Porting FFmpeg to Android

ChatGPT Transalted.

Origin

For those who are not yet familiar with ffmpeg, please refer to the FFmpeg common basic commands, study carefully, and see if its features can bring more fun and powerful functions to your application.

It has many functions such as synthesizing audio and video, playing various encoded audio and video, extracting specific segments, synthesizing multiple images into a video and merging with audio, format conversion, etc.

In applications like dubbing shows, dubbing is a very interesting experience, and we also intend to develop a dubbing module. Therefore, we found this tool ffmpeg and tried to port it to the Android platform. It took me about 3 to 4 days, experiencing multiple versions of attempts. Many articles online were not helpful until I found the article titled “Using ffmpeg on Android and Calling Interfaces”, which finally led to success. I tried many versions, only the combination of ffmpeg 1.2 and ndkr9 could be successfully ported.

Approach

ffmpeg is a program written in C language, which contains a main function. We first compile the libffmpeg.so file under Android NDK, and then use this library to compile the ffmpeg.c file. We modify the main function in the ffmpeg.c file to the video_merge function, that is, video_merge(int argc, char **argv). Next, we can synthesize videos like this:

This operation is similar to calling in the command line: ffmpeg -i src1 -i src2 -y output

Environment

The operating system we use is Ubuntu 12.04.

Before starting, check related tutorials first. If you encounter any problems, you can refer to this article, which introduces the problems you may encounter in that tutorial.

Modify Interface Calls

When modifying interface calls, the article used this Android.mk file:

However, I have been unable to run this file successfully.

This file is used to link libraries, that is, to compile the current module. It searches for required functions and other content in a directory, and then links to the libffmpeg.so file. Here, `-l` means linking, and `ffmpeg`

represents the name of the library. The `lib` and `.so` parts are automatically added by the system.

Therefore, I used this method:

Please note that here we first compile a shared library `myffmpeg`, and then link it to the final `ffmpeg-jni` needed.

During this process, you need to place the libffmpeg.so file in the jni directory, like this:

Debugging ffmpeg

After porting ffmpeg, we also used JNI to call functions, but we may encounter many problems. So how do we debug?

It would be great if we could get debugging information just like in the command line:

This log can help us find the corresponding position, and add a Log.i statement nearby.

In Eclipse, you can press Ctrl and click on a position similar to `av_log` to jump from the original main function entry to the function position and track step by step. It is located in the `av_log_default_callback` function in the `ffmpeg/libavutil/log.c` file, like this:

Please note the `LOGD`, it is:

Here, `...` means variable-length parameters. The simplest example of variable-length parameters is the printf function. Sometimes we can write `printf("%d", a)`, and sometimes we can write `printf("%d %d", a, a)`, the former has two parameters, and the latter has three parameters. Similar syntax is used.

And `__android_log_print` is a function provided by Android to print to Logcat.

In this way, you can view output logs to help solve problems such as unable to synthesize, not supporting mp3 or amr, or video issues.

Sometimes exceptions may suddenly occur.

How do you know where the error occurred?

Enter the following command in the command line: adb shell logcat | ndk-stack -sym obj/local/armeabi

You will find that although the synthesis was successful, the application still crashed and exited, because the original main function of ffmpeg has an exit statement at the end, just comment it out.

After successful synthesis, you may find that when called again, an error "INVALID HEAP ADDRESS IN dlfree ffmpeg" will occur. This is due to memory leaks in ffmpeg, and some dynamically allocated space has not been released. Therefore, we can put the synthesis of ffmpeg in a service, and kill the service after synthesis is completed.

AnroidMainfest.xml:

Register a receiver in your program:

In this way, you can call ffmpeg for the first time to generate a video file without sound, play it, let users do dubbing, and then call again to merge the user's voice with the video.

In my test, I found that if I record with the AAC encoder, although I can synthesize the video, I cannot play AAC files with MediaPlayer (is this a unique problem of Xiaomi 2s?). And if I use ARMNB, I cannot synthesize it either. I need to regenerate libffmpeg.so and enable ARMNB in configure. However, after enabling ARMNB, when running `./config.sh` to compile ARMNB, an error occurs, saying that it cannot find the corresponding file. After finding the file, it says that the included file cannot be found. Similarly, enabling libmp3lame also did not succeed.

The reason for wanting to play is that synthesizing a 10-second 1280*720 video and a 10-second recording takes about 1 minute. I hope users can listen to the dubbing effect before deciding whether to synthesize it.

Observing the dubbing folder of the dubbing show, you will find that the dubbing show records the user's voice, obtains tmp.amr and tmp.pcm files, uploads them to the server for synthesis, and then downloads the synthesized files.

When dubbing, the original video, subtitles, and audio with the segments to be dubbed removed have been downloaded. Therefore, when dubbing, just play this audio, and the part that needs dubbing is silent.

Using ndk-build in Eclipse

Actually, we don't have to use `ndk build` on the command line. Just select the project, right-click, select "Android tools", and then select "Add native Support". This way, `ndk build` will be automatically performed every time you run the project.

One-click generation of JNI function header files

It is cumbersome to manually write these header files every time, but we can simplify this process by one-click generation:

Configure an external tool:

Use the `javah` command to generate, and integrate it into Eclipse.

After clicking on the title bar in Eclipse, you will see a file in the jni directory similar to `com_lzw_iword_video_Myffmpeg.h`, like this:

Then, include this header file in your C file, and copy and paste the function header.