(RunTimeInfo,0,400); //信息缓冲区清零
53             vTaskGetRunTimeStats(RunTimeInfo); //获取任务运行时间信息
54             printf("任务名\t\t\t\t运行时间\t\t\t\t运行所占百分比\r\n");
55             printf("%s\r\n",RunTimeInfo);
56         }
57         vTaskDelay(10); //延时10ms, 也就是1000个时钟节拍
58     }
59 }
60 }
```

