OSP
logcat crunch

NDK Integration (JNI)

Lecture 6

Operating Systems Practical

9 November 2016

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

OSP
logcat crunch

JNI

Android JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

JNI

Android JNI Example

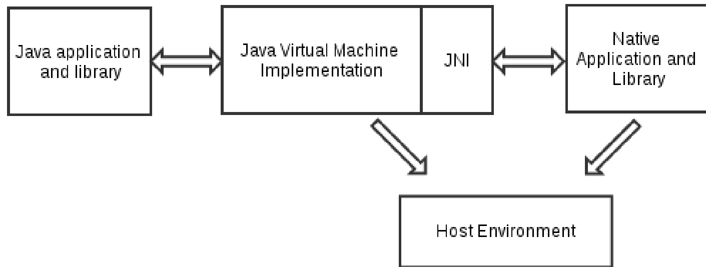
Native Method Arguments

Mapping of Types

Operations on Strings

- Java Native Interface (JNI)
- Native Programming Interface
- Allows Java code to interoperate with apps and libraries written in other programming languages
- When do we use JNI?
  - Java library does not support platform-dependent features
  - Use already existent library written in other language
  - Time-critical code in a low level language (performance)

- Two-way interface
- Allows Java apps to invoke native code and vice versa



Two-way Interface

- Support for two types of native code
- Native libraries
  - Call functions from native libraries
  - In the same way they call Java methods
- Native applications
  - Embed a Java VM implementation into native apps
  - Execute components written in Java
  - e.g. C web browser executes applets in an embedded Java VM implementation

Implications of Using JNI

- Java apps are portable
  - Run on multiple platforms
  - The native component will not run on multiple platforms
  - Recompile the native code for the new platform
- Java is type-safe and secure
  - C/C++ are not
  - Misbehaving native code can affect the whole application
  - Security checks when invoking native code
  - Extra care when writing apps that use JNI
- Native methods in few classes
  - Clean isolation between native code and Java app

- Android Native Development Kit
  - Allows the embed C/C++ (native) code in Android applications
  - Implement your own native code or use existing native libraries
- `ndk-build`
  - Shell script that invokes the NDK build scripts
  - Probe the development system and app project file to decide what to build
  - Generate binaries
  - Copy binaries in the app's project path

Outline

JNI

Android JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

- `Android.mk`
  - Configuration file
  - In `/jni`
  - The name of the native source file: `hello-jni.c`
  - The name of the shared library to build: `hello-jni`
  - Built library: `libhello-jni.so`

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

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)
```

Basic JNI Example - Application.mk

- `Application.mk`
  - In `/jni`
  - Specifies the CPU and architecture for building
  - In this example - all supported architectures

```
APP_ABI := all
```

Basic JNI Example - Java code

- Java source code in `/src`
  - Load native library in a static initializer
  - Declare the native function (`native` keyword)
  - Call the native function

```
package com.example.hellojni;

public class HelloJni extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView  tv = new TextView(this);
        tv.setText( stringFromJNI() );
        setContentView(tv);
    }
    public native String  stringFromJNI();
    static {
        System.loadLibrary("hello-jni");
    }
}
```

Basic JNI Example - Native code

- C source code in `/jni`
  - Implements the native function
  - Includes `jni.h`
  - Function name - based on the Java function name and the path to the file containing it
  - Arguments:
    - JNIEnv * - pointer to the VM, called interface pointer
    - jobject - implicit this object passed from the Java side
  - Creates a string by using a JNI function (`NewStringUTF`)

```
#include <string.h>
#include <jni.h>

jstring
Java_com_example_hellojni_HelloJni_stringFromJNI(JNIEnv* env,
                                                  jobject thiz)
{
    return (*env)->NewStringUTF(env, "Hello from JNI");
}
```

JNI

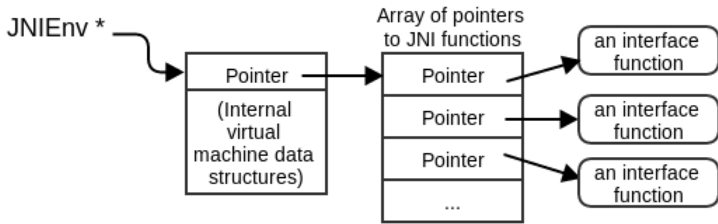
Android JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

- Passed into each native method call as the first argument
- Valid only in the current thread (cannot be used by other threads)
- Points to a location that contains a pointer to a function table
- Each entry in the table points to a JNI function
- Native methods access data structures in the Java VM through JNI functions



JNIEnv interface pointer

- For C and C++ source files the syntax for calling JNI functions differs
- C code
  - `JNIEnv` is a pointer to a `JNINativeInterface` structure
  - Pointer needs to be dereferenced first
  - JNI functions do not know the current JNI environment
    - JNIEnv instance should be passed as the first argument to the function call
  - e.g. `return (*env)->NewStringUTF(env, "Hello!");`
- C++ code
  - `JNIEnv` is a C++ class
  - JNI functions exposed as member functions
  - JNI functions have access to the current JNI environment
  - e.g. `return env->NewStringUTF("Hello!");`

Second Argument

- Second argument depends whether the method is static or instance
- Instance methods - can be called only on a class instance
- Static methods - can be called directly from a static context
- Both can be declared as native
- For instance native method
  - Reference to the object on which the method is invoked (*this* in C++)
  - e.g. `jobject thisObject`
- For static native method
  - Reference to the class in which the method is defined
  - e.g. `jclass thisClass`

Outline

JNI

Android JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Mapping of Types

- JNI defines a set of C/C++ types corresponding to Java types
- Java types
  - Primitive types: int, float, char
  - Reference types: classes, instances, arrays
- The two types are treated differently by JNI
- int -> jint (32 bit integer)
- float -> jfloat (32 bit floating point number)

Primitive Types

Java Type	JNI Type	C/C++ Type	Size
boolean	jboolean	unsigned char	unsigned 8 bits
byte	jbyte	char	signed 8 bits
char	jchar	unsigned short	unsigned 16 bits
short	jshort	short	signed 16 bits
int	jint	int	signed 32 bits
long	jlong	long long	signed 64 bits
float	jfloat	float	32 bits
double	jdouble	double	64 bits

- Objects -> opaque references
- C pointer to internal data structures in the Java VM
- Objects accessed using JNI functions (JNIEnv interface pointer)
  - e.g. `GetStringUTFChars()` function for accessing the contents of a string
- All JNI references have type jobject
  - All reference types are subtypes of `jobject`
  - Correspond to the most used types in Java
  - `jstring`, `jobjectArray`, etc.

Reference Types

Java Type	Native Type
java.lang.Class	jclass
java.lang. String	jstring
java.lang.Throwable	jthrowable
other objects	jobject
java.lang.Object[]	jobjectArray
boolean[]	jbooleanArray
byte[]	jbyteArray
char[]	jcharArray
short[]	jshortArray
int[]	jintArray
long[]	jlongArray
float[]	jfloatArray
double[]	jdoubleArray
other arrays	jarray

JNI

Android JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

String Types

- String is a reference type in JNI (`jstring`)
- Cannot be used directly as native C strings
  - Need to convert the Java string references into C strings and back
  - No function to modify the contents of a Java string (immutable objects)
- JNI supports UTF-8 and UTF-16/Unicode encoded strings
  - UTF-8 compatible with 7-bit ASCII
  - UTF-8 strings terminated with '\0' char
  - UTF-16/Unicode
  - Two sets of functions
  - `jstring` is represented in Unicode in the VM

Convert C String to Java String

- `NewStringUTF`, `NewString`

```
jstring javaString = env->NewStringUTF("Hello!");
```

- Takes a C string, returns a Java string reference type
- If the VM cannot allocate memory
  - Returns `NULL`
  - `OutOfMemoryError` exception thrown in the VM
  - Native code should not continue

Convert Java String to C String

- GetStringUTFChars, GetStringChars

```
const jbyte* str = env->GetStringUTFChars(javaString, &isCopy);
```

```
const jchar* str = env->GetStringChars(javaString, &isCopy);
```

- isCopy
  - JNI_TRUE - returned string is a copy of the chars in the original instance
  - JNI_FALSE - returned string is a direct pointer to the original instance (pinned object in heap)
  - Pass NULL if it's not important
- If the string contains only 7-bit ASCII chars you can use printf
- If the memory allocation fails it returns NULL, throws OutOfMemory exception

Release Strings

- Free memory occupied by the C string
- `ReleaseStringUTFChars`, `ReleaseStringChars`

```
env->ReleaseStringUTFChars(javaString, str);
```

- String should be released after it is used
- Avoid memory leaks
- Frees the copy or unpins the instance (copy or not)

Length and Copy in Buffer

- Get string length
  - `GetStringUTFLength/GetStringLength` on the `jstring`
  - Or `strlen` on the `GetStringUTFChars` result
- Copy string elements into a preallocated buffer

```
env->GetStringUTFRegion(javaString, 0, len, buffer);
```

  - start index and length
  - length can be obtained with `GetStringLength`
  - buffer is `char[]`
  - No memory allocation, no out-of-memory checks

- `GetStringCritical, ReleaseStringCritical`
- Increase the probability to obtain a direct pointer to the string
- Critical Section between the calls
  - Must not make blocking operations
  - Must not allocate new objects in the Java VM
  - Disable garbage collection when holding a direct pointer to a string
  - Blocking operations or allocating objects may lead to deadlock
- No `GetStringUTFCritical` -> usually makes a copy of the string

Bibliography

- `http://www.soi.city.ac.uk/~kloukin/IN2P3/material/jni.pdf`
- `http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html`
- `http://download.java.net/jdk8/docs/technotes/guides/jni/spec/functions.html`
- `http://developer.android.com/training/articles/perf-jni.html`
- Onur Cinar, Pro Android C++ with the NDK, Chapter 3
- Sylvain Ratabouil, Android NDK, Beginner's Guide, Chapter 3

Keywords

- Java Native Interface
- Two-way interface
- Native methods
- Interface pointer
- Static methods
- Instance methods
- Primitive types
- Reference types
- JNI functions
- Opaque reference
- Java string reference
- Conversion operations