JTing学 于 2019-08-13 10:57:29 发布 4859 收藏 16
分类专栏：从零开始写一个RTSP服务器 文章标签：rtsp 流媒体 H.264 多播

从零开始写一个RTS... 专栏收录该内容 187 订阅 10 篇文章 订阅专栏

从零开始写一个RTSP服务器系列

从零开始写一个RTSP服务器（八）一个多播的RTSP服务器

文章目录

本文目的：实现一个多播传输H.264的RTSP服务器

一、多播的RTSP交互过程

我们在前面文章从零开始写一个RTSP服务器（二）RTSP协议的实现中实现的RTSP交互过程是RTSP单播的情况，多播的实现过程与单播还是有所区别的，多播并不需要为每一个RTP和RTCP建立新的UDP套接字，只需要持续向一个多播地址推送RTP包就行，下面来看一个RTSP多播交互的示例

OPTIONS

C–>S

```
OPTIONS rtsp://127.0.0.1:8554/live RTSP/1.0\r\n
CSeq: 2\r\n
\r\n
```

S–>C

```
RTSP/1.0 200 OK\r\n
CSeq: 2\r\n
Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY\r\n
\r\n
```

DESCRIBE

C–>S

```
DESCRIBE rtsp://127.0.0.1:8554/live RTSP/1.0\r\n
CSeq: 3\r\n
Accept: application/sdp\r\n
\r\n
```

S—>C

```
RTSP/1.0 200 OK\r\n
CSeq: 3\r\n
Content-length: 225\r\n
Content-type: application/sdp\r\n
\r\n
v=0
o=- 91565615172 1 IN IP4 127.0.0.1
t=0 0
a=control:*
a=type:broadcast
a=rtcp-unicast: reflection
m=video 39016 RTP/AVP 96
c=IN IP4 232.123.86.248/255
a=rtpmap:96 H264/90000
a=framerate:25
a=control:track0
```

这个sdp描述文件里指明了多播地址和多播端口

这一行表明RTCP反馈采用单播

```
a=rtcp-unicast: reflection
```

在多播的情况下，这表明服务端RTP发送到多播组，RTCP发送到多播组，RTCP接收采用单播

这一行表明的多播目的端口为39016

```
m=video 39016 RTP/AVP 96
```

这一行表明了多播地址为 `c=IN IP4 232.123.86.248/255`

```
c=IN IP4 232.123.86.248/255
```

多播和单播的sdp文件区别主要是多播需要指定好多播地址和多播端口

关于sdp这里就不再详细讲述了，如何有不清楚请看前面的文章

SETUP

C–>S

```
SETUP rtsp://127.0.0.1:8554/live/track0 RTSP/1.0\r\n
CSeq: 4\r\n
Transport: RTP/AVP;multicast;client_port=39016-39017
\r\n
```

S–>C

```
RTSP/1.0 200 OK\r\n
CSeq: 4\r\n
Transport: RTP/AVP;multicast;destination=232.123.86.248;source=192.168.31.115;port=39016-39017;ttl=255
Session: 66334873
\r\n
```

PLAY

C–>S

```
PLAY rtsp://127.0.0.1:8554/live RTSP/1.0\r\n
CSeq: 5\r\n
Session: 66334873
Range: npt=0.000-\r\n
\r\n
```

S–>C

```
RTSP/1.0 200 OK\r\n
CSeq: 5\r\n
Range: npt=0.000-\r\n
Session: 66334873; timeout=60
\r\n
```

二、多播的RTSP服务器实现过程

2.1 创建套接字

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

main()
{
    /* 创建套接字 */
 serverSockfd = createTcpSocket();

    /* 绑定地址和端口 */
    bindSocketAddr(serverSockfd, "0.0.0.0", SERVER_PORT);

    /* 开始监听 */
    listen(serverSockfd, 10);

    ...

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

2.2 创建线程向多播地址推送RTP包

在进入while循环接收客户端前，我们创建一个线程不断地向多播地址发送RTP包

```
main()
{
    ...
    pthread_create(&threadId, NULL, sendRtpPacket, NULL);

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

下面看一看发送函数

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

sendRtpPacket()
{
    ...
    while(1)
    {
        ...

        /* 获取一帧数据 */
        getFrameFromH264File(fd, frame, 500000);

        /* 向多播地址发送RTP包 */
        rtpSendH264Frame(sockfd, MULTICAST_IP, MULTICAST_PORT,
                         rtpPacket, frame+startCode, frameSize);

        ...
    }
}
```

2.2 接收客户端连接

进入while循环后，开始接收客户端，然后处理客户端请求

```
main()
{
    ...

    while(1)
    {
        /* 接收客户端 */
        acceptClient(serverSockfd, clientIp, &clientPort);

        /* 处理客户端 */
        doClient(clientSockfd, clientIp, clientPort);
    }
}
```

下面仔细看一看如何处理客户端请求

2.3 解析命令

先解析请求方法，然后解析序列号，再根据不同地请求做不同的处理，然后再放回结果给客户端

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

doClient()
{
    ...
    while(1)
    {
        /* 接收请求 */
        recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
        ...
        /* 解析请求方法 */
        sscanf(line, "%s %s %s\r\n", method, url, version)
        ...
        /* 解析序列号 */
        sscanf(line, "CSeq: %d\r\n", &cseq);

        /* 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, localIp);
        else if(!strcmp(method, "PLAY"))
            handleCmd_PLAY(sBuf, cseq);

        /* 返回结果给客户端 */
        send(clientSockfd, sBuf, strlen(sBuf), 0);
    }
}
```

2.4 处理请求

OPTIONS

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

DESCRIBE

发送多播的sdp描述文件

```
handleCmd_DESCRIBE()
{
    /* 多播sdp */
    sprintf(sdp, "v=0\r\n"
                 "o=- 9%ld 1 IN IP4 %s\r\n"
                 "t=0 0\r\n"
```

```
                "a=control:*\r\n"
                "a=type:broadcast\r\n"
                "a=rtcp-unicast: reflection\r\n"
                "m=video %d RTP/AVP 96\r\n"
                "c=IN IP4 %s/255\r\n"
                "a=rtpmap:96 H264/90000\r\n"
                "a=control:track0\r\n",
                time(NULL),
                localIp,
                MULTICAST_PORT,
                MULTICAST_IP);

    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);
}
```

SETUP

```
handleCmd_SETUP()
{
   sprintf(result, "RTSP/1.0 200 OK\r\n"
                   "CSeq: %d\r\n"
                   "Transport: RTP/AVP;multicast;destination=%s;"
                   "source=%s;port=%d- %d;ttl=255\r\n"
                   "Session: 66334873\r\n"
                   "\r\n",
                   cseq,
                   MULTICAST_IP,
                   localIp,
                   MULTICAST_PORT,
                   MULTICAST_PORT+1);
}
```

PLAY

```
handleCmd_PLAY()
{
    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);
}
```

在play回复完成之后，客户端就会去多播地址拉取媒体流，然后播放

源码

源码有三个文件：`multicast_rtsp_server`、`rtp.h`、`rtp.c`

multicast_rtsp_server.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 <assert.h>
#include <pthread.h>

#include "rtp.h"

#define H264_FILE_NAME      "test.h264"
#define MULTICAST_IP        "239.255.255.11"
#define SERVER_PORT         8554
#define MULTICAST_PORT      9832
#define BUF_MAX_SIZE        (1024*1024)

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;
}

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 inline int startCode3(char* buf)
{
    if(buf[0] == 0 && buf[1] == 0 && buf[2] == 1)
        return 1;
    else
        return 0;
}

static inline int startCode4(char* buf)
{
    if(buf[0] == 0 && buf[1] == 0 && buf[2] == 0 && buf[3] == 1)
        return 1;
    else
```

```
        return 0;
}

static char* findNextStartCode(char* buf, int len)
{
    int i;

    if(len < 3)
        return NULL;

    for(i = 0; i < len-3; ++i)
    {
        if(startCode3(buf) || startCode4(buf))
            return buf;

        ++buf;
    }

    if(startCode3(buf))
        return buf;

    return NULL;
}

static int getFrameFromH264File(int fd, char* frame, int size)
{
    int rSize, frameSize;
    char* nextStartCode;

    if(fd < 0)
        return fd;

    rSize = read(fd, frame, size);
    if(!startCode3(frame) && !startCode4(frame))
        return -1;

    nextStartCode = findNextStartCode(frame+3, rSize-3);
    if(!nextStartCode)
    {
        lseek(fd, 0, SEEK_SET);
        frameSize = rSize;
    }
    else
    {
        frameSize = (nextStartCode-frame);
        lseek(fd, frameSize-rSize, SEEK_CUR);
    }

    return frameSize;
}

static int rtpSendH264Frame(int socket, const char* ip, int16_t port,
                            struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
{
    uint8_t naluType; // nalu第一个字节
    int sendBytes = 0;
    int ret;

    naluType = frame[0];

    if (frameSize <= RTP_MAX_PKT_SIZE) // nalu长度小于最大包场：单一NALU单元模式
    {
        /*
         *   0 1 2 3 4 5 6 7 8 9
         *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         *  |F|NRI|  Type   | a single NAL unit ... |
         *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         */
        memcpy(rtpPacket->payload, frame, frameSize);
        ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize);
        if(ret < 0)
            return -1;

        rtpPacket->rtpHeader.seq++;
        sendBytes += ret;
        if ((naluType & 0x1F) == 7 || (naluType & 0x1F) == 8) // 如果是SPS、PPS就不需要加时间戳
            goto out;
    }
    else // nalu长度小于最大包场：分片模式
    {
        /*
         *  0                   1                   2
         *  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
         * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         * | FU indicator  |   FU header   |   FU payload   ... |
         * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         */

        /*
         *     FU Indicator
         *    0 1 2 3 4 5 6 7
         *   +-+-+-+-+-+-+-+-+
         *   |F|NRI|  Type   |
         *   +---------------+
         */

        /*
         *     FU Header
         *    0 1 2 3 4 5 6 7
         *   +-+-+-+-+-+-+-+-+
         *   |S|E|R|  Type   |
         *   +---------------+
         */

        int pktNum = frameSize / RTP_MAX_PKT_SIZE;       // 有几个完整的包
        int remainPktSize = frameSize % RTP_MAX_PKT_SIZE; // 剩余不完整包的大小
        int i, pos = 1;

        /* 发送完整的包 */
        for (i = 0; i < pktNum; i++)
        {
            rtpPacket->payload[0] = (naluType & 0x60) | 28;
            rtpPacket->payload[1] = naluType & 0x1F;

            if (i == 0) //第一包数据
                rtpPacket->payload[1] |= 0x80; // start
            else if (remainPktSize == 0 && i == pktNum - 1) //最后一包数据
                rtpPacket->payload[1] |= 0x40; // end

            memcpy(rtpPacket->payload+2, frame+pos, RTP_MAX_PKT_SIZE);
            ret = rtpSendPacket(socket, ip, port, rtpPacket, RTP_MAX_PKT_SIZE+2);
            if(ret < 0)
                return -1;

            rtpPacket->rtpHeader.seq++;
            sendBytes += ret;
            pos += RTP_MAX_PKT_SIZE;
        }

        /* 发送剩余的数据 */
        if (remainPktSize > 0)
        {
            rtpPacket->payload[0] = (naluType & 0x60) | 28;
            rtpPacket->payload[1] = naluType & 0x1F;
            rtpPacket->payload[1] |= 0x40; //end

            memcpy(rtpPacket->payload+2, frame+pos, remainPktSize+2);
            ret = rtpSendPacket(socket, ip, port, rtpPacket, remainPktSize+2);
            if(ret < 0)
                return -1;

            rtpPacket->rtpHeader.seq++;
            sendBytes += ret;
        }
    }

out:

    return sendBytes;
}

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"
                 "a=type:broadcast\r\n"
                 "a=rtcp-unicast: reflection\r\n"
                 "m=video %d RTP/AVP 96\r\n"
                 "c=IN IP4 %s/255\r\n"
                 "a=rtpmap:96 H264/90000\r\n"
                 "a=control:track0\r\n",
                 time(NULL),
                 localIp,
                 MULTICAST_PORT,
                 MULTICAST_IP);

    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, char* localIp)
{
   sprintf(result, "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Transport: RTP/AVP;multicast;destination=%s;source=%s;port=%d-%d;ttl=255\r\n"
                    "Session: 66334873\r\n"
                    "\r\n",
                    cseq,
                    MULTICAST_IP,
                    localIp,
                    MULTICAST_PORT,
                    MULTICAST_PORT+1);

    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)
{
    char method[40];
    char url[100];
    char localIp[40];
    char version[40];
    int cseq;
    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;
        }

        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"))
        {
            sscanf(url, "rtsp://%[^:]:", localIp);
            if(handleCmd_SETUP(sBuf, cseq, localIp))
            {
                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);
    }
out:
    printf("finish\n");
    close(clientSockfd);
    free(rBuf);
    free(sBuf);
```

```
}

void* sendRtpPacket(void* arg)
{
    int fd;
    int frameSize, startCode;
    char* frame = malloc(500000);
    struct RtpPacket* rtpPacket = (struct RtpPacket*)malloc(500000);
    int sockfd = createUdpSocket();
    assert(sockfd > 0);

    fd = open(H264_FILE_NAME, O_RDONLY);
    assert(fd > 0);

    rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_H264, 0,
                          0, 0, 0x88923423);

    while(1)
    {
        frameSize = getFrameFromH264File(fd, frame, 500000);

        if(startCode3(frame))
            startCode = 3;
        else
            startCode = 4;

        frameSize -= startCode;
        rtpSendH264Frame(sockfd, MULTICAST_IP, MULTICAST_PORT,
                         rtpPacket, frame+startCode, frameSize);
        rtpPacket->rtpHeader.timestamp += 90000/25;

        usleep(1000*1000/25);
    }

    free(frame);
    free(rtpPacket);
    close(fd);

    return NULL;
}

int main(int argc, char* argv[])
{
    int serverSockfd;
    int ret;
    pthread_t threadId;

    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;
    }

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

    pthread_create(&threadId, NULL, sendRtpPacket, NULL);

    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);
    }

    return 0;
}
```

rtp.h

```
/*
 * 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

```
/*
 * 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;
}
```

三、测试

将 `multicast_rtsp_server.c`、`rtp.h`、`rtp.c` 保存下来

编译运行，程序默认会打开 `test.h264` 的视频文件，如果你没有视频源的话，可以从RtspServer的example目录下获取

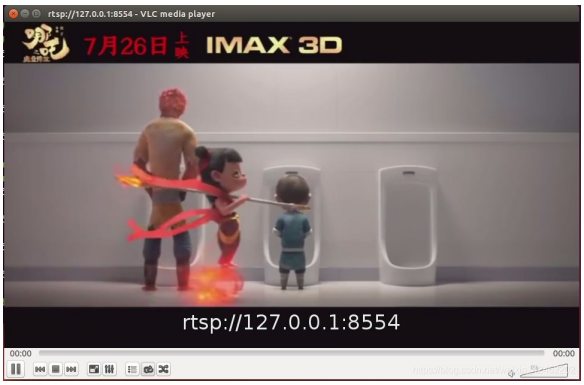
```
# gcc multicast_rtsp_server.c rtp.c -lpthread
# ./a.out
```

运行后会打印一个url

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

在vlc中输入url，就可以播放了

运行效果



JT同学
码龄4年 暂无认证
105 原创 | 2万+ 周排名 | 91万+ 总排名 | 46万+ 访问 | 等级
5440 积分 | 1861 粉丝 | 1016 获赞 | 346 评论 | 3945 收藏
私信 关注

搜博主文章

热门文章
从零开始写一个RTSP服务器（一）RTSP协议讲解 57178
C++ deque的用法与示例 38014
Linux I2C驱动框架（超详细） 35491
我的开源项目-RtspServer 25563
从零开始写一个RTSP服务器（二）RTSP协议的实现 21864

最新评论
我的开源项目-RtspServer
kaifa3k: 我测试 ./h264_rtsp_server test.h264 也会这样的问题？请问解决了吗？ m...
使用mp4v2封装H.264成mp4最简单示例
热心兔斯基: 应该是播放器的原因。换个播放器试试
使用mp4v2封装H.264成mp4最简单示例
ai604233436: h264的标准中，好像默认的设置的90KHZ。我所以我理解timeScale...
使用mp4v2封装H.264成mp4最简单示例
ai604233436: 有遇到，不知道原因呢。
深入学习Linux摄像头（二）v4l2驱动框架
qq_43745897: 请问往缓存区里面采集图像数据的帧率是多少呢？请问如果缓存区里...

您愿意向朋友推荐“博客详情页”吗？
强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章
深入浅出MySQL事务（二）MVCC的实现原理
深入浅出MySQL事务（一）事务隔离
深入浅出MySQL索引（二）InnoDB存储引擎的索引
2020年 4篇 2019年 95篇
2018年 7篇