Research on Audio/Video Codec Based on Android

Xiangling Fu, Xiangxiang Wu, Maoqiang Song, Mian Chen
School of Software Engineering
Beijing University of Posts and Telecommunications
Beijing, China
charlie918@tom.com

Abstract—**Due to its open-source advantage and powerful APIs, Android has attracted a large number of developers. Android's SDK is based on Java, how to reuse excellence C/C++ open-source projects is a problem. This paper first discusses a new component development approach for kernel-level development by using JNI (Java Native Interface) technology, and then describes the principle of FFmpeg (open-source codec project) and its transplantation process. At last, an Android-based codec application is designed and implemented.**

Key words-Android; JNI; FFmpeg; Codec; NDK

I. Introduction

After years of development, the third-generation mobile communication (3G) technology has gained wide acceptance, and it brings mobile users faster data transfer rate. In the 3G networks, mobile terminal is not only a communications network terminal but also an Internet terminal. Therefore, the mobile application software and services will have a great development space. Google launched a new mobile platform called Android which designed specifically for mobile device in November 2007 [1].

On June 26, 2008, Google Android released NDK, which stands for Native Development Kit, a complement for SDK [2]. The NDK allows developers to compile the C, C++ source code to native machine code and to embed it to their respective Android applications package. So a large number of excellent C/C++ open source projects can be transplanted to Android platform to enhance the platform's functionality. This paper transplants the excellent audio/video codec project FFmpeg's core libraries to Android platform, and implements audio/video codec functions in Android. The feasibility of this work will set a good example for transplanting other open-source projects.

The structure of the paper is as follows: Section II introduces the architecture of Android. Section III discusses a new component development approach for kernel-level development. Section IV briefly introduces FFmpeg and discusses how to transplant it to Android. Based on the above preparations, Section V describes the implementation of an audio/video application based on Android in detail and shows the result. Section VI concludes the paper.

II. Android Architecture

The Android architecture and its main components are shown in Fig.1 as follows [3] [4].

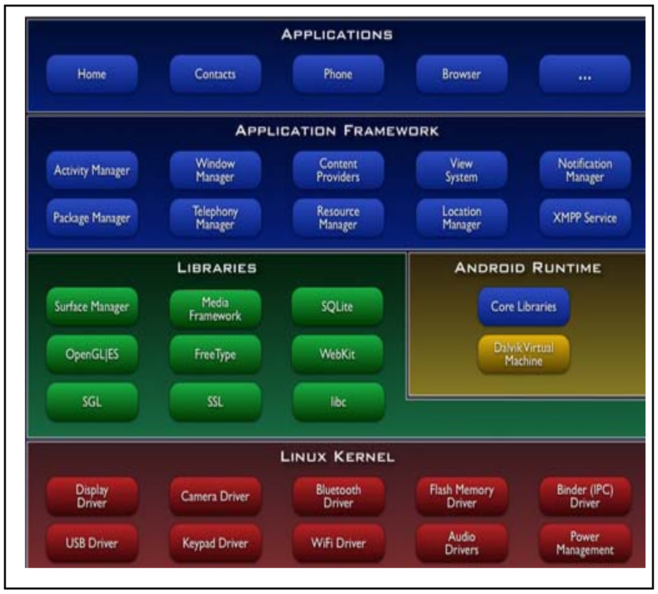

Figure 1. Android architecture.

A. Applications

A set of core applications are on the top level in the framework, including an email client, a SMS app, a calendar, a maps-application, web browser, contacts-app, and many more. All apps are written with Java programming language.

B. Application Framework

Developers have full access to the same framework APIs used by the core applications. The application framework includes a set of visual objects, a resource manager, a message manager, an activity manager and content providers for sharing their own data.

C. Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed in Fig. 1.

D. Android Runtime

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently.

Supported by the Fundamental Research Funds for the Central Universities

E. *Linux Kernel*

Android relies on Linux (Kernel version 2.6) for core system services such as memory management, process management, network stack, security, and driver model. The core also acts as a hardware abstraction layer between the applications and all the hardware.

With the whole architecture Android shows its own characteristics, such as the integration of the WebKit browser, Dalvik virtual machine modules. The application of these modules has become the focus of Android, developers can take full advantage of the interface provided by these modules to develop more distinctive applications.

III. Development of Kernel-level Component

A. *JNI Mechanism in Android*

JNI (Java Native Interface) allows programs written in Java language to interoperate programs written in other languages in the form of library (DLL, SO) or executable file, such as Assembly, C and C+ +. JNI comes from the following requirements:

1) Application needs to use the system-related functions, while Java does not support or is difficult to implement.

2) There are many ready-made class libraries or programs written in other languages. Java programs can reuse them.

3) For higher performance requirements, one needs to use assembly or C/C+ + language to implement some features.

For the same reasons, Android platform also supports the JNI mechanism, and its own component library is written by C/C++, some functions interacted with the hardware are achieved through the JNI. Therefore, this approach does not violate the overall architecture of Android. This paper uses JNI because of the second reason, for reusing excellent open-source C project FFmpeg. Fig. 2 shows the position of JNI in the application.

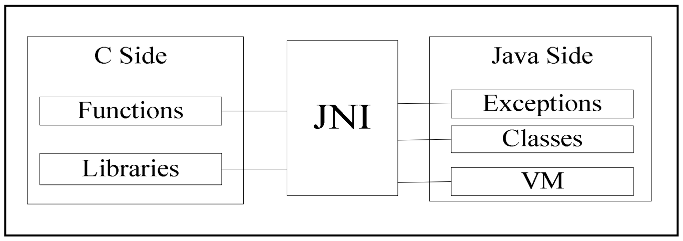

Figure 2. JNI mechanism

B. *The Steps of Kernel-level Component Development*

The steps of kernel-level component development are as follows:

1) Code kernel-level module according to Android's Linux2.6 Kernel and set aside interfaces for JNI functions.

2) Code JNI functions following JNI's rules. Programmer can call the finished kernel-level functions in JNI functions.

3) Use NDK compiler to compile it into a form of library (*. So), and load the lib into the java application package.

4) Program UI and user handling procedures by Java language under the rules of SDK. Using System.loadLibrary() function to load the lib, then JNI functions can be used [5].

IV. FFmpeg Transplantation

A. *Introduction of FFmpeg*

FFmpeg is a full open source solution including recording, conversion and codec. Through configuration it can be used in X86, ARM and other architecture platforms, it does well in cross-platform and transplantation. FFmpeg supports more than 40 kinds of encoders and more than 90 kinds of decoders. It can quickly convert the format of audio/video, get the data from the audio and video sources and save them [6].

Android platform is based on ARM Linux. FFmpeg can be transplanted into it in theory. The goal of this chapter is to compile the FFmpeg core module into the lib files which Android can identify, for calling their API in the JNI functions, achieve the purpose of using FFmpeg codec library.

B. *FFmpeg Cross-compiling*

This paper only transplants avutil, avcodec, avformat core modules of FFmpeg. Download the latest FFmpeg source file. The source code must be cross-compiled first in order to use them in ARM Linux. The cross-compiler tool is provided by NDK. Modify the configure file as follows:

```
#vi configure
prefix="/home/arm/video/libffmpeg"
cross_prefix="/usr/local/android-ndk-1.5_r1/buil
d/prebuilt/linux-x86/arm-eabi-4.2.1/bin"
cc="arm-linux-gcc"
ar="arm-linux-ar"
```

Then run the command

```
./configure--cpu=armv5te--enable-static--enable
-version3--enable-encoder=amr_nb--enable-dec
oder=amr_nb
```

JNI needs the static library of FFmpeg, --enable-static is added. After configuration, programmer commands make then commands make install. System will create the static libraries what is for the next development. The static libraries are avutil.a, avcodec.a and avformat.a.

V. Audio/video Codec Development based on Android

A. *FFmpeg Codec Requirements*

Audio/Video codec's process is shown in Fig. 3. In the encoding part, YUV video format WAV audio format data are compressed into H.263 video format and AMR_NB audio format. Then H.263 video format and AMR_NB audio format data are packed into a 3GP file. The reverse process is 3GP decoding process. Currently the test data are files, the data can also be got from cameras or network instead due to the same principle. In addition, the 3GP format can also be replaced by other formats, such as AVI, MOV, depending on whether you have loaded the corresponding codec.

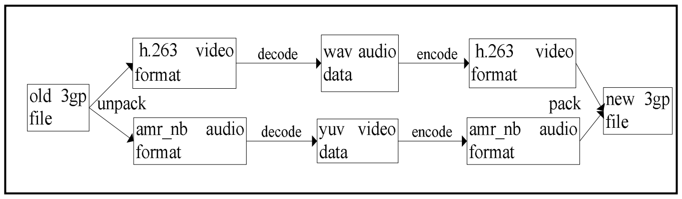

Figure 3. Codec process

B. *Programming Audio/Video Codec*

The audio/video codec programming approach is described by analyzing the encoding code as follows.

a) Encode Flowchart

The main flow of encode process is shown in Fig. 4.

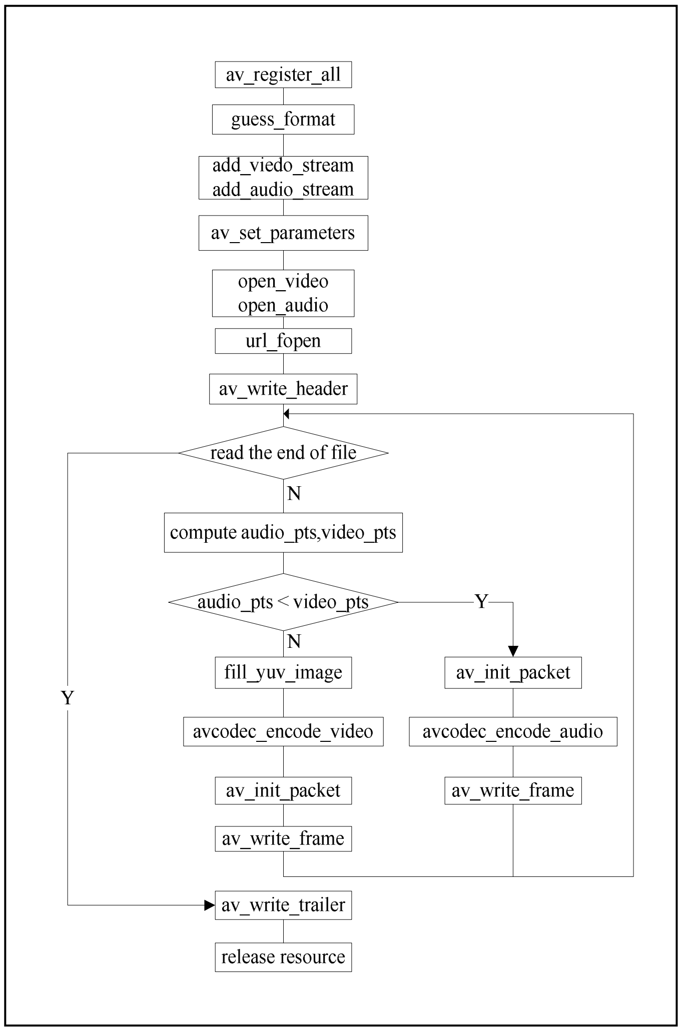

Figure 4. Encoding flowchart

Then the paper will illustrate the use of data structures core functions.

b) Main data structures

The following structures are mainly used in the encode process:

```
AVFormatContext *oc;
AVOutputFormat *fmt;
AVStream *audio_st, *video_st;
AVFrame *picture ；
AVCodecContext *c;
AVCodec *codec; //both video and audio
AVPacket pkt;
int16_t *samples;
```

c) Registration

```
//Register all the available codecs
av_register_all();
//Get the format information of the output file
fmt=guess format(NULL, ofilename, NULL);
```

d) Initialize the AVFormatContext

```
oc = av_alloc_format_context();
oc->oformat = fmt;
```

e) Set basic parameters

```
//Initialize audio/video stream,set basic parameters.
video_st=add_video_stream(oc,fmt->video_codec);
audio_st=add_audio_stream(oc,fmt->audio_codec);
//Set parameters of AVFormatContext.
av_set_parameters(oc, NULL);
```

f) Set encoder, both audio and video

```
codec = avcodec_find_encoder(c->codec_id);
avcodec_open(c, codec);
```

g) Resource allocation

```
//Video buffer
picture=alloc_picture(c->pix_fmt,c->width,c->height);
//Audio buffer
samples=av_malloc(audio_input_frame_size*c->channels)
```

h) Begin encoding

After initialization, program will alternately read a frame raw audio and video data to encode them in a while block.

```
av_init_packet(&pkt);
fill_yuv_image(picture,video_frame_count,fp2,c->wi
dth, c->height);
avcodec_encode_video(c,video_outbuf,video_outbuf
_size, picture);
```

i) End of encoding

If the application has already read the data to the end, it will break out the while block, and write the end file information to the output file, then release resources.

```
av_write_trailer(oc);
for(i = 0; i < oc->nb_streams; i++) {
av_freep(&oc->streams[i]->codec);
av_freep(&oc->streams[i]);
}
av_free(oc);
```

After completing the program of audio/video codec, JNI methods will call the codec modules to encode and decode on Android platform.

C. Writing JNI Functions

In the Java layer two native methods are declared.

```
public native String encode();
public native String decode();
```

Then C language is used to implement the native method. The methods associate the kernel-level code with the Java layer to fulfill the core codec work.

D. Compiling JNI Dynamic Library

After finishing the native code, we will write the Android.mk file of NDK. The Android.mk must include those three static libraries which have been compiled before, compile the code and libs to a dynamic library named "libendec.so", and then put it in the folder named "libs" in the application.

E. Finishing Java Application and Loading JNI Libs

Program the front UI and user handling procedures modules according to Android SDK, use method loadLibrary("endec") to load the compiled dynamic library, then java application can call the native methods as normal method.

So far, the paper has illustrated all the coding work, the next work is to show the availability of our application.

F. Results

The main program interface is shown in Fig. 5. There are four buttons in the UI, click dec3gp button can decode a 3GP file (old.3gp) to a raw audio file and a raw video file, and then click the en3gp button will encode the uncompressed data into a new 3GP file (new.3gp).

The play interfaces of the original file and the encoded file are shown in Fig. 5.

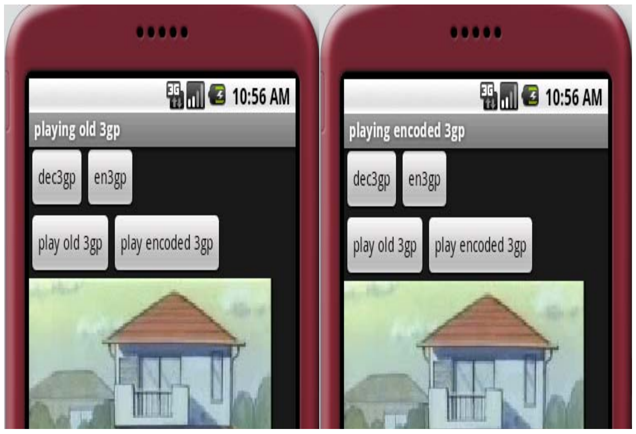

Figure 5. The play interfaces of two files

CONCLUSION

As shown in Fig. 5, the play effect between the original file and the encoded file are almost consistent with each other. That indicates the integration of FFmpeg's core libraries, codec code and Android java application framework does well. It also indicates the feasibility of transplanting open-source project to Android platform. The transplantation can expand the functionality of Android mobile phone. The future work is to design a video surveillance system based on Android.

REFERENCES

[1] OpenHansetAlliance, http://www.openhandsetalliance.com/.
[2] Android-An Open Handset Alliance Project, http://code.google.com-/intl/zh-CN/android/
[3] Android Developers, http://www.androidin.com/.
[4] C. Haseman, Android Essentials, PDF Electronic Book, 2008. Available from: http://androidos.cc/dev/index.php.
[5] N. Gramlich, Android Programming , PDF Electronic Book, 2008. Available from: http://androidos.cc/dev/index.php.
[6] Zhifeng Jiang. The quick codec development method of FFmpeg.Microcontrollers & Embedded Systems, 2008, pp. 169–71