

Native Activities

Lecture 7

Operating Systems Practical

23 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/>.

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

Managing Assets

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

Managing Assets

- ▶ Native activities
- ▶ From Android API level 9 (2.3 Gingerbread)
- ▶ Only native code, no Java
- ▶ API to access Android resources
 - ▶ Windows, assets, device configuration
- ▶ Missing functionality
- ▶ Used mainly for multimedia apps

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

Managing Assets

- ▶ Application named NativeApp
- ▶ Android Tools -> Add Native Support
- ▶ AndroidManifest.xml

- ▶ Min API level 9
- ▶ Specify activity name

```
<activity android:name="android.app.NativeActivity">
```

- ▶ Specify property android.app.lib_name
 - ▶ Native module name without lib prefix and .so suffix

```
<meta-data android:name="android.app.lib_name"  
android:value="NativeApp" />
```

- ▶ android:hasCode must be false
 - ▶ No Java code

```
<application android:hasCode="false">
```

```
#include <jni.h>
#include <android/native_activity.h>
#include <android/log.h>
static void onStart(ANativeActivity* activity){
    __android_log_print(ANDROID_LOG_INFO, "NativeApp",
"Start:_%p\n", activity);
}
[..]
void ANativeActivity_onCreate(ANativeActivity* activity ,
    void* savedInstanceState , size_t savedInstanceStateSize) {
    printf(activity);
    activity->callbacks->onStart = onStart;
    activity->callbacks->onResume = onResume;
    activity->callbacks->onPause = onPause;
    activity->callbacks->onStop = onStop;
    activity->callbacks->onDestroy = onDestroy;
    [..]
    activity->instance = NULL;
}
```

- ▶ `NativeApp.cpp` in `jni/`
- ▶ `ANativeActivity_onCreate()` is an entry point
- ▶ `ANativeActivity` structure defined in `native_activity.h`
- ▶ `savedState`: previously saved state of the activity
- ▶ `savedStateSize`: size in bytes of the saved state
- ▶ Implement callbacks to handle lifecycle events and user inputs

▶ Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := NativeApp
LOCAL_SRC_FILES := NativeApp.cpp
LOCAL_LDLIBS := -landroid -llog
include $(BUILD_SHARED_LIBRARY)
```

▶ Build app and run

```
04-26 15:51:41.535: I/NativeApp(6474): Internal data path:
/data/data/com.test.nativeapp/files
04-26 15:51:41.535: I/NativeApp(6474): External data path:
/storage/emulated/0/Android/data/com.test.nativeapp/files
04-26 15:51:41.535: I/NativeApp(6474): SDK version code: 19
04-26 15:51:41.535: I/NativeApp(6474): Start: 0x750539a8
04-26 15:51:41.535: I/NativeApp(6474): Resume: 0x750539a8
```

- ▶ Android framework provides `android.app.NativeActivity.java`
 - ▶ Helps the creation of a native activity
 - ▶ Subclass of `android.app.Activity`
 - ▶ Wrapper that hides the Java world from the native code
 - ▶ Exposes native interfaces defined in `native_activity.h`
 - ▶ Instance is created when you launch the native activity
 - ▶ Its `onCreate` calls `ANativeActivity_onCreate` through JNI
 - ▶ Invokes the callbacks when the corresponding events occur

```
typedef struct ANativeActivity {
    struct ANativeActivityCallbacks* callbacks;
    JavaVM* vm;
    JNIEnv* env;
    jobject clazz;
    const char* internalDataPath;
    const char* externalDataPath;
    int32_t sdkVersion;
    void* instance;
    AAssetManager* assetManager;
} ANativeActivity;
```

- ▶ callbacks: pointer to the callback function table
 - ▶ Set the functions to your own callbacks
 - ▶ Called by the Android framework
- ▶ vm: global Java VM handle
- ▶ env: JNIEnv interface pointer
- ▶ clazz: reference to `android.app.NativeActivity` object
- ▶ `internalDataPath`, `externalDataPath`, `sdkVersion`
- ▶ instance: native instance of the application
- ▶ `assetManager`: accessing binary assets in the apk

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

Managing Assets

- ▶ `native_activity.h` provides a simple single thread callback mechanism
- ▶ Long callback functions -> app becomes unresponsive to user actions
- ▶ Solution: use multiple threads
- ▶ Static library `android_native_app_glue`
 - ▶ Built on top of `native_activity.h`
 - ▶ Execute callbacks and handle user input in separate threads

```
#include <jni.h>
#include <android_native_app_glue.h>
void handle_activity_lifecycle_events(struct android_app* app,
int32_t cmd) {
    __android_log_print(ANDROID_LOG_INFO, "NativeApp",
"%d: received data %d", cmd, *((int*)(app->userData)));
}
void android_main(struct android_app* app) {
    app_dummy();
    int data = 1234;
    app->userData = &data;
    app->onAppCmd = handle_activity_lifecycle_events;
    while (1) {
        int ident, events;
        struct android_poll_source* source;
        if ((ident=ALooper_pollAll(-1, NULL, &events,
(void**)&source)) >= 0) {
            source->process(app, source);
        }
    }
}
```

- ▶ Implement function `android_main`
 - ▶ Implement event loop which polls for events
 - ▶ Runs in a background thread
- ▶ Two event queues attached to the background thread (by default)
 - ▶ Activity lifecycle event queue and input event queue
 - ▶ Identify the event by ID
 - ▶ `LOOPER_ID_MAIN` or `LOOPER_ID_INPUT`
 - ▶ Additional event queues can be attached
- ▶ `android_app->userData` - transmit data to the processing function

- ▶ When event is received
 - ▶ Pointer to `android_poll_source` structure
 - ▶ Event ID, `android_app` structure, process function
 - ▶ Call `source->process` function
 - ▶ Calls `android_app->onAppCmd` for lifecycle events
 - ▶ Calls `android_app->onInputEvent` for input events
 - ▶ Implement our own processing functions
 - ▶ Set the function pointers to these functions
- ▶ In the example
 - ▶ Function called when lifecycle events occur
 - ▶ Prints `cmd` and transmitted data
 - ▶ `Cmd` is an enum defined in `android_native_app_glue.h`
 - ▶ `APP_CMD_START = 10`, `APP_CMD_RESUME = 11`, etc.

▶ **Android.mk**

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := NativeApp2
LOCAL_SRC_FILES := NativeApp2.cpp
LOCAL_LDLIBS := -llog -landroid
LOCAL_STATIC_LIBRARIES := android_native_app_glue
include $(BUILD_SHARED_LIBRARY)
$(call import-module, android/native_app_glue)
```

▶ **Build app and run**

```
04-26 17:30:13.145: I/NativeApp2(32570): 10: received data
1234
04-26 17:30:13.145: I/NativeApp2(32570): 11: received data
1234
04-26 17:30:13.155: I/NativeApp2(32570): 0: received data
1234
04-26 17:30:13.175: I/NativeApp2(32570): 1: received data
1234
```

- ▶ Implements ANativeActivity_onCreate
 - ▶ Registers callbacks and calls android_app_create function
- ▶ android_app_create
 - ▶ Initializes android_app structure
 - ▶ Creates an unidirectional pipe for inter-thread communication
 - ▶ Creates the background thread to run android_app_entry
 - ▶ The pipe is used between main and background thread
- ▶ android_app_entry
 - ▶ Looper is created
 - ▶ The two event queues are attached to the looper
 - ▶ Calls android_main (our implementation)

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

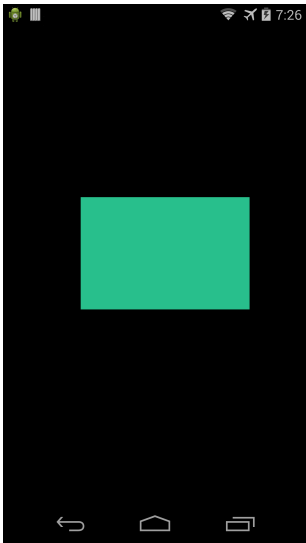
Managing Assets

```
void drawSomething(struct android_app* app) {
    ANativeWindow_Buffer IWindowBuffer;
    ANativeWindow* IWindow = app->window;
    ANativeWindow_setBuffersGeometry(IWindow, 0, 0,
WINDOW_FORMAT_RGBA_8888);
    if (ANativeWindow_lock(IWindow, &IWindowBuffer,
        NULL) < 0) {
        return;
    }
    memset(IWindowBuffer.bits, 0, IWindowBuffer.
stride*IWindowBuffer.height*sizeof(uint32_t));
    int sqh = 400, sqw = 600;
    int wst = IWindowBuffer.stride/2 - sqw/2;
    int wed = wst + sqw;
    int hst = IWindowBuffer.height/2 - sqh/2;
    int hed = hst + sqh;
    for (int i = hst; i < hed; ++i) {
        for (int j = wst; j < wed; ++j) {
            ((char*)(IWindowBuffer.bits))
[(i*IWindowBuffer.stride + j)*sizeof(uint32_t)] = (char)40;

```

```
                ((char*)(IWindowBuffer.bits))
[(i*IWindowBuffer.stride + j)*sizeof(uint32_t) + 1] = (char)191;
                ((char*)(IWindowBuffer.bits))
[(i*IWindowBuffer.stride + j)*sizeof(uint32_t) + 2] = (char)140;
                ((char*)(IWindowBuffer.bits))
[(i*IWindowBuffer.stride + j)*sizeof(uint32_t) + 3] = (char)255;
            }
        }
    ANativeWindow_unlockAndPost(IWindow);
}

void handle_activity_lifecycle_events(struct android_app* app,
                                     int32_t cmd) {
    __android_log_print(ANDROID_LOG_INFO, "NativeApp",
        "%d: _dummy_data_%d", cmd, *((int*)(app->userData)));
    switch (cmd) {
        case APP_CMD_INIT_WINDOW:
            drawSomething(app);
            break;
    }
}
```



- ▶ `native_window.h`
- ▶ Set window buffer format and size
 - ▶ `ANativeWindow_setBuffersGeometry`
 - ▶ Native window `ANativeWindow`
 - ▶ Window size - width and height
 - ▶ Format: `WINDOW_FORMAT_RGBA_8888`,
`WINDOW_FORMAT_RGBX_8888`, `WINDOW_FORMAT_RGB_565`
- ▶ Lock the next drawing surface of the window
 - ▶ `ANativeWindow_lock`
 - ▶ Returns the window buffer as argument
 - ▶ `ANativeWindow_Buffer`

- ▶ Clear buffer
 - ▶ May need to override only some part of the window
 - ▶ Otherwise set all data to 0
- ▶ Draw in the buffer
 - ▶ Set width and height
 - ▶ Compute start and end for width/height
 - ▶ Set red, green, blue, alpha bytes
- ▶ Unlock surface and post buffer to display
 - ▶ `ANativeWindow_unlockAndPost`

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

Managing Assets

```
int32_t handle_input_events(struct android_app* app,
                           AInputEvent* event) {
    int etype = AInputEvent_getType(event);
    switch (etype) {
        case AINPUT_EVENT_TYPE_KEY:
            __android_log_print(ANDROID_LOG_INFO,
                "NativeApp", "Input_event");
            break;
        case AINPUT_EVENT_TYPE_MOTION:
            __android_log_print(ANDROID_LOG_INFO,
                "NativeApp", "Motion_event");
            int32_t action, posX, pointer_index;
            action = AMotionEvent_getAction(event);
            pointer_index = (action &
                AMOTION_EVENT_ACTION_POINTER_INDEX_MASK) >>
                AMOTION_EVENT_ACTION_POINTER_INDEX_SHIFT;
            posX = AMotionEvent_getX(event, pointer_index);
```

```
        if (action == AMOTION_EVENT_ACTION_MOVE) {
            int xMove = posX - mPreviousX;
            USERDATA* userData = (USERDATA*)app->
userData;

            userData->xMove = xMove;
            app->redrawNeeded = 1;
        }
        mPreviousX = posX;
        break;
    }
}

void android_main(struct android_app* app) {
    [...]
    app->onInputEvent = handle_input_events;
    [...]
}
```

- ▶ Assign a handler for input events

```
app->onInputEvent = handle_input_events
```

- ▶ In handler, get event type

```
int etype = AInputEvent_getType(event);
```

- ▶ Two types of events defined in `android/input.h`
 - ▶ `AINPUT_EVENT_TYPE_KEY` - key event
 - ▶ `AINPUT_EVENT_TYPE_MOTION` - motion event
- ▶ `AInputEvent_getDeviceId`: id of the device that generated the input (keyboard, touchscreen, mouse, touchpad, etc.)

- ▶ `AKeyEvent.getAction`: action code
 - ▶ Down, up, multiple
- ▶ `AKeyEvent.getFlags`: key event flags
 - ▶ Soft keyboard, from system, long press, etc.
- ▶ `AKeyEvent.getKeyCode`: key code
 - ▶ The physical key that was pressed
- ▶ `AKeyEvent.getRepeatCount`: repeat count of the event
 - ▶ Key down and up events
- ▶ `AKeyEvent.getTime`: event time

- ▶ `AMotionEvent_getAction`: combined action code and pointer index
 - ▶ Action: down, up, move, cancel, etc.
 - ▶ Get pointer index
- ▶ `AMotionEvent_getFlags`: event flags
- ▶ `AMotionEvent_getX`: current X coordinate for a given pointer index
 - ▶ Whole numbers are pixels, fraction subpixels
 - ▶ Similar `AMotionEvent_getY`
- ▶ `AMotionEvent_getHistoricalX`: a previous X coordinate
- ▶ `AMotionEvent_getPressure`: event pressure for a given pointer index

Native applications

Low-level Native Activities

High-level Native Applications

Native Windows

Handling Input Events

Managing Assets


```

void displayAsset(ANativeActivity* activity){
    AAssetManager* mgr = activity->assetManager;
    AAssetDir* dir = AAssetManager_openDir(
activity->assetManager, "");
    const char* fname = AAssetDir_getNextFileName(dir);
    AAsset* asset = AAssetManager_open(
activity->assetManager, fname, AASSET_MODE_BUFFER);
    if (NULL == asset) {
        __android_log_print(ANDROID_LOG_ERROR,
"NativeApp", "_ASSET_NOT_FOUND-");
        return;
    }
    long size = AAsset_getLength(asset);
    char* buffer = (char*) malloc (sizeof(char)*size);
    AAsset_read (asset ,buffer ,size);
    __android_log_print(ANDROID_LOG_INFO, "NativeApp",
"Message_from_file:_%s", buffer);
    AAsset_close(asset);
    AAssetDir_close(dir);
}
  
```

- ▶ Access text, images, audio, video from assets directory
- ▶ Get native AAssetManager object
 - ▶ From Java: `AAssetManager_fromJava`
 - ▶ In the fully native app: `activity->assetManager`
- ▶ Open assets directory
 - ▶ `AAssetManager_openDir`
 - ▶ To open assets directory set `dirName` to ""
 - ▶ For subdirectories of assets specify directory name
- ▶ Get asset file name
 - ▶ `AAssetDir_getNextFileName`
 - ▶ Iterate over the files in the directory
 - ▶ Returns NULL - all files have been returned / no file in the directory

- ▶ Open file
 - ▶ `AAssetManager_open`
 - ▶ Mode:
 - ▶ `AASSET_MODE_UNKNOWN`: access method unknown
 - ▶ `AASSET_MODE_RANDOM`: read chunks, move forward and backward
 - ▶ `AASSET_MODE_STREAMING`: read sequentially, move forward
 - ▶ `AASSET_MODE_BUFFER`: load contents into memory, fast small reads
- ▶ Read file
 - ▶ `AAsset_read`
 - ▶ Put contents in a buffer, similar to read
- ▶ Close file
 - ▶ `AAsset_close`
- ▶ Close asset directory
 - ▶ `AAssetDir_close`

- ▶ Android Native Development Kit Cookbook, Chapter 5
- ▶ Android Recipes, A Problem - Solution Approach, Chapter 8
- ▶ Android NDK Beginner's Guide, Chapter 5
- ▶ <http://mobilepearls.com/labs/native-android-api/>

- ▶ Native activity
- ▶ Callbacks
- ▶ Looper
- ▶ Input events
- ▶ Lifecycle events
- ▶ Native window
- ▶ Drawing surface
- ▶ Key events
- ▶ Motion events
- ▶ Asset manager