Xilinx Zynq FPGA, TI DSP, MCU 기반의 회로 설계 및 임베디드 전문가 과정

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - TaeYoung Eun(은태영)

zero_bird@naver.com

1.6 세마포어 API

1.6.1 세마포어, 뮤텍스 생성 및 제거

1.6.1.1 vSemaphoreCreateBinary : xSemaphoreCreateBinary 매크로 함수

```
#include “FreeRTOS.h”
#include “semphr.h”

void vSemaphoreCreateBinary ( SemaphoreHandle_t xSemaphore );
```

vSemaphoreCreateBinary()매크로는 이전 버전과의 호환성을 유지하기 위해 소스 코드에 남아있다. 새로운 디자인에서는 xSemaphoreCreateBinary()함수를 사용한다.

바이너리 세마포어를 만드는 매크로 함수로, 세마포어를 사용하기 전에 명시적으로 만들어야한다.

바이너리 세마포어는 뮤텍스와 비슷하지만 약간의 차이점이 존재한다. 뮤텍스는 우선 순위 상속 메커니즘을 포함하지만, 바이너리 세마포어는 그것이 포함되지 않는다. 따라서, 바이너리 세마포어가 동기화(task 간 또는 task 와 인터럽트 간)를 구현하거나, 간단한 상호 배제를 구현하는데 있어서 더 나은 선택이 된다.

바이너리 세마포어 : 동기화에 사용되는 바이너리 세마포어는 성공적으로 ‘taken(획득)’된 후 다시 ‘given’될 필요가 없다. task 동기화는 하나의 task 또는 인터럽트가 세마포어를 ‘give’하고, 다른 task 가 세마포어를 ’take’하여 구현한다. (xSemaphoreGiveFromISR 참조한다.)

뮤텍스 : 뮤텍스를 보유하고 있는 task 의 우선 순위는, 우선 순위가 다른 task 가 동일한 뮤텍스를 얻으려고 하면 발생한다. 이미 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 ‘take’하는 task 의 우선순위를 ‘inherit(상속)’한다. 뮤텍스가 리턴될 때 상속된 우선순위는 ‘disinherited’된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴 될 때 원래 우선순위로 돌아간다.)

상호 배제에 사용되는 뮤텍스를 얻는 task 는 반드시 뮤텍스를 다시 제공해야 한다. 그렇지 않으면, 다른 task 가 같은 뮤텍스를 얻을 수 없다. 상호 배제를 구현하는데 사용되는 뮤텍스의 예는 xSemaphoreTake()섹션에 나와 있다.

뮤텍스와 바이너리 세마포어는 모두 SemaphoreHandle_t 유형을 가진 변수로 참조되며, 해당 유형의 매개 변수를 사용하는 API 함수에서 사용할 수 있다.

xSemaphoreCreateBinary()함수와는 다르게 vSemaphoreCreateBinary()매크로를 사용하여 생성된 뮤텍스와 바이너리 세마포어는 모두 세마포어나 뮤텍스에서 xSemaphoreTake()에 대해 첫번째 호출이 전달되도록 생성된다. 참고로 vSemaphoreCreateBinary()은 더이상 사용되지 않으므로 새 응용 프로그램에서 사용하면 안된다. xSemaphoreCreateBinary()함수를 사용하여 생성된 바이너리 세마포어는 ‘empty’ 으로 만들어지므로, xSemaphoreTake()에 대한 호출을 사용하여 세마포어를 가져올 수 있기 전에 먼저 세마포어를 제공해야한다.

1.6.1.1.1 매개 변수

xSemaphore : 생성되는 세마포어의 핸들을 저장할 SemaphoreHandle_t 유형의 변수이다.

해당 값이 NULL 일 경우, FreeRTOS 의 힙 메모리가 충분하지 않기 때문에 세마포어를 만들 수 없는 상태이다.

1.6.1.1.2 기타

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters ) {

    // 세마포어를 생성한다.
    vSemaphoreCreateBinary( xSemaphore );
```

```
    if( xSemaphore == NULL ) {
      // 세마포어를 성공적으로 만들수 있는 FreeRTOS 의 힙 메모리가 부족하다.
    } else {
      // 세마포어를 사용할 수 있다. 핸들은 xSemaphore 변수에 저장된다.
    }
}
```

1.6.1.2 xSemaphoreCreateBinary : 바이너리 세마포어 동적 생성

```
#include "FreeRTOS.h"
#include "semphr.h"

SemaphoreHandle_t xSemaphoreCreateBinary ( void );
```

바이너리 세마포어를 생성하고, 세마포어를 참조할 수 있는 핸들을 반환한다. 각 바이너리 세마포어는 세마포어 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다.

xSemaphoreCreateBinary()를 사용하여 바이너리 세마포어를 만든 경우 필요한 RAM 이 FreeRTOS 힙 메모리에 자동으로 할당된다. xSemaphoreCreateBinaryStatic()을 사용하여 바이너리 세마포어를 만든 경우 응용 프로그램 작성자가 RAM 을 제공하므로 추가 매개 변수가 필요하지만 컴파일 타임에 RAM 을 정적으로 할당할 수 있다.

세마포어는 '비어 있는'상태로 생성되며, xSemaphoreTake()함수를 사용하여 세마포어를 가져오려면(획득) 먼저 세마포어를 제공해야 한다.

Direct task notificationss 은 보통 바이너리 세마포어보다 가벼우며 빠른 대안을 제공한다.

바이너리 세마포어는 뮤텍스와 비슷하지만 약간의 차이점이 존재한다. 뮤텍스는 우선 순위 상속 메커니즘을 포함하지만, 바이너리 세마포어는 그것이 포함되지 않는다. 따라서, 바이너리 세마포어가 동기화(task 간 또는 task 와 인터럽트 간)를 구현하거나, 간단한 상호 배제를 구현하는데 있어서 더 나은 선택이 된다.

바이너리 세마포어 : 동기화에 사용되는 바이너리 세마포어는 성공적으로 'taken(획득)'된 후 다시 'given'될 필요가 없다. task 동기화는 하나의 task 또는 인터럽트가 세마포어를 'give'하고, 다른 task 가 세마포어를 'take'하여 구현한다. (xSemaphoreGiveFromISR 참조한다.) 직접적인 task 통보를 사용하면 동일한 기능을 보다 효율적으로 달성할 수 있다.

뮤텍스 : 뮤텍스를 보유하고 있는 task 의 우선 순위는, 우선 순위가 다른 task 가 동일한 뮤텍스를 얻으려고 하면 발생한다. 이미 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 'take'하는 task 의 우선순위를 'inherit(상속)'한다. 뮤텍스가 리턴될 때 상속된 우선순위는 'disinherited'된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴 될 때 원래 우선순위로 돌아간다.)

상호 배제에 사용되는 뮤텍스를 얻는 task 는 반드시 뮤텍스를 다시 제공해야 한다. 그렇지 않으면, 다른 task 가 같은 뮤텍스를 얻을 수 없다. 상호 배제를 구현하는데 사용되는 뮤텍스의 예는 xSemaphoreTake()섹션에 나와 있다.

뮤텍스와 바이너리 세마포어는 모두 SemaphoreHandle_t 유형을 가진 변수로 참조되며, 해당 유형의 매개 변수를 사용하는 API 함수에서 사용할 수 있다.

vSemaphoreCreateBinary()매크로를 사용하여 생성된 뮤텍스 및 바이너리 세마포어는 세마포어 또는 뮤텍스에서 xSemaphoreTake()에 대한 첫번째 호출이 전달되도록 생성된다. vSemaphoreCreateBinary()은 더이상 사용되지 않으므로 새 응용 프로그램에서 사용하면 안된다. xSemaphoreCreateBinary()함수를 사용하여 생성된 바이너리 세마포어는 '비어 있는'상태로 만들어 지므로, xSemaphoreTake()에 대한 호출을 사용하기 전에 먼저 세마포어를 제공해야 한다.

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 을 1 로 설정하거나 정의되지 않은 상태로 두어야 한다.

1.6.1.2.1 반환 값

NULL : 세마포어의 데이터 구조를 할당하는데 사용되는 FreeRTOS 힙 메모리가 충분하지 않아 세마포어를 만들 수 없다.

다른 값 : 세마포어가 성공적으로 만들어졌다. 반환된 값은 생성된 세마포어를 참조할 수 있는 핸들이다.

1.6.1.2.2 기타

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters ) {
    // 세마포어를 만든다.
    xSemaphore = xSemaphoreCreateBinary();

    if( xSemaphore == NULL ) {
      // FreeRTOS 힙 메모리가 부족하여 세마포어 생성에 실패했다.
    } else {
      // 세마포어를 사용할 수 있다. 핸들은 xSemaphore 변수에 저장된다.
      // 여기에서 세마포어에 대해 xSemaphoreTake()를 호출하면 세마포어가 제공될 때 까지 실패한다.
    }
}
```

1.6.1.3 xSemaphoreCreateBinaryStatic : 바이너리 세마포어 정적 생성

```
#include "FreeRTOS.h"
#include "semphr.h"

SemaphoreHandle_t xSemaphoreCreateBinaryStatic ( StaticSemaphore_t * pxSemaphoreBuffer );
```

바이너리 세마포어를 생성하고, 세마포어를 참조할 수 있는 핸들을 반환한다. 각 바이너리 세마포어는 세마포어 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다.

xSemaphoreCreateBinary()를 사용하여 바이너리 세마포어를 만든 경우 필요한 RAM 이 FreeRTOS 힙 메모리에 자동으로 할당된다. xSemaphoreCreateBinaryStatic()을 사용하여 바이너리 세마포어를 만든 경우 응용 프로그램 작성자가 RAM 을 제공하므로 추가 매개 변수가 필요하지만 컴파일 타임에 RAM 을 정적으로 할당할 수 있다.

세마포어는 '비어 있는'상태로 생성되며, xSemaphoreTake()함수를 사용하여 세마포어를 가져오려면(획득) 먼저 세마포어를 제공해야 한다.

Direct task notificationss 은 보통 바이너리 세마포어보다 가벼우며 빠른 대안을 제공한다.

바이너리 세마포어는 뮤텍스와 비슷하지만 약간의 차이점이 존재한다. 뮤텍스는 우선 순위 상속 메커니즘을 포함하지만, 바이너리 세마포어는 그것이 포함되지 않는다. 따라서, 바이너리 세마포어가 동기화(task 간 또는 task 와 인터럽트 간)를 구현하거나, 간단한 상호 배제를 구현하는데 있어서 더 나은 선택이 된다.

바이너리 세마포어 : 동기화에 사용되는 바이너리 세마포어는 성공적으로 'taken(획득)'된 후 다시 'given'될 필요가 없다. task 동기화는 하나의 task 또는 인터럽트가 세마포어를 'give'하고, 다른 task 가 세마포어를 'take'하여 구현한다. (xSemaphoreGiveFromISR 참조한다.) 직접적인 task 통보를 사용하면 동일한 기능을 보다 효율적으로 달성할 수 있다.

뮤텍스 : 뮤텍스를 보유하고 있는 task 의 우선 순위는, 우선 순위가 다른 task 가 동일한 뮤텍스를 얻으려고 하면 발생한다. 이미 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 'take'하는 task 의 우선순위를 'inherit(상속)'한다. 뮤텍스가 리턴될 때 상속된 우선순위는 'disinherited'된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴 될 때 원래 우선순위로 돌아간다.)

상호 배제에 사용되는 뮤텍스를 얻는 task 는 반드시 뮤텍스를 다시 제공해야 한다. 그렇지 않으면, 다른 task 가 같은 뮤텍스를 얻을 수 없다. 상호 배제를 구현하는데 사용되는 뮤텍스의 예는 xSemaphoreTake()섹션에 나와 있다.

뮤텍스와 바이너리 세마포어는 모두 SemaphoreHandle_t 유형을 가진 변수로 참조되며, 해당 유형의 매개 변수를 사용하는 API 함수에서 사용할 수 있다.

vSemaphoreCreateBinary()매크로를 사용하여 생성된 뮤텍스 및 바이너리 세마포어는 세마포어 또는 뮤텍스에서 xSemaphoreTake()에 대한 첫번째 호출이 전달되도록 생성된다. vSemaphoreCreateBinary()은 더이상 사용되지 않으므로 새 응용 프로그램에서 사용하면 안된다. xSemaphoreCreateBinary()함수를 사용하여 생성된 바이너리 세마포어는 '비어 있는'상태로 만들어 지므로, xSemaphoreTake()에 대한 호출을 사용하기 전에 먼저 세마포어를 제공해야 한다.

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 을 1 로 설정하거나 정의되지 않은 상태로 두어야 한다.

1.6.1.3.1 매개 변수

pxSemaphoreBuffer : 세마포어를 유지하는데 사용할 StaticSemaphore_t 유형의 변수를 가리킨다.

1.6.1.3.2 반환 값

NULL : pxSemaphoreBuffer 가 NULL 이기 때문에 세마포어를 만들 수 없다.

다른 값 : 세마포어가 성공적으로 만들어졌다. 반환된 값은 생성된 세마포어를 참조할 수 있는 핸들이다.

1.6.1.3.3 기타

```
SemaphoreHandle_t xSemaphoreHandle;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters ) {
    // 동적 메모리 할당 없이 바이너리 세마포어를 만든다.
    xSemaphoreHandle = xSemaphoreCreateBinaryStatic( &xSemaphoreBuffer );

    /* pxSemaphoreBuffer 가 NULL 이 아니기 때문에 바이너리 세마포어가 생성되고,
    xSemaphoreHandle 에 유효한 핸들이 저장된다. 나머지 코드는 여기에 작성한다. */
}
```

1.6.1.4 xSemaphoreCreateCounting : 카운트 세마포어 동적 생성

```
#include “FreeRTOS.h”
#include “semphr.h”

SemaphoreHandle_t xSemaphoreCreateCounting ( UBaseType_t uxMaxCout, UBaseType_t uxInitialCount );
```

카운터 세마포어를 만들고 세마포어를 참조할 수 있는 핸들을 반환한다. 각 카운터 세마포어는 세마포어 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다. xSemaphoreCreateCounting()를 사용하여 카운터 세마포어가 생성되면 필요한 RAM 이 FreeRTOS 힙 메모리에서 자동으로 할당된다. xSemaphoreCreateCountingStatic()을 사용하여 카운터 세마포어를 만든 경우, 추가 매개 변수가 필요로 하고 컴파일 타임에 정적으로 할당할 수 있는 RAM 을 응용 프로그램 작성자가 제공한다.

Direct task notificationss 는 일반적으로 세마포어를 계산하는데 더 가벼우면서도 빠른 대안을 제공한다.

카운터 세마포어는 일반적으로 두 가지 용도로 사용된다.

1. 카운터 이벤트

이벤트 핸들러는 이벤트가 발생할 때마다 세마포어를 '제공'하고 핸들러 task 는 이벤트를 처리할 때마다 세마포어를 가져온다. 세마포어의 카운터 값은 'given'이 될 때마다 증가하고 'taken'될 때마다 감소합니다. 즉, 카운트 값은 발생한 이벤트 수와 처리된 이벤트 수의 차이다. 세마포어가 생성하기 전에는 이벤트가 계산되지 않으므로, 이벤트 카운터를 위해 생성된 세마포어는 초기 카운트 값을 0 으로 해야 한다.

2. 자원 관리

세마포어의 카운트 값은 사용 가능한 리소스의 수를 나타낸다. 자원의 제어권을 얻으려면 task 가 먼저 세마포어를 성공적으로 '가져와야 한다. 세마포어를 가져오는 동작은 세마포어의 카운트 값을 감소시킨다. 카운트 값이 0 이되면 더 이상 사용할 수있는 자원이 없으므로 세마포어를 가져오는 시도가 실패한다.

자원으로 작업이 완료되면 세마포어를 제공해야 한다. 세마포어를 주는 동작은 세마포어의 카운트 값을 증가시켜, 자원이 사용 가능한 상태를 나타내며, 세마포어를 성공적으로 가져오기 시도를 허용한다. 사용 가능한 자원의 수와 동일한 초기 계수 값으로, 자원을 관리하기 위해 작성된 세마포어를 생성해야 한다.

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 을 1 로 설정하거나 정의되지 않은 상태로 둔다.

1.6.1.4.1 매개 변수

uxMaxCount : 도달할 수 있는 최대 카운트 값이다. 세마포어가 값에 도달하면 더이상 'given'될 수 없다.

uxInitialCount : 세마포어를 생성할 때 세마포어에 할당된 카운트 값이다.

1.6.1.4.2 반환 값

NULL : 세마포어 데이터 구조를 할당하기 위한 FreeRTOS 힙 메모리가 부족하여, 세마포어를 만들 수없는 경우 반환된다.

다른 값 : 세마포가 성공적으로 만들어졌다. 반환된 값은 생성된 세마포어를 참조하는 핸들이다.

1.6.1.4.3 기타

```
void vATask( void * pvParameters ) {
    SemaphoreHandle_t xSemaphore;

    /* 세마포어는 xSemaphoreCreateCounting()에 대한 호출을 사용하여 생성되기 전에 사용할 수 없다. 이 예제에서는 세마포어가
    계산할 수 있는 최대 값을 10 으로 설정하고, 카운트에 할당된 초기 값은 0 으로 설정된다. */
    xSemaphore = xSemaphoreCreateCounting( 10, 0 );

    if( xSemaphore != NULL ) {
      // 세마포어가 성공적으로 생성되었다. 이제 세마포어를 사용할 수 있다.
    }
}
```

1.6.1.5 xSemaphoreCreateCountingStatic : 카운트 세마포어 정적 생성

```
#include "FreeRTOS.h"
#include "semphr.h"

SemaphoreHandle_t xSemaphoreCreateCountingStatic ( UBaseType_t uxMaxCout, UBaseType_t uxInitialCount,
                                                   StaticSemaphore_t pxSempahoreBuffer );
```

카운터 세마포어를 만들고 세마포어를 참조할 수 있는 핸들을 반환한다. 각 카운터 세마포어는 세마포어 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다. xSemaphoreCreateCounting()를 사용하여 카운터 세마포어가 생성되면 필요한 RAM 이 FreeRTOS 힙 메모리에서 자동으로 할당된다. xSemaphoreCreateCountingStatic()을 사용하여 카운터 세마포어를 만든 경우, 추가 매개 변수가 필요로 하고 컴파일 타임에 정적으로 할당할 수 있는 RAM 을 응용 프로그램 작성자가 제공한다.

Direct task notificationss 는 일반적으로 세마포어를 계산하는데 더 가벼우면서도 빠른 대안을 제공한다.

카운터 세마포어는 일반적으로 두 가지 용도로 사용된다.

1. 카운터 이벤트

이벤트 핸들러는 이벤트가 발생할 때마다 세마포어를 '제공'하고 핸들러 task 는 이벤트를 처리할 때마다 세마포어를 가져온다. 세마포어의 카운터 값은 'given'이 될 때마다 증가하고 'taken'될 때마다 감소합니다. 즉, 카운트 값은 발생한 이벤트 수와 처리된 이벤트 수의 차이다. 세마포어가 생성하기 전에는 이벤트가 계산되지 않으므로, 이벤트 카운터를 위해 생성된 세마포어는 초기 카운트 값을 0 으로 해야 한다.

2. 자원 관리

세마포어의 카운트 값은 사용 가능한 리소스의 수를 나타낸다. 자원의 제어권을 얻으려면 task 가 먼저 세마포어를 성공적으로 '가져와야 한다. 세마포어를 가져오는 동작은 세마포어의 카운트 값을 감소시킨다. 카운트 값이 0 이되면 더 이상 사용할 수있는 자원이 없으므로 세마포어를 가져오는 시도가 실패한다.

자원으로 작업이 완료되면 세마포어를 제공해야 한다. 세마포어를 주는 동작은 세마포어의 카운트 값을 증가시켜, 자원이 사용 가능한 상태를 나타내며, 세마포어를 성공적으로 가져오기 시도를 허용한다. 사용 가능한 자원의 수와 동일한 초기 계수 값으로, 자원을 관리하기 위해 작성된 세마포어를 생성해야 한다.

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_STATIC_ALLOCATION 을 1 로 설정해야 한다.

1.6.1.5.1 매개 변수

uxMaxCount : 도달할 수 있는 최대 카운트 값이다. 세마포어가 이 값에 도달하면 더이상 지정될 수 없다.

uxInitialCount : 세마포어를 생성할 때 세마포어에 할당된 카운트 값이다.

pxSemaphoreBuffer : 세마포어 상태를 유지하는데 사용할 StaticSemaphore_t 유형의 변수를 가리킨다.

1.6.1.5.2 반환 값

NULL : pxSemaphoreBuffer 가 NIULL 이기 때문에 세마포어를 만들 수 없다.

다른 값 : 세마포가 성공적으로 만들어졌다. 반환된 값은 생성된 세마포어를 참조하는 핸들이다

1.6.1.5.3 기타

```
void vATask( void * pvParameters ) {
    SemaphoreHandle_t xSemaphoreHandle;
    StaticSemaphore_t xSemaphoreBuffer;

    /* 동적 메모리 할당을 사용하지 않고 카운터 세마포어를 만든다.
    이 예제에서 세마포어가 계산할 수 있는 최대 값은 10 으로 설정되어 있고, 카운트에 할당된 초기 값은 0 이다. */
    xSemaphoreHandle = xSemaphoreCreateCountingStatic( 10, 0, &xSemaphoreBuffer );

    // pxSemaphoreBuffer 매개 변수가 NULL 이 아니므로, 세마포어가 작성되어 사용될 준비가 되어있다.
}
```

1.6.1.6 xSemaphoreCreateMutex : 뮤텍스 동적 생성

```
#include “FreeRTOS.h”
#include “semphr.h”

SemaphoreHandle_t xSemaphoreCreateMutex ( void );
```

뮤텍스 유형의 세마포어를 만들고 뮤텍스를 참조할 수 있는 핸들을 반환한다. 각 뮤텍스 유형 세마포어에는 세마포어 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다. xSemaphoreCreateMutex()를 사용하여 뮤텍스를 만들면 필요한 RAM 을 FreeRTOS 힙 메모리에서 자동으로 할당한다. xSemaphoreCreateMutexStatic()을 사용하여 뮤텍스를 만든 경우 추가 매개 변수가 필요하지만, 컴파일 타임에 RAM 을 정적으로 할당할 수 있으므로 응용 프로그램 작성자가 RAM 을 제공한다.

바이너리 세마포어는 뮤텍스와 비슷하지만 약간의 차이점이 존재한다. 뮤텍스는 우선 순위 상속 메커니즘을 포함하지만, 바이너리 세마포어는 그것이 포함되지 않는다. 따라서, 바이너리 세마포어가 동기화(task 간 또는 task 와 인터럽트 간)를 구현하거나, 간단한 상호 배제를 구현하는데 있어서 더 나은 선택이 된다.

바이너리 세마포어 : 동기화에 사용되는 바이너리 세마포어는 성공적으로 ‘taken(획득)’된 후 다시 ‘given’될 필요가 없다. task 동기화는 하나의 task 또는 인터럽트가 세마포어를 ‘give’하고, 다른 task 가 세마포어를 ’take’하여 구현한다. (xSemaphoreGiveFromISR 참조한다.) 직접적인 task 통보를 사용하면 동일한 기능을 보다 효율적으로 달성할 수 있다.

뮤텍스 : 뮤텍스를 보유하고 있는 task 의 우선 순위는, 우선 순위가 다른 task 가 동일한 뮤텍스를 얻으려고 하면 발생한다. 이미 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 ‘take’하는 task 의 우선순위를 ‘inherit(상속)’한다. 뮤텍스가 리턴될 때 상속된 우선순위는 ‘disinherited’된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴 될 때 원래 우선순위로 돌아간다.)

상호 배제에 사용되는 뮤텍스를 얻는 task 는 반드시 뮤텍스를 다시 제공해야 한다. 그렇지 않으면, 다른 task 가 같은 뮤텍스를 얻을 수 없다. 상호 배제를 구현하는데 사용되는 뮤텍스의 예는 xSemaphoreTake()섹션에 나와 있다.

뮤텍스와 바이너리 세마포어는 모두 SemaphoreHandle_t 유형을 가진 변수로 참조되며, 해당 유형의 매개 변수를 사용하는 API 함수에서 사용할 수 있다.

vSemaphoreCreateBinary()매크로를 사용하여 생성된 뮤텍스 및 바이너리 세마포어는 세마포어 또는 뮤텍스에서 xSemaphoreTake()에 대한 첫번째 호출이 전달되도록 생성된다. vSemaphoreCreateBinary()은 더이상 사용되지 않으므로 새 응용 프로그램에서 사용하면 안된다. xSemaphoreCreateBinary()함수를 사용하여 생성된 바이너리 세마포어는 ‘비어 있는’상태로 만들어 지므로, xSemaphoreTake()에 대한 호출을 사용하기 전에 먼저 세마포어를 제공해야 한다.

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 을 1 로 설정하거나 정의되지 않은 상태로 두어야 한다.

1.6.1.6.1 반환 값

NULL : 세마포어 데이터 구조를 할당하는데 있어 FreeRTOS 힙 메모리가 부족하여, 세마포어 생성에 실패한 경우 반환한다.

다른 값 : 세마포어가 성공적으로 만들어졌다. 반환된 값은 생성된 세마포어를 참조할 수 있는 핸들이다.

1.6.1.6.2 기타

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters ) {
    // 뮤텍스 유형의 세마포어를 만든다.
    xSemaphore = xSemaphoreCreateMutex();

    if( xSemaphore == NULL ) {

      // 뮤텍스에서 사용할 수 있는 힙 메모리가 부족하여 세마포어 생성에 실패하였다.
```

```
    } else {
      // 뮤텍스를 사용할 수 있다. 생성된 뮤텍스의 핸들은 xSemaphore 변수에 저장된다.
    }
}
```

1.6.1.7 xSemaphoreCreateMutexStatic : 뮤텍스 정적 생성

```
#include "FreeRTOS.h"
#include "semphr.h"

SemaphoreHandle_t xSemaphoreCreateMutexStatic ( StaticSemaphore_t * pxMutexBuffer );
```

뮤텍스 유형의 세마포어를 만들고 뮤텍스를 참조할 수 있는 핸들을 반환한다. 각 뮤텍스 유형 세마포어에는 세마포어 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다. xSemaphoreCreateMutex()를 사용하여 뮤텍스를 만들면 필요한 RAM 을 FreeRTOS 힙 메모리에서 자동으로 할당한다. xSemaphoreCreateMutexStatic()을 사용하여 뮤텍스를 만든 경우 추가 매개 변수가 필요하지만, 컴파일 타임에 RAM 을 정적으로 할당할 수 있으므로 응용 프로그램 작성자가 RAM 을 제공한다.

바이너리 세마포어는 뮤텍스와 비슷하지만 약간의 차이점이 존재한다. 뮤텍스는 우선 순위 상속 메커니즘을 포함하지만, 바이너리 세마포어는 그것이 포함되지 않는다. 따라서, 바이너리 세마포어가 동기화(task 간 또는 task 와 인터럽트 간)를 구현하거나, 간단한 상호 배제를 구현하는데 있어서 더 나은 선택이 된다.

바이너리 세마포어 : 동기화에 사용되는 바이너리 세마포어는 성공적으로 'taken(획득)'된 후 다시 'given'될 필요가 없다. task 동기화는 하나의 task 또는 인터럽트가 세마포어를 'give'하고, 다른 task 가 세마포어를 'take'하여 구현한다. (xSemaphoreGiveFromISR 참조한다.) 직접적인 task 통보를 사용하면 동일한 기능을 보다 효율적으로 달성할 수 있다.

뮤텍스 : 뮤텍스를 보유하고 있는 task 의 우선 순위는, 우선 순위가 다른 task 가 동일한 뮤텍스를 얻으려고 하면 발생한다. 이미 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 'take'하는 task 의 우선순위를 'inherit(상속)'한다. 뮤텍스가 리턴될 때 상속된 우선순위는 'disinherited'된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴 될 때 원래 우선순위로 돌아간다.)

상호 배제에 사용되는 뮤텍스를 얻는 task 는 반드시 뮤텍스를 다시 제공해야 한다. 그렇지 않으면, 다른 task 가 같은 뮤텍스를 얻을 수 없다. 상호 배제를 구현하는데 사용되는 뮤텍스의 예는 xSemaphoreTake()섹션에 나와 있다.

뮤텍스와 바이너리 세마포어는 모두 SemaphoreHandle_t 유형을 가진 변수로 참조되며, 해당 유형의 매개 변수를 사용하는 API 함수에서 사용할 수 있다.

vSemaphoreCreateBinary()매크로를 사용하여 생성된 뮤텍스 및 바이너리 세마포어는 세마포어 또는 뮤텍스에서 xSemaphoreTake()에 대한 첫번째 호출이 전달되도록 생성된다. vSemaphoreCreateBinary()은 더이상 사용되지 않으므로 새 응용 프로그램에서 사용하면 안된다. xSemaphoreCreateBinary()함수를 사용하여 생성된 바이너리 세마포어는 '비어 있는'상태로 만들어 지므로, xSemaphoreTake()에 대한 호출을 사용하기 전에 먼저 세마포어를 제공해야 한다.

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_STATIC_ALLOCATION 을 1 로 설정하거나 정의되지 않은 상태로 두어야 한다.

1.6.1.7.1 매개 변수

pxMutexBuffer : 뮤텍스 상태를 유지하는데 사용될 StaticSemaphore_t 유형의 변수를 가리킨다.

1.6.1.7.2 반환 값

NULL : pxMutexBuffer 가 NULL 이기 때문에 뮤텍스를 만들 수 없다.

다른 값 : 뮤텍스가 성공적으로 만들어졌다. 반환된 값은 생성된 뮤텍스를 참조할 수 있는 핸들이다.

1.6.1.7.3 기타

```
SemaphoreHandle_t xSemaphoreHandle;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters ) {
    // 동적 메모리 할당을 사용하지 않고 뮤텍스를 만든다.
    xSemaphoreHandle = xSemaphoreCreateMutexStatic( &xSemaphoreBuffer );

    // pxMutexBuffer 매개 변수가 NULL 이 아니기 때문에 뮤텍스가 생성되어 사용할 준비가 되어 있다.
}
```

1.6.1.8 xSemaphoreCreateRecursiveMutex : 재귀 뮤텍스 동적 생성

```
#include “FreeRTOS.h”
#include “semphr.h”

SemaphoreHandle_t xSemaphoreCreateRecursiveMutex ( void );
```

재귀하는 뮤텍스 유형의 세마포어를 만들고 재귀 뮤텍스를 참조할 수 있는 핸들을 반환한다. 재귀 뮤텍스에는 뮤텍스의 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다. 재귀 뮤텍스가 xSemaphoreCreateRecursiveMutex()를 사용하여 생성된 경우 필요한 RAM 은 FreeRTOS 힙 메모리에서 자동으로 할당한다. 재귀 뮤텍스가 xSemaphoreCreateRecursiveMutexStatic()를 사용하여 생성된 경우 추가 매개 변수가 필요로 하지만, 컴파일 타임에 응용 프로그램 작성자가 RAM 을 정적으로 할당할 수 있다.

재귀 뮤텍스는 xSemaphoreTakeRecursive()를 사용하여 가져오고, xSemaphoreGiveRecursive()를 사용하여 준다. xSemaphoreTake() 및 xSemaphoreGive()함수는 재귀 뮤텍스와 함께 사용하면 안된다.

xSemaphoreTakeRecursive()호출은 중첩될 수 있다. 따라서 재귀 뮤텍스가 task 에 의해 성공적으로 획득하면 동일한 task 로 만들어진 xSemaphoreTakeRecursive()에 대한 추가 호출도 성공적일 것이다. 이전에 xSemaphoreTakeRecursive()에서 했던 것처럼 xSemaphoreGiveRecursive()에 동일한 수의 호출이 이루어 져야 뮤텍스가 다른 task 에 사용 가능해진다.

예를 들어 task 가 재귀로 동일한 뮤텍스를 5 번 가져올 경우, 뮤텍스를 성공적으로 획득한 task 가 뮤텍스를 정확히 5 번 돌려줄 때까지는 다른 task 에서 뮤텍스를 사용할 수 없다. 표준 뮤텍스와 마찬가지로 재귀 뮤텍스는 한번에 하나의 task 만 보유(획득)할 수 있다. 재귀 뮤텍스를 보유하고 있는 task 의 우선 순위는 다른 우선 순위가 높은 task 가 같은 뮤텍를 얻으려고하면 발생한다. 이미 재귀 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 가져오는 task 의 우선순위를 상속한다. 뮤텍스가 반환될 때 상속된 우선순위는 ‘disinherited’된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴될 때 원래의 우선 순위로 돌아간다.)

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 을 1 로 설정하거나, 정의되지 않은 상태로 두어야 한다.

xSemaphoreCreateRecursiveMutex()를 사용하려면 FreeRTOSConfig.h 의 configResult_Configuration_RECURSIVE_MUTEXES 을 1 로 설정해야 한다.

1.6.1.8.1 반환 값

NULL ; FreeRTOS 의 뮤텍스 데이터 구조를 할당하기 위해 사용되는 힙 메모리가 부족하여 세마포어를 만들수 없을때 반환한다.

다른 값 : 뮤텍스가 성공적으로 만들어 졌다. 반환된 값은 생성된 뮤텍스를 참조할 수 있는 핸들이다.

1.6.1.8.2 기타

```
void vATask( void * pvParameters ) {
    SemaphoreHandle_t xSemaphore;
```

```
    // 재귀 뮤텍스는 SemaphoreCreateRecursiveMutex()에 대한 호출을 사용하여 명시적으로 작성되기 전에는 사용할 수 없다.
    xSemaphore = xSemaphoreCreateRecursiveMutex();
    if( xSemaphore != NULL ) {
      // 재귀 뮤텍스 세마포어가 성공적으로 만들어 졌으며, 핸들을 xSemaphore 변수가 저장한다. 재귀 뮤텍스를 사용할 수 있다.
    }
}
```

1.6.1.9 xSemaphoreCreateRecursiveMutexStatic : 재귀 뮤텍스 정적 생성

```
#include "FreeRTOS.h"
#include "semphr.h"

SemaphoreHandle_t xSemaphoreCreateRecursiveMutex ( StaticSemaphore_t pxMutexBuffer );
```

재귀하는 뮤텍스 유형의 세마포어를 만들고 재귀 뮤텍스를 참조할 수 있는 핸들을 반환한다. 재귀 뮤텍스에는 뮤텍스의 상태를 유지하는데 사용되는 소량의 RAM 이 필요하다. 재귀 뮤텍스가 xSemaphoreCreateRecursiveMutex()를 사용하여 생성된 경우 필요한 RAM 은 FreeRTOS 힙 메모리에서 자동으로 할당한다. 재귀 뮤텍스가 xSemaphoreCreateRecursiveMutexStatic()를 사용하여 생성된 경우 추가 매개 변수가 필요로 하지만, 컴파일 타임에 응용 프로그램 작성자가 RAM 을 정적으로 할당할 수 있다.

재귀 뮤텍스는 xSemaphoreTakeRecursive()를 사용하여 가져오고, xSemaphoreGiveRecursive()를 사용하여 준다. xSemaphoreTake() 및 xSemaphoreGive()함수는 재귀 뮤텍스와 함께 사용하면 안된다.

xSemaphoreTakeRecursive()호출은 중첩될 수 있다. 따라서 재귀 뮤텍스가 task 에 의해 성공적으로 획득하면 동일한 task 로 만들어진 xSemaphoreTakeRecursive()에 대한 추가 호출도 성공적일 것이다. 이전에 xSemaphoreTakeRecursive()에서 했던 것처럼 xSemaphoreGiveRecursive()에 동일한 수의 호출이 이루어 져야 뮤텍스가 다른 task 에 사용 가능해진다.

예를 들어 task 가 재귀로 동일한 뮤텍스를 5 번 가져올 경우, 뮤텍스를 성공적으로 획득한 task 가 뮤텍스를 정확히 5 번 돌려줄 때까지는 다른 task 에서 뮤텍스를 사용할 수 없다. 표준 뮤텍스와 마찬가지로 재귀 뮤텍스는 한번에 하나의 task 만 보유(획득)할 수 있다. 재귀 뮤텍스를 보유하고 있는 task 의 우선 순위는 다른 우선 순위가 높은 task 가 같은 뮤텍를 얻으려고하면 발생한다. 이미 재귀 뮤텍스를 보유하고 있는 task 는 동일한 뮤텍스를 가져오는 task 의 우선순위를 상속한다. 뮤텍스가 반환될 때 상속된 우선순위는 'disinherited'된다. (뮤텍스를 보유하는 동안 더 높은 우선 순위를 계승한 task 는 뮤텍스가 리턴될 때 원래의 우선 순위로 돌아간다.)

이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_STATIC_ALLOCATION 을 1 로 설정해야 한다.

xSemaphoreCreateRecursiveMutexStatic()를 사용하려면 FreeRTOSConfig.h 의 configResult_Configuration_RECURSIVE_MUTEXES 을 1 로 설정해야 한다

1.6.1.9.1 매개 변수

pxMutexBuffer : 뮤텍스의 상태를 유지하는데 사용될 StaticSemaphore_t 유형의 변수를 가리킨다.

1.6.1.9.2 반환 값

NULL : pxMutexBuffer 가 NULL 이기 때문에 세마포어를 만들 수 없다.

다른 값 : 뮤텍스가 성공적으로 만들어 졌다. 반환된 값은 생성된 뮤텍스를 참조할 수 있는 핸들이다.

1.6.1.9.3 기타

```
void vATask( void * pvParameters ) {
    SemaphoreHandle_t xSemaphoreHandle;
    StaticSemaphore_t xSemaphoreBuffer;
```

```
    // 동적 메모리 할당을 사용하지 않고 재귀 뮤텍스를 만든다.
    xSemaphoreHandle = xSemaphoreCreateRecursiveMutexStatic( &xSemaphoreBuffer );

    // pxMutexBuffer 매개 변수가 NULL 이 아니어서 재귀 뮤텍스가 만들어 졌으므로, 사용할 준비가 되어있다.
}
```

1.6.1.10 vSemaphoreDelete : 세마포어 삭제

```
#include "FreeRTOS.h"
#include "semphr.h"

void vSemaphoreDelete ( SemaphoreHandle_t xSemaphore );
```

vSemaphoreCreateBinary(), xSemaphoreCreateCounting(), xSemaphoreCreateRecursiveMutex(), xSemaphoreCreateMutex()의 호출을 사용하여 이전에 생성하였던 세마포어를 삭제한다.

task 는 사용할 수 없는 세마포어를 얻으려고 하면 세마포어(선택적 타임 아웃 포함)에서 차단하도록 할 수 있다. 한번 차단된 task 가 있다면 세마포어를 삭제하면 안된다.

1.6.1.10.1 매개 변수

xSemaphore : 세마포어의 핸들이 삭제된다.

1.6.2 세마포어 기능 함수

1.6.2.1 uxSemaphoreGetCount : 세마포어 카운트 값 출력

```
#include “FreeRTOS.h”
#include “semphr.h”

UBaseType_t uxSemaphoreGetCount ( SemaphoreHandle_t xSemaphore );
```

세마포어의 카운트 값을 출력한다. 바이너리 세마포어는 0 또는 1 만 가질 수 있다. 카운터 세마포어는 카운터 세마포어를 만들 때 지정된 최대 수와 0 사이의 수를 가질 수 있다.

1.6.2.1.1 매개 변수

xSemaphore : 확인하고자 하는 세마포어의 핸들이다.

1.6.2.1.2 반환 값

xSemaphore 매개 변수에서 전달된 핸들에 의해 참조되는 세마포어의 수다.

1.6.2.2 xSemaphoreGetMutexHolder : 뮤텍스 보유 task 핸들 출력

```
#include “FreeRTOS.h”
#include “semphr.h”

TaskHandle_t xSemaphoreGetMutexHolder ( SemaphoreHandle_t xMutex );
```

함수의 매개 변수로 지정된 뮤텍스를 유지하는 task 의 핸들을 반환한다.

1.6.2.2.1 매개 변수

xMutex : 확인하고자 하는 뮤텍스 핸들이다.

1.6.2.2.2 반환 값

NULL : xMutex 매개 변수로 지정된 세마포어가 뮤텍스 유형의 세마포어가 아니거나, 세마포어를 사용할 수 있는 상태에서 어떤 task 도 보유하지 않을 때다.

다른 값 : xMutex 매개 변수로 지정된 세마포어를 유지하는 task 의 핸들이다.

1.6.2.3 xSemaphoreGive : 세마포어 반환

```
#include “FreeRTOS.h”
#include “semphr.h”

BaseType_t xSemaphoreGive ( SemaphoreHandle_t xSemaphore );
```

vSemaphoreCreateBinary(), xSemaphoreCreateCounting(), xSemaphoreCreateMutex()에 대한 호출을 사용하여 이전에 생성된 세마포어를 ‘gives’한다. 이를 통해 성공적으로 ‘taken’할 수 있다.

1.6.2.3.1 매개 변수

xSemaphore : ‘given’할 세마포어 핸들이다. SemaphoreHandle_t 유형의 변수에 의해 참조되며 사용 전에 명시적으로 작성되야 한다.

1.6.2.3.2 반환 값

pdPASS : 세마포어 ‘give’작업이 성공했다.

pdFAIL : xSemaphoreGive()을 호출하는 task 가 세마포어 홀더가 아니기 때문에 세마포어 ‘give’작업을 성공하지 못했다. task 가 성공적으로 ‘take’하려면 그 전에 세마포어를 ‘give’해야 한다.

1.6.2.3.3 기타

```
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters ) {
    // 공유 자원을 보호하기 위해 세마포어를 사용한다. 이 경우 뮤텍스 타입 세마포어는 우선 순위 상속 기능을 포함하고 있다.
    xSemaphore = xSemaphoreCreateMutex();

    for( ;; ) {
      if( xSemaphore != NULL ) {
        if( xSemaphoreGive( xSemaphore ) != pdTRUE ) {
          // 이 호출은 세마포어가 사용되지 않았기 때문에 실패한다.
        }
        // 세마포어를 획득한다. 세마포어가 즉시 사용 가능하지 않은 경우(지정된 시간이 0 일 때) 블로킹 하지 않는다.
        if( xSemaphoreTake( xSemaphore, 0 ) == pdPASS ) {
          // 세마포어를 성공적으로 가져왔으므로 보호하고 있는 리소스에 안전하게 액세스 할 수 있다.
          /* ... */
          // 세마포어가 지키고 있는 자원에 대한 액세스가 완료되었으므로 세마보어를 반환해야 한다.
          if( xSemaphoreGive( xSemaphore ) != pdPASS ) {
            // 이 호출은 호출하는 task 가 이미 성공적으로 세마포어를 획득했기 때문에 실패하지 않는다.
          }
        }
      } else {
      // 세마포어 데이터 구조를 생성할 FreeRTOS 힙 메모리가 충분하지 않아 세마포어가 성공적으로 생성되지 않았다.
      }
    }
}
```

1.6.2.4 xSemaphoreGiveFromISR : ISR 에서 세마포어 반환

```
#include “FreeRTOS.h”
#include “semphr.h”

BaseType_t xSemaphoreGiveFromISR ( SemaphoreHandle_t xSemaphore, BaseType_t * pxHigherPriorityTaskWoken );
```

ISR 에서 사용할 수 있는 xSemaphoreGive()버전이다. xSemaphoreGive()와 달리 xSemaphoreGiveISR()은 차단 시간을 지정할 수 없다.

1.6.2.4.1 매개 변수

xSemaphore : 반환할 세마포어다. SemaphoreHandle_t 유형의 변수에 의해 참조되며, 사용 전에 명시적으로 작성되야 한다.

*pxHigherPriorityTaskWoken : 단일 세마포어가 세마포어를 사용할 수 있게 되기를 기다리는, 차단된 하나 이상의 task 를 가질 수 있다. xSemaphoreGiveFromISR()을 호출하면 세마포어를 사용할 수 있게 되어 그러한 task 가 차단 상태에서 해제될 수 있다. xSemaphoreGiveFromISR()을 호출하면 차단 상태에서 벗어나고, 차단 해제된 task 의 우선 순위가 현재 실행중인 task(인터럽트 된 task)보다 높거나 같은 경우 내부적으로 xSemaphoreGiveFromISR()는 *pxHigherPriorityTaskWoken 를 pdTRUE 로 설정한다.

xSemaphoreGiveFromISR()가 이 값을 pdTRUE 로 설정하면 인터럽트가 종료되기 전에 컨텍스트 스위치가 수행되야 한다. 이렇게 하면 인터럽트가 가장 높은 우선 순위의 준비 상태 task 로 직접 반환된다.

FreeRTOS V7.3.0 부터 pxHigherPriorityTaskWoken 은 선택적 매개 변수가 되어 NULL 로 설정할 수 있다.

인터럽트 서비스 루틴(ISR)내에서 xSemaphoreGiveFromISR()을 호출하면 차단된 task 를 세마포어를 차단 해제 상태로 만들 수 있다. 차단 해제된 task 의 우선 순위가 해당 task 보다 높거나 같으면 컨텍스트 스위치를 수행해야 한다. 컨텍스트 스위치를 사용하면 인터럽트가 가장 높은 우선 순위의 준비 상태 task 로 직접 반환된다. xSemaphoreGive()와 달리 xSemaphoreGiveFromISR()은 컨텍스트 스위치를 수행하지 않는다. 대신 컨텍스트 스위치가 필요한지 여부만 나타낸다. xSemaphoreGiveFromISR()은 스케줄러가 시작되기 전에 호출되어서는 안된다.

1.6.2.4.2 반환 값

pdTRUE : xSemaphoreGiveFromISR()에 대한 호출이 성공했다.

errQUEUE_FULL : 세마포어를 이미 사용할 수 있는 경우 이를 지정할 수 없으며, errQUEUE_FULL 를 반환한다.

1.6.2.4.3 기타

```
#define LONG_TIME 0xffff
#define TICKS_TO_WAIT 10

SemaphoreHandle_t xSemaphore = NULL;

// 인터럽트가 발생할때마다 액션을 수행하는 task 를 정의한다. 인터럽트 처리는 이 task 로 지연된다.
// task 는 세마포어를 사용하여 인터럽트와 동기화 한다.
void vATask( void * pvParameters ) {
    // 세마포어가 이미 이 task 외부에서 생성되었다고 가정한다.
    for( ;; ) {
      // 다음 이벤트를 기다린다.
      if( xSemaphoreTake( xSemaphore, portMAX_DELAY ) == pdTRUE ) {
        // 이벤트가 발생했다. 해당 위치에서 처리한다

        // 처리가 완료되면 다음 이벤트를 기다린다.
      }
    }
}
```

```
// 처리를 요구하는 이벤트가 발생했음을 나타내는 세마포어를 사용하여 task 에 대한 처리를 지연시키는 ISR 이다.
void vISR( void * pvParameters ) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // 이벤트가 발생했다. 세마포어를 사용하여 task 의 차단을 해제하고, task 에서 이벤트를 처리할 수 있도록 한다.
    xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );

    // 여기서 인터럽트를 지운다.
    /* xHigherPriorityTaskWoken 가 pdTRUE 면 task 가 차단 해제되었기 때문에 컨텍스트 스위치를 수행해야 한다.
    ISR 에서 컨텍스트 스위치를 수행하는데 필요한 구문은 포트마다 다르며, 해당 포트에 대한 웹 설명서 및 예제를 확인해야 한다. */
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

1.6.2.5 xSemaphoreTake : 세마포어 획득

```
#include "FreeRTOS.h"
#include "semphr.h"

BaseType_t xSemaphoreTake ( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait );
```

vSemaphoreCreateBinary(), xSemaphoreCreateCounting(), xSemaphoreCreateMutex()에 대한 호출을 사용하여 이전에 생성된 세마포어를 획득한다. xSemaphoreTake()는 실행중인 task 에서만 호출되어야 하므로, 스케줄러가 초기화 상태(스케줄러가 시작되기 전)에 있는동안 호출해서는 안된다. 또한, 크리티컬 섹션 및 스케줄러가 일시 중지된 동안 호출되어서도 안된다.

1.6.2.5.1 매개 변수

xSemaphore : 세마포어를 획득한다. 세마포어는 SemaphoreHandle_t 유형의 변수에 의해 참조되며 사용되기 전에 명시적으로 작성되어야 한다.

xTicksToWait : 세마포어를 즉시 사용할 수 없는 경우, 세마포어를 사용할 수 있게 될 때까지 대기하는 최대 시간이다. xTicksToWait 을 0 으로 하면 세마포어를 사용할 수 없는 경우 xSemaphoreTake()가 즉시 반환된다. 차단 시간은 tick 시간으로 지정되므로 tick 시간의 절대 시간은 tick 주파수에 따라 달라진다.

pdMS_TO_TICKS()매크로는 밀리 초 단위로 지정된 시간을 tick 으로 변환하는데 사용된다. xTicksToWait 을 portMAX_DELAY 로 설정하고, FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우 task 가 시간초과 없이 무기한 대기하게 된다.

1.6.2.5.2 반환 값

pdPASS : xSemaphoreTake()에 대한 호출이 세마포어를 획득하는데 성공한 경우에 반환한다. 차단 시간이 지정되어 있으면(xTicksToWait 이 0 이 아닌 경우), 호출한 task 가 즉시 사용할 수 없을 때 세마포어를 기다리기 위해 차단 상태로 설정되지만, 차단 시간이 만료되기 전에 세마포어가 사용 가능해질 경우에도 반환한다.

pdFAIL : xSemaphoreTake()에 대한 호출이 세마포어를 성공적으로 획득하지 못한 경우 반환한다. 차단 시간이 지정되어 있으면(xTicksToWait 이 0 이 아닐 경우), 호출한 task 가 세마포어가 사용 가능하게 될 때까지 대기하기 위해 차단 상태로 설정되지만, 세마포어를 획득하기 전에 시간이 만료되면 반환한다.

1.6.2.5.3 기타

```
SemaphoreHandle_t xSemaphore = NULL;

// 뮤텍스 타입의 세마포어를 생성하는 task 다.
void vATask( void * pvParameters ) {
    // 공유 자원을 보호하기 위해 세마포어를 사용한다. 이 경우 뮤텍스 타입 세마포어는 우선순위 상속 기능을 포함하고 있다.
    xSemaphore = xSemaphoreCreateMutex();
```

```
    // 나머지 작업 코드가 위치한다.
    for( ;; ) { /* ... */ }
}

// 뮤텍스를 사용하는 task 다.
void vAnotherTask( void * pvParameters ) {
    for( ;; ) {
      // 다른 작업을 한다.
      if( xSemaphore != NULL ) {
        // 뮤텍스를 얻을 수 있는지 확인한다. 뮤텍스가 사용할 수 없을 때 10 tick 씩 기다리면서 해제한다.
        if( xSemaphoreTake( xSemaphore, 10 ) == pdTRUE ) {
          // 공유 자원을 안전하게 엑세스 할 수 있도록 뮤텍스가 성공적으로 획득하였다.

          // 공유 리소스에 대한 액세스가 완료되었으므로, 뮤텍스가 반환된다.
          xSemaphoreGive( xSemaphore );
        } else {
          // 10 tick 을 기다린 후에도 뮤텍스를 얻을 수 없으므로, 공유 리소스를 액세스 할 수 없다.
        }
      }
    }
}
```

1.6.2.6 xSemaphoreTakeFromISR : ISR 에서 세마포어 획득

```
#include "FreeRTOS.h"
#include "semphr.h"

BaseType_t xSemaphoreTakeFromISR ( SemaphoreHandle_t xSemaphore, signed BaseType_t * pxHigherPriorityTaskWoken );
```

ISR 에서 호출할 수 있는 xSemaphoreTake()버전이다.

xSemaphoreTake()와 달리 xSemaphoreTakeFromISR()는 차단 시간을 지정할 수 없다.

1.6.2.6.1 매개 변수

xSemaphore : 세마포어를 획득한다. 세마포어는 SemaphoreHandle_t 유형의 변수에 의해 참조되며, 사용하기 전에 명시적으로 작성되야 한다.

pxHigherPriorityTaskWoken : 세마포어 유형에 따라 세마포어가 세마포어를 제공하기 위해 대기하는 하나 이상의 task 를 차단할 수 있다. xSemaphoreTakeFromISR()를 호출하면 차단 상태의 task 가 차단 대기 상태로 떠난다.

세마포어를 반환하기 위해 대기하는 하나 이상의 task 가 있다. (가능성은 없지만 세마포어 유형에 따라 다르다.) xSemaphoreTakeFromISR ()을 호출하면 차단된 task 의 세마포어가 차단 상태를 벗어나 대기한다. API 함수를 호출하면 task 가 차단 상태를 벗어나고 차단되지 않은 작업의 우선 순위가 현재 실행중인 task(인터럽트 된 task)보다 크거나 같으면 내부적으로 API 함수가 *pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다. xSemaphoreTakeFromISR()이 *pxHigherPriorityTaskWoken 을 pdTRUE 로 설정하면 인터럽트가 종료되기 전에 컨텍스트 스위치가 수행되야 한다. 이렇게하면 인터럽트가 가장 높은 우선 순위의 준비 상태 태스크로 직접 반환된다. 이 메커니즘은 xQueueReceiveFromISR()함수에서 사용된 것과 동일하며 xQueueReceiveFromISR()설명서에서 추가 설명을 참조한다.

FreeRTOS V7.3.0 부터 pxHigherPriorityTaskWoken 은 선택적 매개 변수이며 NULL 로 설정할 수 있다.

1.6.2.6.2 반환 값

pdPASS : 세마포어를 성공적으로 획득했다.

pdFAIL : 사용할 수 없으므로 세마포어를 성공적으로 가져오지 못했다.

1.6.2.7 xSemaphoreGiveRecursive : 재귀 뮤텍스 반환

```
#include “FreeRTOS.h”
#include “semphr.h”

BaseType_t xSemaphoreGiveRecursive ( SemaphoreHandle_t xMutex );
```

xSemaphoreCreateRecursiveMutex()를 사용하여 이전에 생성된 재귀 뮤텍스 유형의 세마포어를 반환한다.

재귀 뮤텍스는 xSemaphoreTakeRecursive()를 사용하여 가져오고, xSemaphoreGiveRecursive()를 사용하여 준다. xSemaphoreTake() 및 xSemaphoreGive()함수는 재귀 뮤텍스와 함께 사용하면 안된다.

xSemaphoreTakeRecursive()호출은 중첩될 수 있다. 따라서 재귀 뮤텍스가 task 에 의해 성공적으로 획득하면 동일한 task 로 만들어진 xSemaphoreTakeRecursive()에 대한 추가 호출도 성공적일 것이다. 이전에 xSemaphoreTakeRecursive()에서 했던 것처럼 xSemaphoreGiveRecursive()에 동일한 수의 호출이 이루어 져야 뮤텍스가 다른 task 에 사용 가능해진다.

예를 들어 task 가 재귀로으로 동일한 뮤텍스를 5 번 가져올 경우, 뮤텍스를 성공적으로 획득한 task 가 뮤텍스를 정확히 5 번 돌려줄 때까지는 다른 task 에서 뮤텍스를 사용할 수 없다.

xSemaphoreGiveRecursive()는 실행중인 task 에서만 호출되어야 하므로, 스케줄러가 초기화 상태(스케줄러가 시작되기 전)에 있는동안 호출하면 안된다. 또한, 크리티컬 섹션 내에서나 스케줄러가 일시 중지된 동안 호출되어서도 안된다.

1.6.2.7.1 매개 변수

xMutex : 세마포어를 반환한다. 세마포어는 SemaphoreHandle_t 유형의 변수에 의해 참조되며, 사용하기 전에 명시적으로 작성되야 한다.

1.6.2.7.2 반환 값

pdPASS : xSemaphoreGiveRecursive()에 대한 호출이 성공했다.

pdFAIL : xSemaphoreGiveRecursive()에 대한 호출은 호출하는 task 가 뮤텍스의 홀더가 아니기 때문에 실패했다.

1.6.2.7.3 기타

```
// 재귀 뮤텍스를 만드는 task 다.
void vATask( void * pvParameters ) {
    // 재귀 뮤텍스는 xSemaphoreCreateRecursiveMutex()에 대한 호출을 사용하여 명시적으로 생성되기 전에 사용할 수 없다.
    xMutex = xSemaphoreCreateRecursiveMutex();

    // 나머지 작업 코드는 여기에 위치한다.
    for( ;; ) {}
}

// 뮤텍스를 사용하는 함수다. (task 에 의해 호출된다.)
void vAFunction( void ) {
    // 다른 작업을 한다.
    if( xMutex != NULL ) {
      // 뮤텍스를 얻을수 있는지 확인한다. 뮤텍스가 사용 가능하지 않으면 10 tick 을 기다리고 해제한다.
      if( xSemaphoreTakeRecursive( xMutex, 10 ) == pdTRUE ) {
```

```
        // 뮤텍스를 성공적으로 획득했다.

        /* 코드의 특성상 xSemaphoreTakeRecursive()에 대한 추가 호출이 동일한 뮤텍스에서 수행된다.
        실제 코드에서 이것들은 단순하게 순차적으로 호출하지는 않을 것이다. 그것은 아무런 사용 용도가 없기 때문이다.
        대신, 예를 들어 TCP / IP 스택의 복잡한 구조 안에서 사용될 가능성은 있다. */
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

        /* 뮤텍스를 3 번 획득하고, 3 번 다시 반환할 때까지 다른 task 에서 사용할 수 없다.
        해당 예제는 설명을 위해 순차적으로 사용한 것이다. */
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );

        // 이제 뮤텍스를 다른 task 가 가져올 수 있다.
      } else {
        // mutex 가 성공적으로 획득하지 못했다.
      }
    }
}
```

1.6.2.8 xSemaphoreTakeRecursive : 재귀 뮤텍스 획득

```
#include "FreeRTOS.h"
#include "semphr.h"

BaseType_t xSemaphoreTakeRecursive ( SemaphoreHandle_t xMutex, TickType_t xTicksToWait );
```

xSemaphoreCreateRecursiveMutex()를 사용하여 이전에 생성된 재귀 뮤텍스 유형의 세마포어를 획득한다. 재귀 뮤텍스는 xSemaphoreTakeRecursive()를 사용하여 가져오고, xSemaphoreGiveRecursive()를 사용하여 준다. xSemaphoreTake() 및 xSemaphoreGive()함수는 재귀 뮤텍스와 함께 사용하면 안된다.

xSemaphoreTakeRecursive()호출은 중첩될 수 있다. 따라서 재귀 뮤텍스가 task 에 의해 성공적으로 획득하면 동일한 task 로 만들어진 xSemaphoreTakeRecursive()에 대한 추가 호출도 성공적일 것이다. 이전에 xSemaphoreTakeRecursive()에서 했던 것처럼 xSemaphoreGiveRecursive()에 동일한 수의 호출이 이루어 져야 뮤텍스가 다른 task 에 사용 가능해진다.

예를 들어 task 가 재귀로으로 동일한 뮤텍스를 5 번 가져올 경우, 뮤텍스를 성공적으로 획득한 task 가 뮤텍스를 정확히 5 번 돌려줄 때까지는 다른 task 에서 뮤텍스를 사용할 수 없다.

xSemaphoreGiveRecursive()는 실행중인 task 에서만 호출되어야 하므로, 스케줄러가 초기화 상태(스케줄러가 시작되기 전)에 있는동안 호출하면 안된다. 또한, 크리티컬 섹션 내에서나 스케줄러가 일시 중지된 동안 호출되어서도 안된다.

1.6.2.8.1 매개 변수

xMutex : 세마포어를 획득한다. 세마포어는 SemaphoreHandle_t 유형의 변수에 의해 참조되며 사용되기 전에 명시적으로 작성해야 한다.

xTicksToWait : 세마포어를 즉시 사용할 수 없는 경우, 세마포어를 사용할 수 있게 될 때까지 대기하는 최대 시간이다. xTicksToWait 을 0 으로 하면 세마포어를 사용할 수 없는 경우 xSemaphoreTakeRecursive()가 즉시 반환된다. 차단 시간은 tick 시간으로 지정되므로 tick 시간의 절대 시간은 tick 주파수에 따라 달라진다.

pdMS_TO_TICKS()매크로는 밀리 초 단위로 지정된 시간을 tick 으로 변환하는데 사용된다. xTicksToWait 을 portMAX_DELAY 로 설정하고, FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우 task 가 시간초과 없이 무기한 대기하게 된다

1.6.2.8.2 반환 값

pdPASS : xSemaphoreTakeRecursive()에 대한 호출이 세마포어를 획득하는데 성공한 경우에 반환한다. 차단 시간이 지정되어 있으면(xTicksToWait 이 0 이 아닌 경우), 호출한 task 가 즉시 사용할 수 없을 때 세마포어를 기다리기 위해 차단 상태로 설정되지만, 차단 시간이 만료되기 전에 세마포어가 사용 가능해질 경우에도 반환한다.

pdFAIL : xSemaphoreTakeRecursive()에 대한 호출이 세마포어를 성공적으로 획득하지 못한 경우 반환한다. 차단 시간이 지정되어 있으면(xTicksToWait 이 0 이 아닐 경우), 호출한 task 가 세마포어가 사용 가능하게 될 때까지 대기하기 위해 차단 상태로 설정되지만, 세마포어를 획득하기 전에 시간이 만료되면 반환한다.

1.6.2.8.3 기타

```
// 재귀 뮤텍스를 만드는 task 다.
void vATask( void * pvParameters ) {
    // 재귀 뮤텍스는 xSemaphoreCreateRecursiveMutex()에 대한 호출을 사용하여 명시적으로 생성되기 전에 사용할 수 없다.
    xMutex = xSemaphoreCreateRecursiveMutex();

    // 나머지 작업 코드는 여기에 위치한다.
    for( ;; ) {}
}

// 뮤텍스를 사용하는 함수다. (task 에 의해 호출된다.)
void vAFunction( void ) {
    // 다른 일을 한다.
    if( xMutex != NULL ) {
      // 뮤텍스를 얻을수 있는지 확인한다. 뮤텍스가 사용할 수 없으면 10 tick 을 기다린 후 해제된다.
      if( xSemaphoreTakeRecursive( xMutex, 10 ) == pdTRUE ) {
        // 뮤텍스를 성공적으로 획득했다.

        /* 코드의 특성상 xSemaphoreTakeRecursive()에 대한 추가 호출이 동일한 뮤텍스에서 수행된다.
        실제 코드에서 이것들은 단순하게 순차적으로 호출하지는 않을 것이다. 그것은 아무런 사용 용도가 없기 때문이다.
        대신, 예를 들어 TCP / IP 스택의 복잡한 구조 안에서 사용될 가능성은 있다. */
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

        /* 뮤텍스를 3 번 획득하고, 3 번 다시 반환할 때까지 다른 task 에서 사용할 수 없다.
        해당 예제는 설명을 위해 순차적으로 사용한 것이다. */
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );

        // 이제 뮤텍스를 다른 task 가 가져올 수 있다.
      } else {
        // mutex 가 성공적으로 획득하지 못했다.
      }
    }
}
```