FreeRTOS for STM32

Step 4

FreeRTOS - Interrupt Management

이벤트 처리 구현 전략

1. event 검출을 어떻게 할것인가? (Interrupt or polling ?)
2. interrupt가 사용될때, ISR내의 처리와 ISR외부의 처리량을 어떻게 할것인가?
3. event가 main code와 어떻게 통신 하는지, 비동기 처리 할 수 있도록이 코드를 어떻게 구성 할 것인가?

Interrupt 처리

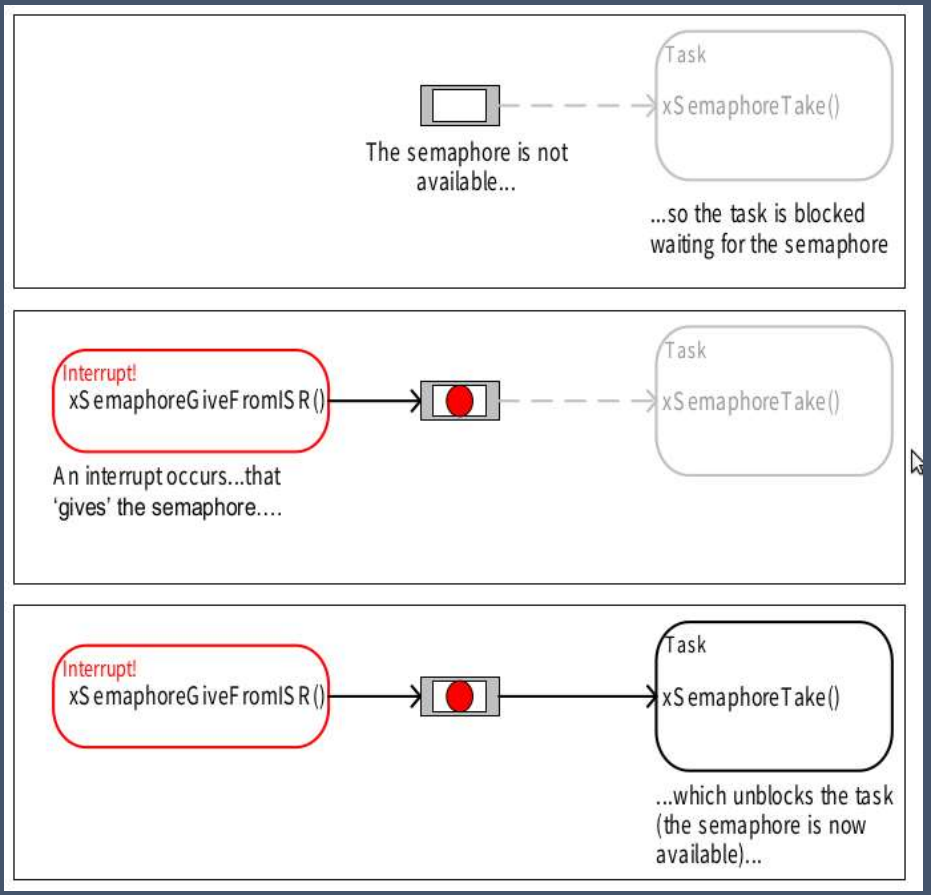
Interrupt의 처리시간은 짧게 해야 한다.
Interrupt처리에 의해서 수행이 지연된 task의 우선순위가 다른 task의 우선순위보다 높으면 ISR처리후 바로 수행된다.

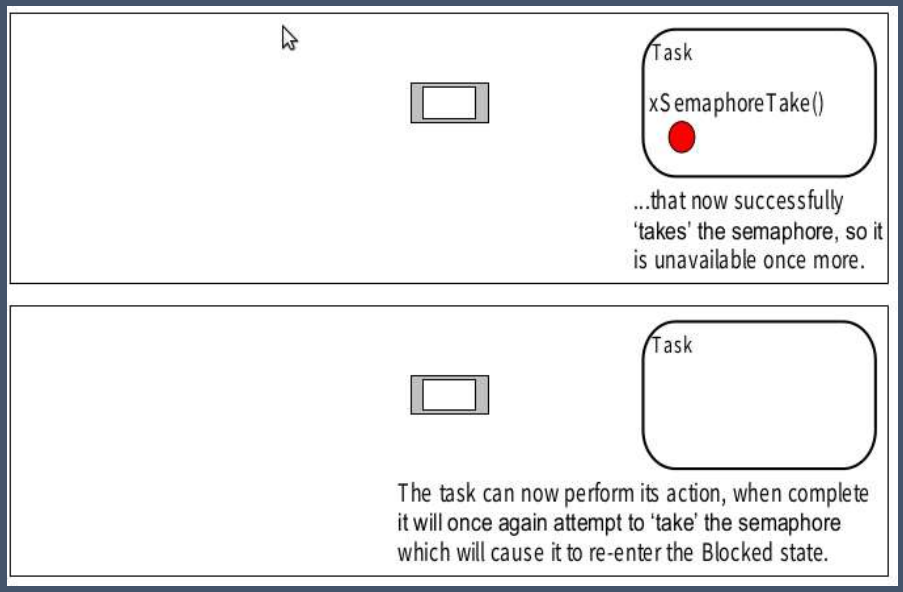
Interrupt Safe API

FreeRTOS에서는 task버전과 ISR버전의 두가지 버전의 함수를 제공
ISR버전 함수는 "FromISR" 접미사가 붙는다

FreeRTOS - Interrupt Management

Binary Semaphore





FreeRTOS - Interrupt Management

Binary Semaphore

xSemaphoreCreateBinary() : binary semaphore 생성

SemaphoreHandle_t xSemaphoreCreateBinary(void);	
return value	NULL: 실패 non-NULL; 성공, SemaphoreHandle_t 타입 handle 반환

xSemaphoreTake() : semaphore를 얻는다

BaseType_t xSemaphoreTake(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait);	
xSemaphore	Semaphore Handle
xTicksToWait	task가 Blocked state에 있을 시간의 최대값 0이면 바로 리턴 portMAX_DELAY은 무한대기 (INCLUDE_vTaskSuspend ("FreeRTOSConfig.h") 1로 설정
return value	pdPASS : semaphore taking 성공 pdFALSE: taking 가능한 semaphore가 없다(시간초과)

xSemaphoreGiveFromISR() : semaphore를 준다.

BaseType_t xSemaphoreGiveFromISR(SemaphoreHandle_t xSemaphore, BaseType_t *pxHigherPriorityTaskWoken);	
xSemaphore	Semaphore Handle
pxHigherPriorityTaskWoken	단일 semaphore가 하나이상의 semaphore를 기다리는 block상태의 task를 가질수 있다. xSemaphoreGiveFromISR ()을 호출하면 task의 Blocked state를 벗어나고 unblock된 task의 우선 순위가 현재 실행중인 task 보다 높으면 내부적으로 xSemaphoreGiveFromISR ()이 pxHigherPriorityTaskWoken을 pdTRUE로 설정한다.
return value	pdPASS: 성공 pdFAIL: 실패

FreeRTOS - Interrupt Management

Binary Semaphore

portYIELD_FROM_ISR() : ISR에서 context switch를 요청하는데 사용되는 macro
taskYIELD()의 interrupt safe version.

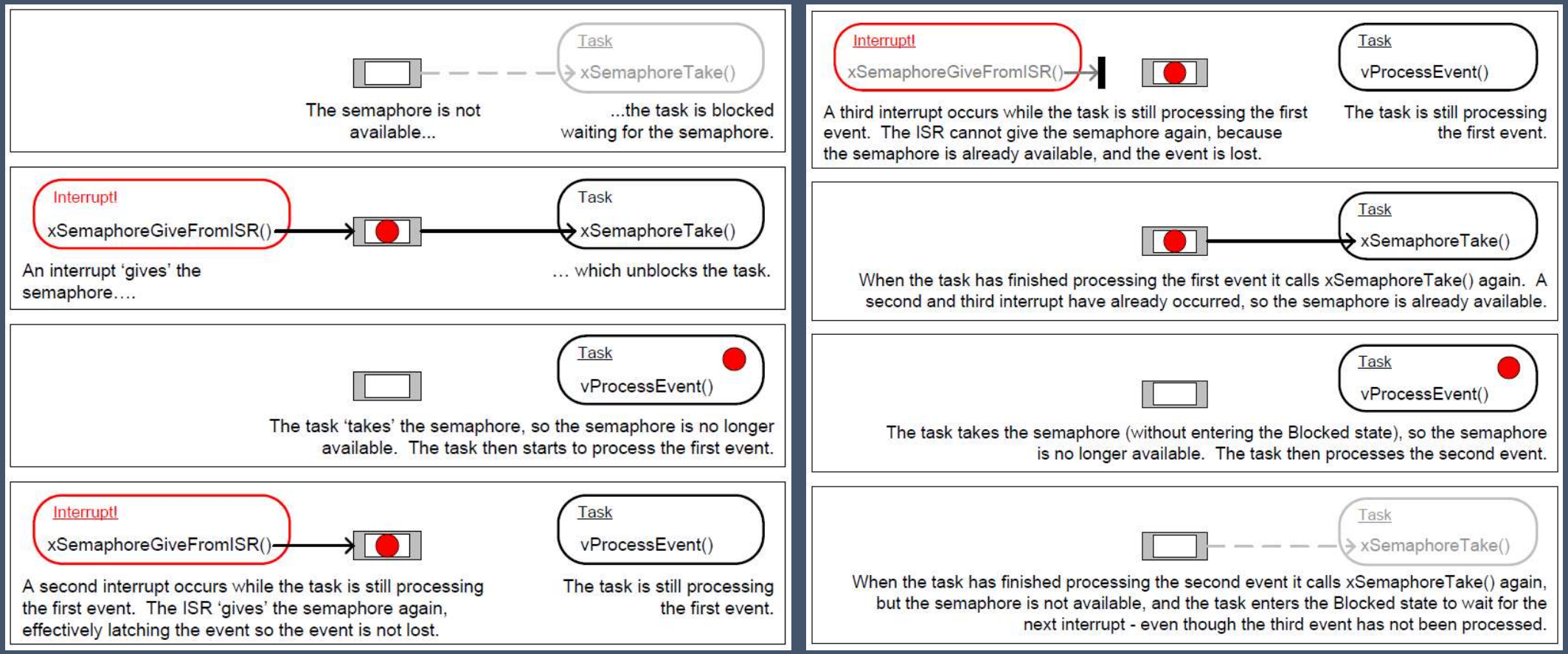
```
portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
```

FreeRTOS - Interrupt Management

Example18 – binary semaphore를 이용한 task와 interrupt 동기화

FreeRTOS - Interrupt Management

Example18-1 – task에서 데이터 처리가 늦어 Interrupt 처리가 지연됨



FreeRTOS - Interrupt Management

Counting Semaphore

binary semaphore는 길이가 하나인 queue, counting semaphore는 하나 이상의 길이를 가지는 queue로 볼수 있다.
configUSE_COUNTING_SEMAPHORES (”FreeRTOSConfig.h”) 설정 1

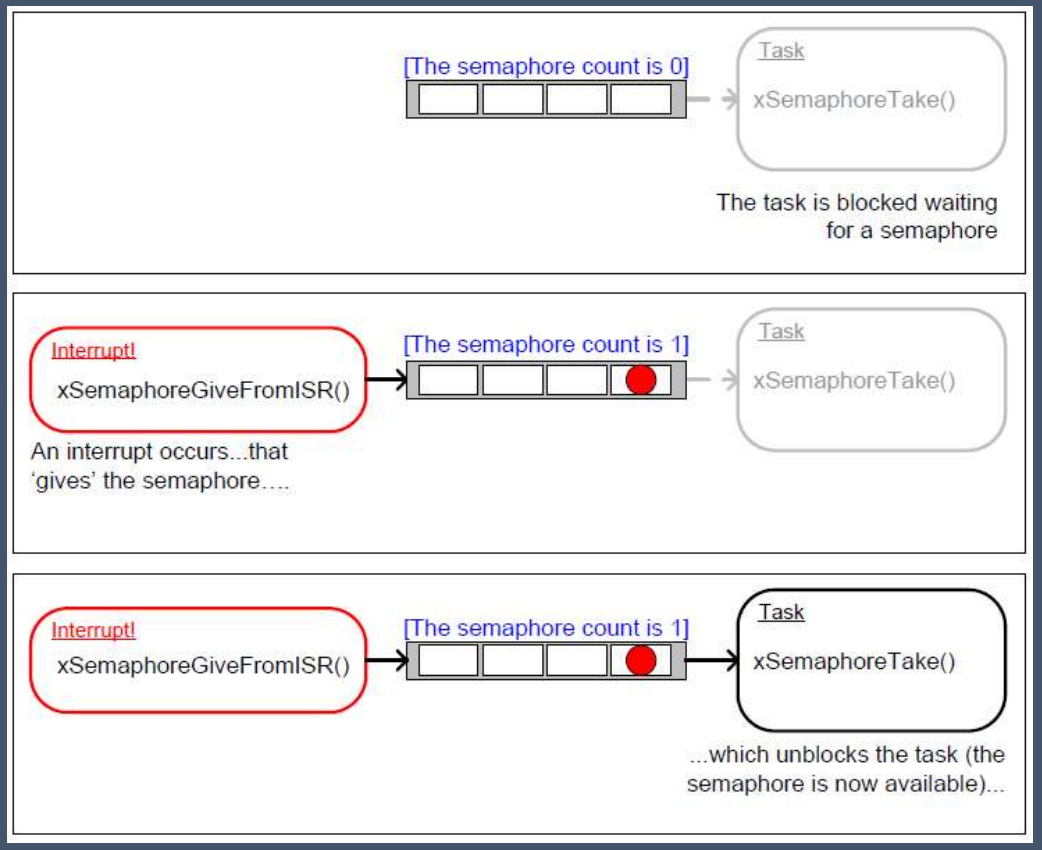
일반적으로 아음의 두경우에 사용

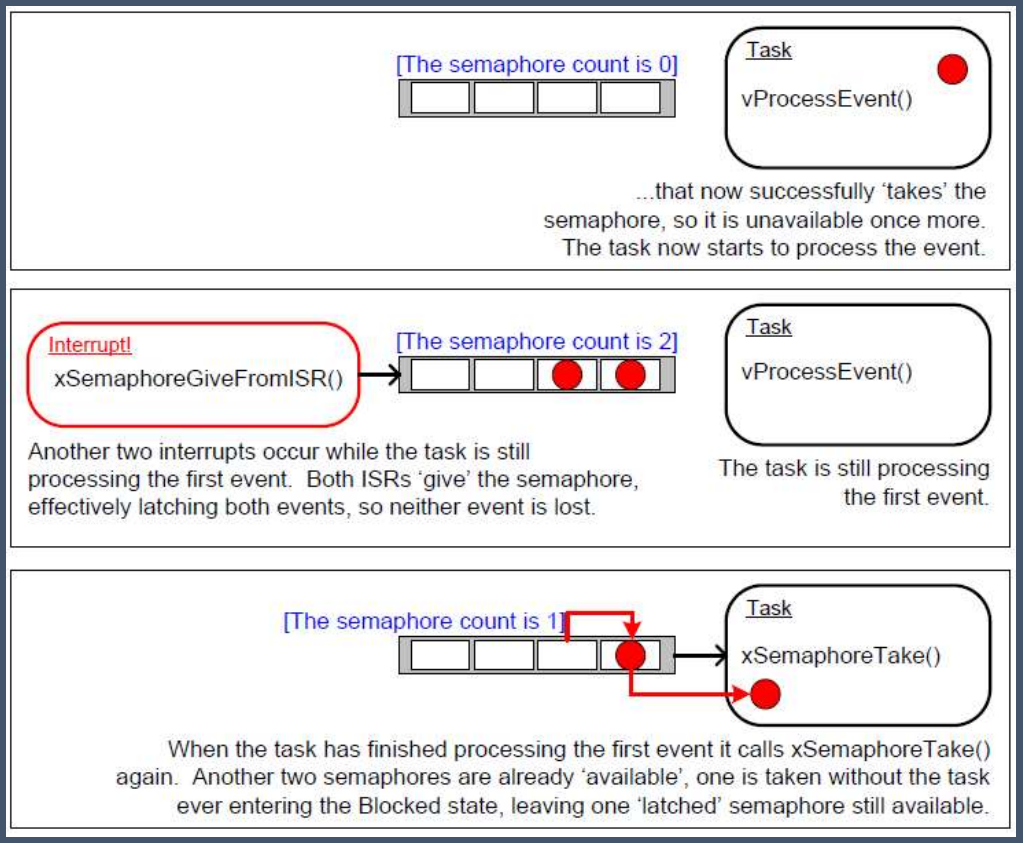
1. Counting event – event count에 사용(초기화 count value는 0)
2. Resource management - 사용가능한 resource의 수를 표시 (초기화 count value는 resource의 수)

FreeRTOS - Interrupt Management

Counting Semaphore

Counting Event 예





FreeRTOS - Interrupt Management

Counting Semaphore

xSemaphoreCreateCounting() : counting semaphore생성

SemaphoreHandle_t xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);	
uxMaxCount	Semaphore의 최대 수
uxInitialCount	Semaphore생성시 초기화 count 값
return value	NULL: 실패 non-NULL; 성공, SemaphoreHandle_t 타입 handle 반환

FreeRTOS - Interrupt Management

Example19 – counting semaphore를 이용한 task와 interrupt 동기화

FreeRTOS - Interrupt Management

Daemon Task에 작업연기 (Deferring Work)

Daemon Task에 사용자가 정의한 callback 함수인 'execute function'을 전달해 실행

'execute function' 함수 원형

```
void vPendableFunction( void *pvParameter1, uint32_t ulParameter2 );
```

xTimerPendFunctionCall() : daemon task에 'execute function' 전달
INCLUDE_xTimerPendFunctionCall ("FreeRTOSConfig.h") 설정 1

BaseType_t xTimerPendFunctionCallFromISR(PendedFunction_t xFunctionToPend, void *pvParameter1, uint32_t ulParameter2, BaseType_t *pxHigherPriorityTaskWoken);	
xFunctionToPend	daemon task에서 실행될 함수 pointer
pvParameter1	함수 pointer로 전달될 parameter
ulParameter2	함수 pointer로 전달될 parameter
pxHigherPriorityTaskWoken	daemon task의 우선순위가 현재 실행중인 task(interrupt를 실행한)의 우선 순위가 높으면 pdTRUE로 설정 daemon task가 Ready state에서 최상위 우선순위로 interrupt가 daemon task로 직접 return 된다.
return value	pdPASS : timer command queue에 'execute function'전달 pdFAIL : 'execute function'전달 실패, queue full

FreeRTOS - Interrupt Management

Example20 – Centralized deferred interrupt processing (중앙 집중식 지연 인터럽트 처리)

FreeRTOS - Interrupt Management

ISR에서 Queue사용

Queue는 데이터를 전달과 communicate event에 사용

BaseType_t xQueueSendToFrontFromISR(QueueHandle_t xQueue, void *pvItemToQueue BaseType_t *pxHigherPriorityTaskWoken);	
BaseType_t xQueueSendToBackFromISR(QueueHandle_t xQueue, void *pvItemToQueue BaseType_t *pxHigherPriorityTaskWoken);	
xQueue	queue handle
pvItemToQueue	queue로 복사할 data의 pointer
pxHigherPriorityTaskWoken	queue의 데이터를 기다리는 하나이상의 task가 있고, 이 task의 우선순위가 현재 실행중인 task(interrupt를 실행한) task보다 우선순위가 높다면 pdTRUE 리턴. isr이 종료되면, 우선순위가 높은 task실행
return value	pdPASS : queue sent 성공 errQUEUE_FULL: 실패 (queue full)

FreeRTOS - Interrupt Management

Example21 – interrupt로부터 queue를 이용해 데이터 송수신

FreeRTOS - Resource Management

critical section

taskENTER_CRITICAL(), taskEXIT_CRITICAL()로 싸인 코드영역
task가 Running state있는것을 보장
짧게 작성해야 한다

```
taskENTER_CRITICAL();
PORTA |= 0x01;
taskEXIT_CRITICAL();
```

taskENTER_CRITICAL_FROM_ISR(), taskEXIT_CRITICAL_FROM_ISR() interrupt safe version
interrupt 중첩을 지원하는 FreeRTOS에서만 제공
taskENTER_CRITICAL_FROM_ISR()의 리턴 값은 taskEXIT_CRITICAL_FROM_ISR()에 전달

```
void ISR(void)
{
   UBaseType_t uxSavedInterruptStatus;

   uxSavedInterruptStatus = taskENTER_CRITICAL_FROM_ISR();
   taskEXIT_CRITICAL_FROM_ISR( uxSavedInterruptStatus );
}
```

FreeRTOS - Resource Management

Suspending (or Locking) the Scheduler

스케줄러 Locking을 이용해서 Critical section구현, task내의 코드만 적용

vTaskSuspendAll() : 스케줄러 suspended
FreeRTOS API함수는 스케줄러가 suspended되면 호출되서는 안된다.

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

xTaskResumeAll() : 스케줄러를 un-suspended

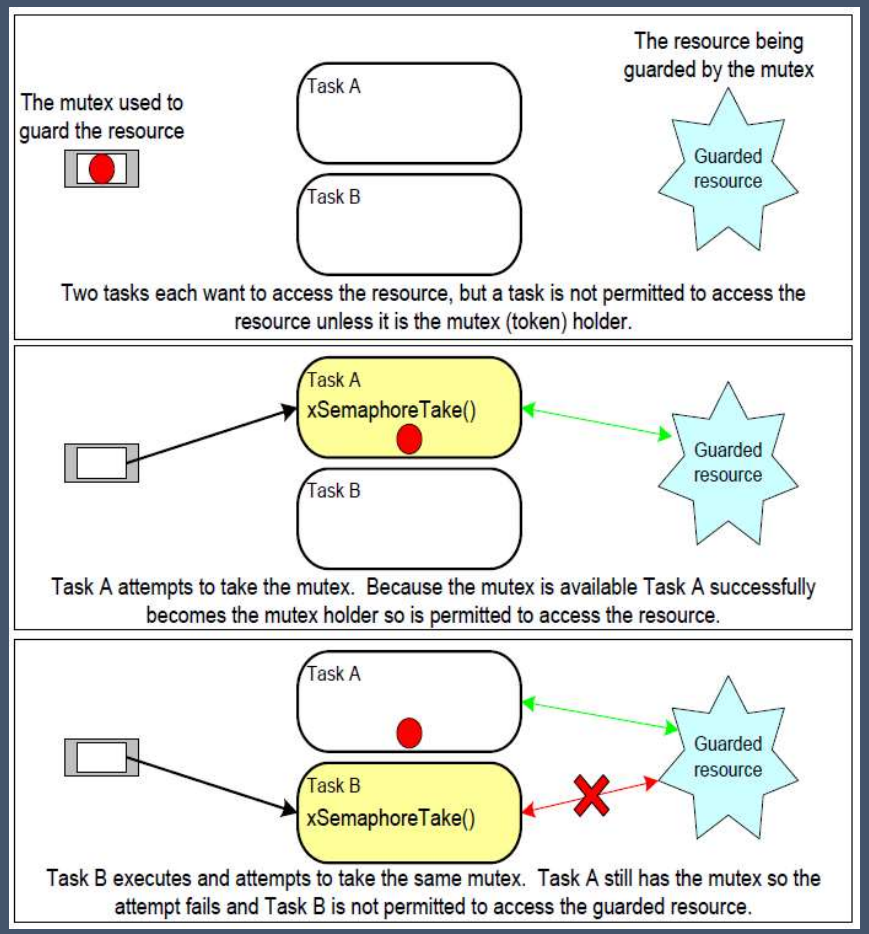
BaseType_t xTaskResumeAll(void);	
return value	스케줄러가 suspended동안 요청된 context switch는 pendding이고 스케줄러가 다시 시작되면 수행됨, xTaskResumeAll()전에 pending context switch가 수행된다면 pdTRUE, 그러지 않으면 pdFALSE

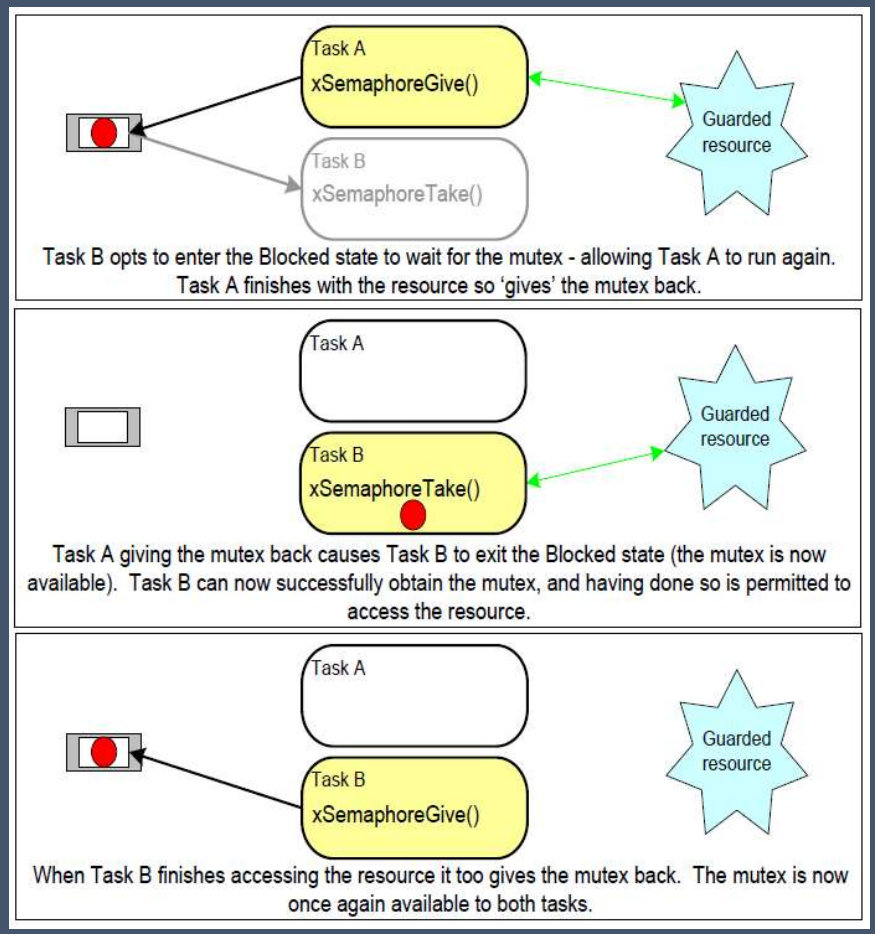
```
void func(void)
{
   vTaskSuspendAll();
   // to do
   xTaskResumeAll();
}
```

FreeRTOS - Resource Management

Mutexes

Mutex: 두 개 이상의 task에 공유되는 리소스에 대한 접근을 제어하는 데 사용되는 특수 유형의 binary Semaphore
configUSE_MUTEXES ("FreeRTOSConfig.h") 설정 1





FreeRTOS - Resource Management

Mutexes

xSemaphoreCreateMutex() : semaphore타입 mutex생성

SemaphoreHandle_t xSemaphoreCreateMutex(void);	
return value	NULL: 생성 실패 non-NULL: 성공,SemaphoreHandle_t 타입 handle 반환

FreeRTOS - Resource Management

Example22 – mutex를 사용한 printf

FreeRTOS - Resource Management

Mutexes

Deadlock

Deadlock은 두 task가 서로 보유하고 있는 resource를 서로 기다리고 있지 때문에 두 task가 진행되지 않을때 발생한다.

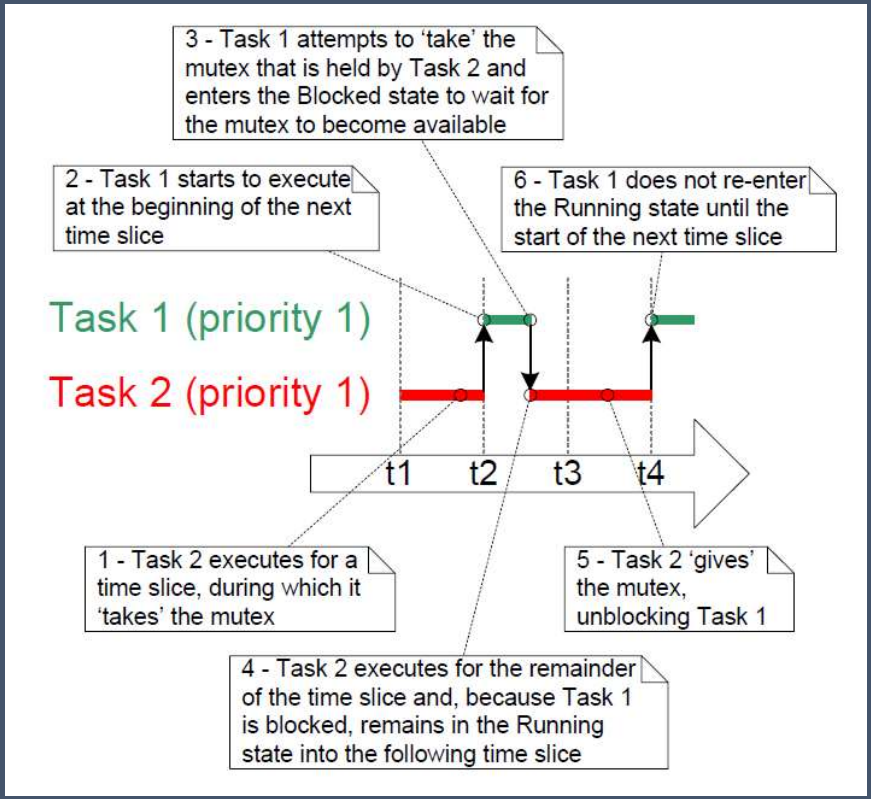
1. task A는 mutex X를 take
2. task B가 task A를 선점
3. task B는 mutex Y를 take하고 mutex X를 기다림
4. 다시 task A가 실행되고 mutex Y를 기다림

최우선적으로 Deadlock이 발생하지 않게 설계, 기본적으로 mutex를 얻기위해 대기하는 시간에 제한을 준다.

FreeRTOS - Resource Management

Mutexes

Mutex and Task Scheduling
task가 동일한 우선순위를 가질 때 task1이 task2의 mutex를 기다리며 blocked state일때 task2가 mutex를 반환하더라도 task1이 선점되지 않는다.
task2는 Running state를 유지, task1은 Ready state로 변경



FreeRTOS - Resource Management

Mutexes

Gatekeeper Tasks
상호배제를 구현하는 깔끔한 방법
Gatekeeper task는 resource의 단독 소유권을 갖는 기때문에 resouce에 access해야 하는 다른 task는 gatekeeper task를 통해서 간접적으로 수행

Tick hook 함수
configUSE_TICK_HOOK ("FreeRTOSConfig.h") 설정 1

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

FreeRTOS - Resource Management

Example23 – Gatekeeper task

감사합니다.