从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

原创 JT同学 于 2019-08-12 16:43:27 发布 4594 收藏 14 版权
分类专栏： 从零开始写一个RTSP服务器 文章标签： rtsp 流媒体 aac

从零开始写一个RTSP服务器系列

★我的开源项目-RtspServer

从零开始写一个RTSP服务器（一）RTSP协议讲解

从零开始写一个RTSP服务器（二）RTSP协议的实现

从零开始写一个RTSP服务器（三）RTP传输H.264

从零开始写一个RTSP服务器（四）一个传输H.264的RTSP服务器

从零开始写一个RTSP服务器（五）RTP传输AAC

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

从零开始写一个RTSP服务器（七）多播传输RTP包

从零开始写一个RTSP服务器（八）一个多播的RTSP服务器

从零开始写一个RTSP服务器（九）一个RTP OVER RTSP/TCP的RTSP服务器

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

文章目录

本文主要是结合

本文主要是结合

从零开始写一个RTSP服务器（一）不一样的RTSP协议讲解

从零开始写一个RTSP服务器（二）RTSP协议的实现

从零开始写一个RTSP服务器（五）RTP传输AAC

这几篇文章的内容总结，然后写出完成一个传输AAC的RTSP服务器，所有的知识点都在这几篇文章中，在看此篇文章前建议先认真看一看前面那几篇文章

本文不会讲解新知识点，如果你仔细看前几篇文章，相信你不需要看本文就能够完成一个传输AAC的RTSP服务器

下面主要是介绍我提供的示例代码的运行流程和其中的一些细节

一、建立套接字

一开始进入main函数后，就监听服务器tcp套接字，绑定端口号，然后开始监听

然后再分别建立用于RTP和RTCP的udp套接字，绑定好端口

然后进入循环中开始服务

```
/*
 * 作者：_JT_
 * 博客：https://blog.csdn.net/weixin_42462202
 */

main()
{
    /* 创建服务器tcp套接字，绑定端口，监听 */
    serverSockfd = createTcpSocket();
    bindSocketAddr(serverSockfd, "0.0.0.0", SERVER_PORT);
    listen(serverSockfd, 10);

    /* 建立用于RTP和RTCP的udp套接字，绑定好端口 */
    serverRtpSockfd = createUdpSocket();
    serverRtcpSockfd = createUdpSocket();
    bindSocketAddr(serverRtpSockfd, "0.0.0.0", SERVER_RTP_PORT);
    bindSocketAddr(serverRtcpSockfd, "0.0.0.0", SERVER_RTCP_PORT);

    while(1)
    {
        ...
    }
}
```

二、接收客户端连接

在while循环中接收客户端，然后调用doClient服务

```
main()
{
    ...
    while(1)
    {
        clientSockfd = acceptClient(serverSockfd, clientIp, &clientPort);
        doClient(clientSockfd, clientIp, clientPort, serverRtpSockfd, serverRtcpSockfd);
    }
}
```

上面其实就是一个TCP服务器的基本步骤，没有什么特别的

下面来看一看doClient函数

三、解析命令

doClient就是一个while循环（这是一个同时只能服务一个客户的服务器），不断地接收命令解析命令，然后调用相应地操作

```
/*
 * 作者：_JT_
 * 博客：https://blog.csdn.net/weixin_42462202
 */

doClient()
{
    while(1)
    {
        recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
        ...
        sscanf(line, "%s %s %s\r\n", method, url, version);
        ...
        sscanf(line, "CSeq: %d\r\n", &cseq)
        ...
    }
}
```

四、处理请求

在解析完客户端命令后，会调用相应的请求，处理完之后讲接收打印到`sBuf`中，然后发送给客户端

```
/*
 * 作者：_JT_
 * 博客：https://blog.csdn.net/weixin_42462202
 */

doClient()
{
    while(1)
    {
        ...
         /* 处理请求 */
```

```
        if(!strcmp(method, "OPTIONS"))
            handleCmd_OPTIONS(sBuf, cseq);
        else if(!strcmp(method, "DESCRIBE"))
            handleCmd_DESCRIBE(sBuf, cseq, url);
        else if(!strcmp(method, "SETUP"))
            handleCmd_SETUP(sBuf, cseq, clientRtpPort);
        else if(!strcmp(method, "PLAY"))
            handleCmd_PLAY(sBuf, cseq);

        /* 放回结果 */
        send(clientSockfd, sBuf, strlen(sBuf), 0);
    }
}
```

下面来看看各个请求的行动

4.1 OPTIONS

返回可用方法

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

static int handleCmd_OPTIONS(char* result, int cseq)
{
    sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
                    "\r\n",
                    cseq);

    return 0;
}
```

4.2 DESCRIBE

返回sdp文件信息，注意这个示例的sdp文件和从零开始写一个RTSP服务器（四）一个传输H.264的RTSP服务器中的sdp文件是不一样的，这是很重要的

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

static int handleCmd_DESCRIBE(char* result, int cseq, char* url)
{
    char sdp[500];
    char localIp[100];

    sscanf(url, "rtsp://%[^:]:", localIp);

    sprintf(sdp, "v=0\r\n"
                 "o=- 9%ld 1 IN IP4 %s\r\n"
                 "t=0 0\r\n"
```

4.3 SETUP

SETUP过程发送服务端RTP端口和RTCP端口

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

static int handleCmd_SETUP(char* result, int cseq, int clientRtpPort)
{
    sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Transport: RTP/AVP;unicast;client_port=%d-%d;server_port=%d-%d\r\n"
                    "Session: 66334873\r\n"
                    "\r\n",
                    cseq,
                    clientRtpPort,
                    clientRtpPort+1,
                    SERVER_RTP_PORT,
                    SERVER_RTCP_PORT);

    return 0;
}
```

4.4 PLAY

PLAY操作回复后，会开始发送RTP包

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

static int handleCmd_PLAY(char* result, int cseq)
{
    sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Range: npt=0.000-\r\n"
                    "Session: 66334873; timeout=60\r\n\r\n",
                    cseq);

    return 0;
}
```

五、AAC RTP打包发送

先读取ADTS头，得到一帧的大小，然后再读取AAC Data，再通过RTP打包传输

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

doClient()
{
    while(1)
    {
        ...

         send(clientSockfd, sBuf, strlen(sBuf), 0);

        if(!strcmp(method, "PLAY"))
        {
            while(1)
            {
                /* 读取ADTS头部 */
                read(fd, frame, 7);

                /* 解析头部 */
                parseAdtsHeader(frame, &adtsHeader);

                /* 读取一帧 */
                read(fd, frame, adtsHeader.aacFrameLength-7);

                /* RTP打包发送 */
                rtpSendAACFrame(localRtpSockfd, clientIP, clientRtpPort,
                                rtpPacket, frame, adtsHeader.aacFrameLength-7);

            }
        }

    }
}
```

看一看AAC的RTP打包发送过程

先填充RTP载荷前4个字节，然后发送RTP包

发送后序列号增加，时间戳增加

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

static int rtpSendAACFrame(int socket, const char* ip, int16_t port,
                            struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
{
    /* 填充前4个字节 */
    rtpPacket->payload[0] = 0x00;
    rtpPacket->payload[1] = 0x10;
    rtpPacket->payload[2] = (frameSize & 0x1FE0) >> 5; //高8位
    rtpPacket->payload[3] = (frameSize & 0x1F) << 3; //低5位
```

```
    /* 发送RTP包 */
```

六、源码

总共由三个文件`aac_rtsp_server.c`、`rtp.h`、`rtp.c`

aac_rtsp_server.c

```c
/*
 * 作者：_JT_
 * 博客：https://blog.csdn.net/weixin_42462202
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "rtp.h"

#define SERVER_PORT     8554
#define SERVER_RTP_PORT  55532
#define SERVER_RTCP_PORT 55533
#define BUF_MAX_SIZE    (1024*1024)
#define AAC_FILE_NAME   "test.aac"

static int createTcpSocket()
{
    int sockfd;
    int on = 1;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
        return -1;

    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));

    return sockfd;
}

static int createUdpSocket()
{
    int sockfd;
    int on = 1;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd < 0)
        return -1;

    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));

    return sockfd;
}

static int bindSocketAddr(int sockfd, const char* ip, int port)
{
    struct sockaddr_in addr;

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = inet_addr(ip);

    if(bind(sockfd, (struct sockaddr *)&addr, sizeof(struct sockaddr)) < 0)
        return -1;

    return 0;
}

struct AdtsHeader
{
    unsigned int syncword;  //12 bit 同步字 '1111 1111 1111'，说明一个ADTS帧的开始
    unsigned int id;        //1 bit MPEG 标示符， 0 for MPEG-4, 1 for MPEG-2
    unsigned int layer;     //2 bit 总是'00'
    unsigned int protectionAbsent;  //1 bit 1表示没有crc，0表示有crc
    unsigned int profile;           //1 bit 表示使用哪个级别的AAC
    unsigned int samplingFreqIndex; //4 bit 表示使用的采样频率
    unsigned int privateBit;        //1 bit
    unsigned int channelCfg; //3 bit 表示声道数
    unsigned int originalCopy;         //1 bit
    unsigned int home;                  //1 bit

    /*下面的为改变的参数即每一帧都不同*/
    unsigned int copyrightIdentificationBit;   //1 bit
    unsigned int copyrightIdentificationStart; //1 bit
    unsigned int aacFrameLength;               //13 bit 一个ADTS帧的长度包括ADTS头和AAC原始流
    unsigned int adtsBufferFullness;           //11 bit 0x7FF 说明是码率可变的码流

    /* number_of_raw_data_blocks_in_frame
     * 表示ADTS帧中有number_of_raw_data_blocks_in_frame + 1个AAC原始帧
     * 所以说number_of_raw_data_blocks_in_frame == 0
     * 表示说ADTS帧中有一个AAC数据块并不是说没有。(一个AAC原始帧包含一段时间内1024个采样及相关数据)
     */
    unsigned int numberOfRawDataBlockInFrame; //2 bit
};

static int parseAdtsHeader(uint8_t* in, struct AdtsHeader* res)
{
    static int frame_number = 0;
    memset(res,0,sizeof(*res));

    if ((in[0] == 0xFF)&&((in[1] & 0xF0) == 0xF0))
    {
        res->id = ((unsigned int) in[1] & 0x08) >> 3;
        res->layer = ((unsigned int) in[1] & 0x06) >> 1;
        res->protectionAbsent = (unsigned int) in[1] & 0x01;
        res->profile = ((unsigned int) in[2] & 0xc0) >> 6;
        res->samplingFreqIndex = ((unsigned int) in[2] & 0x3c) >> 2;
        res->privateBit = ((unsigned int) in[2] & 0x02) >> 1;
        res->channelCfg = ((((unsigned int) in[2] & 0x01) << 2) | (((unsigned int) in[3] & 0xc0) >> 6));
        res->originalCopy = ((unsigned int) in[3] & 0x20) >> 5;
        res->home = ((unsigned int) in[3] & 0x10) >> 4;
        res->copyrightIdentificationBit = ((unsigned int) in[3] & 0x08) >> 3;
        res->copyrightIdentificationStart = (unsigned int) in[3] & 0x04 >> 2;
        res->aacFrameLength = (((((unsigned int) in[3]) & 0x03) << 11) |
                                (((unsigned int)in[4] & 0xFF) << 3) |
                                    ((unsigned int)in[5] & 0xE0) >> 5) ;
        res->adtsBufferFullness = (((unsigned int) in[5] & 0x1f) << 6 |
                                        ((unsigned int) in[6] & 0xfc) >> 2);
        res->numberOfRawDataBlockInFrame = ((unsigned int) in[6] & 0x03);

        return 0;
    }
    else
    {
        printf("failed to parse adts header\n");
        return -1;
    }
}

static int rtpSendAACFrame(int socket, const char* ip, int16_t port,
                            struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
{
    int ret;

    rtpPacket->payload[0] = 0x00;
    rtpPacket->payload[1] = 0x10;
    rtpPacket->payload[2] = (frameSize & 0x1FE0) >> 5; //高8位
    rtpPacket->payload[3] = (frameSize & 0x1F) << 3; //低5位

    memcpy(rtpPacket->payload+4, frame, frameSize);

    ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize+4);
    if(ret < 0)
    {
        printf("failed to send rtp packet\n");
        return -1;
    }

    rtpPacket->rtpHeader.seq++;

    /*
     * 如果采样频率是44100
     * 一般AAC每个1024个采样为一帧
     * 所以一秒就有 44100 / 1024 = 43帧
```

```c
     * 时间增量就是 44100 / 43 = 1025
     * 一帧的时间为 1 / 43 = 23ms
     */
    rtpPacket->rtpHeader.timestamp += 1025;

    return 0;
}

static int acceptClient(int sockfd, char* ip, int* port)
{
    int clientfd;
    socklen_t len = 0;
    struct sockaddr_in addr;

    memset(&addr, 0, sizeof(addr));
    len = sizeof(addr);

    clientfd = accept(sockfd, (struct sockaddr *)&addr, &len);
    if(clientfd < 0)
        return -1;

    strcpy(ip, inet_ntoa(addr.sin_addr));
    *port = ntohs(addr.sin_port);

    return clientfd;
}

static char* getLineFromBuf(char* buf, char* line)
{
    while(*buf != '\n')
    {
        *line = *buf;
        line++;
        buf++;
    }

    *line = '\n';
    ++line;
    *line = '\0';

    ++buf;
    return buf;
}

static int handleCmd_OPTIONS(char* result, int cseq)
{
    sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
                    "\r\n",
                    cseq);

    return 0;
}

static int handleCmd_DESCRIBE(char* result, int cseq, char* url)
{
    char sdp[500];
    char localIp[100];

    sscanf(url, "rtsp://%[^:]:", localIp);

    sprintf(sdp, "v=0\r\n"
                 "o=- 9%ld 1 IN IP4 %s\r\n"
                 "t=0 0\r\n"
                 "a=control:*\r\n"
                 "m=audio 0 RTP/AVP 97\r\n"
                 "a=rtpmap:97 mpeg4-generic/44100/2\r\n"
                 "a=fmtp:97 SizeLength=13;\r\n"
                 "a=control:track0\r\n",
                 time(NULL), localIp);

    sprintf(result, "RTSP/1.0 200 OK\r\nCSeq: %d\r\n"
                    "Content-Base: %s\r\n"
                    "Content-type: application/sdp\r\n"
                    "Content-length: %d\r\n\r\n"
                    "%s",
                    cseq,
                    url,
                    strlen(sdp),
                    sdp);

    return 0;
}

static int handleCmd_SETUP(char* result, int cseq, int clientRtpPort)
{
    sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Transport: RTP/AVP;unicast;client_port=%d-%d;server_port=%d-%d\r\n"
                    "Session: 66334873\r\n"
                    "\r\n",
                    cseq,
                    clientRtpPort,
                    clientRtpPort+1,
                    SERVER_RTP_PORT,
                    SERVER_RTCP_PORT
                    );

    return 0;
}

static int handleCmd_PLAY(char* result, int cseq)
{
    sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Range: npt=0.000-\r\n"
                    "Session: 66334873; timeout=60\r\n\r\n",
                    cseq);

    return 0;
}

static void doClient(int clientSockfd, const char* clientIP, int clientPort,
                        int serverRtpSockfd, int serverRtcppSockfd)
{
    char method[40];
    char url[100];
    char version[40];
    int cseq;
    int clientRtpPort, clientRtcpPort;
    char *bufPtr;
    char* rBuf = malloc(BUF_MAX_SIZE);
    char* sBuf = malloc(BUF_MAX_SIZE);
    char line[400];

    while(1)
    {
        int recvLen;

        recvLen = recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
        if(recvLen <= 0)
            goto out;

        rBuf[recvLen] = '\0';
        printf("---------------C->S--------------\n");
        printf("%s", rBuf);

        /* 解析方法 */
        bufPtr = getLineFromBuf(rBuf, line);
        if(sscanf(line, "%s %s %s\r\n", method, url, version) != 3)
        {
            printf("parse err\n");
            goto out;
        }

        /* 解析序列号 */
        bufPtr = getLineFromBuf(bufPtr, line);
        if(sscanf(line, "CSeq: %d\r\n", &cseq) != 1)
        {
            printf("parse err\n");
            goto out;
        }

        /* 如果是SETUP，那么就再解析client_port */
        if(!strcmp(method, "SETUP"))
        {
            while(1)
            {
                bufPtr = getLineFromBuf(bufPtr, line);
                if(!strncmp(line, "Transport:", strlen("Transport:")))
                {
                    sscanf(line, "Transport: RTP/AVP;unicast;client_port=%d-%d\r\n",
                                    &clientRtpPort, &clientRtcpPort);
```

```c
                    break;
                }
            }
        }

        if(!strcmp(method, "OPTIONS"))
        {
            if(handleCmd_OPTIONS(sBuf, cseq))
            {
                printf("failed to handle options\n");
                goto out;
            }
        }
        else if(!strcmp(method, "DESCRIBE"))
        {
            if(handleCmd_DESCRIBE(sBuf, cseq, url))
            {
                printf("failed to handle describe\n");
                goto out;
            }
        }
        else if(!strcmp(method, "SETUP"))
        {
            if(handleCmd_SETUP(sBuf, cseq, clientRtpPort))
            {
                printf("failed to handle setup\n");
                goto out;
            }
        }
        else if(!strcmp(method, "PLAY"))
        {
            if(handleCmd_PLAY(sBuf, cseq))
            {
                printf("failed to handle play\n");
                goto out;
            }
        }
        else
        {
            goto out;
        }

        printf("---------------S->C--------------\n");
        printf("%s", sBuf);
        send(clientSockfd, sBuf, strlen(sBuf), 0);

        if(!strcmp(method, "PLAY"))
        {
            struct AdtsHeader adtsHeader;
            struct RtpPacket* rtpPacket;
            uint8_t* frame;
            int ret;

            int fd = open(AAC_FILE_NAME, O_RDONLY);
            if(fd < 0)
            {
                printf("failed to open %s\n", AAC_FILE_NAME);
                goto out;
            }

            frame = (uint8_t*)malloc(5000);
            rtpPacket = malloc(5000);

            rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_AAC, 1, 0, 0, 0x32411);

            while(1)
            {
                ret = read(fd, frame, 7);
                if(ret <= 0)
                {
                    break;
                }

                if(parseAdtsHeader(frame, &adtsHeader) < 0)
                {
                    printf("parse err\n");
                    break;
                }

                ret = read(fd, frame, adtsHeader.aacFrameLength-7);
                if(ret < 0)
                {
                    printf("read err\n");
                    break;
                }

                rtpSendAACFrame(serverRtpSockfd, clientIP, clientRtpPort,
                                rtpPacket, frame, adtsHeader.aacFrameLength-7);

                usleep(23000);
            }

            free(frame);
            free(rtpPacket);
        }
    }
out:
    close(clientSockfd);
    free(rBuf);
    free(sBuf);
}

int main(int argc, char* argv[])
{
    int serverSockfd;
    int serverRtpSockfd, serverRtcpSockfd;
    int ret;

    serverSockfd = createTcpSocket();
    if(serverSockfd < 0)
    {
        printf("failed to create tcp socket\n");
        return -1;
    }

    ret = bindSocketAddr(serverSockfd, "0.0.0.0", SERVER_PORT);
    if(ret < 0)
    {
        printf("failed to bind addr\n");
        return -1;
    }

    ret = listen(serverSockfd, 10);
    if(ret < 0)
    {
        printf("failed to listen\n");
        return -1;
    }

    serverRtpSockfd = createUdpSocket();
    serverRtcpSockfd = createUdpSocket();
    if(serverRtpSockfd < 0 || serverRtcpSockfd < 0)
    {
        printf("failed to create udp socket\n");
        return -1;
    }

    if(bindSocketAddr(serverRtpSockfd, "0.0.0.0", SERVER_RTP_PORT) < 0 ||
        bindSocketAddr(serverRtcpSockfd, "0.0.0.0", SERVER_RTCP_PORT) < 0)
    {
        printf("failed to bind addr\n");
        return -1;
    }

    printf("rtsp://127.0.0.1:%d\n", SERVER_PORT);

    while(1)
    {
        int clientSockfd;
        char clientIp[40];
        int clientPort;

        clientSockfd = acceptClient(serverSockfd, clientIp, &clientPort);
        if(clientSockfd < 0)
        {
            printf("failed to accept client\n");
            return -1;
        }

        printf("accept client;client ip:%s,client port:%d\n", clientIp, clientPort);

        doClient(clientSockfd, clientIp, clientPort, serverRtpSockfd, serverRtcpSockfd);
    }

```

```
    return 0;
}
```

rtp.h

```c
/*
 * 作者：_JT_
 * 博客：https://blog.csdn.net/weixin_42462202
 */

#ifndef _RTP_H_
#define _RTP_H_
#include <stdint.h>

#define RTP_VESION              2

#define RTP_PAYLOAD_TYPE_H264   96
#define RTP_PAYLOAD_TYPE_AAC    97

#define RTP_HEADER_SIZE         12
```

rtp.c

```c
/*
 * 作者：_JT_
 * 博客：https://blog.csdn.net/weixin_42462202
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "rtp.h"

void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrcLen, uint8_t extension,
                    uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
                   uint16_t seq, uint32_t timestamp, uint32_t ssrc)
{
    rtpPacket->rtpHeader.csrcLen = csrcLen;
    rtpPacket->rtpHeader.extension = extension;
    rtpPacket->rtpHeader.padding = padding;
    rtpPacket->rtpHeader.version = version;
    rtpPacket->rtpHeader.payloadType =  payloadType;
    rtpPacket->rtpHeader.marker = marker;
    rtpPacket->rtpHeader.seq = seq;
    rtpPacket->rtpHeader.timestamp = timestamp;
    rtpPacket->rtpHeader.ssrc = ssrc;
}

int rtpSendPacket(int socket, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize)
{
    struct sockaddr_in addr;
    int ret;

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = inet_addr(ip);

    rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
    rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
    rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);

    ret = sendto(socket, (void*)rtpPacket, dataSize+RTP_HEADER_SIZE, 0,
                    (struct sockaddr*)&addr, sizeof(addr));

    rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
    rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
    rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);

    return ret;
}
```

七、测试

将`aac_rtsp_server.c`、`rtp.h`、`rtp.c`这三个文件保存下来

编译运行，默认会打开`test.aac`的音频文件，如果你没有音频源的话，可以从RtspServer的example目录下获取

```
# gcc aac_rtsp_server.c rtp.c
# ./a.out
```

运行之后会打印一个url

```
rtsp://127.0.0.1:8554
```

再vlc打开即可听到音频