Real Time Operating System (RTOS)

1. Operation System(OS)

A. 이용자와 Hardware 사이의 Interface에 이용되는 Software로

i. Program 작성과 문제 해결을 용이하게 하고,

ii. 컴퓨터 시스템을 사용하기 편리하게 하고,

iii. 효과적으로 컴퓨터 하드웨어를 사용 할 수 있게 한다.

B. Embedded System에 사용 되는 OS 예

i. Non Real Time Embedded OS

1. Embedded Linux(Kernel 2.4.x), Windows Embedded 등

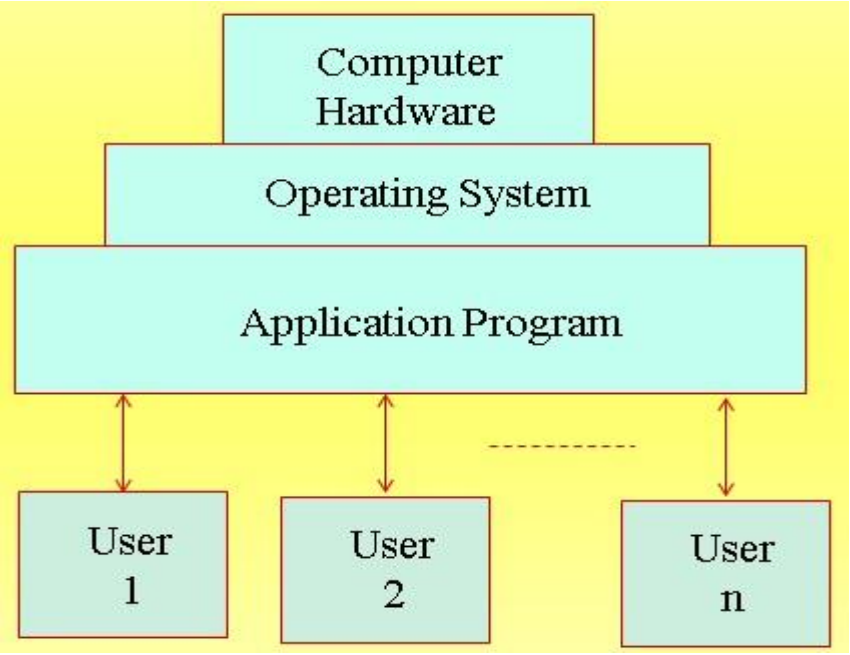
ii. Real Time Embedded OS

1. FreeRTOS, uC/OS-II, Linux(Kernel 2.6.x)

iii. Handheld/Mobile OS

1. Android, iOS , Symbian

Computer System



2. Real Time System

A. 시스템의 처리 결과가 논리적인 처리 결과와 시간의 정확성(제한된 시간 내)에 모두 의존적인 시스템으로 특히 시간의 정확성이 우선시 되는 시스템

B. Hard Real Time System

i. 이벤트의 응답이 Deadline(Time-limit) 내에 처리 되어야 함.

ii. 예: 모터제어 ; 일정한 시간 내에 속도와 토크 제어

C. Soft Real Time System

i. Deadline을 벗어난 응답이 결정적인 문제를 발생 시키지는 않음.

ii. 예: 영상재상; 일정한 시간 내에 처리하지 못하면 일부 영상의 손실이 있으나 계속 재생 가능

3. RTOS의 필요성

A. 제품의 개발 주기 및 라이프 사이클이 빨라짐

i. 개발 기간을 단축 시키기 위하여

ii. 검증된 Code를 재 사용 하거나,

iii. 동시에 여러 명이 하나의 과제를 진행 할 필요

B. 제품 기능의 복잡도 증가

i. Real Time Control 필요,

ii. Multi-Tasking의 기능이 필요한 제품 증가

C. Microcontroller의 성능은 좋아지고 가격은 하락

i. 대부분의 제품에서 RTOS에 의한 오버헤드 가 문제되지 않음

D. 파일 시스템과 네트워크를 필요로 하는 제품 증가

AVR FreeRTOS : Task Management

1. 이 장의 개요

A. 이 장의 중요 목표

이 장에서는 FreeRTOS를 어떻게 사용하고, FreeRTOS 응용 Task가 어떻게 동작하는지에 대한 개념을 설명 한다.

FreeRTOS와 Task에 대한 일반적인 이해를 향상 시키기 위하여 아래의 내용을 설명 한다.

i. 각 Task에게 FreeRTOS 가 어떻게 Processing Time을 배분 하는가?

ii. FreeRTOS 가 실행될 Task를 어떻게 결정 하는 가?

iii. 각 Task의 상대적인 Priority 가 시스템 동작에 미치는 영향

iv. Task 가 존재 할 수 있는 States에 대한 이해

또한, Task 관리 방법에 대한 이해를 향상 시키기 위하여 다음과 같은 내용을 설명 한다.

i. 어떻게 Task를 구현 하나?

ii. 어떻게 Task의 Instance를 하나 또는 여러 개 생성 하는가?

iii. 어떻게 Task Parameter를 사용 하나?

iv. 어떻게 이미 생성된 Task의 Priority를 변경 하나?

v. Task를 Delete 시키는 방법

vi. 어떤 경우에 Idle Task가 실행 되고 사용 되나?

2. Task Functions

freeRTOS에서 Task는 C 언어로 구현된 다음과 같은 ProtoType의 C 함수 이다.

A. Task function prototype

```
void ATaskFunction( void *pvParameters );
```

B. Task는 Entry Point를 갖고, 아래 예와 같이 무한 loop를 계속 실행 (Task 내에 Return 문을 포함 하지 않음)한다. 만약 Task가 더 이상 필요 없게 되면 아래 예와 같이 삭제 하여야 한다.

대표적인 Task Function의 구조

```
void ATaskFunction( void *pvParameters )
{
    // Variables Normal function 과 같이 선언 된다.
    int iVariableExample = 0;

    // Task는 Infinite Loop 로 구현 된다.
    for( ;; )
    {
        // Task 기능을 구현 하는 Code.
    }
    // Task 가 위의 무한 Loop로 부터 Break out 되는 경우
    // 이 함수의 끝에서 Delete 되어야 한다.
    vTaskDelete( NULL );
}
```

C. 하나의 Task 함수가 정의 되면 필요한 만큼의 Task를 Create 하여 사용 할 수 있다. Create 된 각 Task 는 서로 분리되어 실행 되는 Instance 가 된다. 각 Task는 자신의 Stack과 Automatic(Stack) Variable을 갖는다.

3. Top Level Task State

A. Running State

Application은 여러 개의 Task로 구성된다. 그러나 Single Core Processor 인 경우 어느 한 순간에는 오직 하나의 Task 만이 실행 상태(Running State)에 있게 된다.

단순화를 위하여 Running State와 Not Running State 2개의 상태 만 갖는 가장 간단한 Model(Not Running State 는 이후에 더 여러 종류의 Sub State로 나누어 진 Model로 확장 된다.)을 먼저 생각 한다.

B. Not Running State

Not Running State는 Task 현재 실행되지 않는 상태이다. 이 상태는 Scheduler 의 결정에 의하여 다음에 실행 되기 위하여 대기 하는 상태이다.

Top Level Task State 와 Transition

Not Running 상태에 있는 모든 Task는 현재 실행 되지 않는 Task 이다.

어떤 한 순간에는 오직 하나의 Task 만 Running 상태에 있다.

C. Task State Transition

Task는 Running 상태에서 Not Running 상태로, 또는 Not Running 상태에서 Running 상태로 Transition('Switched in' or 'Swapped in') 된다.

4. Creating Task

A. Task는 xTaskCreate() API Function을 사용 하여 Create 된다.

xTaskCreate() API Function Prototype

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const signed char * const pcName,
                           unsigned short usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pxCreatedTask
                         );
```

Parameter Name/ Returned value	Description
pvTaskCode	Task Function의 Pointer(Function Name)
pcName	Task Name, 개발자의 Debugging 목적(읽기 쉽게 하기 위한 목적)으로 만 이용 된다.
usStackDepth	각 Task는 자신의 Stack를 갖는다. usStackDepth는 이 Task가 갖는 Stack의 크기(Word)를 지정 한다.
pvParameter	pvParameter에 Assigned Value 가 Task에 Pass

	된다.
uxPriority	Task의 Priority
pxCreatedTask	Task Handle, Task의 Priority를 바꾸거나 Task를 Delete 하는 등 Task를 참조 할 필요가 있는 경우 사용 된다. Application에서 사용하지 않는 경우에는 NULL로 설정 한다.
Returned value	2가지 가능한 값을 갖는다. pdTRUE: Task가 성공적으로 Create 된 경우 pdFALSE: Task가 성공적으로 Create 되지 못한 경우

B. Task Creation 예 : RT_task_basic_LED
LED를 제어 하는 2개의 Task를 Create하여 실행 시킨다.
이 예에서는 LED의 On/Off 상태를 관찰 하기 위하여 Task에 500mSec Delay를 삽입 하였다.
이 프로그램을 실행하면 LED의 LSB 4 Bits와 MSB 4 Bits 가 1초 주기로 점멸 한다.

Task1 : LED의 LSB 4Bit를 500mSec 간격으로 Toggle 시킨다.

```
void vTask1_flash_led() {

    for(;;)
    {
            PORTF ^= 0x0f;
            vTaskDelay(500);            // 500mSec Delay
    }
}
```

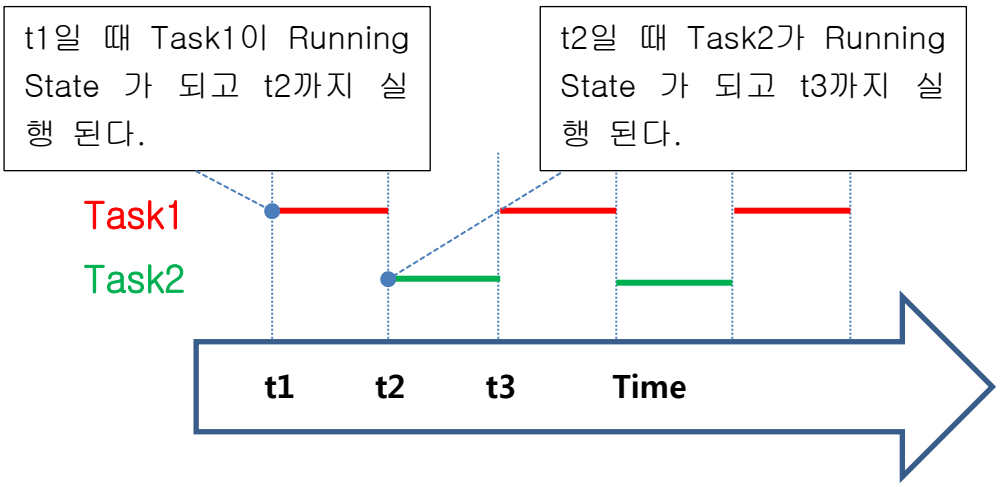
Task2 : LED의 MSB 4Bit를 500mSec 간격으로 Toggle 시킨다.

```
void vTask2_flash_led() {

    for(;;)
    {
            vTaskDelay(500);            // 500mSec Delay
            PORTF ^= 0xf0;
    }
}
```

main() : LED Port를 초기화 하고 2개의 Task를 Create 하여 실행 시킨다.

```
#include <avr/io.h>
#include <compat/deprecated.h>

//FreeRTOS include files
#include "FreeRTOS.h"
#include "task.h"

void vTask1_flash_led();
void vTask2_flash_led();
static void init_port_setting(void);

portSHORT main(void) {
    init_port_setting();

    //Create Tasks
    xTaskCreate(vTask1_flash_led, (signed portCHAR *)
        "vTask1_flash_led", configMINIMAL_STACK_SIZE, NULL,
        tskIDLE_PRIORITY + 1, NULL );
    xTaskCreate(vTask2_flash_led, (signed portCHAR *)
        "vTask2_flash_led", configMINIMAL_STACK_SIZE, NULL,
        tskIDLE_PRIORITY + 1, NULL );

    // RunSchedular
    vTaskStartScheduler();

    for( ;; );
    return 0;
}

static void init_port_setting(void){
   outp(0xFF,DDRF);    // PORTF : LED Port
   outp(0x00,PORTF);   //clear defect led.
}
```

위 예는 하나의 프로세서에서 2개의 Task가 동일한 Priority를 갖고 실행 되는 경우 이다.
아래 그림은 2개의 Task가 동일한 Priority를 갖고 실행되는 Pattern을 보여 준다. 이 그림에서 Task1과 Task2가 교대로 Running State와 Not Running State를 교대로 Switching 하며 실행 된다.



위 예에서는 2개의 Task가 main() 함수에서 Create 되고 Starting 되었다. 다음 예에서는 Task 내에서 다른 Task가 Create 되는 예를 설명 한다.

C. Task Creation 예 : RT_task_create_task_LED
Task는 아래와 같이 Scheduler가 Starting 된 다음에 다른 Task 내에서 Create 될 수 있다. 실행 결과는 동일 함.

```
void vTask1_flash_led() {
    xTaskCreate(vTask2_flash_led, (signed portCHAR *)
                "vTask2_flash_led", configMINIMAL_STACK_SIZE,
                NULL, tskIDLE_PRIORITY + 1, NULL );

    for(;;)
    {
            PORTF ^= 0x0f;
            vTaskDelay(500);            // 500mSec Delay
    }
}
```

D. Task Creation 예(Task Parameter의 사용) : RT_using_task_parameter
하나의 Task Implementation로부터 2개의 Instance를 Create하여 각 Instance에 서로 다른 Task Parameter를 전달 하는 예제 프로그램. 이 프로그램 예에는 Serial 통신에 필요한 함수와 문자열 입출력에 필요한 함수가 이용 되었다.(RT_using_task_parameter.zip 참조 요)

```
#define DELAY_LOOP_COUNT            ( 0x04ffff )
// Task에 Passing 되는 문자열(Task Parameter)
static const char *pcTextForTask1 = " Task 1 is Running ₩r₩n";
static const char *pcTextForTask2 = " Task 2 is Running ₩r₩n";
```

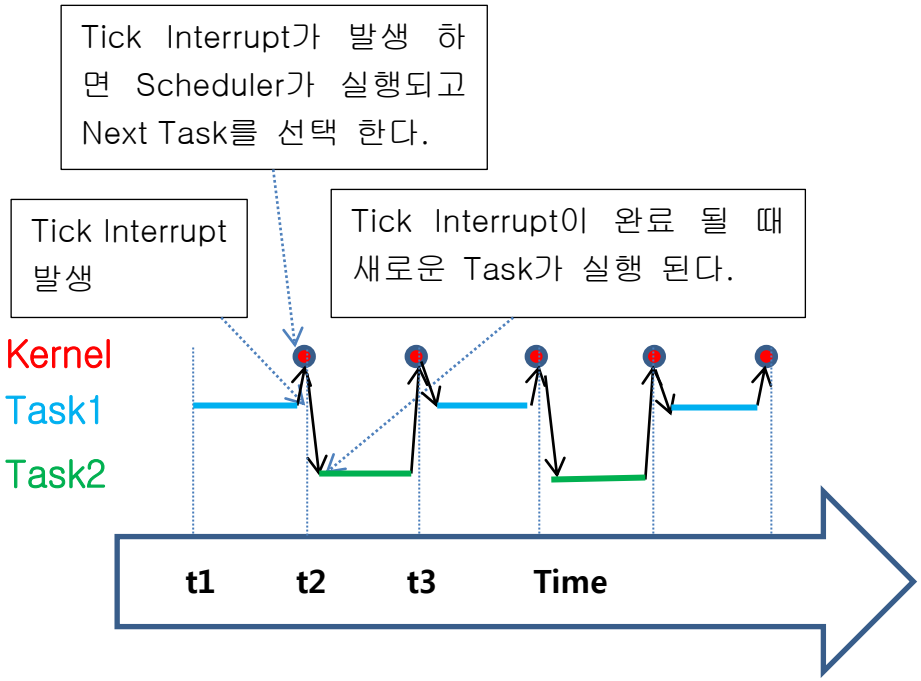
```
    // Task parameter 이용 예를 보여주기 위한 Task
    void vTaskFunction( void *pvParameters )
    {
    char *pcTaskName;
    volatile unsigned long ul;

    pcTaskName = (char *) pvParameters;

    for( ;; )
    {
            SCI_OutString(pcTaskName);
            // vTaskDelay() 함수를 사용 하여 Delay 기능을 구현 하면
            // Delay 상태 동안 Task가 Suspended 상태가 되기
            // 때문에 Priority 에 관계 없이 다음 Task가 실행 된다.
            // 그러므로, Priority 에 따라 Task가 Running 되는 것을
            // 관찰 하기 위하여 for Loop를 이용한 Delay를 사용 한다.
            for( ul = 0; ul < DELAY_LOOP_COUNT; ul++ ){    }
    }
}

//-----------------------------------------------
portSHORT main( void )
{
    // com port 초기화
   xSerialPortInitMinimal(mainCOM_BAUD_RATE, comBUFFER_LEN );

    // 하나의 Task Function으으로 부터 2개의 Task가 Create 된다.
    // 그러나, 각 Task에는 다른 Parameter(문자열)가 Pass 된다.
    xTaskCreate( vTaskFunction, ( signed char * ) "Task1",
            comSTACK_SIZE, (void*)pcTextForTask1,
            tskIDLE_PRIORITY + 1, ( xTaskHandle * ) NULL );
    xTaskCreate( vTaskFunction, ( signed char * ) "Task2",
            comSTACK_SIZE, (void*)pcTextForTask2,
            tskIDLE_PRIORITY + 1, ( xTaskHandle * ) NULL );

    vTaskStartScheduler();

    return 0;
}
```

5. Task Priorities

xTaskCreate() API 함수에 의하여 Task가 Creation 될 때 uxPriority

Parameter 값에 의하여 Priority 값이 초기화 된다.
Scheduler 가 Start 된 다음에 cTaskPrioritySet() API 함수에 의하여 Priority를 변경 할 수 있다.
Priority의 최대값은 FreeRTOSConfig.h 파일의 configMAX_PRIORITIES 값에 의하여 설정 된다.
그러나, 너무 큰 configMAX_PRIORITIES 값은 보다 큰 RAM 용량을 필요로 하기 때문에 Task에 필요한 Priority 값을 설정할 수 있는 정도 값으로 설정 하여야 한다.
낮은 Priority 값을 갖는 Task가 낮은 Priority를 갖는다. 그러므로 가능한 Priority 값의 범위는 0 – (configMAX_PRIORITIES – 1) 이다.
일정한 주기 마다 Interrupt(Tick Interrupt)가 발생 하고 이 때 마다 Scheduler가 실행되어 실행 가능한 상태(Ready State)에 있는 Task 중 가장 Priority가 높은 Task를 Running State가 되도록 한다.

Tick Interrupt의 발생과 Scheduler가 새로운 Task를 실행 하는 과정



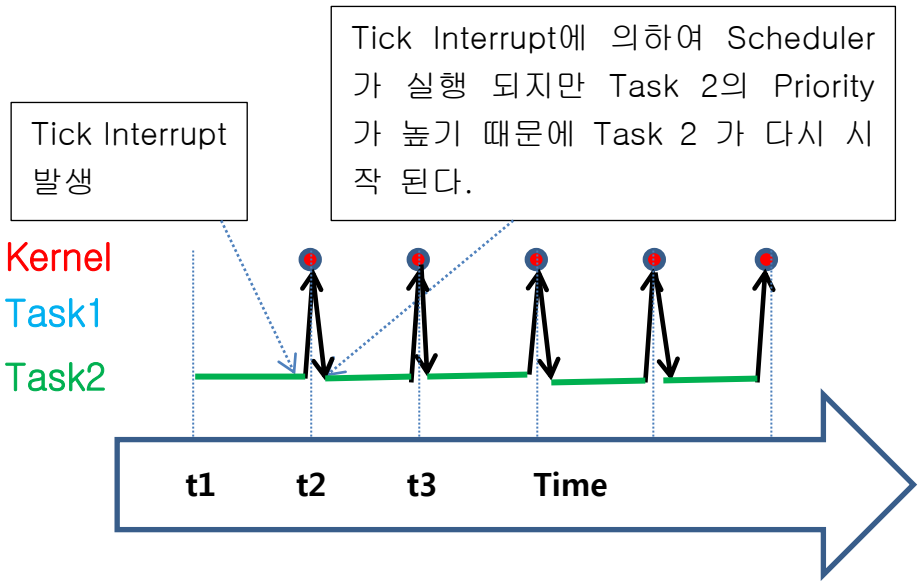
A. Priority 가 다른 Task의 실행 예 : RT_prirorty_change_LED

```
portSHORT main( void )
{
    // Initialise the com port then spawn the Rx and Tx tasks.
    xSerialPortInitMinimal( mainCOM_BAUD_RATE,
                            comBUFFER_LEN );

    // 하나의 Task Function으으로 부터 2개의 Task가 Create 된다.
```

```
        // 각 Task에는 서로 다른 Parameter(문자열)가 Pass 된다.
        xTaskCreate( vTaskFunction, ( signed char * ) "Task1",
            comSTACK_SIZE, (void*)pcTextForTask1, tskIDLE_PRIORITY
            + 1, ( xTaskHandle * ) NULL );
        xTaskCreate( vTaskFunction, ( signed char * ) "Task2",
            comSTACK_SIZE, (void*)pcTextForTask2, tskIDLE_PRIORITY
            + 2, ( xTaskHandle * ) NULL );

        vTaskStartScheduler();

        return 0;
}
```

예제 프로그램의 실행 패턴

한 Task의 Priority가 다른 Task의 Priority 보다 높은 경우 Priority가 높은 Task 가 계속 실행 된다.



6. Not Running State의 확장

Not Running State는 Event Driven 방식 등 다양한 형태의 Task 제어를 가능 하도록 하기 위하여 Ready State, Suspended State, Blocked State로 확장 된다.

A. Blocked State
   i. Event를 기다리는 상태
   ii. Task는 다음과 같은 2종류의 Event를 기다리기 위하여 Blocked State에 놓여 진다.
      1. 시간에 관계된 Event : 시간 지연(Delay)이 종료 되거나, 특정한 시간이 되었을 때 발생 하는 Event

2. 다른 Task 또는 Interrupt와 동기 하기 위한 Event : 이 종류의 Event는 다른 Task 또는 Interrupt에 의하여 발생 된다. Queue에 Data가 도착 하였을 때 발생하는 Event, 다른 장치와 동기(Synchronization) 하기 위한 Event 등

iii. FreeRTOS에서는 타이머, Binary Semaphores, Counting Semaphores, Recursive Semaphores, Mutexes에 의하여 Event 가 발생 된다.

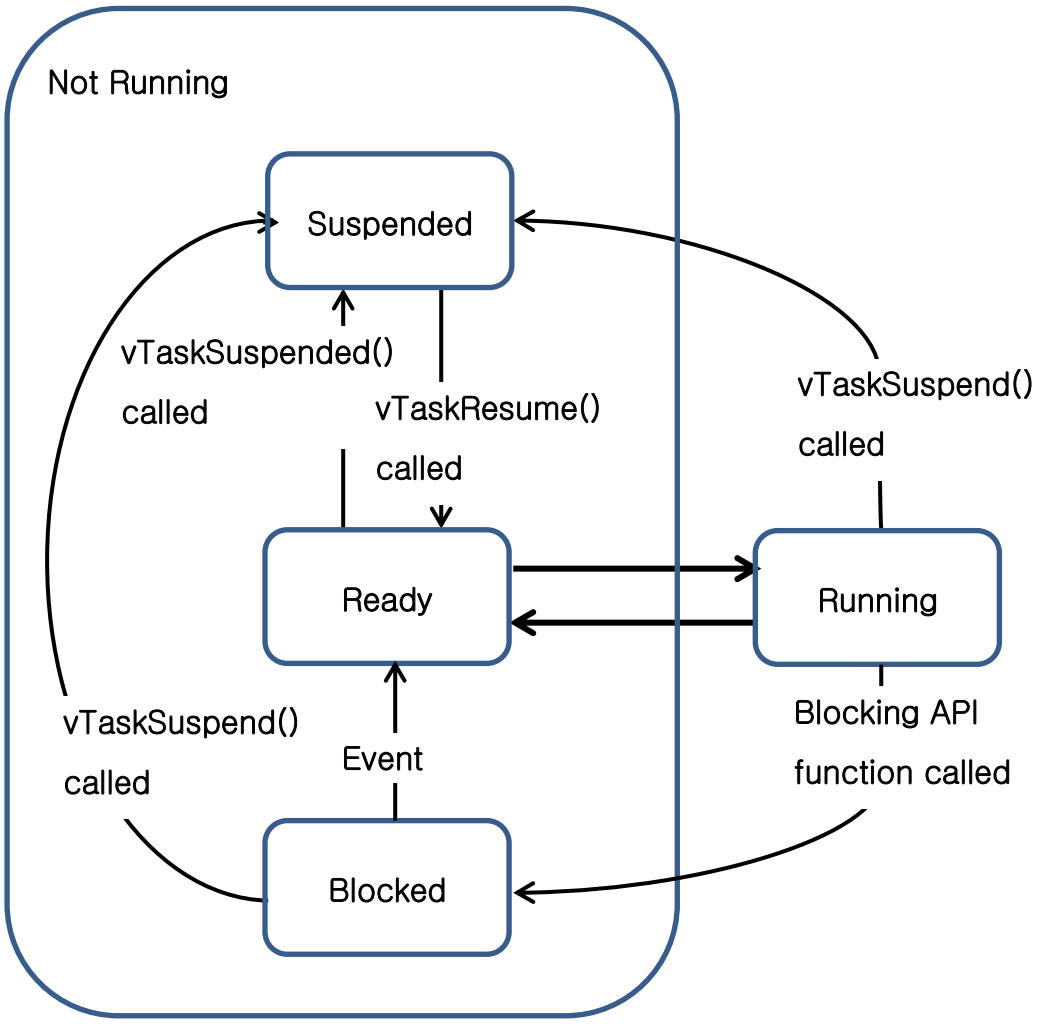
B. Suspended State

i. 다른 태스크에 의하여 일시적으로 정지된 상태로 이 상태에서는 Scheduler에 의하여 Running 상태로 천이 할 수 없다. Running 상태가 되기 위하여는 vTaskResume() 함수에 의하여 Ready 상태로 천이 되어야 Running 상태로 천이 할 수 있다.

C. Ready State

i. 실행 가능 상태, Task Scheduler에 의하여 우선 순위가 높은 Task 먼저 Running 상태로 천이 한다.

State Transition Diagram



D. Blocked State(vTaskDelay() API 이용)를 이용한 Delay 구현 예 : RT_task_delay_LED

Void vTaskDelay() API function prototype

Void vTaskDelay(PortTickType xTicksToDelay);

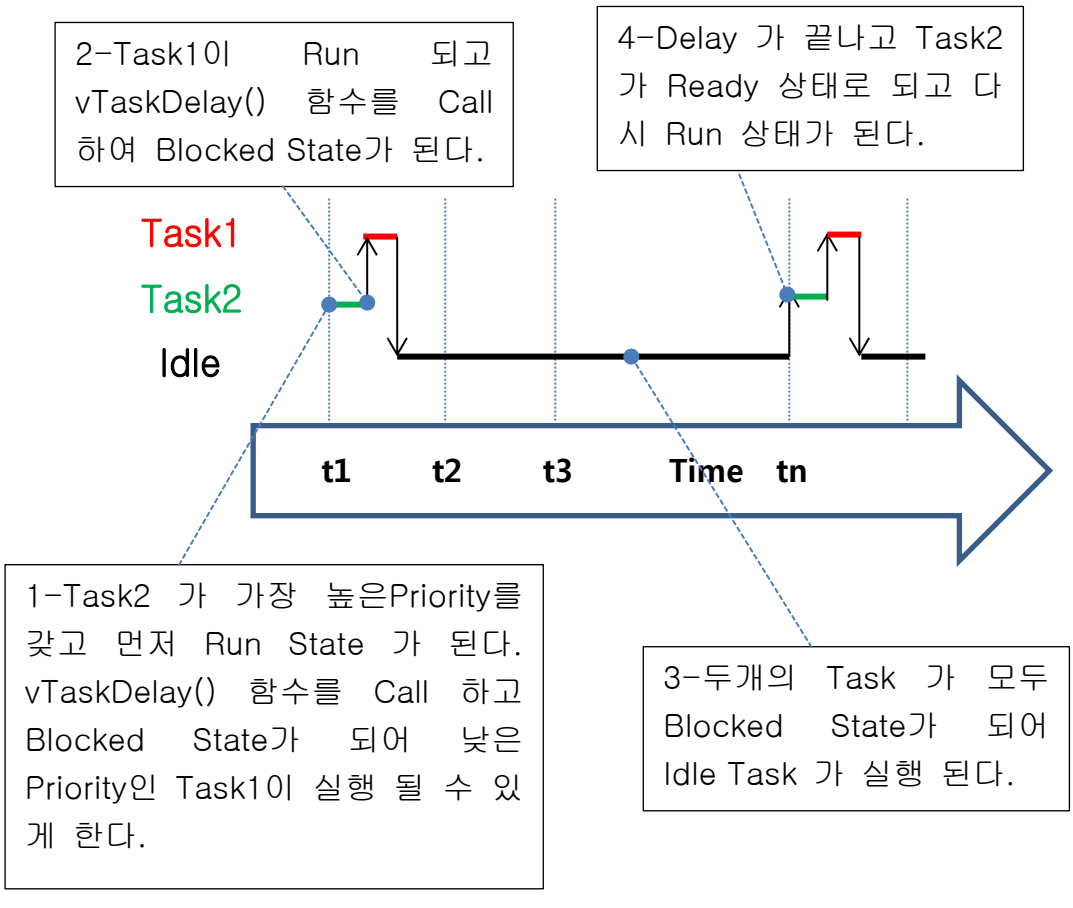
Parameter Name/ Returned value	Description
xTicksToDelay	Calling Task가 Blocked State에 남아있어야 하는 Tick Interrupt 수. 예: Task에서 vTaskDelay(100) 이 실행되는 경우. 현재 Tick Count 값이 10,000이라면 Tick Count 값이 10,100 이 될때 까지 이 Task는 Blocked State에 남아 있는다.

vTaskDelay() API를 이용 하여 500mSec Delay를 실현하여 1Sec에 한번씩 LED가 점멸 하는 프로그램 예이다.

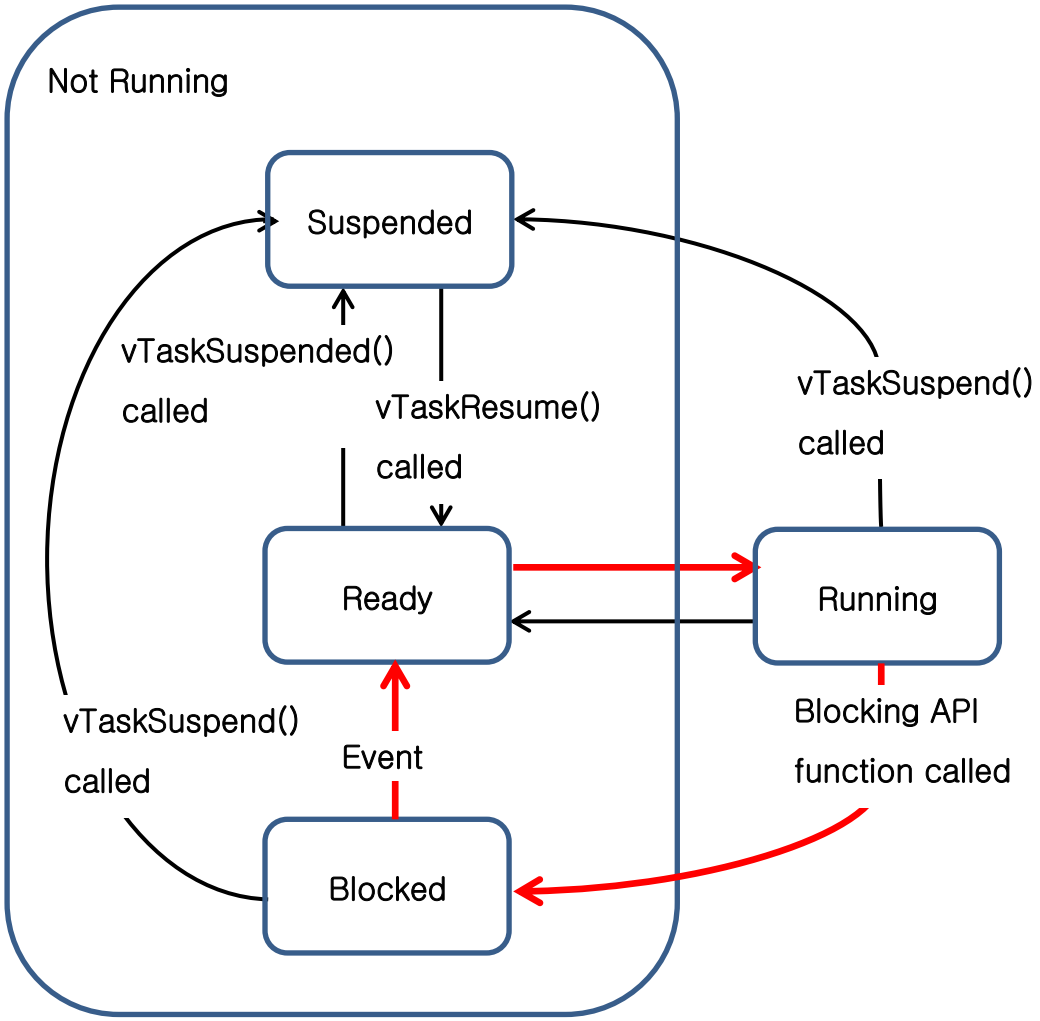
```
//#define TICK_INTERRUPT_TIMER0
#include <avr/io.h>
#include <compat/deprecated.h>
//FreeRTOS include files
#include "FreeRTOS.h"
#include "task.h"

void vTask1_flash_led( void *pvParameters );
static void init_port_setting(void);

void vTask1_flash_led( void *pvParameters ) {
	for(;;)
	{
		PORTF = ~PORTF;
		vTaskDelay(500);		// 500mSec Delay
	}
}
```

```
portSHORT main(void) {
	init_port_setting();

	//Create Tasks
	xTaskCreate(vTask1_flash_led, (signed portCHAR
	*)"vTask1_flash_led", configMINIMAL_STACK_SIZE, NULL,
	tskIDLE_PRIORITY + 1, NULL );

	//RunSchedular
	vTaskStartScheduler();

	for( ;; );
	return 0;
}

static void init_port_setting(void){
    outp(0xFF,DDRF);    // PORTF : LED Port
    outp(0x00,PORTF);   //clear defect led.
}
```

vTaskDelay() 가 실행 되었을 때의 실행 순서



위 예의 State Transitions



E. vTaskDelayUntil() API Function

Void vTaskDelayUntil() API function prototype

Void vTaskDelayUntil(PortTickType * pxPreviousWakeTime,
PortTickType xTimeIncrement);

Parameter Name/ Returned value	Description
pxPreviousWakeTime	일정한 주기 마다 정확히 Task를 실행하는 하도록 하는 vTaskDelayUntil() API 에서 사용 된다. 이 값은 Task가 Blocked State에 머무는 동안 Blocked State에서 Read State로 천이 하는 시간을 계산 하기 위하여 이용 된다. 이 값은 처음 초기화 된 다음 응용 프로그램에서 Modify 하는 것이 아니고, 이

	Task가 Wake Up 될 때 자동으로 Modify 된다.
xTimeIncrement	Task 실행 주기를 결정 하는데 이용 된다. 상수 portTICK_RATE_MS는 Ticks를 Milliseconds로 변환 하도록 하는 상수 이다.

F. vTaskDelayUntil()를 사용하여 고정된 주기마다 Task를 실행 하는 예
vTaskDelay() API를 사용 하면 일정한 시간 지연을 실현 할 수 있지만 Task가 일정한 주기 마다 정확히 실행 된다고 보장 할 수 없다.
Task를 일정한 주기로 정확히 실행 하기 위하여는 vTaskDelayUntil()를 사용 하여야 한다.

vTaskDelayUntil() API를 이용 하여 매 500mSec 마다 일정한 주기로 Task를 실행 한다. 그 결과 1Sec에 한번씩 LED가 점멸 하는 프로그램 예이다.

프로그램 예 : RT_task_delay_until_LED

G. Blocking 과 Non-Blocking Tasks의 결합 예 : (생략)

7. Idle Task와 Idle Task Hook

하나의 Task를 생성하여 실행 하고 있는 경우 이 Task가 Blocked State에 남아 있다면 이 Task는 Scheduler에 의하여 선택 될 수 없다.
그러나 Processor는 항상 어떠한 일이든 실행 하고 있어야 하기 때문에 최소한 하나의 Task는 언제나 실행 상태에 있어야 한다.
그러므로 vTaskStartScheduler() API가 Call 될 때 자동적으로 Idle Task가 Create 된다.
만약 다른 Task가 Ready State가 되면 바로 Ready State에 있는 Task가 실행 되어야 하기 때문에 Idle Task는 가장 낮은 Priority를 갖는다.

A. Idle Task Hook Function
Idle Task 가 실행 될 때 Idle Task에 의하여 실행 된다.
   i. Idle Task HooK의 일반적인 사용 분야
      1. Low Priority로 Background 에서 계속적으로 실행 된다.
      2. Idle Task의 Processing Time을 측정 하여 Processing Time이 어느 정도 여유 있는 지 알 수 있다.
      3. Application Processing 이 없는 경우 자동적으로 Processor를 Low Power Mode로 놓을 수 있다.

B. Idle Task Hook Functions 구현 시 제한되는 사항
   i. Idle Task Hook Functions을 Block 또는 Suspend 상태로 놓으려고 하여서는 안 된다. Idle Task Hook Functions는 오직 다른 Task가 실행되지 않는(Idle Task 만 실행되는 경우)상태에서만 실행 된다. Idle Task가 Blocking 되면 Running 상태에 있는 Task가 하나도 없는 상태가 된다.
   ii. 만약 Idle Task Hook Functions에 대하여 vTaskDelete() API 함수가 사용 되면 Idle Task Hook은 적당한 주기에 Caller에게 Return 된다. 이 것은 Delete 된 Task의 자원을 시스템에 되돌려 줄 수 있도록 한다.

Idle Task Hook Functions의 Prototype은 다음과 같다.
void vApplicationIdleHook(void);

C. Idle Task Hook 함수의 정의 예 :
vTaskDelay() API 를 Call 하면 많은 Idle Time이 발생 한다.
예제 프로그램은 응용 Task가 Blocked State에 있을 때 Idle Hook Function을 이용 하여 Idle Time을 측정 하는 예 이다.
Idle Hook Function은 configUSE_IDLE_HOOK 이 1로 Set 되어야 사용 할 수 있다.

프로그램 예 : RT_idleCycleCount_printf

8. Task Priority 의 변경
A. vTaskPrioritySet() API 함수
vTaskPrioritySet() API 함수는 Task의 Priority를 변경 하는데 사용 한다.
vTaskPrioritySet() API 함수는FreeRTOSConfig.h 에서 INCLUDE_vTaskPrioritySet 가 1로 Set 되어야 사용 할 수 있다.

Void vTaskPrioritySet(xTaskHandle pxTask, unsigned portBASE_Type uxNewPriority);

Parameter Name/ Returned value	Description
pxTask	Priority 를 변경 할 Task의 Handle 현재 Task 자신의 Priority를 변경 할 경우에는 NULL 로 한다.
uxNewPriority	새로 정의할 Priority

B. uxTaskPriorityGet() API 함수
uxTaskPriorityGet() API 함수는 Task의 Priority 값을 구하기 위하여

사용 한다.
uxTaskPriorityGet() API 함수는FreeRTOSConfig.h 에서
INCLUDE_vTaskPriorityGet 가 1로 Set 되어야 사용 할 수 있다.

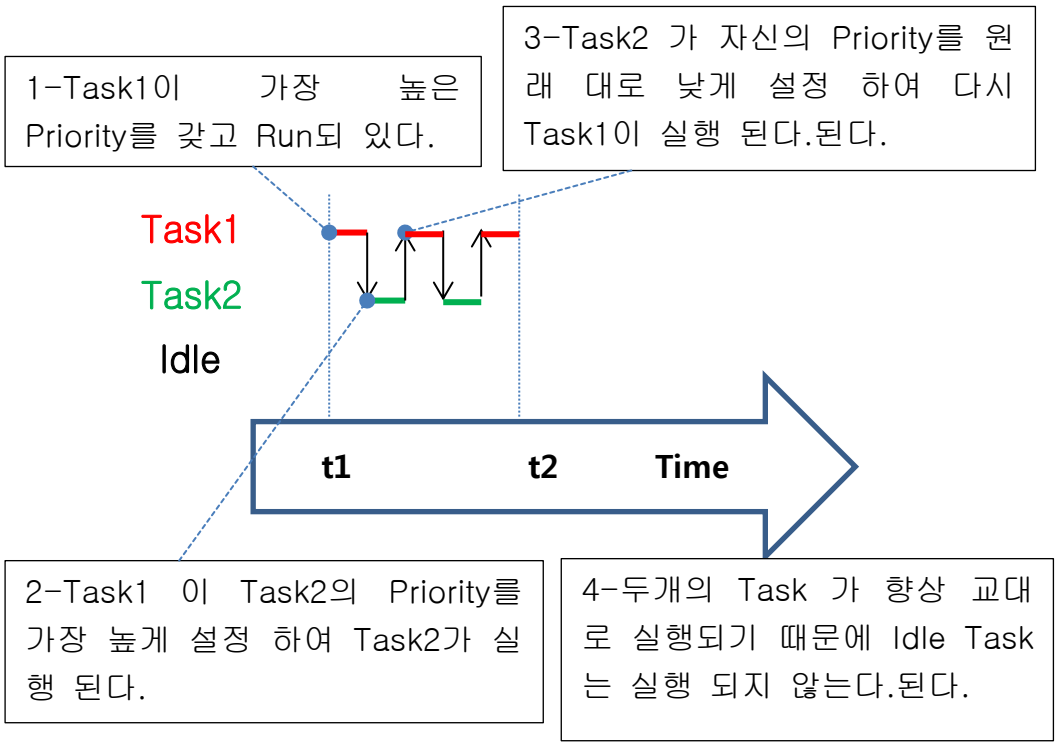
unsigned portBASE_TYPE uxTaskPriorityGet(xTaskHandle pxTask);

Parameter Name/ Returned value	Description
pxTask	Priority 값을 알고자 하는 Task의 Handle 현재 Task 자신의 Priority를 구하는 경우에는 NULL 로 한다.
Returned value	pxTask 에 의하여 지정된 Task의 Priority

C. Task Priorities를 변경 하는 예
Task1(LED 의 상위 4Bits 만 Turn On) 과 Task2(LED 의 상위 4Bits 만 Turn On) 가 동일한 Priority를 갖고 실행되면 LED의 상위 4Bits와 하위 4Bits가 교대로 점멸 한다.
만약 Task2의 Priority를 Task1의 Priority 보다 높게 하면 LED의 상위 4Bits 만 Turn ON 상태가 지속 된다.

프로그램 예 : RT_prirorty_change_LED

Priority Changing 예의 실행 순서



9. Task 삭제 하기
   A. vTaskDelete() API 함수
      vTaskDelete() API 함수는 현재 실행 중인 자기 자신이나 다른 Task를 삭제 하기 위하여 사용 된다.
   vTaskDelete() API 함수는FreeRTOSConfig.h 에서 INCLUDE_vTaskDelete 가 1로 Set 되어야 사용 할 수 있다.
   Task에 Allocate 된 Memory는 Task가 삭제 되면 자동으로 해제되어 재 사용 할 수 있다. 그러나 다른 Resource는 별도로 해제 되어야 한다.

   Void vTaskDelete(xTaskHandle pxTaskToDelete);

Parameter Name/ Returned value	Description
pxTaskToDelete	Delete 하고자 하는 Task의 Handle 현재 Task 자신을 Delete하는 경우에는 NULL 로 한다.

   B. Task Delete 예

      프로그램 예 : RT_task_delete_create

10. Scheduling Algorithm의 종합 정리
   A. Priority Pre-emptive Scheduling
      i. 각 Task는 Priority 를 갖는다.
      ii. 각 Task는 어느 특정한 시간에는 여러 State 중 하나의 State에만 존재 할 수 있다.
      iii. 오직 한 Task 만 어느 특정한 시간에 Running State 에 있을 수 있다.
      iv. Scheduler는 항상 현재 Ready State에 있는 Task 중 가장 높은 Priority를 갖은 Task를 Running State 에 놓는다.