1 FreeRTOS 调研

我们对 FreeRTOS 的系统结构进行了一系列的调研之后，整理了以下重要的 API。

1.1 FreeRTOS v10.2.0 task.c 中任务创建 API

1.1.1 xTaskCreate

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

创建一个任务，并且自动为其分配**栈空间**和**数据空间**。

1.1.2 xTaskCreateStatic

```
TaskHandle_t xTaskCreateStatic( TaskFunction_t pvTaskCode,
                                const char * const pcName,
                                uint32_t ulStackDepth,
                                void *pvParameters,
                                UBaseType_t uxPriority,
                                StackType_t *pxStackBuffer,
                                StaticTask_t *pxTaskBuffer );
```

创建一个任务，此时需要程序员来提供一个**栈空间**和**数据空间**的地址。

1.1.3 xTaskCreateRestricted

```
BaseType_t xTaskCreateRestricted(TaskParameters_t   *
   pxTaskDefinition,
                                 TaskHandle_t *pxCreatedTask );
```

创建一个任务，此时需要程序员手动提供一个**栈空间**地址。

1.1.4 xTaskCreateRestrictedStatic

```
 BaseType_t xTaskCreateRestrictedStatic(
                         TaskParameters_t *pxTaskDefinition,
                         TaskHandle_t *pxCreatedTask );
```

创建一个任务，需要程序员手动提供**栈空间**地址。**数据空间**会被自动地动态分配。

1.1.5 vTaskAllocateMPURegions

```
void vTaskAllocateMPURegions( TaskHandle_t xTask,
        const MemoryRegion_t * const pxRegions );
```

用来为 **restricted task** 分配内存空间。

1.1.6 vTaskDelete

```
void vTaskDelete( TaskHandle_t xTask );
```

删除一个任务。

1.2 FreeRTOS v10.2.0 task.c 中任务控制 API

1.2.1 vTaskDelay

```
void vTaskDelay( const TickType_t xTicksToDelay );
```

以给定参数来延迟任务。

1.2.2 vTaskDelayUntil

```
void vTaskDelayUntil( TickType_t *pxPreviousWakeTime,
                      const TickType_t xTimeIncrement );
```

指定某个确定的时间点来解除阻塞。

1.2.3 xTaskAbortDelay

```
BaseType_t xTaskAbortDelay( TaskHandle_t xTask );
```

让任务从跳出阻塞状态回到它原来被调用的地方。

1.2.4 uxTaskPriorityGet

```
UBaseType_t uxTaskPriorityGet( const TaskHandle_t xTask );
```

获得任务的优先级。

1.2.5 eTaskGetState

```
eTaskState eTaskGetState( TaskHandle_t xTask );
```

获取任务的状态码，是一个枚举类型。

1.2.6 vTaskGetInfo

```
void vTaskGetInfo( TaskHandle_t xTask,
                   TaskStatus_t *pxTaskStatus,
                   BaseType_t xGetFreeStackSpace,
                   eTaskState eState );
```

获取任务的信息。

1.2.7 vTaskPrioritySet

```
void vTaskPrioritySet( TaskHandle_t xTask,
                       UBaseType_t uxNewPriority );
```

设置任务的优先级。

1.2.8 vTaskSuspend

```
void vTaskSuspend( TaskHandle_t xTaskToSuspend );
```

挂起任务。

1.2.9 vTaskResume

```
void vTaskResume( TaskHandle_t xTaskToResume );
```

继续执行被挂起的任务。

1.3 FreeRTOS v10.2.0 task.c 中程序调度 API

1.3.1 vTaskStartScheduler

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

启动任务调度程序。

1.3.2 vTaskEndScheduler

```
void vTaskEndScheduler( void );
```

停止任务调度程序，在处理完后，又重新从 **vTaskStartScheduler** 开始。

1.3.3 vTaskSuspendAll

```
void vTaskSuspendAll( void );
```

在不终止中断的情况下挂起调度程序。

1.3.4 xTaskResumeAll

```
BaseType_t xTaskResumeAll( void );
```

继续执行被挂起的调度程序。

1.3.5 xTaskGetTickCount

```
TickType_t xTaskGetTickCount( void );
```

获取从 ‘vTaskStartScheduler‘被调用到现在的毫秒数。

1.3.6 uxTaskGetNumberOfTasks

```
uint16_t uxTaskGetNumberOfTasks( void );
```

返回内核正在管理的任务的子总数目。

1.3.7 pcTaskGetName

```
char *pcTaskGetName( TaskHandle_t xTaskToQuery );
```

返回任务的名字。

1.3.8 xTaskGetHandle

```
TaskHandle_t xTaskGetHandle( const char *pcNameToQuery );
```

返回任务的句柄。

1.3.9 uxTaskGetStackHighWaterMark

```
UBaseType_t uxTaskGetStackHighWaterMark( TaskHandle_t xTask );
```

返回栈使用空间最大的那次数值。

1.3.10 xTaskCallApplicationTaskHook

```
BaseType_t xTaskCallApplicationTaskHook( TaskHandle_t xTask,
                                         void *pvParameter );
```

执行相应的钩子函数。

1.3.11 xTaskGetIdleTaskHandle

```
TaskHandle_t xTaskGetIdleTaskHandle( void );
```

返回空闲任务的句柄。

1.3.12 xTaskGetIdleRunTimeCounter

```
TickType_t xTaskGetIdleRunTimeCounter( void );
```

返回空闲任务的运行时间。

1.3.13 xTaskNotify

```
BaseType_t xTaskNotify( TaskHandle_t xTaskToNotify,
                        uint32_t ulValue,
                        eNotifyAction eAction );
```

发送广播。

1.3.14 xTaskNotifyWait

```
BaseType_t xTaskNotifyWait( uint32_t ulBitsToClearOnEntry,
                            uint32_t ulBitsToClearOnExit,
                            uint32_t *pulNotificationValue,
                            TickType_t xTicksToWait );
```

等待广播。

1.4 queue.c 文件中的重要函数

1.4.1 xQueueCreate

创建一个队列。

1.4.2 xQueueCreateStatic

使用静态方式创建一个队列。

1.4.3 xQueueSendToToFront

将一个元素送到队首。

1.4.4 xQueueSendToBack

将一个元素送到队尾。

1.4.5 xQueueSend

插入一个元素到队列中。

1.4.6 xQueueOverwrite

插入一个元素到队列中，如果队列已经满，则覆盖。

1.4.7 xQueueGenericSend

和 ‘xQueueSend‘效果相同，但是是推荐的 API。

1.4.8 xQueuePeek

获取一个队列中中的元素但是不删除其在队列中的位置。

1.4.9 xQueueReceive

获取一个队列中的元素，成功访问后就删除该元素在队列中的位置。

1.4.10 uxQueueMessagesWaiting

返回储存在队列中信息的数目。

1.4.11 uxQueueSpacesAvailable

返回队列中的可用空间。

1.4.12 vQueueDelete

删除队列。

> 上面的函数定义是为了在任务间传输数据，下面的函数用于 ‘co-routines‘（联合任务）

1.5 list.c 中的主要函数

1.5.1 vListInitialise

```
void vListInitialise( List_t * const pxList );
```

列表的初始化。

1.5.2 vListInitialiseItem

```
void vListInitialiseItem( ListItem_t * const pxItem );
```

列表项的初始化。

1.5.3 vListInsert

```
void vListInsert( List_t * const pxList,
                  ListItem_t * const pxNewListItem );
```

列表项插入。

1.5.4 vListInsertEnd

```
void vListInsertEnd( List_t * const pxList,
                     ListItem_t * const pxNewListItem );
```

列表项插入到最后。

1.5.5 uxListRemove

```
UBaseType_t uxListRemove( ListItem_t * const pxItemToRemove );
```

从列表中移除一个列表项。