STM32 ???? ? 13 ??FreeRTOS ????????

STM32 ???? ? 13 ??FreeRTOS ????????

? ????

1. 理解 **RTOS** 基本概念 - 掌握任務調度、同步機制
2. 配置 **FreeRTOS** - 在 STM32 上運行多任務
3. 實現任務通訊 - 隊列、信號量、互斥鎖

? RTOS ??

RTOS ????

RTOS (Real-Time Operating System) 是實時操作系統。用途:

- 任務調度：決定哪個任務何時運行
- 優先級管理：高優先級任務優先執行
- 同步機制：保護共享資源
- 定時準確：毫秒級任務切換

FreeRTOS ????

概念	說明
Task (任務)	獨立的程式流，有各自的棧和優先級
Priority (優先級)	0 ~ configMAX_PRIORITIES-1，越高越優先
Scheduler (調度器)	根據優先級決定運行哪個任務

概念	說明
Queue (隊列)	任務間安全傳遞訊息
Semaphore (信號量)	控制資源存取
Mutex (互斥鎖)	防止優先級反轉

STM32 ?? FreeRTOS

優勢:

- □ □□□□□□
- □ □□□□□□□□ 3KB)
- □ □□□□□□□□□□
- □ □□□□□□□

?? CubeMX ??

?? 1??? FreeRTOS

1. 左側選 **Middleware** → 搜尋 **FreeRTOS**
2. **Interface**: CMSIS_V2

?? 2?????

1. **System Timers and Clocks** → **SysTick timer**
2. 設定 **SysTick** 為 1ms（FreeRTOS 心跳)

?? 3???????

FreeRTOS:

- **TOTAL_HEAP_SIZE**: 4096 bytes (根據需要調整)
- **configMAX_PRIORITIES**: 5 (最多 5 級優先級)

?? 4??????

? ?????

FreeRTOS ??????main.c

```
/* STM32 Lesson 13 - FreeRTOS
 * 功能: 多任務調度、任務通訊
 * 難度: 高級
 */

#include "main.h"
#include "cmsis_os.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);

/* 任務宣告 */
void Task1_Start(void *argument);
void Task2_Start(void *argument);
void Task3_Start(void *argument);

UART_HandleTypeDef huart1;

/* 隊列宣告 */
osMessageQueueId_t queueHandle;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== FreeRTOS Multi-Task Demo ===\r\n");
```

```
  /* 創建隊列 */
  queueHandle = osMessageQueueNew(10, sizeof(uint32_t), NULL);

  /* 創建任務 */
  osThreadNew(Task1_Start, NULL, NULL);  /* 優先級: osPriorityNormal */
  osThreadNew(Task2_Start, NULL, NULL);  /* 優先級: osPriorityNormal */
  osThreadNew(Task3_Start, NULL, NULL);  /* 優先級: osPriorityHigh */

  /* 啟動 RTOS 調度器 */
  osKernelStart();

  while (1);
}

/**
  * @brief Task 1: 每 1 秒發送一個數值到隊列
  */
void Task1_Start(void *argument)
{
  uint32_t counter = 0;

  while (1)
  {
    UART_Print("Task1: Sending data %lu\r\n", counter);
    osMessageQueuePut(queueHandle, &counter, 0, 0);
    counter++;
    osDelay(1000);  /* 延遲 1 秒 */
  }
}

/**
  * @brief Task 2: 從隊列接收資料並列印
  */
void Task2_Start(void *argument)
{
  uint32_t rx_data;
  osStatus_t status;

  while (1)
```

```
  {
    status = osMessageQueueGet(queueHandle, &rx_data, NULL, osWaitForever);
    if (status == osOK)
    {
      UART_Print("Task2: Received %lu\r\n", rx_data);
    }
  }
}

/**
  * @brief Task 3□□□□□□□□□□ 500ms □□□□
  */
void Task3_Start(void *argument)
{
  while (1)
  {
    UART_Print("Task3: High Priority Task Running\r\n");
    osDelay(500);  /* □□ 500ms */
  }
}

void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    Error_Handler();

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
                              | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    Error_Handler();
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
    Error_Handler();
}
```

```
void Error_Handler(void)
{
  while(1);
}
```

? ????

UART ????

```
=== FreeRTOS Multi-Task Demo ===
Task3: High Priority Task Running
Task1: Sending data 0
Task2: Received 0
Task3: High Priority Task Running
Task1: Sending data 1
Task2: Received 1
Task3: High Priority Task Running
...
```

??????

1. **Task1** 每 1000ms 發送計數值到隊列
2. **Task2** 在隊列中等待，收到資料後立即列印
3. **Task3** 每 500ms 執行一次（優先級最高）
4. 調度器根據優先級和時間片進行任務切換

? ????

?? CMSIS-RTOS 2 API

函數	用途
`osThreadNew()`	創建任務
`osDelay()`	任務延遲（毫秒）

函數	用途
`osMessageQueueNew()`	創建訊息隊列
`osMessageQueuePut()`	訊息入隊
`osMessageQueueGet()`	訊息出隊
`osSemaphoreNew()`	創建信號量
`osMutexNew()`	創建互斥鎖
`osKernelStart()`	啟動 RTOS 調度器

????????

```
/* 兩個任務的同步範例 */
osSemaphoreId_t semaphore;

void Task_Producer(void *argument)
{
  while (1)
  {
    /* 進行一些工作 */
    UART_Print("Producer: Data ready\r\n");

    /* 釋放信號量，喚醒消費者 */
    osSemaphoreRelease(semaphore);

    osDelay(1000);
  }
}

void Task_Consumer(void *argument)
{
  while (1)
  {
    /* 等待信號量被釋放 */
    if (osSemaphoreAcquire(semaphore, osWaitForever) == osOK)
    {
      UART_Print("Consumer: Processing data\r\n");
    }
  }
```

```
}
```

??????????

```
/* 多個任務共享 UART 列印 */
osMutexId_t uart_mutex;

void Safe_UART_Print(const char *format, ...)
{
  /* 獲取互斥鎖 */
  osMutexAcquire(uart_mutex, osWaitForever);

  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);

  /* 釋放互斥鎖 */
  osMutexRelease(uart_mutex);
}
```

? ????

1. ?????Event Flags?

用於多個任務的事件同步

```
osEventFlagsId_t event_flags;

/* 設置標誌 */
osEventFlagsSet(event_flags, 0x01);

/* 等待標誌 */
```

```
osEventFlagsWait(event_flags, 0x01, osFlagsWaitAny, osWaitForever);
```

2. ??????Software Timer?

實現週期性任務或延遲執行

```
void Timer_Callback(void *argument)
{
  UART_Print("Timer fired!\r\n");
}

osTimerId_t timer = osTimerNew(Timer_Callback, osTimerPeriodic, NULL, NULL);
osTimerStart(timer, 1000);  /* 每 1000ms 執行一次 */
```

3. ?????Task Notification?

輕量級的任務喚醒機制（比隊列和信號量快）

```
/* Task A 喚醒 Task B */
osThreadFlagsSet(task_b_id, 0x01);

/* Task B 等待通知 */
osThreadFlagsWait(0x01, osFlagsWaitAny, osWaitForever);
```

4. ???????

RTOS 提供的安全動態分配

```
void *pvPortMalloc(size_t xSize);  /* 動態分配 */
void vPortFree(void *pv);          /* 動態釋放 */
```

? ????

???????

```
/* 獲取任務資訊 */
osThreadState_t state = osThreadGetState(task_id);


/* 狀態值 */
switch (state)
{
  case osThreadInactive:
    UART_Print("Task is inactive\r\n");
    break;
  case osThreadReady:
    UART_Print("Task is ready to run\r\n");
    break;
  case osThreadRunning:
    UART_Print("Task is currently running\r\n");
    break;
  case osThreadBlocked:
    UART_Print("Task is blocked\r\n");
    break;
}
```

? ?? 13 ??????

????

□ STM32嵌入式開發完整知識體系
□ □□□□□□□□□
□ 13個主題 50+個實踐範例
□ □□□□□□□□□□

??????

- □□ 無人機飛控系統 - 使用RTOS進行多傳感器融合
- □□ 物聯網 **IoT** 設備 - 集成通訊協議和雲連接
- □□ 工業控制系統 - MODBUS網路和實時控制
- □□ □□□□□□ - 複雜的多軸運動控制
- □□ 汽車電子應用 - CAN網路和診斷協定

? ????

- **FreeRTOS** 官方文檔 : https://www.freertos.org/
- **CMSIS-RTOS 2** 標準 : ARM 官方規範
- 嵌入式實時系統設計 : 相關教科書

STM32 完整系列教學圓滿完成！祝您在嵌入式開發中取得成就！

Revision #1
Created 2026-04-01 02:06:33 UTC by TaipeiTechRacing
Updated 2026-04-01 02:12:26 UTC by TaipeiTechRacing