Android Internals

(This is not the droid you're loking for...)

Giacomo Bergami
giacomo90@libero.it

Università di Bologna

My thesis work

Main Goals

- **Can a `pjsip`-based VoIP application (`pjsua`) run on Android?**
  The question "seems legitimate", as `pjsua` is a non-standard Java-Android application. ***It's a C-native app***.
  - Can I crosscompile a GNU/Linux application to Android?
  - Does a native application directly interact with the Kernel?
  - How does Android *know* that I want to gain access to the microphone?
  - How can I dodge Android's controls?

My thesis work

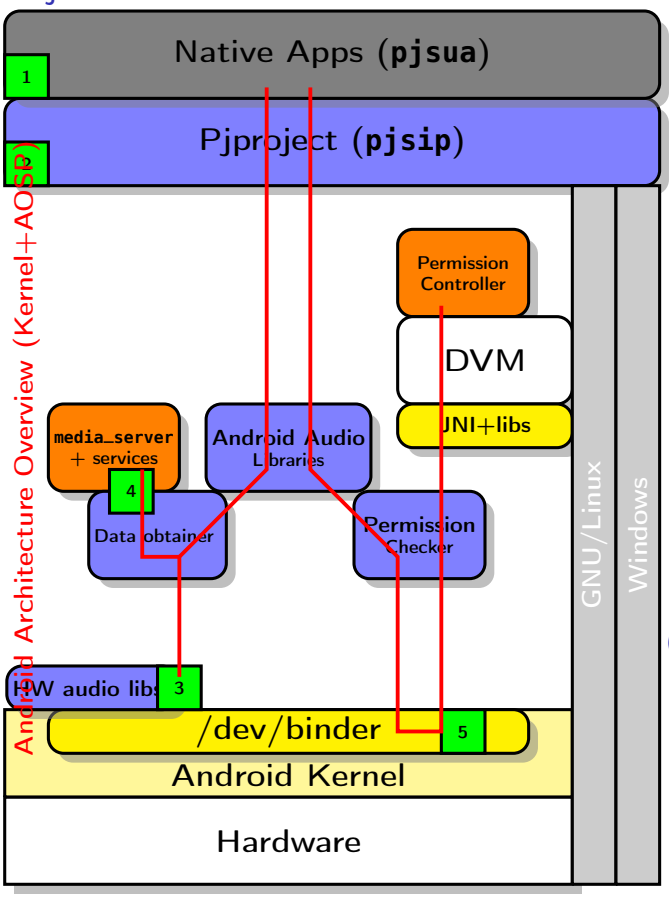
Subproblems

- Android SDK Emulator
  - Communication between emulators.
  - Audio hardware emulation is not provided.
- Olivetti Olitab (*Medion Life Tab*)
  - No factory image
  - No sourcecode support
  - Non standard "rooting" procedure (`nvflash`)

    ⇒ Samsung Galaxy Nexus.

My thesis work

PjProject Architecture

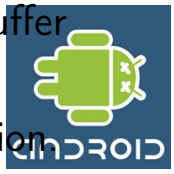


Modifications

1. Redefinition of entry-point `_start` inside Android NDK.
2. Resizing "Conference" Buffer for previous overflow.
3. Removal of the access limit to audio sampling to a client only.

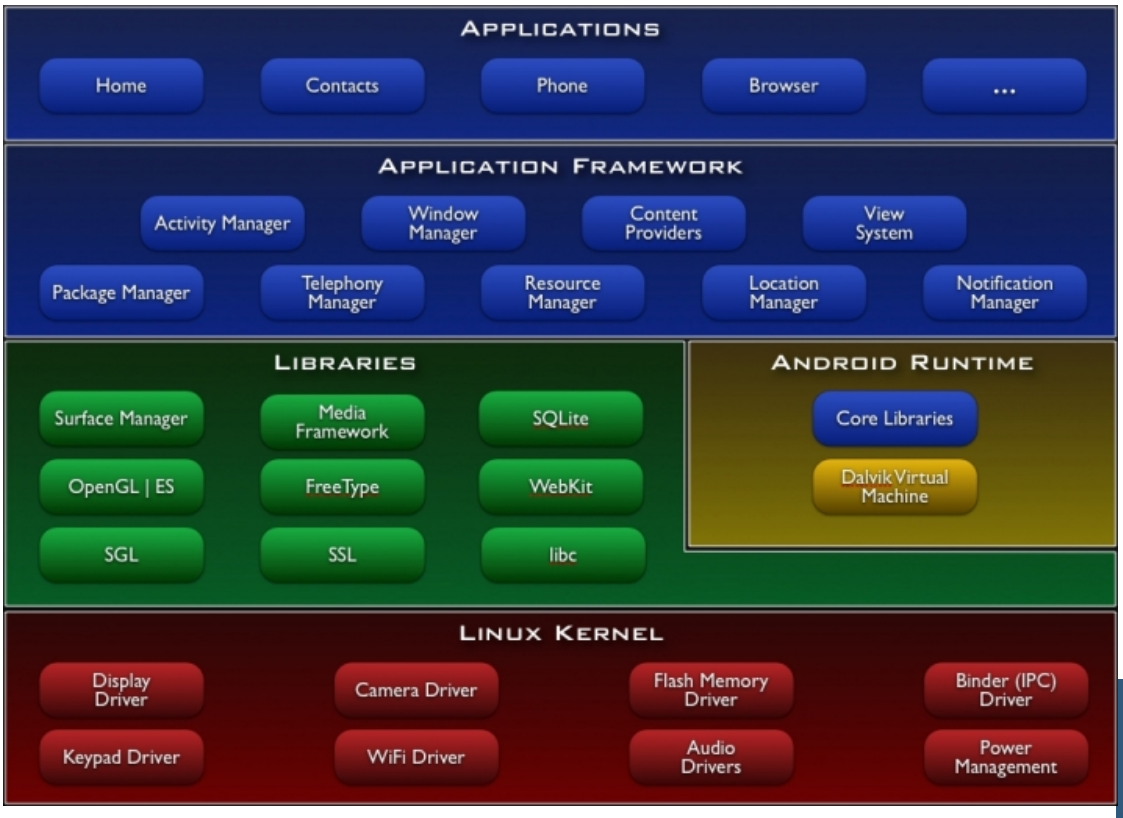
Code Analysis

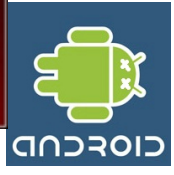
4. Analysis on the IPC Buffer for sampled audio.
5. Client/Service Interaction.



How Android is (really) made

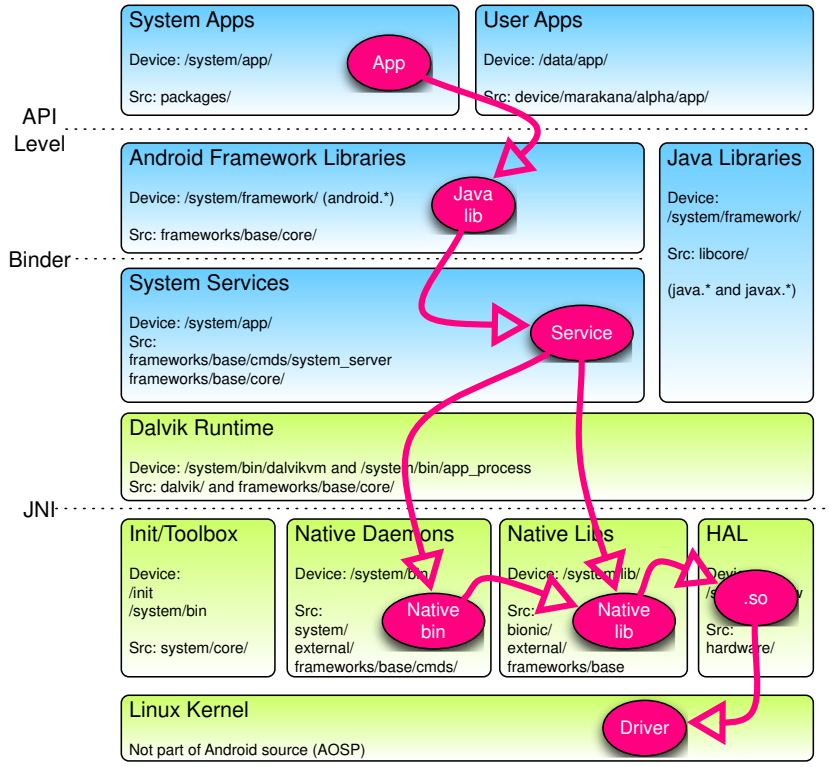
Google Point of View





How Android is (really) made

marakana.com Point of View

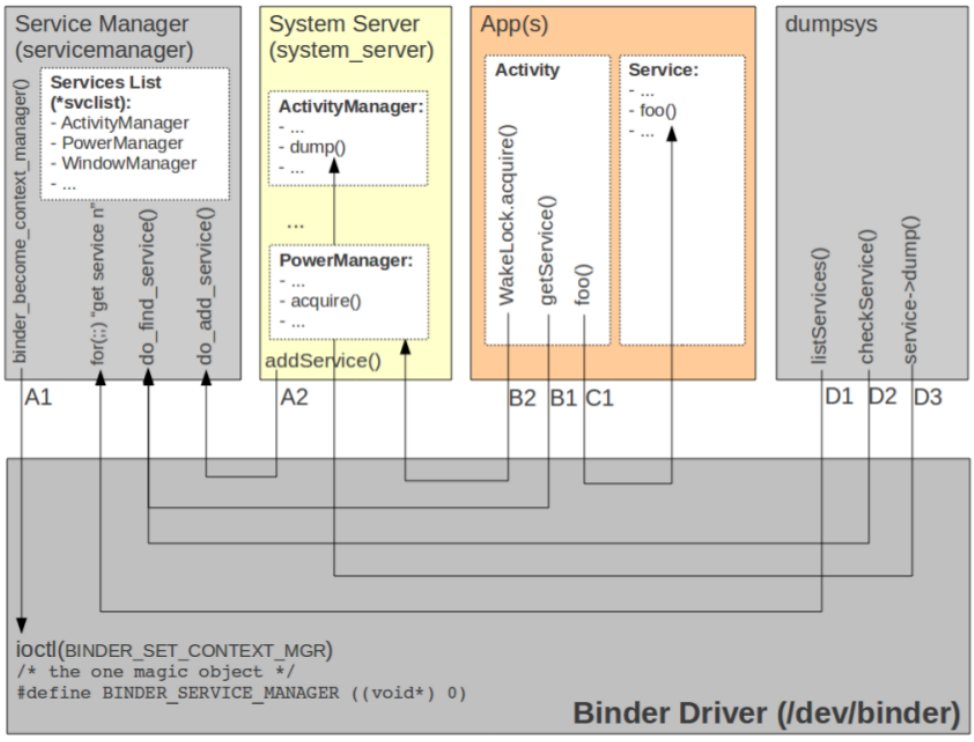


The site has been updated!!

How Android is (really) made

Yaghmour Point of View

AOSP Source, Upsyscalls and Services... But where is the "middleware"??



But what's *my* point of view? I'll explain it later...

Definitions

Android Applications

Java apps All-Java code. Compiled with javac and SDK API-s. (Good for Google Play...)

Native apps (JNI) All-Java code with JNI to access to system-dependant *ad hoc* features. (How to sell your app? - **ndk-build** script)

Native apps Using the processor directly without any DVM - *but is it for real??* (No package, no aptitude: **nerd mode on!!**)

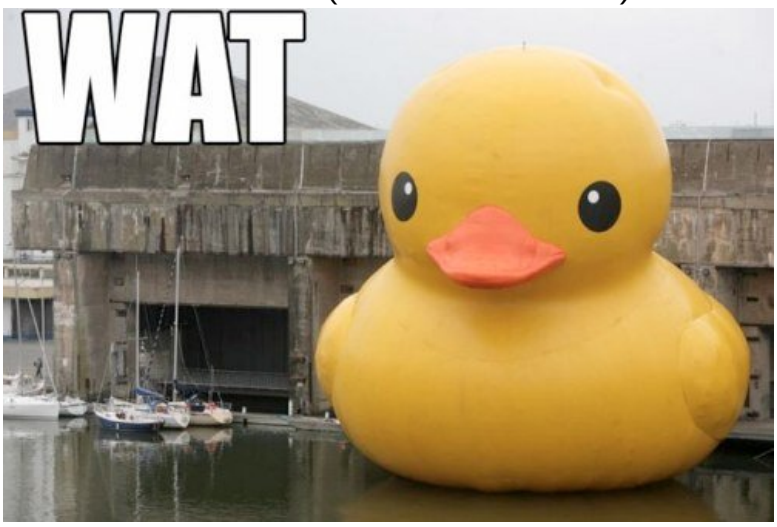
I call **Android Open Source Project Source** (*AOSP Source* for short) the superstructure that implements the **Android Middleware**, wich is the collection of services and native libraries given by Google, immediately over the Kernel Level.

Impact Therapy

Native applications (1)

Let's start with native applications...

- Is it really possible to create native applications? **yes!**
- How could we do it? **crosscompilers!**
- Does Google provide a crosscompiler? **yes!**
- Does it work? **no** (*android-ndk-r8b*)



- Why? Let's see...

Impact Therapy

Native applications (2) - NDK problems

- (NDK): The cross-compiler didn't use the `_start` entry point and the one provided (**well hidden**) didn't match with the crosscompiler version.
- An example of this entry point (`crt0.s`) is given with the sources.
- Necessary to initialize the C library... **libc?** *no, bionic.* Here's a different shared memory implementation via Android Services.

```
#define MAX 4096
#define NAME "regione"

void* data;
int fd = ashmem_create_region(NAME,MAX);
if (fd<=0) return;

if (data = mmap(NULL,MAX,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0)) {
   /* no further ancillary data is provided */
}
```

Impact Therapy

Native applications (3) - NDK Flags

- Not really essentials for SDK Emulators.
- Not necessary when you compile the AOSP.
- You must use them if you compile for a non standard ARM device.

ARMv5:

```
-march=armv5te -mtune=xscale -msoft-float\
 -fpic -ffunction-sections -funwind-tables -fstack-protector \
 -fno-exceptions -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ \
 -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__  -Wno-psabi -mthumb -Os \
 -fomit-frame-pointer -fno-strict-aliasing -finline-limit=64 \
 -DANDROID  -Wa,--noexecstack -O2 -mfpu=vfpv3-d16 -DNDEBUG -g
```

ARMv4:

```
-march=armv4t -mcpu=arm920t -mtune=xscale \
-msoft-float -fpic \
-mthumb-interwork \
-ffunction-sections \
-funwind-tables \
-fstack-protector \
-fno-short-enums \
-D__ARM_ARCH_4__ -D__ARM_ARCH_4T__ \
-D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
```

And in some cases you could simply compile...

Example

Our first Client/Server Native C program (1)

I show that we could create a mobile application and then execute it inside an Android Emulator. But first, we must setup an Android Machine. Better if without Eclipse. See for instance the Android UniBo Page: `http://www.cs.unibo.it/projects/android`. Inside the SDK folder:

```
tools/android sdk
```

installs the Android APIs for the emulator. Then we shall create an *sdcard* image in order to store our files.

```
tools/mksdcard size outfile
```

Example

Our first Client/Server Native C program (2)

Then we could create an Android Virtual Device instance.

```
tools/android create avd -n name_emu -t api -sdcard file
```

After this, we could run our new device:

```
tols/emulator -avd name_emu -partition-size 2047
```

And after that we could access the shell, push or pull some file.

```
platform-tools/adb -s number shell|push
```

where the number of the running device is given by:

```
platform-tools/adb devices
```

Example

Our first Client/Server Native C program (3)

Notice that `/sdcard` is mounted as not executable: you should place your binaries into `/data/local` and create a subfolder `./bin`.

Example

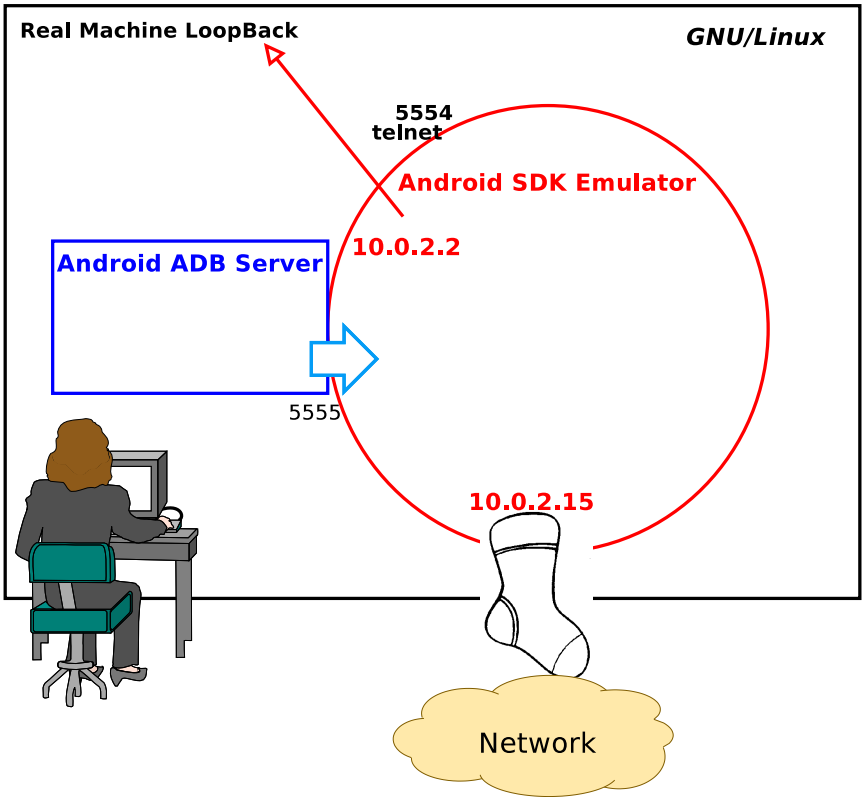
Our first Client/Server Native C program (3)

```
export LDFLAGS=" -nostdlib -Wl,-rpath-link=${ANDROID_SYSROOT}/usr
    /lib -L${ANDROID_SYSROOT}/usr/lib "
export LIBS=" -lc -lgcc -lm"
export CFLAGS=" -I${ANDROID_SYSROOT}/usr/include -I${ANDROID_TC}/
    include -mfloat-abi=softfp -mfpu=vfp -fpic -ffunction-
    sections -fstack-protector -msoft-float -Os -fomit-frame-
    pointer -fno-strict-aliasing -finline-limit=64 -
    D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -
    D__ARM_ARCH_5TE__  -DANDROID  -Wa,--noexecstack -O2 -DNDEBUG
    -g "
export CPPFLAGS=" ${CFLAGS} "
export CXXFLAGS=" --sysroot=${ANDROID_SYSROOT}"
```

- The source is given with the tarball: notice that is a simple C program. (No Google APIs whatsoever - `cliserver.c`).
- The compilation script is also provided - from PjProject (`ndk-make-test`)
  - `ANDROID_NDK` is the NDK path.
  - `API_LEVEL` selects the desired API level.
  - Selection of the **target** architecture and flags as showed above.

Example

Our first Client/Server Native C program (4)



Example

Our first Client/Server Native C program (5)

```
telnet localhost 5554
```

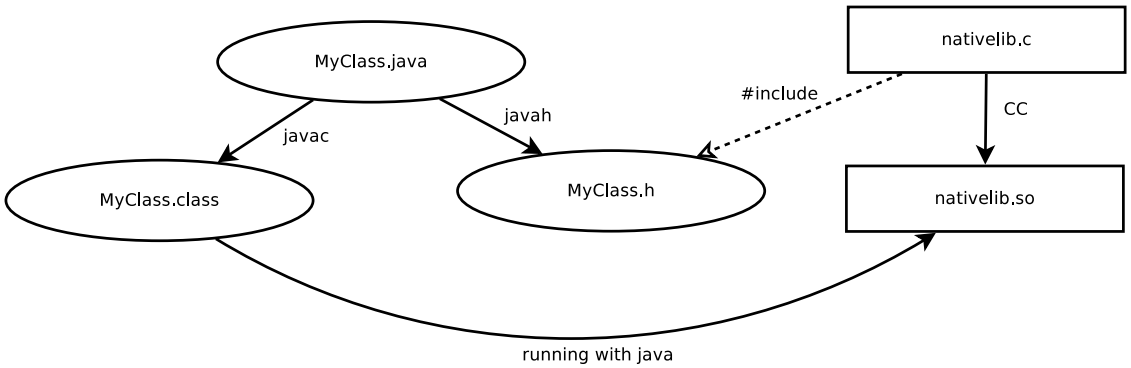
- A Telnet prompt appears.
- Invoke **help**: sms, gsm, network emulations.

```
> redir add tcp:12345:12345
```

- Pipe linking the emulator and the real host machine.
- A server in the host machine receives the requests from the emulator as they were from the real loopback.
- A server in the emulator receives the requests from the host machin as they were from the emulator loopback.

Definitions

JNI



JNI

The Java Native Interface is a programming framework that enables Java code running in a Java Virtual Machine (e.g. DVM) to call, and to be called by, native applications (programs specific to a hardware and operating system platform) and libraries written in other languaeges such as C, C++ and assembly.

— en.Wikipedia

Java:

```
class MyClass {
  private native void method();
  public void othermethod() {
       /* no further ancillary data is provided */
  }
}
```

C:

```
#include <jni.h>
#include "MyClass.h"

JNIEXPORT void JNICALL Java_MyClass_method(JNIEnv *env, jobject
    this) {
jclass class = (*env)->GetObjectClass(env,this);
jmethodID callee = (*env)->GetMethodID(env,class,"othermethod","
    ()V");
(*env)->CallVoidMethod(end,obj,callee);
}
```

Java-to-C examples:

UEventObserver Observes some netlink events at kernel level, and retrieves some informations (such as *usb plug'n'play*).

Binder A virtual Kernel Driver that implements IPC features (do you remember *marshalling/unmarshalling? Bundle Passing around Activities? Intents? Android Java "Developer Services"?*).

The Binder permits also to communicate the other way around!

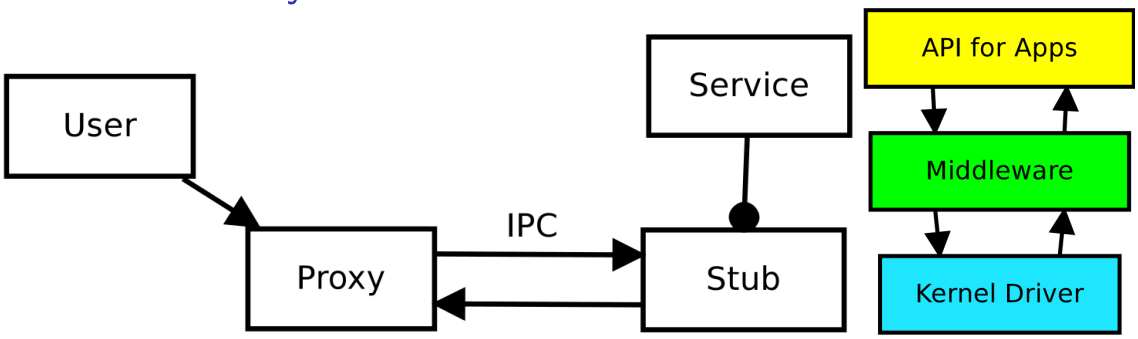
Why do we need to talk about the Binder?

- PjProject for Android uses standard Android native libraries.
- By executing a correctly compiled native binary (`pjsua`), we have that `logcat` claims that:
  - On a rooted emulator, we cannot access to the audio device (in fact, we have that the emulator don't emulate any audio). *Ergo* the simulator is not useful at all! (Some GoogleMaps problems in Java too!)
  - On a un-rooted device, a permission error while accessing the audio library.
  - On a rooted device, a permission error while performing a double access to the microphone device.
- *Let's get down to business!! Donwload the source with*

  `https://dl-ssl.google.com/dl/googlesource/git-repo/repo`

```
repo init -u https://android.googlesource.com/platform/
    manifest
repo sync
```

Binder's Anatomy



The Binder is a hierarchically structured Android Structure that is implemented over the following levels:

Java API interface It calls native methods implemented on the JNI library level.

JNI the file `android_util_Binder.cpp` links Java code and C++ "middleware" interface level.

C++ "middleware" Implements Binder middleware facilities for C++ apps.

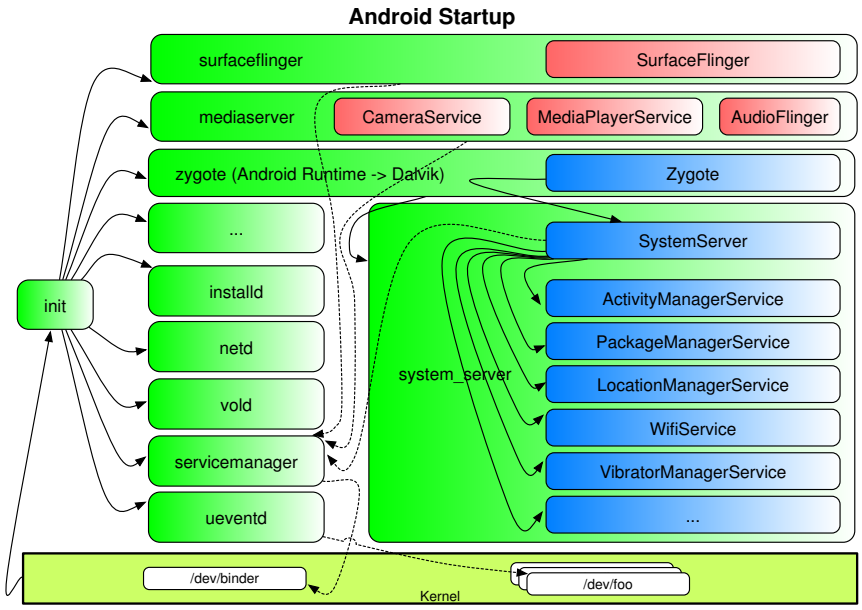
Kernel Driver Implements a driver that answeres to the primitive `ioctl`, `poll` syscalls. This code is part of the `servicemanager` itself.

System startup (1)

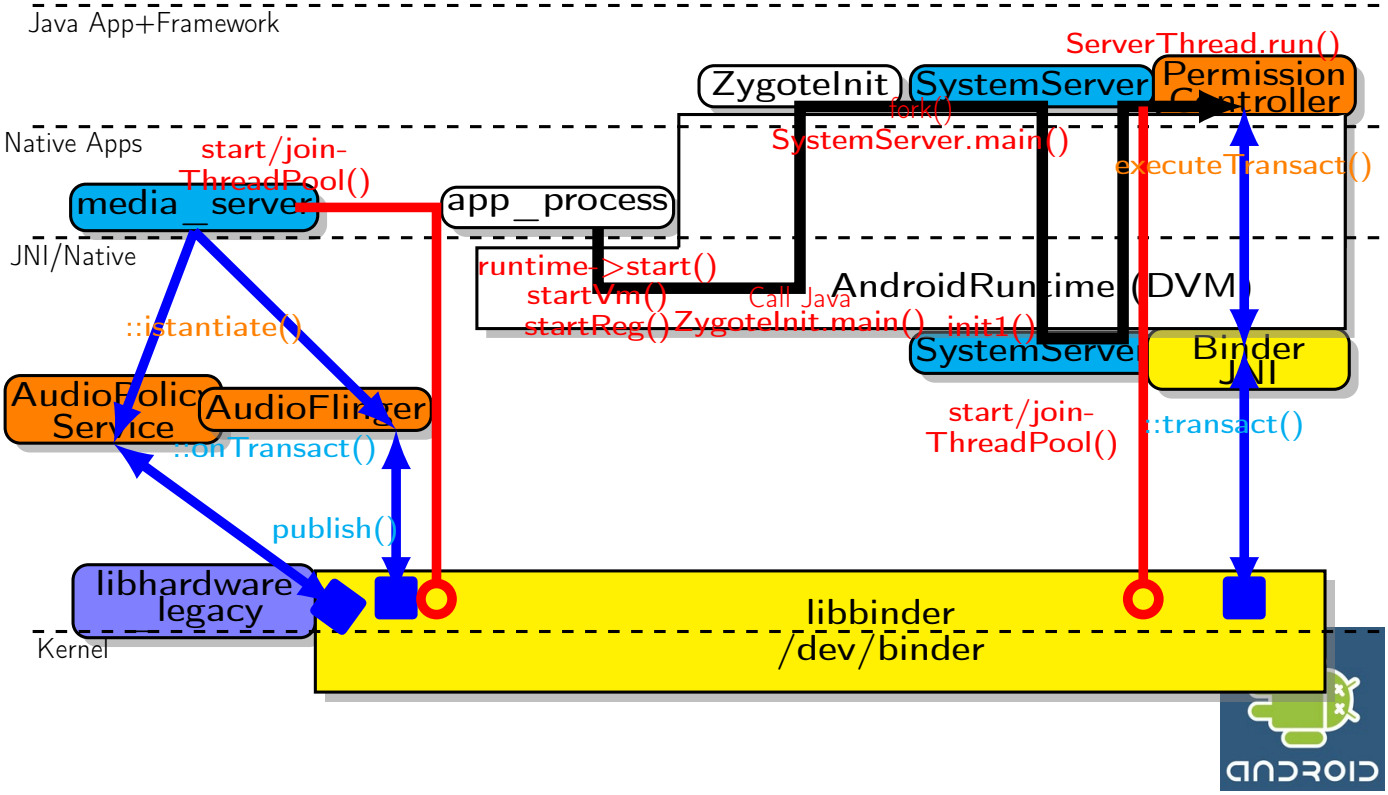
app_process Starts the DVM, which initializes the JNI layer.

Zygote Initializes the SystemServer, which registers the Java services through the Binder.java.

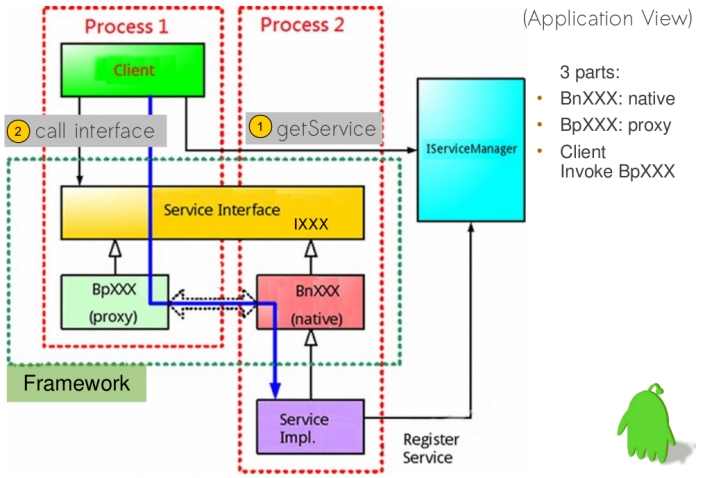
servicemanager The *Binder server*, aka the Android System *Context Manager*.

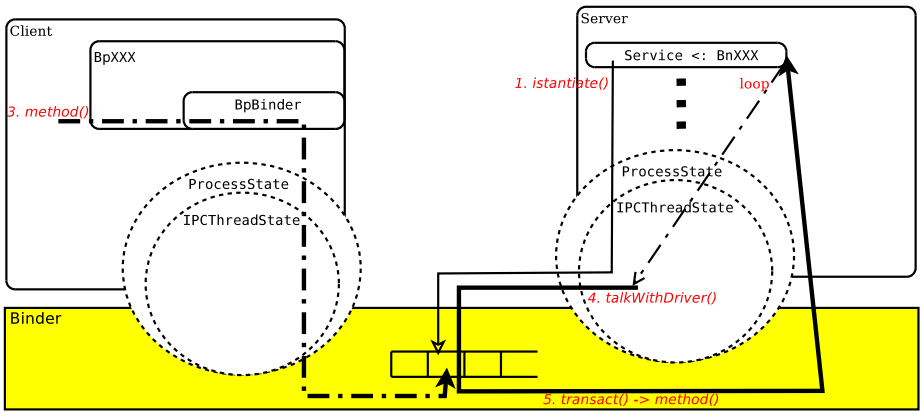


System startup (2)



C++ Services





C++ Services

Definitions

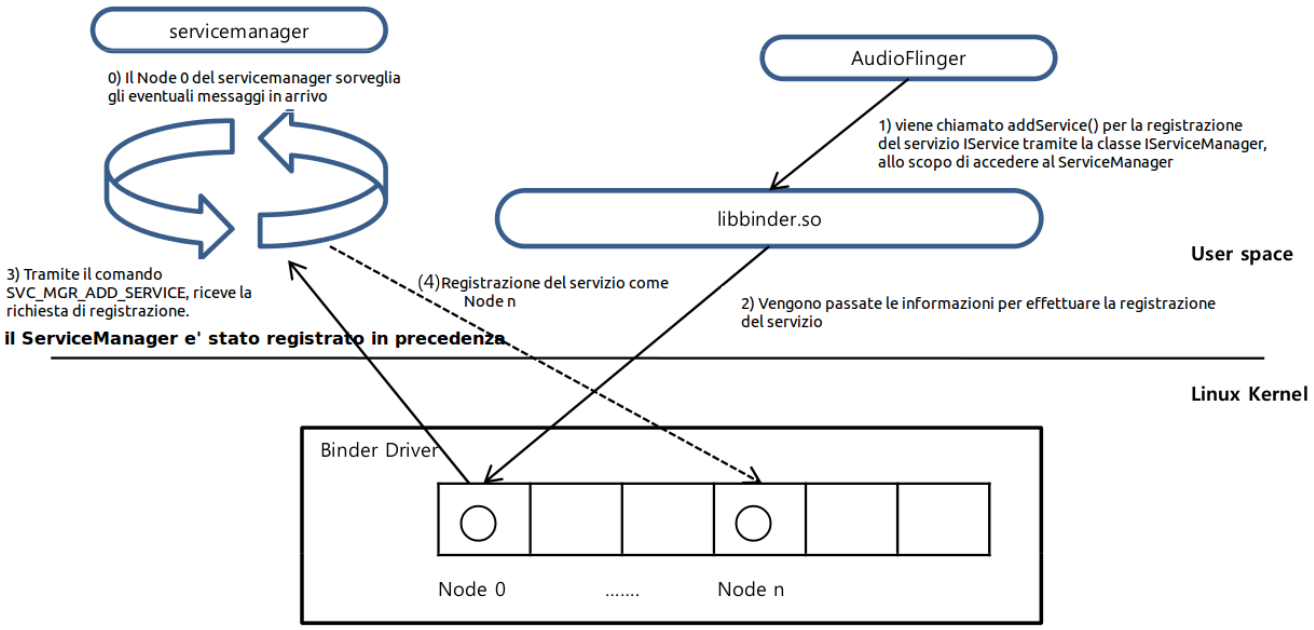
BpBinder Provides a Proxy for the C++ application (and in particular to an BpXXX implementation) via the `ProcessState` and `IPCThreadState`. It retreives services references and adds new ones.

BpXXX Is a general name for a C++ Proxy with interface `IXXX`, that is partially implemented with a `IMPLEMENT_META_INTERFACE` macro.

BnXXX Is a general name for a C++ Stub which is an abstract class implemented from the actual service. In a manner of speaking, it's the object returned from the `TalkWithDriver` method and over which the final RPC is done via some `Parcel` data.

C++ Services

Registration: A Visual Example



C++ Services

Registration: AudioFlinger Example (1)

The `media_server` initialization is given as follows:

```
using namespace android;

int main(int argc, char** argv)
{
    sp<ProcessState> proc(ProcessState::self()); //new Service
        Server
    sp<IServiceManager> sm = defaultServiceManager();//BpBinder
    AudioFlinger::instantiate(); // C++ Service Creation
    /* ... */
    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();  //Listening IPCs
}
```

Where **ProcessState** opens the Binder's Shared Memory in order to receive IPC Data (`mmap`) from the given Binder fd.

C++ Services

Registration: AudioFlinger Example (2)

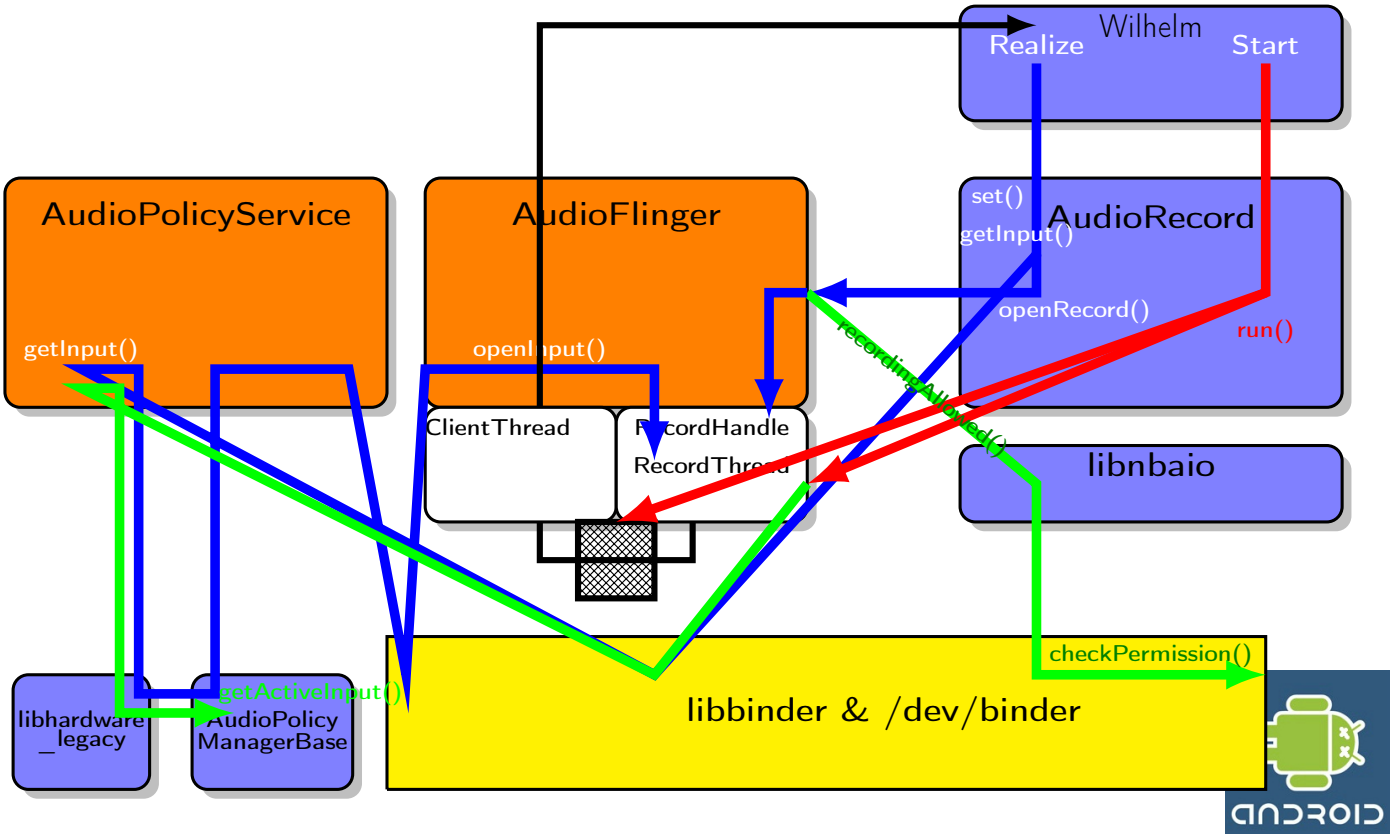
Where the registration procedes via `instantiate` as follows:

```
static status_t publish(bool allowIsolated = false) {
        sp<IServiceManager> sm(defaultServiceManager());
        return sm->addService(String16(SERVICE::getServiceName())
            , new SERVICE(), allowIsolated);
}
```

In a manner of speaking, the binder driver stores the generated `AudioFlinger` class (subclass of a `BnAudioFlinger`) as its "pointer", called **handle**.

C++ Services

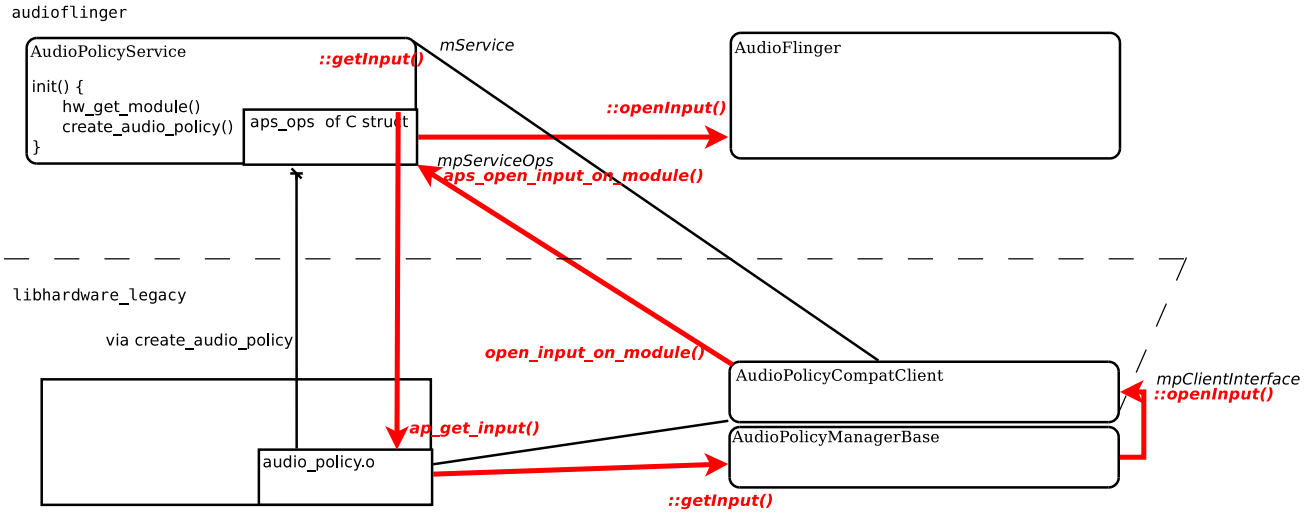
Invocation Example: recordingAllowed() and checkPermission() - (1)



C++ Services

Invocation Example: recordingAllowed() and checkPermission() - (2)

- Why all those messy lines? Because of Google's spaghetti code!



- Security issue with C-Structures and *dlopen*.

C++ Services

Invocation Example: recordingAllowed() and checkPermission() - (3)

In this example, Android firstly retreives the permission service via `BpBinder`:

```
sp<IBinder> binder = defaultServiceManager()->checkService(
    _permission); /* some other code */
pc = interface_cast<IPermissionController>(binder);
```

which will call the `asInterface` method generated via the `IMPLEMENT_META_INTERFACE` macro.

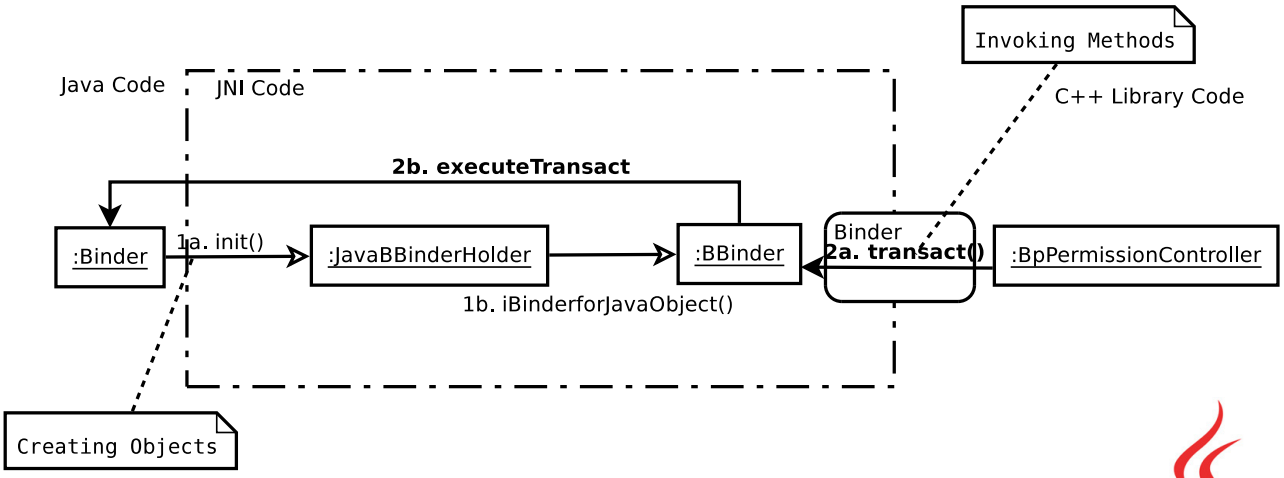
```
android::sp<I##INTERFACE> I##INTERFACE::asInterface(const android
    ::sp<android::IBinder>& obj) {
   android::sp<I##INTERFACE> intr;
   if (obj != NULL) {
       intr = static_cast<I##INTERFACE*>(
       obj->queryLocalInterface(I##INTERFACE::descriptor).get());
       if (intr == NULL) intr = new Bp##INTERFACE(obj);
   }
   return intr;
}
```

returning a `BpPermissionController` that calls transact over `BpBinder`. But where is BnPermissionController implemented, since there is no C++ class that extends it?

Java Services

Yet another Java Dirty Trick

Let's examine now the C++ "middleware" and JNI level that underly the Java Binder APIs.



Let's see the Registration and Invocation mechanism.

Java Services

Proxy and Stub Generation (1)

```
static class PermissionController extends IPermissionController.
    Stub {
    ActivityManagerService mActivityManagerService;
    PermissionController(ActivityManagerService
        activityManagerService) {
        mActivityManagerService = activityManagerService;
    }

    public boolean checkPermission(String permission, int pid,
        int uid) {
        return mActivityManagerService.checkPermission(permission
            , pid,
                uid) == PackageManager.PERMISSION_GRANTED;
    }
}
```

This is the final method that will be invoked from C++. After a few passages, we arrive to a `ActivityManager` class.

Java Services

Proxy and Stub Generation (2)

Proxy And Stubs are automatically generated in Java by **Android Interface Definition Language**.

```
package android.os;

interface IPermissionController {
    boolean checkPermission(String permission, int pid, int uid);
}
```

The `Stub.java` inside the *tarball* contains the compilation of the above example via `SDK/platform-tools/aidl`
The generated Stub is then extended in the way showed in the following slide.

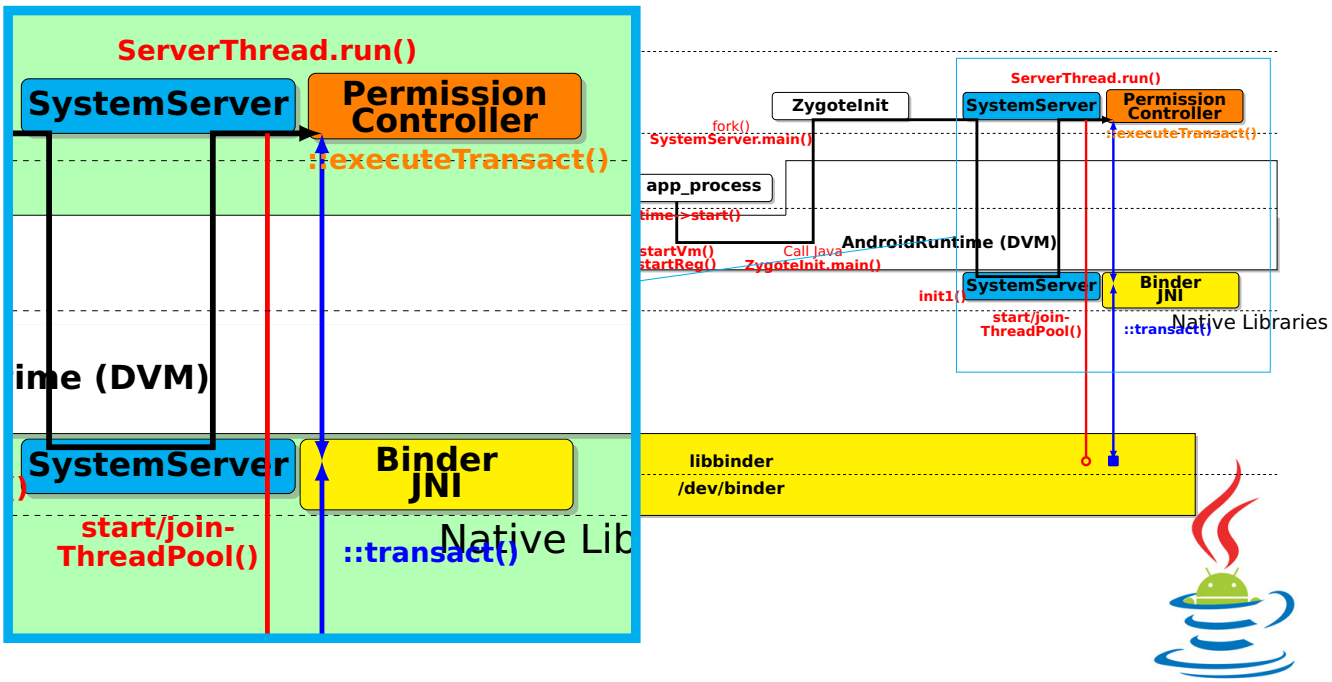
Java Services

Proxy and Stub Generation (3)

```
public static int checkComponentPermission(String permission, int
     uid, int owningUid, boolean exported) {
    // Root, system server get to do everything.
    if (uid == 0 || uid == Process.SYSTEM_UID) {
        return PackageManager.PERMISSION_GRANTED;
    }
    // Isolated processes don't get any permissions.
    if (UserId.isIsolated(uid)) {
        return PackageManager.PERMISSION_DENIED;
    }
    // If there is a uid that owns whatever is being accessed, it
         has blanket access to it regardless of the permissions
        it requires.
    if (owningUid >= 0 && UserId.isSameApp(uid, owningUid)) {
        return PackageManager.PERMISSION_GRANTED;
    }
    return AppGlobals.getPackageManager()
                .checkUidPermission(permission, uid);
    //...
}
```

Java Services

Registration at System Startup - Initialization (1)



Java Services

Registration at System Startup - Initialization (2)

Let's analyze `android_util_Binder.cpp`. As far as:

```
Binder┘ :> IPermissionController.Stub┘ :> PermissionController┘
```

the Java binder class Binder calls the native `init`, and so:

```
static void android_os_Binder_init(JNIEnv* env, jobject obj)
{
    JavaBBinderHolder* jbh = new JavaBBinderHolder();
    if (jbh == NULL) {
        jniThrowException(env, "java/lang/OutOfMemoryError", NULL
            );
        return;
    }
    jbh->incStrong((void*)android_os_Binder_init);
    env->SetIntField(obj, gBinderOffsets.mObject, (int)jbh);
}
```

Java Services

Registration at System Startup - Initialization (3)

The Binder JNI initialization is carried out as follows:

```
static int int_register_android_os_Binder(JNIEnv* env)
{
    jclass clazz=clazz = env->FindClass(kBinderPathName);
    // Obtains the reference to the Class "definition"
    gBinderOffsets.mClass = (jclass) env->NewGlobalRef(clazz);
    gBinderOffsets.mExecTransact
        = env->GetMethodID(clazz, "execTransact", "(IIII)Z");
    assert(gBinderOffsets.mExecTransact);

    gBinderOffsets.mObject
        = env->GetFieldID(clazz, "mObject", "I");

    /* ... */
}
```

We have that we memorize the ID of each method and.

Java Services

Registration at System Startup - Initialization (4)

Even Java Needs the native Context Manager to operate and so, at JNI level:

```
static jobject android_os_BinderInternal_getContextObject(JNIEnv*
     env, jobject clazz)
{
    sp<IBinder> b = ProcessState::self()->getContextObject(NULL);
    return javaObjectForIBinder(env, b);
}
```

Where `javaObjectForIBinder` casts the Binder Proxy into a Java `BinderProxy` object, in order to invoke natively the `addService` method defined in *Binder.java* method.

Java Services

Registration at System Startup - Adding Service (1)

ServiceManagerNative.java

```
public void addService(String name, IBinder service, boolean
    allowIsolated) throws RemoteException {
    Parcel data = Parcel.obtain();
    Parcel reply = Parcel.obtain();
    data.writeInterfaceToken(IServiceManager.descriptor);
    data.writeString(name);
    data.writeStrongBinder(service);
    data.writeInt(allowIsolated ? 1 : 0);
    mRemote.transact(ADD_SERVICE_TRANSACTION, data, reply, 0);
    reply.recycle();
    data.recycle();
}
```

- Passing a Java object inside the `Parcel` via a native method.
- Invoking with `mRemote` the Binder connection.

Java Services

Registration at System Startup - Adding Service (2)

In the native JNI method there is the following call:

```
const status_t err = parcel->writeStrongBinder(
    ibinderForJavaObject(env, object));
```

And for instance:

```
sp<IBinder> ibinderForJavaObject(JNIEnv* env, jobject obj)
{
    if (obj == NULL) return NULL;
    if (env->IsInstanceOf(obj, gBinderOffsets.mClass)) {
        JavaBBinderHolder* jbh = (JavaBBinderHolder*)
            env->GetIntField(obj, gBinderOffsets.mObject);
        return jbh != NULL ? jbh->get(env, obj) : NULL;
    }
    //Omissis
}
```

Java Services

Registration at System Startup - Adding Service (3)

In this case, for a correct execution, `true` is returned, and hence the get invocation produces a `JavaBBinder` object:

```
b = new JavaBBinder(env, obj);
```

that is a `public BBinder` subclass, where the following association is formed inside the constructor:

```
mObject = env->NewGlobalRef(object);
```

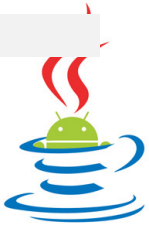
where we remember that, during the method calls we have that:

```
mObject = env->NewGlobalRef(object ≡ obj ≡ service)
```

As far as `ibinderForJavaObject` returns:

```
env->GetIntField(obj,gBinderOffsets.mObject);
```

this means returning `service.mObject`, and that will be written inside the *Parcel*, that is the `BBinder` object.



Java Services

Registration at System Startup - Adding Service (4)

Now, let's see the transaction system. Returning to `ServiceManagerNative.java`, we could see the following code:

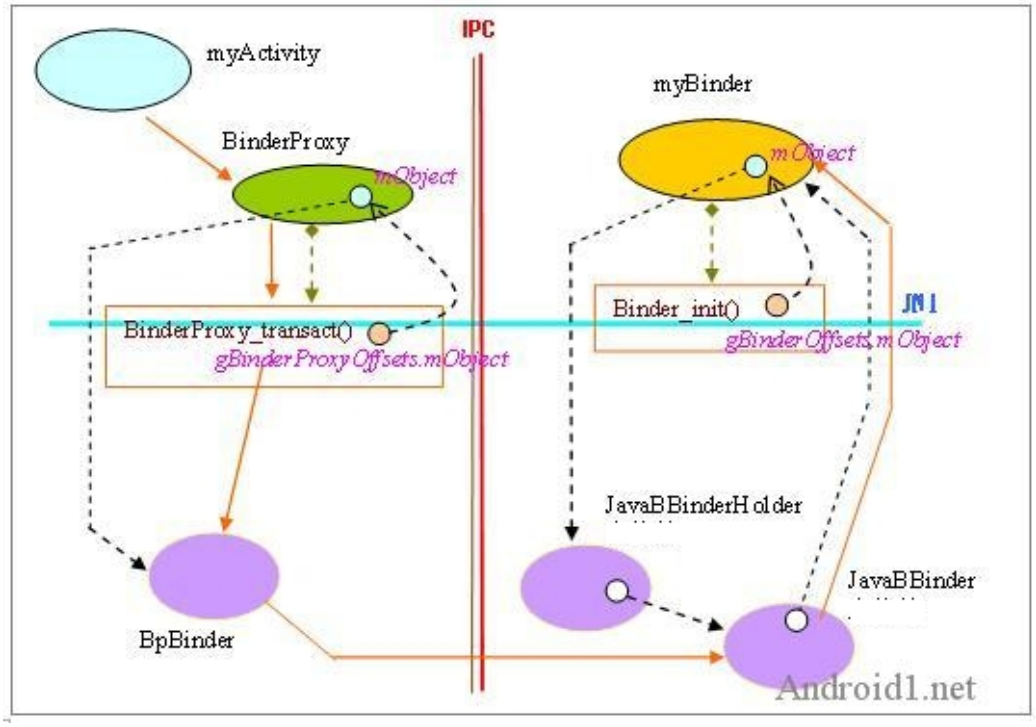
```
static jboolean android_os_BinderProxy_transact(JNIEnv* env,
    jobject obj, jint code, jobject dataObj, jobject replyObj,
    jint flags) // throws RemoteException
{
    //Error checks or logs are omitted...
    Parcel* reply = parcelForJavaObject(env, replyObj);

    //Previous Singleton
    IBinder* target = (IBinder*)
        env->GetIntField(obj, gBinderProxyOffsets.mObject);

    status_t err = target->transact(code, *data, reply, flags);
}
```

Java Services

Java Applications Interaction (New!)



I don't show how an Android Activity interacts with the Binder in order to obtain a service, but the previous considerations could explain that picture really well.

Java Services

Invocation Example: checkPermission() - (1)

- Remember the previous `checkPermission()` invocation?
- Which main loop does `PermissionController` use?
- How a C++ class could invoke a Java method, in order to call `checkPermission`?

Let's get back to system initialization...

Java Services

Invocation Example: checkPermission() - (2)

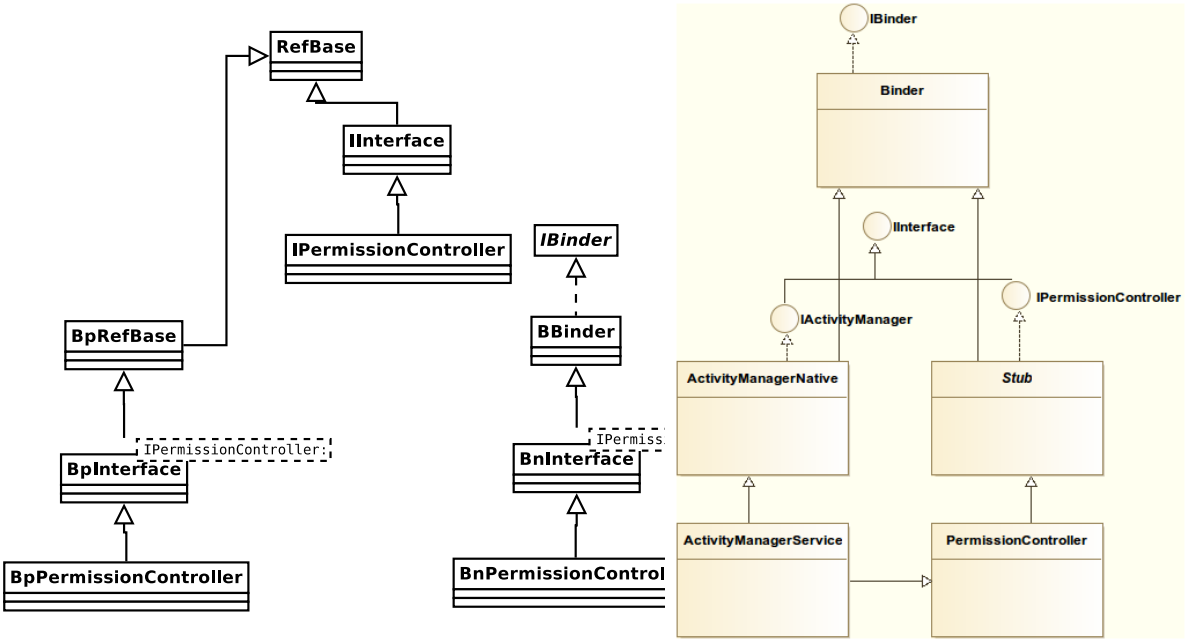
```
extern "C" status_t system_init()
{
    // And now start the Android runtime.  We have to do this bit
         of nastiness because the Android runtime initialization
        requires some of the core system services to already be
        started. All other servers should just start the Android
        runtime at the beginning of their processes's main(),
        before calling the init function.
    AndroidRuntime* runtime = AndroidRuntime::getRuntime();
    JNIEnv* env = runtime->getJNIEnv();
    jclass clazz = env->FindClass("com/android/server/
        SystemServer");
    ALOGI("System server: starting Android services.\n");
    jmethodID methodId = env->GetStaticMethodID(clazz, "init2", "
        ()V");
    env->CallStaticVoidMethod(clazz, methodId);

    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
}
```

Java Services

Invocation Example: checkPermission() - (3)

So we have our main loop. That example showed also a way to call a Java Medhod (init2). Let's analyze our class hierarchy (C++ and then Java):



Java Services

Invocation Example: checkPermission() - (4)

```
//Some check code was omitted
virtual status_t onTransact(uint32_t code, const Parcel& data,
    Parcel* reply, uint32_t flags = 0)
{
    IPCThreadState* thread_state = IPCThreadState::self();

    jboolean res = env->CallBooleanMethod(mObject, gBinderOffsets
        .mExecTransact,
        code, (int32_t)&data, (int32_t)reply, flags);
    jthrowable excep = env->ExceptionOccurred();

    // Need to always call through the native implementation of
    // SYSPROPS_TRANSACTION.
    if (code == SYSPROPS_TRANSACTION) {
        BBinder::onTransact(code, data, reply, flags);
    }
}
```

Services

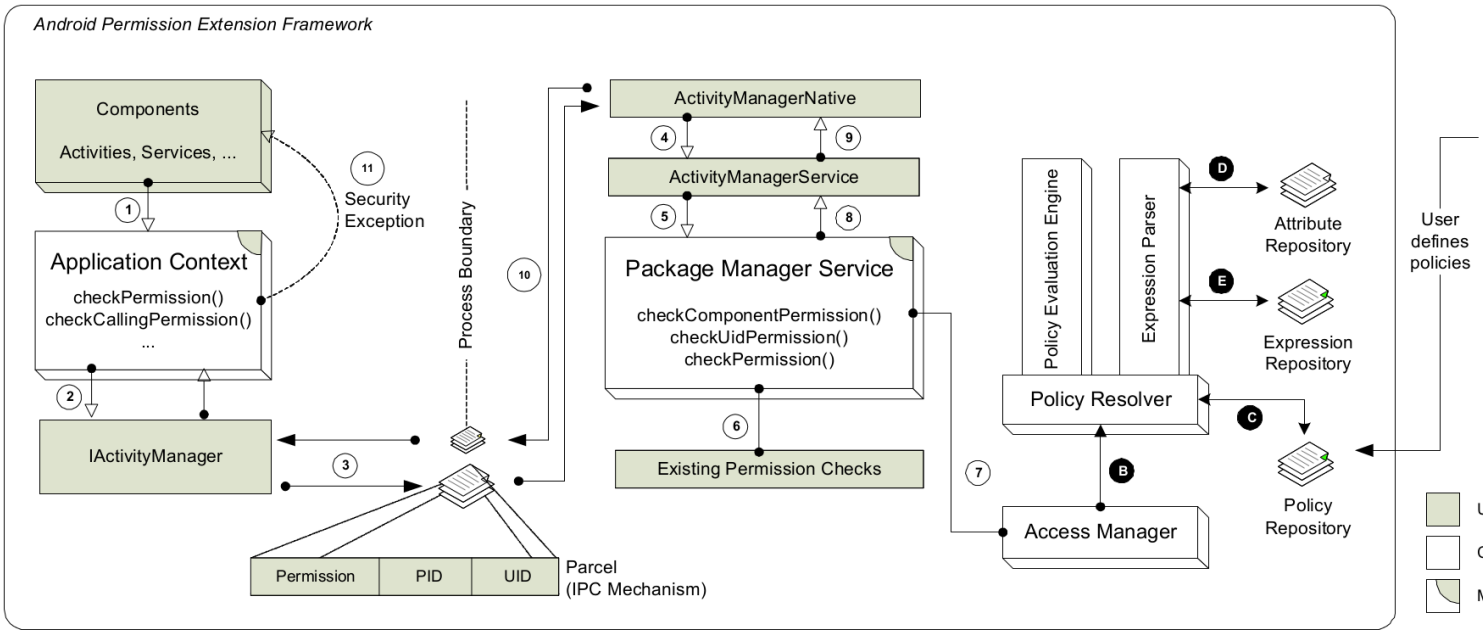
A final review (1)

- I showed how application (C++ and Java) could interact throught Binder.
- In particular, I showed how the *Wilhelm* library depends on Java based code to security issues.
- Hence, why rooting is needed? (Think, does native apps have capability lists?)
- Why we should root our devices to do what we want?

Services

A final review (2)

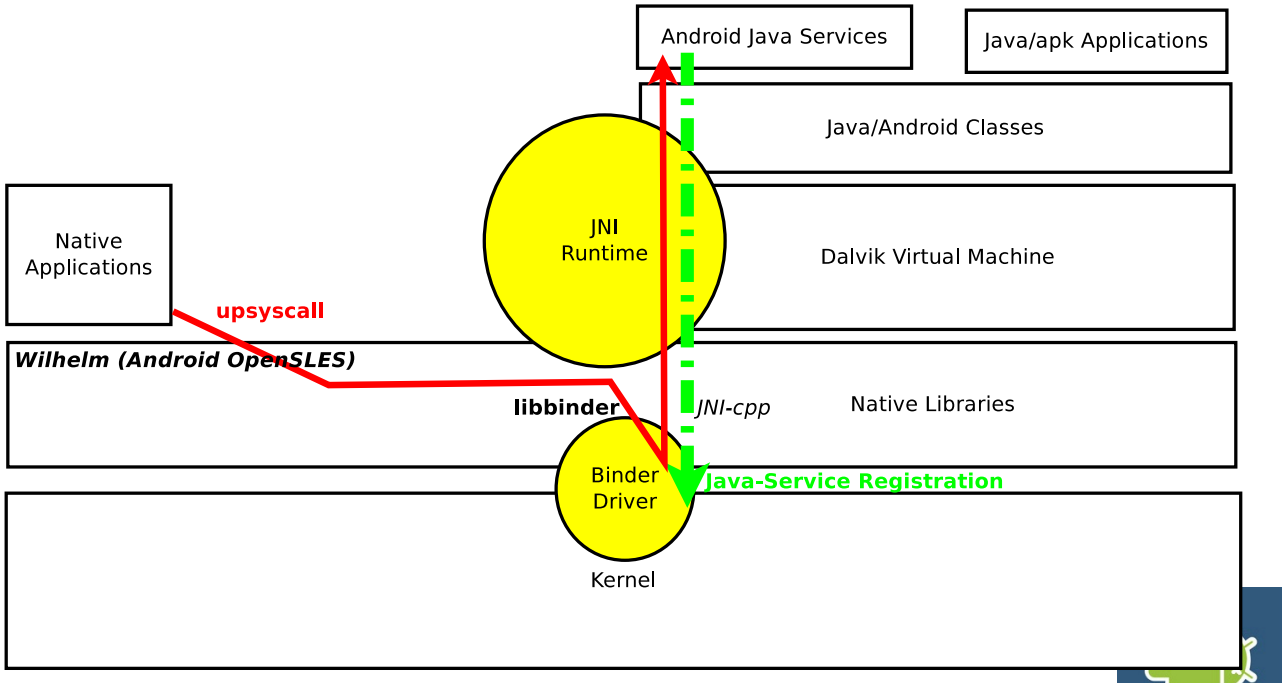
A proposed architecture by other researchers.



Services

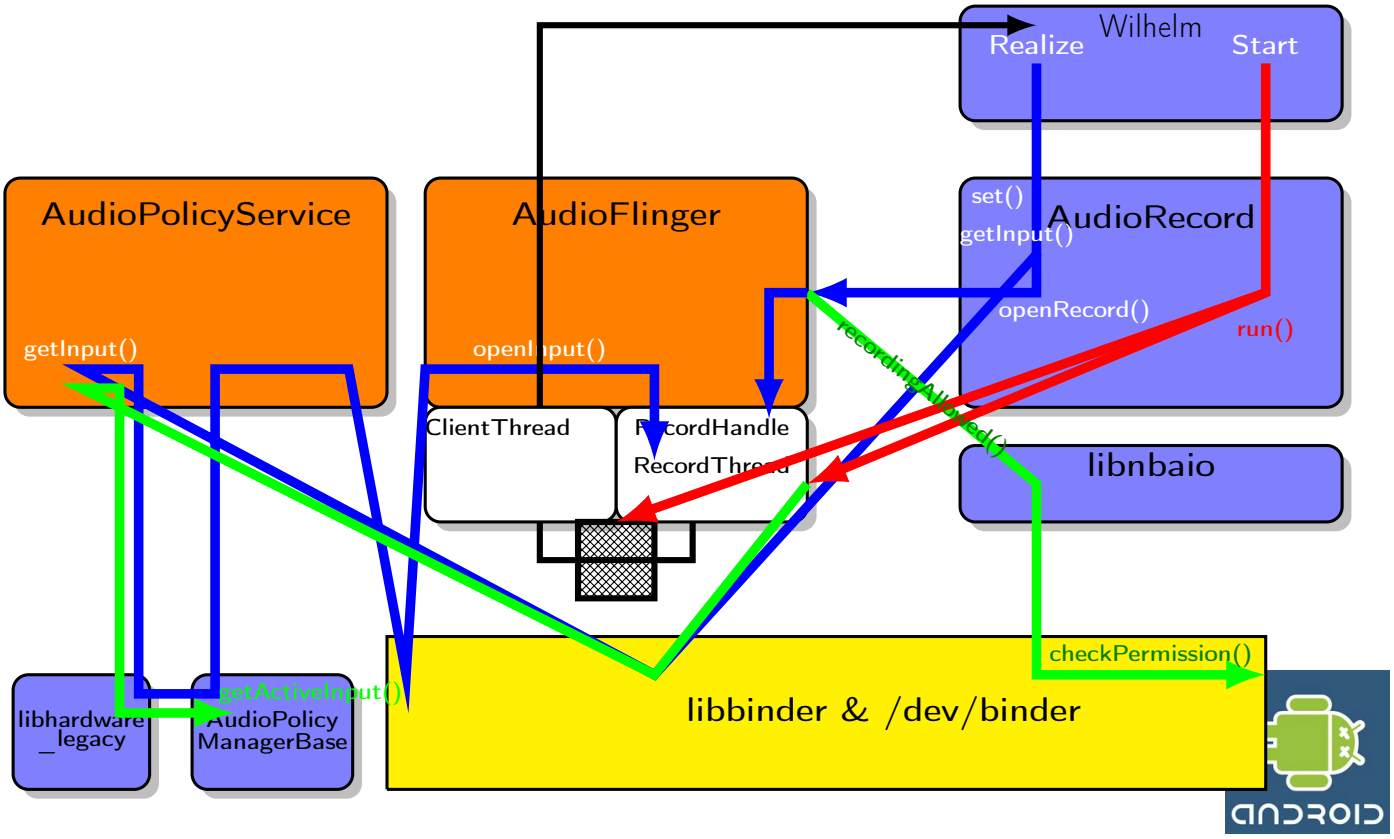
A final review (3)

A final high-level overview.



Yet Another Android Hotchpotch

AudioRecorder... Remember?



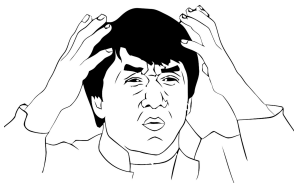
Yet Another Android Hotchpotch

AudioPolicyManagerBase

I obtained an error about having multiple devices running altogether.

```
// refuse 2 active AudioRecord clients at the same time
if (getActiveInput() != 0) {
    ALOGW("startInput() input %% failed: other input already
        started", input);
    return INVALID_OPERATION;
}
```

- Is it a bogus limitation?? Then I removed that control...
- ...And another error occurred while starting the second audio recorder: the logcat told me that no data was read from the second...
- But the first one was reading the microphone data!



ANDROID

Android AOSP compilation

Libraries needed for the compilation process

```
sudo apt-get install git-core gnupg flex bison gperf build-
    essential \
  zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
  libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
  libgl1-mesa-dev g++-multilib mingw32 openjdk-6-jdk tofrodos \
  python-markdown libxml2-utils xsltproc zlib1g-dev:i386

sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-
    linux-gnu/libGL.so

sudo apt-get install xmlto doxygen
```

Android AOSP compilation and Flashing

Java reconfiguration and compilation

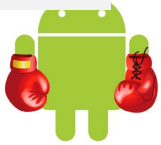
Java Reconfiguration:

```
sudo update-alternatives --install /usr/bin/java java /usr/lib/
    jvm/jdk1.6.0_33/bin/java 1
sudo update-alternatives --install /usr/bin/javac javac /usr/lib/
    jvm/jdk1.6.0_33/bin/javac 1
sudo update-alternatives --install /usr/bin/javaws javaws /usr/
    lib/jvm/jdk1.6.0_33/bin/javaws 1
sudo update-alternatives --config java
sudo update-alternatives --config javac
sudo update-alternatives --config javaws
```

Compile:

```
make clobber
. build/envsetup.sh
make
```

Now take a meal, go outside, take a trip...



Android AOSP compilation and Flashing

Flashing

Be sure you have a 3.2.x Linux Kernel... Inside the AOSP path (aosp):

```
fastboot oem unlock
export PATH=aosp/out/host/linux-x86/bin/:aosp/
export ANDROID_PRODUCT_OUT=aosp/out/target/product/maguro
cd aosp/out/target/product/maguro
fastboot -w flashall
```

Backup all your data via terminal first!!

Yet Another Android Hotchpotch

getInput()

Why to analyze this problem? I want to execute two `pjsua` instances on the same node.

```
AudioPolicyService::getInput()
↳ mpAudioPolicy->get_input()
  ↳ lap->apm->getInput() [audio_policy_hal.cpp] (ovvero
      AudioPolicyManagerBase)
    ↳ AudioPolicyManagerBase::getInput()
      ↳ mpClientInterface->openInput() [AudioPolicyManagerBase.
          cpp]
        ↳ AudioPolicyCompatClient::openInput()
          ↳ mServiceOps->open_input_on_module() [
              AudioPolicyCompatClient.cpp]
           ↳ aps_open_input_on_module() [AudioPolicyService.cpp]
             ↳ AudioFlinger::openInput()
              ↳ mRecordThreads.add(id,new RecordThread(this,...))
```

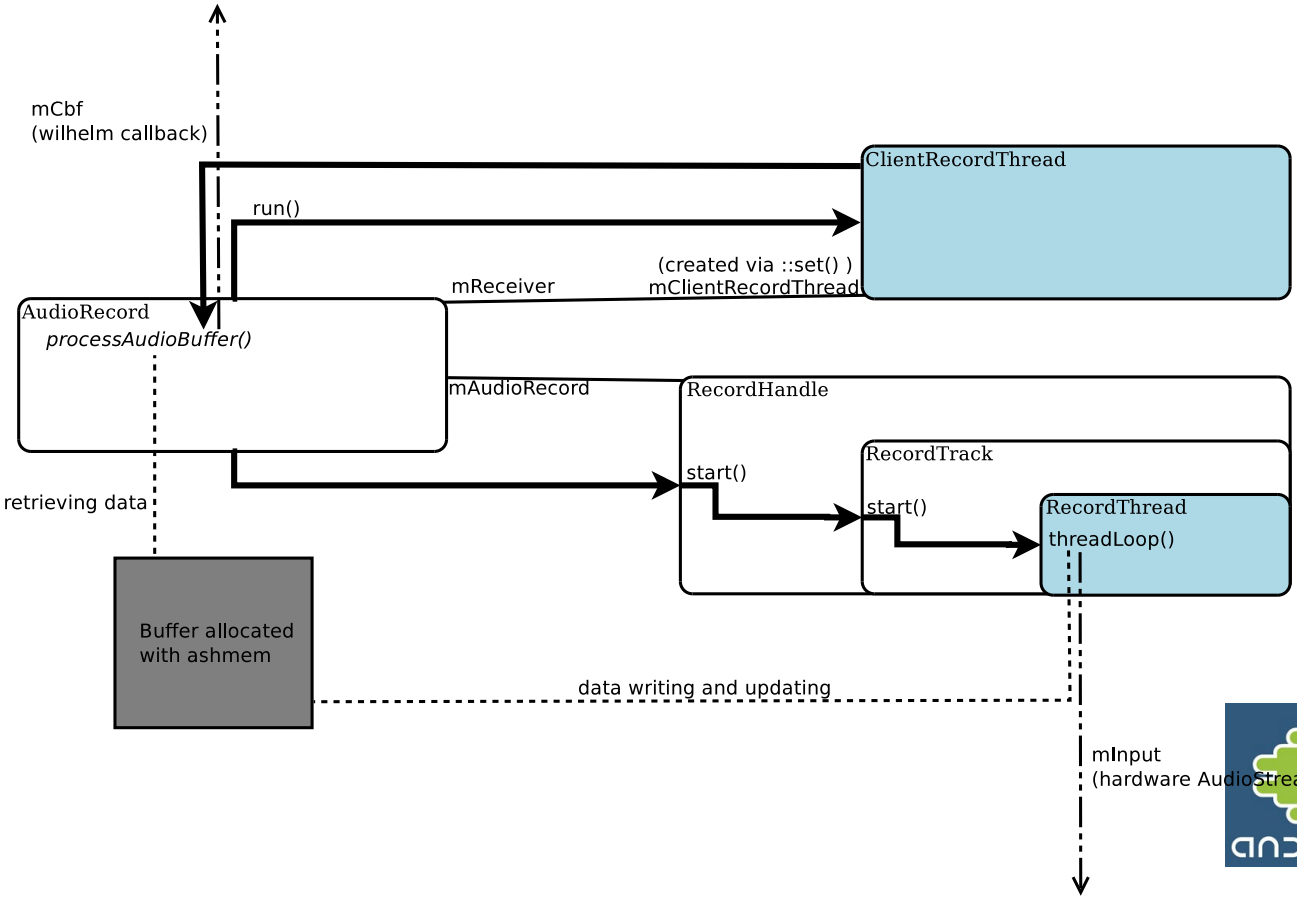
Yet Another Android Hotchpotch

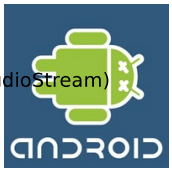
openRecord()

- The system checks for an existant `RecordThreads`: **yes!** It has been created before.
- By `registerPid_l`, a `Client` object is created in order to acheive an *ashmem* through `MemoryDealer`, initializated only after a following step.
- A `ClientRecordThread` is created, in order to send to Wilhelm data with a callback.

Yet Another Android Hotchpotch

The final Hotchpotch





That's all for Android...

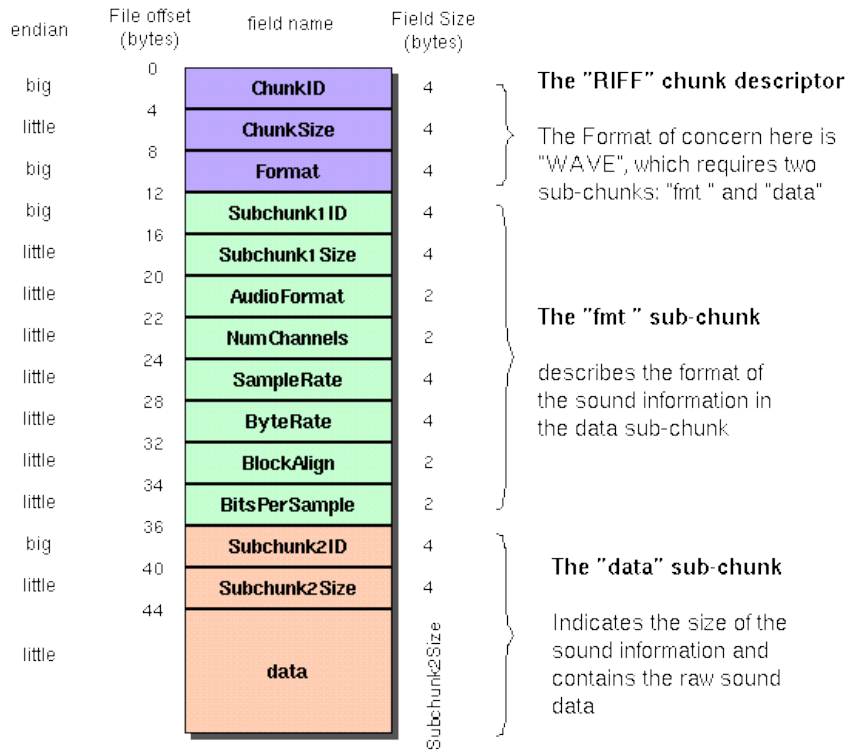
...but do not think that it's over yet!

We've seen that:

- Android Native libraries create a permission control-middleware.
- Android (4.1) doesn't support resource sharing.
- Problems with Android FileSystem system permission (statically cabled inside the AOSP).
- Now, time for some PjMedia issues...

Wave

The Canonical WAVE file format



N.B.:

SampleRate $\equiv$ *ClockRate*

Wave

The problem...

Error:

```
21:19:09.101   conference.c !WARNING: EXCEEDING. bufcount = 0,
    bufcap = 429, tmpsize=438, spf=219
21:19:09.102   conference.c  bufcount = 219, bufcap = 429,
    tmpsize=438, spf=219
21:19:09.102   conference.c  WARNING: EXCEEDING. bufcount = 219,
    bufcap = 429, tmpsize=438, spf=219
21:19:09.102   conference.c  bufcount = 438, bufcap = 429,
    tmpsize=438, spf=219
assertion "cport->rx_buf_count <= cport->rx_buf_cap" failed: file
     "../src/pjmedia/conference.c", line 1513, function "
    read_port"
```

- What is a resampling buffer?
- `bufcount` vs. `bufcap`

Wave

...and some accounting (1)

$$ByteRate = \textbf{SampleRate} \cdot BlockAlign$$

$$BlockAlign = {}^{\textbf{bps}}/_{8} \cdot \textbf{NumChannels}$$

From pjmedia:

$$\begin{aligned} spf_c &= \mu ptime_c \cdot SampleRate_c \cdot cha_c \cdot 10^{-6} \\ &= ptime_c \cdot SampleRate_c \cdot cha_c \cdot 10^{-3} \end{aligned}$$

$$ptime_\iota = \frac{spf_\iota}{cha_\iota} \frac{10^3}{clock_\iota} \quad \iota \in \{c, p\}$$

where c is for conference port, and p is for the incoming/outcoming audio port.

$$2 \cdot bufcap = tmpsize = 2 \cdot spf_c \cdot$$

Wave

...and some accounting (2)

$$bufcap = clock_p \cdot \left[10^3 \left(\frac{spf_p}{cha_p \cdot clock_p} + \frac{spf_c}{cha_c \cdot clock_c}\right)\right] \cdot 10^{-3}$$
$$= \left(\frac{spf_p}{cha_p} + \frac{spf_c \cdot clock_p}{cha_c \cdot clock_c}\right)$$

As far as:

$$bufcap = clock_p \cdot buff\_ptime \cdot 10^{-3}$$

```
if (port_ptime > conf_ptime) {
  buff_ptime = port_ptime;
  if (port_ptime % conf_ptime)
     buff_ptime += conf_ptime;
} else {
  buff_ptime = conf_ptime;
  if (port_ptime % conf_ptime)
     buff_ptime += port_ptime;
}
```

$buff\_ptime < \max\{ptime_p, ptime_c\} + \min\{ptime_p, ptime_c\} = \sum_l ptime_l$

Wave

...and some accounting (2)

And hence:

$$bufcap \approx spf_c + sfp_c \frac{1}{crate} \quad {}^1/_{crate} = {}^{clock_p}/_{clock_c}$$

Supposed that a Wave file could have max. 2 audio channels, and that in pjmedia they state that:

```
if (conf_port->channel_count > conf->channel_count)
   conf_port->rx_buf_cap *= conf_port->channel_count;
else
   conf_port->rx_buf_cap *= conf->channel_count;
```

$$bufcap \approx 2 \cdot (spf_c + sfp_c \frac{1}{crate}) \leqslant 4 \cdot spf_c$$

Insights

- From my Bachelor Thesis, of course [Italian]: `http://amslaurea.unibo.it/4441/1/bergami_giacomo_tesi.pdf`
- You could find some more informations on C++-Binder: `http://blogimg.chinaunix.net/blog/upfile2/081203105044.pdf`
- Some free infos about the JNI are given in: `http://www.soi.city.ac.uk/~kloukin/IN2P3/material/jni.pdf`
- Some more informations about the Java JNI service registration [Chinese]: `http://book.51cto.com/art/201208/353342.htm`, `http://blog.csdn.net/tjy1985/article/details/7408698`

-