프로젝트

Name: 영문자 대문자로 시작

Package name: 영문자 소문자로 시작

Save location: 영문으로 지정

Language: Java

TextView 태그의 text 속성을 바꾸면 글자를 바꿀 수 있음

버튼을 눌렀을 때 메시지가 나타나게 하기

XML 레이아웃 파일의 버튼에 onClick 속성 값 넣기

소스파일에 이벤트 처리 함수 추가하기

xml

android:onClick=”onButton1Clicked”

java

public void onButton1Clicked(View v) {

Toast.makeText(this, “확인1 버튼이 눌렸어요.”, Toast.LENGTH\_LONG).show();

}

더 많은 활용

public void onButton2Clicked(View v) {

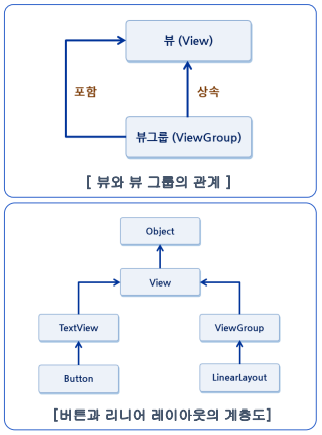
Intent myIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse(“http://m.naver.com”));

startActivity(myIntent);

}

public void onButton3Clicked(View v) {

Intent myIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse(“tel:010-1000-1000”));

 startActivity(myIntent);

}

뷰(View)

화면에 보이는 모든 것, UI 구성 요소

뷰 그룹(View Group)

뷰들을 여러 개 포함하고 있는 것

뷰 그룹도 뷰에서 상속하여 뷰가 됨

위젯(Widget)

뷰 중에서 일반적인 컨트롤의 역할을 하고 있는 것

버튼, 텍스트 등등

레이아웃(Layout)

뷰 그룹 중에서 내부에 뷰들을 포함하고 있으면서

그것들을 배치하는 역할을 하는 것

XML 레이아웃

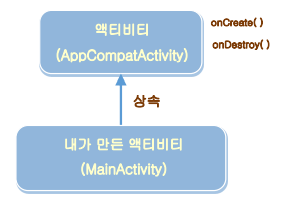
뷰 태그와 속성으로 구분됨



상속

객체지향의 가장 기본적인 개념 중 하나

부모의 특성을 그대로 물려받는 것으로 변수나 메소드 재사용 가능



액티비티의 상속

extends 키워드 사용

public class MainActivity extends AppCompatActivity

부모 클래스의 메소드를 재정의

onCreate() 메소드는 이미 부모 클래스에 정의되어 있음

기능을 추가하고 싶을 때 재정의(Override)

this와 super

나 자신은 this, 부모는 super를 사용하여 변수나 메소드 참조

super.onCreate( … );

제약 레이아웃(ConstraintLayout)

제약 조건(Constraint) 기반 모델

제약 조건을 사용해 화면을 구성하는 방법

안드로이드 스튜디오에서 자동으로 설정하는 디폴트 레이아웃

리니어 레이아웃(LinearLayout)

박스(Box) 모델

한 쪽 방향으로 차례대로 뷰를 추가하며 화면을 구성하는 방법

뷰가 차지할 수 있는 사각형 영역을 할당

상대 레이아웃(RelativeLayout)

규칙(Rule) 기반 모델

부모 컨테이너나 다른 뷰와의 상대적 위치로 화면을 구성하는 방법

프레임 레이아웃(FrameLayout)

싱글(Single) 모델

가장 상위에 있는 하나의 뷰 또는 뷰그룹만 보여주는 방법

여러 개의 뷰가 들어가면 중첩하여 쌓게 됨. 가장 단순하지만 여러 개의 뷰를 중첩한 후 각 뷰를 전환하여 보여주는 방식으로 자주 사용함

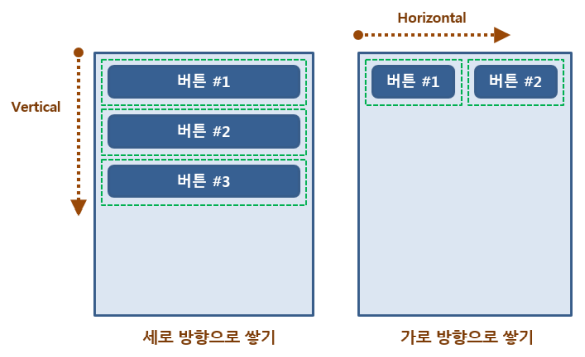
테이블 레이아웃(TableLayout)

격자(Grid) 모델

격자 모양의 배열을 사용하여 화면을 구성하는 방법

HTML에서 많이 사용하는 정렬 방식과 유사하지만 많이 사용하지는 않음

리니어 레이아웃 사용방식



Vertical 세로 방향으로 쌓기

Horizontal 가로 방향으로 쌓기

리니어 레이아웃 뷰 정렬하기

두 가지 정렬 속성

layout\_gravity [외부] 부모 컨테이너의 여유 공간에 뷰가 모두 채워지지 않아 여유 공간 안에서 뷰를 정렬할 때

gravity [내부] 뷰에서 화면에 표시하는 내용물을 정렬할 때, 텍스트 뷰의 내용물은 글자, 이미지 뷰의 내용물은 이미지가 됨

리니어 레이아웃 정렬을 위해 사용할 수 있는 값

top – 객체를 위쪽 끝

bottom – 객체를 아래쪽 끝

left – 객체를 왼쪽 끝

right – 객체를 오른쪽 끝

center\_vertical – 객체를 수직 방향의 중앙

center\_horizontal – 객체를 수평 방향의 중앙

fill\_vertical - 객체를 수직 방향으로 여유 공간만큼 확대하여 채우기

fill\_horizontal – 객체를 수평 방향으로 여유 공간만큼 확대하여 채우기

center – 객체를 수직 방향과 수평 방향의 중앙

fill – 객체를 수직 방향과 수평 방향의 여유 공간만큼 확대하여 채우기

clip\_vertical – 객체의 상하 길이가 여유 공간보다 클 경우 남는 부분을 잘라내기

- top|clip\_vertical 아래쪽 남는 부분 잘라내기, bottom|clip\_vertical 위쪽 남는 부분 잘라내기

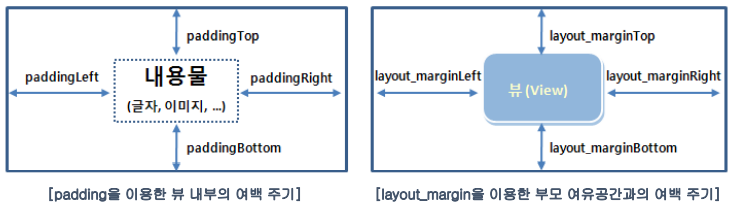
- center\_vertical|clip\_vertical 위쪽과 아래쪽에 남는 부분 잘라내기

clip\_horizontal – 객체의 좌우 길이가 여유 공간보다 클 경우 남는 부분을 잘라내기

- right|clip\_horizontal 왼쪽에 남는 부분 잘라내기, left|clip\_horizontal 오른쪽 남는 부분 잘라내기

- center\_horizontal|clip\_horizontal 왼쪽과 오른쪽에 남는 부분 잘라내기

리니어 레이아웃 – 마진과 패딩 설정



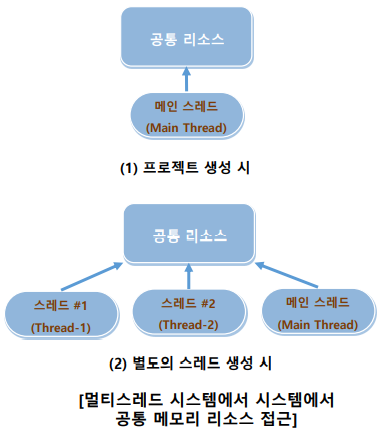
padding 속성

뷰 안의 내용물이 텍스트나 이미지와 뷰 안의 영역 사이의 여백을 줄 수 있는 방법

layout\_margin 속성

부모 컨테이너와 여유 공간과 뷰 사이의 여백을 줄 수 있는 방법

멀티 스레드



메인 액티비티

애플리케이션이 실행될 때 하나의 프로세스에서 처리

이벤트를 처리하거나 필요한 메소드를 정의하여 기능을 구현하는 경우에도 동일한 프로세스 내에서 실행

문제점

대기 시간이 길어지는 네트워크 요청 등의 기능을 수행할 때는 화면에 보이지 않는 UI도 멈춤 상태로 있게 됨

해결 방안

하나의 프로세스 안에서 여러 개의 작업이 동시에 수행되는 멀티 스레드 방식을 사용

멀티 스레드

같은 프로세스 안에 들어 있으면서 메모리 리소스를 공유하게 되므로 효율적인 처리가 가능

동시에 리소스를 접근할 경우 문제 발생

안드로이드는 main스레드에서만 UI 접근 가능

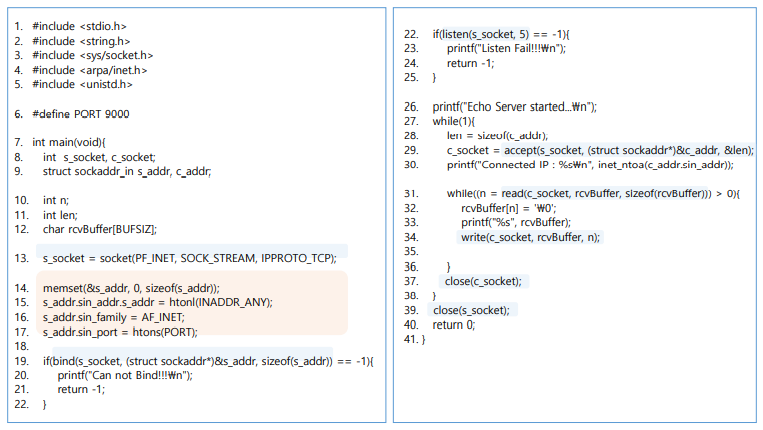
핸들러 사용하기

스레드 사용

스레드는 동일 프로세스 내에 있기 때문에 수행의 결과를 바로 처리할 수 있음

그러나 UI 객체는 직접 접근할 수 없으므로 핸들러(Handler) 객체를 사용함

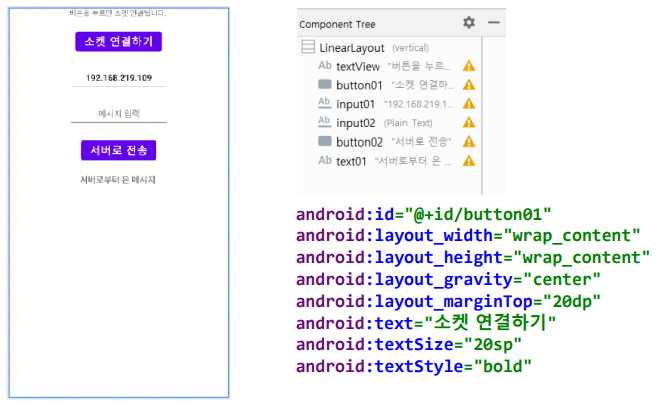
안드로이드 소켓 통신 예제\_서버



에코 서버를 시작하고

클라이언트로부터 받아온 메시지를 출력한다

안드로이드 소켓 통신 예제\_클라이언트





핸들러 = UI 객체는 직접 접근할 수가 없기에 핸들러 객체를 사용함

Handler handler = new Handler();

서버 주소로 소켓 연결

String addr = input01.getText().toString().trim();

thread = new ConnectThread(addr);

thread.start();

서버로부터 받은 메시지

msg = input02.getText().toString().trim();

new Thread() {

public void run() {

out.println(msg);

out.flush();

thread.readServer();

}

}.start();

메시지가 bye면 종료

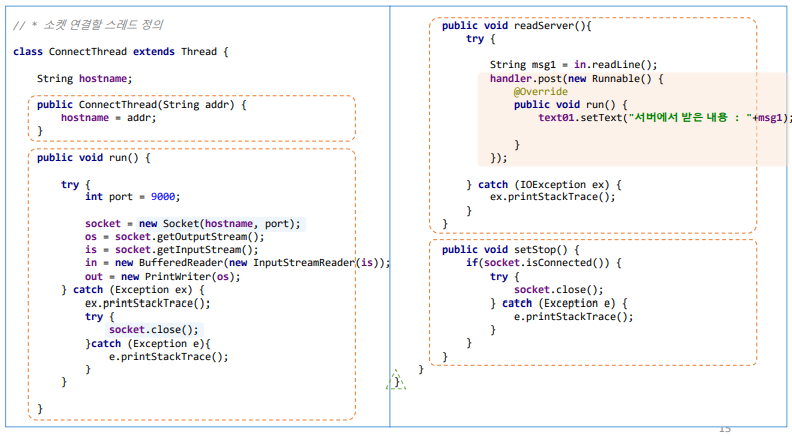
if (msg.equals(“bye”)) {

thread.setStop();

}

input02의 내용 초기화

input02.setText(“”);



소켓 연결할 스레드 정의

서버에서 보낸 메시지 읽어오기

readServer() {

String msg1 = in.readLine();

handler.post(new Runnable() {

@Override

public void run() {

text01.setText(“서버에서 받은 내용 : “ + msg1);

}

});

}

서버 종료하기

setStop() {

if(socket.isConnected()) {

socket.close();

}

}

AndroidManifest

설치된 앱의 구성요소가 어떤 것인지, 어떤 권한이 부여되었는지 시스템에게 알려줌

<uses-permission android:name=”android.permission.INTERNET”/>

추가해야 함

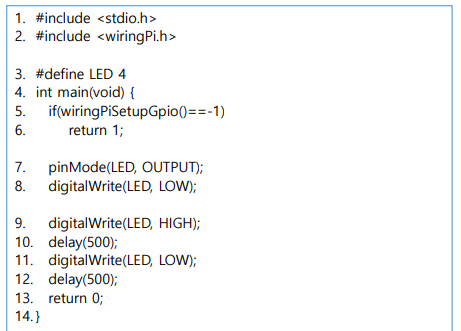
네트워킹 사용 시 주의할 점

네트워킹을 사용할 때에는 반드시 스레드 사용

UI 업데이트를 위해서는 반드시 핸들러 사용

안드로이드로 LED 켜고 끄기

라즈베리파이\_LED 제어



#define LED 4

int main(void) {

pinMode(LED, OUTPUT);

digitalWrite(LED, LOW);

digitalWrite(LED, HIGH);

delay(500);

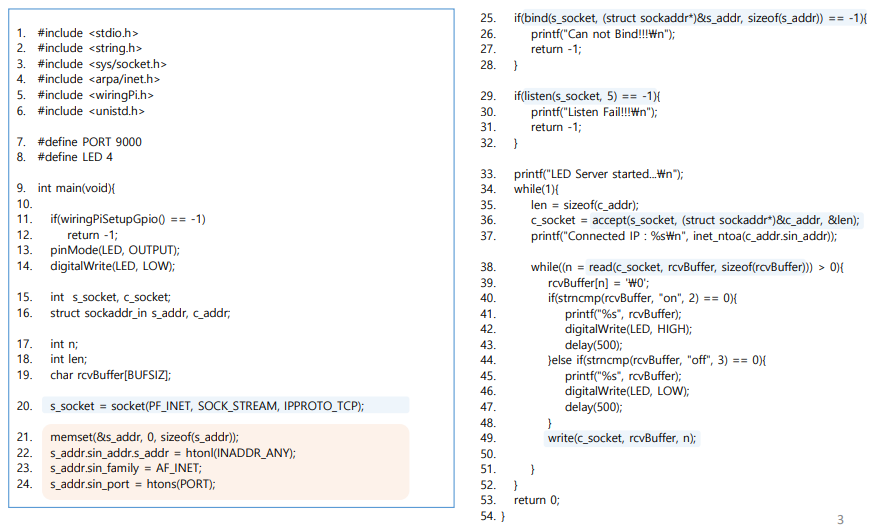
digitalWrite(LED, LOW);

delay(500);

return 0;

}

LED 소켓 통신\_서버 (led\_server.c)



포트번호 9000번과 받아온 아이피로 서버 소켓을 생성하고 연결

LED의 pinMode를 OUTPUT으로 설정하고 digitalWrite LOW 상태로 초기화

클라이언트로부터 받아온 메시지가 있으면 동작

c\_socket으로부터 받아온 rcvBuffer

받은 메시지가 on이면 on을 프린트하고 LED 전원을 켠 후 0.5초 대기

off이면 off를 프린트하고 LED 전원을 끈 후 0.5초 대기

c\_socket에 rcvBuffer의 내용을 보낸다

while((n = read(c\_socket, rcvBuffer, sizeof(rcvBuffer))) > 0) {

rcvBuffer[n] = ‘\0’;

if(strncmp(rcvBuffer, “on”, 2) == 0){

printf(“%s”, rcvBuffer);

digitalWrite(LED, HIGH);

delay(500);

else if(strncmp(rcvBuffer, ‘off’, 3) == 0){

printf(“%s”, rcvBuffer);

digitalWrite(LED, LOW);

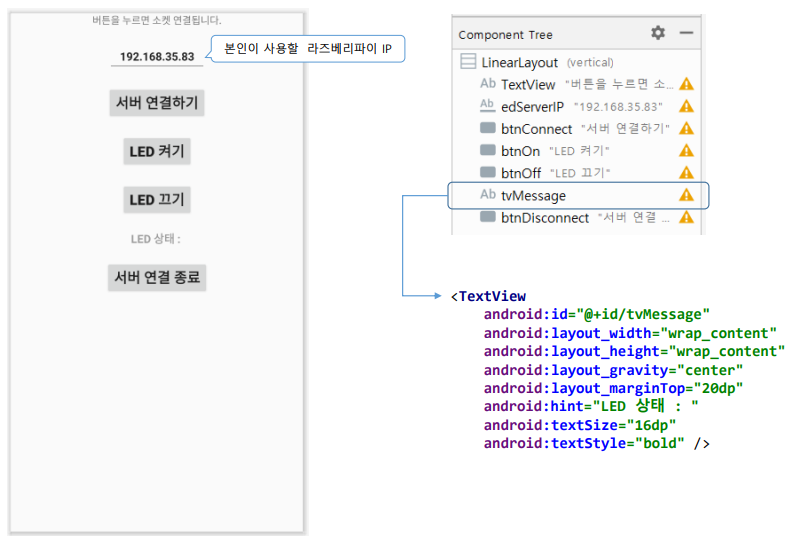
delay(500);

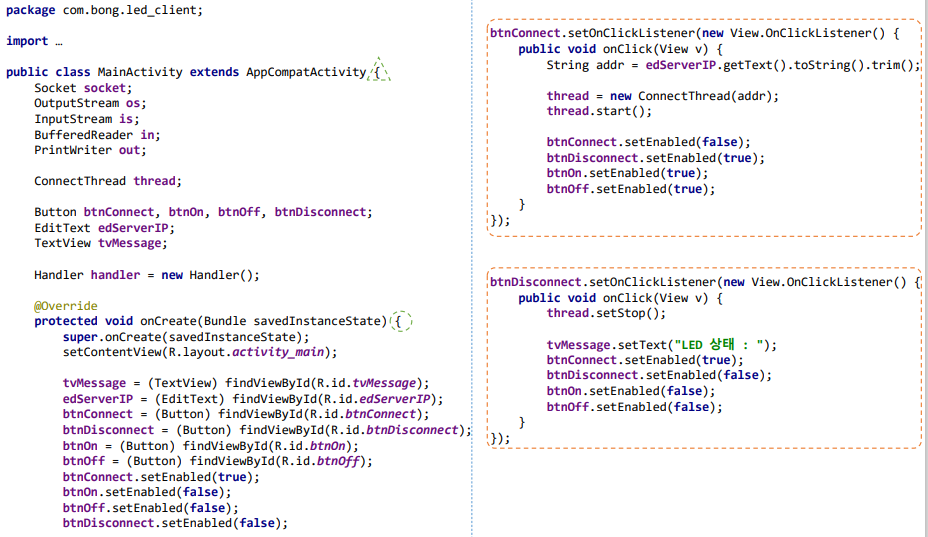
}

write(c\_socket, rcvBuffer, n);

}

안드로이드 소켓 통신 예제\_클라이언트





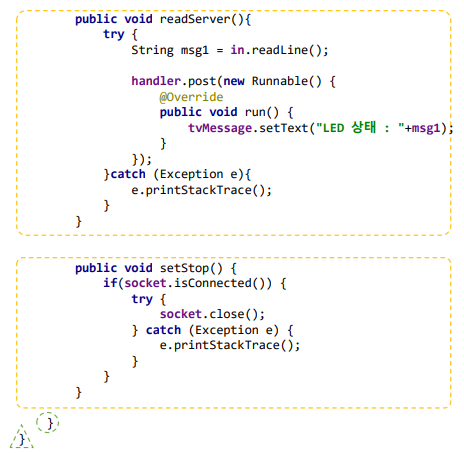


LED켜기 버튼을 누르면 btnOn의 동작이 실행

메시지 on을 OutPutStream으로 flush 해주고 readServer 실행

LED끄기 버튼을 누르면 btnOff의 동작이 실행

메시지 off를 OutPutStream으로 flush 해준다 readServer 실행



서버로부터 받아온 메시지가 msg1에 저장

handler 이용해서 UI 업데이트

public void readServer() {

try {

String msg1 = in.readLine();

handler.post(new Runnable() {

@Override

public void run() {

tvMessage.setText(“LED 상태 : “ + msg1);

}

});

} catch (Exception e) {

e.printStackTrace();

}

}

AndroidManifest에 <uses-permission android:name”android.permission.INTERNET”/> 추가

안드로이드\_초음파센서

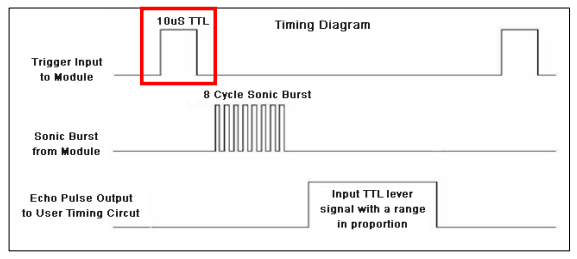
초음파 센서 동작 방식

초음파를 발생하고 물체에 부딪혀 돌아오는 시간을 측정함으로써 거리를 센싱

Trig 핀 출력 신호와 Echo 핀 입력 신호를 이용

최소 10us HIGH 레벨 신호를 통한 IO Trigger를 사용하며, 모듈은 자동으로 8번의 40kH 신호를 보내고 펄스 신호가 돌아오는지 여부를 검출

타이밍 다이어그램



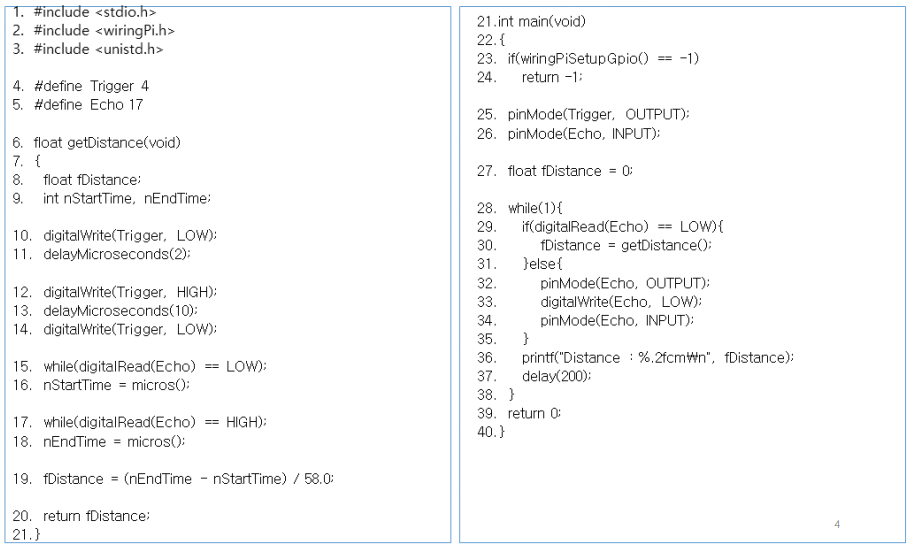
센싱을 시작할 때, 트리거 입력에 짧은 10uS 펄스를 제공

모듈은 40kHz의 초음파 버스트를 8차례 전송하고, 그것의 에코를 수신

전송하는 트리거 신호와 받는 에코 신호 사이의 시간 간격을 통하여 그 범위를 계산

센싱이 시작되고 다음 센싱이 시작되기까지의 간격, 즉, 측정 주기는 60ms 이상으로 사용하며, 이는 에코 신호 중에 트리거 신호가 전달되는 것을 방지하기 위함

라즈베리파이\_초음파센서



#define Trigger 4

#define Echo 17

float fDistance;

int nStartTime, nEndTime;

digitalWrite(Trigger, LOW);

delayMicroseconds(2);

digitalWrite(Trigger, HIGH);

delayMicroseconds(10);

digitalWrite(Trigger, LOW);

while(digitalRead(Echo) == LOW); echo가 HIGH가 될 때까지 기다림

nStartTime = micros(); HIGH가 되면 그때의 시간을 microseconds 단위로 받아옴

while(digitalRead(Echo) == HIGH); echo가 LOW가 될 때까지 기다림

nEndTime = micros(); LOW가 되면 그때의 시간을 microseconds 단위로 받아옴

fDistance = (nEndTime – nStartTime) / 58.0; nEndTime과 nStartTime을 뺌으로써 echo 핀이 HIGH였던 시간을 계산함

Trigger 핀 출력 신호

Echo 핀 입력 신호

pinMode(Trigger, OUTPUT);

pinMode(Echo, INPUT);

while(1) {

if(digitalRead(Echo) == LOW) {

fDistance = getDistance();

}else{

pinMode(Echo, OUTPUT);

digitalWrite(Echo, LOW);

pinMode(Echo, INPUT);

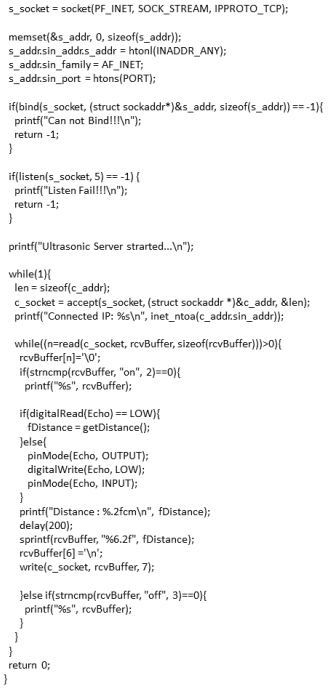
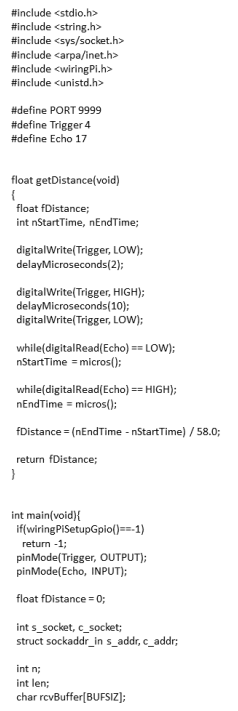
}

printf(“Distance : %.2fcm\n”, fDistance);

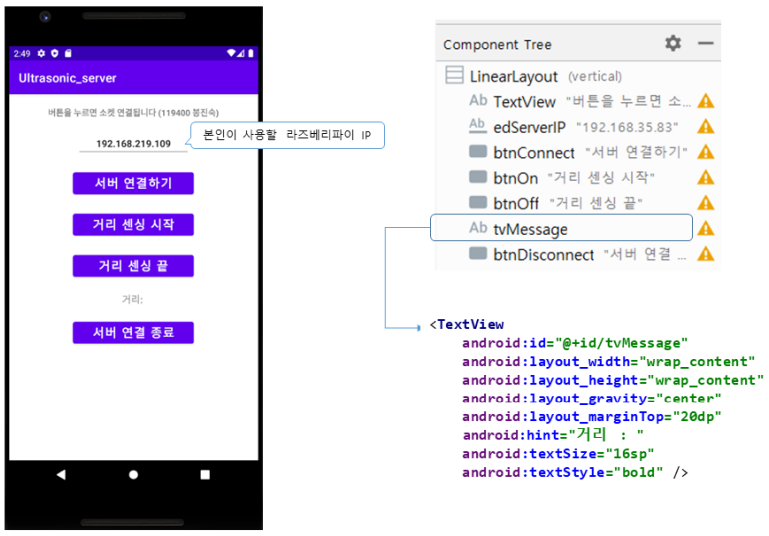
delay(200);

}

초음파 센서\_서버



초음파 센서\_클라이언트 (ultrasonic\_client)





Timer timer;

Boolean threadStop;

timer는 1초마다 반복하기 위한 timer.schedule(); 을 사용하기 위함

btnConnect 동작하면 threadStop = false;



btnOn 동작하면 timer 객체를 새로 생성

timer.schedule(new TimerTask() {}, 1000, 1000); 1초마다 반복

timer.schedule(new TimerTask() {

@Override

public void run() {

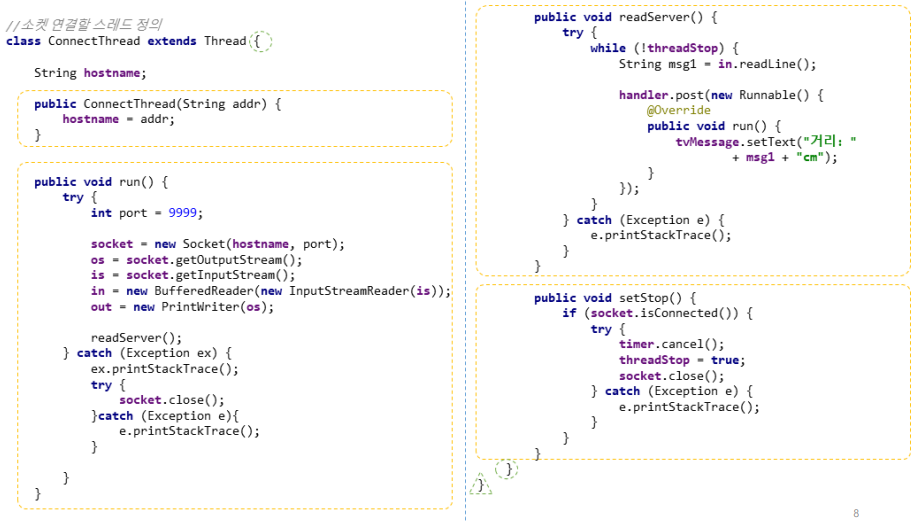
out.println(“on”);

out.flush();

}

}, 1000, 1000);

}



setStop에서 threadStop를 true로

안드로이드\_온습도 센서

온/습도 센서

온도와 습도 등을 측정하는 센서

온/습도 센서에서 받아들인 정보는 하드웨어를 통하여 디지털화된 결과를 도출

SHT11(온/습도) 센서

디지털 출력 온/습도 센서인 SHT11은 하나의 칩에 두 개의 센서를 집적화한 제품으로 특징은 다음과 같음

보정된 디지털 출력 (2 와이어 인터페이스)

빠른 반응속도

저전력 소비

낮은 가격

대량 생산용으로 디자인 됨, 가격이 민감한 응용 제품용

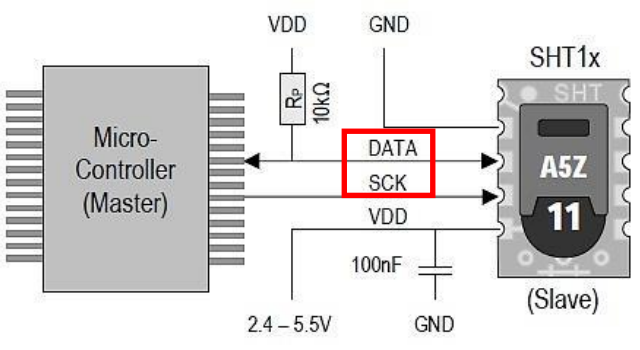
뛰어난 장기 안정성

보정과 디지털 2 와이어 인터페이스 기능으로 사용이 쉬움

온/습도 센서(내부의 A/D 변환기)와 라즈베리파이의 통신

센싱된 온습도의 아날로그 수치는 14비트 A/D 변환기를 통해 디지털로 수치화하고 I2C를 통하여 정보를 처리할 MCU(라즈베리파이)로 전달

일반적인 회로구성 예



온/습도 센서 동작 방식

센서가 전원을 입력 받으면 온/습도를 센싱하는 하드웨어를 통해 아날로그 정보를 측정

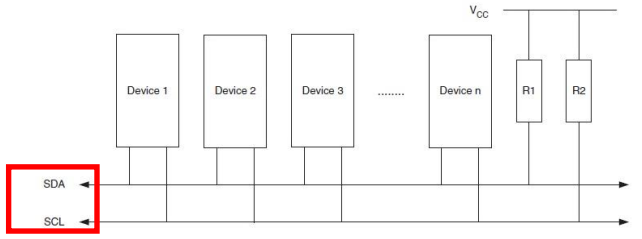
측정된 정보는 센서 자체가 가진 A/D 변환 기능을 통해 디지털 수치로 변환

변환된 수치는 직렬 버스를 통해 CPU로 전달

2-Wire 직렬 인터페이스(Two-wire Serial Interface, TWI)

TWI 프로토콜은 단지 2개의 양방향 버스 라인, 클럭(SCL)을 위한 하나의 버스와 데이터(SDA)를 위한 또 하나의 버스를 이용한 128개 이상 다른 장치의 상호연결을 위한 시스템 디자이너를 허용

디바이스를 구분하기 위해 주소 개념 사용



TWI(I2C) 특징

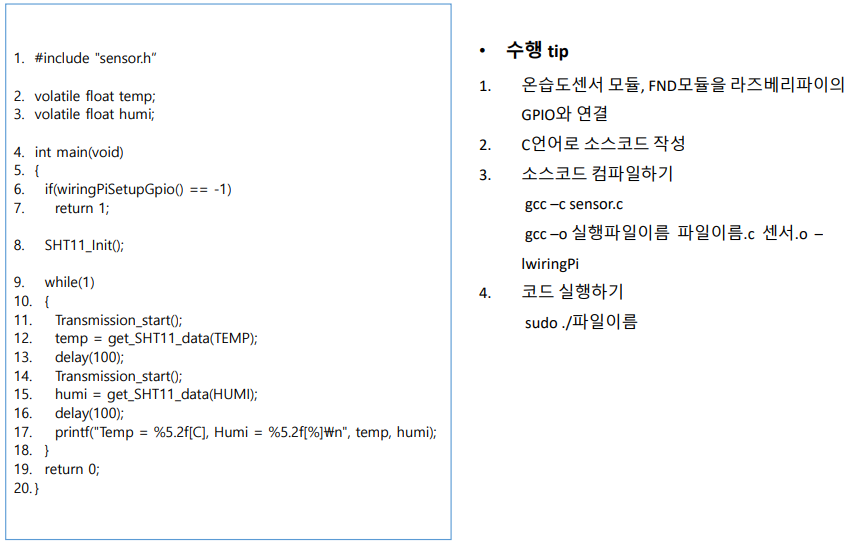
단순하지만 파워풀하고 유연한 통신 인터페이스, 단지 2개 버스 라인 필요

마스터와 슬레이브 동작을 지원

장치는 전송자와 수신자로써 동작할 수 있음

클럭 신호는 단방향이며, 마스터에 의해 발생되고 데이터 신호는 양방향으로 데이터를 송수신하기 위해 사용

라즈베리파이\_동작 센서



volatile float temp;

volatile float humi;

SHT11\_Init();

Transmission\_start();

temp = get\_SHT11\_data(TEMP);

delay(100);

Transmission\_start();

humi = get\_SHT11\_data(HUMI);

delay(100);

printf(“Temp = %5.2f[C], HUMI = %5.2f[%]\n”, temp, humi);

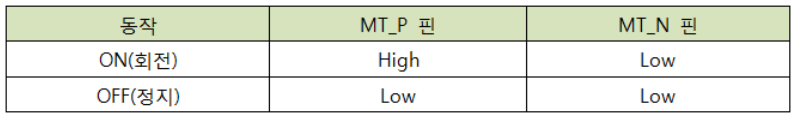
Fan 개념 및 제어방법

전력(전기적 에너지)을 회전운동의 힘(기계적 에너지)으로 바꾸어 바람, 즉 공기의 흐름을 만드는 기계로서, 전력을 공급하면 회전 날개를 회전

선풍기에 FAN이 포함되고, 컴퓨터도 칩의 열을 식혀 주기 위해 작은 FAN이 들어 있으며, 내부의 공기를 환기하는 목적의 FAN도 있음

FAN의 동작은 ON/OFF 두 가지

Fan 모듈의 FAN 포트 각 핀에 입력되는 신호에 따른 동작



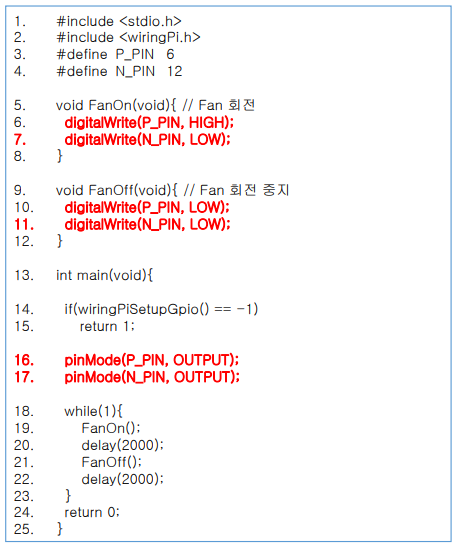
동작 ON(회전)

MT\_P 핀 High MT\_N 핀 Low

동작 OFF(정지)

MT\_P 핀 Low MT\_N 핀 Low

라즈베리파이\_FAN



#define P\_PIN 6

#define N\_PIN 12

void FanOn(void){

digitalWrite(P\_PIN, HIGH);

digitalWrite(N\_PIN, LOW);

}

void FanOff(void){

digitalWrite(P\_PIN, LOW);

digitalWrite(N\_PIN, LOW);

}

pinMode(P\_PIN, OUTPUT);

pinMode(N\_PIN, OUTPUT);

while(1){

FanOn();

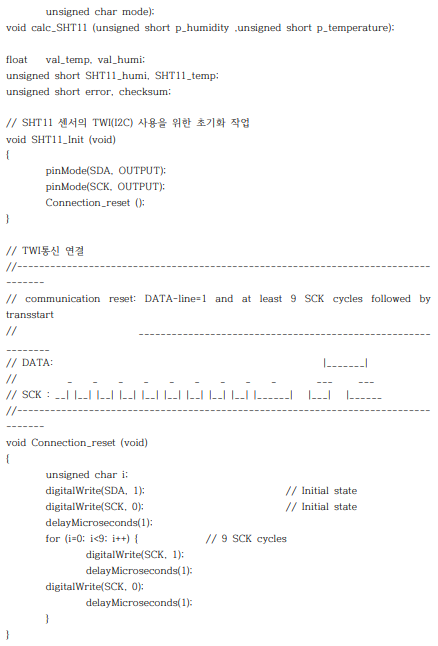
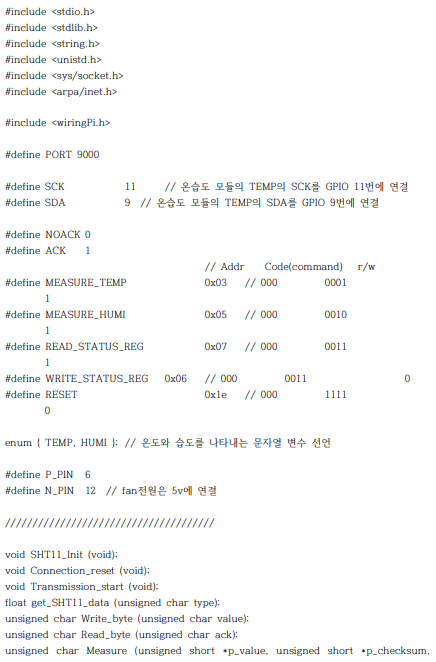
delay(2000);

FanOff();

delay(2000);

}

온/습도, 펜 서버



온습도 모듈의 TEMP의 SCK를 GPIO 11번에 연결

온습도 모듈의 TEMP의 SDA를 GPIO 9번에 연결

enum{TEMP, HUMI} 온도와 습도를 나타내는 문자열 변수

SHT11 센서의 TWI(I2C) 사용을 위한 초기화 작업

void SHT11\_Init (void)

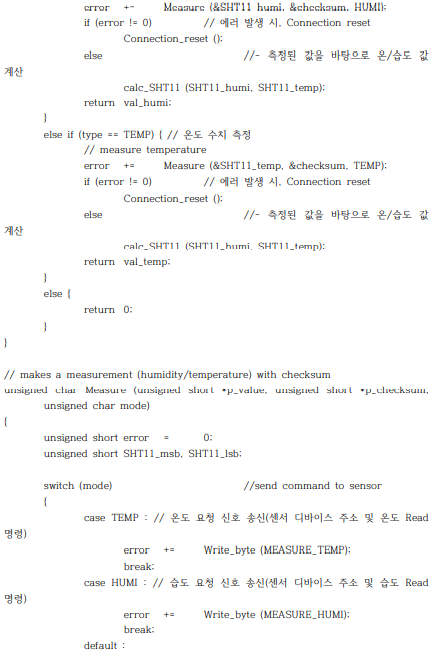
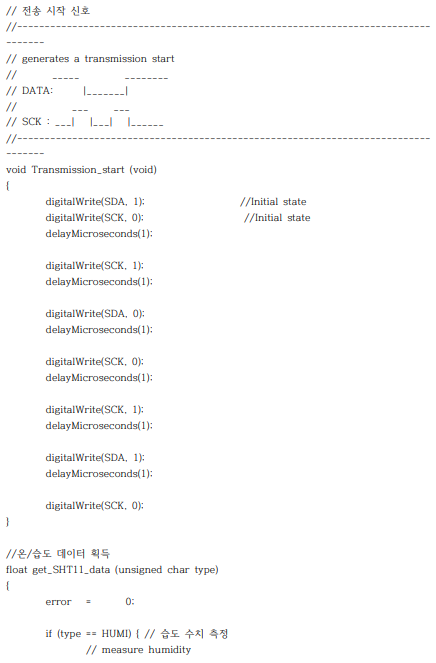
{

pinMode(SDA, OUTPUT);

pinMode(SCK, OUTPUT);

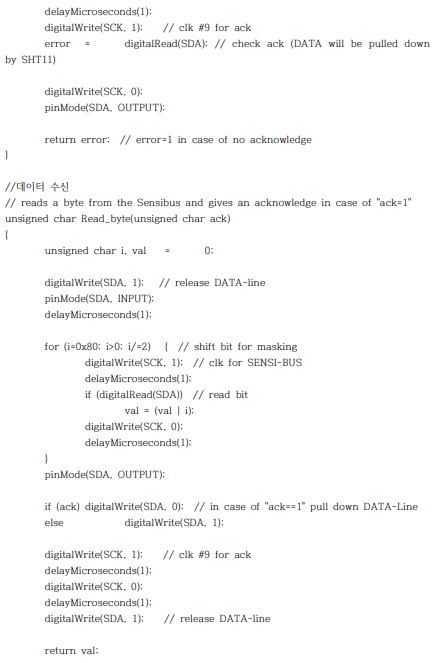
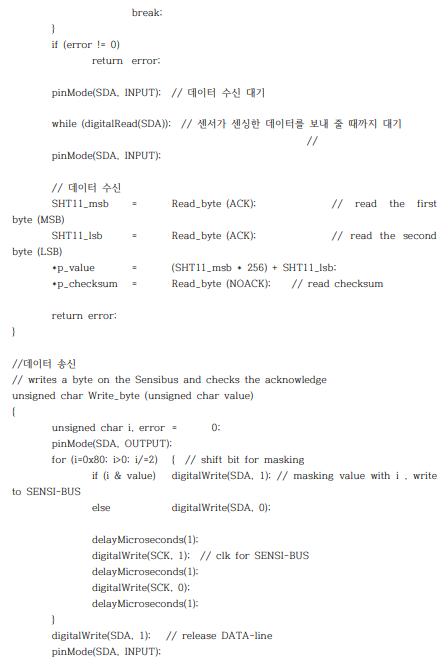
Connection\_reset();

}



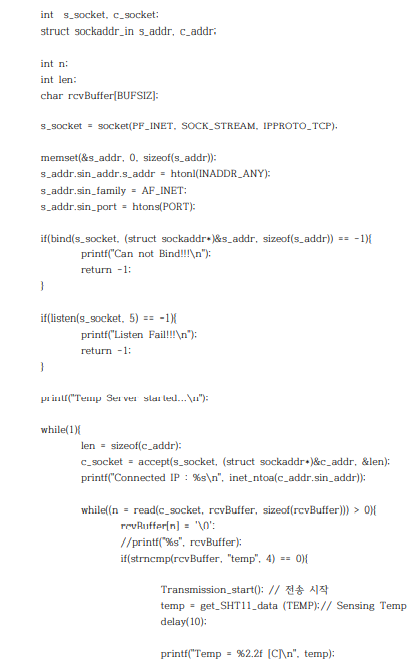
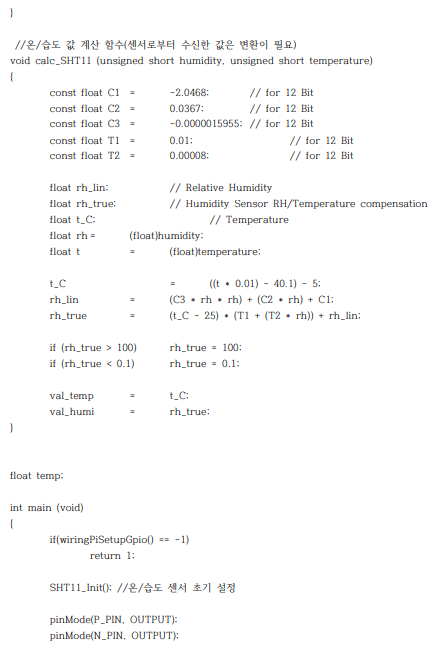
HUMI 습도

TEMP 온도



pinMode(SDA, INPUT); 데이터 수신 대기

while(digitalRead(SAD)): 센서가 센싱한 데이터를 보내 줄 때까지 대기



메인 스레드

SHT11\_Init(); 온/습도 센서 초기 설정

pinMode(P\_PIN, OUTPUT);

pinMode(N\_PIN, OUTPUT);

서버 소켓 생성 및 연결

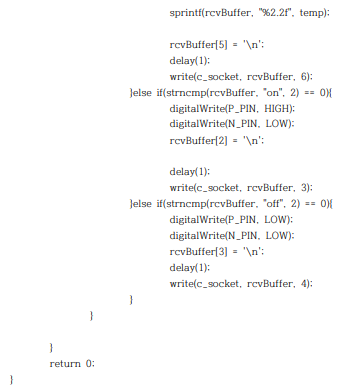
rcvBuffer로 temp 수신했으면

Transmission\_start(); 전송 시작

temp = get\_SHT11\_data(TEMP); 센싱한 온도

온도를 프린트

c\_socket에 rcvBuffer의 내용 반납



on 수신했으면 펜을 켬

digitalWrite(P\_PIN, HIGH);

digitalWrite(N\_PIN, LOW);

off 수신했으면 펜을 끔

digitalWrite(P\_PIN, LOW);

digitalWrite(N\_PIN, LOW);

온/습도 클라이언트



