

PART IV

아두이노 시스템 프로그래밍

목차

1. 동시 프로그래밍(*concurrent programming*)

2. 인터럽트

3. 운영체제 및 개념

4. *freeRTOS* 설치 및 프로그래밍

1. 동시 수행

- 스위치를 눌렀을 때 LED가 5초간 켜졌다가 꺼지고, 시리얼 모니터 창에 a라는 문자를 입력하면 'Hello'라는 문구를 출력

```
#define LED 13;
#define BUTTON 5;
void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
  Serial.begin(9600);
}
void loop() {
  char c = Serial.read();          // 시리얼 입력값을 읽음
  if (digitalRead(BUTTON) == LOW) { // 버튼 누르면 LED 5초 ON
    digitalWrite(LED, HIGH);
    delay(5000);
    digitalWrite(LED, LOW);
  }
  else { // 버튼이 눌리지 않았을 경우 'a' 입력하면 'Hello'
    if (c == 'a')
      Serial.println("Hello");
  }
}
```

Arduino	부품
D5	Button
D13	LED
5V	VCC
GND	GND

- delay()함수는 실행될 경우 아두이노의 모든 코드 동작을 일시적으로 중단
- 특히 통신에서 값을 받아서 특정한 출력을 하고 싶을 때 delay()가 존재할 경우 값을 보내도 출력이 지연되어 나온다던가 자기 멋대로 출력되는 결과가 생길 수 있음

- 버튼을 눌러 LED를 켜었을 때, 시리얼 모니터 창에 'a'라는 값을 입력 'Hello'가 출력되지 않음
- 그러다가 LED가 꺼지고 나서야 'Hello'가 시리얼 모니터 창에 출력됨
- 만약에 'Hello' 라는 글씨가 뜨지 않는다고 a를 계속적으로 입력 할 경우 LED가 꺼지고 나서야 a를 입력한 횟수만큼 'Hello' 가 출력됨

1. 동시 수행

- 스위치를 눌렀을 때 LED가 5초간 켜졌다가 꺼지고, 시리얼 모니터 창에 a라는 문자를 입력하면 'Hello'라는 문구를 출력

```
#define LED 13;
#define BUTTON 5;
long previousMillis = 0;
long interval = 5000;
void setup() {
    pinMode(LED, OUTPUT);
    pinMode(BUTTON, INPUT_PULLUP);
    Serial.begin(9600);
}
void loop() {
    unsigned long currentMillis = millis(); // 현재 시간 측정
    char c = Serial.read();
    if (digitalRead(BUTTON) == LOW) {
        previousMillis = currentMillis;
        digitalWrite(LED, HIGH);
    }
    else {
        if (c == 'a')
            Serial.println("Hello");
    }
    if (currentMillis - previousMillis > interval)
        digitalWrite(LED, LOW);
}
```

- 동시 수행을 위해, delay()로 정지시키는 것이 아니고, millis() 함수로 시간을 측정하는 방법을 사용하여, 각 루틴을 독립적으로 구현
- 우선, millis() 함수를 이용해 현재 시간을 측정(현재 시간은 아두이노가 시작된 이후의 밀리 초(1/1000)를 의미함)하여, 그 후에 특정 이벤트가 발생하면 그 이벤트의 발생시간을 측정하고 변수에 저장(previousMillis).
- 이벤트의 발생시간과 현재 시간을 비교하여 특정 시간 이상이 되면 수행해야 할 명령어를 실행함
- 굳이 delay()를 사용하지 않고도 시간을 계속적으로 체크하여 delay()의 효과를 유지하면서도 소스의 흐름이 끊기지 않게 만들 수 있음
- 더 간단한 소스는 파일-예제-02.Digital-BlinkWithoutDelay로 볼 수 있음.

1. 동시 수행

■ 두개의 LED를 별도로 제어(Blink 와 Fading)

Arduino	부품
11번 포트	Yellow LED
13번 포트	Red LED
GND 포트	GND 핀

- 빨간 LED와 노랑 LED가 차례로 깜박이고(Blink), 밝아졌다 어두워졌다(Fading) 하는 것을 볼 수 있음
- 그러나, 동시에 수행하지는 않고, **순차적으로 수행**되는 것을 볼 수 있음
- 빨간 LED 입장에서는, 노랑 LED가 하나의 delay(3060)한 것과 같고, 반대로 노랑 LED의 경우 delay(2000)한 것과 같음
- 두 LED가 독립적으로 동작하도록 하려면 어떻게 해야 할까?
- millis() 함수로 시간을 측정하는 방법 사용하여, 각 루틴을 독립적으로 구현

```
#define digiPin 13
#define anaPin 11;
void setup() {
  pinMode(digiPin, OUTPUT);
}
void loop() {
  // Blink Routine
  digitalWrite(digiPin, HIGH);
  delay(1000);
  digitalWrite(digiPin, LOW);
  delay(1000);
  // Fading Routine
  for(int i=0; i <= 255; i+=5) { // 점점 밝아 짐
    analogWrite(anaPin, i);
    delay(30);
  }
  for(int i=255; i >= 0; i-=5) { // 점점 어두워 짐
    analogWrite(anaPin, i);
    delay(30);
  }
}
```

1. 동시 수행(두 개의 LED가 동시에 별도로 동작)

```
int blink_ledState = LOW;
long blink_previousMillis = 0;
long blink_interval = 1000; // 깜박이는 시간 간격
int fadeValue = 0;
int fadeDir = 1; // 밝아지는 방향은 1, 어두워지는 방향은 0
long fading_previousMillis = 0;
long fading_interval = 30; // 밝아지거나 어두워지는 시간 간격
void setup() { pinMode(13, OUTPUT); }
void loop() {
    blink_loop();
    fading_loop();
}

void blink_loop() {
    unsigned long currentMillis = millis(); // 현재 시간 값
    // 현재 시간과 이전 시간을 비교해서, 1초가 되었는지 체크
    if(currentMillis - blink_previousMillis >= blink_interval) {
        blink_previousMillis = currentMillis;
        if(blink_ledState == LOW)
            blink_ledState = HIGH; // 1초가 되었으면 ON
        else
            blink_ledState = LOW; // 1초가 아직 안 되었으면 OFF
        digitalWrite(13, blink_ledState); // LED를 ON/OFF 수행
    }
}
```

```
void fading_loop() {
    unsigned long currentMillis = millis(); // 현재 시간 값
    // 밝아지거나 어두워지는 동작을 하기 위한 시간 체크(0.3초)
    if(currentMillis - fading_previousMillis >= fading_interval) {
        fading_previousMillis = currentMillis;
        if(fadeDir == 1) { // 방향 판단(밝아지는)
            if(fadeValue <= 255) { // 아직 밝아지는 중인지 체크
                fadeValue += 5;
                if(fadeValue > 255) { // 최대한 밝아졌는지 체크
                    fadeValue = 255;
                    fadeDir = 0;
                }
            }
        } else { // 방향(어두워지는)
            if(fadeValue >= 0) { // 아직 어두워지는 중인지 체크
                fadeValue -= 5;
                if(fadeValue < 0) { // LED가 꺼졌는지 체크
                    fadeValue = 0;
                    fadeDir = 1;
                }
            }
        }
        analogWrite(11, fadeValue); // LED에 값을 조절
    }
}
```

1. 동시 수행

■ millis() 함수

- millis() 함수가 어떻게 현재 시간을 리턴 할까?
- ATmega328 칩 내부에는 타이머 모듈이 3개 있으며, 각각 8 bit T/C 0, 16 bit T/C 1, 8 bit T/C 2 라고 함
- **millis() 함수**는 아두이노 보드상의 ATmega328 칩 내부에 있는 **타이머 모듈 0번**과 관련된 함수임
- 아두이노에서 타이머 0은 **주기적으로 AVR CPU로 신호를 보내도록 설정**되어 있음. 예를들어 초당 1000번 CPU로 신호를 보낼 수 있음. 이러한 신호를 인터럽트 신호(Interrupt signal)라고 함
- CPU는 인터럽트 신호를 받을 때마다 하드웨어적으로 정해진 메모리 번지에 있는 즉, 플래시(flash) 내에 있는 함수를 수행하게 됨
- 즉, 인터럽트 신호에 의해 하드웨어적으로 함수 호출이 일어나게 할 수 있으며, 이 때의 수행 함수를 인터럽트 처리기라고 함.
- 타이머 0의 인터럽트 처리기에서는 **SRAM에 카운터 변수**를 하나 두고 타이머 0 인터럽트가 발생할 때마다 이 변수를 증가시킴. 카운터 변수는 **초당 1,000씩 증가함**.
- millis() 함수는 이 변수의 값을 리턴해 주는 역할을 함
- 아두이노에서 시간은 약 50일이 지나면 오버플로우(overflow) 되어 0부터 다시 시작함

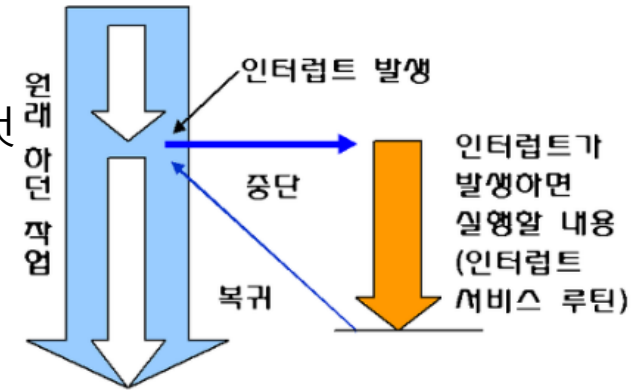
■ micros() 함수

- millis() 함수와 같이, 프로그램이 실행된 때부터 흐른 시간을 반환하는데, 마이크로초 단위로 함
- 약 70분이 지나면 오버플로우되어 0부터 다시 시작
- 아두이노 UNO R3는 4마이크로초 간격으로 시간이 흐름

2. 인터럽트

■ 인터럽트(Interrupt) 란

- 현재 어떤 프로그램이 실행중인 상태에서 인터럽트의 조건을 만족하는 어떤 변화가 감지되었을 경우, 현재 실행되는 프로그램을 중단시키고 인터럽트 처리 프로그램을 실행하는 것
- 인터럽트 서비스 루틴(ISR; Interrupt Service Routine)은 인터럽트가 발생되었을 때 이에 대응되는 서비스와 장소를 의미한다. 즉, 인터럽트 발생시 현재에 동작을 멈추고 해당 인터럽트를 제공하는 장소(메모리)에 가서 인터럽트에 대응하는 동작을 하는 것
- 마이크로컨트롤러의 외부 입출력 장치의 변화를 알아내는 방법은 2가지으로, 폴링(polling) 방식은 사용자 프로그램에서 입력 핀의 값을 계속 읽어서 변화를 알아내는 방식이며, 인터럽트 방식은 MCU 자체가 하드웨어적으로 그 변화를 체크하여 변화 시에만 일정한 동작을 하는 방식
- 인터럽트의 종류는 발생 원이 존재하는 곳에 따라 내부(internal)와 외부(external)로 나누고 차단 가능(Maskable) 인터럽트와 차단 불가(Non Maskable, NMI) 인터럽트로 나눈다.
- 내부 인터럽트는 프로그램 수행 과정에서 영(0)으로 나누는 등의 금지된 연산이나 금지된(protected) 메모리로의 접근 시도 등에 따라 MCU 내부에서 발생하는 인터럽트이며, 외부 인터럽트는 입출력 장치의 작업 종료, 타이머의 오버플로우, 외부 장치의 통신 요청 등 외부 장치에 의해 발생하는 인터럽트이다.
- 일반적으로 인터럽트라고 하면 외부 인터럽트를 의미한다.



2. 인터럽트 – 타이머/카운트 인터럽트

- 타이머/카운트 인터럽트는 프로그래머가 설정한 주기적인 시간마다 하던 일을 멈추고 인터럽트 서비스 루틴(ISR)으로 진입하여 서비스를 제공
- 예를 들어, 프로그램에서 0.001초에 타이머/카운터 인터럽트를 발생하라고 설정하면, MCU는 평상시에는 루프(loop)에서 머물고 있다가, 시간을 카운팅 한 후 0.001초가 되면 인터럽트를 발생시켜 MCU를 인터럽트 서비스 루틴으로 이동시켜 서비스를 제공한 후 다시 루프(loop)로 복귀함. 이 순서대로 MCU는 전원이 꺼질 때 까지 주기적인 시간(0.001초)마다 인터럽트의 발생을 반복한다.
- 타이머/카운트 인터럽트를 요약
 - 무한루프, 인터럽트 서비스 루틴을 번갈아 가면서 프로그램 실행이 주기적인 시간마다 반복
 - 크리스탈 소자를 이용, 발진 시스템 클럭을 카운팅하여 인터럽트를 발생시켜서 시간을 만들기 때문에 정확한 시간 제어 가능
 - 주기적으로 인터럽트가 발생하는 원리를 이용하여 실시간적이고 주기적으로 하드웨어 요청에 의한 응답을 하기 위해 현장에서 용이하게 사용 됨(예, 스위치 스캔, 센싱 체크, AD컨버터함수)
- 타이머/카운트 인터럽트 목적
 - 내.외부 장치들과 연동되어 시간 제어 목적으로 주로 사용
 - 주기적으로 인터럽트가 발생한다는 특징을 착안하여 프로그래머가 주기적으로 체크해야 하는 하드웨어 및 소프트웨어(사용자 변수)를 관리할 수 있다.
 - 예를들어, 스톱워치에서 시,분,초를 관리하거나 실시간으로 입력되는 디바이스를 체크

2. 인터럽트- 타이머/카운트 인터럽트

- 아두이노의 시간관련 함수들 - delay(), millis(), micros(), delayMicroseconds(), PWM 함수인 analogWrite() 및 tone(), noTone() 함수들 및 서보모터 라이브러리 등이 내부적으로는 타이머를 사용
- 타이머/카운트는 아두이노 컨트롤러에 내장된 하드웨어의 한 구성요소
- 아두이노 UNO의 ATmega328 컨트롤러는 3개의 타이머 - timer0, timer1, timer2
- timer0, timer2 는 8비트 타이머고 timer1은 16비트 타이머로, 8비트는 256단계의 값을 가지고 16비트는 65536 단계의 값을 가지고 있음
- Timer0 레지스터를 변경하면 delay(), millis(), micros() 와 같은 시간관련 함수들도 영향을 받고, 아두이노 UNO 보드의 경우 서보모터 라이브러리(Servo)가 Timer1을 사용하며, Timer2 는 8비트 타이머로 tone() 함수 등에 사용
- 인터럽트는 interrupts(), noInterrupts() 함수로 **활성화/비활성화** 할 수 있습니다. 아두이노의 기본설정은 인터럽트 활성화 상태임
- 일반적으로 코드에서 **attachInterrupt(), detachInterrupt()** 함수에 의해 설정되는 인터럽트는 **외부 인터럽트 핀에 의해 활성화** 됨.
- 타이머와 PWM 핀은 밀접하게 연관되어 있는데, Pin 5, 6은 timer0 에 의해 컨트롤, Pin 9, 10은 timer1 에, Pin 11, 3은 timer2 에 의해 컨트롤 됨
- 따라서, **주의 사항**으로, 서보 라이브러리(Servo library)는 Timer1 을 사용하기 때문에, 서보 라이브러리를 사용할 경우 9, 10번 PWM 은 사용할 수 없고, tone() 함수는 timer2를 사용하기 때문에 3, 11번 핀의 PWM을 사용할 수 없다.

2. 인터럽트- 타이머/카운트 인터럽트

- **타이머 레지스터**(TCCR, OCR, TCNT 등등)를 수정해가며 타이머를 사용한다는 것은 굉장히 소모적이고 **복잡**한 방법이기 때문에, 보다 간편하게 사용할 수 있는 **라이브러리**를 추천함
- <http://playground.arduino.cc/Main/LibraryList#Timing>
- Timer1, Timer3 라이브러리
 - <http://playground.arduino.cc/Code/Timer1>
 - ATmega128/328 의 16비트 타이머(Timer1)를 설정해주는 라이브러리로, 일반적인 라이브러리 설정경로와는 다르게 (Arduino/hardware/libraries/) 에 설치해야 함
- 타이머를 이용한 IR 리모트 콘트롤 예제 및 라이브러리
 - <http://www.righto.com/2009/08/multi-protocol-infrared-remote-library.html>
- SWTimer 라이브러리, SimpleTimer
 - <http://playground.arduino.cc/Code/Timer>
 - <http://playground.arduino.cc/Code/SimpleTimer>
 - 아두이노에 built-in 된 hardware 타이머를 직접 사용하는 방식이 아니라 단순히 millis() 함수를 이용해서 타이머를 구현한 라이브러리. 단순하고 하드웨어적인 정밀도와 제어가 필요치 않은 경우 유용.
- MStimer2라이브러리(유사하게 FlexiTimer2 라이브러리도 있음)
 - <http://playground.arduino.cc/Main/MsTimer2>
 - Timer2를 이용하는 작고 간단한 라이브러리. 0.001초 간격이라 millisecond(Ms)

2. 인터럽트- 타이머/카운트 인터럽트

<http://www.leonardomiliani.com/en/2011/swrtc-un-orologio-in-tempo-reale-via-software/>

■ swRTC 라이브러리 이용

- 타이머에 기반한 아두이노와 Atmel 마이크로컨트롤러를 위한 RTC(Real-Time Clock)의 S/W 구현

```
void startRTC(void)
```

```
void stopRTC(void)
```

```
boolean setTime(byte hourT, byte minuteT, byte secondT)
```

```
boolean setDate(byte dayT, byte monthT, int yearT)
```

```
byte getSeconds(void)
```

```
byte getMinutes(void)
```

```
byte getHours(void)
```

```
byte getDay(void)
```

```
byte getMonth(void)
```

```
int getYear(void)
```

COM5 (Arduino/Genuino Uno)

```
13:10:0--25/3/2017(Lead year: no)
Day of week :6
13:10:2--25/3/2017(Lead year: no)
Day of week :6
13:10:4--25/3/2017(Lead year: no)
Day of week :6
13:10:6--25/3/2017(Lead year: no)
Day of week :6
13:10:8--25/3/2017(Lead year: no)
Day of week :6
```

☒ 자동 스크롤

■ 날짜, 시간과 윤년을 판단하여 출력하는 프로그램

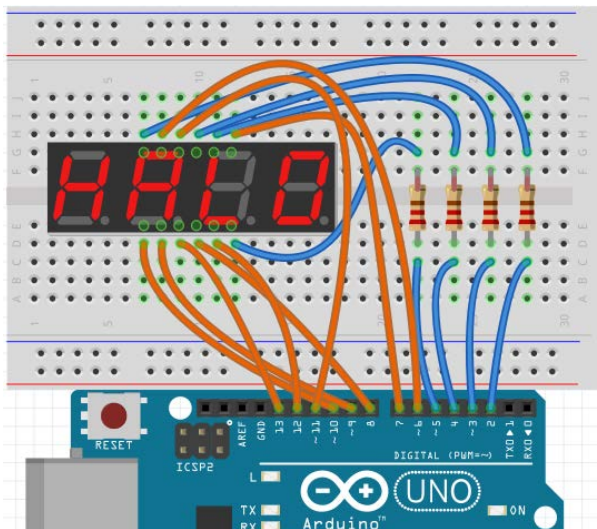
```
#include <swRTC.h> //swRTC 라이브러리 추가
#include <core_build_options.h> //swRTC 라이브러리에 포함
swRTC rtc; // 객체 선언
void setup() {
    rtc.stopRTC(); // RTC 정지
    rtc.setTime(13,10,0); // setTime 함수에 시, 분, 초를 설정
    rtc.setDate(25,3,2017); // setDate 함수에 일, 월, 년을 설정
    rtc.startRTC(); // 시간 설정 후 RTC 동작
    Serial.begin(9600);
}
void loop() {
    Serial.print(rtc.getHours(),DEC); // 시간, DEC는 십진수
    Serial.print(":");
    Serial.print(rtc.getMinutes(),DEC); // 분
    Serial.print(":");
    Serial.print(rtc.getSeconds(),DEC); // 초
    Serial.print("--");
```

```
Serial.print(rtc.getDay(),DEC); // 일
Serial.print("/");
Serial.print(rtc.getMonth(),DEC); // 월
Serial.print("/");
Serial.print(rtc.getYear(),DEC); // 년도
Serial.print("(Leap year: ");
if (rtc.isLeapYear()) // 윤년 판단 함수 호출
    Serial.print("yes");
else
    Serial.print("no");
Serial.println(")");

Serial.print("Day of week : ");
Serial.println(rtc.getWeekDay(),DEC);
delay(2000);
}
```

2. 인터럽트- 타이머/카운트 인터럽트

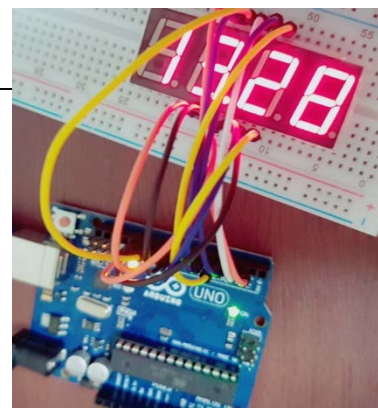
■ swRTC를 이용 시계 만들기(현재 시간을 7세그먼트에 출력)



Arduino	4bit 7 Segment
D2	Pin 1 (자리선택)
D3	Pin 2 (자리선택)
D4	Pin 3 (자리선택)
D5	Pin 4 (자리선택)
D6	Pin 1 (제어핀)
D7	Pin 2 (제어핀)
D8	Pin 3 (제어핀)
D9	Pin 4 (제어핀)
D10	Pin 5 (제어핀)
D11	Pin 6 (제어핀)
D12	Pin 7 (제어핀)
D13	Pin 8 (제어핀)

```
#include <SevSeg.h>
#include <core_build_options.h>
#include <swRTC.H>
SevSeg sevseg;
swRTC rtc;
void setup(){
  rtc.stopRTC();
  rtc.setTime(13,10,0);
  rtc.startRTC();
  byte numDigits = 4; //4bit 설정
  byte digitPins[] = {2, 3, 4, 5}; //공통 핀 설정
  byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13}; //제어 핀 설정
  sevseg.begin(COMMON_ANODE, numDigits, digitPins, segmentPins);
  sevseg.setBrightness(10);
}
void loop(){
  sevseg.setNumber(rtc.getHours()*100+rtc.getMinutes(),2);
  sevseg.refreshDisplay();
}
```

7세그먼트에 시간이
13(시간) . 28(분)과 같은
형식으로 출력



2. 인터럽트

■ SimpleTimer 라이브러리 이용

<http://playground.arduino.cc/Code/SimpleTimer>

에서 소스를 받아서, libraries 폴더에

- 시간 제어에 간단한 라이브러리로, millis() 함수를 기반으로 작성됨(인터럽트를 이용하지 않았음)
- Timer.setTimeout(밀리초, 함수명); // 밀리초 후에 함수 실행
- Timer.run(); // 타이머 업데이트, 실행

■ 버튼을 누르면 LED가 켜졌다가 5초가 지나면 LED가 꺼지면서 알람이 울림

```
#include<SimpleTimer.h>
#define BUTTON 2
#define ALAM 8
#define LED 13
SimpleTimer timer; // 객체생성
int pState = LOW;
boolean booked = false; //예약
void setup() {
  pinMode(BUTTON, INPUT);
  pinMode(LED, OUTPUT);
}
void loop() {
  int state = digitalRead(BUTTON);
  if (pState==LOW && state==HIGH) { // 버튼이 눌러졌다면
    if(booked==false) {
      booked=true;
      digitalWrite(LED, HIGH); // LED on
      timer.setTimeout(5000, beep); // 알람 5초 설정
    }
  }
  pState = state;
  timer.run(); // 타이머 업데이트
}
```

Arduino	센서
D13	LED
D8	Buzzer
D2	Button
VCC	5V
GND	GND

```
void beep() {
  booked=false;
  digitalWrite(LED, LOW);
  tone(8,1000,20);
  delay(100);
  tone(8,1000,20);
  delay(100);
  tone(8,1000,20);
}
```

2. 인터럽트

■ SimpleTimer 라이브러리 이용

- `Timer.setInterval(밀리초, 함수명);` // 밀리초 마다 함수 반복 실행

■ LED는 1초마다 blink되고, 조도센서는 5초 마다 값을 읽어 시리얼 모니터에 출력

```
#include<SimpleTimer.h>
#define LED 13
SimpleTimer timer;
boolean ledOn = false;
void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  timer.setInterval(1000,toggle);//특정 간격마다 함수 실행
  timer.setInterval(5000,notify);
}
void loop() {
  timer.run();
}
void toggle() {
  if(ledOn == true)
    digitalWrite(LED, LOW);
  else
    digitalWrite(LED, HIGH);
  ledOn = !ledOn;
}
void notify() {
  Serial.print("Brightness:");
  Serial.println(analogRead(A0));
}
```

Arduino	센서
D13	LED
A0	조도
VCC	5V
GND	GND

2. 인터럽트

■ MsTimer 라이브러리 이용

<http://playground.arduino.cc/Main/MsTimer2>
에서 라이브러리 다운받아 추가

- MsTimer2::set(ms, name); // 타이머 인터럽트를 설정 함수
- MsTimer2::start(); // 타이머 인터럽트를 실행시키는 함수
- MsTimer2::stop(); // 타이머 인터럽트를 정지시키는 함수

■ 0.5초 마다 LED의 상태가 바뀌는 프로그램

```
// Toggle LED on pin 13 each second
#include <MsTimer2.h>
void flash() {
  static boolean output = HIGH;
  digitalWrite(13, output);
  output = !output;
}
void setup() {
  pinMode(13, OUTPUT);
  MsTimer2::set(500, flash); // 500ms period
  MsTimer2::start();
}
void loop() {
}
```

■ playground.arduino.cc/Main/MsTimer2 에 다양한 예제

2. 인터럽트 - 외부 인터럽트 핀

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	
Due	모든 핀 지원					

■ 아두이노에서 인터럽트(Interrupt)

- UNO에서는 디지털 입출력 포트 D2, D3 값이 변할 때 지정한 함수를 불러오는 기능
- 외부 인터럽트 처리 함수 `attachInterrupt(pin, ISR, mode)`를 이용. 해제는 `detachInterrupt(pin)`
- UNO는 인터럽트 번호 0(D2 핀) 또는 1(D3 판)의 값이 변할 때 특정 함수를 불러와 실행

■ 버튼(D2)을 누르면 LED의 상태가 바뀌는 프로그램

Arduino	부품
D2	Button
D13	LED
5V	VCC
GND	GND

```
// www.arduino.cc/en/Reference/AttachInterrupt
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
// attachInterrupt(0, blink, CHANGE);
}
void loop() {
  digitalWrite(ledPin, state);
}
void blink() {
  state = !state;
}
```

- LOW : 신호가 LOW일 때 인터럽트 발생, LOW인 동안 지속적으로 ISR 호출
- CHANGE : 신호가 바뀔 때 인터럽트 발생
- RISING : 신호가 LOW에서 HIGH로 바뀔 때 인터럽트 발생
- FALLING : 신호가 HIGH에서 LOW로 바뀔 때 인터럽트 발생

- 블루투스 모듈과의 통신을 위해 주로 사용되는 SoftwareSerial 라이브러리의 경우 내부적으로 인터럽트를 사용하는 것으로 알려져 있음.
- 따라서 UNO 보드의 경우 D2, D3 에 연결해야만 정상동작 함. 이를 회피하기 위해서 D0, D1 핀에 연결해서 Hardware Serial 로 동작시킬 수 있음

주의점 • 인터럽트 함수 처리는 짧은 시간에 끝나게 한다. 긴 처리를 실행하면 때에 따라 다른 인터럽트 처리(시리얼 통신 등)가 멈춰버려 오류가 발생한다.

- ISR 에서는 `delay()`와 `millis()` 함수가 제한됩니다. `delayMicroseconds()` 는 사용이 가능
- 전역 변수를 참조 혹은 변경하려면 처음 그 변수를 정의할 때, `volatile` 를 앞에 붙여준다.

3. 운영체제 및 개념

- 복잡한 동시 수행이나 자원 등의 관리 등을 위해 필요한 프로그램을 미리 작성
- OS와 RTOS
- OS(Operating System), 운영체제
 - 시스템 **하드웨어를 관리**할 뿐 아니라 응용 소프트웨어를 실행하기 위하여 하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공하는 **시스템 소프트웨어**
 - 실행되는 응용 프로그램들이 메모리와 CPU, 입출력 장치 등의 **자원들을 사용**할 수 있도록 만들어 주고, 이들을 추상화하여 파일 시스템 등의 서비스를 제공한다. 또한 **멀티태스킹**을 지원하는 경우, 여러 개의 응용 프로그램을 실행하고 있는 동안, 운영 체제는 이러한 모든 **프로세스들을 스케줄링**하여 마치 그들이 **동시에 수행**되는 것처럼 보이는 효과를 낸다.
 - 핵심적 기능은 커널(kernel)이며, 사용자가 접근할 수 있도록 하는 UI나 Utility까지를 포함
- 실시간 운영체제(RTOS; Real Time Operating System)
 - 실시간 응용 프로그램을 위해 개발된 운영 체제이다. 운영 체제의 기능 중 **CPU 시간 관리** **부분에 초점**을 맞추어 설계되었다.
 - 여러가지 작업을 태스크(task) 단위로 처리하는 다중 작업 처리 방식 지원
- 두 가지 기본적인 설계 방식이 존재한다.
 - **이벤트 구동(event-driven) 방식** : 우선 순위 기반 스케줄링 또는 선점형 스케줄링 이라고 부른다. 태스크(task) 전환이 현재 수행중인 태스크보다 높은 우선 순위를 갖는 이벤트가 서비스를 요청할 경우에 일어난다.
 - **시분할(time-sharing) 스케줄링 방식** : 클럭 인터럽트나 라운드 로빈과 같은 주기적인 이벤트가 발생할 때 태스크의 전환이 일어난다.

3. 운영체제 및 개념

■ RTOS(Real-Time Operating System)

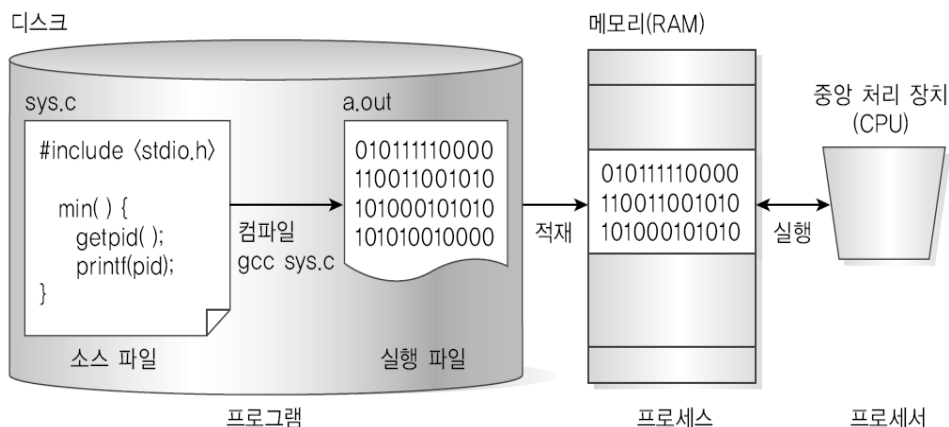
운영체제	제조사	국내 대리점	로열티 정책	구조
VxWorks	WindRiver	WindRiver Korea	○	멀티쓰레드
OSE	Enea OSE Systems	트라이콤텍	△	멀티쓰레드
VRTX	Mentor Graphic	다산인터넷	○	멀티쓰레드
PSoS	WindRiver	WindRiver Korea	○	멀티쓰레드
Nucleus Plus	Accelerated Technology	ATI Korea	×	멀티쓰레드
Super Task	US Software	아라전자	×	멀티쓰레드
C/OS II	Micrium	디오이즈	×	멀티쓰레드
QNX	QNX Software Systems	다산인터넷	○	멀티프로세스
OS-9	Microware	Microware Korea	○	멀티프로세스
LynxOS	LinuxWorks	●	△	멀티프로세스
RTLinux	Finite State Machine Labs	●	△	멀티프로세스
Windows CE	MicroSoft	MicroSoft	○	멀티프로세스

* 로열티 정책의 ○는 로열티를 받는다는 표시이고 △는 명확하지 않다는 표시이며 ×는 로열티를 받지 않는다는 표시이다.

3. 운영체제 및 개념

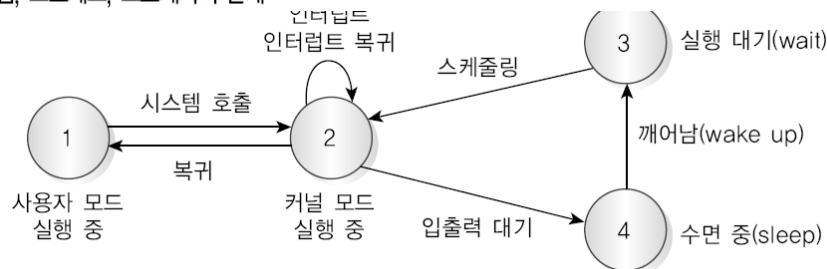
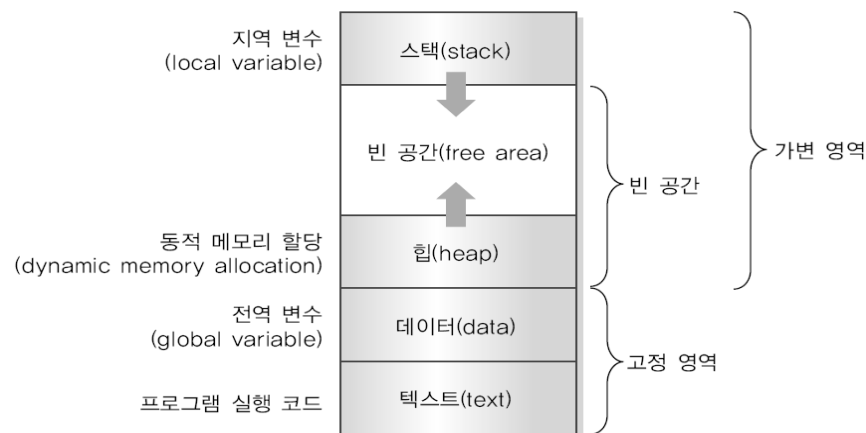
■ 개념과 용어 - 프로세스

- **프로세스(Process)**는 현재 메인 메모리에 저장되어 실행되는 프로그램으로 운영 시스템 내에서는 작업의 단위가 된다. 다시 말하면 프로세스는 프로그램이 실행되기 위해 메인 메모리에 저장된 상태로 실행 코드와 전역 데이터 그리고 지역 데이터 및 그 외의 실행에 필요한 데이터들이 메인 메모리 내의 각 영역에 저장된 상태를 말한다.
- 프로세스를 간단히 설명하면, 실행 중인 상태의 프로그램이다. 또한, 동일한 프로그램으로 여러 개의 프로세스를 생성할 수 있으며, 생성된 각 프로세스를 프로그램의 인스턴스(instance)라고도 한다.



[그림 5-1] 프로그램, 프로세스, 프로세서의 관계

[그림 5-2] 프로세스의 기본 구조



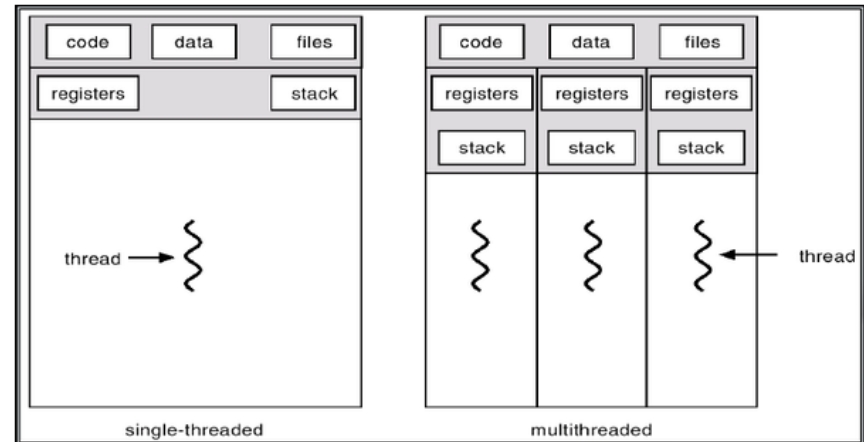
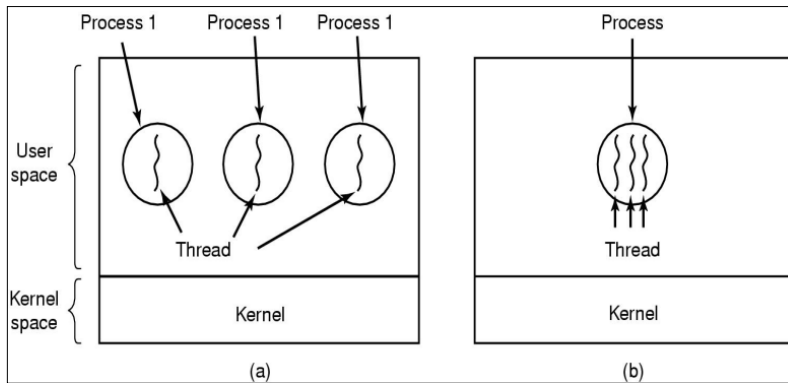
[그림 5-3] 프로세스의 상태 및 전이

3. 운영체제 및 개념

■ 개념과 용어 - 스레드(Thread)



- **스레드(Thread)**의 영어 단어의 뜻은 실, 가닥, 줄기이며, **태스크(Task)**라는 용어를 사용하기도 함
- 프로그래밍에서의 스레드는 프로세스 내에서 실행되는 흐름의 단위를 말한다. 일반적으로 한 프로세스는 최소한 하나의 스레드를 가지고 있다. 프로그램 환경에 따라 둘 이상의 스레드가 동시에 실행 수 있으며, 멀티스레드(Multi-Thread)라고 한다.
- 컴퓨터 CPU의 스펙을 살펴보면 2코어 2스레드, 4코어 4스레드, 4코어 8스레드 등의 표현을 하는데, 위에서 얘기 했듯이 스레드는 프로세서(프로그램)를 수행하는 일의 흐름을 말하기 때문에 스레드가 많으면 많을 수록 같은 시간 동안 동시에 수행할 수 있는 일의 갯수가 많아지게 됨.

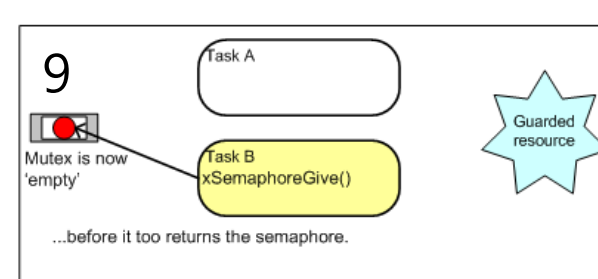
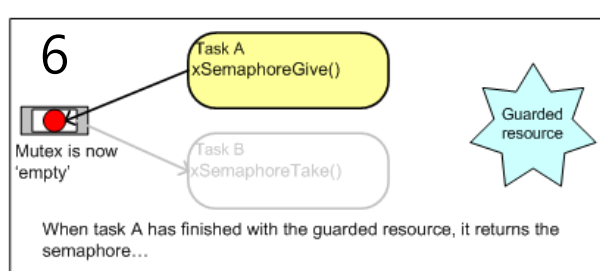
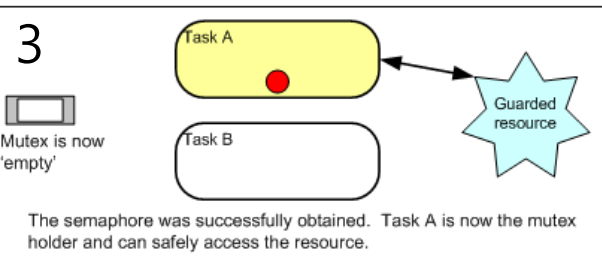
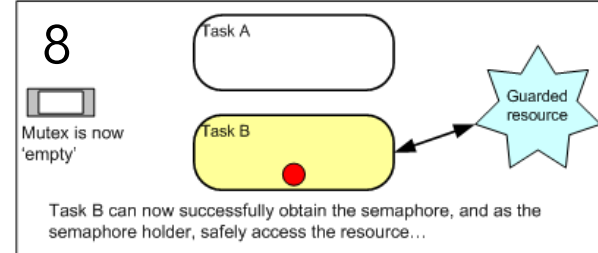
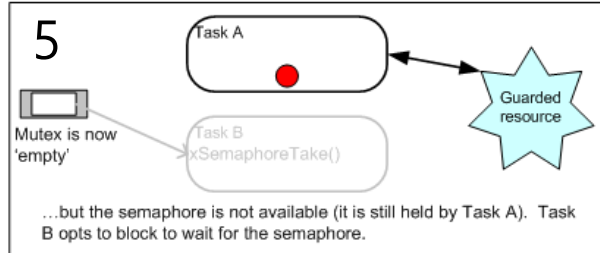
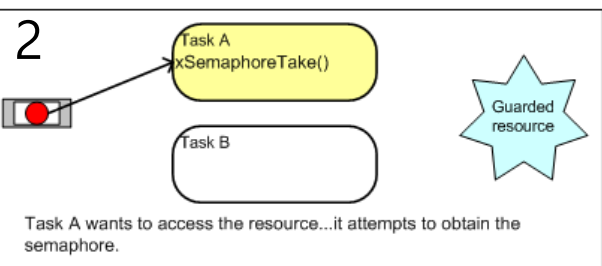
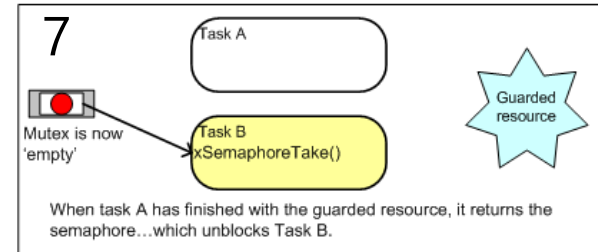
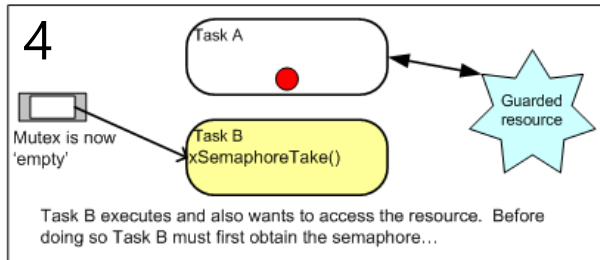
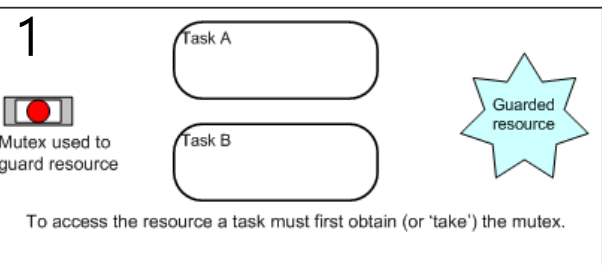
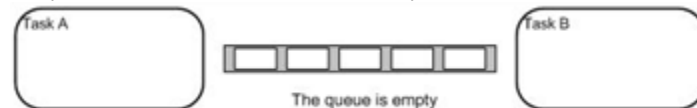


- **아두이노**는 기본적으로 **싱글스레드(Single thread)**이기 때문에 같은 시간 동안 하나의 명령어만 처리할 수 있음. 그렇기 때문에 `delay()`가 코드 중간에 걸릴 경우 다수의 모듈을 사용할 때 상당히 애매해지는데 이때 스레드를 사용하면 `delay()`에 영향을 받는 다른 모듈을 스레드를 추가하여 작동시킨다면 `delay()`의 영향없이 동시에 두 가지 모듈을 작동시킬 수 있게 됨

3. 운영체제 및 개념

■ 개념과 용어 – Inter-task Communication

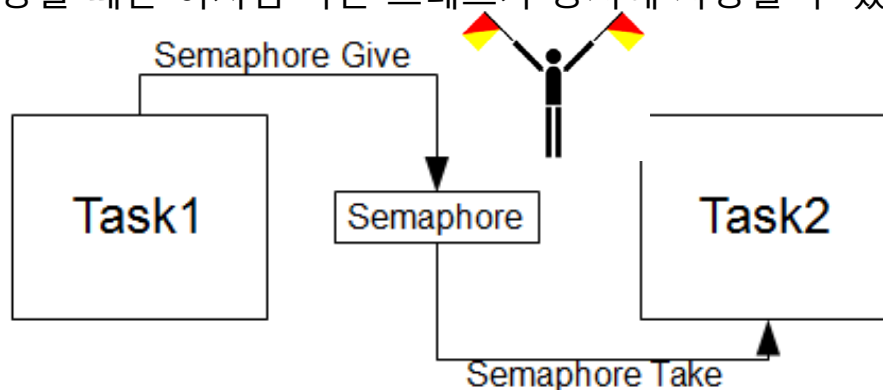
- 프로세스 간에 데이터를 주고 받는 방법(방식) 들을 총칭해서, IPC(Inter-Process Communication) 이라 함
- IPC로는 Signal, Pipe, Shared Memory, Message Queue, Socket, RMI 등 다양함. 또한, 프로세스 간 동기화를 위해, Semaphore, Mutexes 등을 사용
- 스레드 또는 태스크(Task) 간 통신하기 위한 방법으로, Queue 를, 동기화로 Mutexes, Semaphore 를 이용



3. 운영체제 및 개념

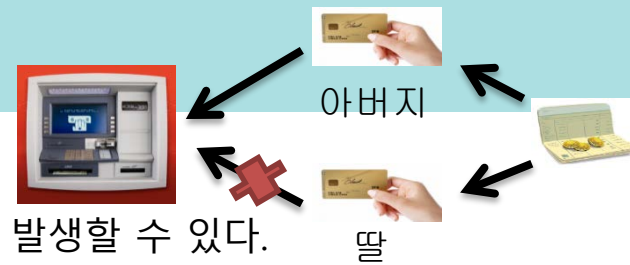
■ 개념과 용어 - 세마포어(Semaphore)

- 멀티스레드(Multi-thread)를 사용하다 보면 서로 다른 스레드에서 하나의 데이터(디바이스)를 동시에 사용하려고 하는 경우가 생기는데, 이럴 때 문제가 생길 수 있는데 누가 먼저 공유 데이터에 접근하느냐에 따라 데이터가 변할 수 있음
- 현실 세계에서는 신호등이나 깃발 등을 통해 순서를 제어함
- 프로그래밍에서는 세마포어를 사용하여 공유데이터에 접근하는 스레드를 조절할 수 있음
- 아래 그림은 세마포어를 Task1이 Task2로 건네주는 그림. Task1이 공유 데이터를 사용한 뒤 Task2에게 Semaphore Give를 사용하여 세마포어를 Task2로 넘겨주고, 그러면 Task2에서는 Semaphore Take를 통해 세마포어를 받고 공유 데이터에 접근할 수 있는 권한이 생겨 공유자원을 사용할 수 있음
- 세마포어는 1개가 될 수도 있고 2개가 될 수도 있고, 여러 개가 될 수도 있으며, 세마포어는 단지 공유 데이터에 접근할 수 있는 권한을 부여할 뿐임
- 멀티 스레드를 사용할 때는 이처럼 다른 스레드가 동시에 사용할 수 있는 공유 데이터가 존재할 수 있으니 유의해야 함



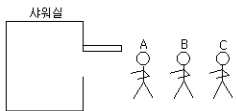
3. 운영체제 및 개념

■ 개념과 용어 - 세마포어(Semaphore)

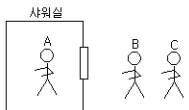


- 하나의 자원에 사용자 두 명이 동시에 접근할 시 이는 치명적인 오류가 발생할 수 있다.
- 간단한 예로, ATM 기계에 한통장을 사용하고 있는 카드 2개가 동시에 접근할 시, 늦게 접근한 하나는 대기상태가 되어 있어야 한다. 조회는 괜찮지만, 출금이나 입금 시에는 문제가 발생
- 이와 같이, 동시에 접근해 처리하면, 문제가 발생할 수 있는 영역을 critical section(임계영역, 위험지역)
- 임계 영역을 관리하기 위한 방법이 필요. 프로그래머가 하거나 시스템에서 제공하는 기능이용

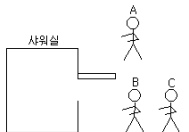
(1) 하나의 샤워실이 있다고 하자. 이 샤워실을 이용하기 위해서는 문이 열려있어야 한다. 다행히도 문이 열려 있어서 A가 샤워실에 들어간다.



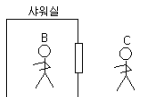
(2) A는 샤워실에 들어가서 문을 잠근다.



(3) B와 C는 샤워실에 들어가고 싶어도 문이 잠겨 있어서 들어갈 수 없다. 샤워를 끝낸 A는 문을 열고 나간다.



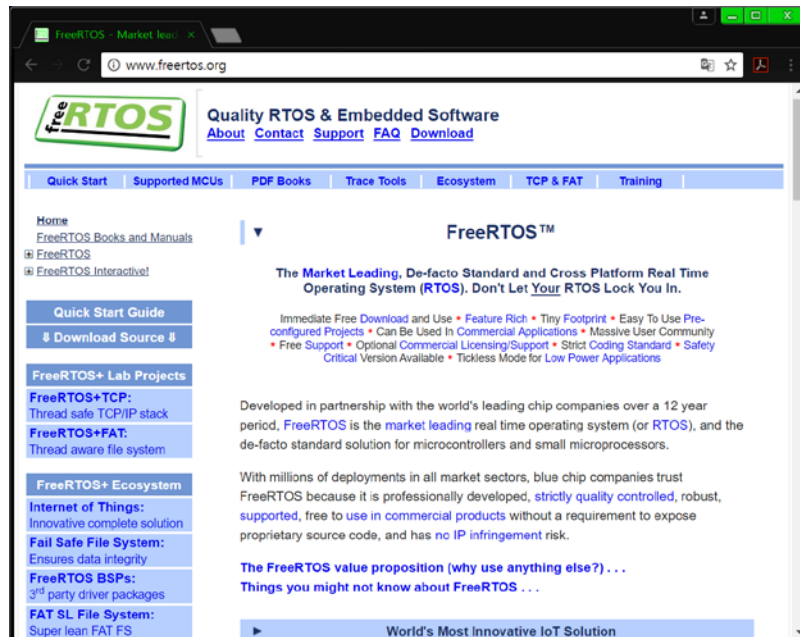
(4) 샤워실 문이 열려 있으므로 B가 들어가서 문을 잠근다. C는 여전히 문이 잠겨 있어서 못들어 간다. 결국 샤워실이라는 하나의 자원을 한 명 즉 하나의 프로세스만 이용할 수 있게 된다.



- 세마포어는 빈 화장실 열쇠의 갯수
- 즉, 네 개의 화장실에 자물쇠와 열쇠가 있다고 한다면 세마포어는 열쇠의 갯수를 계산하고 시작할 때 4의 값을 갖는다. 이 때는 이용할 수 있는 화장실 수가 동등하게 된다.
- 이제 화장실에 사람이 들어갈 때마다 숫자는 줄어들게 되며, 4개의 화장실에 사람들이 모두 들어가게 되면 남은 열쇠가 없게 되기 때문에 세마포어 카운트가 0이 된다.
- 이제 다시 한 사람이 화장실에서 볼일을 다 보고 나온다면 세마포어의 카운트는 1이 증가된다.
- 따라서 열쇠 하나가 사용가능하기 때문에 줄을 서서 기다리고 있는 다음 사람이 화장실에 입장할 수 있게 된다.
- 세마포어 Counter의 갯수에 따라 1개의 경우 Binary Semaphore, 2개 이상의 경우 Counting Semaphore라고 불린다.
- Binary Semaphore의 경우 개념적으로 Mutex와 같다고 볼 수 있다.

4. FreeRTOS 소개

- 임베디드 시스템을 위한 운영체제이며, GPL 라이선스를 따름
- 프리입니까?(예), 사용제품에 적용할수 있습니까?(예), 로열티가 있습니까? (아니오)
- 제 어플리케이션을 오픈해야 합니까?(아니오), 만약 제가 커널 소스를 수정한다면 오픈해야 합니까?(예)
- 적용하는 제품의 매뉴얼에 FreeRTOS 를 사용하고 있음을 표시해야 합니까?(예, 링크만)
- 내가 만든 어플리케이션 사용자에게 FreeRTOS 를 제공해도 되나요?(예)
- 사용 제품에 적용할 때 기술지원을 받을 수 있나요?(아니오, 별도로 커머셜 라이선스에서 제공하고 있습니다)
- 보증이 가능합니까?(아니오, 별도로 커머셜 라이선스에서 제공하고 있습니다)
- 복제 방지가 제공되나요?(아니오, 별도로 커머셜 라이선스에서 제공하고 있습니다)



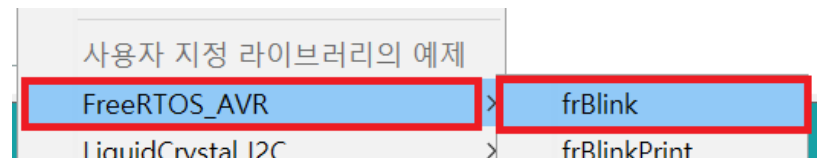
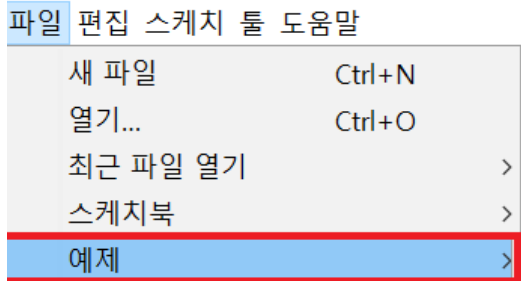
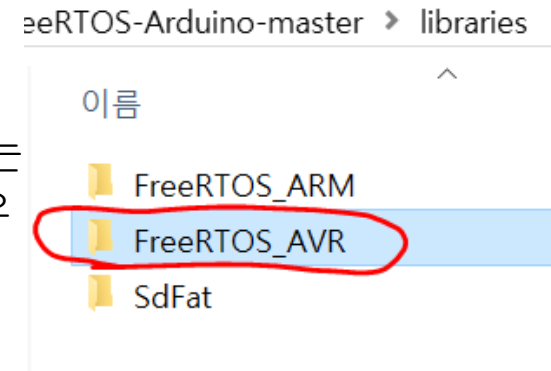
Supported MCUs

- **Atmel** Supported processor families: SAMV7 (ARM Cortex-M7), SAM3 (ARM Cortex-M3), SAM4 (ARM Cortex-M4), SAMD20 (ARM Cortex-M0+), SAMA5 (ARM Cortex-A5), SAM7 (ARM7), SAM9 (ARM9), AT91, **AVR** and AVR32 UC3
- **ST** Supported processor families: **STM32** (ARM Cortex-M0, ARM Cortex-M7, ARM Cortex-M3 and ARM Cortex-M4F), STR7 (ARM7), STR9 (ARM9)
- **TI** Supported processor families: RM48, TMS570, ARM Cortex-M4F MSP432, MSP430, MSP430X, SimpleLink, Stellaris (ARM Cortex-M3, ARM Cortex-M4F)

4. FreeRTOS 소개

■ FreeRTOS 라이브러리 설치

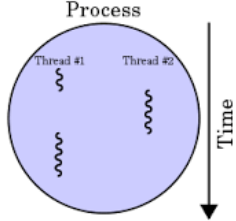
- 라이브러리를 다운로드 받는다.
<https://github.com/greiman/FreeRTOS-Arduino>
- FreeRTOS-Arduino-master\FreeRTOS-Arduino-master\libraries 압축을 푼다(FreeRTOS_ARM의 경우에는 ARM cortex-M3를 사용하는 arduino DUE를 사용할 경우 사용하고 나머지 Atmel칩을 사용하는 AVR 보드는 FreeRTOS_AVR을 사용)
- FreeRTOS_AVR 폴더를 아두이노의 libraries폴더로 복사
- 아두이노를 재시작
- 파일 - 예제에 아래와 같이 두개의 예제폴더가 보인다면 FreeRTOS라이브러리 설치 완료



4. FreeRTOS: <https://github.com/greiman/FreeRTOS-Arduino>

- The folder examples/FreeRTOSBook has sixteen examples. These examples are described in the book: Using The FreeRTOS Real Time Kernel - a Practical Guide
- http://shop.freertos.org/RTOS_primer_books_and_manual_s/1819.htm
- The following examples are fully described in the above book. I have modified
- The examples to run as Arduino sketches. The original code is here.
- <http://www.freertos.org/Documentation/code/source-code-for-standard-edition-examples.zip>
- Warning: The interrupt examples use attachInterrupt() to generate interrupts.
- The pin for interrupt zero must be free for use as an output on AVR. On ARM Due you must connect pin 2 to pin 3. Input characters will stop an example.
- Example 1. Creating Tasks
- Example 2. Using the Task Parameter
- Example 3. Experimenting with priorities
- Example 4. Using the Blocked state to create a delay
- Example 5. Converting the example tasks to use vTaskDelayUntil()
- Example 6. Combining blocking and non-blocking tasks
- Example 7. Defining an Idle Task Hook Function
- Example 8. Changing task priorities
- Example 9. Deleting tasks
- Example 10. Blocking When Receiving From a Queue
- Example 11. Blocking When Sending to a Queue / Sending Structures on a Queue
- Example 12. Using a Binary Semaphore to Synchronize a Task with an Interrupt
- Example 13. Using a Counting Semaphore to Synchronize a Task with an Interrupt
- Example 14. Sending and Receiving on a Queue from Within an Interrupt
- Example 15. Rewriting vPrintString() to Use a Semaphore
- Example 16. Re-writing vPrintString() to Use a Gatekeeper Task

4. FreeRTOS 실습(두 개 Task가 동시에 시리얼로 출력)



```
#include "FreeRTOS_AVR.h"
#include "basic_io_avr.h"
#define mainDELAY_LOOP_COUNT 400000 // 0xfffff
void vTask1( void *pvParameters );
void vTask2( void *pvParameters );
void setup( ) {
    Serial.begin(9600);
    static const char *string1 = "Task 1 is running\r\n";
    xTaskCreate( vTask1, //태스크 구현 함수명
        "Task 1",      //태스크 이름
        200,           //스택 깊이
        (void*)string1, //태스크에 넘겨 줄 인자 포인터
        1,             //태스크의 우선순위
        NULL );        //태스크 핸들
    //같은 방식으로 두번째 태스크를 생성한다.
    static const char *string2 = "Task 2 is running\r\n";
    xTaskCreate( vTask2,"Task 2", 200, (void*)string2, 1, NULL);
    vTaskStartScheduler();//스케줄러를 시작
    for( ;; );
}
```

```
void vTask1( void *pvParameters ){
    const char *pcTaskName = (char*)pvParameters;
    volatile unsigned long ul;
    for( ;; ) { //태스크는 무한 루프
        vPrintString(pcTaskName); //시리얼로 출력
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ );
    }
}

void vTask2( void *pvParameters ){
    const char *pcTaskName = (char*)pvParameters;
    volatile unsigned long ul;
    for( ;; ) {
        vPrintString( pcTaskName );
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ );
    }
}

void loop(){ }//freeRTOS에서는 loop()를 사용하지 않음
```

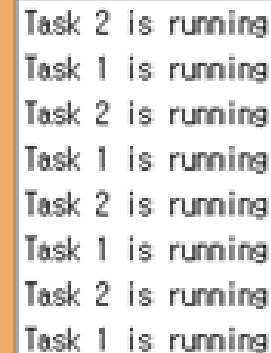
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running

- vTaskDelay() 함수를 사용 하여 Delay 기능을 구현 하면 Delay 상태 동안 Task가 Suspended 상태가 되기 때문에 Priority 에 관계 없이 다음 Task가 실행
- 그러므로, Priority 에 따라 Task가 Running 되는 것을 관찰 하기 위하여 for Loop를 이용한 Delay를 사용.
- 참고 <http://goo.gl/eizFMp>

4. FreeRTOS 실습(같은 함수의 두 개의 태스크가 시리얼로 출력)

```
#include "FreeRTOS_AVR.h"
#include "basic_io_avr.h"
#define mainDELAY_LOOP_COUNT 400000// ( 0xfffff )
void vTask( void *pvParameters ); //태스크에서 사용할 함수
const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\r\n";
void setup( void ) {
    Serial.begin(9600);
    //같은 함수를 사용하는 두 개의 태스크를 생성, 다른 인자
    xTaskCreate( vTask, "Task 1", 200, (void*)pcTextForTask1, 1, NULL );
    xTaskCreate( vTask, "Task 2", 200, (void*)pcTextForTask2, 1, NULL );
    vTaskStartScheduler(); //스케줄러를 시작
    Serial.println( F("Insufficient RAM") );
    while(1);
}
void vTask ( void *pvParameters ) {
    char *pcTaskName;
    volatile unsigned long ul;
    //인자로 받은 문자열을 char*로 캐스팅해서 저장한다.
    pcTaskName = (char*)pvParameters;
    for( ;; ) {
        vPrintString( pcTaskName ); //태스크 이름을 출력
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ ); //일정 시간 대기
    }
}
void loop() {}
```

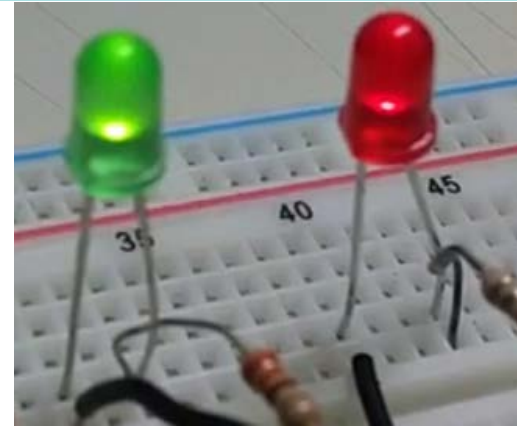
- 앞의 예제와 같이 태스크가 동일한 기능을 한다면 하나의 함수를 여러 개의 태스크로 동시 실행 가능
- 앞의 예제의 Task1과 Task2를 vTask 함수로 하여 2개의 태스크로 동시 실행(구분을 위한 인자가 다름)



```
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
```

4. FreeRTOS 실습(한 LED는 깜박이고, 다른 LED는 천천히 밝아지게)

```
#include<FreeRTOS_AVR.h>
#include "basic_io_avr.h"
void setup() {
    portBASE_TYPE s1,s2;
    s1=xTaskCreate(vTaskBlink, NULL, configMINIMAL_STACK_SIZE, NULL, 1, NULL);
    s2=xTaskCreate(vTaskFading, NULL, configMINIMAL_STACK_SIZE, NULL, 1, NULL);
    vTaskStartScheduler();
    while(1);
}
void loop() { }
void vTaskBlink(void *pvParameters) { //깜박이게
    pinMode(13,OUTPUT);
    while(1) {
        digitalWrite(13,HIGH);
        vTaskDelay(1000/portTICK_PERIOD_MS);
        digitalWrite(13,LOW);
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
void vTaskFading(void *pvParameters) { //천천히 밝아지게
    pinMode(11,OUTPUT);
    while(1) {
        for(int i=0; i<=255; i+=5) {
            analogWrite(11, i);
            vTaskDelay(30/portTICK_PERIOD_MS);
        }
    }
}
```

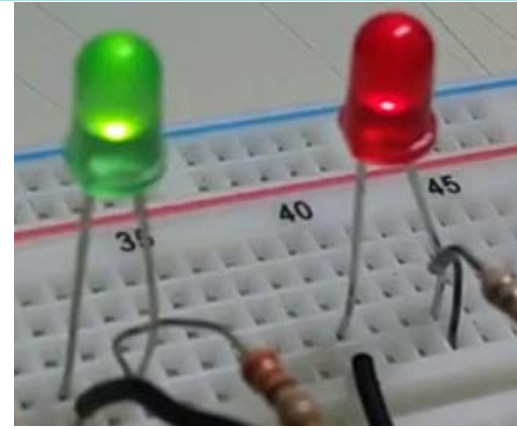


- 만약, 하나의 LED에 Blink와 Fading을 동시에 하게 되면?
- Blink의 13을 11로 수정해서 실행
- 두 가지 동작이 뒤섞여 문제 발생
- 하나의 태스크에서 사용할 때는 다른 태스크에서는 접근하지 못하게
- 멀티태스킹에서 임계영역(Critical Section)이 발생
- 이를 해결하기 위해, 뮤텁스와 세마포어

4. FreeRTOS 실습(한 LED는 깜박이고, 다른 LED는 천천히 밝아지게)

```
#include <FreeRTOS_AVR.h>
SemaphoreHandle_t sem1, sem2;
void setup() {
    portBASE_TYPE s1,s2;
    sem1=xSemaphoreCreateCounting(1,1);
    sem2=xSemaphoreCreateCounting(1,0);
    s1=xTaskCreate(vTaskBlink, NULL, configMINIMAL_STACK_SIZE, NULL, 1, NULL);
    s2=xTaskCreate(vTaskFading, NULL, configMINIMAL_STACK_SIZE, NULL, 1, NULL);
    vTaskStartScheduler();
    while(1);
}
void loop() { }
void vTaskBlink(void *pvParameters) { //깜박이게
    pinMode(11,OUTPUT);
    while(1) {
        xSemaphoreTake(sem1, portMAX_DELAY);
        digitalWrite(11,HIGH);
        vTaskDelay(1000/portTICK_PERIOD_MS);
        digitalWrite(11,LOW);
        vTaskDelay(1000/portTICK_PERIOD_MS);
        xSemaphoreGive(sem2);
    }
}
void vTaskFading(void *pvParameters) { //천천히 밝아지게
    pinMode(11,OUTPUT);
    while(1) {
        xSemaphoreTake(sem2, portMAX_DELAY);
        for(int i=0; i<=255; i+=5) {
            analogWrite(11, i);
            vTaskDelay(30/portTICK_PERIOD_MS);
        }
        xSemaphoreGive(sem1);
    }
}
```

Thread: digitalWrite doesn't work after analogWrite on the same pin



- 11번핀 LED를 두 개의 테스트에서 접근
- 멀티태스킹에서 임계 영역(Critical Section)이 발생
- 세마포어를 이용해 하나만 접근할 수 있게
- Blink 태스크가 끝나고 Fading 태스크가 실행

semphr. h
SemaphoreHandle_t xSemaphoreCreateCounting(
UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);

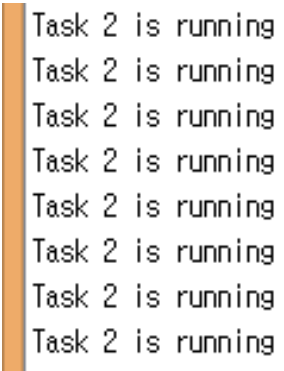
uxMaxCount : The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'given'.

uxInitialCount : The count value assigned to the semaphore when it is created.

4. FreeRTOS 실습(우선 순위가 다른 두 개의 태스크)

```
#include "FreeRTOS_AVR.h"
#define mainDELAY_LOOP_COUNT 400000// ( 0xfffff )
void vTask( void *pvParameters ); //태스크에서 사용할 함수
const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\r\n";
void setup( void ) {
    Serial.begin(9600);
    //같은 함수를 사용하는 두 개의 태스크를 생성, 다른 인자
    xTaskCreate( vTask, "Task 1", 200, (void*)pcTextForTask1, 1, NULL );
    xTaskCreate( vTask, "Task 2", 200, (void*)pcTextForTask2, 2, NULL );
    vTaskStartScheduler(); //스케줄러를 시작
    Serial.println( F("Insufficient RAM") );
    while(1);
}
void vTask ( void *pvParameters ) {
    char *pcTaskName;
    volatile unsigned long ul;
    //인자로 받은 문자열을 char*로 캐스팅해서 저장한다.
    pcTaskName = ( char * ) pvParameters;
    while(1) {
        vPrintString( pcTaskName ); //태스크 이름을 출력
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ ); //일정 시간 대기
    }
}
void loop() {}
```

- 똑같이 두 개의 태스크를 실행하는데 우선순위를 다르게 하면 어떻게 되는지
- 아두이노에 업로드 시켜보며 우선순위가 높은 태스크2만 계속 출력되는 것을 확인할 수 있음.
- 이전 예제들에서는 두 개의 태스크가 모두 우선순위가 같아서 번갈아 문자열이 출력



```
Task 2 is running
Task 2 is running
Task 2 is running
Task 2 is running
Task 2 is running
Task 2 is running
Task 2 is running
Task 2 is running
```

4. FreeRTOS 실습

■ Lab.

- Task 1.

R, G, B 3개의 LED가 각각 1초, 2초, 간격으로 Blink 하고, B는 4초 동안 Fading하며, R은 8번, G는 4번, B는 2번 수행하도록 한다.

4. FreeRTOS 실습

■ LED를 Blink 하면서, 조도값을 시리얼모니터로 송신

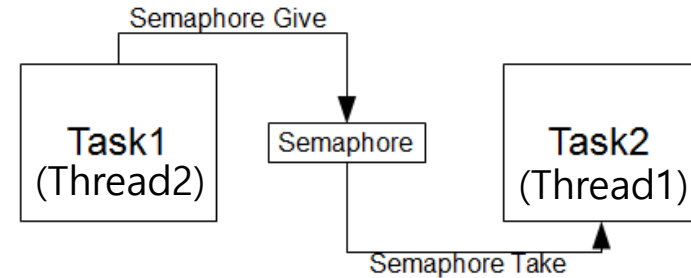
- RTOS를 사용하는 가장 큰 목적이 멀티태스킹 이므로 어떻게 멀티태스킹 환경을 설정하고, 태스크를 만들고, 선점 스케줄러를 시작 시키는가를 보여주기 위해 두개의 독립적인 태스크가 동시에 실행되는 프로그램
- 첫번째는 "LED"라는 이름으로 LED를 일정한 시간 간격으로 깜빡이게 하는 것이고,
- 두번째는 "Light"라는 이름으로 주기적으로 조도 센서의 측정값을 읽어 시리얼 포트에 결과를 출력하는 것이다.
- 두 태스크는 같은 우선순위로 실행되고, 스케줄러는 다른 태스크가 실행될 수 있도록 실행중인 태스크를 선점한다.
- 두 개의 태스크에서 시간을 delay하기 위해 서로 다른 함수를 사용한 것을 알 수 있다. LED task에서는 vTaskDelay() 함수를 사용하는데 비해 Light task에서는 vTaskDelayUntil() 함수를 사용하고 있다.
- 두 함수 모두 스케줄러에 일정 시간동안 delay를 요청하는건 동일하나 vTaskDelay()는 함수가 호출된 시점부터 일정한 tick 동안 태스크 실행을 블럭하는데 비해 vTaskDelayUntil() 함수는 이전에 블럭에서 깨어난 시간인 pxPreviousWakeTime 의 값 부터 xTimeIncrement 만큼만 태스크의 실행을 블럭한다. vTaskDelayUntil() 함수를 사용하면 루프의 내용을 한번 실행하는데 얼마의 시간이 거리건 상관 없이 일정한 시간 간격으로 루프를 실행할 수 있다. 또한 pxPreviousWakeTime 변수는 태스크가 블럭에서 깨어나는 순간 값이 자동으로 업데이트 된다.

```
#include <FreeRTOS_AVR.h>
#define MS(x) (((unsigned long)(x)*configTICK_RATE_HZ)/1000L)
static void Led(void* arg);
static void Light(void* arg);
void setup() {
    portBASE_TYPE s1, s2;
    Serial.begin(9600);
    s1 = xTaskCreate(Led, NULL, 200, NULL, 1, NULL);
    s2 = xTaskCreate(Light, NULL, 200, NULL, 1, NULL);
    vTaskStartScheduler();
    while(1);
}
void loop() {
}
```

```
static void Led(void* arg) {
    boolean ledState = false;
    pinMode(13, OUTPUT);
    while (1) {
        digitalWrite(13, ledState);
        ledState = !ledState;
        vTaskDelay(MS(500));
    }
}
static void Light(void* arg) {
    short l = 0;
    TickType_t xLastWakeTime = xTaskGetTickCount();
    const TickType_t xWakePeriod = 2000/portTICK_PERIOD_MS; //2초
    while (1) {
        l = analogRead(0);
        Serial.print("Light: ");
        Serial.println(l);
        vTaskDelayUntil(&xLastWakeTime, xWakePeriod);
    }
}
```

4. FreeRTOS 실습(LED blink 0.2초간격), 예제-frblink

```
#include <FreeRTOS_AVR.h>
const uint8_t LED_PIN = 13;
SemaphoreHandle_t sem; //세마포어 핸들 선언
static void Thread1(void* arg) {
    while (1) {
        //세마포어를 받을때까지 스레드1은 계속 기다리게 됩니다.
        xSemaphoreTake(sem, portMAX_DELAY); //쓰레드 2의 신호를 기다림
        digitalWrite(LED_PIN, LOW); //LED끄기
    }
}
static void Thread2(void* arg) {
    pinMode(LED_PIN, OUTPUT);
    while (1) {
        digitalWrite(LED_PIN, HIGH); //LED 켜기
        vTaskDelay((200L * configTICK_RATE_HZ) / 1000L); //200ms대기
        xSemaphoreGive(sem); //쓰레드 1에 신호를 보냄
        vTaskDelay((200L * configTICK_RATE_HZ) / 1000L); //200ms대기
    }
}
void setup() {
    portBASE_TYPE s1, s2;
    Serial.begin(9600);
    sem = xSemaphoreCreateCounting(1, 0); //세마포어 생성과 초기화
    s1 = xTaskCreate(Thread1, NULL, configMINIMAL_STACK_SIZE, NULL, 2, NULL); //Thread1 생성(우선순위 2)
    s2 = xTaskCreate(Thread2, NULL, configMINIMAL_STACK_SIZE, NULL, 1, NULL); //Thread2 생성(우선순위 1)
    vTaskStartScheduler(); //태스크 스케줄러 시작
    while(1);
}
void loop() {} //FreeRTOS 사용 시 loop 함수는 사용하지 않음
```



- thread2에서는 13번 LED를 켜고 200ms 기다린 뒤에 세마포어를 thread1로 넘겨주게 됨.
- 세마포어를 기다리다가 세마포어를 받은 thread1은 13번 LED를 끄는 작업을 수행하고 while(1)에 의해 세마포어를 기다리는 형태가 됨.
- 한편 thread2에서는 세마포어를 전달하고 200ms를 기다린 뒤에 다시 13번 LED를 켜 다음 200ms뒤에 세마포어를 thread1에 전달하는 작업을 반복.
- 이렇게 2개의 스레드의 반복적인 작업을 통해 13번 LED는 깜빡임을 반복하게 됨.

4. FreeRTOS 실습

■ Lab.

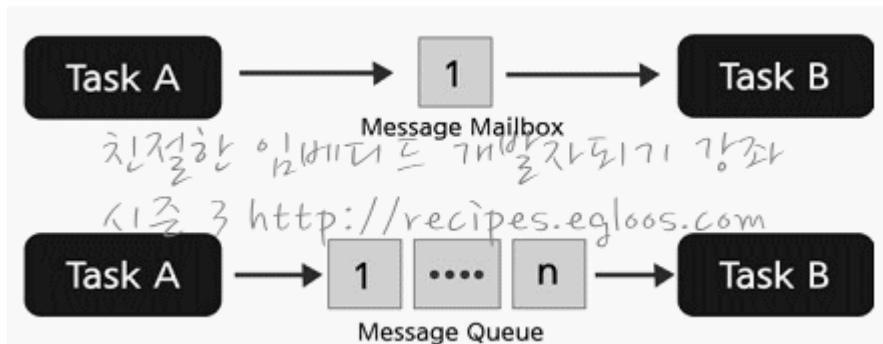
- Task 1.

R, G, B 3개의 LED가 각각 1초, 2초, 간격으로 Blink 하고, B는 4초 동안 Fading하며, R은 8번, G는 4번, B는 2번 수행하도록 한다.

단, 한 순간 LED 중 하나만 Blink 할 수 있다.

4. FreeRTOS- Queue

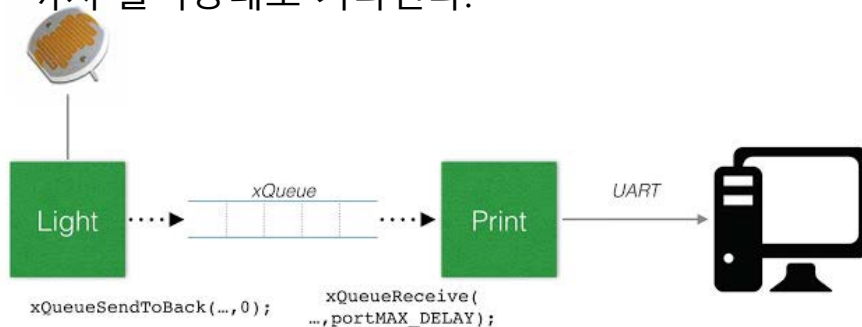
- FreeRTOS의 Task는 서로 독립된 Task로 서로 데이터를 주고 받으며(통신을 하며) 일을 처리 하여야 하는 경우가 많다. Queue(큐)는 Task 간 통신과 동기화에 이용 된다.
- Queue의 특성
 - A. Data Storage
 - Queue는 일정한 크기의 유한한 Data를 저장 한다. Queue 가 저장할 수 있는 데이터의 최대 개수를 Queue Length라고 한다. Queue의 Length와 Size는 Queue가 생성될 때 설정 된다.
 - 보통 Queue는 First In First Out(FIFO) 버퍼로 사용 된다. 그러나 Queue의 앞 부분에 Data를 쓰는 것도 가능 하다.
 - Queue는 어떤 특별한 Task에 소유 되지 않기 때문에 여러 개의 Task에서 같은 Queue에 데이터를 쓸 수 있다. 그리고 같은 Queue에서 여러 개의 Task가 데이터를 읽을 수 있다. 그러나 동일한 Queue에서 여러 개의 Task가 데이터를 읽는 경우는 드문 일 이다.
 - Task가 Queue로부터 데이터를 읽을 때 필요에 따라 옵션으로 Block Time을 선언 할 수 있다. 이 경우 Queue에 읽을 수 있는 데이터가 준비 될 때까지 Task는 Blocking State가 된다. Task의 Blocking 상태는 다른 Task나 Interrupt에서 Queue에 데이터를 저장 하면 자동으로 해제 되어 Ready 상태가 된다. 또한 Queue 가 비어 있는 상태에서 지정한 Block Time 이 지난 경우에도 Ready 상태가 된다. 만약 여러 개의 Task가 Queue로부터 데이터를 받기 위하여 Blocked 상태에 있는 경우에 Queue에 새로운 데이터가 하나 도달 하면 가장 우선 순위(Priority)가 높은 하나의 Task만 Unblocked 상태(Ready 상태)로 된다. 만약 동일한 Priority를 갖는 Task가 여러 개 대기 중일 경우에는 가장 오래 대기한 Task가 Unblocked 된다.



두 개의 차이점은 메시지를 하나만 가지고 있을지와 여러 개를 가지고 있을지 랍니다. 여기서 하나의 메시지는 워드 크기로 구별된답니다.

4. FreeRTOS- Queue

- 두개의 태스크가 동시에 실행되면서 서로간에 데이터를 주고 받는 예제
- Light 태스크는 측정한 조도 값을 시리얼 포트를 통해 출력했는데, 이번에는 시리얼 포트로 출력하는 대신 측정 값을 Print라는 이름의 태스크로 보낸다. Print 태스크는 데이터가 들어오기를 기다렸다가 들어온 내용을 시리얼 포트로 출력한다.
- Print 태스크는 UART를 초기화 한 후 큐에 데이터가 들어오기를 기다리는 무한 루프를 돌면서 데이터가 들어오면 그 내용을 읽어 출력한다. Light 태스크는 이전 예제와 같이 무한 루프를 돌면서 조도 값을 측정하여 그 값을 큐에 집어 넣는다.
- Light 태스크는 큐가 풀린 경우 기다리지 않지만, Print 태스크는 큐가 비어 있는 경우 데이터가 들어올 때까지 블럭상태로 기다린다.



```

#include <FreeRTOS_AVR.h>
#define MS(x) ((unsigned long)(x)/portTICK_PERIOD_MS)
QueueHandle_t xQueue;
static void Print(void* arg);
static void Light(void* arg);

void setup() {
    Serial.begin(9600);
    xQueue = xQueueCreate(5, sizeof(uint16_t));
    xTaskCreate(Print, NULL, 200, NULL, 1, NULL);
    xTaskCreate(Light, NULL, 200, NULL, 2, NULL);
    vTaskStartScheduler();
    while(1);
}

void loop() { }
  
```

```

static void Light(void* arg) {
    uint16_t l = 0;
    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();
    while (1) {
        l = analogRead(A0);
        xQueueSendToBack(xQueue, &l, 0);
        vTaskDelayUntil( &xLastWakeTime, (2000/portTICK_PERIOD_MS));
    }
}

static void Print(void* arg) {
    uint16_t light;
    uint16_t sampleCount = 0;
    while (1) {
        if (xQueueReceive(xQueue, &light, portMAX_DELAY)) {
            sampleCount++;
            Serial.print("Sample ");    Serial.print(sampleCount);
            Serial.print(" Light = ");  Serial.println(light);
        }
    }
}
  
```

4. FreeRTOS- Queue

From Sender 2 = 200

Queue is Full

From Sender 1 = 100

Queue is Full

From Sender 2 = 200

큐에 값을 넣는 태스크를 두 개 선언하는데 파라미터로 자신을 나타내는 ID와 출력할 값을 구조체에 넣어 넘김. 다른 하나의 태스크가 구조체를 꺼내어 ID와 값을 출력.

```
#include "FreeRTOS_AVR.h"
// #include "task.h"
// #include "queue.h"
/* Demo includes. */
#include "basic_io_avr.h"
#define mainSENDER_1 1
#define mainSENDER_2 2
static void vSenderTask( void *pvParameters );
static void vReceiverTask( void *pvParameters );
QueueHandle_t xQueue; // 큐에 접근한 핸들이 저장됨
typedef struct { // 큐에 넣을 때 사용될 구조체 타입을 정의
    unsigned char ucValue;
    unsigned char ucSource;
} xData;
// xData 구조체 타입의 두 개의 변수를 선언
static const xData xStructsToSend[ 2 ] = {
    { 100, mainSENDER_1 }, // Sender1에서 사용됨
    { 200, mainSENDER_2 } // Sender2에서 사용됨
};
void setup( void ) {
    Serial.begin(9600);
    // xData 타입의 값을 최대 3개 넣을 수 있는 큐를 생성함.
    xQueue = xQueueCreate( 3, sizeof( xData ) );
    if ( xQueue != NULL ) {
        // 큐에 값을 넣을 태스크를 두 개를 똑같이 우선순위 2로 생성
        // 각각 다른 파라미터를 넘겨줌. (자신의 ID와 출력할 값)
        xTaskCreate( vSenderTask, "Sender1", 200, ( void * ) &(
            xStructsToSend[ 0 ] ), 2, NULL );
        xTaskCreate( vSenderTask, "Sender2", 200, ( void * ) &(
            xStructsToSend[ 1 ] ), 2, NULL );
        // 큐에서 데이터를 가져올 태스크를 우선순위 1로 생성
        xTaskCreate( vReceiverTask, "Receiver", 200, NULL, 1, NULL );
        // 스케줄러 시작
        vTaskStartScheduler();
    }
    else
    { // 메모리 부족으로 큐가 생성되지 못함
        // 메모리 부족으로 여기가 실행됨
        for( ;; );
    }
}
```

```
static void vSenderTask( void *pvParameters ) {
    portBASE_TYPE xStatus;
    const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;
    for( ;; ) { // 인자(값이 저장될 큐 핸들, 큐에 넣을 구조체의 주, 블록되는 시간)
        // 블록되는 시간으로 우선순위 낮은 Receiver 태스크가 실행될 수 있도록 함
        xStatus = xQueueSendToBack( xQueue, pvParameters, xTicksToWait );
        if ( xStatus != pdPASS ) {
            vPrintString( "Queue is Full\r\n" );
        }
        // 다른 sender 태스크가 실행되도록 문맥교환을 강제로 함.
        taskYIELD();
    }
}
static void vReceiverTask( void *pvParameters ) {
    xData xReceivedStructure;
    portBASE_TYPE xStatus;
    for( ;; ) { // sender 태스크들이 큐가 꽉차서 블록된 상태일때에만 이 태스크가 실행
        if ( uxQueueMessagesWaiting( xQueue ) == 3 ) {
            vPrintString( "Queue is Full\r\n" );
        }
        // 인자(큐 핸들, 꺼내온 데이터를 저장할 구조체의 주소, 비어있을 경우 대기시간)
        xStatus = xQueueReceive( xQueue, &xReceivedStructure, 0 );
        if ( xStatus == pdPASS ) {
            if ( xReceivedStructure.ucSource == mainSENDER_1 )
                vPrintStringAndNumber( "From Sender 1 = ", xReceivedStructure.ucValue );
            else
                vPrintStringAndNumber( "From Sender 2 = ", xReceivedStructure.ucValue );
        }
        else
            vPrintString( "Queue is Empty\r\n" );
    }
}
void loop() {}
```

4. FreeRTOS 실습

■ Lab.

- Task 1.

온습도 센서로 부터 2초 간격으로 센싱하고, 시리얼 모니터에는 4초 간격으로 출력하도록 한다.