Embedded System Software

NDK

Dept. of Computer Science and Engineering
Sogang University, Seoul, KOREA

Android NDK 설치

- NDK
  - The NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++. For certain types of apps, this can be helpful so you can reuse existing code libraries written in these languages, but most apps do not need the Android NDK.

  - wget https://dl.google.com/android/repository/android-ndk-r10e-linux-x86_64.zip
  - android-ndk-r10e-linux-x86_64.zip

Android NDK 설치

- **On Linux**
  - unzip android-ndk-r10e-linux-x86_64.zip
  - mv android-ndk-r10e /work/mydroid
  - vi /root/.bashrc 수정

```
101 #Cross Compiler
102 export CROSS_COMPILE=arm-none-linux-gnueabi-
103 export PATH=/opt/toolchains/arm-2014.05/bin:$PATH
104 export ARCH=arm
105
106 #JAVA JDK
107 export PATH=$PATH:/usr/lib/jvm/java-6-oracle/bin
108 export JAVA_HOME=/usr/lib/jvm/java-6-oracle/jre/bin/java
109 export ANDROID_JAVA_HOME=/usr/lib/jvm/java-6-oracle
110 #sdk add
111 export PATH=/work/mydroid/adt-bundle-linux-x86_64-20140702/sdk/platform-tools:$PATH
112 #ndk path add
113 export PATH=/work/mydroid/android-ndk-r10e:$PATH
114
~
```

  - **source /root/.bashrc**

Primitive Types

- Java primitive types and their machine-dependent native equivalents

Java type	Java native type	Byte size
byte	jbyte	1
short	jshort	2
int	jint	4
long	jlong	8
float	jfloat	4
double	jdouble	8
char	jchar	2
boolean	jboolean	1
void	void	

Java reference type	Java native type
class	jclass
object	jobject
String	jstring

Field Descriptors

- Field descriptors of reference types begin with the "L" character, followed by the class descriptor, and terminated by the ";" character.
- Field descriptors of array types are formed following the same rule as class descriptors of array classes

Field Descriptors

- ex)
  - (*env)->GetFieldID(env, cls, "s", "Ljava/lang/String;");
  - (*env)->GetStaticFieldID(env, cls, "s", "I");

Type Signatures	Java Language Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double

Type Signatures	Java Language Type
"Ljava/lang/String;"	String
"[I"	int[]
"[Ljava/lang/Object;"	Object[]

Method Descriptors

- Method descriptors are formed by placing the field descriptors of all argument types in a pair of parentheses, and following that by the field descriptor of the return type.
- There are no spaces or other separator characters between the argument types.
- "V" is used to denote the void method return type.
- Constructors use "V" as their return type, and use "<init>" as their name.

Method Descriptors

- ex)
  - (*env)->GetMethodID(env, cls, “callback", "Ljava/lang/String;");
  - (*env)->GetStaticMethodID(env, cls, " callback ", “()V");

Method Descriptor	Java Language Type
"()Ljava/lang/String;"	String f();
"(ILjava/lang/Class;)J"	long f(int i, Class c);
"([B)V"	String(byte[] bytes);

JNI Interface Pointer

- javah가 생성하는 함수 원형
  - JNIEnv* 와 jobject라는 디폴트 매개변수를 포함
  - JNI를 지원하는 함수는 이 두 개의 공통 매개변수를 반드시 포함해야 한다.
- JNIEnv*
  - JNI 인터페이스 포인터로서, 이 포인터를 통해 여러 인터페이스 함수를 호출할 수 있다. 그리고 인터페이스 함수는 JNI 함수를 호출한다.
  - jobject : JNI에서 제공되는 java native type이며 C 코드에서 JAVA 객체에 접근 시 사용된다.

JNI

- native 키워드
  - 자바 클래스에서는 C/C++로 작성된 JNI 네이티브 함수와 연결할 메서드를 native 키워드를 이용해서 선언
- System.loadLibrary()
  - 네이티브 메서드가 실제로 구현돼 있는 C라이브러리를 로딩
  - 일반적으로 자바에서 static block에서 로드함
  - System.loadLibrary("jni")를 호출했을 경우, 실제 로드되는 C 라이브러리
    - 윈도우 : jni.dll
    - 리눅스 : libjni.so

JNI

- 함수의 시그너처
  - JNIEXPORT void JNICALL Java_HelloJNI_printHello(...)

반환타입 접두사 클래스이름 네이티브 메서드 이름

- JNIEXPORT, JNICALL
  - 플랫폼 종속적인 매크로

JNI 예제

- HelloJni.java

```
class HelloJni {
        native void printHello();
        native void printString(String str);

        static {
                System.loadLibrary("hellojni");
        };

        public static void main(String args[]) {
                HelloJni jni = new HelloJni();
                jni.printHello();
                jni.printString("hello from java");
        }
}
```

- java 컴파일
  - source /root/.bashrc
  - javac HelloJni.java

혹시 compile 안되면

- /root/.bashrc 다시 확인!

```
#fi

#cross compiler
export CROSS_COMPILE=arm-none-linux-gnueabi-
export PATH=/opt/toolchains/arm-2014.05/bin:$PATH:/usr/lib/jvm/java-6-oracle/bin

export ARCH=arm

#JAVA JDK
export JAVA_HOME=/usr/lib/jvm/java-6-oracle/jre/bin/java
export ANDROID_JAVA_HOME=/usr/lib/jvm/java-6-oracle
```

JNI 예제

- JNI 헤더파일 생성
  - javah HelloJni
- 생성된 JNI 헤더파일 HelloJni.h

```
...
JNIEXPORT void JNICALL Java_HelloJni_printHello
  (JNIEnv *, jobject);
...
JNIEXPORT void JNICALL Java_HelloJni_printString
  (JNIEnv *, jobject, jstring);
...
```

JNI 예제

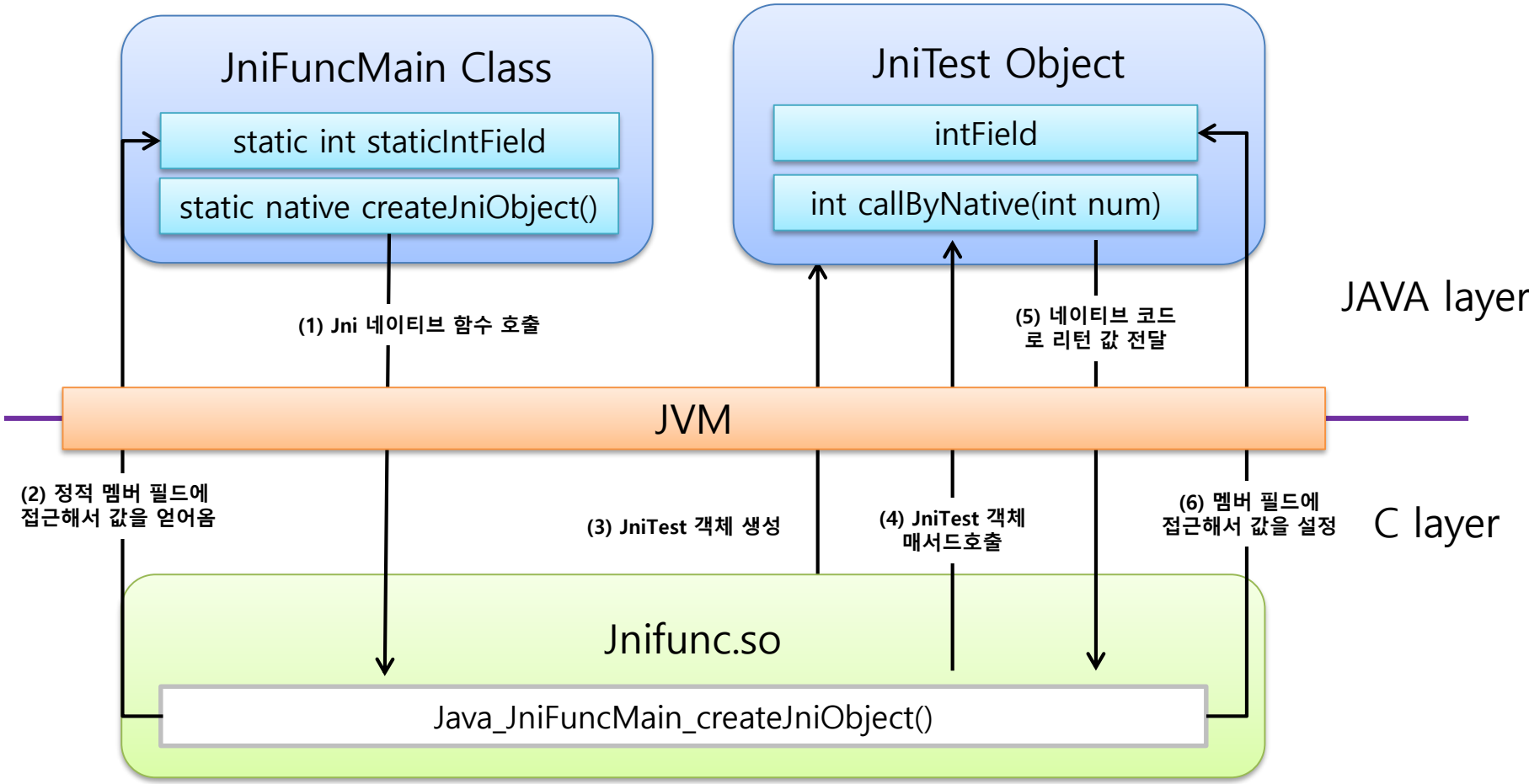
- hellojni.c

```
#include "HelloJni.h"

JNIEXPORT void JNICALL Java_HelloJni_printHello(JNIEnv *env, jobject obj) {
        printf("hello₩n");
}

JNIEXPORT void JNICALL Java_HelloJni_printString(JNIEnv *env, jobject obj, jstring
string) {
        const char *str = (*env)->GetStringUTFChars(env, string, 0);
        printf("%s₩n", str);
}
```

JNI 예제

- .so 라이브러리 생성 (sh 파일 제공)
  - gcc –fPIC –I /usr/lib/jvm/java-6-oracle/include –I /usr/lib/jvm/java-6-oracle/include/linux –shared –m64 –o libhellojni.so hellojni.c
- lib패스 추가
  - vi /etc/profile
  - 끝에
  - export LD_LIBRARY_PATH=/usr/lib:/usr/local/lib 추가
  - source /etc/profile
- .so library 이동
  - cp libhellojni.so /usr/lib/libhellojni.so
  - java HelloJni
- 실행
  - java HelloJni

JNI 함수를 활용하는 예제



JNI 함수를 활용하는 예제

- JniFuncMain.java

```
public class JniFuncMain
{
        private static int staticIntField = 300;
        static {
                System.loadLibrary("jnifunc");
        }
        public static native JniTest createJniObject();

        public static void main(String[] args)
        {
                JniTest t = createJniObject();
                t.callTest();
                t.callTest2();
        }
}
```

JNI 함수를 활용하는 예제

- JniTest.java

```
public class JniTest
{
        native public void callNative2(int num);
        private int intField;

        public JniTest(int num) {
                intField = num;
                System.out.println("java JniTest "+num);
        }
        public int callByNative(int num) {
                System.out.println("java callByNative "+num);
                return num;
        }
```

JNI 함수를 활용하는 예제

- JniTest.java(Cont.)

```
        public void callTest() {
                System.out.println("java callTest "+intField);
        }

        public void callTest2() {
                System.out.println("java callTest2 ");
                callNative2(10);
        }
        public void callByNative2(int num) {
                System.out.println("java callByNative2 "+num);
        }

}
```

JNI 함수를 활용하는 예제

- jnifunc.c

두번째 매개변수가 jobject가 아닌 jclass이유는
static 메서드이기 때문(public static native JniTest createJniObject())
즉, 객체가 아닌 클래스를 통해 호출되기 때문

```
JNIEXPORT jobject JNICALL Java_JniFuncMain_createJniObject(JNIEnv* env, jclass clazz)
{
        jfieldID fid = (*env)->GetStaticFieldID( env, clazz, "staticIntField", "I" );
        jint staticIntField = (*env)->GetStaticIntField(env, clazz,  fid);

        printf("native staticIntfiled %d₩n", staticIntField );

        jclass targetClass = (*env)->FindClass( env, "JniTest");
        jmethodID mid = (*env)->GetMethodID(env, targetClass, "<init>", "(I)V");
```

c코드일 경우는 (*env)->GetStaticFieldID(...)
c++코드일 경우는 env->GetStaticFieldID(...)
형식으로 호출한다

```
java JniTest 100
java callByNative 200
callTest 200
```

JNI 함수를 활용하는 예제(cont.)

- jnifunc.c

```
        jobject newObject = (*env)->NewObject( env, targetClass, mid, 100 );

        mid = (*env)->GetMethodID(env, targetClass, "callByNative", "(I)I" );
        int result = (*env)->CallIntMethod( env, newObject, mid, 200 );
        printf("native result %d₩n", result);

        fid = (*env)->GetFieldID( env, targetClass, "intField", "I");
        (*env)->SetIntField( env, newObject, fid, result );

        return newObject;
}
```

실행

- 컴파일
  - **javac JniFuncMain.java**
- 헤더파일 생성
  - **javah JniFuncMain**
  - **javah JniTest**
- 헤더파일 참고하여 **jnifunc.c** 작성
- 이후
  - gcc -fPIC -I /usr/lib/jvm/java-6-oracle/include -I /usr/lib/jvm/java-6-oracle/include/linux -shared -m64 -o libjnifunc.so jnifunc.c (sh 파일 제공)
  - cp libjnifunc.so /usr/lib/libjnifunc.so
  - java JniFuncMain

JNI Methods

GetObjectClass

jclass GetObjectClass(JNIEnv *env, jobject obj);

Returns the class of an object.

env: the JNI interface pointer.
obj: a Java object (must not be NULL)..

NewObject

jobject NewObject(JNIEnv *env, jclass clazz, jmethodID methodID, ...);

Constructs a new Java object

env: the JNI interface pointer.
clazz: a Java class object.
methodID: the method ID of the constructor.

JNI Methods

GetStaticFieldID

jfieldID GetStaticFieldID(JNIEnv *env, jclass clazz, const char *name, const char *sig)

Returns the field ID for a static field of a class

env: the JNI interface pointer.
clazz: a Java class object.
name: the static field name in a 0-terminated UTF-8 string.
sig: the field signature in a 0-terminated UTF-8 string.

GetFieldID

jfieldID GetFieldID(JNIEnv *env, jclass clazz, const char *name, const char *sig)

Returns the field ID for an instance (nonstatic) field of a clas

env: the JNI interface pointer.
clazz: a Java class object.
name: the field name in a 0-terminated UTF-8 string.
sig: the field signature in a 0-terminated UTF-8 string.

JNI Methods

SetStatic<type>Field
void SetStatic<type>Field(JNIEnv *env, jclass clazz,
jfieldID fieldID, NativeType value);

This family of accessor routines sets the value of a static field of an object

SetStaticIntField(), SetStaticLongField() …

GetStatic<type>Field

NativeType GetStatic<type>Field(JNIEnv *env, jclass clazz,
jfieldID fieldID);

This family of accessor routines returns the value of a static field of an object

GetStaticIntField(), GetStaticLongField() …

JNI Methods

Set<type>Field
void SetStatic<type>Field(JNIEnv *env, jobject obj,
jfieldID fieldID, NativeType value);

This family of accessor routines sets the value of an instance (nonstatic) field of an object.

SetIntField(), SetLongField() …

Get<type>Field

NativeType Get<type>Field(JNIEnv *env, jobject obj,
jfieldID fieldID);

This family of accessor routines returns the value of an instance (nonstatic) field of an object

GetIntField(), GetLongField() …

JNI Methods

GetStaticMethodID

jmethodID GetStaticMethodID(JNIEnv *env, jclass clazz, const char *name, const char *sig);

Returns the method ID for a static method of a class. The method is specified by its name and signature.

env: the JNI interface pointer.
clazz: a Java class object.
name: the static method name in a 0-terminated modified UTF-8 string.
sig: the method signature in a 0-terminated modified UTF-8 string.

JNI Methods

GetMethodID

jmethodID GetMethodID(JNIEnv *env, jclass clazz, const char *name, const char *sig))

Returns the method ID for an instance (nonstatic) method of a class or interface

env: the JNI interface pointer.
clazz: a Java class object.
name: the static field name in a 0-terminated UTF-8 string.
sig: the field signature in a 0-terminated UTF-8 string.

JNI Methods

CallStatic<type>Method

NativeType CallStatic<type>Method(JNIEnv *env, jclass clazz,
jmethodID methodID, ...);

This family of operations invokes a static method on a Java object

env: the JNI interface pointer.
clazz: a Java class object.
methodID: a static method ID

CallStaticVoidMethod(), CallStaticObjectMethod(), CallStaticIntMethod()…

JNI Methods

Call<type>Method

NativeType Call<type>Method(JNIEnv *env, jobject obj,
jmethodID methodID, ...);

Methods from these three families of operations are used to call a Java instance method from a native method

env: the JNI interface pointer.
clazz: a Java class object.
methodID: a static method ID

CallVoidMethod(), CallObjectMethod(), CallIntMethod()…

JNI Methods

NewStringUTF

jstring NewStringUTF(JNIEnv *env, const char *bytes);

Constructs a new java.lang.String object from an array of characters in modified UTF-8 encoding.

env: the JNI interface pointer.
bytes: the pointer to a modified UTF-8 string.

FindClass

jclass FindClass(JNIEnv *env, const char *name)

This function loads a locally-defined class

env: the JNI interface pointer.
name: a fully-qualified class name

JNI Methods

GetStringUTFLength

jsize GetStringUTFLength(JNIEnv *env, jstring string);

Returns the length in bytes of the modified UTF-8 representation of a string.

env: the JNI interface pointer.
string: a Java string object

GetStringUTFChars

const char * GetStringUTFChars(JNIEnv *env, jstring string,
jboolean *isCopy);

Returns a pointer to an array of bytes representing the string in modified UTF-8 encoding. This array is valid until it is released by ReleaseStringUTFChars()

env: the JNI interface pointer.
string: a Java string object.
isCopy: a pointer to a boolean.

JNI Methods

ReleaseStringUTFChars

void ReleaseStringUTFChars(JNIEnv *env, jstring string,
const char *utf);

Informs the VM that the native code no longer needs access to utf

env: the JNI interface pointer.
string: a Java string object.
utf: a pointer to a modified UTF-8 string.

Android.mk

- 안드로이드 빌드 시스템이 빌드를 하는데 필요한 정보를 알려줌
- NDK 경우는 기본적으로 ./jni/에 Android.mk파일들이 있다고 생각함
  - LOCAL_PATH
    - Android.mk 파일 전반에 사용되는 파일의 기본경로
    - Android.mk 파일 처음에 지정해줘야하며 특별한 경우가 아닌 경우 LOCAL_PATH:=$(call my-dir)로 작성함.
    - $(call my-dir)은 매크로 함수로써 현재 폴더 위치를 나타냄
  - include $(CLEAR_VARS)
    - Android.mk에서 사용되는 LOCAL_XXX 변수(LOCAL_MODULE, LOCAL_SRC_FILES)의 값을 초기화. 단 LOCAL_PATH 제외
    - 안드로이드 빌드 시스템이 LOCAL_XXX변수들을 전역적으로 사용하기 때문에 이변수들의 기존 설정값들을 초기화하기 위해 사용

Android.mk

- LOCAL_MODULE
  - 생성할 라이브러리 이름
- LOCAL_SRC_FILES
  - 라이브러리 생성에 필요한 소스코드 목록
- include $(BUILD_SHARED_LIBRARY)
  - LOCAL_MODULE, LOCAL_SRC_FILES 등의 변수값을 이용해서 lib$(LOCAL_MODULE).so라는 공유 라이브러리 생성

Android Application에서 lib위치

- Android Application을 빌드를 하게되면 어플리케이션 패키지 파일인 apk가 생성됨
- Eclipse에서 Android Application 만들 경우
  - .so파일은 libs/armeabi/ 폴더에 있어야함
    - 그렇지 않으면 apk안에 .so 라이브러리가 포함되지 않아서 에러가 발생한다.

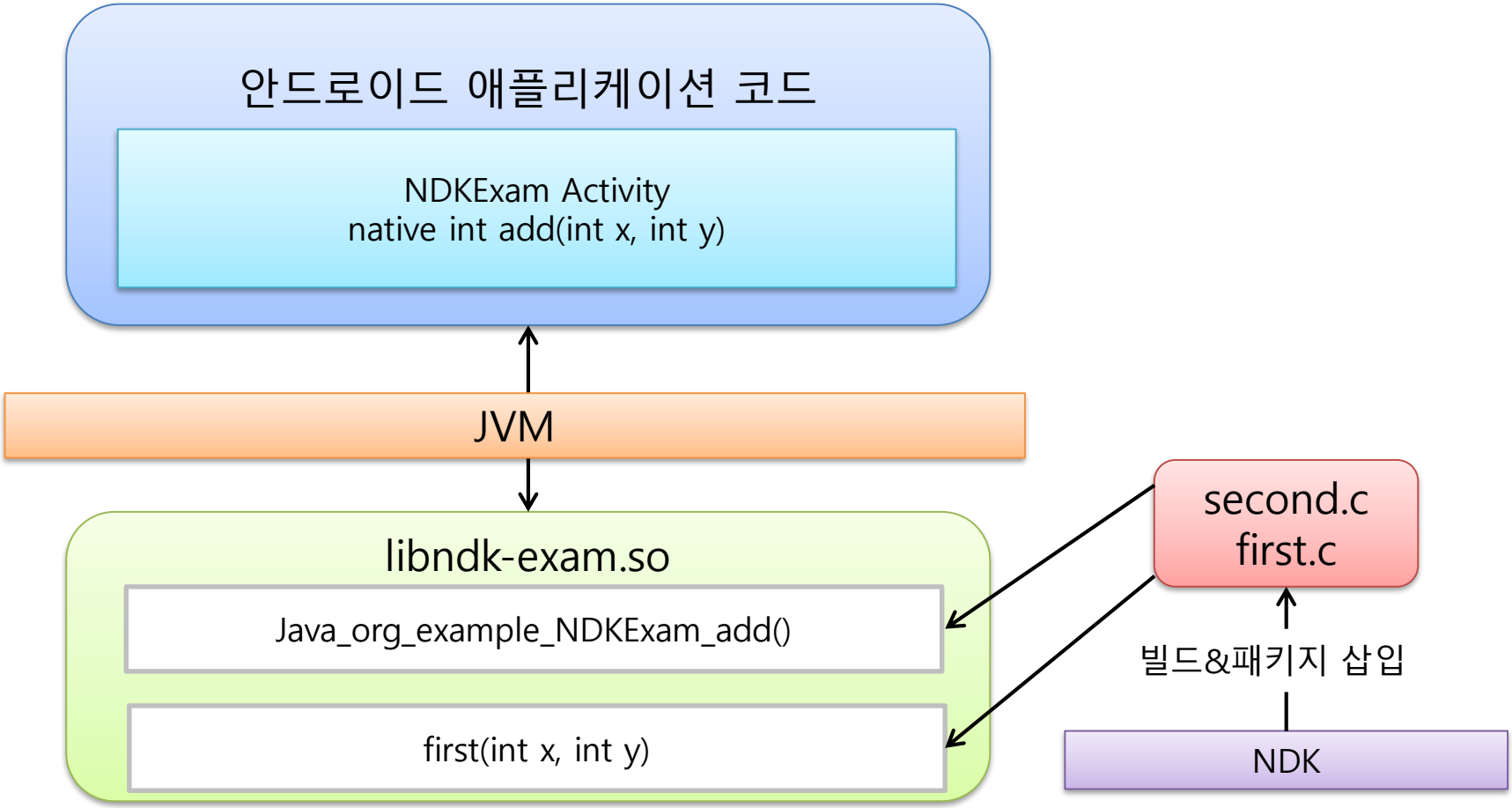
NDK 사용시 로그 출력

- **Android.mk 수정**
  - **LOCAL_LDLIBS := -llog**
  - **또는 LOCAL_LDFLAGS += $(TARGET_OUT_INTERMEDIATE_LIBRARIES)/liblog.so**

```
#include "android/log.h"

#define LOG_TAG "MyLogTag"

#define LOGV(...)   __android_log_print(ANDROID_LOG_VERBOSE, LOG_TAG, __VA_ARGS__)
#define LOGD(...)   __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, __VA_ARGS__)
#define LOGI(...)   __android_log_print(ANDROID_LOG_INFO, LOG_TAG, __VA_ARGS__)
#define LOGW(...)   __android_log_print(ANDROID_LOG_WARN, LOG_TAG, __VA_ARGS__)
#define LOGE(...)   __android_log_print(ANDROID_LOG_ERROR, LOG_TAG, __VA_ARGS__)
```

- **예 : LOGV("log test %d", 1234);**

NDK 예제


안드로이드 애플리케이션 코드
NDKExam Activity
native int add(int x, int y)
JVM
libndk-exam.so
Java_org_example_NDKExam_add()
first(int x, int y)
second.c
first.c
빌드&패키지 삽입
NDK

NDK 예제

- New / Android Application Project
  - Application Name : NDKExam
  - Project Name : NDKExam
  - Package Name : org.example.ndk
  - Build SDK : Android 4.2.2

NDK 예제

- NDKExam.java

```
package org.example.ndk;

public class NDKExam extends Activity {
   public native int add(int x, int y);
   public native void testString(String str);
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        System.loadLibrary("ndk-exam");

        TextView tv = new TextView(this);
        int x = 1000;              int y = 42;
        int z = add(x, y);

        tv.setText("The sum of " + x + " and " + y + " is " + z);
        setContentView(tv);
        testString("test");
    }
}
```

NDK 예제

- JNI 네이티브 함수 원형 생성하기
  - (eclipse에서 컴파일을 한 후에)
  - **eclipse가 아니라 터미널에서 해당 프로젝트 폴더 로 이동**
  - **ex) cd ~/Downloads/9week/NDKExam**
  - javah –classpath bin/classes org.example.ndk.NDKExam
  - 헤더파일 생성됨

NDK 빌드

jni/second.c

```
#include <jni.h>
#include "android/log.h"

#define LOG_TAG "MyTag"
#define LOGV(...)   __android_log_print(ANDROID_LOG_VERBOSE, LOG_TAG, __VA_ARGS__)

extern int first(int x,int y);

jint JNICALL Java_org_example_ndk_NDKExam_add(JNIEnv *env, jobject this, jint x, jint y) {
        LOGV("log test %d", 1234);
        return first(x, y);
}

void JNICALL Java_org_example_ndk_NDKExam_testString(JNIEnv *env, jobject this, jstring string) {
        const char *str=(*env)->GetStringUTFChars( env, string, 0);
        jint len = (*env)->GetStringUTFLength( env, string );
        LOGV("native testString len %d", len);
        LOGV("native testString %s", str);

        (*env)->ReleaseStringUTFChars( env, string, str );
}
```

NDK 빌드

- jni/first.c

```
extern int first(int x,int y);

int first(int x, int y)
{
        return x + y;
}
```

NDK 빌드

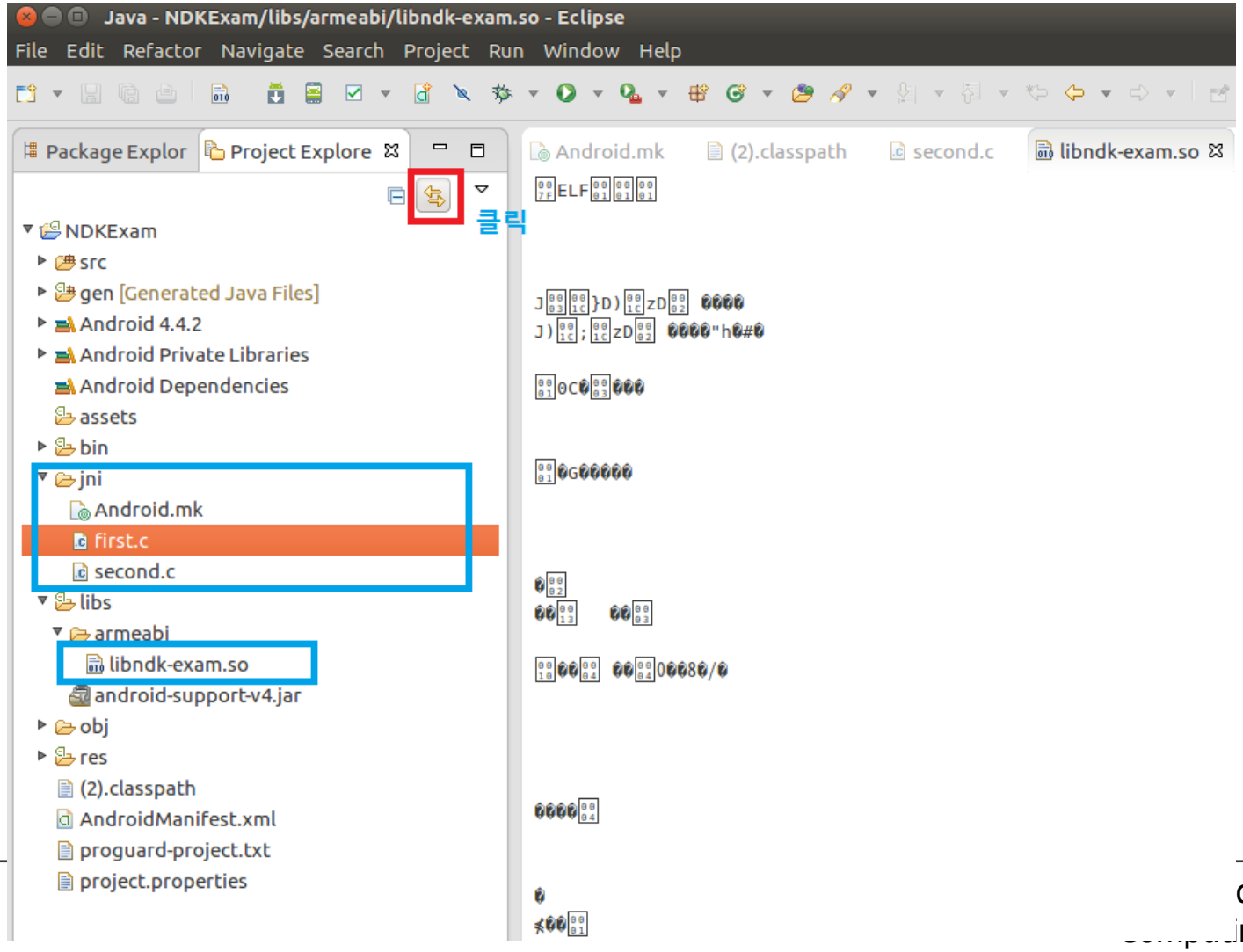
- jni/Android.mk

```
LOCAL_PATH:=$(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE:=libndk-exam
LOCAL_MODULE_TAGS := optional eng
LOCAL_PRELINK_MODULE := false
LOCAL_SRC_FILES:=first.c second.c
#LOCAL_LDLIBS := -llog
LOCAL_LDFLAGS += $(TARGET_OUT_INTERMEDIATE_LIBRARIES)/liblog.so

include $(BUILD_SHARED_LIBRARY)
```

NDK 빌드

- **eclipse가 아니라 터미널에서 해당 프로젝트 폴더/jni 로 이동**
  - **ex) cd ~/Downloads/9week/NDKExam/jni**
  - **ndk-build**
  - **그럼 자동으로 NDKExam/libs/areabi/lib{이름}.so 가 생성됨!**
  - **코드를 바꿨을 경우도 코드 저장 후, 해당 작업을 통해 다시 library 빌드 해야함!**

eclipse에서 libs/armeabi 밑에 해당 파일이 안보일 경우



JNI 함수직접 등록하기

- 자바 가상머신은 위에서 로드된 라이브러리의 함수 심볼을 검색해서 자바에서 선언된 네이티브 메서드의 시그너처와 일치하는 JNI 함수를 찾은 다음 네이티브 메서드와 실제 구현인 JNI네이티브 함수를 매핑한다. 이 심볼을 일일이 검색해서 매핑하는 작업은 성능 저하의 원인이 된다.
- RegisterNatives()
  - JNI 네이티브 함수를 직접 자바 클래스의 네이티브 메서드에 매핑

JNI 함수직접 등록하기

- JNI_OnLoad()
  - 라이브러리를 로딩할 때 자동으로 호출됨
  - JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM *jvm, void *reserved);

```
JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM *vm, void *reserved) {
 JNIEnv *env = NULL; JNINativeMethod nm[2]; jclass cls;  jint result = -1;
 if( (*vm)->GetEnv(vm, (void**)&env, JNI_VERSION_1_4) != JNI_OK) {
  printf("Error");    return JNI_ERR;
 }

 cls = (*env)->FindClass(env,"HelloJni");

 nm[0].name = (char*)"printHello";
 nm[0].signature = (char*)"()V";
 nm[0].fnPtr = (void *)printHelloNative;

 nm[1].name = (char*)"printString";
 nm[1].signature = (char*)"(Ljava/lang/String;)V";
 nm[1].fnPtr = (void *)printStringNative;

 (*env)->RegisterNatives(env, cls, nm, 2);

 return JNI_VERSION_1_4;
}
```

과제 Tip (JNI 사용 예제)

```
1  #include <jni.h>
2  #include "android/log.h"
3  #include <unistd.h>
4  #include <fcntl.h>
5
6  #define LOG_TAG "MyTag"
7  #define LOGV(...)   __android_log_print(ANDROID_LOG_VERBOSE, LOG_TAG, __VA_ARGS__)
8
9  jint JNICALL Java_com_example_project3_MainActivity_driveropen(JNIEnv *env, jobject this){
10     int fd=open("/dev/project_driver",O_RDWR);
11     return fd;
12 }
13 jint JNICALL Java_com_example_project3_MainActivity_driverclose(JNIEnv *env, jobject this, jint fd){
14     return close(fd);
15 }
```

과제 Tip (JNI 사용 예제)

```
17 jint JNICALL Java_com_example_project3_MainActivity_driverwrite(JNIEnv *env, jobject this, jint fd, jint mode, jstring value){
18     char buf[200];
19     const char *str=(*env)->GetStringUTFChars(env,value,0);
20     int len;
21     int i,result;
22     for(i=0;;i++){
23         if(str[i]=='\0'){len=i; break;}
24     }
25     if(mode==20){
26         buf[0]=2;
27         buf[1]=0;
28         result=write(fd,buf,2);
29     }
30     else if(mode==21){
31         buf[0]=2; buf[1]=1;
32         for(i=2;i<len+2;i++){
33             buf[i]=str[i-2];
34         }
35         result=write(fd,buf,len+2);
36     }
37     else{
38         buf[0]=mode;
39         for(i=1;i<len+1;i++){
40             buf[i]=str[i-1];
41         }
42         result=write(fd,buf,len+1);
43     }
44     return result;
45 }
```

과제 Tip

- **Android -> Kernel Module로 정보 전달**
  - **정보 전달이 필요할 때 Android Application에서 Jni를 통해 write()와 같은 함수를 호출해 정보 전달 가능**

- **Kernel Module에서 Android로의 정보 전달은 어떻게?**
  - **방법1**
    - 1. Android thread를 하나 생성하여, write와 같은 함수를 호출해서 kernel module 내로 들어가게 한 뒤, thread를 sleep 시킨다!
    - 2. 이후, 정보 전달이 필요할 때, kernel 모듈이 해당 thread를 깨우고, 깨어난 thread가 kernel module이 생성한 데이터를 가져가는 방식
  - **방법2**
    - thread 하나가 짧은 시간을 주기로 module에서 데이터를 계속 읽어오는 방식

과제 tip

- button 색 변경
  - Button b=new Button(this);
  - b.setBackgroundColor(Color.BLACK);
- 현재 Activity(화면) 종료
  - finish()