**Budankov Alexey**

Q1: How to send messages to logcat from native code?

A1: See the examples in our course, chapter 9.7-9.8 (slide 218). Code example:

*SLOGE("Failed to open %s: %s", LOG\_FILE, strerror(errno)); /\* 1 \*/*

*SLOGV("Flushing %s", LOG\_FILE);*

*SLOGI("Flushed log (%d, %d of %d bytes). Waiting %d second before the next flush.", count, usedSize, totalSize, frequency);*

etc

platform implementation done in

*#include <cutils/log.h>*

*#include <cutils/logger.h>*

Q2: Can the logs be deactivated system wide?

A2: No, but you can mess with *android/log.h*

**Lyalin Sergey**

Q1: How to query the power consumption for the whole system and also for particular parts (CPU, display etc)?

A1: Check the tech paper from Google <https://source.android.com/devices/tech/power.html>

Check the official platform implementation of Battery tab in Settings app, the project app is located at <https://github.com/android/platform_packages_apps_settings> and the class doing all the magic is <https://github.com/android/platform_packages_apps_settings/blob/master/src/com/android/settings/BatteryInfo.java>

Check the Intel PowerTop open source project at <https://01.org/powertop/> and sources at: <https://github.com/fenrus75/powertop>

The *BatteryInfo.java* has an included *BroadcastReceiver* for *Intent.ACTION\_BATTERY\_CHANGED* and is taking from the passed Intent most of

the informations need it like temperature, etc.

Check the Power Tutor application website at : <http://ziyang.eecs.umich.edu/projects/powertutor/> and the source code is available at: <https://github.com/msg555/PowerTutor>

Check also the SysPower app website <https://code.google.com/p/syspower/> and the source code is available at: <https://code.google.com/p/syspower/>

Also command line alternatives + java code alternatives:

*adb shell dumpsys batteryinfo*

*public static void dumpBatteryInfo() {*

*try {*

*String cmd = "dumpsys battery";*

*Process script = Runtime.getRuntime().exec(cmd);*

*BufferedReader in = new BufferedReader(*

*new InputStreamReader(script.getInputStream()));*

*String line = null;*

*while ((line = in.readLine()) != null) {*

*Log.i ("BATTERY","Battery stats: " + line);*

*}*

*} catch (Exception ex) {*

*}*

*}*

*<uses-permission android:name="android.permission.DUMP" />*

**Panov Roman**

Q1: UI development, some example of custom controls;

A1: Check the example provided in Intel internals github - <https://github.com/mailat/internals-2013-11-11/tree/master/Wishes>

Examples provided in the git repository:

* 00\_CustomViewDraw – custom Pie component example via SDK sample
* 00\_Paint\_2D – Marakana example on how you can pain on an Canvas, a custom component DrawView
* 12\_Styles – how you can use styles, inherit styles
* OaiaAnimata – a custom SurfaceView with a animated threaded sheep
* WheelMenu – custom control with a plate that can be moved

**Korobycin Slava**

Q1: Content Providers example

A1: Check the example provided in Intel internals github - <https://github.com/mailat/internals-2013-11-11/tree/master/Wishes>

Examples provided in the git repository:

* 02\_ContentProvider – access to the list of Contacts via Contacts Provider (see ContactsActivity.java)
* 02\_ContentProvider – access to the images in gallery via MediaStore Provider (see ImageGalleryActivity.java)

**Cheportuzov Artyom**

Q1: Simple app to react on on poweroff and reboot

A1: Check the example provided in Intel internals github - <https://github.com/mailat/internals-2013-11-11/tree/master/Wishes>

I am using in these cases 2 separate Broadcast receivers ( reference <http://developer.android.com/reference/android/content/BroadcastReceiver.html> ) for catching 2 events. In the AndroidManifest.xml we must have the permission to catch these events. The application has minimal interface and is logging these states.

Examples provided in the git repository:

* 14\_BroadcastReceiver – catch the reboot event (see RebootReceiver.java)
* 14\_BroadcastReceiver – catch the poweroff event (see PoweroffReceiver.java)
* TBD reference how the system is closing all apps at reboot

**Kukanov Alexey**

Q1: 3rd party library used in my application, native library

A1: In the course materials (slide 200) we have in the 4th day a complete example on “Exposing our Native Library via Java (i.e. JNI)”. Additionally in the ANDROID/NDK/SAMPLES folder we have some examples on how the communication is done.

The theoretical part into the course is located at slide 54.

In the 5th day we have also a example of FibonacciNative using native code.

**Deev Vladimir**

Q1: After flashing a device how to test some features ( not explorative testing, but ui testing )?

A1: More ways to do it:

* see Olga responses bellow
* use command line and setup parameters like *adb shell setprop dalvik.vm.checkjni true* and after that your test
* regression test on platform side?!, unit test for the code on the platform itself

**Kasatkina Olga**

Q1: Automatic test UI

A1: Official UI testing <http://developer.android.com/tools/testing/testing_ui.html> with alternatives Robotium - <https://code.google.com/p/robotium/> and Testdroid/Testrecorder - <http://bitbar.com/>

**Kozyrev Alexander**

Q1: How to manage the activity stack in c++ ?

A1: Android framework provides an android.app.NativeActivity.java class to help us

create a "native" activity. In a typical Java activity, we extend android.app.Activity and

overwrite the activity lifecycle methods. NativeActivity is also a subclass of android.

app.Activity and does similar things. At the start of a native activity, NativeActivity.

java will call ANativeActivity\_onCreate, which is declared in native\_activity.h

and implemented by us. In the ANativeActivity\_onCreate method, we can register

our callback methods to handle activity lifecycle events and user inputs.

*At runtime, NativeActivity will invoke these native callback methods when the corresponding events occurred. In a word, NativeActivity is a wrapper that hides the managed Android Java world for our native code, and exposes the native interfaces defined in native\_activity.h.*

NativeActivity - <http://developer.android.com/reference/android/app/NativeActivity.html>

**Krasichkov Eugene**

Q1: If ( not Inject own library into a not rooted process without debuggable permissions )

* else { react on other applications launches }

A1: There is no broadcast to know when an app is launched. You can have a service running which has to periodically check the currently running tasks list to see if a particular app has been launched.

You definitely can make the service run with no notification icon, but that will simply increase the probability of your service getting killed.

You can learn about services from here : <http://developer.android.com/reference/android/app/Service.html>

You can get the list of running tasks from *getRunningTasks()* method of the *ActivityManager*.

**Kukanova Svetlana**

Q1: I’m interested in setting this property - *debug.atrace.tags.enableflags*

and notifying the system that the property has changed.

atrace.cpp in 4.4 or 4.3 (in 4.2.2 it’s atrace.c and it doesn’t do notification)

sets this property with some propert\_set func (from <cutils/properties.h>)

and then notifies the system with this:

#include <binder/IBinder.h>

#include <binder/IServiceManager.h>

#include <binder/Parcel.h>

….

sp<IServiceManager> sm = defaultServiceManager();

Vector<String16> services = sm->listServices();

for (size\_t i = 0; i < services.size(); i++) {

sp<IBinder> obj = sm->checkService(services[i]);

if (obj != NULL) {

Parcel data;

if (obj->transact(IBinder::SYSPROPS\_TRANSACTION, data,

NULL, 0) != OK)

…

but atrace is a part of Android build.

Would be great to find a way to do all the same from a c++ app built with NDK.

Or to have 100% evidence that it’s not possibleJ

Actually doing this from java code could also be an option, though I’m not sure we’ll be able to make use of it, because calling java would probably be slow.

The last option is to ask the user to set this property in the developer options, but this is not that niceJ

A1:

You can setup a system property via command line (on rooted or with system rights):

**Via command line :**

*adb shell setprop dalvik.vm.checkjni true*

**Via Java(via** [**http://goo.gl/iuuuU2**](http://goo.gl/iuuuU2) **):**

*String tag = options.getTraceTag();*

*if (tag != null) {*

*CountDownLatch setTagLatch = new CountDownLatch(1);*

*CollectingOutputReceiver receiver = new CollectingOutputReceiver(setTagLatch);*

*try {*

*String cmd = "setprop debug.atrace.tags.enableflags " + tag;*

*device.executeShellCommand(cmd, receiver);*

*setTagLatch.await(5, TimeUnit.SECONDS);*

Not sure if is helping, but here is some additional info:

*“The mechanism works by setting debug.atrace.tags.enableflags to an integer value, broadcasting a "hey everybody, re-read the flags" through Binder IPC, running the trace, and then resetting enableflags to 0. If the flags don't get set or the apps+surfaceflinger don't see the broadcast, they might not know to log data for those tags. (cf. frameworks/native/cmds/atrace/atrace.cpp in the 4.3 sources.) The way this is implemented changed quite a bit in 4.3.”*

via <http://stackoverflow.com/questions/18220366/systrace-gives-no-such-option-error-for-set-tags>

In platform the DeviceView.java is doing some kind of the same thing, setup using command line. It is not sending an event in the system.

**Malyshev Andrey**

Q1: Mechanism pattern for „sliding the screens“.

A1: Check the official ViewPager pattern - <http://developer.android.com/reference/android/support/v4/view/ViewPager.html> or older alternatives <http://actionbarsherlock.com/>

Q2: Gestures?

A2: See my next answer A3. A good example with a small app for gesture detection is to be found on the official Android page : <http://developer.android.com/training/gestures/detector.html> and basically is using the same mechanism as described in A3.

For each View (textView, Edittext etc) you can register a listener and do the following

*View myView = findViewById(R.id.my\_view);*

*myView.setOnTouchListener(new OnTouchListener() {*

*public boolean onTouch(View v, MotionEvent event) {*

*// ... Respond to touch events*

*return true;*

*}*

*});*

Q3: Event mechanism, how to intercept different types of touch gestures (singleplay, long play, zoom etc) – complex

A3: Check the events in the project Wheel Menu from the github. The class where this is visible is located in *customcontrols/WheelMenuSurface.java* in the method *onTouchedEvent*:

*public boolean onTouchEvent(MotionEvent event) {*

*if (event.getAction() == MotionEvent.ACTION\_DOWN)*

*if (event.getAction() == MotionEvent.ACTION\_UP)*

*if (event.getAction() == MotionEvent.ACTION\_MOVE)*

Q4: Sandboxing and execute system calls (what is possible)

A4: See Sergey response or use bellow code:

*String myStringArray[]= {"logcat","-d"};*

*Process process = Runtime.getRuntime().exec(myStringArray);*

Q5: How to run the compiled emulator on the Windows, which files are need it?

A5: No solution yet, use for the moment

*out/host/linux-x86/bin/emulator -sysdir out/target/product/generic/ -system out/target/product/generic/system.img -ramdisk out/target/product/generic/ramdisk.img -data out/target/product/generic/userdata.img -kernel prebuilt/android-arm/kernel/kernel-qemu -sdcard sdcard.img -skindir sdk/emulator/skins -skin WVGA800 -scale 0.7 -memory 512 -partition-size 1024*

Q6: How to upload custom images to the phone?

A6: For each device groups are different way of flashing a device. For most nexus devices is enough to follow the steps bellow:

* have a file update.zip in the root of the device sd card
* reboot in recovery mode using some combinations of keyboard
* choose the update.zip file for flashing the device
* flash the device
* reboot the device

How do you get it for Nexus 4 for example?

*make updatepackage*

*fastboot -w update $ANDROID\_PRODUCT\_OUT/$TARGET\_PRODUCT-img-eng.$USER.zip*

Official documentation is to be found on <http://source.android.com/source/building-devices.html> All depends on the bootloader and it is also possible for some devices to not be able to flash an image! My recommended resource is the one for compiling the Android 4.4 for Nexus 4 with steps also for installation and the fastboot command line at the end:

* How to build Android AOSP for Nexus 4 - <http://nosemaj.org/howto-build-android-nexus-4>
* How to build Android 4.3 for Nexus 4 - <http://nosemaj.org/build-android-4-3-nexus-4>

Nexus 4 and Nexus 5 drivers are already available at <https://developers.google.com/android/nexus/drivers> so is possible to do a Android 4.4 build for Nexus 5 and Nexus 4!

**Matrosova Ekaterina**

Q1: I run the shell in android phone and I run the apps from command line, how I can debug these crashes, dumps etc

A1: Check the extended resource <http://bytesthink.com/blog/?p=133>

**Perepelkin Sergey**

Q1: NDK samples

A1: Please refer to the sample files in /android/ndk/samples and also to our example FibonacciNative. Also a good tutorial is to be found on <https://thenewcircle.com/s/post/49/using_ndk_to_call_c_code_from_android_apps>

**Ryabtsev Dmitry**

Q1: Bad html 5 apps and reasons why

A1: We did this in class.

**Tutin Michael**

Q1: How to communicate between Android layers; how to pass data

A1: Intents - <http://developer.android.com/reference/android/content/Intent.html> , Inter Process Communication (IPC) <http://developer.android.com/guide/components/aidl.html> etc

How do I share raw data with native code?

You can store the data in a *byte[*]. This allows very fast access from managed code. On the native side, however, you're not guaranteed to be able to access the data without having to copy it. In some implementations, GetByteArrayElements and GetPrimitiveArrayCritical will return actual pointers to the raw data in the managed heap, but in others it will allocate a buffer on the native heap and copy the data over.

The alternative is to store the data in a direct byte buffer. These can be created with *java.nio.ByteBuffer.allocateDirect*, or the *JNI NewDirectByteBuffer* function. Unlike regular byte buffers, the storage is not allocated on the managed heap, and can always be accessed directly from native code (get the address with *GetDirectBufferAddress*). Depending on how direct byte buffer access is implemented, accessing the data from managed code can be very slow (more on this on <http://developer.android.com/training/articles/perf-jni.html> ).

**How to call Java from native?**

**C code**

#include <string.h>

#include <stdio.h>

#include <jni.h>

jstring Java\_the\_package\_MainActivity\_getJniString( JNIEnv\* env, jobject obj){

jstring jstr = (\*env)->NewStringUTF(env, "This comes from jni.");

jclass clazz = (\*env)->FindClass(env, "com/intel/MainActivity");

jmethodID messageMe = (\*env)->GetMethodID(env, clazz, "messageMe", "(Ljava/lang/String;)Ljava/lang/String;");

jobject result = (\*env)->CallObjectMethod(env, obj, messageMe, jstr);

const char\* str = (\*env)->GetStringUTFChars(env,(jstring) result, NULL);

printf("%s\n", str);

return (\*env)->NewStringUTF(env, str);

}

**Java code:**

public class MainActivity extends Activity {

private static String LIB\_NAME = "thelib";

static {

System.loadLibrary(LIB\_NAME);

}

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

TextView tv = (TextView) findViewById(R.id.textview);

tv.setText(this.getJniString());

}

public String messageMe(String text) {

System.out.println(text);

return text;

}

public native String getJniString();

}

**Lazarev Evgeny**

Q1: Show on the top of the application a visible layer with information

A1: Hard to find the solution but here you go <http://stackoverflow.com/questions/4481226/creating-a-system-overlay-always-on-top-button-in-android> and working example at <http://goo.gl/2qX50G>

Check also the settings->gpu equivalent source code <https://github.com/android/platform_packages_apps_settings/tree/master/src/com/android/settings>

**Leskinen Eugene**

Q1: Java and native parts in the app; the apps should use some Android specific features.

A1: This is part of the 5th day class, enjoy.

**Aranovsky Sergei**

Q1: How I can subscribe for some events from native code ?

A1: Android Native Development Kit Cookbook page 180, chapter *Detecting and handling input events at Android NDK*.

Q2: RenderScript example

A2: Official resource <http://developer.android.com/guide/topics/renderscript/compute.html> and <http://android-developers.blogspot.ru/2012/01/levels-in-renderscript.html>

Q3: Split screen apps running in the same time

A3: We discussed in class the solutions, without a modification on Application Framework we have no change. Also possible if we are on a Samsung device and we use the special split Windows Manager and the app is running as root.