DoubleLi
qq: 517712484 wx: ldbgliet

< 2021年10月 >

日	一	二	三	四	五	六
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

公告

昵称: DoubleLi
园龄: 11年9个月
粉丝: 2080
关注: 29
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类 (5164)

android(2)
ASP.NET(30)
ASP.NET MVC(11)
Boost(118)
c#(10)
C++/C(778)
c++11(15)
cmake/autotool(66)
com/ATL/Activex(75)
Css(16)
CxImage(12)
darwin stream server(3)
DataBase(32)
DirectX(16)
Extjs(13)
更多

随笔档案 (3864)

2021年10月(33)
2021年9月(4)
2021年8月(10)
2021年7月(43)
2021年6月(1)
2021年5月(29)
2021年4月(15)
2021年3月(13)
2021年2月(96)
2021年1月(47)
2020年12月(2)
2020年11月(27)
2020年10月(44)
2020年9月(14)
2020年8月(4)
更多

FFMPEG基于内存的转码实例——输入输出视频均在内存

我在6月份写了篇文章《FFMPEG基于内存的转码实例》，讲如何把视频转码后放到内存，然后通过网络发送出去。但该文章只完成了一半，即输入的数据依然是从磁盘文件中读取。在实际应用中，有很多数据是放到内存的，比如播放从服务器接收到的视频，就是在内存中的。时隔2个月，项目终于完成了，虽然在收尾阶段会花费大量时间，但也算空闲了点。于是就继续完善。

本文中，假定的使用场合是，有一个已经放到内存的视频，需要将它转码成另一种封装格式，还是放到内存中。由于是测试，首先将视频从文件中读取到内存，最后会将转换好的视频写入到另一个文件以检查是否正常。当然，限于能力，代码不可能适合于所有情况，但却可以清楚演示出自定义的IO输入输出的用法。

技术要点简述如下：

1、用户自定义的操作

对于内存的操作使用结构体封装：

```
typedef struct AVIOBufferContext {
    unsigned char* ptr;
    int pos;
    int totalSize;
    int realSize;
}AVIOBufferContext;
```

输入、输出均使用该结构体：

```
AVIOBufferContext g_avbuffer_in;
AVIOBufferContext g_avbuffer_out;
```

实现，read、write、seek函数。其中read为读取时使用到的，其它2个是写入内存要使用的。以read为例：

```
static int my_read(void *opaque, unsigned char *buf, int size)
{
    AVIOBufferContext* op = (AVIOBufferContext*)opaque;
    int len = size;
    if (op->pos + size > op->totalSize)
    {
        len = op->totalSize - op->pos;
    }
    memcpy(buf, op->ptr + op->pos, len);
    if (op->pos + len >= op->realSize)
    op->realSize += len;

    op->pos += len;

    return len;
}
```

实质进行的是读取已有内存的size数据，拷贝到buf中。opaque方便参数传递。注意，在拷贝时要对pos累加。

其它函数类似。

2、输出配置关键代码：

```
    avio_out =avio_alloc_context((unsigned char *)g_ptr_out, IO_BUFFER_SIZE, 1,
                &g_avbuffer_out, NULL, my_write, my_seek);

    avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);
    if (!ofmt_ctx)
    {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    ofmt_ctx->pb=avio_out; // 赋值自定义的IO结构体
    ofmt_ctx->flags=AVFMT_FLAG_CUSTOM_IO; // 指定为自定义
```

这个跟上述提到的文章是一致的。只是多了个自定义的结构体。

3、输入配置关键代码：

文章分类 (2)
SilverLight(1)
sql server(1)

参考博客
linux驱动
回忆未来-向东
Nginx模块开发与原理剖析
大坡3D软件开发
Dean Chen的专栏
Sloan
音视频FFmpeg等
opencv教程
个人开发历程知识库
关注DirectX
chenyujing1234
morewindows
雷霄骅(leixiaohua1020)的专栏
ffmpeg参考
webrtc参考一
更多

阅读排行榜
1. Nginx之location 匹配规则详解(240898)
2. cmake使用方法详解(178488)
3. MinGW安装和使用(103448)
4. RTMP、RTSP、HTTP视频协议详解（附：直播流地址、播放软件）(102153)
5. C语言字符串操作总结大全(超详细)(94264)

评论排行榜
1. 非IE内核浏览器支持activex插件(37)
2. Nginx之location 匹配规则详解(19)
3. Javascript中定义类(15)
4. C++中的头文件和源文件(9)
5. RTSP协议详解(8)

推荐排行榜
1. C++中的头文件和源文件(25)
2. Nginx之location 匹配规则详解(22)
3. Javascript中定义类(12)
4. JavaScript中typeof知多少？(11)
5. MinGW安装和使用(9)

最新评论
1. Re:windows下搭建nginx-rtmp服务器
configuration-nginx.bat执行报错啊 'auto' 不是内部或外部命令，也不是可运行的程序 或批处理文件。 '--conf-path' 不是内部或外部命令，也不是可运行的程序 或批...
--猫爷°
2. Re:深入理解linux系统下proc文件系统内容
怎么联系作者
--ㅇㅋ
3. Re:go mod模式下引用本地包/模块(module)的方法
go mod用法，不错
--立志做一个好的程序员
4. Re:谷歌浏览器Chrome播放rtsp视频流解决方案
目前市面上已经有很成熟且商用Chrome播放海康威视大华的H.264或H.265的RTSP视频流解决方案

```
avio_in =avio_alloc_context((unsigned char *)g_ptr_in, IO_BUFFER_SIZE, 0,
        &g_avbuffer_in, my_read, NULL, NULL);
if (!avio_in)
{
    printf( "avio_alloc_context for input failed\n");
    ret = AVERROR_UNKNOWN;
    goto end;
}
// 分配输入的AVFormatContext
ifmt_ctx=avformat_alloc_context();
if (!ifmt_ctx)
{
    printf( "Could not create output context\n");
    ret = AVERROR_UNKNOWN;
    goto end;
}
ifmt_ctx->pb=avio_in; // 赋值自定义的IO结构体
ifmt_ctx->flags=AVFMT_FLAG_CUSTOM_IO; // 指定为自定义
if ((ret = avformat_open_input(&ifmt_ctx, "wtf", NULL, NULL)) < 0)
{
    printf("Cannot open input file\n");
    return ret;
}
if ((ret = avformat_find_stream_info(ifmt_ctx, NULL)) < 0)
{
    printf("Cannot find stream information\n");
    return ret;
}
```

对于avio_alloc_context的赋值和输出一样，只是没有了write和seek。对于输入所用的AVFormatContext变量，用avformat_alloc_context来分配。由于是读内存的数据，因此avformat_open_input就不用指定文件名了。

我在代码中尽量加了注释，下面是代码：

```
/**
他山之石，学习为主，版权所无，翻版不究，有错无责
                    Late Lee  2015.08
基于内存的格式封装测试(从内存视频转换到另一片内存视频)
使用
./a.out a.avi a.mkv

支持的:
avi mkv mp4 flv ts ...

参考:
http://blog.csdn.net/leixiaohua1020/article/details/25422685

log
新版本出现:
Using AVStream.codec.time_base as a timebase hint to the muxer is
deprecated. Set AVStream.time_base instead.

test passed!!

mp4->avi failed
出现:
H.264 bitstream malformed, no startcode found, use the h264_mp4toannexb bitstream filter
解决见:
http://blog.chinaunix.net/uid-11344913-id-4432752.html
官方解释:
https://www.ffmpeg.org/ffmpeg-bitstream-filters.html#h264_005fmp4toannexb


ts -> avi passed

其它:

1、传递给ffmpeg的avio_alloc_context中的内存p和大小size，可以使用32768。
如果转换后的数据保存在内存p1，这个内存p1一定要和前面所说的p不同。因为
在自定义的write中的buf参数，就是p，所以要拷贝到其它内存。
```

了，就是猿大师中间件，底层调用VLC的ActiveX控件可实现在网页中内嵌播放多路RTSP的实时...

--喵大侠

5. Re:如何使用UDP进行跨网段广播

主机A：192.168.3.100 子网掩码255.255.0.0（手动临时修改）主机B：192.168.120.100 子网掩码255.255.255.0 主机A 广播192.168.255...

--zzhilling

```
如定义p为32768，但定义p1为50MB，可以转换50MB的视频
测试：
p为32768时，需调用write 1351次
2倍大小时，调用write 679次
p越大，调用次数最少，内存消耗越大
(用time测试，时间上没什么变化，前者为4.7s，后者为4.6s，但理论上内存大一点好)

2、优化：
    转换功能接口封装为类，把write、seek等和内存有关的操作放到类外部实现，
    再传递到该类中，该类没有内存管理更好一些。

todo
    重复读文件，如何做？
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern "C" {
#include "libavcodec/avcodec.h"
#include "libavformat/avformat.h"
#include "libswscale/swscale.h"
}

#include "file_utils.h"

#ifndef min
#define min(a,b) ((a) > (b) ? (b) : (a))
#endif

#define _LL_DEBUG_

// low level debug
#ifdef _LL_DEBUG_
    #define debug(fmt, ...) printf(fmt, ##__VA_ARGS__)
    #define LL_DEBUG(fmt, ...) printf("[DEBUG %s().%d @ %s]: " fmt, \
    __func__, __LINE__, P_SRC, ##__VA_ARGS__)
#else
    #define debug(fmt, ...)
    #define LL_DEBUG(fmt, ...)
#endif

#define DEFAULT_MEM (10*1024*1024)

//参考file协议的内存，使用大小32768，大一点也可以
#define IO_BUFFER_SIZE (32768*1)

typedef struct AVIOBufferContext {
    unsigned char* ptr;
    int pos;
    int totalSize;
    int realSize;
}AVIOBufferContext;

// note 这两个是用户视频数据，
// g_avbuffer_in为已经读取的视频
// g_avbuffer_out是ffmpeg转换后的视频，直接将该内存写入文件即可
AVIOBufferContext g_avbuffer_in;
AVIOBufferContext g_avbuffer_out;

// note这两个是FFMPEG内部使用的IO内存，与AVIOBufferContext的ptr不同
// 在测试时，发现直接定义为数组，会有错误，故使用malloc
static char *g_ptr_in = NULL;
static char *g_ptr_out = NULL;

// 每次read_frame时，就会调用到这个函数，该函数从g_avbuffer_in读数据
```

```
static int my_read(void *opaque, unsigned char *buf, int size)
{
    AVIOBufferContext* op = (AVIOBufferContext*)opaque;
    int len = size;
    if (op->pos + size > op->totalSize)
    {
        len = op->totalSize - op->pos;
    }
    memcpy(buf, op->ptr + op->pos, len);
    if (op->pos + len >= op->realSize)
    op->realSize += len;

    op->pos += len;

    return len;
}

static int my_write(void *opaque, unsigned char *buf, int size)
{
    AVIOBufferContext* op = (AVIOBufferContext*)opaque;
    if (op->pos + size > op->totalSize)
    {
        // 重新申请
        // 根据数值逐步加大
        int newTotalLen = op->totalSize*sizeof(char) * 3 / 2;
        unsigned char* ptr = (unsigned char*)av_realloc(op->ptr, newTotalLen);
        if (ptr == NULL)
        {
            // todo 是否在此处释放内存？
            return -1;
        }
        debug("org ptr: %p new ptr: %p size: %d(%0.fMB) ", op->ptr, ptr,
                    newTotalLen, newTotalLen/1024.0/1024.0);
        op->totalSize = newTotalLen;
        op->ptr = ptr;
        debug(" realloc!!!!!!!!!!!!!!!!!!!!!!\n");
    }
    memcpy(op->ptr + op->pos, buf, size);

    if (op->pos + size >= op->realSize)
        op->realSize += size;

    //static int cnt = 1;
    //debug("%d write %p %p pos: %d len: %d\n", cnt++, op->ptr, buf, op->pos, size);

    op->pos += size;

    return 0;
}

static int64_t my_seek(void *opaque, int64_t offset, int whence)
{
    AVIOBufferContext* op = (AVIOBufferContext*)opaque;
    int64_t new_pos = 0; // 可以为负数
    int64_t fake_pos = 0;

    switch (whence)
    {
        case SEEK_SET:
            new_pos = offset;
            break;
        case SEEK_CUR:
            new_pos = op->pos + offset;
            break;
        case SEEK_END: // 此处可能有问题
            new_pos = op->totalSize + offset;
            break;
        default:
            return -1;
    }

```

```
    fake_pos = min(new_pos, op->totalSize);
    if (fake_pos != op->pos)
    {
        op->pos = fake_pos;
    }
    //debug("seek pos: %d(%d)\n", offset, op->pos);
    return new_pos;
}

int remuxer_mem_read(int argc, char* argv[])
{
    //输入对应一个AVFormatContext，输出对应一个AVFormatContext
    AVFormatContext *ifmt_ctx = NULL, *ofmt_ctx = NULL;
    AVIOContext *avio_in = NULL, *avio_out = NULL;
    const char *in_filename = NULL, *out_filename = NULL;
    AVPacket pkt;

    int ret = 0;

    if (argc < 3)
    {
        printf("usage: %s [input file] [output file]\n", argv[0]);
        printf("eg %s foo.avi bar.ts\n", argv[0]);
        return -1;
    }

    in_filename  = argv[1];
    out_filename = argv[2];

    memset(&g_avbuffer_in, '\0', sizeof(AVIOBufferContext));
    memset(&g_avbuffer_out, '\0', sizeof(AVIOBufferContext));

    read_file(in_filename, (char**)&g_avbuffer_in.ptr, &g_avbuffer_in.totalSize);

    // 分配输出视频数据空间
    g_avbuffer_out.ptr = (unsigned char*)av_realloc(NULL, DEFAULT_MEM*sizeof(char));  // new
    if (g_avbuffer_out.ptr == NULL)
    {
        debug("alloc output mem failed.\n");
        return -1;
    }
    g_avbuffer_out.totalSize = DEFAULT_MEM;
    memset(g_avbuffer_out.ptr, '\0', g_avbuffer_out.totalSize);

    g_ptr_in = (char*)malloc(IO_BUFFER_SIZE*sizeof(char));
    g_ptr_out = (char*)malloc(IO_BUFFER_SIZE*sizeof(char));

    // 初始化
    av_register_all();

    // 输出相关
    // note 要指定IO内存，还在指定自定义的操作函数，这里有write和seek
    avio_out =avio_alloc_context((unsigned char *)g_ptr_out, IO_BUFFER_SIZE, 1,
                &g_avbuffer_out, NULL, my_write, my_seek);
    if (!avio_out)
    {
        printf( "avio_alloc_context failed\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    // 分配AVFormatContext
    // 为方便起见，使用out_filename来根据输出文件扩展名来判断格式
    // 如果要使用如“avi”、“mp4”等指定，赋值给第3个参数即可
    // 注意该函数会分配AVOutputFormat
    avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);
    if (!ofmt_ctx)
    {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
```

```
    ofmt_ctx->pb=avio_out; // 赋值自定义的IO结构体
    ofmt_ctx->flags=AVFMT_FLAG_CUSTOM_IO; // 指定为自定义

    debug("guess format: %s(%s) flag: %d\n", ofmt_ctx->oformat->name,
            ofmt_ctx->oformat->long_name, ofmt_ctx->oformat->flags);


    //  输入相关
    // 分配自定义的AVIOContext 要区别于输出的buffer
    // 由于数据已经在内存中，所以指定read即可，不用write和seek
    avio_in =avio_alloc_context((unsigned char *)g_ptr_in, IO_BUFFER_SIZE, 0,
                    &g_avbuffer_in, my_read, NULL, NULL);
    if (!avio_in)
    {
        printf( "avio_alloc_context for input failed\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    // 分配输入的AVFormatContext
    ifmt_ctx=avformat_alloc_context();
    if (!ifmt_ctx)
    {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    ifmt_ctx->pb=avio_in; // 赋值自定义的IO结构体
    ifmt_ctx->flags=AVFMT_FLAG_CUSTOM_IO; // 指定为自定义

    // 注:第二个参数本来是文件名，但基于内存，不再有意义，随便用字符串
    if ((ret = avformat_open_input(&ifmt_ctx, "wtf", NULL, NULL)) < 0)
    {
        printf("Cannot open input file\n");
        return ret;
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx, NULL)) < 0)
    {
        printf("Cannot find stream information\n");
        return ret;
    }

    // 复制所有的stream
    for (int i = 0; i < (int)(ifmt_ctx->nb_streams); i++)
    {
        //根据输入流创建输出流
        AVStream *in_stream = ifmt_ctx->streams[i];
        AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
        if (!out_stream)
        {
            printf( "Failed allocating output stream\n");
            ret = AVERROR_UNKNOWN;
            goto end;
        }
        //复制AVCodecContext的设置
        ret = avcodec_copy_context(out_stream->codec, in_stream->codec);
        if (ret < 0)
        {
            printf( "Failed to copy context from input to output stream codec context\n");
            goto end;
        }
        out_stream->codec->codec_tag = 0;
        if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
            out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
    }
    //输出一下格式------------------
    printf("output format:\n");
    av_dump_format(ofmt_ctx, 0, out_filename, 1);

    // 写文件头
    ret = avformat_write_header(ofmt_ctx, NULL);
    if (ret < 0)
```

```
    {
        printf( "Error occurred when opening output file\n");
        goto end;
    }

    // 帧
    while (1)
    {
        AVStream *in_stream, *out_stream;
        //获取一个AVPacket
        ret = av_read_frame(ifmt_ctx, &pkt);
        if (ret < 0)
        {
            printf("av_read_frame failed or end of stream.\n");
            break;
        }
        in_stream  = ifmt_ctx->streams[pkt.stream_index];
        out_stream = ofmt_ctx->streams[pkt.stream_index];

        //转换PTS/DTS
        pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base,
            out_stream->time_base, (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
        pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base,
            out_stream->time_base, (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
        pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
        pkt.pos = -1;

        // 写入一帧
        ret = av_interleaved_write_frame(ofmt_ctx, &pkt);
        if (ret < 0) {
            printf( "Error muxing packet\n");
            break;
        }
        av_free_packet(&pkt);
    }

    //写文件尾(Write file trailer)
    printf("--------write trailer------------\n");
    av_write_trailer(ofmt_ctx);

    // 把输出的视频写到文件中
    printf("write to file: %s %p %d\n", out_filename, g_avbuffer_out.ptr, g_avbuffer_out.realSize);
    write_file(out_filename, (char*)g_avbuffer_out.ptr, g_avbuffer_out.realSize, 1);

end:
    if (avio_in != NULL)  av_freep(avio_in);
    if (avio_out != NULL) av_freep(avio_out);
    //if (g_ptr_in != NULL) free(g_ptr_in);
    if (g_ptr_out != NULL) free(g_ptr_out);

    // 该函数会释放用户自定义的IO buffer, 上面不再释放, 否则会corrupted double-linked list
    avformat_close_input(&ifmt_ctx);
    avformat_free_context(ofmt_ctx);

    if (g_avbuffer_in.ptr != NULL) free(g_avbuffer_in.ptr);
    if (g_avbuffer_out.ptr != NULL) free(g_avbuffer_out.ptr);

    return ret;
}
```

from:http://blog.csdn.net/subfate/article/details/48001433

分类: ffmpeg、ffplay

« 上一篇： FFMPEG内存操作（一） avio_reading.c 回调读取数据到内存解析

» 下一篇： ffmpeg 从内存中读取数据（或将数据输出到内存）

posted on 2017-08-11 15:58 DoubleLi 阅读(1070) 评论(0) 编辑 收藏 举报

刷新评论 刷新页面 返回顶部

登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

编辑推荐：

· 聊聊我在微软外服的工作经历及一些个人见解

· 死磕 NIO — Reactor 模式就一定意味着高性能吗？

· 消息队列那么多，为什么建议深入了解下RabbitMQ？

· 技术管理进阶——管人还是管事？

· 以终为始：如何让你的开发符合预期

最新新闻：

· 何小鹏：争取2024年实现飞行汽车量产 价格100万以内（2021-10-24 23:35）

· 供应链危机提振美国在线二手市场 全年销售额预计超650亿美元（2021-10-24 22:00）

· CityTree：一款利用苔藓和机器学习来捕捉空气污染的设备（2021-10-24 20:53）

· 1024程序员节各家怎么过：送霸王洗发水、集体穿格子衫、盲人按摩（2021-10-24 20:00）

· 新卫星图展示泰国季风洪水所带来的巨大影响（2021-10-24 19:00）

» 更多新闻...