第 3 章 FreeRTOS 时间管理

CHAPTER 3

FreeRTOS 时间管理包括时间片轮转以及任务延时。

本章通过对 FreeRTOS 时间管理和相关原理的介绍，并以实例开发帮助读者掌握 FreeRTOS 的时间管理。

视频

3.1 时间片轮转

通过学习本节内容，读者应掌握 FreeRTOS 的时间片轮转的使用方法。

3.1.1 开发原理

FreeRTOS 的时间片轮转回调函数。

函数原型：

```
void vApplicationTickHook(void);
```

vApplicationTickHook()是用户启用时间片轮转时定义的回调函数。用户如果需要使用此功能，那么需要将 FreeRTOSConfig. h 中的 configUSE_TICK_HOOK 宏定义为 1。configUSE_TICK_HOOK 用于控制是否启用时间片轮转回调函数。宏定义为 1 时启用，宏定义为 0 时不启用。时间片轮转回调函数在 xTaskIncrementTick()函数中被调用。因为 xTaskIncrementTick()函数在 PendSV_Handler()中断函数中被调用，所以时间片轮转回调函数执行的时间必须很短。

参数描述：

无。

3.1.2 开发步骤

(1) 在 main. c 中创建一个任务，任务 vTaskPrint 完成计算系统运行时间和打印信息的功能。

```
void vTaskPrint(void * pvParameters)
{
    uint32_t TickNum_Minute = 0;
    uint32_t TickNum_old = 0;
    while(1)
    {
        if(TickNum/configTICK_RATE_HZ != 0 && TickNum_old != TickNum/configTICK_RATE_HZ)
        {
```

```
            TickNum_old = TickNum/configTICK_RATE_HZ;
            if(TickNum_old > 59)
            {
                TickNum_Minute += 1;
                TickNum_old = 0;
                TickNum -= 60 * configTICK_RATE_HZ;
            }
             printf("The current system runs for %d minute %d seconds\n",TickNum_Minute,
TickNum/configTICK_RATE_HZ);
        }
    }
}
```

(2) 定义任务创建函数,代码如下:

```
void Task_Cerate(void)
{
    xTaskCreate(vTaskPrint,                          // 任务指针
                "vTaskPrint",                        // 任务描述
                100,                                 // 堆栈深度
                NULL,                                // 给任务传递的参数
                2,                                   // 任务优先级
                &TaskPrint_Handle                    // 任务句柄
                );
}
```

(3) 在 main()函数中调用创建任务函数和启动调度器函数,代码如下:

```
#include "FreeRTOS.h"
#include "task.h"
#include "bsp_uart.h"

TaskHandle_t TaskPrint_Handle;                       // 定义任务句柄

void vTaskPrint(void * pvParameters);                // 声明任务
void Task_Cerate(void);

uint32_t TickNum;                                    // 定义变量,用来记录时间片轮转次数
int main(void)
{
    UART1_Init();
    Task_Cerate();
    vTaskStartScheduler();                           // 启动调度器函数
    while(1);
}
```

(4) 在时间片轮转回调函数中完成对时间片轮转的计数,代码如下:

```
// 时间片轮转回调函数
void vApplicationTickHook(void)
{
    TickNum++;
}
```

3.1.3 运行结果

下载程序,程序运行后会通过串口打印出系统的运行时间。

练习

(1) 简述启用时间片轮转时定义的回调函数所需要的条件。
(2) 简述实现时间片轮转回调任务的过程。
(3) 通过实例讨论时间片轮转的优先级。


视频

3.2 任务延时

通过学习本节内容,读者应掌握 FreeRTOS 的时间管理函数的用法。

3.2.1 开发原理

1. FreeRTOS 的时钟节拍

任何操作系统都需要提供一个时钟节拍,以供系统处理诸如延时、超时等与时间相关的事件。时钟节拍是特定的周期性中断,这个中断可以看作系统心跳。中断之间的时间间隔取决于不同的应用,一般是 1～100ms。时钟的节拍中断使得内核可以将任务延迟若干时钟节拍,以及当任务等待事件发生时,提供等待超时等依据。时钟节拍频率越高,系统的额外开销就越大。

FreeRTOS 的系统时钟节拍可以在配置文件 FreeRTOSConfig.h 中设置:

```
#define configTICK_RATE_HZ (( TickType_t ) 1000)
```

如上所示的宏定义配置表示系统时钟节拍是 1kHz。

2. FreeRTOS 的获取系统当前运行的时钟节拍数函数

函数原型:

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

功能:获取系统当前运行的时钟节拍数。注意:这个函数不能在中断中使用。

参数描述:

无。

返回值:

返回从调用启动调度器函数以来的系统时钟节拍数。

3. FreeRTOS 的从中断中获取系统当前运行的时钟节拍数函数

函数原型:

```
volatile TickType_t xTaskGetTickCountFromISR(void);
```

功能:从中断中获取系统当前的时钟节拍数。

参数描述:

无。

返回值:

返回从调用启动调度器函数以来的系统时钟节拍数。

4. FreeRTOS 的相对延时函数

函数原型:

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

功能：用于任务相对延时。用户如果需要使用相对延时函数需要将 FreeRTOSConfig.h 中的 include_vTaskDelay 宏定义为 1。调用 vTaskDelay()函数后，任务会进入阻塞状态，持续时间由 vTaskDelay()函数的参数 xTicksToDelay 指定，单位是系统节拍时钟周期。常量 portTICK_RATE_MS 用来辅助计算真实时间，此值是系统节拍时钟中断的周期，单位是毫秒，其分辨率为一个时钟周期。注意：vTaskDelay()指定的延时时间是从调用 vTaskDelay()后开始计算的相对时间，也就是说，vTaskDelay()至少会延时指定的延时时间。vTaskDelay 函数不适合延时周期性任务，此函数会受到中断的影响。

参数描述：

xTicksToDelay——延时时间总数，单位是系统时钟节拍周期，范围为 1～0xFFFFFFFF。延迟时间的最大值在 portmacro.h 文件中有定义：

```
typedef uint32_t TickType_t;
#define portMAX_DELAY ( TickType_t )0xffffffffUL
```

即延迟时间的范围是 1～0xFFFFFFFF。

5. FreeRTOS 的绝对延时函数

函数原型：

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

功能：用于任务绝对延时。用户如果需要使用绝对延时函数需要将 FreeRTOSConfig.h 中的 INCLUDE_vTaskDelayUntil 宏定义为 1。调用 vTaskDelayUntil()后，任务会进入阻塞状态，将任务延迟到指定的时间。适用于周期性的任务，此函数可以保持稳定的周期。

参数描述：

pxPreviousWakeTime——指向一个变量的指针，该变量保存任务上次解除阻塞的时间。变量必须在第一次使用之前用当前时间初始化。在此之后，变量将在 vTaskDelayUntil()中自动更新。

xTimeIncrement——周期性延时时间。任务将在时间上被解除阻塞(*pxPreviousWakeTime+xTimeIncrement)。使用相同的 xTimeIncrement 参数值调用 vTaskDelayUntil 将导致任务以固定的间隔执行。

FreeRTOS 的相对延时函数与绝对延时函数的区别：

相对延时函数调用时至少延时给定的延时时间，而绝对延时函数刚好延时给定的延时时间。相对延时函数不适用于周期性任务，绝对延时函数更适合周期性任务和延时精度较高的任务。

6. FreeRTOS 的空闲任务回调函数

函数原型：

```
void vApplicationIdleHook(void);
```

vApplicationIdleHook()是用户启用空闲任务回调函数时定义的回调函数，在系统运行后，当没有用户任务运行时，空闲任务运行时会调用空闲任务回调函数。如果需要使用此功能，则需要将 FreeRTOSConfig.h 中的 configUSE_IDLE_HOOK 宏定义为 1。configUSE_IDLE_HOOK 是控制是否启用空闲任务回调函数。宏定义为 1 时启用，宏定义为 0 时不启用。空闲任务回调函数常用来实现低功耗模式的开启。使用空闲任务回调函数一定要切记不可以在空闲任务回调函数中使用任何能引起任务阻塞的函数。

参数描述：

无。

3.2.2 开发步骤

(1) 在 main.c 中创建 4 个任务：任务 vTaskLED 完成 LED 闪烁功能；任务 vTaskKEY 完成按键检测和打印功能；任务 vTaskDD 完成统计 vTaskDelay 延时时间的功能；任务 vTaskDUD 完成统计 vTaskDelayUntil 延时时间的功能。

```
void vTaskKEY(void * pvParameters)
{
    while(1)
    {
        if(!KEY0_Read)
        {
            printf("Idle tasks run %d times\n",IdleNumber);
        }
        if(!KEY1_Read)
        {
            printf("Delay of %d system beats\r\n",DataDelay);
        }
        if(!KEY2_Read)
        {
            printf("DelayUntil of %d system beats\r\n",DataDelayUntil);
        }
        vTaskDelay(30/portTICK_PERIOD_MS);
    }
}

void vTaskDD(void * pvParameters)
{
    TickType_t Data_Old = 0;
    TickType_t Data_New = 0;
    while(1)
    {
        Data_Old = xTaskGetTickCount();                    // 获取系统当前运行的时钟节拍数
        vTaskDelay(20/portTICK_PERIOD_MS);
        Data_New = xTaskGetTickCount();                    // 获取系统当前运行的时钟节拍数
        DataDelay = Data_New - Data_Old;                   // 计算任务延时时间
    }
}

void vTaskDUD(void * pvParameters)
{
    TickType_t Data_Old = 0;
    TickType_t Data_New = 0;
    TickType_t xLastWakeTime;
    TickType_t xFrequency = 20;
    xLastWakeTime = xTaskGetTickCount();                   // 获取系统当前运行的时钟节拍数
    while(1)
    {
        Data_Old = xTaskGetTickCount();                    // 获取系统当前运行的时钟节拍数
        vTaskDelayUntil(&xLastWakeTime,xFrequency);
        Data_New = xTaskGetTickCount();                    // 获取系统当前运行的时钟节拍数
```

```
        DataDelayUntil = Data_New - Data_Old;          // 计算任务延时时间
    }
}

void vTaskLED(void * pvParameters)
{
    while(1)
    {
        LED1_OFF;
        LED2_ON;
        vTaskDelay(300/portTICK_PERIOD_MS);
        LED1_ON;
        LED2_OFF;
        vTaskDelay(300/portTICK_PERIOD_MS);
    }
}
```

(2) 定义任务创建函数,代码如下:

```
void Task_Cerate(void)
{
    xTaskCreate(vTaskKEY,                              // 任务指针
                "vTaskKEY",                            // 任务描述
                100,                                   // 堆栈深度
                NULL,                                  // 给任务传递的参数
                4,                                     // 任务优先级
                &TaskKEY_Handle                        // 任务句柄
                );
    xTaskCreate(vTaskDD,                               // 任务指针
                "vTaskDD",                             // 任务描述
                100,                                   // 堆栈深度
                NULL,                                  // 给任务传递的参数
                3,                                     // 任务优先级
                &TaskDD_Handle                         // 任务句柄
                );
    xTaskCreate(vTaskDUD,                              // 任务指针
                "vTaskDUD",                            // 任务描述
                100,                                   // 堆栈深度
                NULL,                                  // 给任务传递的参数
                2,                                     // 任务优先级
                &TaskDUD_Handle                        // 任务句柄
                )
    xTaskCreate(vTaskLED,                              // 任务指针
                "vTaskLED",                            // 任务描述
                100,                                   // 堆栈深度
                NULL,                                  // 给任务传递的参数
                1,                                     // 任务优先级
                &TaskLED_Handle                        // 任务句柄
                );
}
```

(3) 在 main()函数中调用任务创建函数和启动调度器函数,代码如下:

```
#include "FreeRTOS.h"
#include "task.h"
#include "bsp_uart.h"
#include "bsp_tim.h"
```

```
#include "bsp_key.h"
#include "Bsp_Led.h"

TaskHandle_t TaskKEY_Handle;                    //定义任务句柄
TaskHandle_t TaskLED_Handle;                    //定义任务句柄
TaskHandle_t TaskDD_Handle;                     //定义任务句柄
TaskHandle_t TaskDUD_Handle;                    //定义任务句柄

uint32_t DataDelay;
uint32_t DataDelayUntil;
uint32_t IdleNumber;

void vTaskLED(void * pvParameters);             //声明任务
void vTaskKEY(void * pvParameters);             //声明任务
void vTaskDD(void * pvParameters);              //声明任务
void vTaskDUD(void * pvParameters);             //声明任务
void Task_Cerate(void);

int main(void)
{
    LED_Init();
    KEY_Init();
    UART1_Init();
    TIM3_Init(50,84-1);
    Task_Cerate();
    vTaskStartScheduler();                      //启动调度器函数
    while(1);
}
```

(4) 在空闲任务回调函数中统计空闲任务运行次数,代码如下:

```
//空闲状态回调函数
void vApplicationIdleHook(void)
{
    IdleNumber++;     //当用户任务都阻塞之后,空闲任务开始运行,统计空闲任务运行次数
}
```

(5) 开启定时器 3,对系统运行节拍数进行统计,代码如下:

```
volatile uint32_t ulHighFrequencyTimerTicks = 0UL;
TIM_HandleTypeDef TIM3_Handle;

void TIM3_Init(uint16_t arr,uint16_t psc)
{
    TIM3_Handle.Instance = TIM3;                                    //定时器 3
    TIM3_Handle.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;        //时钟分频因子
    TIM3_Handle.Init.CounterMode = TIM_COUNTERMODE_UP;              //计数方向
    TIM3_Handle.Init.Period = arr;                                  //自动重装载值
    TIM3_Handle.Init.Prescaler = psc;                               //分频系数
    HAL_TIM_Base_Init(&TIM3_Handle);
    HAL_TIM_Base_Start_IT(&TIM3_Handle);
}

void HAL_TIM_Base_MspInit(TIM_HandleTypeDef * htim)
{
    if(htim->Instance == TIM3)
    {
```

```
        __HAL_RCC_TIM3_CLK_ENABLE();
        HAL_NVIC_SetPriority(TIM3_IRQn,1,3);
        HAL_NVIC_EnableIRQ(TIM3_IRQn);
    }

}

void TIM3_IRQHandler(void)
{
    HAL_TIM_IRQHandler(&TIM3_Handle);
}


void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef * htim)
{
    ulHighFrequencyTimerTicks++;
}
```

(6) 在 FreeRTOSConfig.h 中定义外部变量和时间统计函数,代码如下:

```
#ifdef __CC_ARM
    #include <stdint.h>
    extern uint32_t SystemCoreClock;
    extern volatile uint32_t ulHighFrequencyTimerTicks;            // 定义外部变量
#endif

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS 1
#define configUSE_STATS_FORMATTING_FUNCTIONS 1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() (ulHighFrequencyTimerTicks = 0ul)
#define portGET_RUN_TIME_COUNTER_VALUE() ulHighFrequencyTimerTicks
```

3.2.3 运行结果

下载程序,LED 呈现流水灯状态,按下 KEY0 键后通过串口打印出空闲任务的运行次数,按下 KEY1 键后通过串口打印出 vTaskDelay 的延时时间,按下 KEY2 键后通过串口打印出 vTaskDelayUntil 的延时时间。

练习

(1) 简述 FreeRTOS 的相对延时函数与绝对延时函数的区别。

(2) 使用空闲任务函数时需要注意什么?