

# FreeRTOS 프로그래밍

copyright © 2015 guileschool.com™

---

## 저작권에 관한 경고

- 이 책의 저작권은 인용 등 제3자가 집필한 부분을 제외하고 집필자인 홍영기에 속하며 그 내용은 한국 저작권법에 의해 보호를 받습니다. 이 책 내용의 일부 또는 전부를 복제(홈페이지나 인쇄 매체에 전재하는 것, 전자 파일로 복사하는 것도 포함함)할 때는, 한국 저작권법 제27조에 규정한 사적 사용을 제외하고, 영리목적 여부와 상관없이 한국 저작권법 제42조에 따라 저작권자인 홍영기의 허락이 필요하며 복제시에는 한국 저작권법 제12조와 제34조에 따라 저작자 이름과 출처를 명기해야 합니다.
- 이와 같은 절차를 밟지 않고 이 책 내용을 무단으로 사용하면 한국 저작권에 의해 처벌 대상이 될 수 있으며 이 책의 저작자인 홍영기도 저작권 보호를 위해 법적 조치를 취할 수 있으니 충분한 유의를 바랍니다.

copyright © 2015 guileschool.com™

---

## 교 육 목 표

- FreeRTOS 기반의 임베디드 시스템 프로그래밍 방법을 이해하여 실무에서 바로 적용 가능하도록 한다.
  - ◆ 실시간 운영체제의 특징 이해
  - ◆ 태스크 관리 방법의 이해
  - ◆ 공유 자원 보호 기법의 이해
  - ◆ 태스크 간 통신 기법의 이해
  - ◆ 포팅 기법의 이해

copyright © 2015 guileschool.com™

## 교 육 기 자 재

- 교재
  - ◆ FreeRTOS프로그래밍 과정, 강사 저작
- 실습용 PC
  - ◆ Pentium 급 이상의 IBM 호환 기종
  - ◆ CD-ROM 드라이브
  - ◆ 운영체제 : Windows 10
  - ◆ Adobe Reader 10.0 이상
- ARM compiler 환경
  - ◆ STmicro STM32CubeIDE
- 보드 디버깅
  - ◆ STM32CubeIDE
  - ◆ PRINTF 디버깅
- 실습용 보드
  - ◆ Nucleo64 보드 (Cortex-M3 기반 STM32F103RB 사용)
  - ◆ 원산 : STmicro (주)

copyright © 2015 guileschool.com™

# FreeRTOS 프로그래밍

## 목 차

0. 과정 소개 .....	0-1
1. 실시간 시스템 개론 .....	1-1
2. 실습 : 임베디드 시스템 개발 .....	2'-1
3. 태스크(TASK) 운용 .....	3-1
실습 : 태스크(TASK) 운용 .....	3'-1
4. 태스크간 통신(IPC) .....	4-1
실습 : 태스크간 통신(IPC) .....	4'-1
5. FreeRTOS 포팅 .....	5-1
6. 커널 API 레퍼런스 .....	6-1

# 실시간 시스템 개론

copyright © 2015 guileschool.com™



## 교육 목표

- 멀티 태스킹의 의미에 대하여 이해한다.
- 선점형(**Preemptive**) 스케줄링의 특징에 대하여 이해한다.
- 인터럽트 응답시간의 의미에 대하여 이해한다.
- 커널의 시계 **TICK** 이 갖는 의미를 이해한다.
- RTOS** 의 장점과 그 활용성이 갖는 의미를 이해한다.
- FreeRTOS** 의 특징을 이해한다.

copyright © 2015 guileschool.com™



## 목 차

---



멀티 태스크(**Multi Task**)

스케줄링(**Scheduling**)

인터럽트

커널의 시계(**TICK**)

**RTOS** 의 특징

**FreeRTOS** 소개

---

copyright © 2015 guileschool.com™



## 멀티태스킹이란

---



# YouTube



## 멀티태스킹이란

```
int c; //key 변수
main()
{
    while(1)
    {
        c=getButtonkey();      getButtonkey() 함수는
        . . .                   응용 프로그램에서
    }                         호출(call) 하는 방법으로만 실행
}                           할 수 있다.
                            하지만, 멀티태스킹 구조에서는
                            . . .
```

getButtonkey()  
{ 버튼 키처리 ...  
 return(HW\_KEY);  
}

5

copyright © 2015 guileschool.com™



## 멀티태스킹이란

```
int c; //key 변수
main()
{
    int tid;

    taskCreate("task_Buttonkey", . . .);
    . . .
}

task_Buttonkey()
{ while(1)
{ 버튼키처리. . .
    c=HW_KEY;
}
}
```

멀티태스킹 구조에서는  
task\_Buttonkey 의  
동시 실행이 가능하다

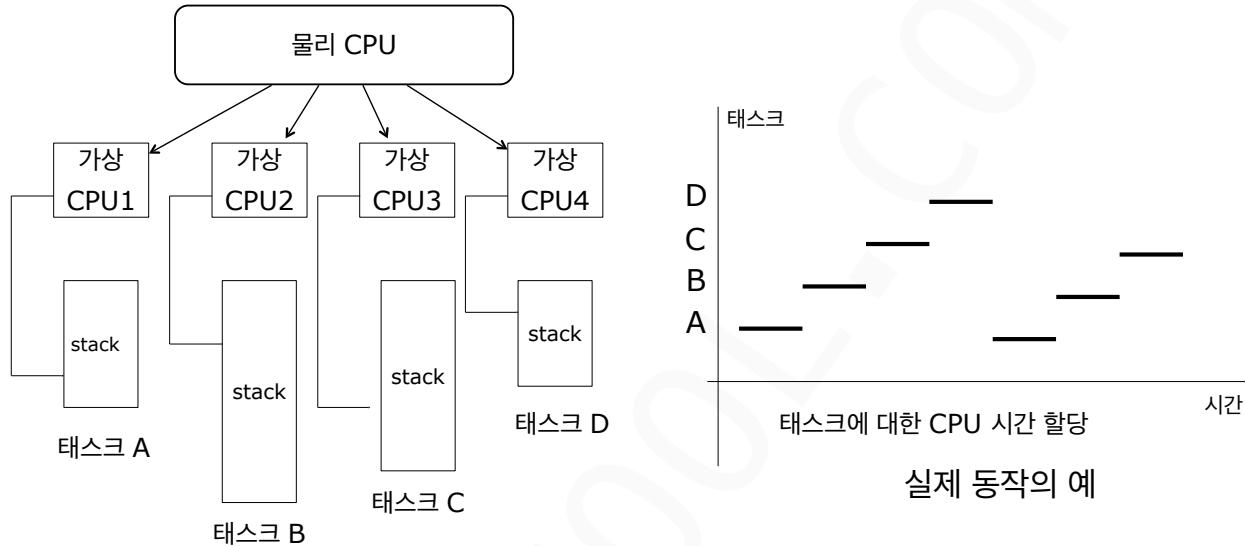
6

copyright © 2015 guileschool.com™



## 태스크(TASK)

- 실제로 물리적인 **CPU**는 하나이다
- 태스크는 가상의 **CPU** 을 각자 가지고 있는 것 처럼 동작 한다



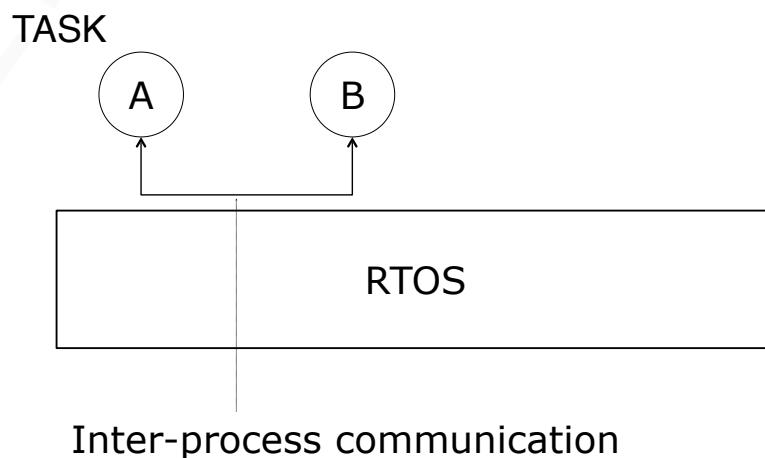
7

copyright © 2015 guileschool.com™



## 태스크간의 통신(IPC)

- 태스크 **A** 와 태스크 **B** 는 커널(**kernel**) 서비스인 **Inter Process Communication** 방법을 이용하여 통신한다



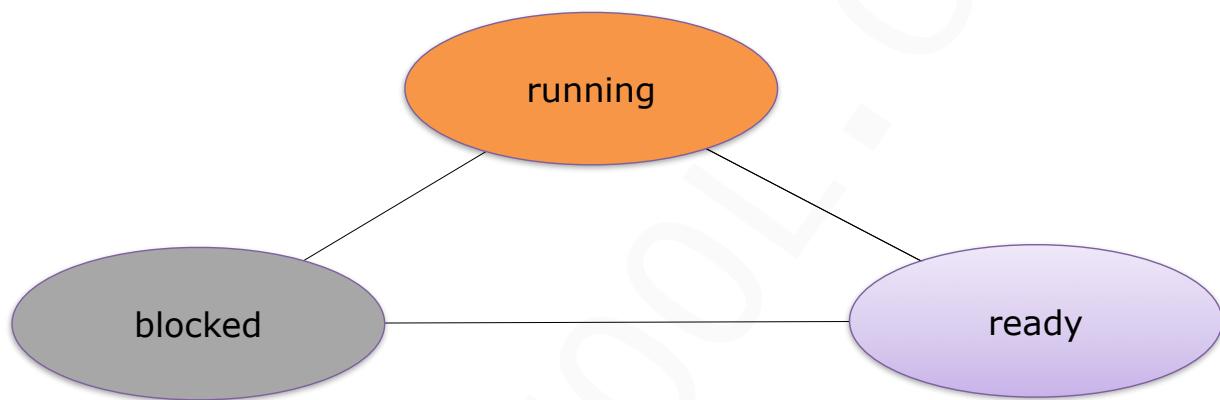
8

copyright © 2015 guileschool.com™



## 태스크 상태도(TASK STATE DIAGRAM) : Task's life cycle

- 멀티 태스킹 환경에서 커널은 실행 중에 상태를 계속 변경
- 태스크는 상시 다음과 같은 상태 중의 하나에 위치:
  - ◆ Running
  - ◆ Ready
  - ◆ Blocked (waiting)



9

copyright © 2015 guileschool.com™



## 목 차

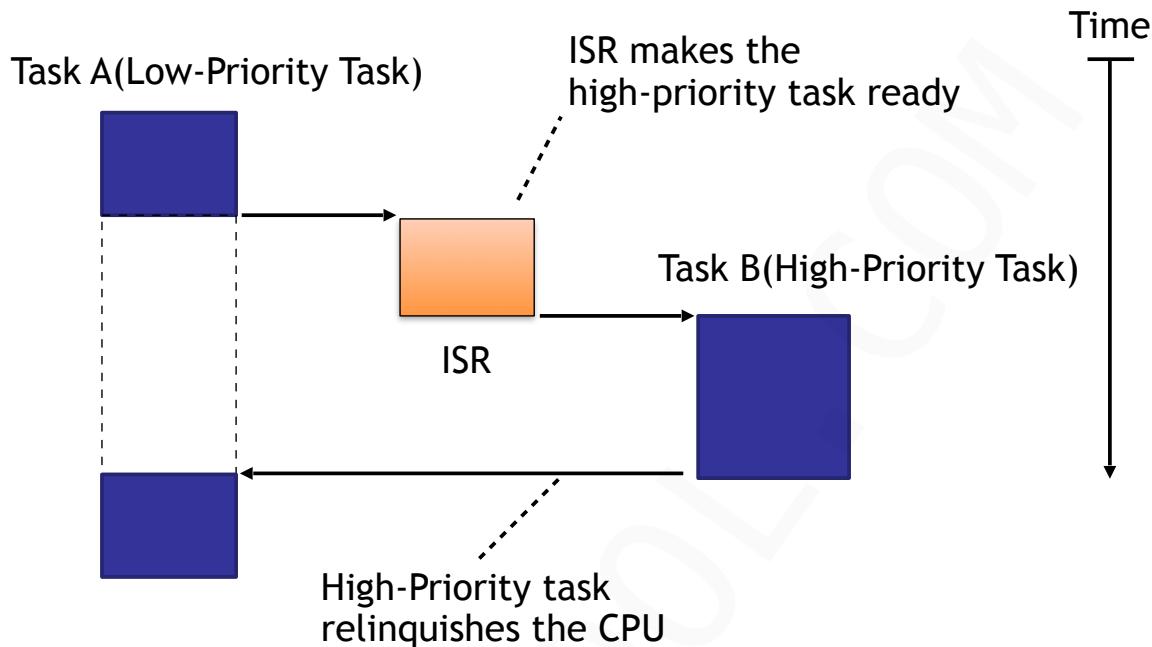


멀티 태스크(**Multi Task**)  
**스케줄링(Scheduling)**  
인터럽트  
커널의 시계(**TICK**)  
**RTOS** 의 특징  
**FreeRTOS** 소개

copyright © 2015 guileschool.com™



## 선점형 커널(Preemptive Kernel)



11

copyright © 2015 guileschool.com™



## 선점형 커널(Preemptive Kernel)

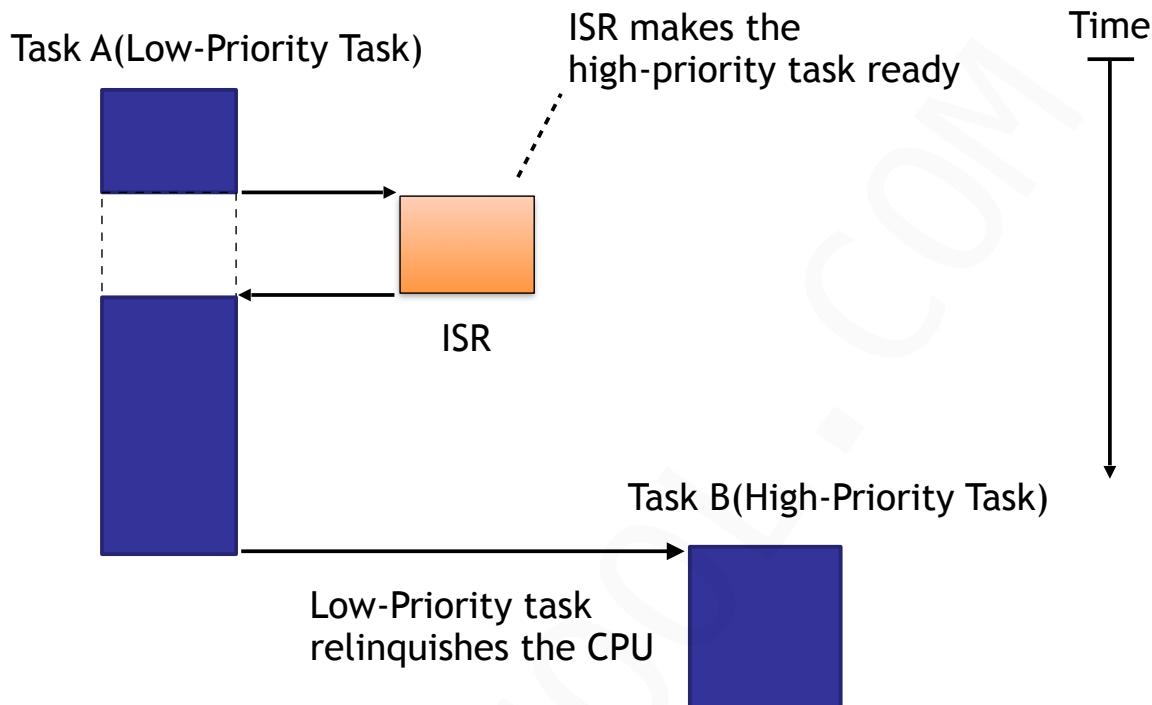
- 선점형 커널은 어떤 한 **task**가 수행하고 있는 도중에도 **kernel**이 그 **task**의 수행을 중지 시키고 다른 **task** (중지되는 **task** 보다 **priority**가 높은)를 수행시킬 수 있는 능력을 소유
- 가장 높은 우선순위의 **task**가 **CPU**를 점유하여 수행될 수 있음
- 시스템 응답성이 중요한 경우 사용됨
- 실 시간 운영체제에서 사용하는 구조임
  - ◆ 예) Windows 95/98/NT, UNIX, RTOS(VxWorks, uC/OS-II . . .)

12

copyright © 2015 guileschool.com™



## 비선점형 커널(Non-preemptive Kernel)



13

copyright © 2015 guileschool.com™



## 비선점형 커널(Non-preemptive Kernel)

- 비선점형 커널은 어떤 한 **task**가 수행하고 있을 때 **kernel**이 그 **task**의 수행을 강제로 중지시키고 다른 **task**를 수행시킬 수 있는 능력이 없음
- 실 시간 시스템에서는 사용될 수 없는 구조
  - 예) Windows 3.1

14

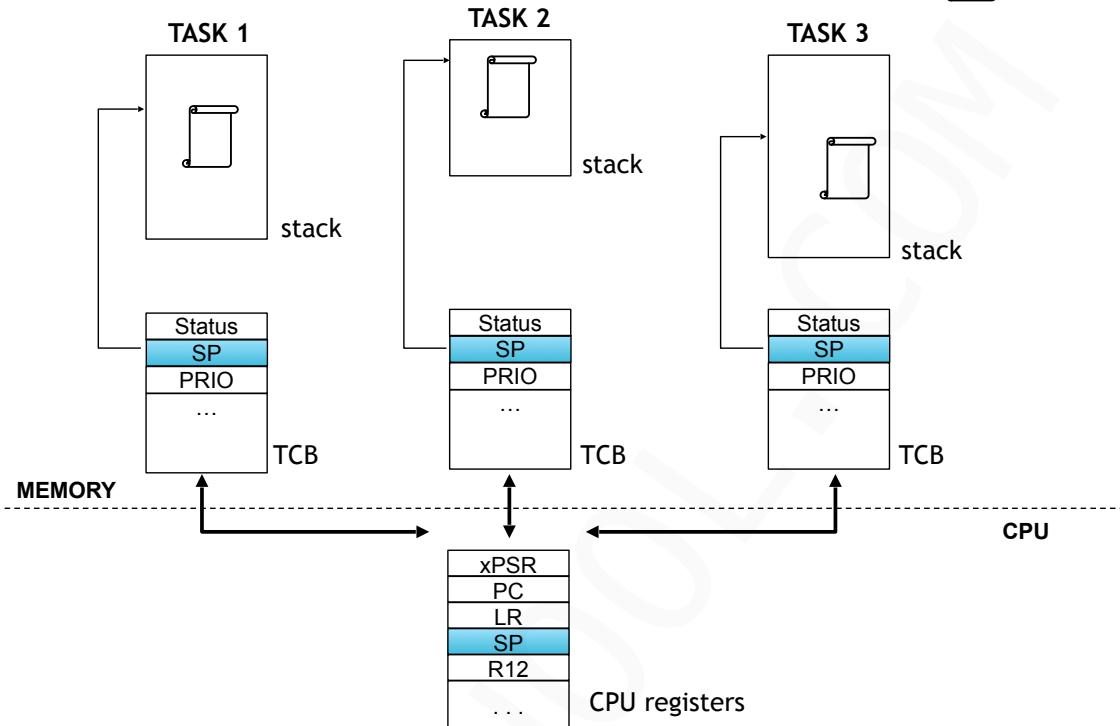
copyright © 2015 guileschool.com™



## TCB(Task Control Block)

- 태스크마다 각각 가지고 있는 멀티태스킹 자료구조

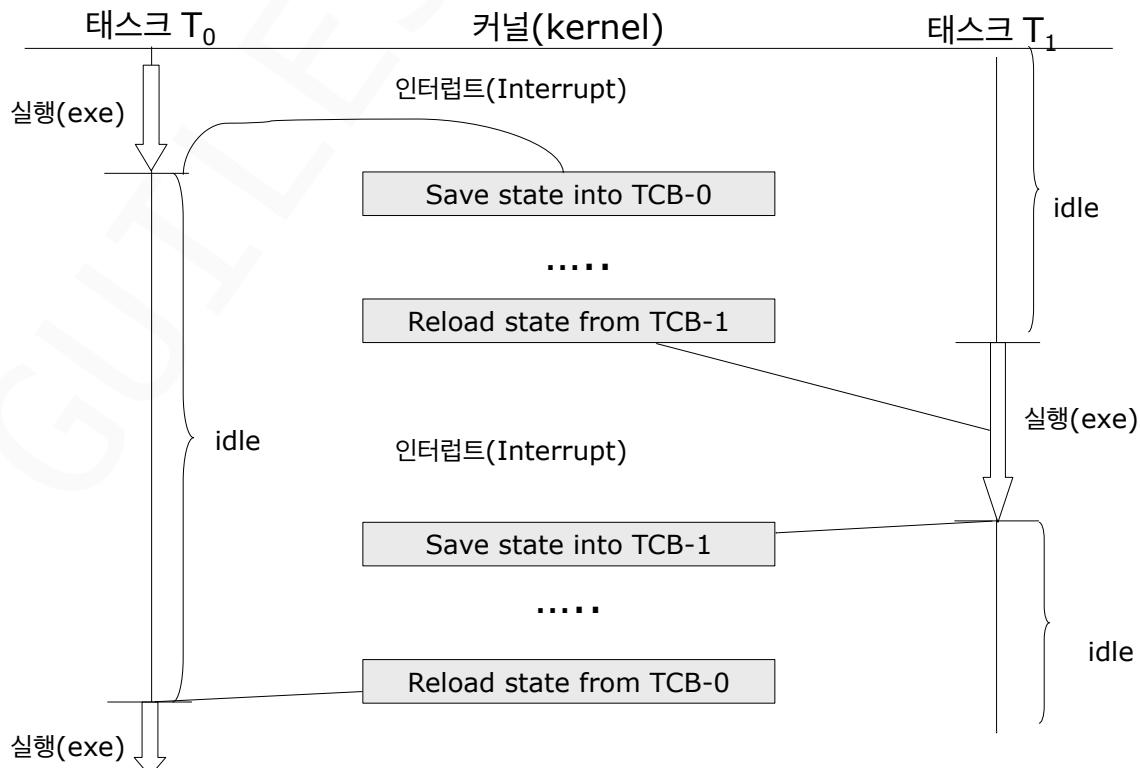
CONTEXT



copyright © 2015 guileschool.com™



## 문맥 전환(Context Switch)

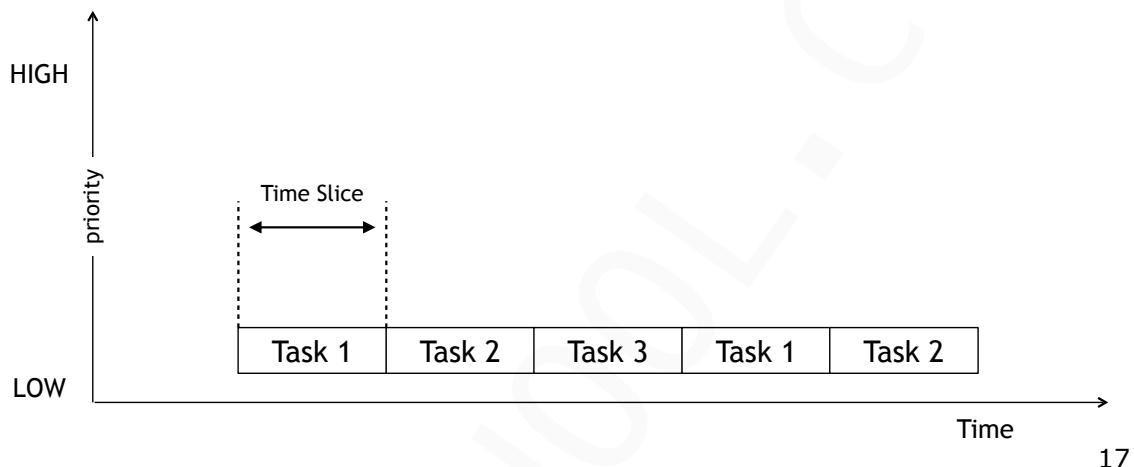


copyright © 2015 guileschool.com™



## Round-Robin 스케줄링 (1)

- 모든 태스크들이 시간 자원(**CPU**)을 공평히 나누어 사용하는 개념
- 각 태스크들이 점유하는 시간을 '타임퀀텀(**quantum**)' 이라고 한다
- 타임 퀀텀의 크기는 특별히 정해지지 않으나 대략 **1ms ~ 20ms** 을 많이 사용 함 <-- 타임퀀텀이 너무 짧으면 문맥 전환을 위한 시간 자원의 낭비가 심함
- 하지만 어플리케이션이 단순한 비 선점형 스케줄링 만을 필요로 할 경우 최적



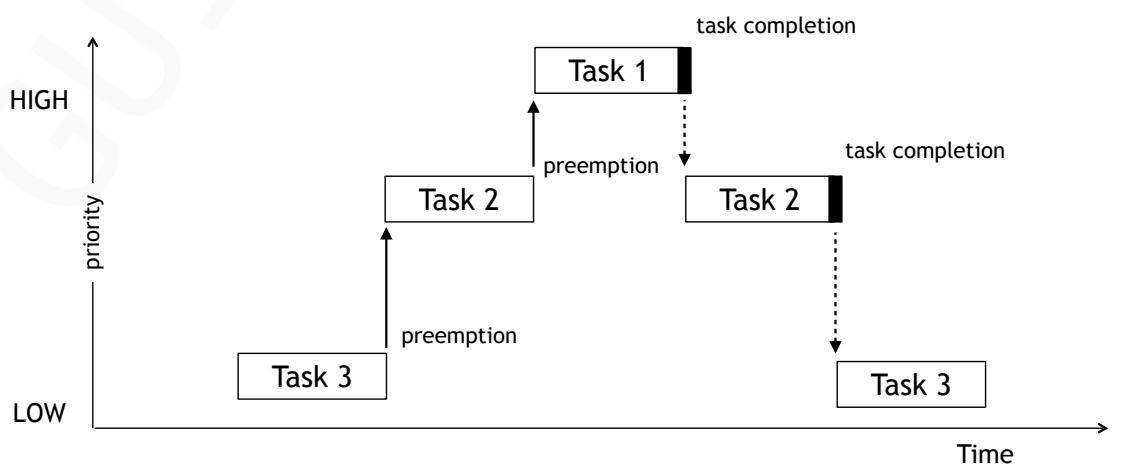
17

copyright © 2015 guileschool.com™



## Priority 스케줄링 (2)

- 태스크를 중요도에 의해 가중치를 두어 우선적으로 실행 할 수 있도록 하겠다는 개념
- 실시간 운영체제(**RTOS**)에서 필수적으로 지원하는 스케줄링 방법임
- 선점형 스케줄링의 특성을 부여받음



18

copyright © 2015 guileschool.com™



# 목 차

## 멀티 태스크(**Multi Task**) 스케줄링(**Scheduling**)



인터럽트

커널의 시계(**TICK**)

**RTOS** 의 특징

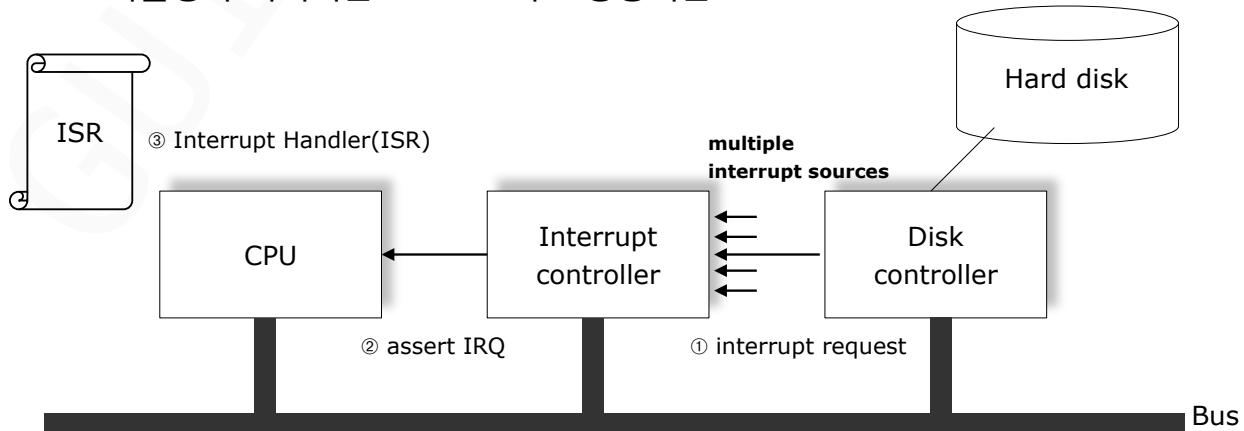
**FreeRTOS** 소개

copyright © 2015 guileschool.com™



## 인터럽트(**Interrupt**)

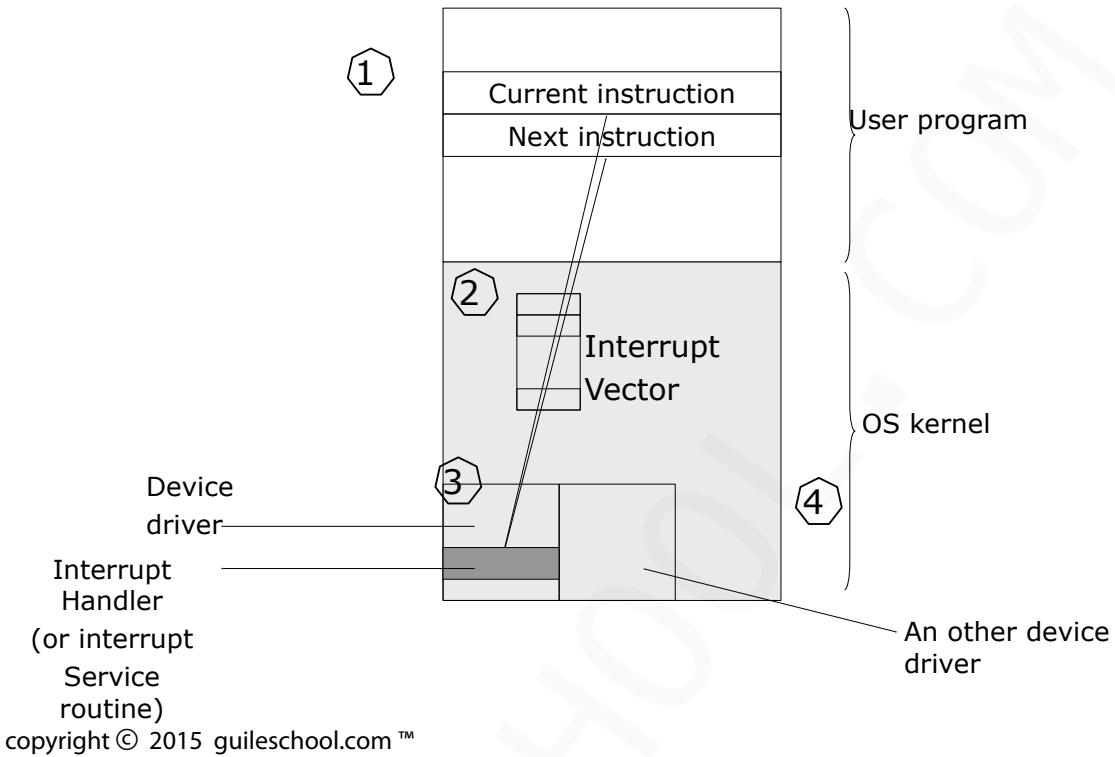
- 비동기적 이벤트의 발생을 처리하기 위한 메커니즘
- 인터럽트 발생시 문맥을 저장하고 **ISR**로 점프
- 활성, 비활성화 가능
  - ◆ 비활성화 시간은 가능한 짧게 해야함
- 지연시간
  - ◆ 비활성화 최대시간 + ISR 최초 명령시간



copyright © 2015 guileschool.com™

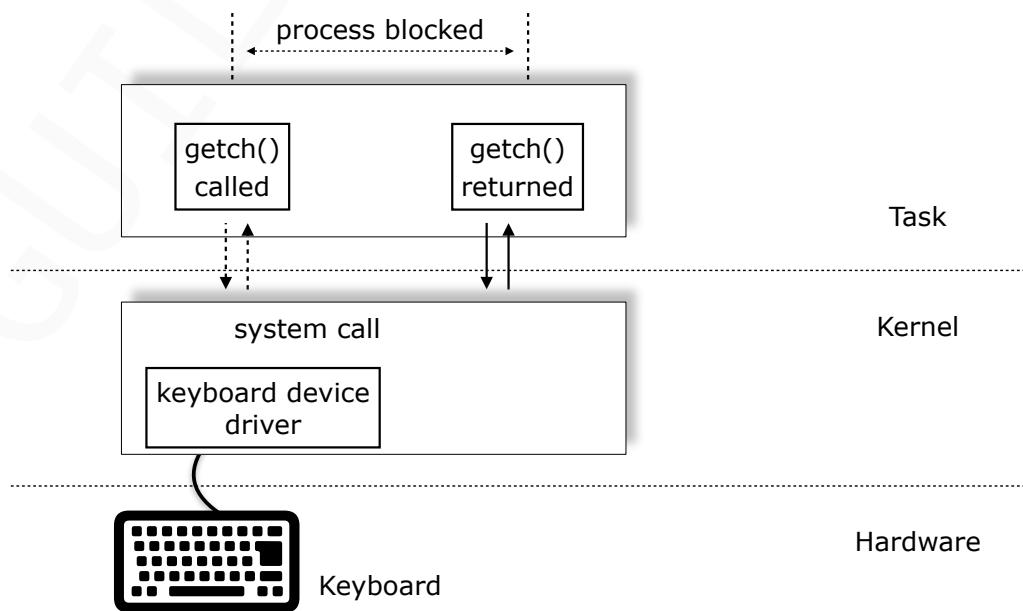


## 인터럽트(Interrupt) 소프트웨어 처리



## Blocking I/O

- 블록킹 I/O 동작은 태스크가 시스템콜을 호출 했으나 점유하고자 하는 데이터가 즉시 가용하지 않을 경우 그 동작이 완료 될 때 까지 '**suspending(blocked)**' 상태로 유지되는 것을 말함





## Blocking I/O

```
#include <conio.h>
#include <stdio.h>

void main()
{
    int c;
    clrscr();
    printf("Press any key\n");
    while(1){
        c = getch();
        if (c)
            printf(" A key is pressed from keyboard ");
        else
            printf("An error occurred ");
        .... 코드 중략
    }
}
```

Blocking I/O 함수의 예

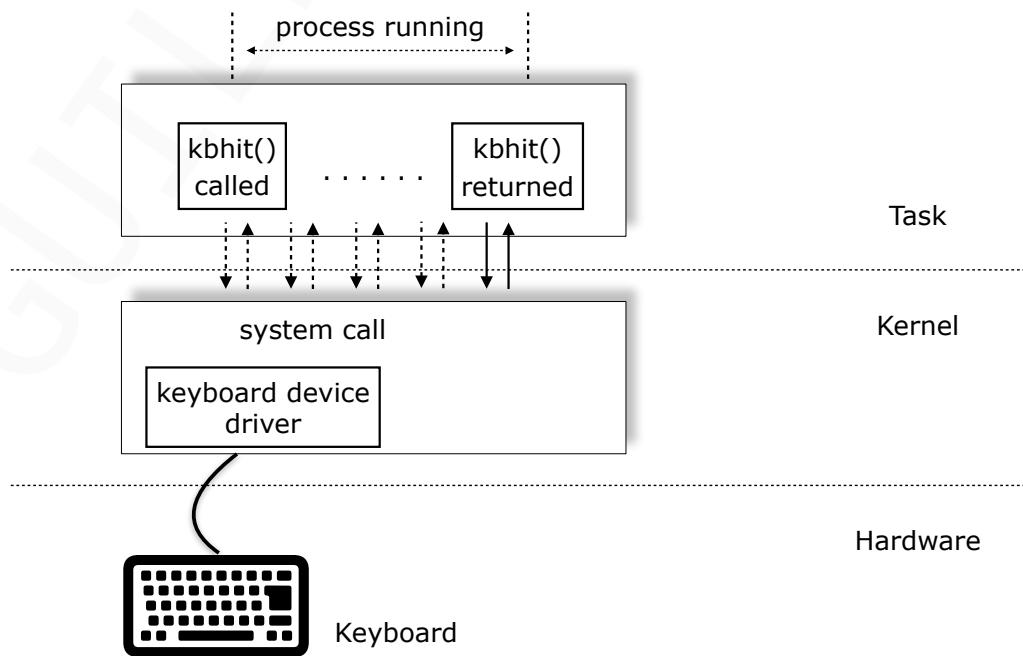
23

copyright © 2015 guileschool.com™



## Non-Blocking I/O

- 비-블록킹 I/O 동작은 태스크가 시스템콜을 호출 했으나 점유하고자 하는 데이타가 즉시 가용하지 않을 경우 즉각 리턴을 하는 것을 특징으로 함



24

copyright © 2015 guileschool.com™



# Non-Blocking I/O

```
#include <conio.h>
#include <stdio.h>

int main ()
{
    char c;
    while (1)
    {
        if (kbhit ())
        {
            int c;
            c = getch ();
            if (c == 'q' || c == 'Q' || c == EOF)
                break;
        }
        ..... 코드 중략
    }
    return 0;
}
```

Non-Blocking I/O 함수의  
예

25

copyright © 2015 guileschool.com™



## 목 차

멀티 태스크(**Multi Task**)  
스케줄링(**Scheduling**)  
상호배제(**Mutual Exclusion**)

태스크 간 통신(**IPC**)

인터럽트

커널의 시계(**TICK**)

**RTOS** 의 특징

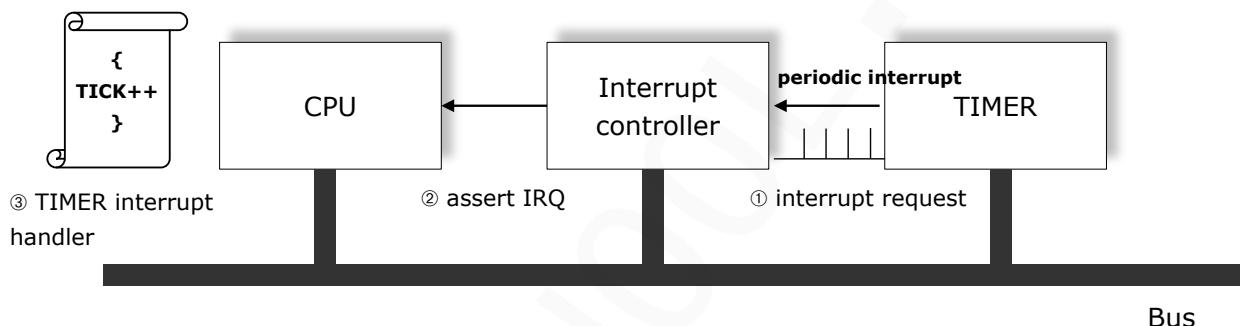
**FreeRTOS** 소개

copyright © 2015 guileschool.com™



## 커널의 시계(TICK)

- 정기적(**PERIODIC**)으로 일어나는 인터럽트
- 태스크 지연(**delayed**), 타임아웃(**timeout**)을 제공
- 태스크 지연의 분해능(**clock resolution**)은 클럭 틱(**TICK**) 하나
- 클럭 틱 단위 정확도로 지연 가능한 것은 아님
  - ◆ 선점형 커널에서 가능



copyright © 2015 guileschool.com™



## 목 차

### 멀티 태스크(**Multi Task**) 스케줄링(**Scheduling**)

인터럽트

커널의 시계(**TICK**)

**RTOS** 의 특징

**FreeRTOS** 소개

copyright © 2015 guileschool.com™



## RTOS의 특징

- Hard Realtime**
- Scalability**
- Preemptive**
- Multitasking**
- Deterministic**
- Portability**
- Robustness**



copyright © 2015 guileschool.com™



## RTOS 종류

 TEXAS INSTRUMENTS

  
μC/OS-II®  
*The Real-Time Kernel*



 Windows Embedded



  
Green Hills  
SOFTWARE











copyright © 2015 guileschool.com™



## 실시간 시스템의 개념

- 리얼타임 시스템 – 정해진 시간 내에 임무를 수행하는 시스템
- 리얼타임 시스템의 분류
  - ◆ 소프트 리얼타임 시스템 (**Soft RealTime System**)  
: 가능한 한 빠르게 임무를 수행하지만 반드시 정해진 시간 내에 수행할 필요는 없다. (timeout 이어도 계속 수행)
  - ◆ 하드 리얼타임 시스템 (**Hard RealTime System**)  
: 어떤 사건이 발생했을 때 정확히 동작하는 것은 물론이고 반드시 정해진 시간 내에 그 임무를 마쳐야 한다. (timeout 이면 failure)

31

copyright © 2015 guileschool.com™



## 목 차

멀티 태스크(**Multi Task**)  
스케줄링(**Scheduling**)  
인터럽트  
커널의 시계(**TICK**)  
**RTOS** 의 특징



**FreeRTOS 소개**

copyright © 2015 guileschool.com™



## FreeRTOS 시작하기

- FreeRTOS는 **2003**년 Richard Barry 가 만든 실시간 운영체제
- 35개 이상의 마이크로 컨트롤러에 이식됨
- 아마존에서 AWS IoT 엣지 서비스 확장을 위해 **2017**년 11월 인수
- **MIT License**
- 용도 : 각종 장비 개발이 가능
  
- 공식 사이트 [www.freertos.org](http://www.freertos.org) (소스 다운로드 가능)
  - ◆ MIT 라이센스에 따라 소스코드 공개 등의 조건없이 상업적인 목적으로 사용 가능

33

copyright © 2015 guileschool.com™



## FreeRTOS 을 이용한 상용 RTOS

### □ SafeRTOS

**SafeRTOS**는 무료 RTOS로 유명한 Amazon FreeRTOS의 **Functional model**을 기반으로 개발된 RTOS로, FreeRTOS로는 만족할 수 없는 안전-결정적 요구사항을 만족하기 위해서 등장하였다. 의료기기, 철도, 비행기 등 안전-결정적(**safety-critical**) 시스템에 바로 적용가능한 안정적이고 효율적인 RTOS

### □ OpenRTOS

**OpenRTOS**는 간단히 말해서 FreeRTOS의 상용 버전이다. 소스코드 및 기능이 FreeRTOS와 **100%** 동일하다. 다만 비용을 지불하고 정식 상용 라이선스를 구매해야 하므로 정식으로 기술지원을 받을 수 있다는 점이 FreeRTOS와 가장 큰 차이점이다. 또한 IP 침해 주장에 대한 보호를 제공받을 수 있다

위 두 제품 모두 WITTENSTEIN high integrity systems 사에서 상용 제품으로서 공급

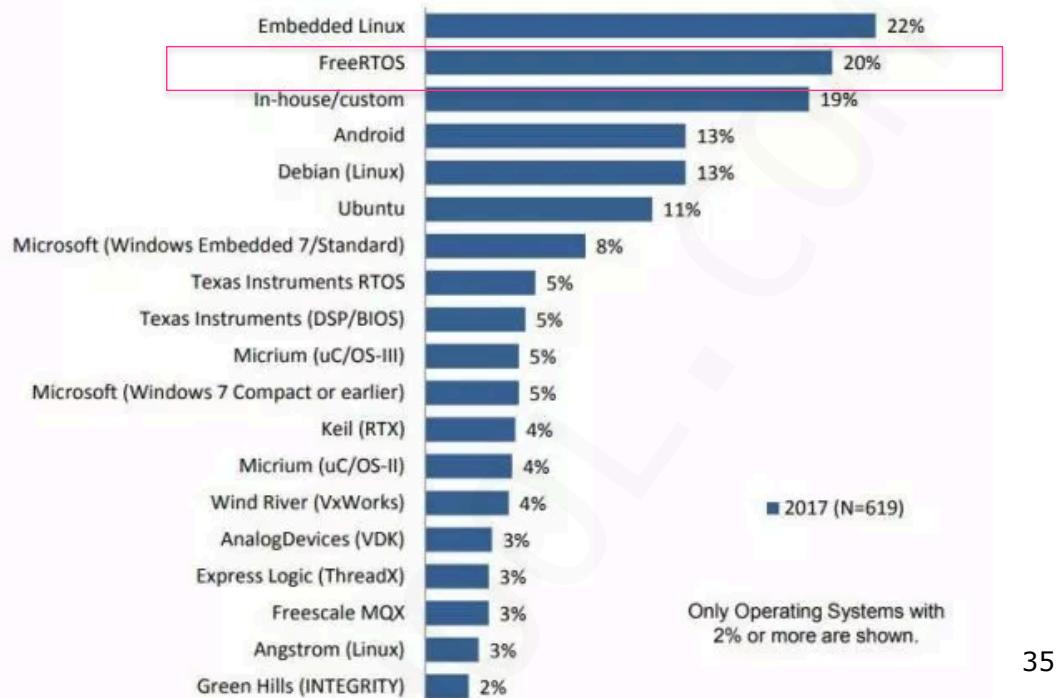
34

copyright © 2015 guileschool.com™



# FreeRTOS 시작하기

Please select ALL of the operating systems you are currently using.



35

copyright © 2015 guileschool.com™



## FreeRTOS의 특징(1)

- Portable**
  - ◆ 35개 이상의 마이크로컨트롤러에 이식. 커널 코드의 대부분이 이식 가능한 **ANSI C**를 기반 (일부 마이크로프로세서에 국한된 부분은 어셈블러로 코딩)
- ROMable**
  - ◆ 해당 C 컴파일러, 어셈블러, 링커, 로더가 필요, 설정이나 응용프로그램에 따라 메모리의 크기 를 자유로이 결정할 수 있다.
- 커널 바이너리 이미지는 **4K**에서 **9K** 바이트 영역
- Preemptive Scheduling**
  - ◆ 높은 우선 순위 작업이 먼저 수행된다.
- Round Robin Support**
- Real-Time**
  - ◆ 빠른 응답성을 갖는다.
- Multitasking**
  - ◆ 독자적 우선순위를 가지는 무제한으로 생성 가능한 태스크

36

copyright © 2015 guileschool.com™



## FreeRTOS의 특징(2)

---

- 임베디드 운영체제로써 대표적인 공개형 소스코드 기반 커널
- 신뢰성과 안정성을 가진다.
  - ◆ 작은 사이즈 – 많은 시스템에 적용가능
    - 작은 임베디드 시스템에 탑재 가능하며 임베디드 시스템 중에서도 강력한 네트워크가 필요 한 곳과 높은 성능 시스템에 사용하는 것이 적합
- 프로젝트에 따른 소스 코드의 절약이 가능
- 스택오버플로우체크, 처리시간 체크, 후크(HOOK), Queue, Semaphore, 5가지 Heap memory 등의 시스템 서비스 제공
- 저전력 애플리케이션을 위한 틱리스 (tickless) 모드 지원
- 효율적인 소프트웨어 타이머
- 인터럽트 관리
  - ◆ 태스크의 수행을 일시 중지하거나 재개가 가능하다.
  - ◆ 인터럽트 중첩

37

copyright © 2015 guileschool.com™

---



## FreeRTOS의 특징(3)

---

- 강력한 실행 추적 기능
- 스택 오버플로 감지 옵션
- 단점
  - ◆ 디바이스 드라이버 부재
  - ◆ 멀티코어 지원 안됨

38

copyright © 2015 guileschool.com™

---



## 연습 문제

---

- 선점형 커널이 비선점형 커널에 비해 가장 큰 장점은 무엇인가?
- 문맥 전환(**Context Switch**)의 동작 원리를 설명하라
- Priority** 와 **Round-Robin** 스케줄링 기법을 각각 설명 하라
- 우선순위역전(**Priority Inversion**)의 사례를 그림으로 설명 하라
- 세마포어(**Semaphore**)의 의의에 대해 설명하라
- 태스크 동기화(**Task Synchronization**)에 대해서 설명하라
- 하드 리얼 타임 이란 용어에 대해서 설명하라
- 논 블럭킹 **vs** 블럭킹 **I/O**의 장점에 대해서 설명하라
- FreeRTOS**에서 **IPC**로 사용되는 커널 서비스 세가지는
- 상호 배제 기법 **3가지**를 각각 설명하라

---

copyright © 2015 guileschool.com™

---



## 질의 응답

---

copyright © 2015 guileschool.com™

---

# 태스크(TASK) 운용

copyright © 2015 guileschool.com™



## 교육 목표

- 태스크을 활용하고 그 의미를 이해한다.
- 시간관리 서비스를 활용하고 그 의미를 이해한다.
- CRITICAL SECTION** 처리 기법들을 이해한다.
- 실시간 시스템 성능 최적화 기법에 대하여 이해한다.
- 문맥 전환의 동작 원리에 대하여 이해한다.
- IDLE**태스크와 통계 태스크를 활용하고 그 의미를 이해한다.
- 메모리 관리를 활용하고 그 의미를 이해한다

copyright © 2015 guileschool.com™



## 목 차



### 태스크(Task)

시간관리 서비스(**TIME**)

임계영역(**CRITICAL SECTION**)

문맥전환(**CONTEXT SWITCH**)

**IDLE** 태스크

인터럽트와 클럭 틱(**TICK**)

시스템 성능 최적화(**OPTIMIZATION**)

메모리 관리

copyright © 2015 guileschool.com™



## 태스크의 형태 (1)

```
void YourTask( void *pvParameters )
{
    for (;;) {
        /* USER CODE is here */
        Call one of FreeRTOS's services:
        vTaskDelay();
        vTaskSuspend();
        vTaskPrioritySet();
        vTaskResume();
        /* USER CODE is here */
    }
}
```

유형1. 무한루프  
함수

```
void YourTask( void *pvParameters )
{
    /* USER CODE is here */
    .....
    .....
    vTaskDelete (NULL);
}
```

유형2. 실행후 스스로를  
삭제하는 함수

copyright © 2015 guileschool.com™



## 태스크의 형태 (2)

---

- 무한루프 함수와 스스로를 삭제하는 형태의 함수가 가능
- 태스크는 절대 리턴하면 안되므로 항상 '**void**'로 리턴형을 사용
  - ◆ 예) **void userTask( void \*pvParameters )**

```
// TASK CREATE
xTaskCreate( Task1,
    "Task1",
    1024,
    NULL,
    TASK_1_PRIO,
    &xHandle1 );

.....
// 태스크 'START'
void Task1 ( void *pvParameters )
{ .....
```

5

copyright © 2015 guileschool.com™

---



---

## 실습 : 태스크 생성 및 운용(**01\_TASKMAN**)

copyright © 2015 guileschool.com™

---



## 태스크와 우선순위

- 우선순위 갯수는 사실상 무제한
  - 큰 숫자가 **높은** 우선순위를 나타냄
    - ◆ 낮은 우선순위 : **0, 1, 2, 3**
    - ◆ 높은 우선순위 : **100, 101, 102, ...**
  - 하지만, 우선 순위의 사용 범위를 다음처럼 제한할 수 있다
- #define configMAX\_PRIORITIES (100)**
- 동일한 우선순위도 사용 가능(ROUND ROBIN 지원)
  - 새로 생성된 태스크가 우선순위가 높다면 생성과 동시에 **CPU** 제어권을 할당받아 실행된다
  - FreeRTOS에서는 **vTaskStartScheduler()** 함수를 호출해서 멀티태스킹을 시작한다

7

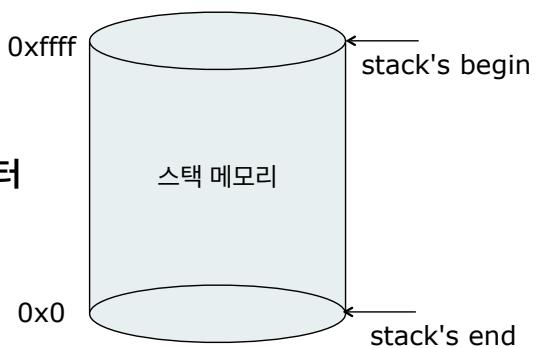
copyright © 2015 guileschool.com™



## Task 생성

□ **BaseType\_t xTaskCreate( TaskFunction\_t pxTaskCode, const char \* const pcName, const configSTACK\_DEPTH\_TYPE usStackDepth, void \* const pvParameters, UBaseType\_t uxPriority, TaskHandle\_t \* const pxCreatedTask )**

- ◆ **pxTaskCode:** 태스크 함수
- ◆ **\*pcName:** 태스크 함수의 이름
- ◆ **usStackDepth:** 스택 항목의 개수
- ◆ **\*pvParameters:** 태스크 전달 파라메터
- ◆ **uxPriority:** 태스크 우선순위
- ◆ **\*pxCreatedTask:** 태스크 핸들



8

copyright © 2015 guileschool.com™

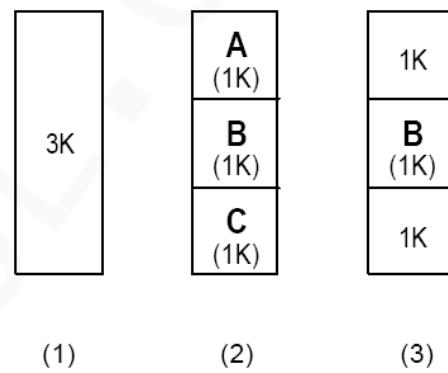


## Task 와 스택(STACK) 메모리 (1)

- 태스크 스택 메모리는 정적, 동적 할당(default) 사용 가능
- **#define configSUPPORT\_DYNAMIC\_ALLOCATION 1**
- 동적 메모리 사용은 메모리 단편화(**Memory Fragmentation**) 현상이 발생하지 않도록 응용프로그램이 일단 한번 생성한 태스크는 프로그램 종료까지 삭제 않고 유지 하는 경우에만 사용하는 것이 바람직
- 메모리 단편화 (**Memory Fragmentation**)란 ?

기억 장치의 빈 공간 또는 자료가 여러 개의

조각으로 나뉘는 현상을 말한다. 이는 요청한  
메모리의 크기가 가용한 메모리의 크기보다  
작은데도 불구하고 메모리 할당이 실패할 수  
있는 문제를 낳는다.



9

copyright © 2015 guileschool.com™



## Task 와 스택(STACK) 메모리 (2)

- 태스크 스택의 크기는 응용 프로그램마다 다르다
- 태스크 스택의 크기를 결정하는 요소들
  - ◆ 태스크에서 호출하는 함수들의 중복 호출 횟수
  - ◆ 해당 함수에서 사용되는 모든 지역변수의 갯수
- 동적 메모리를 이용한 태스크 스택의 생성

**#define configSUPPORT\_DYNAMIC\_ALLOCATION 1**

TaskHandle\_t xTaskCreateStatic(TaskFunction\_t pxTaskCode,...)  
함수를 이용하여 태스크 생성

- 정적 메모리를 이용한 태스크 스택의 생성과 그 실행

**#define configSUPPORT\_STATIC\_ALLOCATION 1**

BaseType\_t xTaskCreate(TaskFunction\_t pxTaskCode,...) 함수를 이용하여  
태스크 생성

10

copyright © 2015 guileschool.com™



---

## 실습 : 스택 오버플로우 검사 **(02\_STACKOVERFLOW)**

copyright © 2015 guileschool.com™

---



### 정적 메모리를 이용한 태스크

---

- **#define configSUPPORT\_STATIC\_ALLOCATION** 1 선언후 사용 가능
- 사용 예(**Example**)

```
/* For example, if each stack item is 32-bits, and this is set to 100,
then 400 bytes (100 * 32-bits) will be allocated. */
#define STACK_SIZE 200

/* Structure that will hold the TCB of the task being created. */
StaticTask_t xTaskBuffer;

/* Buffer that the task being created will use as its stack.
StackType_t xStack[ STACK_SIZE ];

/* Function that implements the task being created. */
void vTaskCode( void * pvParameters )
{
    for( ; ; )
    {
        /* Task code goes here. */
```

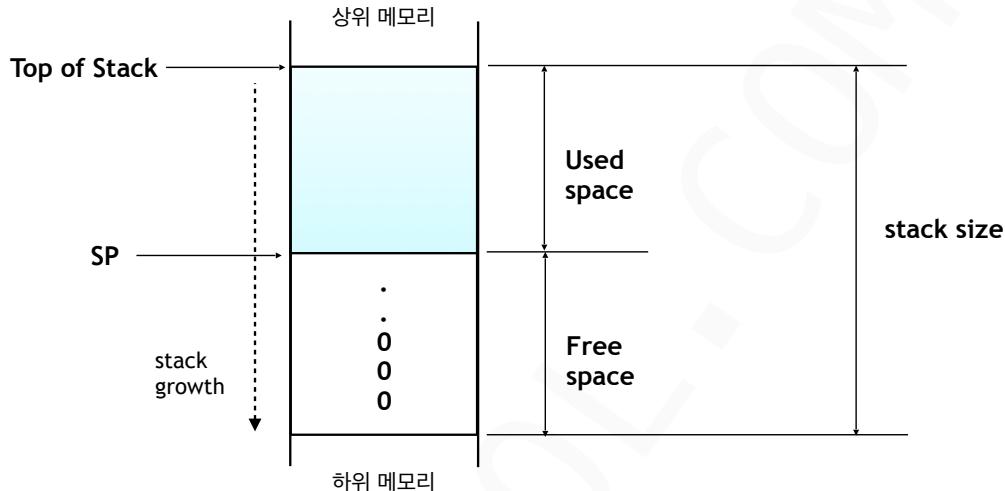
12

copyright © 2015 guileschool.com™

---



## 스택(STACK) 메모리 크기 최적화



13

copyright © 2015 guileschool.com™



## 런타임 스택 검사 방법 2가지(1/2)

- **#define configCHECK\_FOR\_STACK\_OVERFLOW 1**
- configCHECK\_FOR\_STACK\_OVERFLOW가 1로 설정 되어있고, 스택 포인터가 유효한 범위를 벗어난 스택 공간 내에 있는 것이 실시간으로 확인되면 사용자가 미리 정의한 hook 함수가 호출됨
- 속도는 빠르지만 문맥전환시에 스택 오버플로우를 검사하는 방식이므로, 이외의 시간에 발생할 수 있는 스택 오버플로우는 찾아내지 못할 수 있다
- 후크 함수의 예

```
void vApplicationStackOverflowHook( TaskHandle_t xTask, char *pcTaskName )
{
    configASSERT(( volatile void * ) NULL);
}
```

14

copyright © 2015 guileschool.com™



## 런타임 스택 검사 방법 2가지(2/2)

- **#define configCHECK\_FOR\_STACK\_OVERFLOW 2**
- 태스크가 생성되면 스택 공간을 알려진 0(zero) 패턴으로 덮어 채운다
- 이 방법은 유효한 스택내 패턴 정보 20바이트를 테스트한다. 20 바이트 중 하나라도 변경된 경우, 오버플로 후크 함수가 호출된다
- 스택 검사 방법1에 비해 속도는 다소 느리지만, 보다 정확하게 스택 오버플로우를 찾아낼 수 있는 장점이 있다
  
- **후크 함수의 예**

```
void vApplicationStackOverflowHook( TaskHandle_t xTask, char *pcTaskName )  
{  
    configASSERT(( volatile void * ) NULL);  
}
```

15

copyright © 2015 guileschool.com™



## Task 삭제

- 삭제된 태스크는 더이상 스케줄링 되지 않음
- **void vTaskDelete( TaskHandle\_t xTaskToDelete )**
- **xTaskToDelete** : 삭제할 태스크 핸들
- **IDLE** 태스크는 삭제 할 수 없음
- 자기 자신을 삭제하는 것도 가능. xTaskToDelete(핸들)에 **NULL** 을 전달

16

copyright © 2015 guileschool.com™



## TASK 우선 순위의 변경

---

- **vTaskPrioritySet()** 을 이용
- **void vTaskPrioritySet( TaskHandle\_t xTask, UBaseType\_t uxNewPriority )**
  - ◆ **xTask** : 변경코자 하는 태스크 핸들
  - ◆ **uxNewPriority** : 새로운 우선 순위
- 자기 자신(**NULL**) 혹은 다른 태스크의 우선 순위를 변경
- **IDLE** 태스크의 우선 순위 변경은 불가

17

copyright © 2015 guileschool.com™

---



## TASK 일시 중단

---

- 태스크의 실행을 일시 중단
- **void vTaskSuspend( TaskHandle\_t xTaskToSuspend )**
  - ◆ **xTaskToSuspend** : 중지시킬 태스크 핸들
- 태스크를 다시 동작시키기 위하여 **vTaskResume()** 을 이용

18

copyright © 2015 guileschool.com™

---



## 중단된 TASK 의 실행 재개

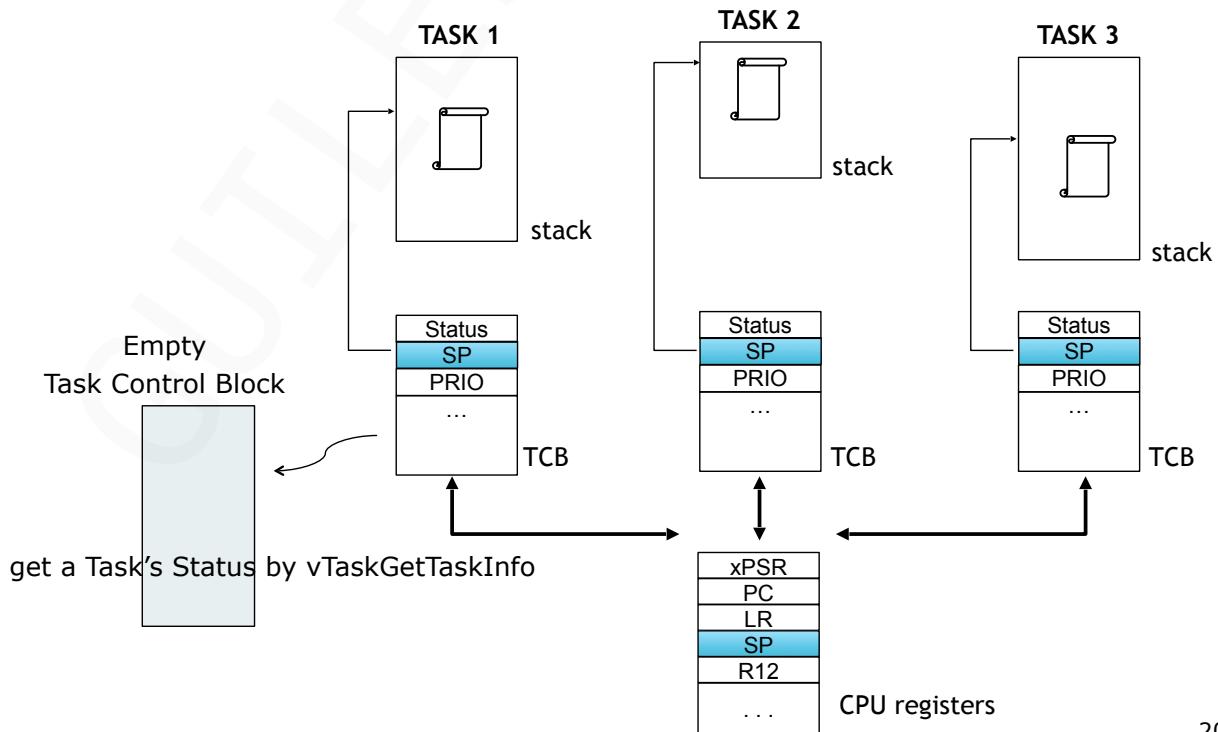
- 중단된 태스크를 준비 상태로 재개
- **void vTaskResume( TaskHandle\_t xTaskToResume )**
  - ◆ **xTaskToResume** : 준비 상태로 재개시킬 태스크 핸들
- 일시 중단된 태스크는 **vTaskResume ()**에 의해서만 준비 상태로 돌아 올 수 있다

19

copyright © 2015 guileschool.com™



## TASK 의 정보를 얻어 오기(1)



20

copyright © 2015 guileschool.com™



## TASK 의 정보를 얻어 오기(2)

- 지정한 태스크의 주요 정보를 얻어 온다
- **void vTaskGetTaskInfo( TaskHandle\_t xTask, TaskStatus\_t \*pxTaskStatus, BaseType\_t xGetFreeStackSpace, eTaskState eState )**
  - ◆ **xTask** : 정보를 얻어 올 태스크 핸들
  - ◆ **\*pxTaskStatus** : **TaskStatus\_t** 유형의 변수를 가리켜야 함
- 디버깅(**debugging**)시 활용 가능한 정보

```
void vAFunction( void )
{
    TaskHandle_t xHandle;
    TaskStatus_t xTaskDetails;

    vTaskGetTaskInfo( xHandle, /* The handle of the task being queried. */
                     &xTaskDetails, /* The TaskStatus_t structure to complete with
information on xHandle. */
                     pdTRUE, /* Include the stack high water mark value in the TaskStatus_t
structure. */
                     eInvalid /* Include the task state in the TaskStatus_t structure. */ );
}
```

21

copyright © 2015 guileschool.com™



## 목 차

### 태스크(**Task**)

- **시간관리 서비스(**TIME**)**
- 임계영역(**CRITICAL SECTION**)**
- 문맥전환(**CONTEXT SWITCH**)**
- IDLE** 태스크
- 인터럽트와 클럭 틱(**TICK**)
- 시스템 성능 최적화(**OPTIMIZATION**)**
- 메모리 관리

copyright © 2015 guileschool.com™



## FreeRTOS 변수(Variable) 이름 규칙

### □ 변수 프리픽스

type: 'c' for char

type: 's' for int16\_t (short)

type: 'l' for int32\_t (long)

type: 'x' for BaseType\_t and any other non-standard types  
(structures, task handles, queue handles, etc.)

a variable is unsigned, it is also prefixed with a 'u'

a variable is a pointer, it is also prefixed with a 'p'

23

copyright © 2015 guileschool.com™



## FreeRTOS 함수(Functions) 이름 규칙

### □ 함수 프리픽스

vTaskPrioritySet() returns a void

xQueueReceive() returns a variable of type BaseType\_t

pvTimerGetTimerID() returns a pointer to void

private functions are prefixed with 'prv'

24

copyright © 2015 guileschool.com™



## FreeRTOS 매크로(Macros) 이름 규칙

### □ 매크로 프리픽스

<b>port</b> (for example, <b>portMAX_DELAY</b> )	portable.h or portmacro.h
<b>task</b> (for example, <b>taskENTER_CRITICAL()</b> )	task.h
<b>pd</b> (for example, <b>pdTRUE</b> )	projdefs.h
<b>config</b> (for example, <b>configUSE_PREEMPTION</b> )	FreeRTOSConfig.h
<b>err</b> (for example, <b>errQUEUE_FULL</b> )	projdefs.h
<b>pdTRUE(pdPASS)</b>	1
<b>pdFALSE(pdFAIL)</b>	0

25

copyright © 2015 guileschool.com™



## 시간 관리 서비스

### □ FreeRTOSConfig.h 의 상수를 설정하여 사용

FreeRTOS 시간관리 서비스	FreeRTOSConfig.h 설정상수
<b>vTaskDelay()</b>	INCLUDE_vTaskDelay
<b>vTaskDelayUntil()</b>	INCLUDE_vTaskDelayUntil
<b>xTaskGetTickCount()</b>	

26

copyright © 2015 guileschool.com™



## 시간 관리 서비스 - vTaskDelay

- **void vTaskDelay( const TickType\_t xTicksToDelay )**
  - ◆ xTicksToDelay : 시간 지연 틱(TICK)
- TICK 인터럽트는 1ms의 주기( 단, 타이머의 설정에 따라 달라짐 )
- 문맥 전환(Context Switch) 발생
- 1 TICK 주기는 configTICK\_RATE\_HZ 설정에 따라 다르다
- 밀리초를 틱(TICK)으로 변환해 주는 매크로 pdMS\_TO\_TICKS()
- 사용 예(Example)

```
#define TIME_100ms (INT16U)((INT32U)configTICK_RATE_HZ *  
100L / 1000L)  
  
void MyTask (void *pdata)  
{  
    for (;;) {  
        vTaskDelay(OS_TIME_100ms); // 100ms 지연  
        vTaskDelay(pdMS_TO_TICKS(100)); // 100ms 지연  
        /* User code */  
    }  
}
```

27

copyright © 2015 guileschool.com™



## 시간 관리 서비스 - vTaskDelayUntil

- **void vTaskDelayUntil (**  
**TickType\_t \*pxPreviousWakeTime,**  
**TickType\_t xTimeIncrement**  
**)**
- 절대 시간에 도달 할 때까지 vTaskDelayUntil ()을 호출하는 작업을 차단됨 상태로 만듬
- 주기적 태스크는 vTaskDelayUntil ()을 사용하여 실행 빈도를 일정하게 유지 할 수 있다
- **pxPreviousWakeTime** 이 시간은 작업이 다음에 차단됨 상태를 벗어나는 시 간을 계산하기위한 참조 점으로 사용됨
- **xTimeIncrement**는 틱 단위로 지정됨. pdMS\_TO\_TICKS () 매크로는 밀리초를 틱으로 변환 하는 데 사용될 수 있다
- 

28

copyright © 2015 guileschool.com™



## 시간 관리 서비스 - vTaskDelayUntil

### □ 사용 예(Example)

```
/* Define a task that performs an action every 50 milliseconds. */
void vCyclicTaskFunction( void * pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = pdMS_TO_TICKS( 50 );
    xLastWakeTime = xTaskGetTickCount();

    /* Enter the loop that defines the task behavior. */
    for ( ;; )
    {
        /* xLastWakeTime is automatically updated within vTaskDelayUntil()
         * so is not explicitly updated by the task. */
        vTaskDelayUntil( &xLastWakeTime, xPeriod );
        /* Perform the periodic actions here. */
    }
}
```

29

copyright © 2015 guileschool.com™



## vTaskDelay 와 vTaskDelayUntil 의 차이점

- vTaskDelay()는 호출 태스크가 vTaskDelay를 호출한 시간부터 지정된 틱 수만큼 Blocked 상태로 들어가고 남아있게한다. 때문에 vTaskDelay()를 호출한 태스크가 Blocked 상태를 종료할 시간은 vTaskDelay()를 호출한 시점과 관련이 있다
- vTaskDelayUntil()은 호출 태스크를 입력한 다음 목표 절대 시간에 도달할 때까지 차단됨 상태를 유지한다. vTaskDelayUntil()을 호출한 태스크는 vTaskDelayUntil()을 호출한 때가 아닌 특정 시간에 Blocked 상태를 정확히 종료합니다

30

copyright © 2015 guileschool.com™



## 시스템 시간 얻기

- **TickType\_t xTaskGetTickCount( void );**  
: 현재의 시스템 시간을 얻어 온다. 틱 수는 스케줄러가 시작된 이후 발생한 틱 인터럽트의 총 수. xTaskGetTickCount()는 현재 틱 계수 값을 반환
- **틱(TICK)** 수는 언젠가는 결국 오버 플로우를 넘어서고 0으로 되돌아감. 작업 이 차단됨 상태인 동안 틱 수가 오버 플로우 되더라도 작업은 항상 지정된 기간 동안 차단되기 때문에 내부적으로 아무런 문제가 없다. 하지만, 응용 프로그램이 틱 수 값을 직접 사용하는 경우 호스트 응용 프로그램에서 오버 플로우를 고려해야한다
- #define **configUSE\_16\_BIT\_TICKS**가 1로 설정된 경우 틱 계수는 **16 비트** 변수에 보관
- #define **configUSE\_16\_BIT\_TICKS**가 0으로 설정된 경우 틱 계수는 **32 비트** 변수에 보관된다

31

copyright © 2015 guileschool.com™



## 목 차

- 태스크(Task)**
- 시간관리 서비스(TIME)**
- **임계영역(CRITICAL SECTION)**
- 문맥전환(CONTEXT SWITCH)**
- IDLE 태스크**
- 인터럽트와 클럭 틱(TICK)**
- 시스템 성능 최적화(OPTIMIZATION)**
- 메모리 관리**

copyright © 2015 guileschool.com™



## 동시성 문제(Concurrent Entrancy)

### TASKS

```
taskOne (void)
{
    myFunc();
    ...
}
```

```
taskTwo (void)
{
    myFunc();
    ...
}
```

### SHARED CODE

```
myFunc (void)
{
    ...
}
```

taskOne's Prio < taskTwo's Prio

static int **COINS** = 21687; //공유변수

myFunc(void) <--- 공유함수

{

**COINS** += 1; <--- 변경(업데이트) 비원자적연산(읽고/수정/쓰기)

}

원쪽 그림처럼 각 태스크가 myFunc 함수에  
동시에 진입하였다.  
**COINS** 값은 얼마가 되어야 하는가?  
보기 (가)21687, (나)21688, (다)21689

33

copyright © 2015 guileschool.com™



## 임계 영역(Critical Section)

- 공유 자원을 사용 중인 함수내의 일부 혹은 전체 영역
- 일단 이 코드 영역의 실행이 시작되면 적어도 다른 태스크(설령 우선순위가 높은 태스크라 할지라도)가 이 영역을 선점하여 실행하는 일이 없어야 하고 또 그렇게 함
- 임계영역을 보호하기 위한 장치는 다음과 같은 것이 있다
  - ◆ 인터럽트 중단
  - ◆ 스케줄링 중단
  - ◆ 세마포어(상호배제 커널 서비스)

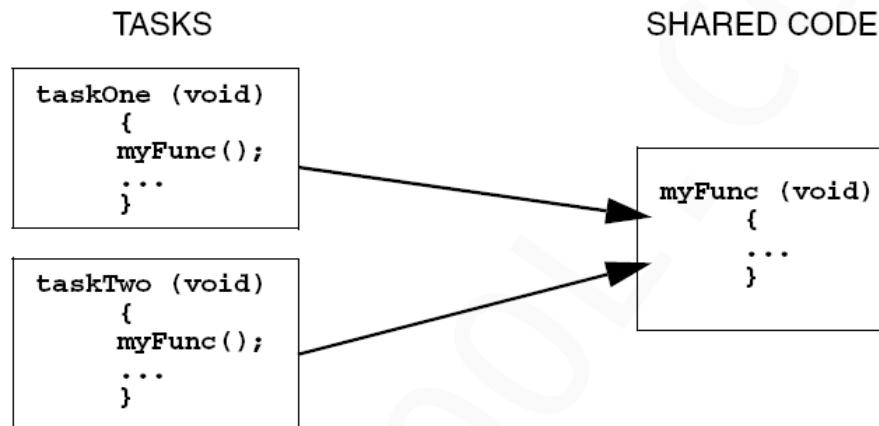
34

copyright © 2015 guileschool.com™



## 재진입(Reentrancy)

- 멀티태스킹 환경 그리고 다수의 태스크에서 호출 하여 사용 할 수 있기 위하여는 해당 함수는 재 진입 가능하도록 작성 되어야 함
- 멀티태스킹 환경이긴 하지만 단일 태스크만이 독점하여 사용 할 것으로 확신 할 수 있는 경우에는 해당 함수는 재 진입이 아니어도 무방



35

copyright © 2015 guileschool.com™



## 재진입(Reentrancy)

- 멀티태스킹 환경에서 함수의 재진입 사용이 불가능한 경우

```
int Temp; /*전역변수*/  
void swap(int *x, int *y)  
{  
    Temp = *x;  
    *x = *y;  
    *y = Temp;  
}
```

36

copyright © 2015 guileschool.com™



## 재진입(Reentrancy)

- 함수를 재진입 가능하도록 하려면
  - ◆ 전역변수를 사용하지 않는다
  - ◆ 세마포어 같은 커널 리소스로 전역변수 보호
  - ◆ 전역변수 사용 동안 인터럽트 작동 임시 중지

```
int Temp; /*전역변수*/  
void swap(int *x, int *y)  
{  
    int Temp;  
    Temp = *x;  
    *x = *y;  
    *y = Temp;  
}
```

37

copyright © 2015 guileschool.com™



## 상호 배제(Mutual Exclusion) 방법 (1)

- 인터럽트 작동을 잠금(중지)
  - ◆ 공유자원을 사용하는 동안 인터럽트를 금지시킴
  - ◆ 임계영역코드의 실행 시간이 비교적 아주 짧은 경우 효과적
  - ◆ 이 때문에 타임 **TICK** 인터럽트의 주기를 놓치지 않도록 사용 해야 함
- 사용 예(**Example**)

```
task1( )  
{  
    ...  
    인터럽트 비 활성화 (disable)  
    공유 자원 액세스 (임계영역)  
    인터럽트 활성화 (enable)  
    ...  
}
```

38

copyright © 2015 guileschool.com™



## 상호 배제(Mutual Exclusion) 방법 (2)

- 스케줄링을 중단(**PAUSE**)
  - ◆ 공유자원을 사용하는 동안 스케줄링을 금지시킴
  - ◆ FreeRTOS 에서는 지원하지 않음
  - ◆ 임계영역코드의 실행 시간이 비교적 아주 짧은 경우 효과적
  - ◆ 이 실행이 빈번 할 경우 높은 우선 순위 태스크 실행이 늦어지는 현상이 발생 할 가능성이 있음

- 사용 예(**Example**)

```
task1( )  
{  
    ...  
    스케줄링 중단(disable)  
    공유 자원 액세스 (임계영역)  
    스케줄링 재개 (enable)  
    ...  
}
```

39

copyright © 2015 guileschool.com™



## 상호 배제(Mutual Exclusion) 방법 (3)

- 세마포어류( 세마포어, 뮤텍스 )의 커널서비스
  - ◆ 상호 배제으로 즐겨 사용되는 방법
  - ◆ 사용이 지나치게 많을 경우 태스크의 블럭킹이 잦아지며 이로 인한 오버헤드가 심함
- 사용 예(**Example**)

```
task1( )  
{  
    ...  
    세마포어 LOCK  
    공유 자원 액세스 (임계영역)  
    세마포어 UN-LOCK  
    ...  
}
```

40

copyright © 2015 guileschool.com™



## 상호 배제(Mutual Exclusion) 방법 (4)

- 상호배제를 사용하지 않는다
  - ◆ 가장 이상적인 방법임
  - ◆ 공유 자원을 되도록 사용하지 않는다
  - ◆ **공유 자원(변수, I/O장치)을 사용하더라도 이 자원을 다수의 태스크가 공유 하도록 하지 않는다**

### □ 사용 예(**Example**)

```
task1()
{
    ...
    세마포어 LOCK
    공유 자원 액세스 (task1이 독점)
    세마포어 UN-LOCK
    ...
}
```

41

copyright © 2015 guileschool.com™



## 크리티컬 섹션(Critical Section)

- **taskENTER\_CRITICAL(), taskEXIT\_CRITICAL()**  
: FreeRTOS에서의 크리티컬 섹션 보호
- 인터럽트 비활성화 시간은 실시간 운영체제의 중요한 성능 지표  
: 리얼 타임 이벤트의 응답 시간 결정
- 응용 프로그램에서도 사용 할 수 있음  
: 단, 인터럽트 비활성화 상태에서 사용 하면 시스템이 멈출 수도 있음  
: ex. vTaskDelay()
- **freeRTOS** 서비스를 호출 할때 인터럽트는 꼭 활성화 상태여야 한다.

### □ 사용 예(**Example**)

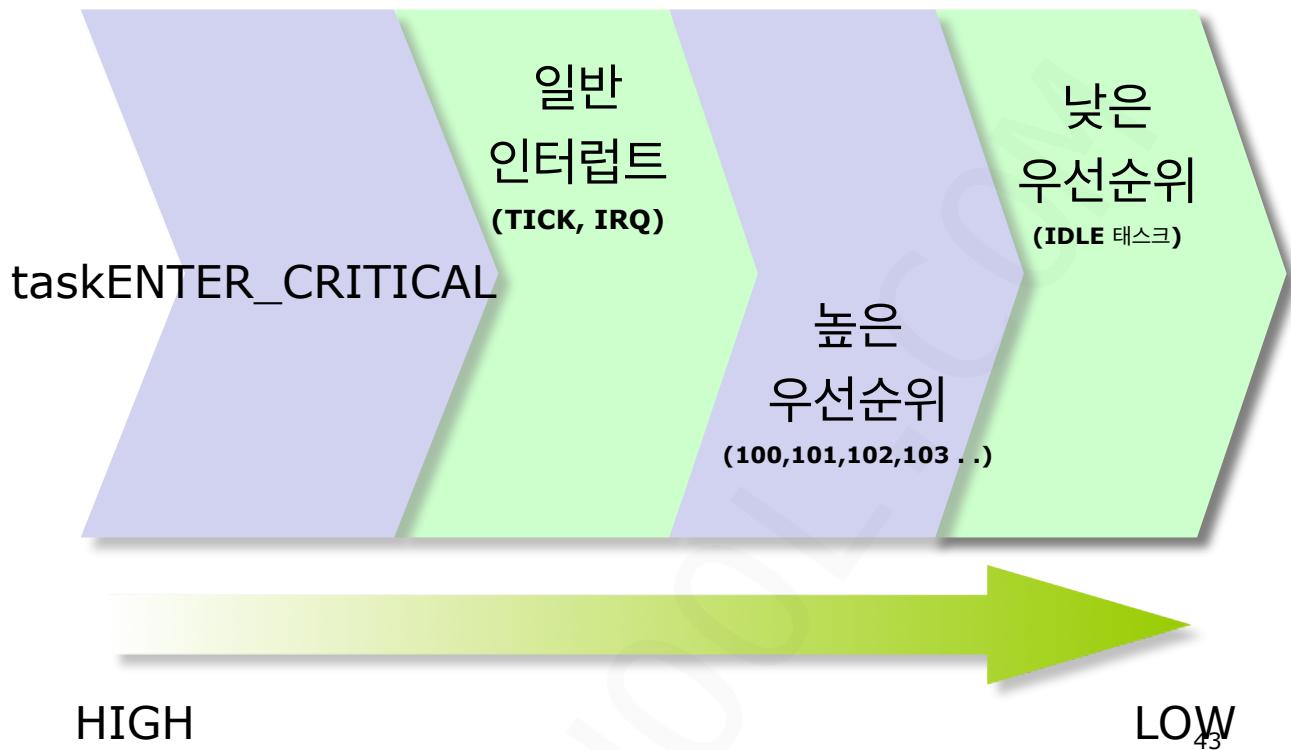
```
{
    taskENTER_CRITICAL();
    /* Critical Section*/
    taskEXIT_CRITICAL();
}
```

42

copyright © 2015 guileschool.com™



## FreeRTOS 의 절대 우선 순위



copyright © 2015 guileschool.com™



실습 : 임계영역 보호(**03\_CRITICAL**)

copyright © 2015 guileschool.com™



## 목 차

---



태스크(**Task**)

시간관리 서비스(**TIME**)

임계영역(**CRITICAL SECTION**)

문맥전환(**CONTEXT SWITCH**)

**IDLE** 태스크

인터럽트와 클럭 틱(**TICK**)

시스템 성능 최적화(**OPTIMIZATION**)

메모리 관리

---

copyright © 2015 guileschool.com™



## TCB(Task Control Blocks)

---

- 태스크 당 각 **1개의 TCB(TCB\_t)**를 갖는다
- 태스크가 다시 **CPU** 사용권을 받을 때 **TCB**에 저장했던 데이터를 이용해서 마지막으로 실행했던 부분부터 코드를 정확히 재개한다
- **TCB\_t**의 내부 구성
  - ◆ **pxTopOfStack** : 현재 태스크가 마지막으로 사용한 스택의 위치
  - ◆ **uxPriority** : 태스크 우선순위
  - ◆ **pxStack** : 태스크 스택의 시작 주소
  - ◆ **pcTaskName** : 태스크 제목
  - ◆ **pxEndOfStack** : 태스크 스택의 마지막 주소
  - ◆ **uxTCBNumber** : TCB 구조체 일련 번호
  - ◆ **uxTaskNumber** : 사용자 정의형 태스크 번호
  - ◆ **uxBasePriority** : Mutex's PIP(Priority Inheritance Priority)



## TCB 메모리 최적화

- **TCB\_t** 구조체에서 어떤 필드는 조건부 컴파일로 싸여져 있다.  
이를 이용하면 사용하지 않는 기능의 필드를 조건부 컴파일로 제거하여  
메모리의 크기를 절약 할 수 있다

47

copyright © 2015 guileschool.com™



## 태스크 문맥전환(CONTEXT SWITCH) (1)

- 문맥 (**CONTEXT**)이란? 모든 **CPU**의 레지스터를 일컫음
- 문맥 전환은 선점된 태스크의 모든 레지스터 값을 스택에 저장한 뒤, 전에 저장했던 레지스터 값을 **CPU**로 복구하여 새로운 태스크를 실행하는 것을 말함
- **vPortYieldProcessor()** 을 이용하여 문맥 전환 수행
- 문맥 전환 코드

```
void vPortYieldProcessor( void )
{
    __asm volatile ( "ADD LR, LR, #4" );
    /* Perform the context switch. First save the context of the current task. */
    portSAVE_CONTEXT();
    /* Find the highest priority task that is ready to run. */
    __asm volatile ( "bl vTaskSwitchContext" );
    /* Restore the context of the new task. */
    portRESTORE_CONTEXT();
}
```

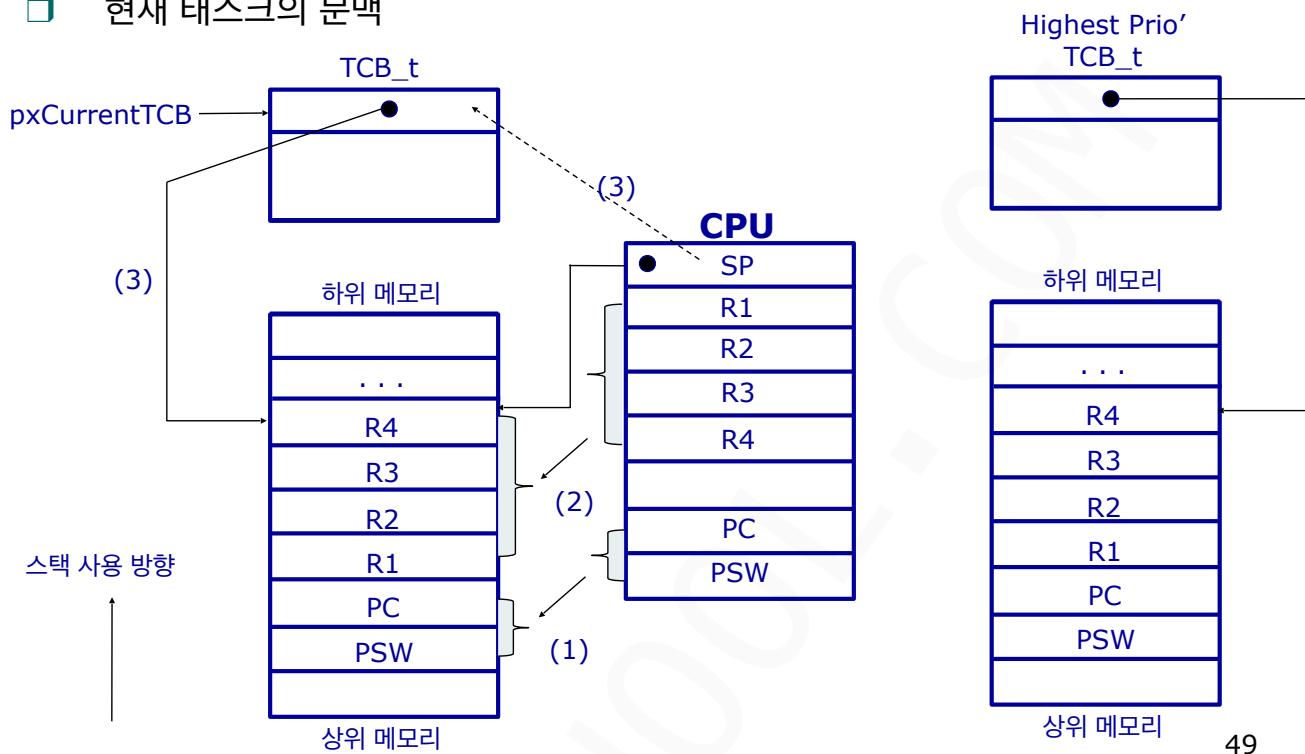
48

copyright © 2015 guileschool.com™



## 태스크 문맥전환(CONTEXT SWITCH) (2)

### 현재 태스크의 문맥



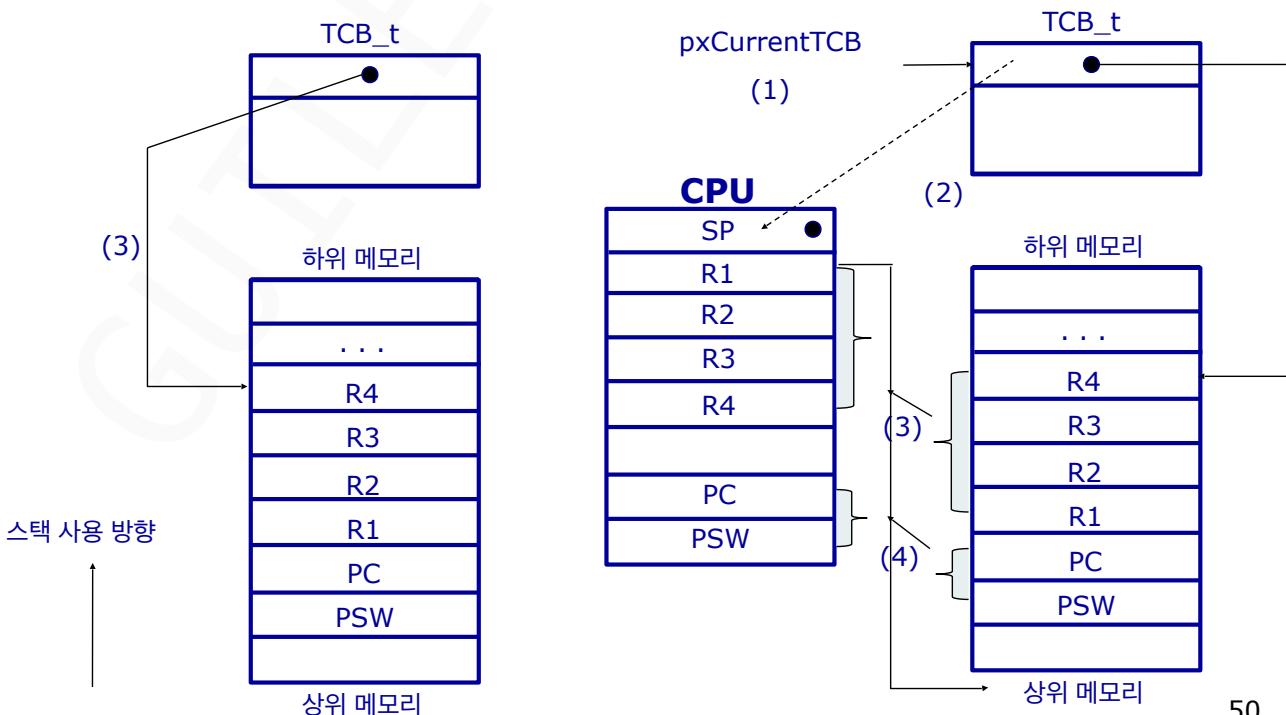
49

copyright © 2015 guileschool.com™



## 태스크 문맥전환(CONTEXT SWITCH) (3)

### 새로운 태스크의 실행



50

copyright © 2015 guileschool.com™



## 목 차

---

태스크(**Task**)

시간관리 서비스(**TIME**)

임계영역(**CRITICAL SECTION**)

문맥전환(**CONTEXT SWITCH**)

□ **IDLE 태스크**

인터럽트와 클럭 틱(**TICK**)

시스템 성능 최적화(**OPTIMIZATION**)

메모리 관리

---

copyright © 2015 guileschool.com™



## IDLE 태스크

---

- **prvIdleTask()** 함수는 우선순위 0으로 최하위 우선순위
- 삭제할 수 없는 태스크
- vTaskStartScheduler()에서 생성됨
- **vApplicationIdleHook()<--- CPU 저전력 모드 구현**

```
static portTASK_FUNCTION( prvIdleTask, pvParameters )
{
    for( ;; )
    {
        #if ( configUSE_PREEMPTION == 0 )
        {
            taskYIELD();
        }
        #endif /* configUSE_PREEMPTION */

        #if ( ( configUSE_PREEMPTION == 1 ) && ( configIDLE_SHOULD_YIELD == 1 ) )
        {
            #if ( configUSE_IDLE_HOOK == 1 )
            {
                vApplicationIdleHook();
            }
            #endif /* configUSE_IDLE_HOOK */
        }
        ...
    }
}
```

---

copyright © 2015 guileschool.com™



## 목 차

태스크(**Task**)

시간관리 서비스(**TIME**)

임계영역(**CRITICAL SECTION**)

문맥전환(**CONTEXT SWITCH**)

**IDLE** 태스크



인터럽트와 클럭 틱(**TICK**)

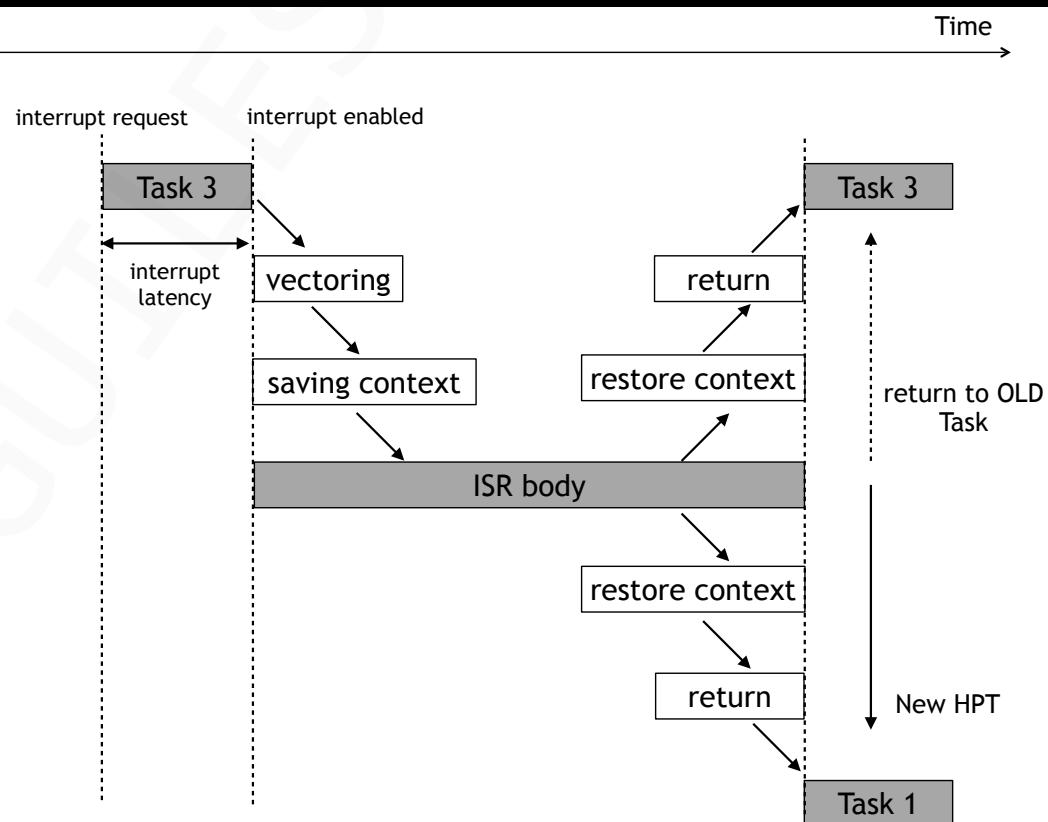
시스템 성능 최적화(**OPTIMIZATION**)

메모리 관리

copyright © 2015 guileschool.com™



## 인터럽트 처리



copyright © 2015 guileschool.com™



## 클럭 틱(Clock TICK)

- 클럭 틱은 초당 **10**회에서 **1000**회 정도
- 틱 발생 주기가 짧으면 시스템의 오버헤드도 함께 증가
- **xTickCount** 전역 변수가 **TICK** 변수로 사용

```
 BaseType_t xTaskIncrementTick( void )
{
    if( uxSchedulerSuspended == ( UBaseType_t ) pdFALSE )
    {
        /* Minor optimisation. The tick count cannot change in this
         * block. */
        const TickType_t xConstTickCount = xTickCount + ( TickType_t ) 1;

        /* Increment the RTOS tick, switching the delayed and overflowed
         * delayed lists if it wraps to 0. */
        xTickCount = xConstTickCount;
        #if ( configUSE_TICK_HOOK == 1 )
        {
            /* Guard against the tick hook being called when the pended tick
             * count is being unwound (when the scheduler is being unlocked). */
            if( uxPendedTicks == ( UBaseType_t ) 0U )
            {
                vApplicationTickHook();
            }
        }
    }
}
```

55

copyright © 2015 guileschool.com™



## FreeRTOS 의 현재 버전 확인

- “**task.h**” 파일에서 다음처럼 선언

```
#define tskKERNEL_VERSION_NUMBER "V10.1.1"
#define tskKERNEL_VERSION_MAJOR 10
#define tskKERNEL_VERSION_MINOR 1
#define tskKERNEL_VERSION_BUILD 1
```

56

copyright © 2015 guileschool.com™



## 목 차

태스크(**Task**)

시간관리 서비스(**TIME**)

임계영역(**CRITICAL SECTION**)

문맥전환(**CONTEXT SWITCH**)

**IDLE** 태스크

인터럽트와 클럭 틱(**TICK**)

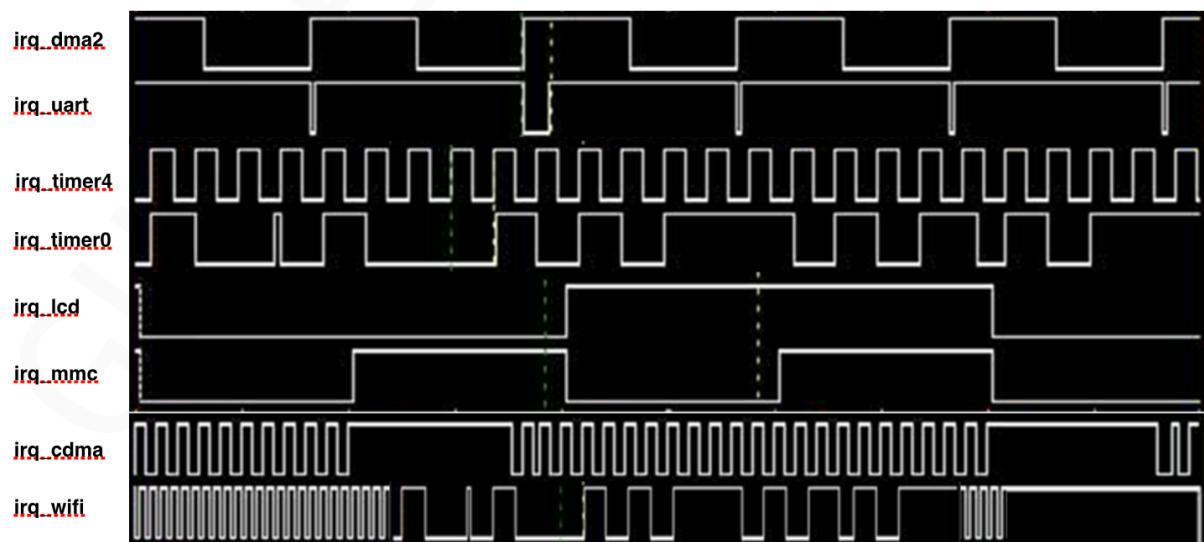
□ 시스템 성능 최적화(**OPTIMIZATION**)

메모리 관리

copyright © 2015 guileschool.com™



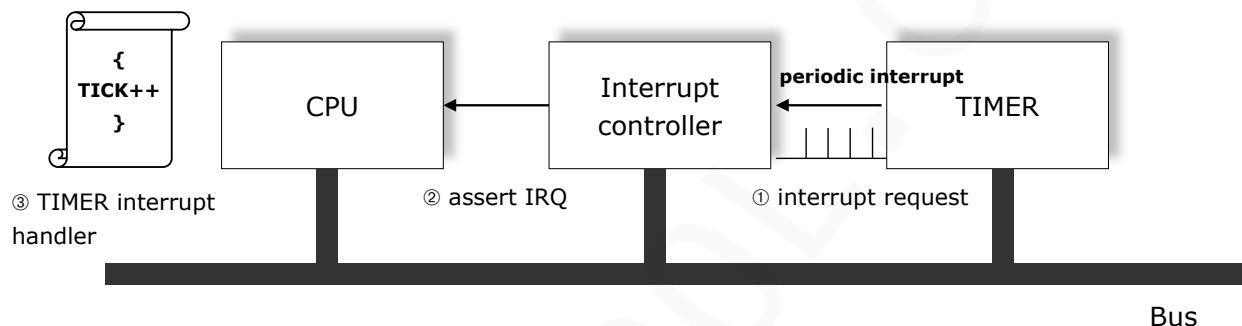
## 인터럽트 성능 최적화





## 지연된 인터럽트(Deferred Interrupt Processing)

- 실행 시간이 많이 소요되는 인터럽트 서비스 루틴의 실행을 지연하여 실행하는 것을 말함
- 처리 가능한 한 ISR의 실행 시간을 짧게 유지하는 것이 중요함
- 태스크의 우선 순위가 아무리 높더라도 실행하고 있는 ISR 보다 먼저 실행될 수 없기 때문. 즉, 태스크는 인터럽트 서비스 루틴이 실행하고 있지 않을 때만 실행 가능하다



59

copyright © 2015 guileschool.com™



## 지연된 인터럽트(Deferred Interrupt Processing)

### 수정전(Before)

```
volatile char *mdeBuff= (char*)0x75600000; // cdma's buffer
```

```
__interrupt void cdma_isr(void)
{
```

....

```
    memcpy(mdeBuff, sysmem, 64000); //23ms
```

....

```
}
```

60

copyright © 2015 guileschool.com™



## 지연된 인터럽트(Deferred Interrupt Processing)

수정후(After)

```
__interrupt void cdma_isr(void)
{
    vTaskResume(task0's handle);
    ...
}

void task0(void* pdata)
{
    ...

    while(1){
        vTaskSuspend(NULL);
        memcpy(mdeBuff, sysmem, 4096*16 ); //23ms
        ...
    }
}
```

61

copyright © 2015 guileschool.com™



## 시스템 성능 최적화 기법

- 전역 변수 선언량 , 이를 사용하는 태스크수와 그 참조 횟수를 줄인다
- 시간 소요가 많은 인터럽트 루틴은 가급적 **IPC(xSemaphoreGive)** 을 활용하여 태스크에서 그 일을 수행하도록 구현한다
- 간단한 코드의 임계영역 보호시 에는 세마포어 보다는 가급적 **taskENTER\_CRITICAL** 을 활용한다
- 빈번히 호출되는 함수를 인라인 어셈블리 등을 이용 하여 속도 최적화
- 가능 구현 이후 프로세서의 클럭 속도를 가감하여 최적의 스피드를 결정한다. <-- **IDLE** 태스크의 **CPU** 점유율 판단
- **TICK** 시간이 절대적으로 지켜지는지 감시한다. <-- 로직스코프 활용

62

copyright © 2015 guileschool.com™



## 우를 범하기 쉬운 ? 우선순위 배정

	태스크명	우선순위		비고
1	<b>TASK_PLAY</b>			음악재생
2	<b>TASK_REC</b>			음성녹음
3	<b>TASK_LCD</b>			화면 출력
4	<b>TASK_I2C</b>			<b>FM 라디오, EEPROM</b>
5	<b>TASK_KEY</b>			사용자 입력
6	<b>TASK_USB</b>			음악 다운로드
7	<b>TASK_JPEG</b>			이미지 뷰어

63

copyright © 2015 guileschool.com™



## 우를 범하기 쉬운 ? 우선순위 배정

- 각 태스크 단위의 기능으로 우선순위의 경쟁을 따져선 곤란하다  
: 예, **I2C** 클럭생성 기능을 구현한 태스크
- 본질적으로 단위 일의 마감시간(혹은 응답시간)을 철저히 준수 하는 개념으로 접근하는 것이 바람직
- 결론  
해당 기능의 구현 단계 부터 그 일의 마감시간을 항시 염두에 두어 진행되어야 한다.

64

copyright © 2015 guileschool.com™



## 목 차

---

태스크(**Task**)

시간관리 서비스(**TIME**)

임계영역(**CRITICAL SECTION**)

문맥전환(**CONTEXT SWITCH**)

**IDLE** 태스크

인터럽트와 클럭 틱(**TICK**)

시스템 성능 최적화(**OPTIMIZATION**)

**메모리 관리**



copyright © 2015 guileschool.com™

---



## FreeRTOS 의 동적 메모리

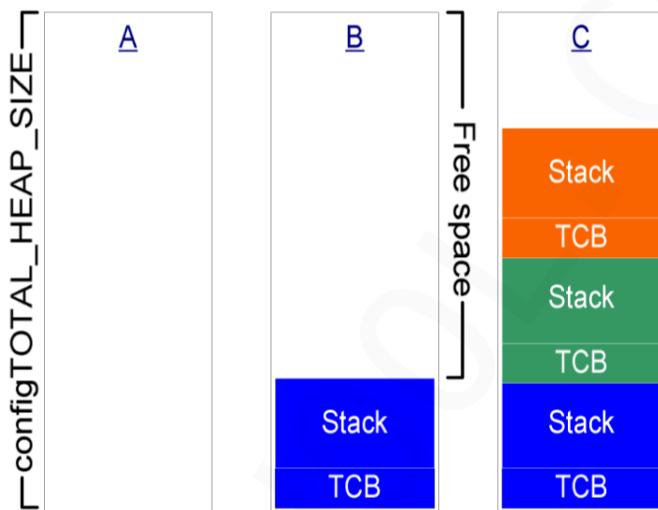
---

- 동적 메모리 사용의 문제점
  - ◆ 메모리 할당 및 해제 시 메모리 단편화 현상 발생
  - ◆ malloc(), free() 함수의 복잡한 알고리즘에 기인한 수행시간 예측 어려움
- FreeRTOS** 메모리 관리는 동적 메모리의 할당 해제 시 메모리 단편화 현상에 대한 대안
- 4가지의 동적 메모리 방법 제공
- heap\_1.c**, **heap\_2.c**, **heap\_3.c** and **heap\_4.c** ( 2번이나 4번을 추천 )
- 메모리 단편화 현상 없음 <-- 고정 동일 크기의 메모리 블럭 구조
- 수행 시간이 일정
  
- pvPortMalloc(), vPortFree() 함수를 사용



## heap\_1.c

- 응용 프로그램이 할당한 메모리는 할당된 채로 계속 남아 있다
- 이 할당 방식은 더 복잡한 메모리 관리 기법을 모두 배제한 방법을 제공한다
- 태스크나 커널에서 할당한 메모리를 삭제하지 않고 사용해도 된다면, 이 경우 heap\_1을 사용할 수 있다
- 힙의 크기는 configTOTAL\_HEAP\_SIZE에 의해 결정된다



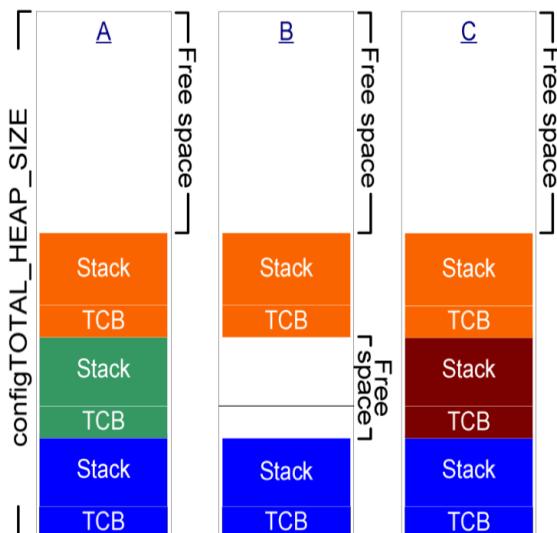
67

copyright © 2015 guileschool.com™



## heap\_2.c

- heap\_1과 달리 할당한 메모리를 언제든지 필요할 때 해제할 수 있다
- 요청한 메모리 사이즈에 가장 맞는 블럭을 할당한다
- 사용 후 해제된 빈 블럭(free mem)은 다음 요청시 재 사용될 수 있다
- heap\_4와 달리 heap\_2는 인접 자유 블록을 하나의 큰 블록으로 결합하지 않음



68

copyright © 2015 guileschool.com™



## heap\_3.c

- heap\_3.c는 표준 라이브러리 malloc() 및 free() 함수를 사용하므로 힙의 크기는 링커 구성 표준에 의해 라이브러리 정의되며 configTOTAL\_HEAP\_SIZE 설정은 힙의 크기에 아무런 영향을 미치지 않는다
- heap\_3은 일시적으로 FreeRTOS를 정지시킴으로써 malloc()과 free()를 thread-safe로 만든다

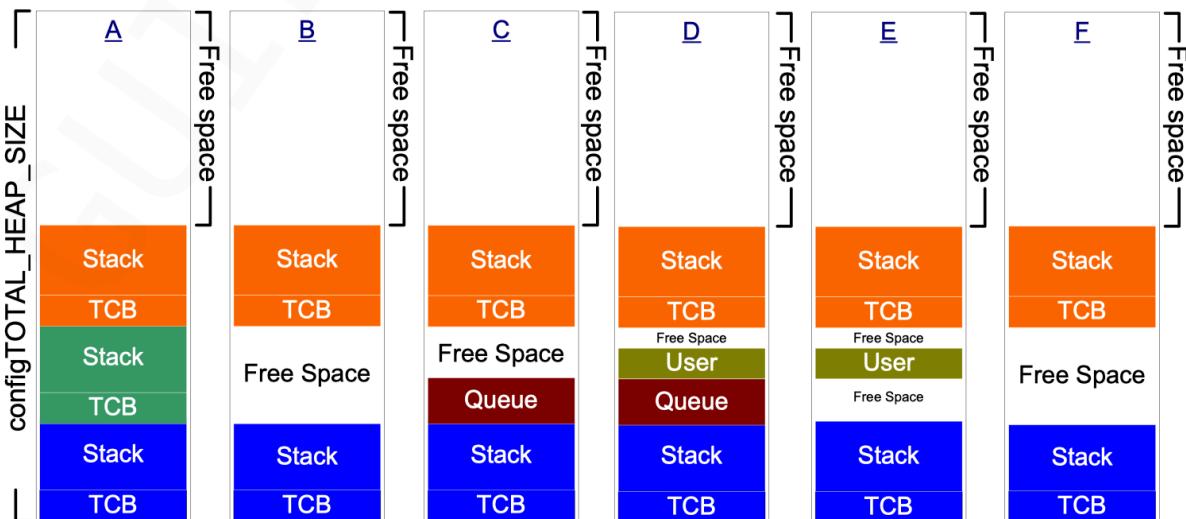
69

copyright © 2015 guileschool.com™



## heap\_4.c

- heap\_4는 heap\_2와 같이 메모리 요청 크기 적합 알고리즘을 사용하여 메모리를 할당한다
- heap\_2와는 달리 heap\_4는 메모리의 인접한 블록을 하나의 큰 블록으로 결합하여 메모리 조각화의 위험을 제거한다



70

copyright © 2015 guileschool.com™



## 연습 문제

---

- FreeRTOS**에서 사용자가 쓸 수 있는 우선순위 범위는 ?
- 스택 메모리 최적화 방법에 대해서 설명하라
- 태스크 문맥전환(**CONTEXT SWITCH**)의 동작 원리에 대해서 설명하라
- FreeRTOS**에서의 인터럽트 처리에 대해서 설명하라
- 클럭 틱(**Clock TICK**)의 주기란 무엇인가?
- 실시간 운영체제에서 메모리 관리상의 주의 할 점을 소개하라

---

copyright © 2015 guileschool.com™

---



## 질의 응답

---

copyright © 2015 guileschool.com™

---

# 태스크간 통신(IPC)

copyright © 2015 guileschool.com™



## 교육 목표

- 세마포어를 활용하고 그 의미를 이해한다.
- 뮤텍스를 활용하고 그 의미를 이해한다.
- 이벤트 플래그를 활용하고 그 의미를 이해한다.
- 메시지 큐를 활용하고 그 의미를 이해한다.
- 교착상태의 정의와 예방 및 해결 방안을 알아본다.

copyright © 2015 guileschool.com™



## 목 차

---



### 세마포어(Semaphore)

**FreeRTOS** 서비스 함수 사용시 주의 사항

뮤텍스(Mutex)

이벤트 플래그(Event Flag)

메시지 큐(Queue)

교착상태(Deadlock)

---

copyright © 2015 guileschool.com™



## 세마포어(Semaphore)

---

```
static int coin= 1; //전역(공유)변수  
R0(태스크1)= 1  
R0'(태스크2)= 1  
task1 > task2 > task3          (HPT: Highest Priority Task)  
SEMAPHORE *sem= sem_create(1); //초기화(1의 값)  
//공유함수, 상호배제!  
deposit( )  
{int tmp;  
P(sem); //세마포어 잠금. if((*sem-1)<0) 태스크블럭(Blocked),  
    else *sem--;  
coin ++; // 임계영역(Critical Section)
```

```
V(sem); //세마포어 잠금해제. *sem++;
```

```
}
```

4

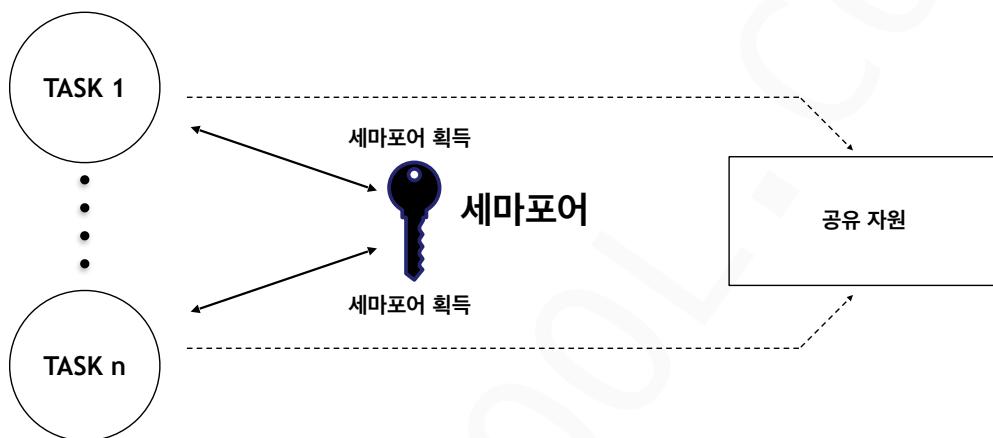
---

copyright © 2015 guileschool.com™



## 세마포어(Semaphore)

- 네덜란드의 수학자 **Edsger Dijkstra**(에츠하르 데이크스트라) 는 상호배제의 개념을 소위 세마포어(**Semaphore**)라는 것으로 추상화 시킴
- 세마포어는 **P**와 **V** 그리고 초기화. 이 세 가지에 의해서만 액세스 되는 일종의 보호된 변수



5

copyright © 2015 guileschool.com™



## 세마포어(Semaphore)을 활용한 IPC

### 생산자 프로세스

```
do {  
    ...  
    아이템을 생산한다.  
    ...  
    아이템을 버퍼에 추가한다.  
    ...  
    //버퍼에 아이템이 있다고 알려준다.  
    sem_signal(sem);  
} while (1);
```

### 소비자 프로세스

```
do {  
    //버퍼에 아이템이 생길 때까지 기다린다.  
    sem_wait(sem);  
    ...  
    버퍼로부터 아이템을 가져온다.  
    ...  
    아이템을 소비한다.  
    ...  
} while (1);
```

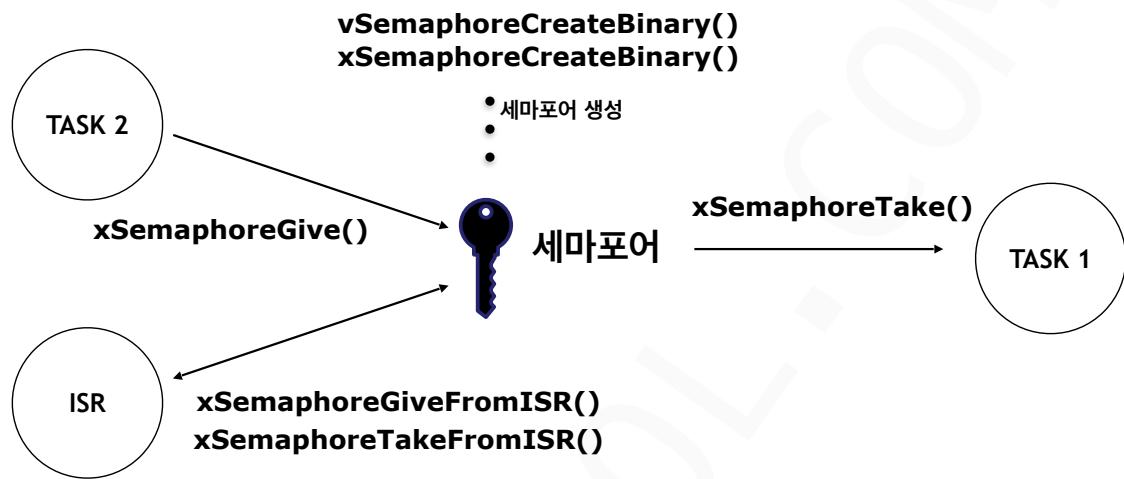
6

copyright © 2015 guileschool.com™



## 세마포어(Semaphore)의 개념도

- 동기화(Synchronization), 이벤트전달, 공유자원보호 용도로 사용



7

copyright © 2015 guileschool.com™



## 세마포어(Semaphore)

- 세마포어 설정 상수(FreeRTOSConfig.h 의 상수를 설정하여 사용)

FreeRTOS 세마포어 서비스	FreeRTOSConfig.h 설정상수
<b>vSemaphoreCreateBinary()</b>	
<b>xSemaphoreCreateBinary()</b>	
<b>xSemaphoreGive()</b>	
<b>xSemaphoreTake()</b>	
<b>xSemaphoreGiveFromISR()</b>	
<b>xSemaphoreTakeFromISR()</b>	
<b>vSemaphoreDelete()</b>	

8

copyright © 2015 guileschool.com™



## 세마포어 서비스 - vSemaphoreCreateBinary

- **void vSemaphoreCreateBinary( SemaphoreHandle\_t xSemaphore )**
  - ◆ **xSemaphore:** 생성되는 세마포어의 핸들을 저장할 SemaphoreHandle\_t 유형의 변수
- 사용 예(**Example**)

```
SemaphoreHandle_t xSemaphore;
void vATask( void * pvParameters ) {
    /* Attempt to create a semaphore. */
    vSemaphoreCreateBinary(xSemaphore);

    if( xSemaphore == NULL ) {
        ...
    }
}
```

9

copyright © 2015 guileschool.com™



## 세마포어 서비스 - xSemaphoreCreateBinary

- **SemaphoreHandle\_t xSemaphoreCreateBinary( )**
  - ◆ **xSemaphore:** 생성되는 세마포어의 핸들을 저장할 SemaphoreHandle\_t 유형의 변수
- 사용 예(**Example**)

```
SemaphoreHandle_t xSemaphore;
void vATask( void * pvParameters ) {
    /* Attempt to create a semaphore. */
    xSemaphore = xSemaphoreCreateBinary();

    if( xSemaphore == NULL ) {
        ...
    }
}
```

10

copyright © 2015 guileschool.com™



## 세마포어 서비스 - vSemaphoreDelete

- **void vSemaphoreDelete( SemaphoreHandle\_t xSemaphore )**
- 세마포어 삭제전 해당 세마포어를 사용하고 있는 모든 태스크를 먼저 삭제하여야 함
- 가급적 사용하지 않는 것이 바람직.

11

copyright © 2015 guileschool.com™



## 세마포어 서비스 - xSemaphoreTake, xSemaphoreGive

- **BaseType\_t xSemaphoreTake( SemaphoreHandle\_t xSemaphore, TickType\_t xTicksToWait )**
  - : 세마포어 대기
    - ◆ **xSemaphore** : 세마포어 핸들
    - ◆ **xTicksToWait** : 'TICK' 단위의 타임아웃 값
- **BaseType\_t xSemaphoreGive( SemaphoreHandle\_t xSemaphore )**
  - : 세마포어 전달(반환)

12

copyright © 2015 guileschool.com™



## xSemaphoreTakeFromISR / xSemaphoreGiveFromISR

- **BaseType\_t xSemaphoreGiveFromISR( SemaphoreHandle\_t xSemaphore,  
BaseType\_t \*pxHigherPriorityTaskWoken )**
  - : ISR에서 사용할 수있는 xSemaphoreGive ()의 버전
    - ◆ **xSemaphore** : 세마포어 핸들
    - ◆ **\*pxHigherPriorityTaskWoken** : 휴면상태 깨어남과 동시 문맥전환 결정
  
- **BaseType\_t xSemaphoreTakeFromISR( SemaphoreHandle\_t xSemaphore,  
BaseType\_t \*pxHigherPriorityTaskWoken )**
  - : ISR에서 호출 할 수있는 xSemaphoreTake () 버전
    - ◆ **xSemaphore** : 세마포어 핸들
    - ◆ **\*pxHigherPriorityTaskWoken** : 휴면상태 깨어남과 동시 문맥전환 결정

13

copyright © 2015 guileschool.com™



## 세마포어 서비스 - xSemaphoreCreateCounting

- **SemaphoreHandle\_t xSemaphoreCreateCounting( UBaseType\_t uxMaxCount,  
UBaseType\_t uxInitialCount )**
  - : 카운팅 세마포어를 만들고 세마포어를 참조 할 수있는 핸들을 반환
    - ◆ **uxMaxCount** : 도달 할 수있는 최대 카운트 값
    - ◆ **uxInitialCount** : 세마포어에 할당 된 카운트 초기 값
  
- **UBaseType\_t uxSemaphoreGetCount( SemaphoreHandle\_t xSemaphore )**
  - : 세마포어의 수를 돌려줌
    - ◆ **xSemaphore** : 세마포어 핸들

14

copyright © 2015 guileschool.com™



- **V** stands for *verhogen* ("increase").  
**P** stands for *proberen* for "to test" or "to try"
  
- **P(proberen) / V(verhogen)**  
**SemPend** == **SemTake** == **SemObtain**  
**SemPost** == **SemGive** == **SemRelease**  
**VRTXsa**      **VxWorks**      **NucleusPlus**  
**UC/OS II**      **FreeRTOS**

15

copyright © 2015 guileschool.com™

---



## 실습 : 세마포어(**05\_SEM**)

copyright © 2015 guileschool.com™

---



## 태스크 간 통신(IPC)

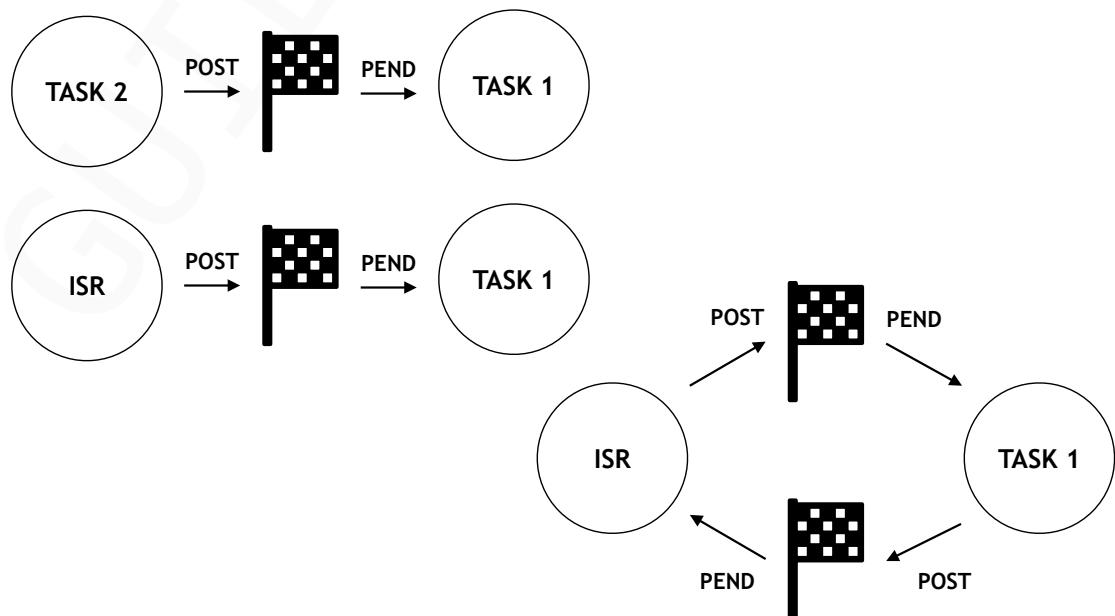
- 태스크나 ISR이 다른 태스크에게 정보 전달 시 필요
- IPC 방법
  - ◆ 전역 변수 이용 --> 각 태스크나 ISR 변수에 독점적 액세스 보장
  - ◆ 메시지 전송 : 메시지 메일박스(MAILBOX), 메시지 큐(QUEUE)

copyright © 2015 guileschool.com™



## 태스크 동기화(Task Synchronization)

- 태스크 동기화 유형
  - ◆ **ISR** 과 태스크
  - ◆ 태스크와 태스크( 단방향, 양방향 )



18

copyright © 2015 guileschool.com™



## 목 차

### 세마포어(Semaphore)

- **FreeRTOS 서비스 함수 사용시 주의 사항**
  - 뮤텍스(Mutex)
  - 이벤트 플래그(Event Flag)
  - 메시지 큐(Queue)
  - 교착상태(Deadlock)

copyright © 2015 guileschool.com™



## FreeRTOS 서비스 함수 사용시 주의 사항

- 각종 **FreeRTOS** 서비스 함수를 사용 할시 각 함수의 리턴값을 확인 하는 습관을 들여야 함
- 예를 들어 **xSemaphore = xSemaphoreCreateBinary(1)** 의 경우 **semaphore** 객체 할당에 실패 할 경우 리턴값으로 '**NULL**' 을 전달함.
- 사용 예(**Example**)

```
SemaphoreHandle_t xSemaphore;  
  
void vATask( void * pvParameters ) {  
  
    /* Attempt to create a semaphore. */  
    xSemaphore = xSemaphoreCreateBinary();  
    if( xSemaphore == NULL ) {  
        /* There was insufficient FreeRTOS heap available for the  
         * semaphore to be created successfully. */  
    }  
    else {  
        ...  
    }  
}
```

20

copyright © 2015 guileschool.com™



## 목 차

### 세마포어(Semaphore)

FreeRTOS 서비스 함수 사용시 주의 사항



### 뮤텍스(Mutex)

### 이벤트 플래그(Event Flag)

### 메시지 큐(Queue)

### 교착상태(Deadlock)

copyright © 2015 guileschool.com™

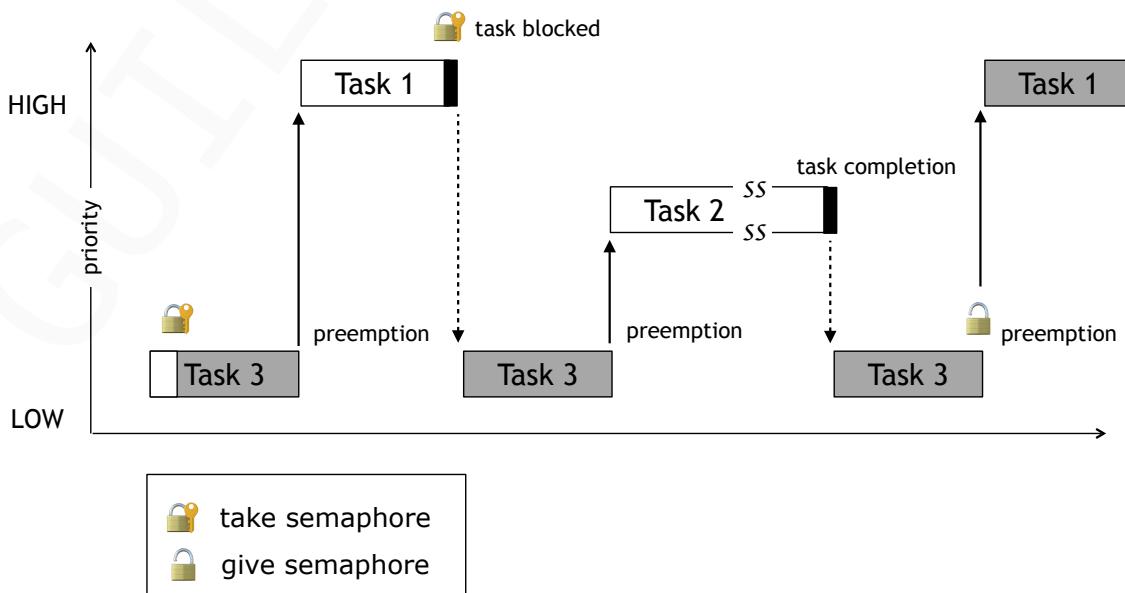


## 우선순위역전(Priority Inversion)



### Priority Inversion

- ◆ 높은 우선순위의 task가 낮은 우선순위 task의 수행이 끝날 때까지 기다리는 상황



22

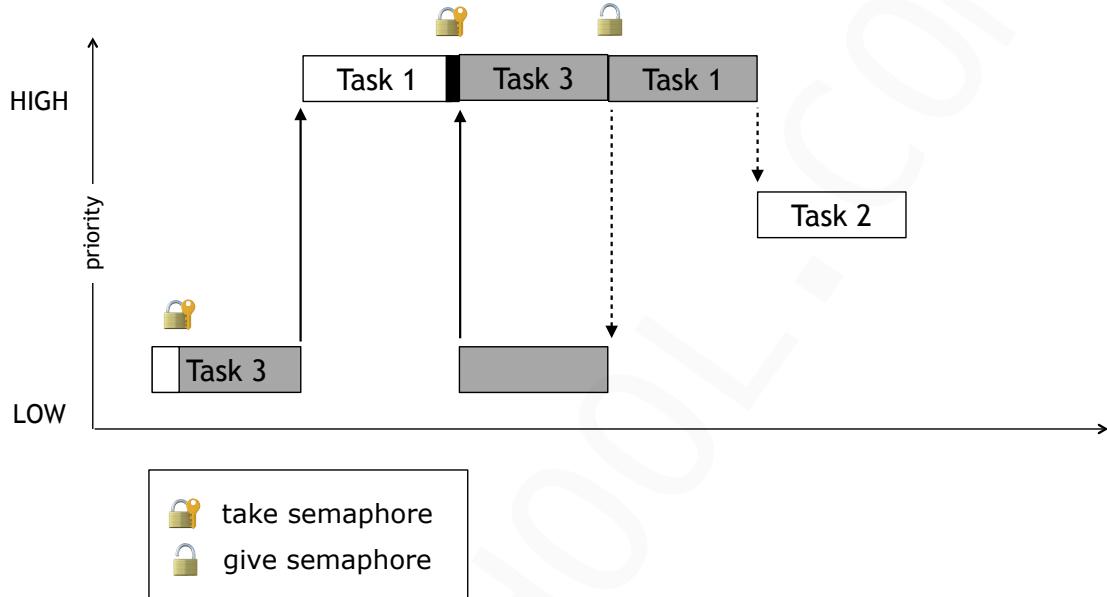
copyright © 2015 guileschool.com™



## 우선순위상속(Priority Inheritance)

### □ Priority Inheritance

- ◆ Priority Inversion 현상이 안 생기도록 하는 효과
- ◆ 높은 우선순위의 task가 블럭상태인 동안, 그 task를 기다리도록 만든 task의 우선순위를 그 보다 높은 task 우선순위로 올리는 것

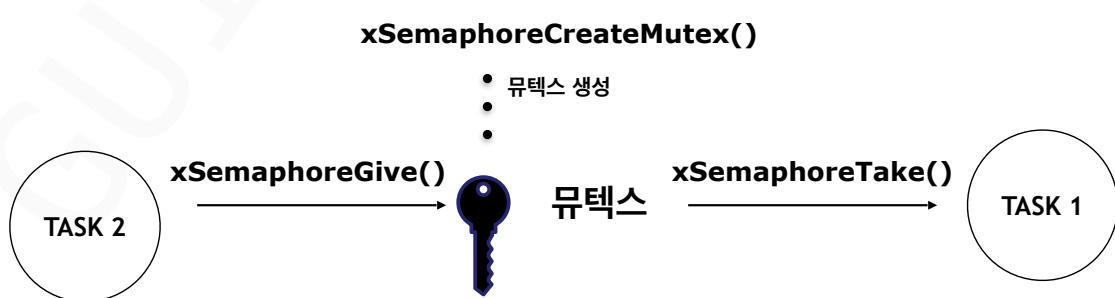


23



## 상호배제세마포어(MUTEX)의 개념도

- 태스크가 자원에 대한 독점적인 액세스를 얻고자 할때 용도로 사용
- 일종의 '바이너리 세마포어' 이다
- 우선순위전도 (**Priority Inversion**) 문제를 해결하는 능력이 있다



24



## 상호배제세마포어(MUTEX)

- 뮤텍스 설정 상수(**FreeRTOSConfig.h** 의 상수를 설정하여 사용)

FreeRTOS 뮤텍스 서비스	FreeRTOSConfig.h 설정상수
<b>xSemaphoreCreateMutex()</b>	
<b>xSemaphoreTake()</b>	
<b>xSemaphoreGive()</b>	

25

copyright © 2015 guileschool.com™



## 뮤텍스 서비스 - xSemaphoreCreateMutex

- **SemaphoreHandle\_t xSemaphoreCreateMutex( void )**
- 뮤텍스 구성요소
- 사용 예(**Example**)

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters ) {
    /* Attempt to create a mutex type semaphore. */
    xSemaphore = xSemaphoreCreateMutex();
    if( xSemaphore == NULL ) {
        /* There was insufficient heap memory available for the mutex to
        be created. */
    } else {
        /* The mutex can now be used. The handle of the created mutex
        will be stored in the xSemaphore variable. */
        ...
    }
}
```

26

copyright © 2015 guileschool.com™

---

## 실습 : 상호배제의 도구(**07\_MUTEX**)

copyright © 2015 guileschool.com™

---



### 멀티 프로세서 환경에서의 상호배제 문제

---

- 멀티 프로세서 환경에서는 인터럽트 금지, 세마포어가 더 이상 효력을 갖지 못함
- **LDREX**는 메모리로 부터 데이터를 읽고, 동시에 메모리 주소에 태깅
- **STREX**는 데이터를 메모리에 저장하는데, 태그가 여전히 유효한 경우만 저장 함. 그외의 경우에는 메모리는 수정되지 않음.
  
- **LDREX/STREX** 을 활용한 잠금 기능 구현(**lock\_mutex**) 예제

```
LOCKED EQU 1  
UNLOCKED EQU 0
```

```
lock_mutex  
; Is mutex locked?  
LDREX r1, [r0] ; Check if locked  
CMP r1, #LOCKED ; Compare with "locked"  
WFEEQ ; Mutex is locked, go into standby  
BEQ lock_mutex ; On waking re-check the mutex
```

계속 . . .



## 멀티 프로세서 환경에서의 상호배제 문제

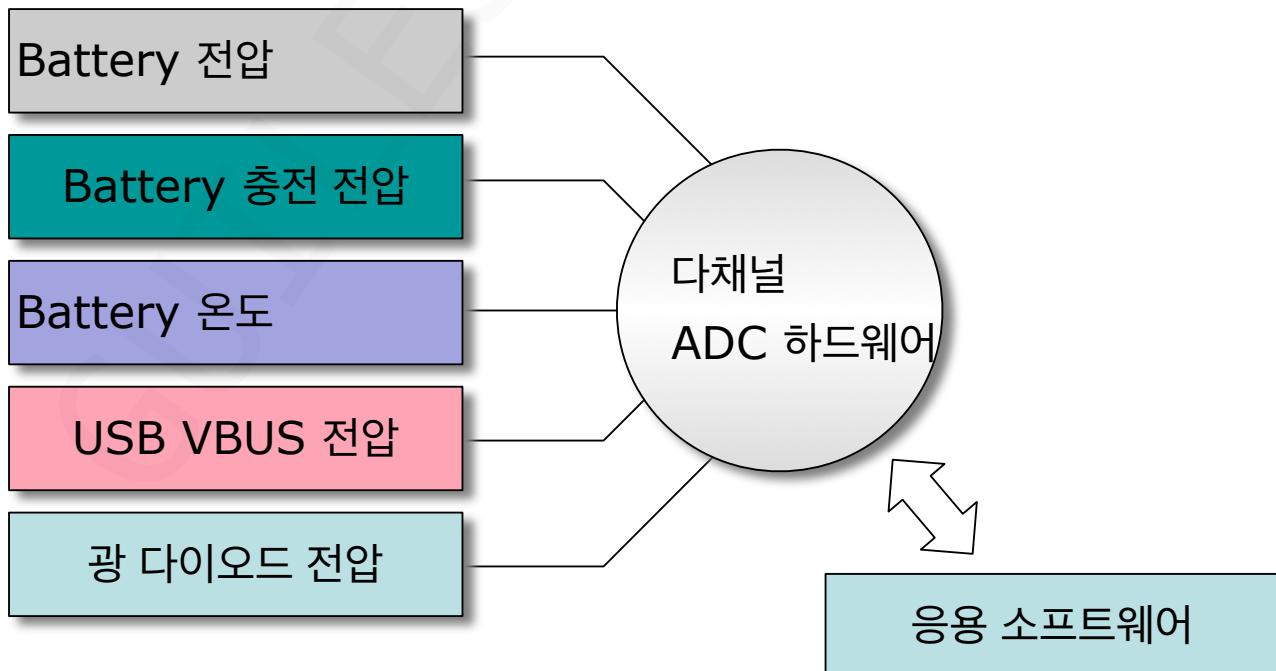
```
; Attempt to lock mutex  
MOV r1, #LOCKED  
STREX r2, r1, [r0] ; Attempt to lock mutex  
CMP r2, #UNLOCKED ; Check whether store completed BNE lock_mutex ;  
                     If store failed, try again  
BNE lock_mutex  
  
BX lr  
  
unlock_mutex  
MOV r1, #UNLOCKED ; Write "unlocked" into lock field  
STR r1, [r0]  
SEV ; Send event to other CPUs, wakes any CPU waiting on using  
WFE BX lr  
BX lr
```

29

copyright © 2015 guileschool.com™



## 상호배제를 고려한 I/O 장치 드라이버의 예 (1)

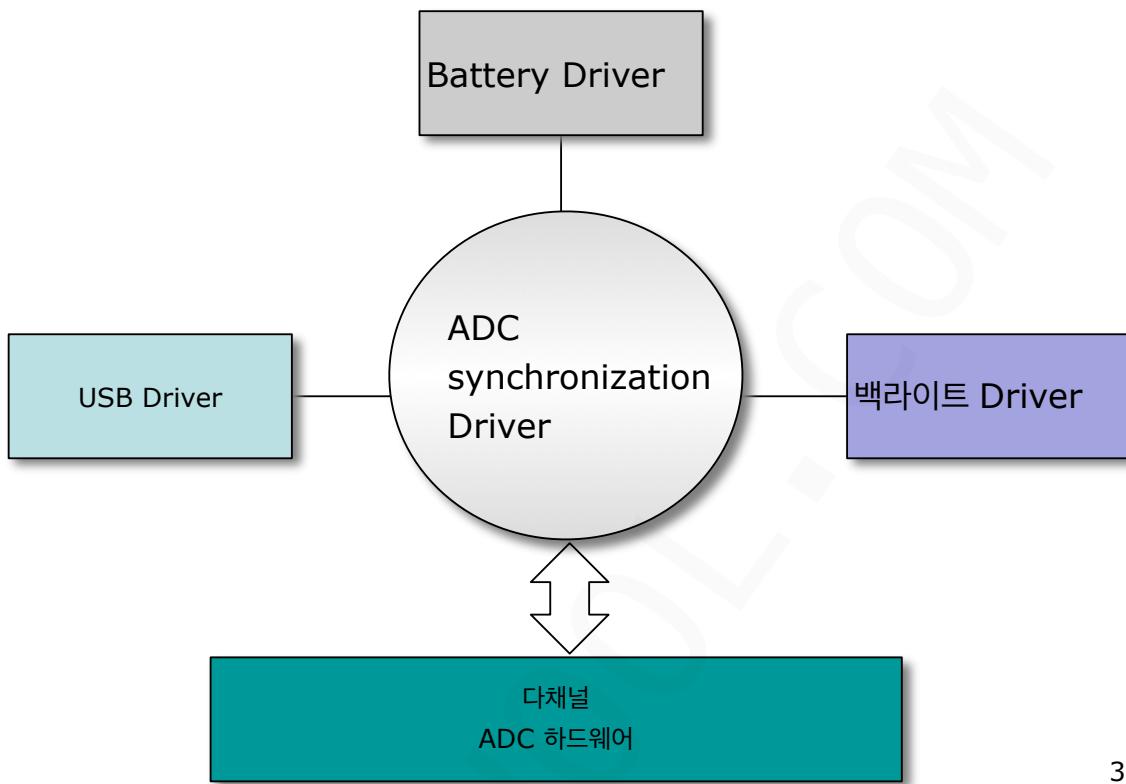


30

copyright © 2015 guileschool.com™



## 상호배제를 고려한 I/O 장치 드라이버의 예 (2)



31

copyright © 2015 guileschool.com™



## 상호배제를 고려한 I/O 장치 드라이버의 예 (3)

```
BOOL ADConversion  
(USHORT *pOutBuffer, DWORD OutSize)  
{  
    입력 파라메터의 유효성 검사  
    ↓  
    ENTER_CRITICAL_SECTION  
    AD 변환 시작  
    변환완료까지 대기(wait)  
    결과를 사용자 버퍼(pOutBuffer)에 저장  
    LEAVE_CRITICAL_SECTION  
  
    return SUCCESS;  
}
```

상대적으로 느린 Synchronization object 인 MUTEX 보다 light weight critical section 을 사용 함으로써 성능 향상을 기대 해 볼 수도 있다.

```
enum ADChannels  
{  
    AD_CHANNEL_BAT_VOLTAGE = 0,  
    AD_CHANNEL_BAT_CHG_VOLTAGE,  
    AD_CHANNEL_BAT_CHG_CURRENT,  
    AD_CHANNEL_BAT_TEMP,  
    AD_CHANNEL_USB_VBUS,  
    AD_CHANNEL_PHOTODIODE,  
    TOTAL_AD_CHANNELS  
};
```

copyright © 2015 guileschool.com™



## 목 차

### 세마포어(Semaphore)

FreeRTOS 서비스 함수 사용시 주의 사항

### 뮤텍스(Mutex)



### 이벤트 플래그(Event Flag)

### 메시지 큐(Queue)

### 교착상태(Deadlock)

copyright © 2015 guileschool.com™



## 이벤트 플래그(EVENT FLAG)의 활용 예

□ 자동차(Automobile) 의 예

```
#define STATUS_ENGINE (1<<0)
#define STATUS_MISSION (1<<1)
#define STATUS_STEERING (1<<2)
#define STATUS_BREAK (1<<3)
#define STATUS_AIRBAG (1<<4)
```

**EventGroupHandle\_t group\_id;**

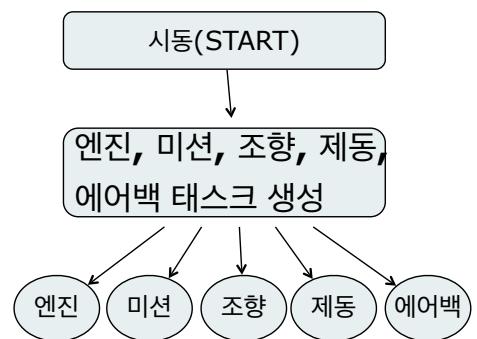
메인 태스크

**0)시동(START!)**

**1)엔진, 미션, 조향, 제동, 에어백 태스크 생성**

**2)각 태스크의 진단 결과를 대기**

**3)주행 시작**



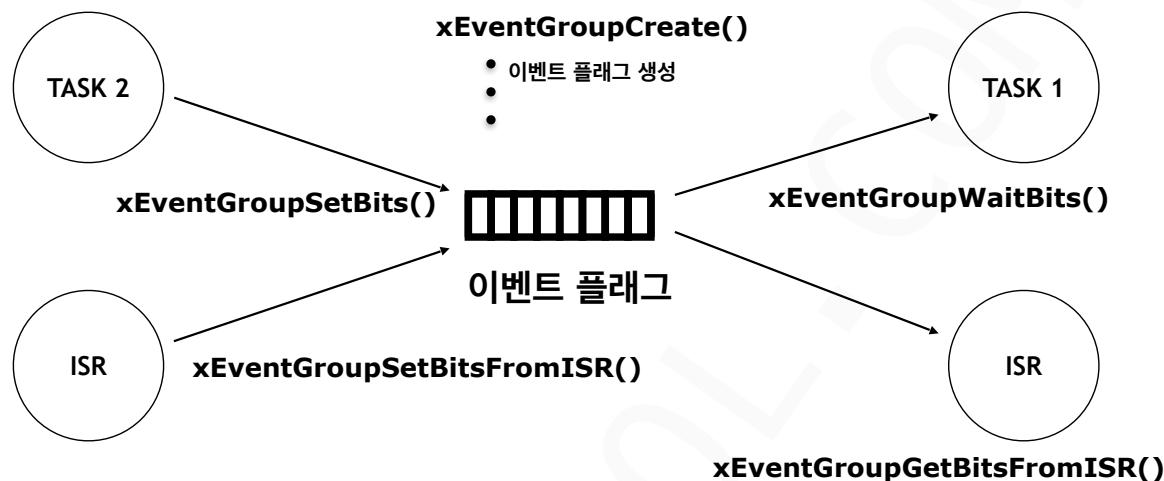
주행 시작

copyright © 2015 guileschool.com™



## 이벤트 플래그(EVENT FLAG)의 개념도

- 이벤트를 이용한 동기화 용도로 사용



35

copyright © 2015 guileschool.com™



## 이벤트 플래그(EVENT FLAG)

- 이벤트플래그 설정 상수(**FreeRTOSConfig.h** 의 상수를 설정하여 사용)

FreeRTOS 이벤트플래그 서비스	FreeRTOSConfig.h 설정상수
<b>xEventGroupCreate()</b>	
<b>vEventGroupDelete()</b>	
<b>xEventGroupWaitBits()</b>	
<b>xEventGroupSetBits()</b>	

36

copyright © 2015 guileschool.com™



## 이벤트 플래그 서비스 - xEventGroupCreate

- **EventGroupHandle\_t xEventGroupCreate( void )**
- 새 이벤트 그룹을 작성하고 작성된 이벤트 그룹을 참조 할 수있는 핸들을 리턴
- 사용 예(**Example**)

```
void main (void)
{
/* Declare a variable to hold the created event group. */
EventGroupHandle_t xCreatedEventGroup;

xCreatedEventGroup = xEventGroupCreate(); /* Was the event group
created successfully? */

if( xCreatedEventGroup == NULL ) {
    /* The event group was not created because there was insufficient
       FreeRTOS heap available. */
}
...
}
```

37

copyright © 2015 guileschool.com™



## 이벤트 플래그 서비스 - vEventGroupDelete

- **void vEventGroupDelete( EventGroupHandle\_t xEventGroup )**
- 이벤트 플래그 삭제전 해당 이벤트 플래그를 사용하고 있는 모든 태스크를 먼저 삭제하여야 함
- 가급적 사용하지 않는 것이 바람직.

38

copyright © 2015 guileschool.com™



## xEventGroupWaitBits, xEventGroupSetBits

- **EventBits\_t xEventGroupWaitBits(**  
    **const EventGroupHandle\_t xEventGroup,**  
    **const EventBits\_t uxBitsToWaitFor,**  
    **const BaseType\_t xClearOnExit,**  
    **const BaseType\_t xWaitForAllBits,**  
    **TickType\_t xTicksToWait )**
- : 이벤트 플래그 대기
  - ◆ **xEventGroup** : 이벤트 플래그 그룹
  - ◆ **uxBitsToWaitFor** : 이벤트 플래그 대기 패턴(비트)
  - ◆ **xClearOnExit** : 이벤트 플래그 값을 읽어내고, 해당 값은 초기값으로 돌아감
  - ◆ **xWaitForAllBits** : AND / OR 조건의 대기 방법
  - ◆ **xTicksToWait** : 대기 시간
- **EventBits\_t xEventGroupSetBits( EventGroupHandle\_t xEventGroup, const EventBits\_t uxBitsToSet )**  
: 이벤트 플래그 전달(반환)

39

copyright © 2015 guileschool.com™



## 이벤트 플래그 사용 예

```
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup ) {
    EventBits_t uxBits;
    const TickType_t xTicksToWait = pdMS_TO_TICKS( 100 );
    uxBits = xEventGroupWaitBits(
        xEventGroup, /* The event group being tested. */
        BIT_0 | BIT_4, /* The bits within the event group to wait for. */
        pdTRUE, /* BIT_0 and BIT_4 should be cleared before returning. */
        pdFALSE, /* Don't wait for both bits, either bit will do. */
        xTicksToWait );/* Wait a maximum of 100ms */

    if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) ) {
        /* xEventGroupWaitBits() returned because both bits were set. */
    }
    ...
}
```

이 함수(xEventGroupSetBits)에 의해 휴면에서 깨어남  
err = xEventGroupSetBits(xEventGroup,  
                          BIT\_4);

40

copyright © 2015 guileschool.com™



---

## 실습 : 이벤트 플래그(**09\_FLAG**)

copyright © 2015 guileschool.com™

---



### 목 차

---

세마포어(**Semaphore**)

**FreeRTOS** 서비스 함수 사용시 주의 사항

뮤텍스(**Mutex**)

이벤트 플래그(**Event Flag**)

메시지 큐(**Queue**)

교착상태(**Deadlock**)



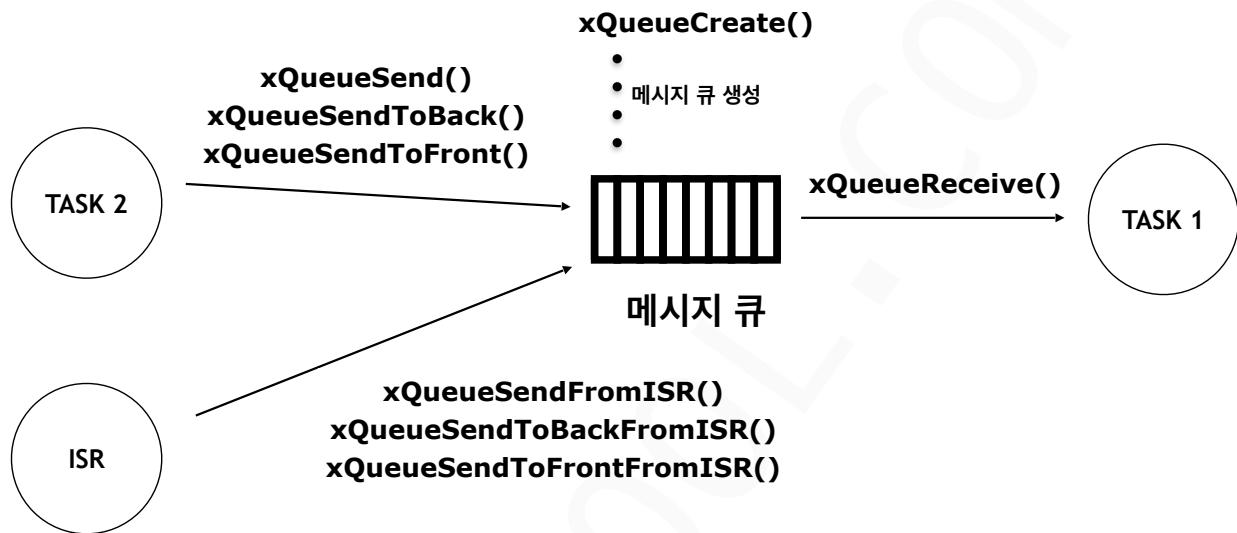
---

copyright © 2015 guileschool.com™



## 메시지큐(QUEUE)의 개념도

- 태스크 혹은 **ISR**에서 다른 태스크로 포인터 변수를 전송
- 포인터는 '사용자 정의 메시지' 자료를 가리킨다



43

copyright © 2015 guileschool.com™



## 메시지큐(QUEUE)

- 메시지큐 설정 상수(**FreeRTOSConfig.h** 의 상수를 설정하여 사용)

FreeRTOS 메시지큐 서비스	FreeRTOSConfig.h 설정상수
<b>xQueueCreate()</b>	
<b>xQueueReceive()</b>	
<b>xQueueSend()</b>	
<b>xQueueSendToFront()</b>	
<b>xQueueSendToBack()</b>	
<b>uxQueueMessagesWaiting()</b>	
<b>xQueueReset()</b>	
<b>xQueueSendFromISR</b>	
<b>xQueueSendToFrontFromISR</b>	
<b>xQueueSendToBackFromISR</b>	

44

copyright © 2015 guileschool.com™



## 메시지큐 서비스 - xQueueCreate

### ❑ QueueHandle\_t xQueueCreate(     UBaseType\_t uxQueueLength,     UBaseType\_t uxItemSize )

- ◆ **uxQueueLength** : 생성되는 큐가 한 번에 보유 할 수 있는 최대 항목 수
- ◆ **uxItemSize** : 대기열에 저장할 수 있는 각 데이터 항목의 크기 (바이트)

### ❑ 사용 예(Example)

```
struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
};

void vATask( void *pvParameters )
{
    QueueHandle_t xQueue1, xQueue2;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );
    if( xQueue1 == 0 )
```

45

copyright © 2015 guileschool.com™



## 메시지큐 서비스 - vQueueDelete

### ❑ void vQueueDelete( TaskHandle\_t pxQueueToDelete )

- ❑ 메시지큐 삭제전 해당 메시지큐를 사용하고 있는 모든 태스크를 먼저 삭제하여야 함
- ❑ 가급적 사용하지 않는 것이 바람직.

46

copyright © 2015 guileschool.com™



## 메시지큐 서비스 - xQueueReceive, xQueueSend

### □ **BaseType\_t xQueueReceive(**     **QueueHandle\_t xQueue,**     **void \*pvBuffer,**     **TickType\_t xTicksToWait )**

: 대기열에서 항목을 수신 (읽음)

- ◆ **xQueue:** 큐의 핸들
- ◆ **\*pvBuffer :** 수신된 데이터가 복사 될 메모리에 대한 포인터
- ◆ **xTicksToWait :** 최대 대기 시간

### □ **BaseType\_t xQueueSend(**     **QueueHandle\_t xQueue,**     **const void \* pvItemToQueue,**     **TickType\_t xTicksToWait )**

: 항목을 대기열의 뒤쪽으로 보내거나 쓴다

47

copyright © 2015 guileschool.com™



## 메시지큐 서비스 - xQueueSendToFront, xQueueSendToBack

### □ **BaseType\_t xQueueSendToFront(**     **QueueHandle\_t xQueue,**     **const void \* pvItemToQueue,**     **TickType\_t xTicksToWait )**

: 항목을 대기열의 앞으로 보내거나 쓴다

### □ **BaseType\_t xQueueSendToBack(**     **QueueHandle\_t xQueue,**     **const void \* pvItemToQueue,**     **TickType\_t xTicksToWait )**

: 항목을 대기열의 뒤로 보내거나 쓴다

**xQueueSend ()**와 **xQueueSendToBack ()**은 동일한 연산을 수행

48

copyright © 2015 guileschool.com™



## 메시지큐 서비스 - uxQueueMessagesWaiting

- **UBaseType\_t uxQueueMessagesWaiting( const QueueHandle\_t xQueue )**
  - : 큐에 현재 보관 유지되고있는 아이템의 수를 돌려준다
  - ◆ **xQueue** : 큐 핸들
  
- **BaseType\_t xQueueReset( QueueHandle\_t xQueue )**
  - : 메시지큐 안의 메시지를 모두 버리고 막 생성된 큐 상태로 만듦
  - ◆ **xQueue** : 큐 핸들

49

copyright © 2015 guileschool.com™



실습 : 메시지 큐(**10\_QUE**)

copyright © 2015 guileschool.com™



## 목 차

세마포어(**Semaphore**)

**FreeRTOS** 서비스 함수 사용시 주의 사항

뮤텍스(**Mutex**)

이벤트 플래그(**Event Flag**)

메시지큐(**Queue**)

교착상태(**Deadlock**)



copyright © 2015 guileschool.com™



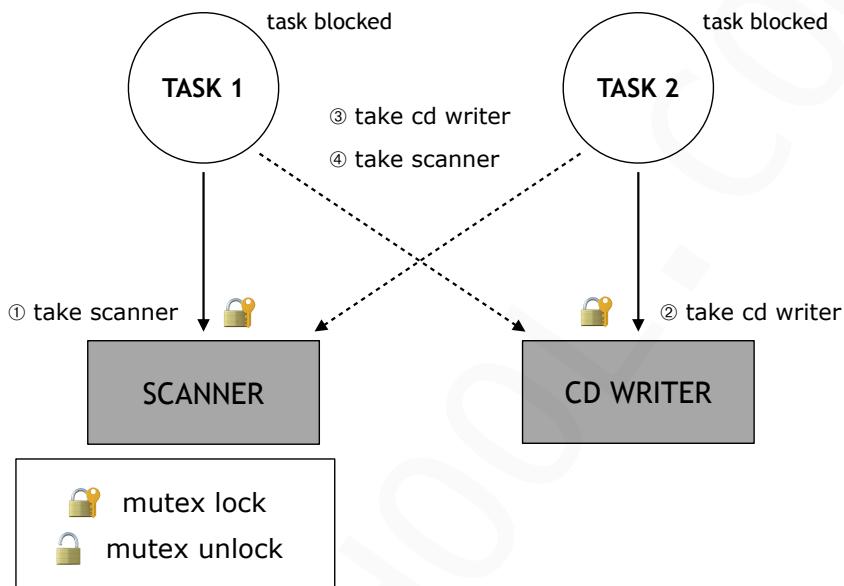
## 교착상태(Deadlock)





## 교착상태(Deadlock)의 사례

- 시나리오
  - ◆ A 프로세스는 실행을 위해 스캐너를 우선 요구하며 이후에 CD 레코더를 필요로 한다
  - ◆ B 프로세스는 CD 레코더를 먼저 사용하고 이후에 스캐너를 필요로 한다



53

copyright © 2015 guileschool.com™



## 교착상태(Deadlock)

- 두개의 태스크가 각자 다른 태스크에서 쓰고 있는 자원을 상호간에 무한정 기다리는 상태를 교착상태(**Deadlock**)라 한다
- 하나의 태스크에서 사용 할 수 있는 공유자원은 종류나 수가 많다
  - ◆ 예) 스캐너, 프린터, 테입드라이버, slots in process table, ...
- 운영체제는 공유자원의 배타적인 사용을 보장한다
  - ◆ Exclusive access.
- 그러나 하나의 태스크가 다수의 공유자원을 요구하는 경우도 있다. ---> 교착상태의 가능성 대두
- 교착상태가 발생하지 않도록 하는 가장 간단한 방법
  - ◆ 프로세스를 진행하기 전에 필요한 모든 자원을 획득
  - ◆ 순서대로 자원을 획득

54

copyright © 2015 guileschool.com™



## 연습 문제

---

- 바이너리 세마포어(**Semaphore**)에 대하여 설명하라
- 상호배제를 위하여 사용되는 세마포어와 뮤텍스의 가장 큰 차이점은?
- FreeRTOS** 서비스 함수 사용시 주의 사항 은?

---

copyright © 2015 guileschool.com™

---



## 질의 응답

---

copyright © 2015 guileschool.com™

---

# FreeRTOS 포팅

copyright © 2015 guileschool.com™



## 교육 목표

---

- **FreeRTOS**의 포팅 기법을 이해한다.
- 포팅 단계별 디버깅(테스트) 방법을 이해한다.

---

copyright © 2015 guileschool.com™



## 목 차

---



### 포팅(Porting)

포팅 과정

포팅 후 시험(Testing)

---

copyright © 2015 guileschool.com™



## 포팅(Porting) 이란?

---

- 실행 가능한 프로그램이 원래 설계된 바와 다른 컴퓨팅 환경(이를테면 CPU, 운영 체제, 서드 파티 라이브러리 등)에서 동작할 수 있도록 하는 과정
- 소스 하나로 여러 플랫폼에서 사용할 수 있는 것을 크로스 플랫폼이라고 함



## RTOS 포팅이란?

- 리얼 타임 커널을 마이크로 프로세서나 마이크로 컨트롤러에서 사용할 수 있도록 하는 작업
- 프로세서 의존적인 코드를 대상으로 작업

5

copyright © 2015 guileschool.com™



## 계속 진행하기전에 미리 알아둘 것

- **하나.** 기존 포트(하드웨어) 및 데모 어플리케이션 중 하나를 사용하는 경우라면 이 이후부터의 포팅을 설명하고 있는 페이지를 꼭 읽거나 이해할 필요는 없습니다
- **둘.** FreeRTOS를 완전히 다른 종류의 미지원 마이크로 컨트롤러로 이식하는 것은 쉬운 일이 아닙니다
- **셋.** 각 포트는 적당히 독특하며 사용되는 프로세서와 도구에 크게 의존하므로 앞으로 설명할 페이지들은 포팅(이식) 세부 사항에 대한 구체적인 정보를 제공 할 수는 없습니다. 그러나 다행이도 이미 많은 FreeRTOS 포트가 존재하며 이들 포트 예제들이 참조로 사용되는 것이 좋습니다
- **넷.** 동일한 프로세서 제품군 내에서 이식하는 것은 훨씬 간단한 작업입니다

6

copyright © 2015 guileschool.com™



## 포팅의 종류

- Kernel Porting
- File System Porting
- Network Porting
- USB Porting
- Bluetooth Porting
- GUI Porting
- CAN Bus Porting
- etc ...



7

copyright © 2015 guileschool.com™



## 포팅의 원리



This should be all that  
you have to do

copyright © 2015 guileschool.com™



## 포팅의 원리

---

- 10만 라인 규모의 TCP/IP 네트워크 소프트웨어를 우리가 직접 작성한다고 가정해 보자. 얼마만한 시간과 노력이 들까?
- 포팅은 바로 이러한 질문에 대한 답으로 나온 결과물인 것이다. 잘 작동하고 무엇보다 충분한 테스트(시험)을 통과한 누군가의 소프트웨어를 우리가 재 사용 할 수 만 있다면 좋을 것이다.
- 소스에 약간의 수정에 가해 우리가 원하는 하드웨어에서 훌륭히 동작하는 것을 볼 수만 있다면 누구나 이 행위(포팅)를 가치있는 일이라고 생각할 것이다.

9

copyright © 2015 guileschool.com™

---



## 포팅의 원리

---

- 많은 소프트웨어 개발자들은 소스코드의 재 활용에 관심을 가지고 있다. 자신이 만든 소프트웨어가 가급적 다양한 종류의 하드웨어에서 실행될 수 있다면 분명 좋은 일이다
- 이식성이 좋은 소프트웨어를 만들기 위해서는 해당 소스의 제작 초기부터 이를 염두에 두어야 한다.
- 또한 하드웨어 차이점에 따라, 달라지는 코드를 가급적 최소화 함으로써 코드의 간결, 유지 보수의 편리함을 도모하게 되었다.
- 예를 들자면 다음 그림과 같다

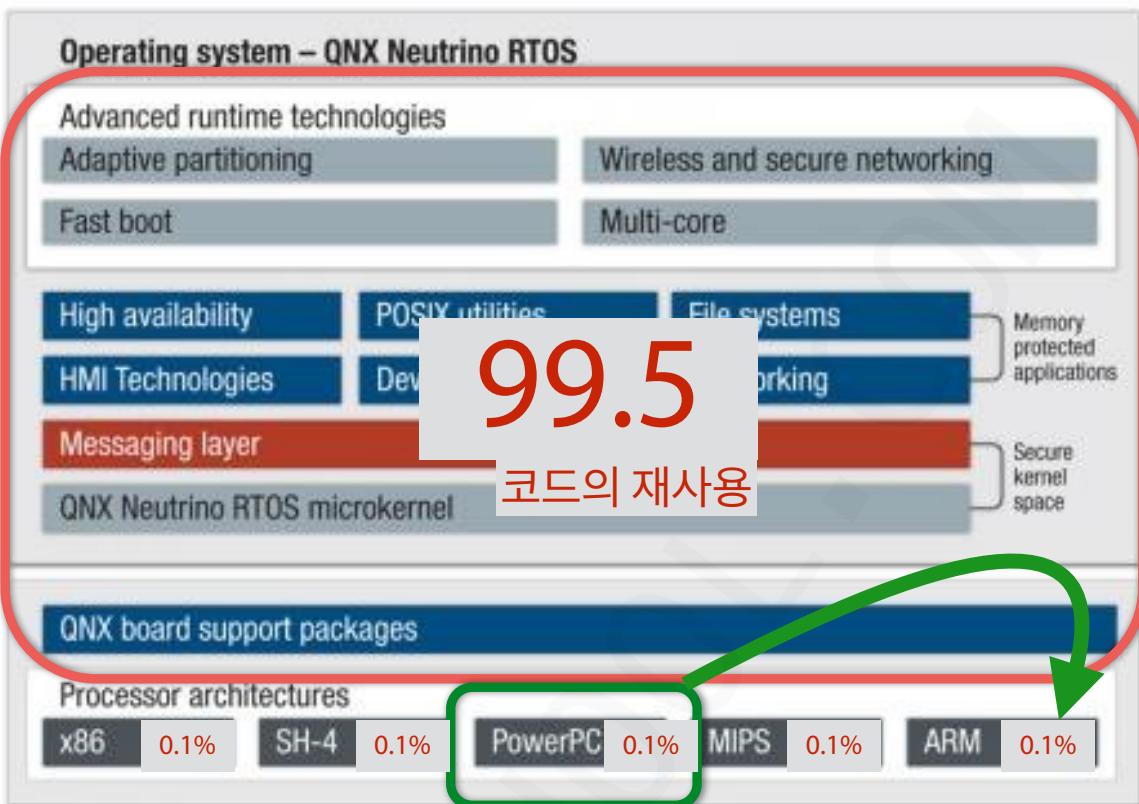
10

copyright © 2015 guileschool.com™

---



## 포팅의 원리



11

copyright © 2015 guileschool.com™



## RTOS 포팅이란?

- QNX (RTOS의 하나) 소프트웨어가 있다고 하자
- 이는 이식성이 좋은 소프트웨어이다.
- 45만 라인의 소프트웨어이며 99.5%(퍼센트)는 하드웨어와 무관한 코드이며 나머지 0.5% 만이 의존성 소스라고 가정하자.
- MIPS를 지원하는 소스가 있고 이를 ARM으로 포팅을 해야 한다. 과연 얼마의 노력과 시간이 발생할까?
- 이번엔 ARM 소스(CORTEX-A8)도 있다고 가정해 보자. 삼성전자의 CORTEX-A8 내장 S5PC100에 포팅한다고 하면 이때는 얼마의 노력과 시간이 발생할까?

12

copyright © 2015 guileschool.com™



## 포팅 단계(1/3)

사양, 포팅참조소스의 선례를 확인(가능성 타진)

차이점 분석(DIFF)

찾았다! 포팅의 시작( 포팅하려는 대상을 이미지 시각화 )

빌드후 실행(테스트)

다음 페이지 계속...

13

copyright © 2015 guileschool.com™



## 포팅 단계(2/3)

crt0.S + main() 부트코드 준비및 테스트

타이머 & 인터럽트 포팅

디렉토리 구조 생성 & 파일을 배치

크린빌드

다음 페이지 계속...

14

copyright © 2015 guileschool.com™



디버깅( using LED, printf, etc . . . )

커널 소스의 수정

커널의 스케줄러가 동작하는지 확인

기타(LCD, 파일시스템, 네트워크, USB, SD카드) 기능 구현

- 끝 -

15

copyright © 2015 guileschool.com™



## FreeRTOS 소스 코드 포팅 단계-1

- FreeRTOS 커널 소스 코드는 일반적으로 모든 포트에 공통적인 3 개의 소스 파일(코루틴이 사용되는 경우 4 개)과 RTOS 커널을 특정 아키텍처에 맞춘 하나 또는 두 개의 '포트'파일이 포함된다
- FreeRTOS 소스 코드의 최신 버전을 다운로드
- 파일을 편리한 위치에 압축을 풀고 디렉토리 구조를 유지 관리
- 우선 소스 코드 조직 및 디렉토리 구조에 익숙해질 필요가 있다
- [architecture] 포트에 대한 'port'파일을 포함 할 디렉토리를 만든다. 디렉토리의 형식은 FreeRTOS / Source / portable / [컴파일러 이름] / [프로세서 이름]이어야 함. 예를 들어, GCC 컴파일러를 사용하는 경우 기존 FreeRTOS / Source / portable / GCC 디렉토리에서 [architecture] 디렉토리를 만들 수 있다

16

copyright © 2015 guileschool.com™



## FreeRTOS 소스 코드 포팅 단계-2

- 빈 **port.c** 및 **portmacro.h** 파일을 방금 작성한 디렉토리에 복사 할것. 이 파일은 구현이 필요한 함수와 매크로의 스텝을 포함해야 함. 이러한 함수 및 매크로 목록은 기존 **port.c** 및 **portmacro.h** 파일을 참조할 것. 함수 및 매크로 본문을 단순히 삭제하여 이러한 기존 파일 중 하나에서 스텝 파일을 만들 수 있다
- 마이크로 컨트롤러의 스택이 상위 메모리에서 하위 메모리로 아래로 이동하도록 이식 된 경우 **portmacro.h**의 **portSTACK\_GROWTH**를 -1로 설정하고 그렇지 않으면 **portSTACK\_GROWTH**를 1로 설정한다
- [architecture] 포트에 대한 데모 응용 프로그램 파일을 포함 할 디렉토리를 만든다. 이것은 **FreeRTOS / Demo / [architecture\_compiler]** 또는 유사한 형식이어야 한다

17

copyright © 2015 guileschool.com™



## FreeRTOS 소스 코드 포팅 단계-3

- 기존 **FreeRTOSConfig.h** 및 **main.c** 파일을 방금 작성한 디렉토리에 복사 할것. 다시 스텝 파일로 편집해야한다
- **FreeRTOSConfig.h** 파일을 살펴볼 것. 여기에는 선택한 하드웨어에 대한 설정이 필요한 일부 매크로가 포함되어 있음
- 방금 작성한 디렉토리에서 디렉토리를 작성하고 **FreeRTOS / Demo / [architecture\_compiler] / ParTest** 를 호출할 것. 이 디렉토리에 **ParTest.c** 스텝 파일을 복사한다
- **ParTest.c**에는 다음과 같은 세 가지 간단한 함수가 포함되어 있다
- 몇 개의 LED를 깜박일 수있는 GPIO를 설정하고, 특정 LED를 설정 또는 해제합니다. LED의 상태를 토글한다

18

copyright © 2015 guileschool.com™



## FreeRTOS 소스 코드 포팅 단계-4

- 프로젝트 만들기
- 프로젝트를 성공적으로 빌드 하기위해 프로젝트파일 (또는 makefile)를 생성하는게 필요하다.
- 프로젝트에는 다음 파일이 있어야한다

다음 페이지 계속...

19

copyright © 2015 guileschool.com™



## FreeRTOS 소스 코드 주요 구성

- Source/tasks.c <-- 커널 소스 디렉토리(**src**)
- Source/Queue.c
- Source/List.c
- Source/portable/[compiler name]/[processor name]/port.c <-- 포팅 소스  
Source/portable/MemMang/heap\_1.c (or heap\_2.c or heap\_3.c or heap\_4.c)  
<-- **heap\_x** 파일중 택1
- Demo/[processor name]/main.c <-- 메인(**main**)함수가 포함된 소스
- Demo/[Processor name]/ParTest/ParTest.c <-- **LED** 제어 소스

다음 디렉토리는 include 경로에 있어야한다. Demo/[Process name] 디렉토리 기준의 상대 경로를 사용할것(절대 경로가 아님)

- Demo/Common (i.e. ../Common)
- Demo/[Processor Name]
- Source/include
- Source/portable/[Compiler name]/[Processor name]

20

copyright © 2015 guileschool.com™



- 스텁 구현
- 이제 어려운 문제. 프로젝트가 포터블 레이어를 컴파일하면 스텁에는 구현이 필요하다. `pxPortInitialiseStack()`이 구현해야 할 첫 번째 함수이다. `pxPortInitialiseStack()`을 구현하려면 먼저 아키텍처에 따라 매우 달라지는 작업 컨텍스트 스택 프레임 구조를 결정해야한다

21

copyright © 2015 guileschool.com™

---



## 연습 문제

---

- 포팅(**Porting**) 을 위해 필요한 지식은?
- 포팅과정을 단계 별로 설명하라

copyright © 2015 guileschool.com™

---



---

## 질의 응답

---

copyright © 2015 guileschool.com™

---

실습 :

실습 응용 프로젝트

# Contents

1.	태스크 생성 및 운용( <i>01_TASKMAN</i> ).....	5
1.1.	소스 트리 .....	5
1.2.	TODO #1 .....	5
1.3.	TODO #2 .....	7
1.4.	TODO #3 .....	9
1.5.	TODO #4 .....	11
2.	스택오버플로우 겸사( <i>02_STACKOVERFLOW</i> ).....	12
2.1.	소스 트리 .....	12
2.2.	TODO #1 .....	12
2.3.	TODO #2 .....	13
2.4.	TODO #3 .....	14
2.5.	TODO #4 .....	15
3.	임계영역 보호( <i>03_CRITICAL</i> ).....	16
3.1.	소스 트리 .....	16
3.2.	TODO #1 .....	16
3.4.	TODO #2 .....	19
4.	코루틴( <i>04_cROUTINE</i> ).....	21
4.1.	소스 트리 .....	21
4.2.	TODO #1 .....	21
4.3.	TODO #2 .....	22
5.	세마포어( <i>05_SEM</i> ) .....	23
5.1.	소스 트리 .....	23
5.2.	TODO #1 .....	23

5.3.	TODO #2 .....	25
5.4.	TODO #3 .....	27
6.	카운팅 세마포어( <i>06_COUNT_SEM</i> ).....	29
6.1.	소스 트리 .....	29
6.2.	TODO #1 .....	29
6.3.	TODO #2 .....	32
7.	상호배제의 도구( <i>07_MUTEX</i> ).....	34
7.1.	소스 트리 .....	34
7.2.	TODO #1 .....	34
7.3.	TODO #2 .....	36
8.	지연 인터럽트( <i>08_DEFERRED_INTERRUPT</i> ) .....	38
8.1.	소스 트리 .....	38
8.2.	TODO #1 .....	38
8.3.	TODO #2 .....	40
8.9.	TODO #3 .....	42
9.	이벤트 플래그( <i>09_FLAG</i> ) .....	43
9.1.	소스 트리 .....	43
9.2.	TODO #1 .....	43
9.3.	TODO #2 .....	45
10.	소프트웨어 타이머( <i>10_SOFT_TIMER</i> ).....	47
10.1.	소스 트리 .....	47
10.2.	TODO #1 .....	47
10.3.	TODO #2 .....	48
11.	메시지 큐( <i>11_QUE</i> ).....	50

11.1.	소스 트리	.....
		50
11.2.	TODO #1	.....
		50
11.4.	TODO #2	.....
		52

## 1. 태스크 생성 및 운용(01\_TASKMAN)

FreeRTOS 의 커널 서비스인 태스크 운용 함수들을 사용하여 직접 태스크를 생성및 실행 해 보며 멀티태스킹의 의미를 알아보자.

### 1.1. 소스 트리

“.app\01\_TASKMAN” 디렉토리에 실습할 파일이 있다

### 1.2. TODO #1

FreeRTOS의 커널 서비스인 태스크 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 태스크 생성 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
// TASK CREATE
/* TODO #1:
   Task1을 생성
   use 'xTaskCreate' */

#endif // TODO #1
```

태스크 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 “Task1 is running” 메시지가 보인다면 성공



### 1.3. TODO #2

FreeRTOS의 커널 서비스인 태스크 실행 중단 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 태스크 실행 중단 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

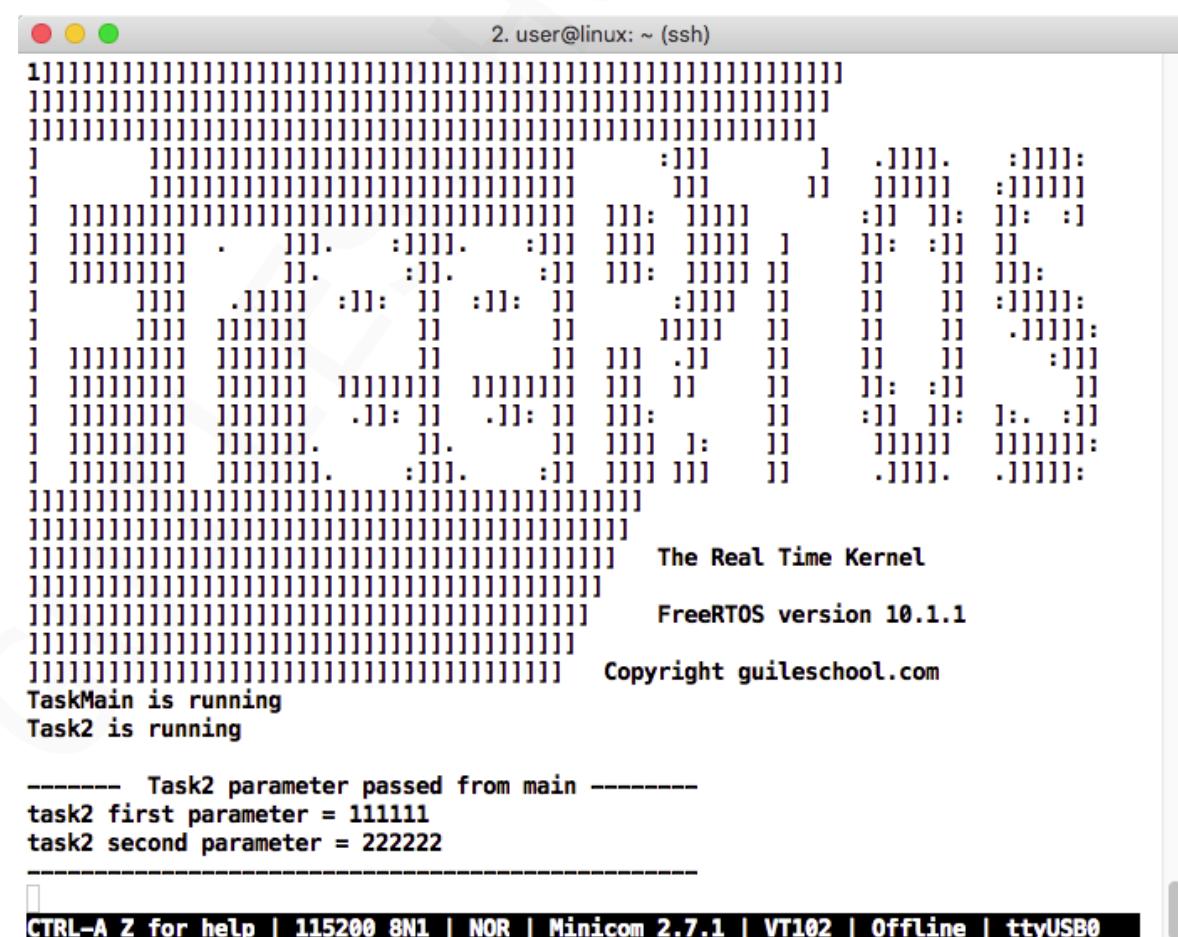
"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2:  
   Task1을 중지  
   use 'vTaskSuspend' */  
#if 1  
  
#endif // TODO #2
```

태스크 동작 중지 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 "Task2 is running" 메시지가 보인다면 성공



The terminal window displays the FreeRTOS kernel boot sequence and the execution of two tasks. The output includes:

- Kernel boot messages: "The Real Time Kernel", "FreeRTOS version 10.1.1", and "Copyright guileschool.com".
- Task initialization: "TaskMain is running" and "Task2 is running".
- Task parameter output: "----- Task2 parameter passed from main -----", "task2 first parameter = 111111", and "task2 second parameter = 222222".
- Terminal status bar at the bottom: "CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0".

❖ 여기서 잠깐. 태스크1 은 왜 출력 되지 않았을까?

## 1.4. TODO #3

FreeRTOS의 커널 서비스인 태스크 관리 함수를 이용한 실습 예이다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```

태스크 1 중에서 . . .
/* TODO #3:
   코드를 실행 하여 보고
   vTaskDelay() 코드를 주석 처리한 후 그 결과를 설명한다 */
#ifndef // No comment
.....
#endif // TODO #3

태스크 2 중에서 . . .
/* TODO #3:
   코드를 실행 하여 보고
   vTaskDelay() 코드를 주석 처리한 후 그 결과를 설명한다 */
#ifndef // No comment
.....
#endif // TODO #3

```

코드를 동작 시켜 본다

```

2. user@linux: ~ (ssh)
The Real Time Kernel
FreeRTOS version 10.1.1
Copyright guileschool.com

TaskMain is running
Task2 is running

----- Task2 parameter passed from main -----
task2 first parameter = 111111
task2 second parameter = 222222
----- bbbbbb
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0

```

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 “b b b b . . .” 문자가 반복적으로 출력되어 보인다면 성공

vTaskDelay() 함수를 주석 처리하고 다시 동작 시켜 보아 그 차이를 확인한다

❖ 여기서 잠깐. 태스크1 의 문자 ‘a’ 은 왜 출력 되지 않았을까?

## 1.5. TODO #4

FreeRTOS의 커널 서비스인 **태스크 우선 순위 변경** 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 태스크 우선 순위 변경 함수의 사용법을 확인한 후 코드를 완성하면 된다.

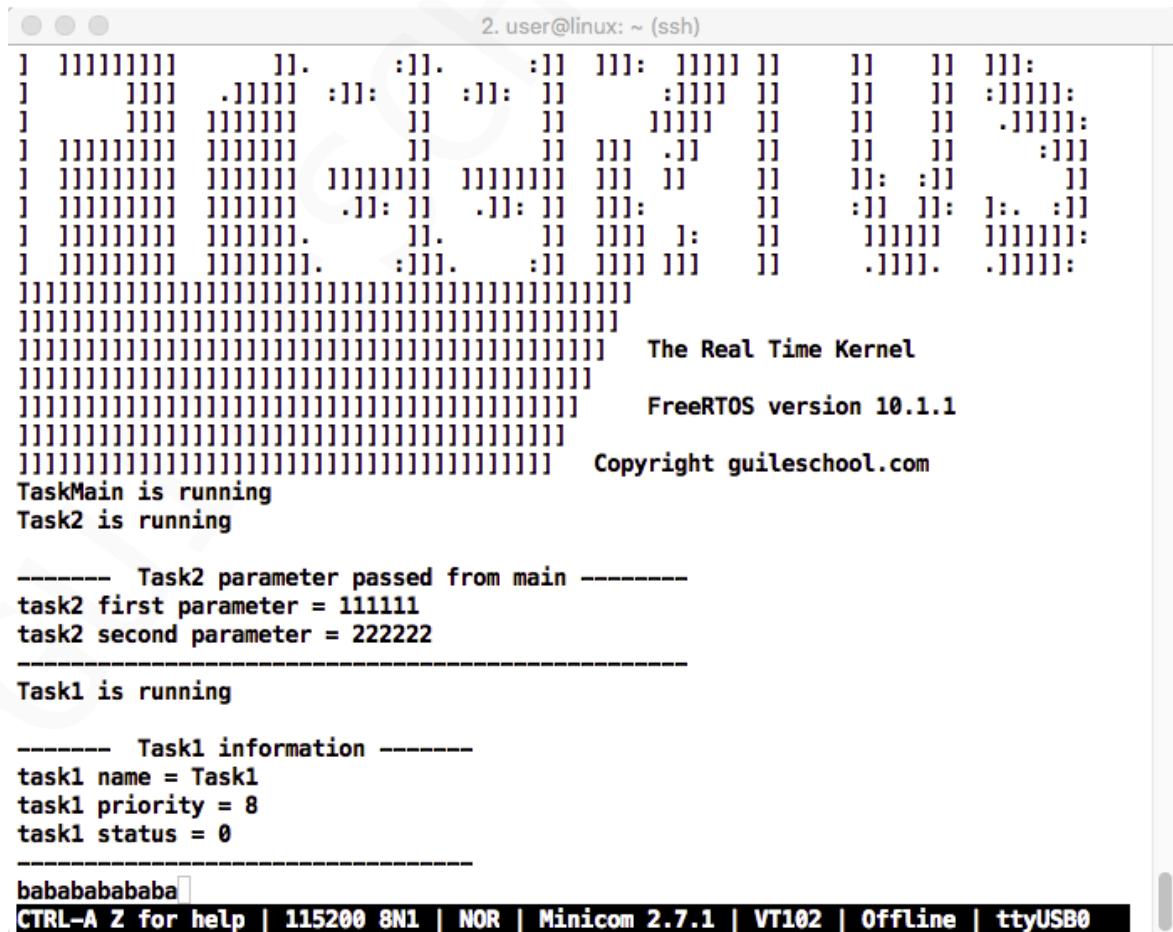
"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #4:  
   Task1의 우선 순위를 'TASK_3_PRIO' 으로 변경  
   use 'vTaskPrioritySet' and 'vTaskResume' */  
#if 1  
  
#endif // TODO #4
```

태스크 우선 순위 변경 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널창에 그림처럼 "ba ba ba ..." 문자가 반복적으로 출력되어 보인다면 성공



The terminal window shows the following output:

```
2. user@linux: ~ (ssh)  
[...]  
The Real Time Kernel  
FreeRTOS version 10.1.1  
Copyright guileschool.com  
TaskMain is running  
Task2 is running  
----- Task2 parameter passed from main -----  
task2 first parameter = 111111  
task2 second parameter = 222222  
-----  
Task1 is running  
----- Task1 information -----  
task1 name = Task1  
task1 priority = 8  
task1 status = 0  
-----  
babababababa  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```



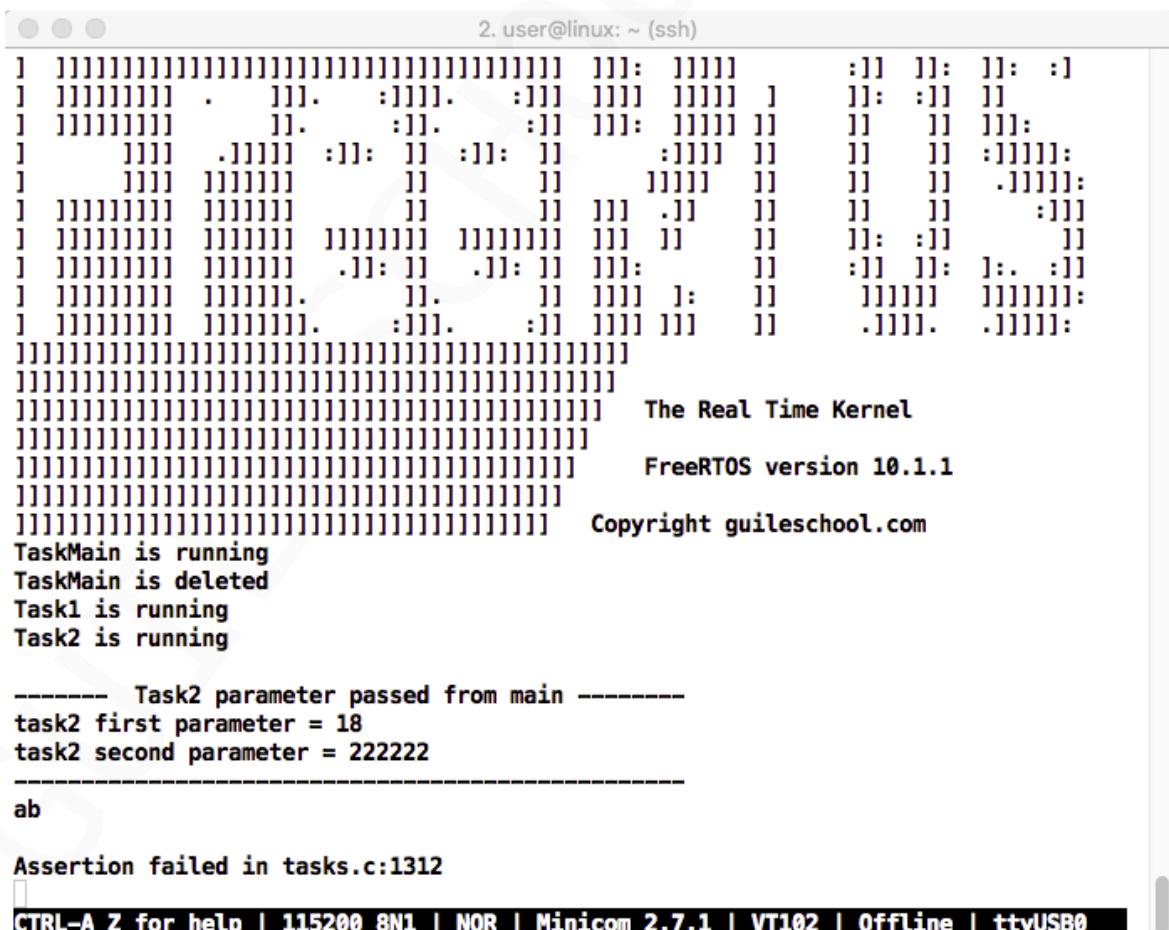
### 2.3. TODO #2

FreeRTOS의 커널 서비스인 **HOOK** 함수를 이용한 실습 예이다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2:  
 * 스택에 다음처럼 데이터를 넣어본다  
 * eg. memset(bbb, 'a', 4096);  
 */  
#if 1  
....  
#endif
```

태스크 스택메모리에 이미 사용 가능한 범위를 넘는 데이터 배열(4K 크기)을 선언한 상태이다.  
더구나, 이번에는 이 스택메모리에 데이터를 넣는 작업을 시도하였다. 화면 출력 상에 특이점이  
보이는가?



The terminal window displays the following output:

```
2. user@linux: ~ (ssh)  
[...] The Real Time Kernel  
[...] FreeRTOS version 10.1.1  
[...] Copyright guileschool.com  
TaskMain is running  
TaskMain is deleted  
Task1 is running  
Task2 is running  
----- Task2 parameter passed from main -----  
task2 first parameter = 18  
task2 second parameter = 222222  
-----  
ab  
Assertion failed in tasks.c:1312  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

## 2.4. TODO #3

FreeRTOS의 커널 서비스인 훅(HOOK) 함수를 이용한 실습 예이다.

<FreeRTOSConfig.h> 파일내의 상수를 다음처럼 변경해 준다

```
/* TODO #3:  
 * 스택 검사 옵션을 다음과 같이 설정한다  
 *#define configCHECK_FOR_STACK_OVERFLOW 1  
 */  
#ifdef configCHECK_FOR_STACK_OVERFLOW  
#undef configCHECK_FOR_STACK_OVERFLOW  
#define configCHECK_FOR_STACK_OVERFLOW 1  
#endif
```

컴파일하고 실행하여 어떤 결과가 출력 되는지 확인한다

TODO #2 의 실행 결과 화면과 비교해 보자! 다른 점은?

```
2. user@linux: ~ (ssh)  
[...] The Real Time Kernel  
[...] FreeRTOS version 10.1.1  
[...] Copyright guileschool.com  
TaskMain is running  
TaskMain is deleted  
Task1 is running  
Task2 is running  
----- Task2 parameter passed from main -----  
task2 first parameter = 18  
task2 second parameter = 222222  
-----  
Assertion failed in main.c:172  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

## 2.3. TODO #4

FreeRTOS의 커널 서비스인 **HOOK** 함수를 이용한 실습 예이다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2 #4:  
   스택에 다음처럼 데이터를 넣어본다  
   eg. memset(bbb, 'a', 4096); // 4096 외 다른 값(1, 8, 1024, ...)도 시도해 본다  
*/  
#if 1  
memset(bbb, 'a', 4096); // 4096 외 다른 값(1, 8, 1024, ...)도 시도해 본다  
#endif
```

<FreeRTOSConfig.h> 파일내의 상수는 원래(디폴트) 값으로 다음처럼 변경해 준다

```
/* TODO #3:  
   스택 검사 옵션을 다음과 같이 설정한다  
   #define configCHECK_FOR_STACK_OVERFLOW      1  
*/  
#ifdef configCHECK_FOR_STACK_OVERFLOW  
#undef configCHECK_FOR_STACK_OVERFLOW  
#define configCHECK_FOR_STACK_OVERFLOW      0  
#endif
```

스택검사 기능 비활성 그리고, 스택배열에 4096 외 다른 값(1, 8, 1024,...)도 적용해 본다.

각각의 수정 전과 수정 후 화면 출력 결과에 차이점이 있는가?

👉 만약 실행 결과마다 차이가 있었다면 그것을 어떻게 설명할 수 있을까?

### 3.

## 임계영역 보호(03\_CRITICAL)

FreeRTOS 의 커널 서비스인 task???\_CRITICAL 함수를 사용하여 임계영역의 공유자원 을 보호 하는 방법을 구현 해 보고 이의 의미를 알아보자.

가상의 비행기 티켓 무인 발급 시스템을 가정하고 만들어진 실습이다.

비행기 티켓수(tickets) 자료를 2개의 태스크가 공유하여 작동시 발생되는 문제점과 그의 해결 방법을 고민 해 본다.

### 3.1. 소스 트리

“.app\03\_CRITICAL” 디렉토리에 실습할 파일이 있다

### 3.2. TODO #1

이후의 TODO#2 실습을 위한 사전 준비이다

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
   초기 티켓수를 작은수(10,000 ~ 100,000정도) 입력 하여 테스트 한후  
   그 결과를 설명한다 */  
#if 1 // No comment  
    tickets= backupTickets = 100000; // MAX value( 10 million )  
#endif // TODO #1
```

Tickets 변수에 초기 값으로 10,000 ~ 100,000 정도를 입력한다.

컴파일하고 실행하여 어떤 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 “Good!. Matched” 메시지가 보인다면 성공





### 3.4. TODO #2

공유 자원(tickets)의 보호를 위하여 FreeRTOS 의 커널 서비스 task???\_CRITICAL 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, task???\_CRITICAL 함수의 사용법을 확인한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
#if 1 // No comment
    tickets= backupTickets = 10 * TICKET_1MILLION; // MAX value( 1 million )
#endif // TODO 2

/* TODO #2:
   taskENTER_CRITICAL() 과 taskEXIT_CRITICAL()을 이용하여
   공유변수(tickets)를 보호한다 */
#if 1
// CRITICAL SECTION(ENTER)
?????????????????????????????????
#endif // TODO #2

    tickets --;      // ticket count

#if 1
// CRITICAL SECTION(EXIT)
?????????????????????????????????
#endif // TODO #2
```

Tickets 변수에 초기 값으로 적당히 큰 값 TICKET\_1MILLION 이나 10 x TICKET\_1MILLION 정도를 입력한다.

컴파일하고 실행하여 어떤 결과가 출력 되는지 확인한다



## 4.

## 코루틴(04\_coROUTINE)

FreeRTOS에서 제공하는 코루틴 함수를 직접 구현 및 실행해 보고 그 기능의 유용성과 활용 범위를 생각해 본다.

### 4.1. 소스 트리

".app\04\_coROUTINE" 디렉토리에 실습할 파일이 있다

### 4.2. TODO #1

FreeRTOS의 커널 서비스인 코루틴 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 코루틴 함수의 사용법을 확인한 후 코드를 완성하면 된다.

"#if 0"를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
   코루틴 스케줄러 실행  
   vCoRoutineSchedule */  
#if 1  
    vCoRoutineSchedule ();  
#endif // TODO #1
```

LED가 점멸하는가? 만약 그렇다면 성공 😊

### 4.3. TODO #2

FreeRTOS의 커널 서비스인 코루틴 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 코루틴 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2:  
   두번째 코루틴 함수 생성 */  
#if 1  
    . . . . .  
#endif // TODO #2
```

첫번째와 두번째 LED을 동시에 점멸하기 위한 코루틴 함수를 기존의 코드에 추가한다

컴파일하고 실행하여 두 개의 LED 가 동시에 점멸(on/off) 한다면 성공 😊

## 5.

## 세마포어(05\_SEM)

FreeRTOS 의 커널 서비스인 세마포어 함수들을 사용하여 태스크 동기화를 구현하고 그 특징을 이해 한다.

### 5.1. 소스 트리

“.app\05\_SEM” 디렉토리에 실습할 파일이 있다

### 5.2. TODO #1

FreeRTOS의 커널 서비스인 **세마포어** 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 세마포어 생성 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
     create a binary semaphore  
     use sem_id */  
#if 1  
    ....  
#endif // TODO #1
```

세마포어 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 별도의 에러(ERROR) 메시지가 보이지 않는다면 성공



### 5.3. TODO #2

FreeRTOS의 커널 서비스인 **세마포어** 전달/획득 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 세마포어 전달/획득 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

```
/* TODO #2:  
   post to a semaphore */  
#if 1  
....  
#endif // TODO #2
```

세마포어 전달/획득 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다



## 5.4. TODO #3

FreeRTOS의 커널 서비스인 **세마포어** 삭제 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 세마포어 삭제 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #3:  
   delete a semaphore */  
#if 1  
vSemaphoreDelete(sem_id);  
#endif // TODO #3
```

```
/* TODO #3:  
   check if semaphore is available */  
#if 1  
if (err != pdPASS) printf("xSemaphoreGive error(%d) found\n", (int)err);  
#endif // TODO #2
```

세마포어 삭제 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 에러(ERROR) 메시지가 출력 된다면 성공

## 6. 카운팅 세마포어(06\_COUNT\_SEM)

FreeRTOS 의 커널 서비스인 카운팅 세마포어 함수를 사용하여 태스크 동기화를 구현하고 그 특징을 이해 한다.

### 6.1. 소스 트리

“.app\06\_COUNT\_SEM” 디렉토리에 실습할 파일이 있다

### 6.2. TODO #1

FreeRTOS의 커널 서비스인 **바이너리 세마포어** 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 바이너리 세마포어 생성 함수의 사용법을 확인한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
     create a binary semaphore  
     use sem_id */  
#if 1  
    sem1=xSemaphoreCreateBinary();  
    //      sem1 = xSemaphoreCreateCounting( 10, 0 );  
    if (sem1 == NULL) printf("xSemaphoreCreateBinary error found\n");  
#endif // TODO #1
```

바이너리 세마포어 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다





### 6.3. TODO #2

FreeRTOS의 커널 서비스인 **카운팅 세마포어** 전달/획득 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 카운팅 세마포어 전달/획득 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

TODO#1 에서의 코드를 약간만 수정한다

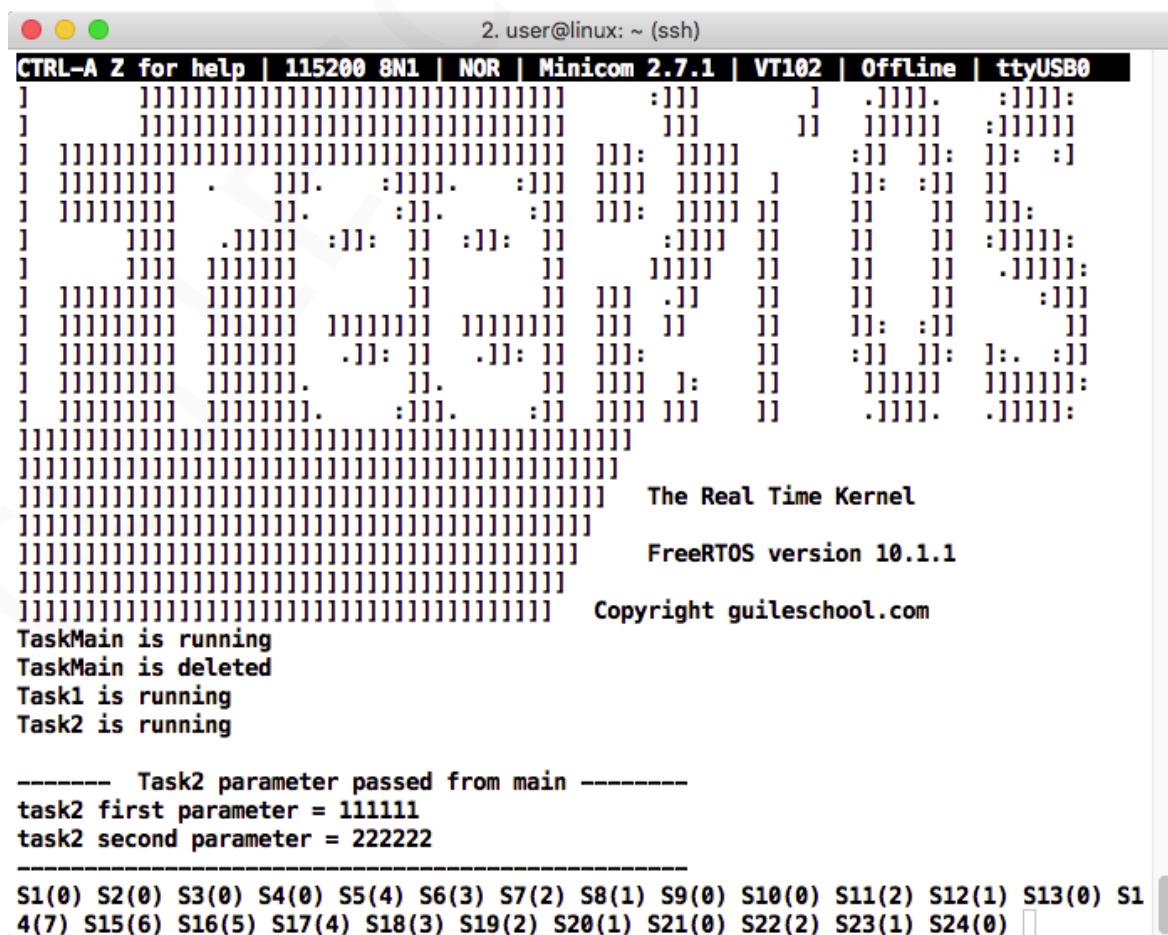
```
/* TODO #1 #2:  
     create a binary semaphore  
     use sem_id */  
#if 1  
//      sem1=xSemaphoreCreateBinary();  
sem1 = xSemaphoreCreateCounting( 10, 0 );  
if (sem1 == NULL) printf("xSemaphoreCreateBinary error found\n");  
#endif // TODO #1
```

세마포어 전달/획득 함수를 구현한다

테스트 방법은 TODO#1 에서와 같다. 버튼을 누르는데 천천히도 눌러보고 빠르게도 눌러본다.

그리고, TODO#1 에서의 경험과 비교해 본다.

어떤 차이가 있는가?



The screenshot shows a terminal window titled "2. user@linux: ~ (ssh)". The window displays the FreeRTOS kernel version 10.1.1 running on a Real Time Kernel. The output includes:

- Kernel configuration: CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttUSB0
- System status: The Real Time Kernel, FreeRTOS version 10.1.1, Copyright guileschool.com
- Task execution: TaskMain is running, TaskMain is deleted, Task1 is running, Task2 is running
- Task2 parameters: ----- Task2 parameter passed from main -----, task2 first parameter = 111111, task2 second parameter = 222222
- System state: S1(0) S2(0) S3(0) S4(0) S5(4) S6(3) S7(2) S8(1) S9(0) S10(0) S11(2) S12(1) S13(0) S14(7) S15(6) S16(5) S17(4) S18(3) S19(2) S20(1) S21(0) S22(2) S23(1) S24(0)

빠르게, 20회 이상도 눌러보자.  
결과는 예상대로 나오는가?  
혹시 결과가 아래 터미널 창 출력과 비슷하지 않는가?  
버튼을 누른 횟수만큼 출력되지 않는다. 왜 그럴까?

```
2. user@linux: ~ (ssh)
[...]
The Real Time Kernel
FreeRTOS version 10.1.1
Copyright guileschool.com
TaskMain is running
TaskMain is deleted
Task1 is running
Task2 is running
----- Task2 parameter passed from main -----
task2 first parameter = 111111
task2 second parameter = 222222
-----
S1(9) S2(8) S3(7) S4(6) S5(5) S6(4) S7(3) S8(2) S9(1) S10(0) □
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

```
/* TODO #1 #2:
     create a binary semaphore
     use sem_id */
#if 1
//      sem1=xSemaphoreCreateBinary();
sem1 = xSemaphoreCreateCounting( 10, 0 );
if (sem1 == NULL) printf("xSemaphoreCreateBinary error found\n");
#endif // TODO #1
```

## 7.

## 상호배제의 도구(07\_MUTEX)

FreeRTOS 의 커널 서비스인 뮤텍스 함수들을 사용하여 공유 자원을 보호하는데 이용해 보고 세마포어나 task??\_CRITICAL 와의 차이점등을 비교 해 본다

### 7.1. 소스 트리

“.app\07\_MUTEX” 디렉토리에 실습할 파일이 있다

### 7.2. TODO #1

FreeRTOS의 커널 서비스인 **뮤텍스** 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 뮤텍스 생성 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
     create a mutex  
     use mutex_id */  
#if 1  
    . . . . .  
    if (mutex_id == NULL) printf("xSemaphoreCreateMutex error found\n");  
#endif // TODO #1
```

뮤텍스 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 “Not good. ...” 보인다면 성공

```
2. user@linux: ~ (ssh)
[...]
The Real Time Kernel
FreeRTOS version 10.1.1
Copyright guileschool.com
TaskMain is running
TaskMain is deleted
Task1 is running
Task2 is running
----- Task2 parameter passed from main -----
task2 first parameter = 111111
task2 second parameter = 222222
-----
[TASK2]Total Tickets = 10000110
[TASK1]COUNTER up to 971
[TASK2]COUNTER up to 9999139
Not good. expectation(10000000)
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

### 7.3. TODO #2

FreeRTOS의 커널 서비스인 **뮤텍스** 대기/전달 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 뮤텍스 대기/전달 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2:  
   MUTEX 을 이용하여  
   공유변수(tickets)를 보호한다 */  
#if 0  
    // CRITICAL SECTION(ENTER)  
    ???????????  
#endif // TODO #2  
  
    tickets --;      // ticket count  
  
#if 0  
    // CRITICAL SECTION(EXIT)  
    ???????????  
#endif // TODO #2
```

뮤텍스 대기/전달 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다



## 8. 지연 인터럽트(08\_DEFERRED\_INTERRUPT)

FreeRTOS 의 커널 서비스를 이용하여 지연된 인터럽트(deferred interrupt) 를 구현하고 그 특징을 이해 한다.

### 8.1. 소스 트리

“.app\08\_DEFERRED\_INTERRUPT” 디렉토리에 실습할 파일이 있다

### 8.2. TODO #1

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
상수 선언 __MY_ENABLE_DEFERRED_INTERRUPT 을 비활성화 한다 */  
  
#define __MY_ENABLE_DEFERRED_INTERRUPT 0
```

일반적인 인터럽트를 이용한 예제를 우선 구현해 본다.

실습보드 버튼을 누를때마다,

// 시간이 많이 소요되는 특징으로써 정의한 함수  
void heavyCopyLoader(void) <----- main.c 소스코드 참조

위 함수가 실행된다. 이 함수는 대략 수십메가 바이트 이상의 데이터를 다루는 메모리 관련 함수이다.



### 8.3. TODO #2

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

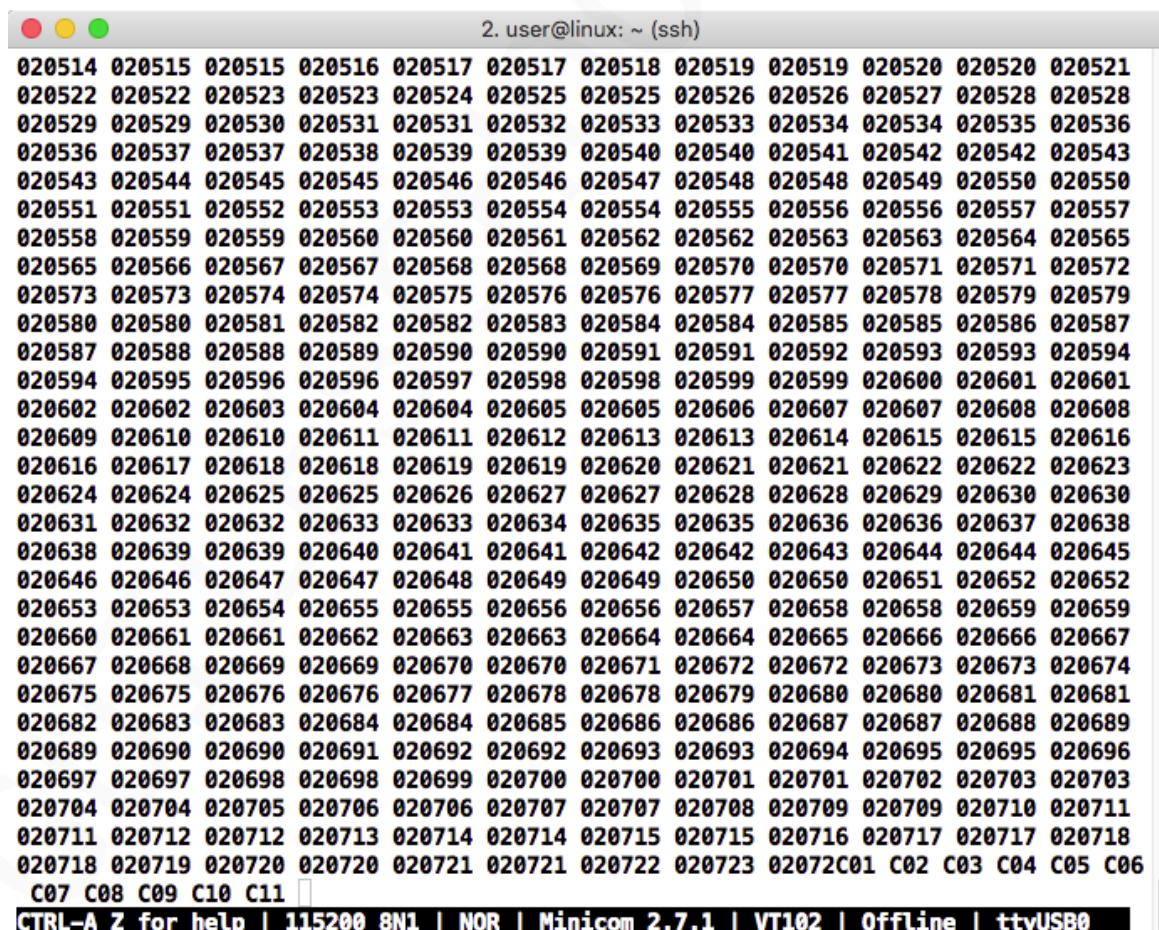
```
/* TODO #2:  
   터미널 화면에 현재 시간 정보(Tick변수값)을 출력 */  
#if 1 // No comment  
serial_printf("%06d ", xTaskGetTickCount()); // 현재의 시간(xTickCount) 출력  
#endif // TODO #2
```

현재 시간 정보라고 할 수 있는 xTickCount(커널 틱변수) 값이 화면을 가득 채울 것이다.

대략 xTickCount 값 1000 이 1초에 해당한다고 보면 된다.

이제부터 버튼을 빠르게 20회 정도 눌러보자.

자. 어떤 문제가 발생할까?



The screenshot shows a terminal window titled "2. user@linux: ~ (ssh)". The window displays a continuous stream of tick counts, starting from 020514 and increasing sequentially. The output is as follows:

```
020514 020515 020515 020516 020517 020517 020518 020519 020519 020520 020520 020521  
020522 020522 020523 020523 020524 020525 020525 020526 020526 020527 020528 020528  
020529 020529 020530 020531 020531 020532 020532 020533 020533 020534 020534 020535 020536  
020536 020537 020537 020538 020539 020539 020540 020540 020541 020541 020542 020542 020543  
020543 020544 020545 020545 020546 020546 020547 020547 020548 020548 020549 020550 020550  
020551 020551 020552 020553 020553 020554 020554 020555 020555 020556 020556 020557 020557  
020558 020559 020559 020560 020560 020561 020562 020562 020563 020563 020564 020565  
020565 020566 020567 020567 020568 020568 020569 020569 020570 020570 020571 020571 020572  
020573 020573 020574 020574 020575 020576 020576 020577 020577 020578 020578 020579 020579  
020580 020580 020581 020582 020582 020583 020584 020584 020585 020585 020586 020587  
020587 020588 020588 020589 020590 020590 020591 020591 020592 020593 020593 020594  
020594 020595 020596 020596 020597 020598 020598 020599 020599 020600 020601 020601  
020602 020602 020603 020604 020604 020605 020605 020606 020607 020607 020608 020608  
020609 020610 020610 020611 020611 020612 020613 020613 020614 020615 020615 020616  
020616 020617 020618 020618 020619 020619 020620 020621 020621 020622 020622 020623  
020624 020624 020625 020625 020626 020627 020627 020628 020628 020629 020630 020630  
020631 020632 020632 020633 020633 020634 020635 020635 020636 020636 020637 020638  
020638 020639 020639 020640 020641 020641 020642 020642 020643 020644 020644 020645  
020646 020646 020647 020647 020648 020649 020649 020650 020650 020651 020652 020652  
020653 020653 020654 020655 020655 020656 020656 020657 020658 020658 020659 020659  
020660 020661 020661 020662 020663 020663 020664 020664 020665 020666 020666 020667  
020667 020668 020669 020669 020670 020670 020671 020672 020672 020673 020673 020674  
020675 020675 020676 020676 020677 020678 020678 020679 020680 020680 020681 020681  
020682 020683 020683 020684 020684 020685 020686 020686 020687 020687 020688 020689  
020689 020690 020690 020691 020692 020692 020693 020693 020694 020695 020695 020696  
020697 020697 020698 020698 020699 020699 020700 020700 020701 020701 020702 020703 020703  
020704 020704 020705 020706 020706 020707 020707 020708 020708 020709 020709 020710 020711  
020711 020712 020712 020713 020714 020714 020715 020715 020716 020717 020717 020717 020718  
020718 020719 020720 020720 020721 020721 020722 020723 02072C01 C02 C03 C04 C05 C06  
C07 C08 C09 C10 C11
```

At the bottom of the terminal window, there is a status bar with the following text: "CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0".

자. 어떤 문제가 발생할까?

버튼키 인터럽트가 실행되는 동안 시간이 멈추는 극단적인 상황이 발생하게 된다.

- 버튼키 누르기 전 : 095394

- 버튼키 누르기 후 : 095428

타이머 인터럽트는 버튼키를 빠르게 누르는 동안 거의 실행이 안되었음

이 문제는 과연 왜 발생한 걸까?

힌트. void heavyCopyLoader(void) <----- main.c 소스코드 참조

그럼 다음 TODO#3에서 이 문제를 해결해 보자

```
2. user@linux: ~ (ssl)
95318 095318 095319 095320 095320 095321
95325 095326 095326 095327 095327 095328
95332 095333 095334 095334 095335 095335
95340 095340 095341 095341 095342 095343
95347 095347 095348 095349 095349 095350
95354 095355 095355 095356 095357 095357
95361 095362 095363 095363 095364 095364
95369 095369 095370 095371 095371 095372
95376 095377 095377 095378 095378 095379
95383 095384 095385 095385 095386 095386
95391 095391 095392 095392 095393 095394
C31 C32 C33 C34 C35 C36 C37 C38 C39 5396
95414 095414 095415 095415 095416 095417
95421 095421 095422 095423 095423 095424
95428 095429 095429 095430 095431 095431
95435 095436 095437 095437 095438 095438
95443 095443 095444 095445 095445 095446
95450 095451 095451 095452 095452 095453
95457 095458 095459 095459 095460 095460
95465 095465 095466 095466 095467 095468
95472 095473 095473 095474 095474 095475
95479 095480 095480 095481 095482 095482
95486 095487 095488 095488 095489 095490
95494 095494 095495 095496 095496 095497
005501 005502 005502 005502 005502 005501
```

## 8.9. TODO #3

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1 #3:  
상수 선언 __MY_ENABLE_DEFERRED_INTERRUPT 을 활성화 한다 */  
  
#define __MY_ENABLE_DEFERRED_INTERRUPT 1
```

Deferred 인터럽트를 이용한 예제를 구현해 본다.

과연 문제가 해결되었을까? 다음 터미널 결과 화면을 보자  
버튼키 인터럽트 중에도 시간 처리는 되고 있었다는 확실한 증거(?)이다  
- 버튼키 누르기 전 : 027385  
- 버튼키 누르기 후 : 031124

타이머 인터럽트는 버튼키를 빠르게 누르는 동안에도 실행이 잘 되었음

```
2. user@linux: ~ (ssh)  
027323 027323 027324 027325 027325 027326 027326 0  
027330 027331 027331 027332 027332 027333 027334 0  
027337 027338 027339 027339 027340 027340 027341 0  
027345 027345 027346 027346 027347 027348 027348 0  
027352 027353 027353 027354 027354 027355 027356 0  
027359 027360 027360 027361 027362 027362 027363 0  
027367 027367 027368 027368 027369 027370 027370 0  
027374 027374 027375 027376 027376 027377 027378 0  
027381 027382 027382 027383 027384 027384 027385 0  
027388 027389 027390 027390 027391 027392 027392 0  
1 C13 N1 C14 N1 C15 N2 C16 C17 N3 C18 C19 C20 N4 C2  
116 031117 031117 031118 031118 031119 031120 031121  
123 031124 031125 031125 031126 031126 031127 031128  
131 031131 031132 031132 031133 031134 031134 031135  
138 031139 031139 031140 031140 031141 031142 031143  
145 031146 031146 031147 031148 031148 031149 031150  
153 031153 031154 031154 031155 031156 031156 031157  
160 031160 031161 031162 031162 031163 031164 031165  
167 031168 031168 031169 031170 031170 031171 031172  
175 031175 031176 031176 031177 031178 031178 031179  
182 031182 031183 031184 031184 031185 031185 031186  
189 031190 031190 031191 031192 031192 031193 031194  
196 031197 031198 031198 031199 031199 031200 031201  
204 031204 031205 031206 031206 031207 031207 031208  
211 031212 031212 031213 031213 031214 031215 031216  
218 031219 031220 031220 031221 031221 031222 031223  
226 031226 031227 031227 031228 031229 031229 031230  
233 031234 031234 031235 031235 031236 031237 031238  
240 031241 031241 031242 031243 031243 031244 031245  
248  
115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline
```

## 9.

## 이벤트 플래그(09\_FLAG)

FreeRTOS 의 커널 서비스인 이벤트 플래그 함수들을 사용하여 태스크간 메시지 전달(IPC)에 사용 해보고 그 활용 용도(응용)에 대해서 생각 해 본다.

### 9.1. 소스 트리

“.app\09\_FLAG” 디렉토리에 실습할 파일이 있다

### 9.2. TODO #1

FreeRTOS의 커널 서비스인 이벤트 플래그 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 이벤트 플래그 생성 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

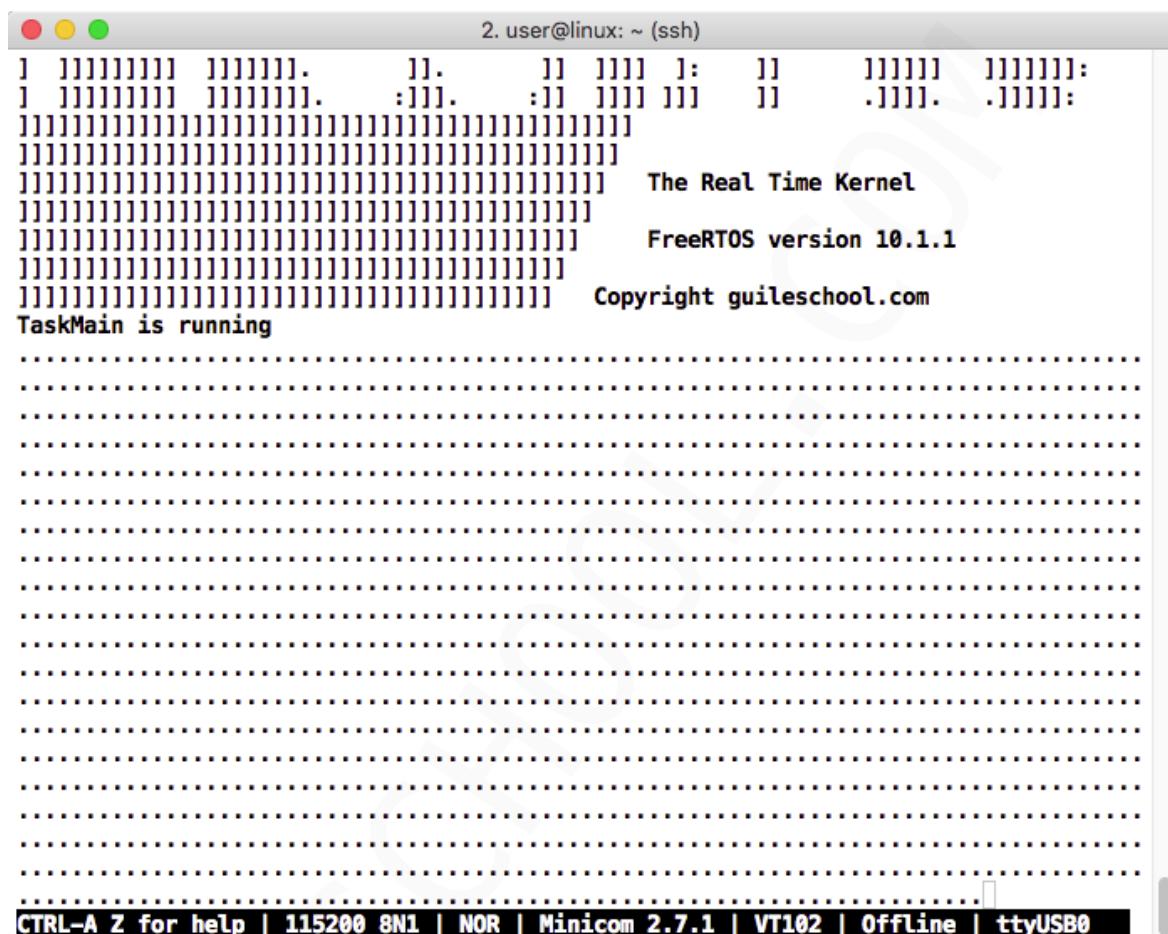
"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
     create a event flag  
     use group_id */  
#if 1  
group_id = ??????????????  
if (group_id == NULL) printf("xEventGroupCreate error found\n");  
#endif // TODO #1
```

이벤트 플래그 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 여러 메시지 없이 점(POINT)들만이 계속 출력 된다면 성공



The screenshot shows a terminal window titled "2. user@linux: ~ (ssh)". The window displays a series of characters forming a graphical representation of a face. The text includes:

- Kernel version information: "The Real Time Kernel", "FreeRTOS version 10.1.1", and "Copyright guileschool.com".
- A message "TaskMain is running" followed by a series of dots.
- A status bar at the bottom with the text "CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0".

### 9.3. TODO #2

FreeRTOS의 커널 서비스인 이벤트 플래그 생성 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 이벤트 플래그 생성 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
TaskMain 중에서
/* TODO #2:
   get flag 'ENGINE_OIL_PRES_OK' OR 'ENGINE_OIL_TEMP_OK' from TASK2 */
#if 1
uxBits = ??????????
#endif // TODO #2
```

```
Task1 중에서
/* TODO #2:
   post flag 'ENGINE_OIL_PRES_OK' to TASKMain */
#if 0
uxBits = ??????????
#endif // TODO #2
```

이벤트 플래그 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 1초마다 점(POINT) 들이 계속 출력 된다면 성공

```
2. user@linux: ~ (ssh)

Loading: #####  
        1.2 MiB/s  
done  
Bytes transferred = 88928 (15b60 hex)  
## Starting application at 0x80100000 ...  
0] ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]]]  
]]]]]]]]]]]]]  
]]]]]]]]]]]]]  
]]]]]]]]]]]  
]]]]]]]]]  
]]]]]]]  
]]]]]  
]]]  
.....]  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

## 10.

## 소프트웨어 타이머(10\_SOFT\_TIMER)

FreeRTOS 의 커널 서비스인 소프트웨어 타이머를 구현해본다.

### 10.1. 소스 트리

“.app\10\_SOFT\_TIMER” 디렉토리에 실습할 파일이 있다

### 10.2. TODO #1

FreeRTOS의 커널 서비스인 **소프트웨어 타이머** 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 소프트웨어 타이머 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

“#if 0” 를 “#if 1”로 하여 컴파일 할 수 있다

```
/* TODO #1:  
   원샷 소프트웨어 타이머(xOneShotTimer)의 구현 */  
#if 1  
    /* Create the one shot timer, storing the handle to the created timer in xOneShotTimer.  
     */  
    xOneShotTimer = xTimerCreate(  
        /* Text name for the software timer - not used by  
         FreeRTOS. */  
        "OneShot",  
        /* The software timer's period in ticks. */  
        mainONE_SHOT_TIMER_PERIOD,  
        /* Setting uxAutoReload to pdFALSE creates a one-  
         shot software timer. */  
        pdFALSE,  
        /* This example does not use the timer id. */  
        0,  
        /* The callback function to be used by the software  
         timer being created. */  
        prvOneShotTimerCallback );  
#endif // TODO #1
```

소프트웨어 타이머 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 예러 메시지 없이 메시지가 출력 된다면 성공

```
2. user@linux: ~ (ssh)
Loading: #####  
1.2 MiB/s
done
Bytes transferred = 86816 (15320 hex)
## Starting application at 0x80100000 ...
1]|||||||||:|||||||:|||||||:|||||||:|||||||:|||||||:|||||||:|||||||:  
1]|||||||||:|||||||:|||||||:|||||||:|||||||:|||||||:|||||||:|||||||:  
1]|||||||||:|||||||:|||||||:|||||||:|||||||:|||||||:|||||||:|||||||:  
1]|||||. . . . :|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.  
1]|||||||:|||.:|||.:|||.:|||.:|||.  
The Real Time Kernel  
FreeRTOS version 10.1.1  
Copyright guileschool.com  
xTimer1Started GOOD  
One-shot timer callback executing 100  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

### 10.3. TODO #2

FreeRTOS의 커널 서비스인 **소프트웨어 타이머** 생성 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 소프트웨어 타이머 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2:  
자동 반복 소프트웨어 타이머(xAutoReloadTimer)의 구현 */  
#if 1  
....  
#endif // TODO #2
```

TODO#1 을 최대한 잘 활용하여 소프트웨어 타이머 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

```
2. user@linux: ~ (ssh)
[...]
[...] The Real Time Kernel
[...] FreeRTOS version 10.1.1
[...] Copyright guileschool.com
xTimer1Started GOOD
xTimer1Started BAD
xTimer2Started GOOD
xTimer2Started BAD
Auto-reload timer callback executing 50
One-shot timer callback executing 100
Auto-reload timer callback executing 103
Auto-reload timer callback executing 150
Auto-reload timer callback executing 200
Auto-reload timer callback executing 250
Auto-reload timer callback executing 300
Auto-reload timer callback executing 350
Auto-reload timer callback executing 400
Auto-reload timer callback executing 450
Auto-reload timer callback executing 500
Auto-reload timer callback executing 550
Auto-reload timer callback executing 600
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

## 11.

## 메시지 큐(11\_QUE)

FreeRTOS 의 커널 서비스인 메시지 큐 함수를 사용하여 태스크간 메시지 전달(IPC)에 사용 해 보고 메일 박스와의 활용의 차이점에 대해서 생각 해 본다.

### 11.1. 소스 트리

“.app\11\_QUE” 디렉토리에 실습할 파일이 있다

### 11.2. TODO #1

FreeRTOS의 커널 서비스인 **메시지 큐** 생성 함수를 이용한 실습 예이다.

‘FreeRTOS 커널 API 레퍼런스 가이드’를 참조, 메시지 큐 생성 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #1:  
     create a Queue  
     use qid */  
#if 1  
qid = ??????????????????  
if (qid == NULL) printf("xQueueCreate error found\n");  
#endif // TODO #1
```

메시지 큐 생성 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다



## 11.4. TODO #2

FreeRTOS의 커널 서비스인 **메시지 큐** 전송 함수를 이용한 실습 예이다.

'FreeRTOS 커널 API 레퍼런스 가이드'를 참조, 메시지 큐 전송 함수의 사용법을 확인 한 후 코드를 완성하면 된다.

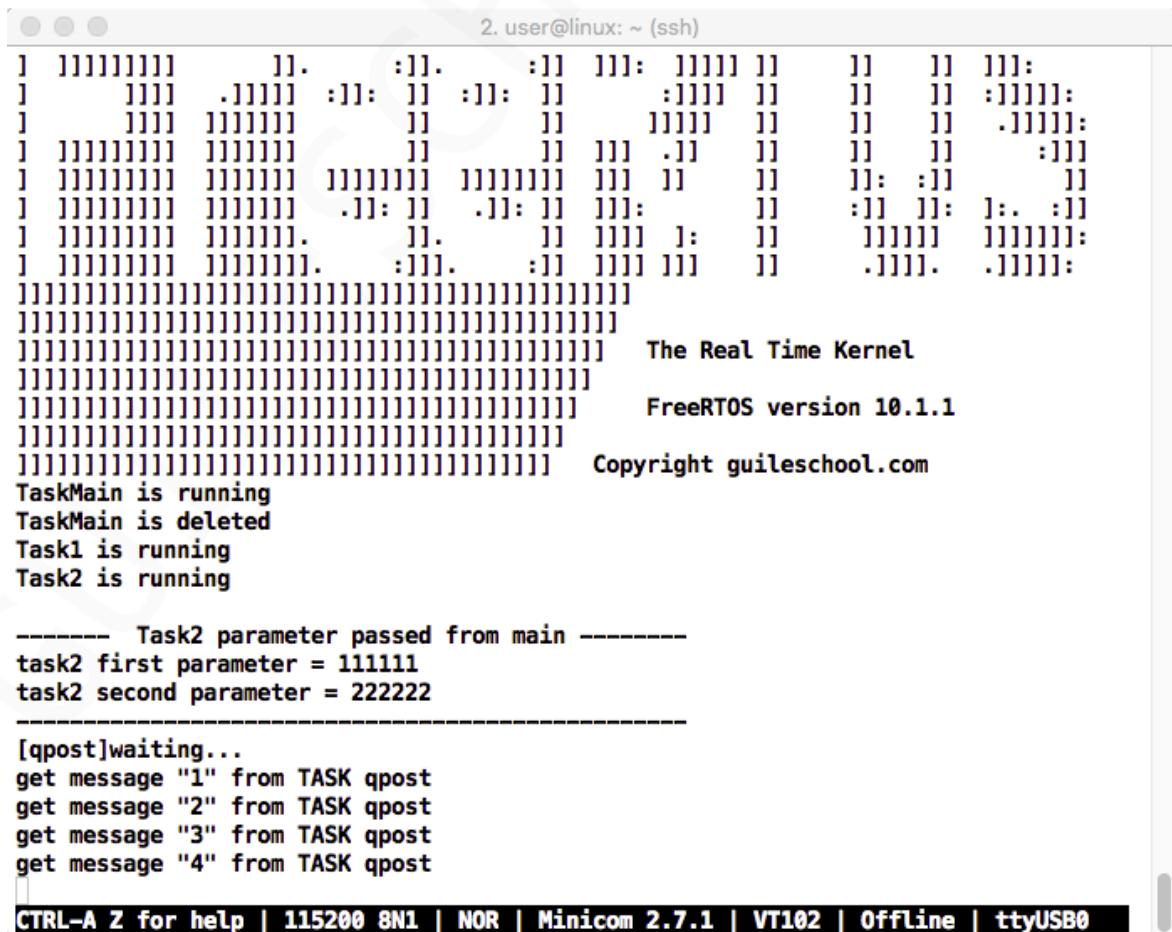
"#if 0" 를 "#if 1"로 하여 컴파일 할 수 있다

```
/* TODO #2:  
     post a message to TASK qpend  
     use msg[] */  
#if 1  
p=?????????????????  
    if (p != pdPASS) printf("xQueueSendToBack error found\n");  
#endif // TODO #2
```

메시지 큐 전송 함수를 구현한다

컴파일하고 실행하여 원하는 결과가 출력 되는지 확인한다

터미널 창에 그림처럼 에러 메시지 없이 메시지가 출력 된다면 성공



```
2. user@linux: ~ (ssh)  
[...]  
The Real Time Kernel  
FreeRTOS version 10.1.1  
Copyright guileschool.com  
TaskMain is running  
TaskMain is deleted  
Task1 is running  
Task2 is running  
----- Task2 parameter passed from main -----  
task2 first parameter = 111111  
task2 second parameter = 222222  
-----  
[qpost]waiting...  
get message "1" from TASK qpost  
get message "2" from TASK qpost  
get message "3" from TASK qpost  
get message "4" from TASK qpost  
-----  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

