**1주 – 2018년 3/10**

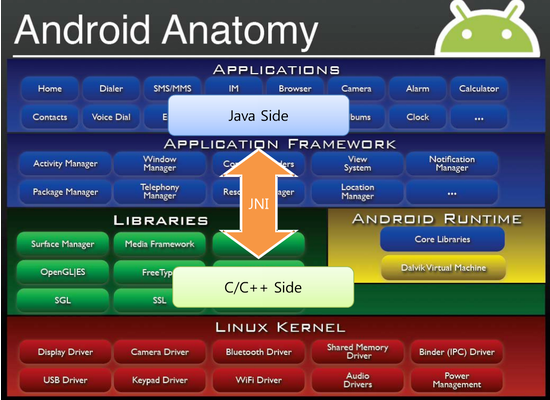
**안드로이드 gui 개발을 위한 NDK / JNI 설정 및 실습. 2012742027 홍준혁**

Objective

기존 라즈베리파이로 개발하고 있는 프로그램 같은 경우에는 운영체제가 리눅스로 돌아가기 때문에 보편적으로 쓰이고 있지 않다. 그리하여 최근 널리 사용되고 있는 안드로이드 기반의 프로그램을 개발하고자 한다. 다만 안드로이드 기반의 프로그램을 개발하기 위해선 라즈베리파이의 사양이 부족하기 때문에 좀더 높은 성능의 cpu를 가지고있는 odroid를 이용해 개발 할 것이다. 다만 안드로이드 같은 경우엔 Java언어가 주를 이루지만 시스템적인 부분에선 c로 작동하므로 Java와 C가 서로 연동되어야 할 필요가 있다. 그리하여 이번주는 NDK(Native Develop Kit)을 이용하여 안드로이드를 다루어보겠다.

Principle

NDK에 대해 간략하게 설명한다.



Java만 사용하여 필요한 기능과 성능을 모두 이끌어내기 어렵다 그리하여 C와 C++의 기능을 java에서도 사용하게 해주는 것이 필요한데 이을 jni 라고한다.

장점 :

1. 기존 C로 이루어진 대규모 코드를 JAVA에서 다시 재사용 가능하다.
2. 시스템 디바이스 접근과 JAVA성능을 넘어선 작업이 필요할 때 유용하다.
3. 속도 및 성능을 향상시킬 수 있다.

NDK는 주로 영상처리, 게임, 신호처리, 물리 시뮬레이션 등에 사용된다.

Experiment

실행환경은 리눅스이나 간단한 예제로 NDK가 잘 작동하는지 확인해보기위해 WINDOW 상의 안드로이드 스튜디오로 확인해보고 리눅스로 넘어가도록 하겠다.

대략적인 흐름은

1. MainActivity에 해당하는 java코드 작성
2. Opencv를 이용하면 코드가 많아질 것으로 예상되므로 유지보수를 위해 별도의 native 함수가 모여있는 jniClass 생성
3. Javah 명령어를 통해 native 함수가 필요로하는 c의 헤더파일을 생성.
4. C의 헤더파일을 토대로 c로 구성된 코드 작성
5. C의 코드와 java를 연결해줄 make 파일 생성

과정 1.

**public class** MainActivity **extends** AppCompatActivity {  
  
 **static**{  
 System.*loadLibrary*(**"ndktest"**);  
 }  
  
 jniClass **jni** = **new** jniClass();  
   
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 Button bt1=(Button)findViewById(R.id.***button***);  
 Button bt2=(Button)findViewById(R.id.***button2***);  
 **final** TextView tv=(TextView)findViewById(R.id.***textView***);  
  
 bt1.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 tv.setText(**jni**.jniM());  
 }  
 });  
  
  
 bt2.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 tv.setText(**jni**.getNumString(**"hi"**,1));  
 }  
 });  
  
 }  
  
  
 }

간단하게 버튼 두개로 textview의 내용을 변경해 보도록 한다.

버튼 1 >> jniM 함수를 통해 변경

버튼 2 >> getNumString 함수를 통해 변경

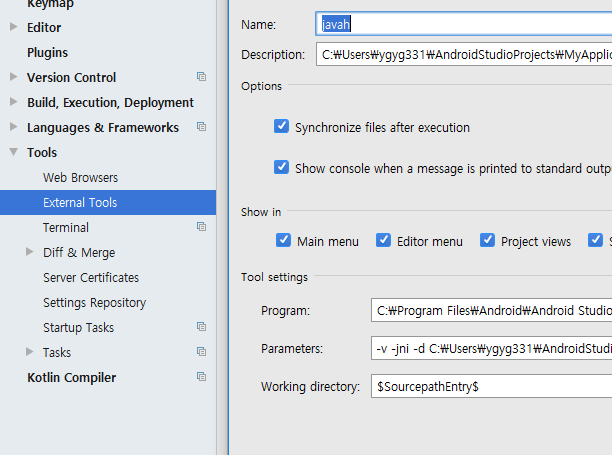
과정 2.

**public class** jniClass {  
 **static** {  
 System.*loadLibrary*(**"ndktest"**);  
 }  
 **public native** String jniM();  
 **public native** String getNumString(String str, **int** num);  
 **public native void** hello();  
  
 **private** String callback(String str, **int** num) {  
  
 **return** str + **" : "** + num;  
 }  
}

main함수에서 사용될 native함수를 모아놓은 class 이다. Callback 함수는 getnumstring에서 전달 받은 string과 num을 메인함수에 전달한다.

과정 3.

File > setting > tools > external tools 에 들어가 javah 파일이 있는 곳과 헤더파일이 만들어질 곳을 설정 할 수 있다.



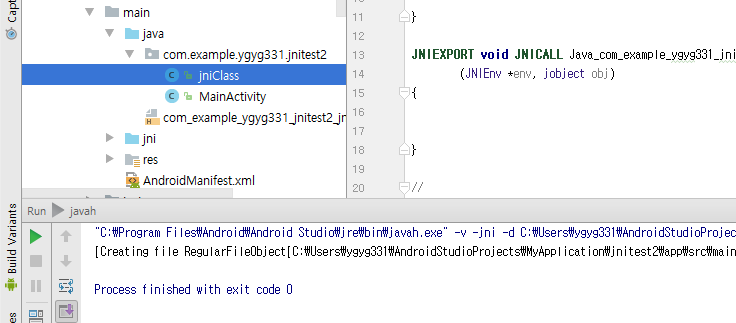
Program : javah실행파일이 있는 위치

Parameters :

-v –jni –d C:$module 위치$ #class 이름

Working directory : 헤더파일 만들어질 곳 위치

다만 javah 설정에 많은 에러사항이 있으니 디렉토리 위치에 신경써서 정확하게 입력해 주어야 에러가 나지않고 정상적으로 파일이 만들어진다.



성공했을 시 나오는 메시지

Javah로 jniClass를 실행시켜보면 자동적으로 jniclass 에맞는 헤더파일을 생성시킨다.

과정 4#

#include **"../java/jniClass.h"**#include **<jni.h>**#include **<stdio.h>**#include **<string.h>  
  
JNIEXPORT** jstring **JNICALL** Java\_com\_example\_ygyg331\_jnitest2\_jniClass\_jniM  
 (JNIEnv \*env, jobject obj) {  
  
 **return** (\*env)->NewStringUTF(env, **"성공입니다.!!"**);  
  
}  
  
  
  
**JNIEXPORT** jstring **JNICALL** Java\_com\_example\_ygyg331\_jnitest2\_jniClass\_getNumString(JNIEnv \*a, jobject b, jstring str, jint num)  
  
{  
 jclass cls = (**struct**)a->getObjectClass(b);  
 jmethodID func=GetMethodID(cls, **"callback"**, **"(Ljava/lang/String;I)Ljava/lang/String;"**);  
 **return** (\*a)->CallObjectMethod(b, func, str, num);  
}

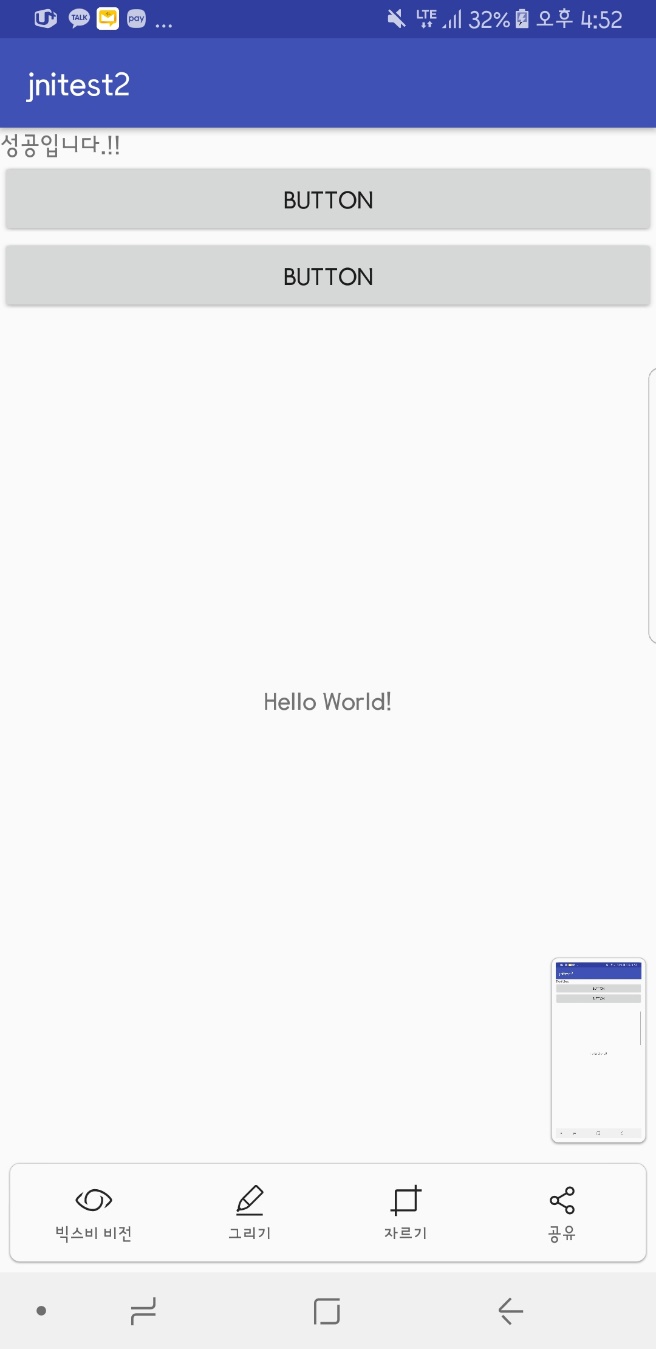
c언어로 구성된 파일 작성

과정 5

Android.mk 파일 작성

LOCAL\_PATH := $(call my-dir)  
  
include $(CLEAR\_VARS)  
  
LOCAL\_MODULE := ndktest  
LOCAL\_SRC\_FILES := ndktest.c  
  
include $(BUILD\_SHARED\_LIBRARY)

Result



어플은 잘 작동되는 것을 확인 할 수 있었고 별다른 이상은 없었다.

다음주엔 odroid를 이용해 안드로이드 환경세팅을 하고 작동유무를 확인해보겠다. 또한 c언어로 구성된 프로그램을 java로 변형시킬 것이기에 따로 언어를 짤 필요는 없으나 상황에맞게 수정해야 할 필요가 있다. c언어를 구성하기엔 실력이 부족하므로 좀 더 코드를 분석할 수 있는 능력을 기를 필요가 있겠다.