基于 SDL 的 H.264 流媒体播放系统①

江俊杰, 王志明

(南京理工大学 机械工程学院, 南京 210094)

摘 要: 针对目前广泛使用的 H.264 标准, 设计了一种基于 SDL 和 ffmpeg 的流媒体播放系统. 将经过 RTP 封装的流媒体信息解除封装处理后, 利用 ffmpeg 良好的解码能力对数据进行解码, 之后再利用 SDL 优异的视频性能进行实时显示, 并同时将流媒体数据保存在本地以供随时调用. 实验证明, 该播放系统解码播放的实时性出色, 画质良好, 此外凭借 ffmpeg 和 SDL 的跨平台特性, 系统具有良好的移植性和拓展性, 适用于嵌入式设备和手机平台.

关键词: H.264; SDL; 流媒体; 实时性; 跨平台

H.264 Streaming Media Player System Based on SDL

JIANG Jun-Jie, WANG Zhi-Ming

(School of Mechanical Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

Abstract: Aiming at widely used H.264 protocol, a kind of streaming media player system was designed based on SDL and ffmpeg. After unpacked, streaming media data packed by RTP protocol coule be decoded by ffmpeg, and shown by SDL, streaming media data could be saved in local file system for invocation at the same time. Experiment showed that the system had good real time performance and image quality. Besides, the system also had a good extensibility and portability because of the professional platform independence provided by ffmpeg and SDL, these features made the system suitable for embedded device and mobile terminal.

Key words: H.264; SDL; streaming media; real-time performance; platform independence

流媒体是一种将网络技术和流式传输技术结合起来的连续时基媒体, 对实时性要求高且数据量大[1]. RTP(Real-time Transfer Protocol)提供端对端实时媒体数据传输的协议, 其配套协议是 RTCP(Real-time Transfer Control Protocol)用来监控实时数据的传输, 适合通过组播和点播传送实时数据[2]. 以 RTP/RTCP 为传输机制, 以流媒体为载体进行实时数据传输, 是一种有效的实时数据传输手段.

文献[3]和[4]均采用 Directshow 进行解码播放, 基于 Windows 平台下的 Directshow 解码播放系统性能较好, 但是具有较大的局限性, 不适于嵌入式及手机平台. ffmpeg 是当下最先进的视频和音频流方案之一, 且具有优良的可移植性, 对 H.264 也具有良好的编解码能力, 后者被普遍认为是最有影响力的流媒体视频压缩标准[5], 因此符合 H.264 标准的视频流更适合网络媒体传输. SDL(Simple Direct Media Layers)是一种跨平台的多媒体开发库, 提供了对音视频开发的底层接口[6], 具有优异的音视频功能. 本系统设计之初考虑的是实现一款简单更改便可进行跨平台移植的播放系统, 因此选用具有良好可移植性的 SDL 和 ffmpeg 作为基础, 设计了一款基于 SDL 的 H.264 流媒体播放系统. 本文设计的系统开发流程短, 系统精简高效, 移植性好, 具有广泛的应用前景.

1 系统概述

本系统基于 SDL 和 ffmpeg, ffmpeg 采用 C 语言实现, 常被移植到各种嵌入式系统中[7], SDL 也具有类似特征. 系统代码经过简单更改后即可运行于 windows 和 Linux 等主流平台. 本文以 Linux 平台为例, 系统主

① 收稿时间:2013-05-14;收到修改稿时间:2013-06-13

要工作流程如下: 接收到经 RTP 封装的 NALU (Network Abstract Layer Unit)数据包后, 首先去除 RTP 封装, 封装的 NALU 被取出后, 根据数据类型, 把 NALU 还原成完整的一帧交由 ffmpeg 解码处理, 被解码后的一帧数据最后通过 SDL 进行实时显示. 为了方便后来的随时调用, 被解码出的 H.264 流数据会被即时存储在本地. 系统方案如图 1 所示.

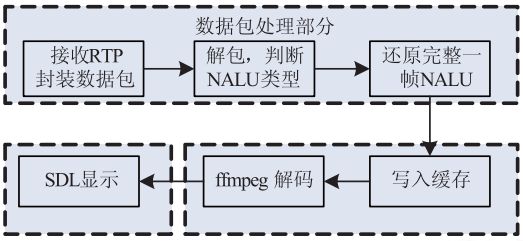

图 1　系统方案框图

2　软件平台设计

2.1 ffmpeg 及 SDL 库移植

fmpeg 和 SDL 具有良好的可移植性, 在 Linux 平台上的移植比较简单. 从网络上下载到 ffmpeg 和 SDL 后, 进行解压编译即可. 参考的编译命令如下:

ffmpeg 编译:[root@localhost ffmpeg]./configure --enable-shared--enable-gpl--enable-libx264--prefix=/ffmpeg_pc--extra-cflags=-I/ffmpeg_pc/include --extra-ldflags=-L/ffmpeg_pc/lib/

SDL 编译: [root@localhost SDL]./configure --prefix=/usr/local/SDL

2.2 解包处理

发送端将实时采集到的 H.264 数据经 RTP 封装发送到系统之后, 形成的流信息以数据包的形式被系统接收, 但此时封装的数据包中并不能被 ffmpeg 直接解码, 需要进行解包处理, 解包流程图如图 2 所示.

由于经 RTP 封装的 H.264 的 NALU 数据包大小不一, 主要分为三种: 单个的 NAL 单元分组, 聚合分组和片分组[8]. 一个完整的 NALU 可能经单包或者分包发送, 所以在接收到数据包去除 RTP 头后, 首先需要判断该数据包是否是单包, 对于单包类型, 需要进一步判断帧类型, 比较重要的是 SPS(序列参数集 Sequence Parameter Set),PPS(图像参数集 Picture Parameter Set)[9]和 I 帧, SPS 和 PPS 串包含了初始化 H.264 解码器所需要的信息参数, 包括编码所用的图像的高和宽, profile, level 等信息, 理论上在 ffmpeg 解

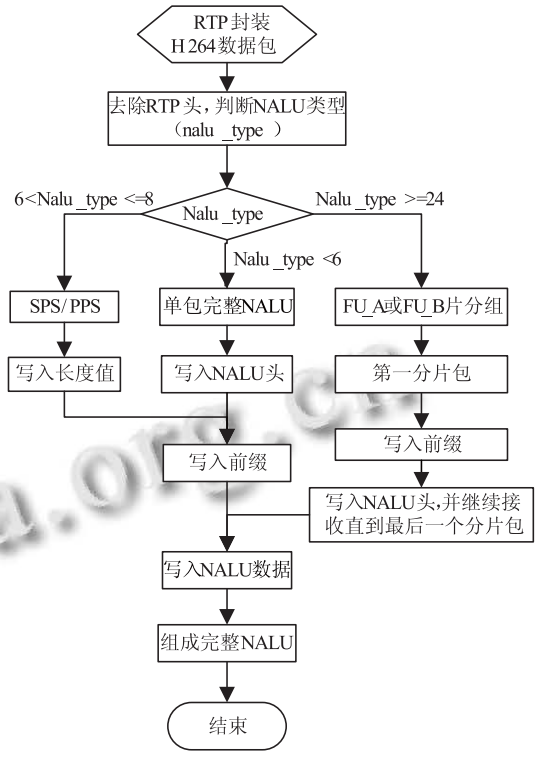

图 2　解包流程图

码每一帧前均需传入这两个参数, 但实际上在开始解码前传入一次即可. 对于 SPS 和 PPS, 除了写入前缀 (0x00 00 00 01 或者 0x00 00 01)外, 还需在此之前写入 SPS 和 PPS 的长度值, 长度值规定为 2 个字节, 表明了包括 SPS 长度表示值、SPS 前缀和 SPS 数据的总长度, 以上三部分组成了完整一个 SPS 格式, PPS 格式同 SPS. 需要指出的时, 在较新版本的 FFmpeg 中, SPS、PPS、I 帧可以分开独立输入进行解码. 对于片分组类型数据包, 通常来讲是指 FU_A 或者 FU_B 方式, 对于分包, 将至少会有两个分片包构成一个完整的 NALU, 通过判断相应标志位来判定分片包所处位置, 在取出第一个分片包后, 需要写入前缀, 写入前缀后, 该分片包仍然只是包含一个完整的 NALU 的一部分, 不能被 ffmpeg 解码, 需要继续接收直到接收到最后一个分片包后, 将这些分片包组合在一起形成完整的一个 NALU, 此时才能将该 NALU 送入 ffmpeg. 解包的目的是去除 RTP 包头, 并还原为完整的 NALU.

2.3 ffmpeg 解码

通过步骤 2.2 获得一个完整的 NALU 后, 此时可以将其送入 ffmpeg 缓存进行解码. ffmpeg 解码 H.264 的流程图如图 3 所示.

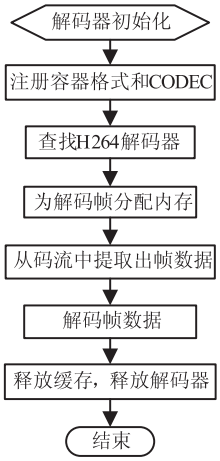

图 3 ffmpeg 解码流程图

H.264 码流由多种 NALU 类型形成, 包括 SPS、PPS、SEI、I 帧、B 帧、P 帧等, 但是如前所述, SPS 和 PPS 串包含了码流信息参数, 这些参数对于 H.264 解码器具有重要作用, 应当首先被传入 ffmpeg 进行解码. 而 ffmpeg 中最关键的数据结构是 AVCodecContext, 存储 AVCodec 和与 Codec 相关的数据[10]. 本文设计的 ffmpeg 解码 H.264 流的伪代码如下:

```
avcodec_init(); //初始化解码器
av_register_all();//注册所有容器及 CODEC
avcodec_find_decoder();//查找 H.264 解码器
avcodec_alloc_context();//分配编码器上下文
avcodec_alloc_frame();//分配解码帧缓冲
while(输入 H.264 流)
 {
avcodec_decode_video2();//解码
if(解码成功)
save_frame();//保存解码后的文件
}
avcodec_close();//关闭解码器
av_free();//释放缓存
```

值得注意的问题是, 在不同版本的 FFMPEG 中, 对于 SPS、PPS、I 帧的传入方法可能不尽相同, 这里的传入方式是指这三种类型的 NALU 是独自传入, 还是放在一起传入 ffmpeg, 但在本文的版本中, 这几种方式均可以有效解码.

2.4 SDL 显示

通过步骤 2.3 将完整一帧 H.264 流解码完毕后, ffmpeg 将 H.264 格式转为 YUV 格式视频帧, 这种格式的视频帧将能够被 SDL 程序显示播放, 本文设计的 SDL 显示程序流程图如图 4 所示.

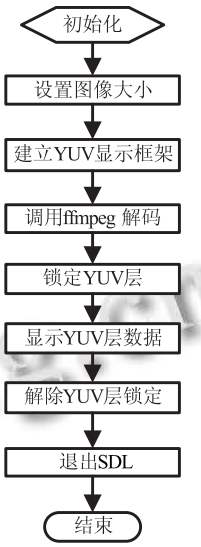

图 4 SDL 显示流程图

当由 SDL_CreateYUVOverlay()函数创建好 YUV 覆盖后, 调用 ffmpeg 解码, ffmpeg 同时提供图像缩放函数, 在解码完成后, 需要锁定 YUV 覆盖层, 利用 ffmpeg 图像缩放函数进行图像缩放, 再通过 SDL_DisplayYUVOverlay ()函数进行图像显示, 最后由 SDL_UnlockYUVOverlay()函数来释放 YUV 覆盖.

3 系统测试

本文采用的 ffmpeg 版本为(0.10 版), SDL 为 1.2.13 版, 分别运行于 PC 端 RHEL 系统和嵌入式端 Linux,PC 端硬件为 i3 处理器, 主频 3.3GHz, 2G 内存. H.264 流来源于同一局域网另一台 PC 上分辨率为 1280x720 高清 H.264 视频, 将该视频经 RTP 发送到本机, 由本机系统进行播放, 系统输出流信息如下:

Stream #0: 0: Video: h264 (High), yuv420p, 1280x720 [SAR 1:1 DAR 16:9],23.98 fps, 23.98 tbr, 1200k tbn,47.95 tbc

播放效果如图 5 所示.

可以看到, 在 PC 端, 播放高清视频效果清晰流畅, 证明系统性能良好. 然而本系统使用对象更多的是考虑嵌入式终端和手机终端, 因此另一组实验为实现嵌入式 Linux 终端的解码播放. 硬件平台为处在同一局域网中的 ARM11 开发板和 PC 机(RHEL 系统), ARM11 开发板运行嵌入式 Linux 系统, 嵌入式系统中也需移

植 ffmpeg 和 SDL, 移植步骤可以参考文献[11], 系统代码基本不需要大的修改. 在本组实验中, H.264 流由一台 PC 机通过采集 USB 图像压缩为 H.264 后, 经 RTP/RTCP 协议进行实时采集压缩传输, 开发板负责接收解码并实时显示, 在启动开发板的播放系统后, 实时画面如图 6 所示, (图 6.1 的图片来源于用相机拍摄 ARM 板显示屏播放的图像所得).

图 5　高清视频流播放效果

图 6.1　ARM 板解码播放画面

图 6.2　PC 端未压缩视频播放画面

为了进行对比, 在采集图像的 PC 端, 未经压缩的画面(格式为 YUV422, 320x240, 18fps)经 SDL 实时显示在 PC 屏幕上. 将 PC 机播放画面与 ARM 板系统播放的画面进行对比, 结果表明两者播放画面基本同步, 延时很小, 且画质并没有明显降低. 将两种类型的图像保存为本地文件后发现, 11M 大小的 YUV 格式视频被压缩为不到 300K 的 H.264 视频. 视频大小的大幅减少意味着占用带宽的减少, 但却并没有严重影响视频效果, H.264 的性能可见一斑.

4　结语

本系统采用了具有良好编解码性能的 ffmpeg 和音视频性能的 SDL 实现了对 H.264 格式流媒体的播放, 该方法开发流程短, 系统精简, 具有很好的可移植性. 两组实验表明系统解码播放性能良好, 非常适用于嵌入式系统. 然而, 本文所设计系统的视频播放能力显然不能仅限于对视频的处理. ffmpeg 和 SDL 同样音频具有良好解码播放的能力, 所以下一步工作的内容即是实现音频解码播放, 并实现音视频同步. 此外, ffmpeg 对其他格式视频也能有效支持, 在对本系统接收和解码代码进行有限类似的扩展就可实现对多种格式的视频流进行解码, 这些也将成为本文的进一步工作, 从而实现系统对多种格式流媒体的解码播放.

参考文献

1 方新勇,马瑞芳,刘萍芬,杨文革.基于 RTP 的流媒体传输系统的设计与实现.微电子学与计算机,2007,24(11):183–185.

2 Connie A, Nasiopoulos T, Leung PVCM, Fallah YP. Video paeketization techniques for enhancing H.264 video transmission over 3G networks. Consumer Communications and Networking Conference, CCNC 2008. 5th IEEE. 2008, (12): 800–804.

3 彭锋,林和志,黄联芬.基于 Directshow 的 H.264 网络视频监控客户端实现.现代电子技术,2011,34(8):118–120.

4 刘辉,魏玉琛,蒲布.基于 Directshow 的 H.264 解码器的设计与实现.电子技术应用,2011,37(9):139–141,148.

5 鲁漫红,段益群.基于 DirectShow 的 H.264 视频流过滤器的设计与实现.科学技术与工程,2009,9(9):2347–2350,2355.

6 朱靖宇,杨树堂,薛质.基于 SDL 的 Mpeg-4 视频流实时解码与回放方法.计算机应用与软件,2006,23(6):5–7,58.

7 吴张顺,张珣.基于 FFmpeg 的视频编码存储研究与实现.杭州电子科技大学学报,2006,26(3):30–34.

8 杨伟伟,胡黎玮,孟利民,胡海勇.基于H.264的无线视频监控系统设计与实现.杭州电子科技大学学报,2010,30(5):161–164.

9 ITU-T Recommendation H.264. Advanced video coding for generic audiovisual services.2005.

10 FFmpeg 工程组. FFmpeg Documentation. http://ffmpeg.org/ ffmpeg.html.

11 王莹,王华.Symbian 平台下基于 FFmpeg 的 H.264 解码器的移植.现代电子技术,2011,34(11):43–46.