

[STM32 FreeRTOS] 자료 Review#4



이지훈

2020. 3. 8. 16:59

이웃추가

※ 내용에 오류가 있을 수 있습니다. 오류에 대해서는 Feedback 부탁 드리겠습니다.

https://github.com/eziya/STM32F4_HAL_FREERTOS_LAB

**eziya/STM32F4_HAL_FREERTOS_LAB**

FreeRTOS Testing for STM32F4. Contribute to eziya/STM...

github.com

ST 에서 제공하는 자료들을 기반으로 FreeRTOS 의 기본 특성에 대해서 정리하여 보고 해당 자료의 Lab 코드들을 테스트해 보고자 합니다.

Event Group

세마포어, 뮉텍스 등이 하나의 이벤트만을 사용한다면 Event Group 은 여러개의 이벤트를 조합하여 동기화 가능. 현재 CMSIS-RTOS API v1 에서는 미지원하며 FreeRTOS API 사용 필요.

[LAB#10]

프로젝트 구성

- 3개의 Task 를 생성한다.
- EventSender1, EventSender2, EventReceiver 태스크는 모두 osPriorityNormal 로 우선순위 설정
- Event Group 용 변수와 Flag 를 설정한하고 생성한다.
- EventSender1 Task 는 1초, EventSender2 Task 는 3초마다 이벤트를 전송한다.
- EventReceiver Task 는 2초간 Event Group 을 수신 대기하고 Event 가 수신되면 수신한 flag 값을 확인한다.

동작

- EventSender1 은 1초, EventSender2 는 3초마다 이벤트를 생성하고 EventReceiver 는 2개의 이벤트가 모두 수신되는 것을 2초간 기다리기 때문에 최초 2초 시점이 지난 시점에는

evtFlag1 수신 메시지만 출력하고, 4초 시점이 지난 시점에는 evtFlag2 도 수신하면서 모든 이벤트 플래그를 수신하였다고 출력한다.

```
/* USER CODE BEGIN Header_StartEventReceiverTask */
/**
 * @brief Function implementing the EventReceiver thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartEventReceiverTask */
void StartEventReceiverTask(void const *argument)
{
    /* USER CODE BEGIN StartEventReceiverTask */
    uint32_t result;
    /* Infinite loop */
    for (;;)
    {
        result = xEventGroupWaitBits(
            evtGrpHandle, //handle
            (evtFlag1 | evtFlag2), //flags to wait
            pdTRUE, //clear flags
            pdTRUE, //wait all flags
            2000);

        if ((result & (evtFlag1 | evtFlag2)) == (evtFlag1 | evtFlag2))
        {
            printf("evtFlag1 | evtFlag2 set\n");
        }
        else
        {
            if (result & evtFlag1)
            {
                printf("evtFlag1 set\n");
            }
            else if (result & evtFlag2)
            {
                printf("evtFlag2 set\n");
            }
            else
            {
                printf("None is set\n");
            }
        }
    }
}
```

```

        osDelay(1);
    }
    /* USER CODE END StartEventReceiverTask */
}

/* USER CODE BEGIN Header_StartEventSender1Task */
/**
 * @brief Function implementing the EventSender1 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartEventSender1Task */
void StartEventSender1Task(void const *argument)
{
    /* USER CODE BEGIN StartEventSender1Task */
    /* Infinite loop */
    for (;;)
    {
        printf("Sender1 set evtFlag1\n");
        xEventGroupSetBits(evtGrpHandle, evtFlag1);
        osDelay(1000);
    }
    /* USER CODE END StartEventSender1Task */
}

/* USER CODE BEGIN Header_StartEventSender2Task */
/**
 * @brief Function implementing the EventSender2 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartEventSender2Task */
void StartEventSender2Task(void const *argument)
{
    /* USER CODE BEGIN StartEventSender2Task */
    /* Infinite loop */
    for (;;)
    {
        printf("Sender2 set evtFlag2\n");
        xEventGroupSetBits(evtGrpHandle, evtFlag2);
        osDelay(3000);
    }
    /* USER CODE END StartEventSender2Task */
}

```

```
/* USER CODE END StartEventSender2Task */
```

```
}
```

```

Console Problems Executables Debugger Console Memory SWV ITM Data Console
포트 0
Don't remove this printf for debugging.
Sender2 set evtFlag2
Sender1 set evtFlag1
evtFlag1 | evtFlag2 set
Sender1 set evtFlag1
Sender1 set evtFlag1
evtFlag1 set
Sender2 set evtFlag2
Sender1 set evtFlag1
evtFlag1 | evtFlag2 set
Sender1 set evtFlag1
Sender1 set evtFlag1
evtFlag1 set
Sender2 set evtFlag2
Sender1 set evtFlag1
evtFlag1 | evtFlag2 set
Sender1 set evtFlag1
Sender1 set evtFlag1
evtFlag1 set

```

Signal

FreeRTOS task 들은 32비트 notification value 들을 갖고 있으며, Signal API 를 이용해서 task 를 지정하여 메시지 전송이 가능

Task notification 기능은 큐, 세마포어, 이벤트 보다 가볍고 빠름.

하지만, Task notification 은 한번에 한 task 에게만 notification 이 가능하다는 제약이 존재함

[LAB#11]

프로젝트 구성

- 4개의 Task 를 생성한다.
- Sender1, Sender2, Receiver1, Receiver2 태스크는 모두 osPriorityNormal 로 우선순위 설정
- Sender1, Sender2 Task 는 1초 간격으로 Receiver1, Receiver2 에 시그널을 전송한다.
- Receiver1, Receiver2 는 Signal 을 대기하다가 특정 시그널이 들어오면 시그널 수신 여부를 printf 로 출력한다.

동작

- Sender1 이 Receiver1 에게 메시지를 전송하는 경우는 0x11, Receiver2 에게 메시지를 전송하는 경우는 0x12 를 전송한다.

- Sender2 가 Receiver1 에게 메시지를 전송하는 경우는 0x21, Receiver2 에게 메시지를 전송하는 경우는 0x22 를 전송한다.
- 메시지를 수신하면 수신된 메시지를 확인해서 Sender 를 파악할 수 있다.

```

/* USER CODE BEGIN Header_StartSender1Task */
/**
 * @brief Function implementing the Sender1 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartSender1Task */
void StartSender1Task(void const *argument)
{
    /* USER CODE BEGIN StartSender1Task */
    /* Infinite loop */
    for (;;)
    {
        osSignalSet(Receiver1Handle, 0x11);
        osDelay(1000);
        osSignalSet(Receiver2Handle, 0x12);
        osDelay(1000);
    }
    /* USER CODE END StartSender1Task */
}

/* USER CODE BEGIN Header_StartSender2Task */
/**
 * @brief Function implementing the Sender2 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartSender2Task */
void StartSender2Task(void const *argument)
{
    /* USER CODE BEGIN StartSender2Task */
    /* Infinite loop */
    for (;;)
    {
        osSignalSet(Receiver1Handle, 0x21);
        osDelay(1000);
        osSignalSet(Receiver2Handle, 0x22);
        osDelay(1000);
    }
}

```

```

/* USER CODE END StartSender2Task */
}

/* USER CODE BEGIN Header_StartReceiver1Task */
/**
 * @brief Function implementing the Receiver1 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartReceiver1Task */
void StartReceiver1Task(void const *argument)
{
    /* USER CODE BEGIN StartReceiver1Task */
    osEvent evt;
    /* Infinite loop */

    for (;;)
    {
        evt = osSignalWait(0x11 | 0x21, 100);
        if (evt.status == osEventSignal)
        {
            if ((evt.value.signals & 0x11) == 0x11)
            {
                printf("Receiver1: Notify from Sender1\n");
            }

            if ((evt.value.signals & 0x21) == 0x21)
            {
                printf("Receiver1: Notify from Sender2\n");
            }
        }
    }

    /* USER CODE END StartReceiver1Task */
}

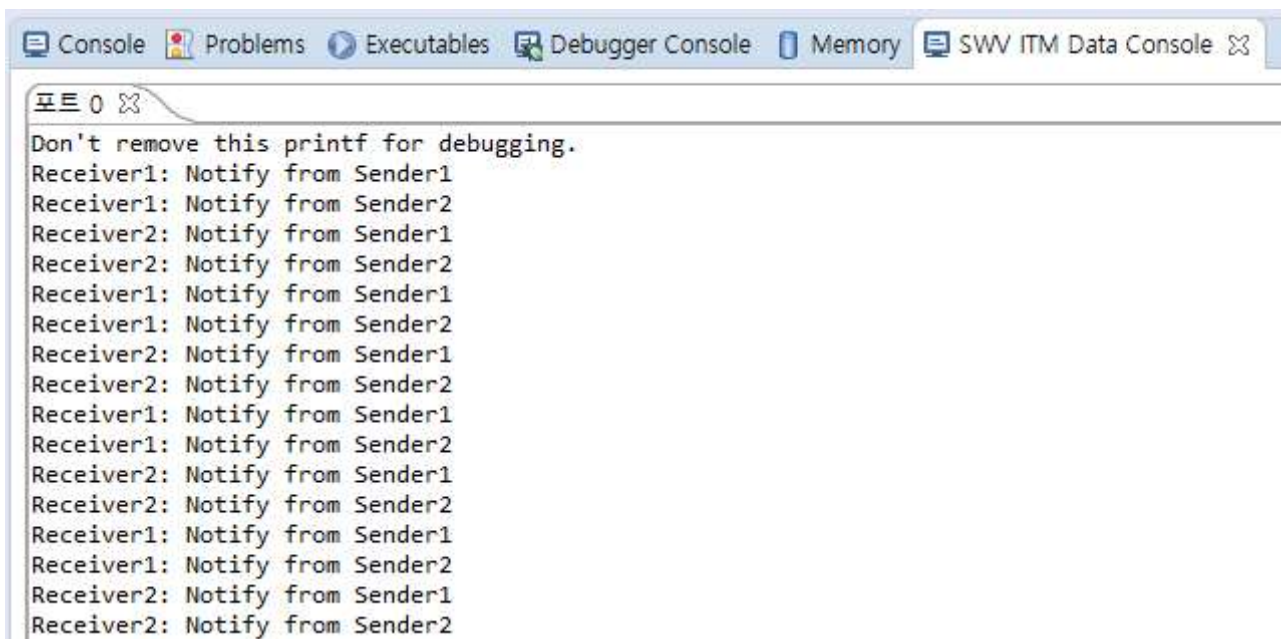
/* USER CODE BEGIN Header_StartReceiver2Task */
/**
 * @brief Function implementing the Receiver2 thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartReceiver2Task */

```

```
void StartReceiver2Task(void const *argument)
{
    /* USER CODE BEGIN StartReceiver2Task */
    osEvent evt;
    /* Infinite loop */

    for (;;)
    {
        evt = osSignalWait(0x12 | 0x22, 100);
        if (evt.status == osEventSignal)
        {
            if ((evt.value.signals & 0x12) == 0x12)
            {
                printf("Receiver2: Notify from Sender1\n");
            }

            if ((evt.value.signals & 0x22) == 0x22)
            {
                printf("Receiver2: Notify from Sender2\n");
            }
        }
    }
    /* USER CODE END StartReceiver2Task */
}
```



The screenshot shows the 'SWV ITM Data Console' window in an IDE. The console output displays a series of messages from two receivers, Receiver1 and Receiver2, each receiving notifications from two different senders, Sender1 and Sender2. The messages are: 'Receiver1: Notify from Sender1', 'Receiver1: Notify from Sender2', 'Receiver2: Notify from Sender1', and 'Receiver2: Notify from Sender2'. This sequence repeats multiple times, demonstrating the interleaved execution of the two tasks. Above the messages, a note states: 'Don't remove this printf for debugging.'

Stack Overflow Hook

FreeRTOS 는 각 task 별로 할당된 Stack 영역에 overflow 가 발생하였는지 여부를 체크하는 기능을 제공한다.

configCHECK_FOR_STACK_OVERFLOW 가 1인 경우에는 context switch 이 발생시마다 stack top 주소 변수와 현재 stack 주소 변수를 비교, 2인 경우에는 stack 을 0xA5 로 채운 후에 패턴이 다른 데이터로 overwrite 되었는지 검사하여 체크

overflow 가 발생한 경우에는 vApplicationStackOverflowHook callback 호출

태스크 생성이후 최소 stack 의 크기를 조회 시에는 uxTaskGetStackHighWaterMark 함수를 호출하여 최소 값 확인 가능

[LAB#12]

프로젝트 구성

- 2개의 Task 를 생성한다.
- OverflowTask, NotifyTask 태스크는 모두 osPriorityNormal 로 우선순위 설정
- Notify Task 는 1초 간격으로 시그널을 OverflowTask 로 전송
- OverflowTask 는 시그널 수신시 StartOverflowTask 를 다시 호출하여 Stack 사용을 증가하도록 구현

동작

- Notify Task 가 시그널을 전송할 때마다, Overflow Task 는 Stack 사용이 증가하며 uxTaskGetStackHighWaterMark 함수를 이용하여 잔여 Stack 크기를 확인
- Stack overflow 가 발생하면 vApplicationStackOverflowHook 함수가 호출되면서 "overflow" 메시지 출력


```

/* USER CODE BEGIN Header_StartOverflowTask */
/**
 * @brief Function implementing the OverflowTask thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartOverflowTask */
void StartOverflowTask(void const *argument)
{
    /* USER CODE BEGIN StartOverflowTask */
    /* Infinite loop */
    for (;;)
    {
        osEvent evt = osSignalWait(0xFF, 100);
        if (evt.status == osEventSignal)
        {
            printf("stack: %lu words\n", uxTaskGetStackHighWaterMark(OverflowTask));
        }
    }
    /* USER CODE END StartOverflowTask */
}

/* USER CODE BEGIN Header_StartNotifyTask */
/**
 * @brief Function implementing the NotifyTask thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartNotifyTask */
void StartNotifyTask(void const *argument)
{
    /* USER CODE BEGIN StartNotifyTask */
    /* Infinite loop */
    for (;;)
    {
        osSignalSet(OverflowTaskHandle, 0xFF);
        osDelay(1000);
    }
    /* USER CODE END StartNotifyTask */
}

```

기타 추가 정리

osStatus

대부분의 함수의 수행 결과를 반환하는 값

시그널 수신, 메시지수신, 메일수신, 타임아웃 발생 등 확인 가능

- Most of the functions returns `osStatus` value, below you can find return values on function completed list (`cmsis_os.h` file)

osStatus	value	description
osOK	0	no error or event occurred
osEventSignal	8	signal event occurred
osEventMessage	0x10	message event occurred
osEventMail	0x20	mail event occurred
osEventTimeout	0x40	timeout occurred
os_status_reserved	0x7FFFFFFF	prevent from <u>enum down-size compiler optimization</u>

```
typedef enum {
    osOK                      = 0,          ///< function completed; no err
    osEventSignal             = 0x08,       ///< function completed; signal
    osEventMessage            = 0x10,       ///< function completed; messag
    osEventMail               = 0x20,       ///< function completed; mail e
    osEventTimeout            = 0x40,       ///< function completed; timeou
    osErrorParameter          = 0x80,       ///< parameter error: a mandatc
    osErrorResource           = 0x81,       ///< resource not available: a
    osErrorTimeoutResource    = 0xC1,       ///< resource not available wit
    osErrorISR                = 0x82,       ///< not allowed in ISR context
    osErrorISRRecursive       = 0x83,       ///< function called multiple t
    osErrorPriority            = 0x84,       ///< system cannot determine pr
    osErrorNoMemory           = 0x85,       ///< system is out of memory: i
    osErrorValue              = 0x86,       ///< value of a parameter is ou
    osErrorOS                 = 0xFF,       ///< unspecified RTOS error: ru
    os_status_reserved        = 0x7FFFFFFF  ///< prevent from enum down-siz
} osStatus;
```

RAM footprint 절감

Task 별로 stack 크기를 최적화하기 위해서 `uxTaskGetStackHighWaterMark` 등을 사용하여 최대 Stack 사용량을 확인.

`vApplicationStackOverflowHook` 를 사용하여 overflow 검토.

전체 Heap 사용량은 `xPortGetFreeHeapSize` 를 사용하여 필요한 전체 Heap 사이즈 최적화.

Software 타이머를 사용하지 않는 경우에는 configUSE_TIMERS 를 사용하여 Disable.

뮤텍스를 사용하지 않는 경우에는 configUSE_MUTEXES 를 사용하여 Disable.

configMAX_PRIORITIES 를 이용하여 불필요한 우선순위 관리용 리스트를 할당하지 않도록 최적화

CubeMx FreeRTOS 지원 기능 중 FreeRTOS Heap Usage 탭을 이용해서 정적으로 FreeRTOS 의 자원 할당에 따른 Heap 사용량 확인 가능.

Configuration

Reset Configuration

Tasks and Queues Timers and Semaphores Mutexes **FreeRTOS Heap Usage**

Config parameters Include parameters Advanced settings User Constants

Summary

HEAP STILL AVAILABLE	191776 Bytes
TOTAL HEAP USED	4832 Bytes
Total amount for tasks	4184 Bytes
Total amount for queues	248 Bytes
Total amount for timers	48 Bytes
Total amount for mutexes and semaphores	352 Bytes
FreeRTOS tasks	
Idle task (FreeRTOS internal)	0 Bytes
Timer service task (FreeRTOS internal)	0 Bytes
myTask01	1144 Bytes
myTask02	1144 Bytes
myTask03	632 Bytes
myTask04	632 Bytes
myTask05	632 Bytes
FreeRTOS queues	
myQueue01	124 Bytes
myQueue02	124 Bytes
TOTAL HEAP USED	
Total amount of the heap used by known objects (user objects, internal freertos objects)	

Idle Task 세부

Idle Task 는 스케줄러 구동 시 자동으로 생성됨

portTASK_FUCTION() 함수 참조

Idle 태스크에서는 메모리에서 해제할 삭제된 task 체크

configUSE_PREEMPTION=0 으로 cooperative 모드로 동작 시에는 taskYIELD 를 호출해서 context switching 수행

configUSE_IDLE_HOOK=1 설정 시, vApplicationIdleHook() callback 을 호출해서 Idle 타임에 필요한 작업 수행 가능

configUSE_TICKLESS_IDLE=1 설정 시, 저전력 모드로 진입하도록 함.

FreeRTOS 초기 구동 시퀀스

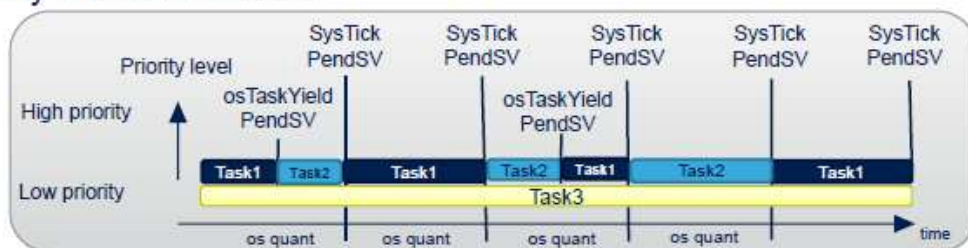
- 1) osKernelStart 함수를 main 함수에서 호출
- 2) vTaskStartScheduler 가 수행되면서 스케줄러 생성
- 3) 스케줄러는 IDLE task 생성하고, 모든 인터럽트를 Masking 한 후에 xPortStartScheduler 수행
- 4) xPortStartScheduler 는 context switching 과 관련된 SysTick, PendSV 인터럽트의 우선순위를 가장 낮은 우선 순위로 설정하고 SysTick 인터럽트 발생을 위한 타이머 구동
- 5) prvPortStartFirstTask 함수를 수행하여 첫번째 Task 구동
- 6) prvPortStartfirstTask 함수는 MSP(Main Stack Pointer) 를 Stack 시작 address 로 설정하고 인터럽트 Masking 을 해제한 후 SVC 인스트럭션을 호출
- 7) SVC 인스트럭션이 호출되면 vPortSVCHandler 가 호출
- 8) vPortSVCHandler 에서는 우선순위가 가장 높은 task 의 context와 TCB 를 stack 에 복구하고 수행 시작

osThreadYield 함수

Running 상태의 task 가 osThreadYield() 함수를 호출하면 Run 상태에서 Ready 상태로 전환되며 context switching 이 수행된다.

아래 그림을 보면 Task1 이 수행 중에 osThreadYield() 를 호출하면 강제적으로 Task2 로 context switching 이 발생하고 SysTick 에 의해서 다시 context switching 이 발생할 때까지 Ready 상태로 대기하게 된다.

- **osThreadYield()** – move the task from Run to Ready state. Next task with the same priority will be executed.



하지만 osThreadYield() 를 호출한 task 의 priority 가 가장 높다면 다시 해당 task 가 가장 우선순위가 높기 때문에 다시 수행되게 된다.

2

0



이지훈

달릴 준비만 하는거 아냐...달려야 하는데...^^; <https://github.com/eziya>
