**H.264远程视频监控系统项目实验：**

一、H.264的认识

1、MP4是视频格式，视频格式分为封装格式和编码格式。H.264是一种编码方式，是高度压缩数字视频编解码器标准。常见的视频封装格式有MP4、FLV、AVI、MOV、3GP等。

编码格式还有MPEG，其中MP4就是用的MPEG，而FLV用的就是H.264。MOV是苹果公司的。

2、首先要弄明白编码的目的。编码是为了将数据进行压缩，这样在传输的过程中就不会使资源被浪费，用一个简单的例子来说明编码的必要性：

当你此刻显示器正在播放一个视频，分辨率是1280\*720，帧率是25，那么一秒所产生正常的数据大小为：

1280\*720(位像素)\*25(张) / 8(1字节8位)(结果:B) / 1024(结果:KB) / 1024 (结果:MB) = 2.75MB

显然一秒这么大的数据你是无法接受的，所以如果不将数据进行压缩，那么只能一首凉凉表达此刻的感受了；

2.1

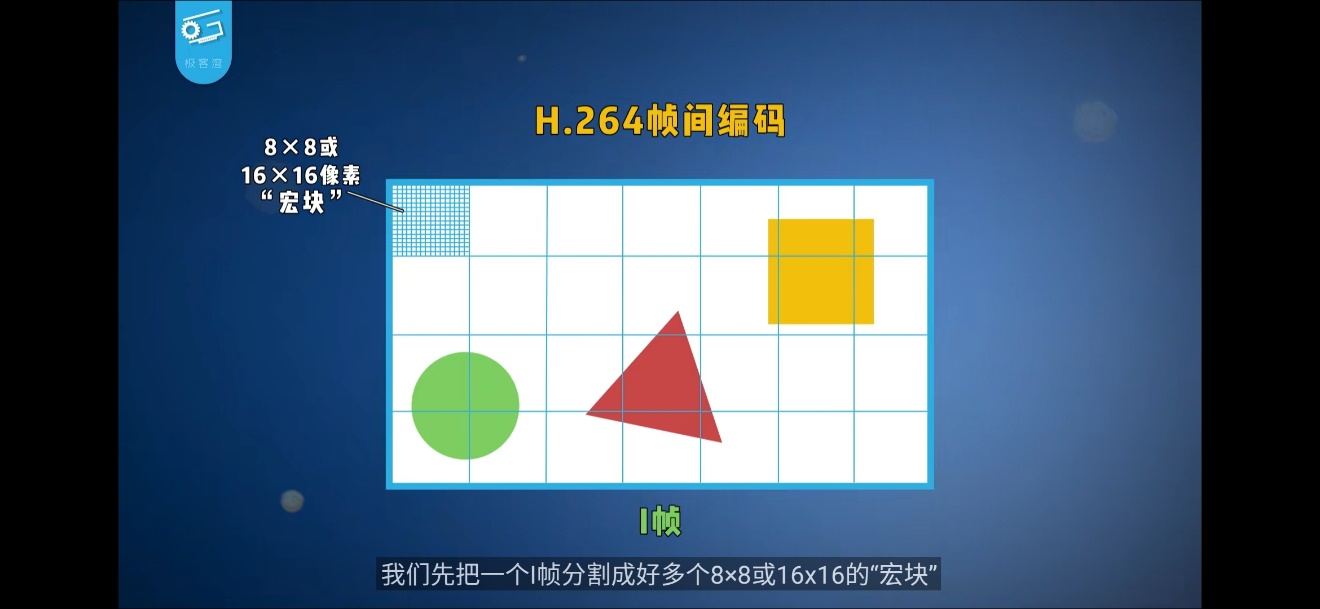
视频压缩是由帧内压缩和帧间压缩两种方式结合的，帧内压缩就是把视频里的每一帧画面都压缩（编码成）成JPEG（一种图片压缩方法，因为人类对亮度敏感，颜色不敏感，所以对亮度保留，颜色进行大幅压缩）这类有损图像，（这里开始是帧间压缩）但是还是压缩不够，因为压缩后只有几帧是原生的JPEG图像，其他的都是算法脑补的，所以只要记住关键的，有变化的图像就可以了。

2.2

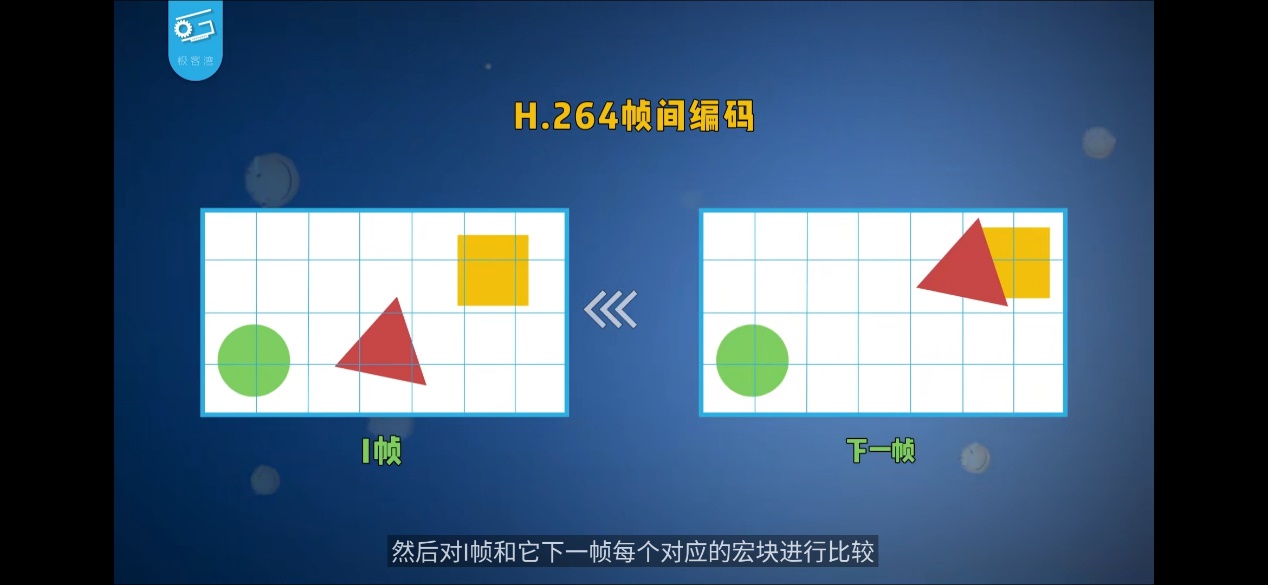
H264结构中，一个视频图像编码后的数据叫做一帧，一帧由一个片（slice）或多个片组成，一个片由一个或多个宏块（MB）组成，一个宏块由16x16的yuv数据组成。宏块作为H264编码的基本单位。

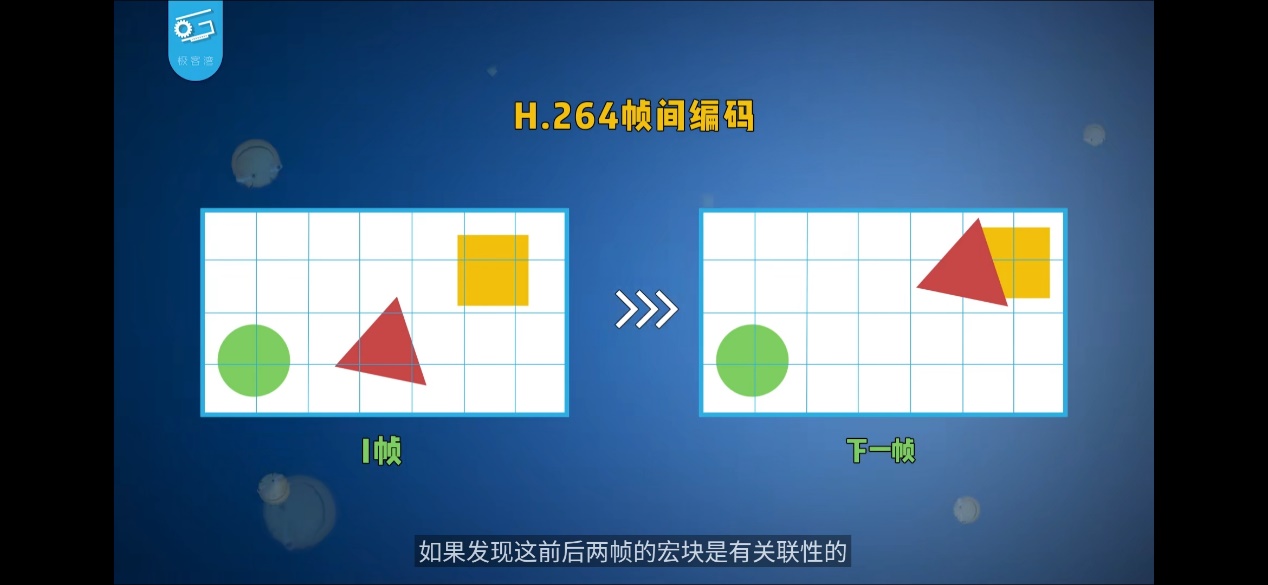
在H264协议内定义了三种帧，分别是I帧、B帧与P帧。I帧就是之前所说的一个完整的图像帧，而B、帧与P帧所对应的就是之前说的不编码全部图像的帧。P帧与B帧的差别就是P帧是参考之前的I帧而生成的，而B帧是参考前后图像帧编码生成的。

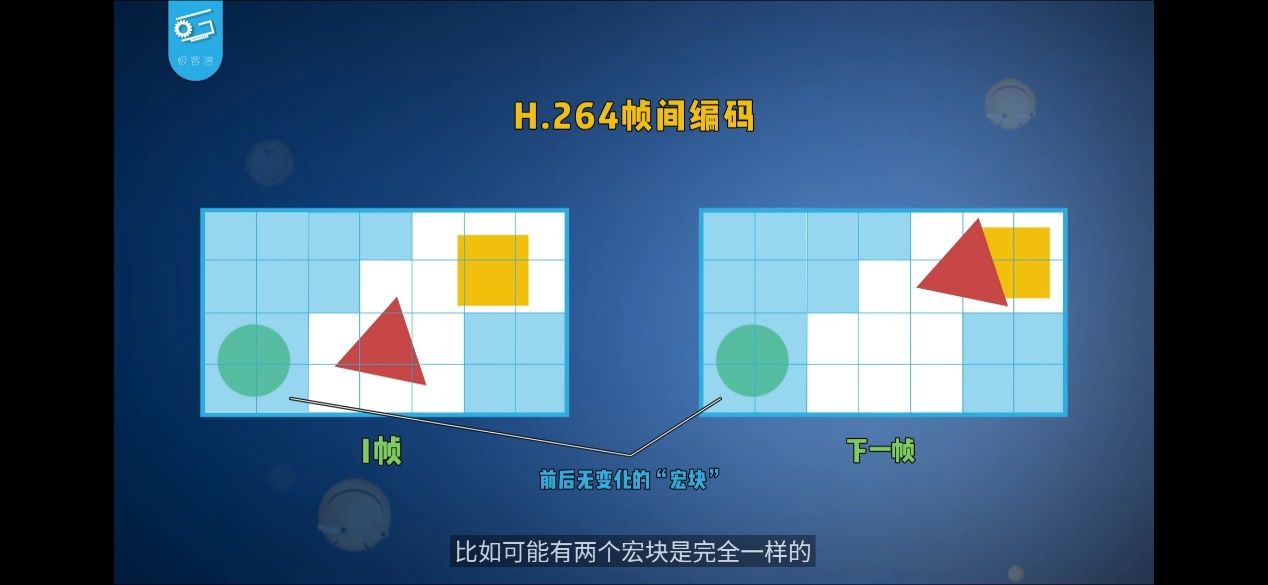
I帧是完整关键帧，

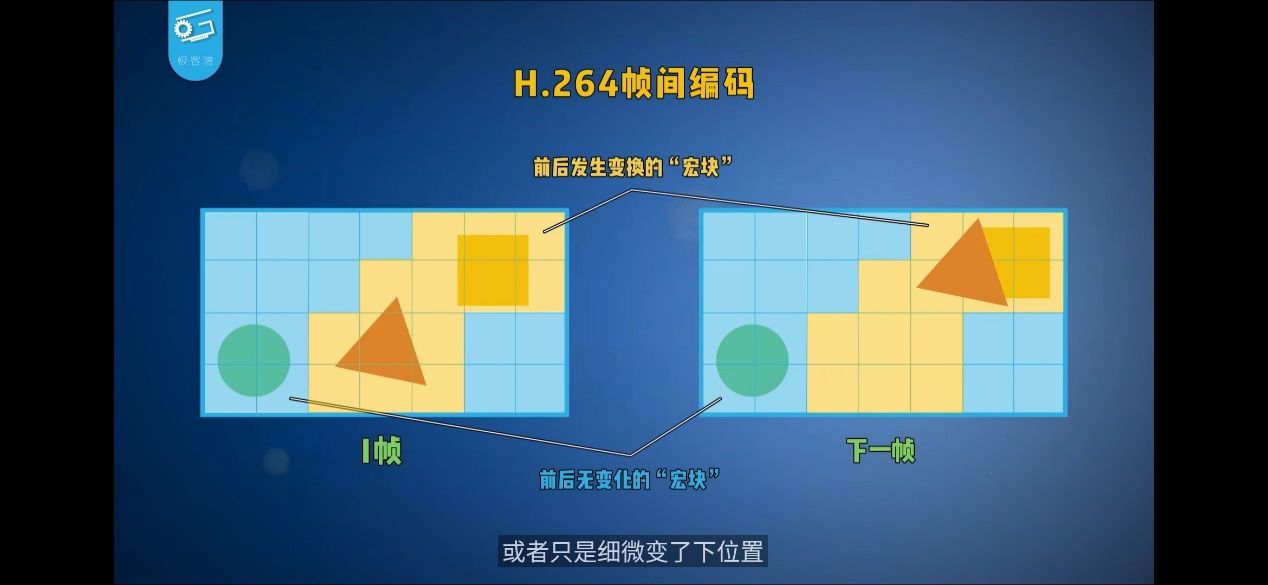


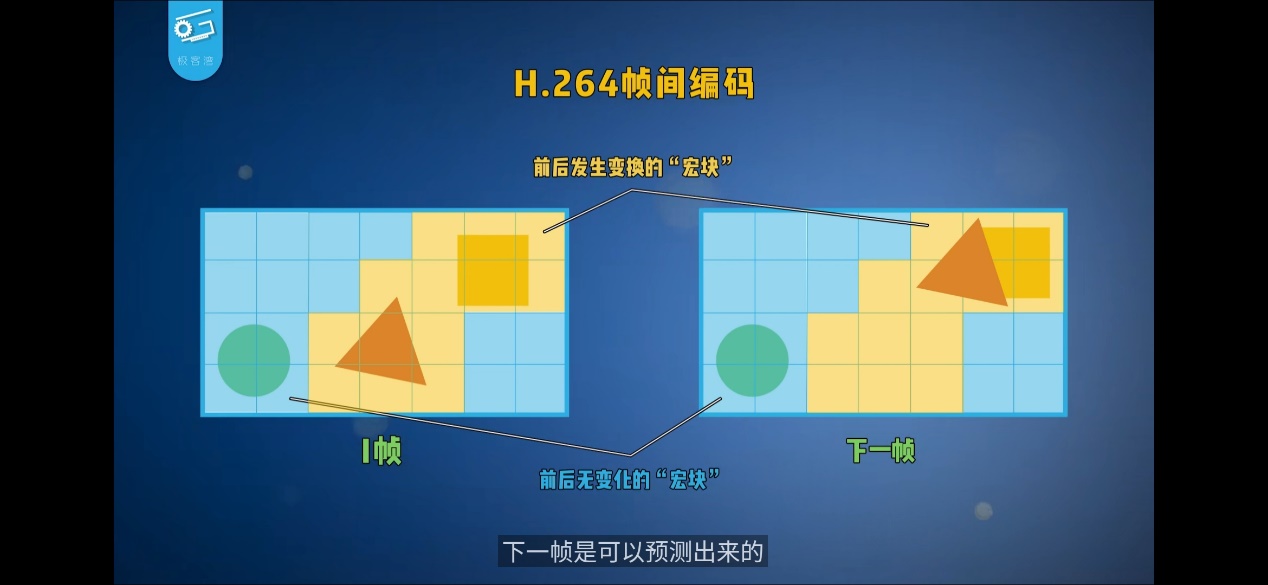
3、H264在视频采集到输出中属于编解码层次的数据，如图1所示，是在采集数据后做编码压缩时通过编码标准编码后所呈现的数据。

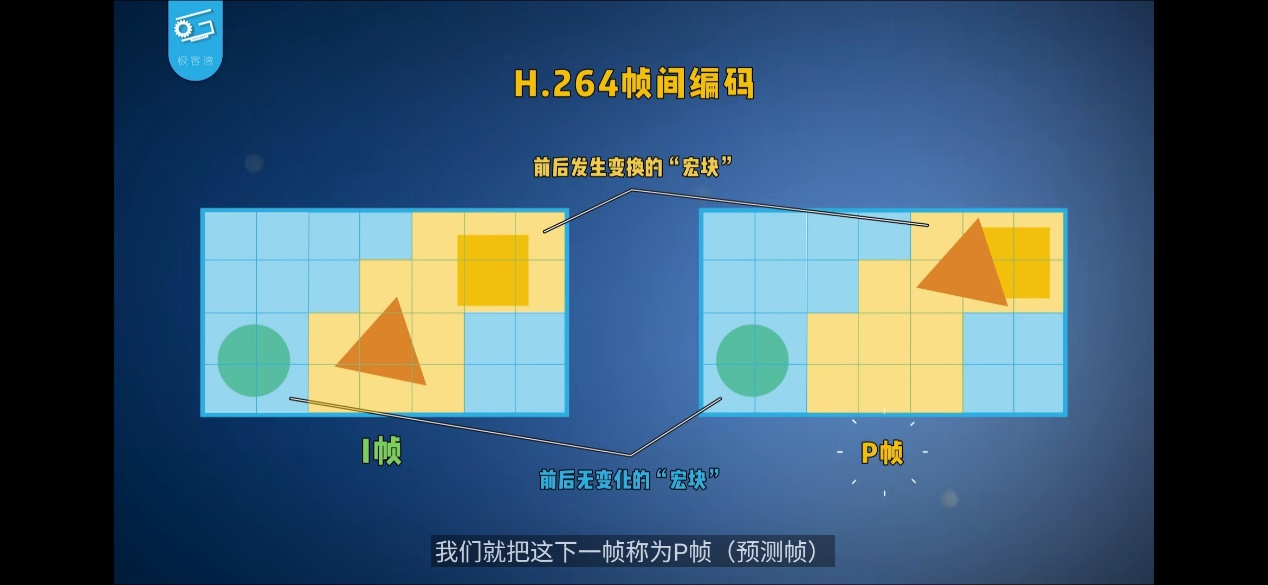


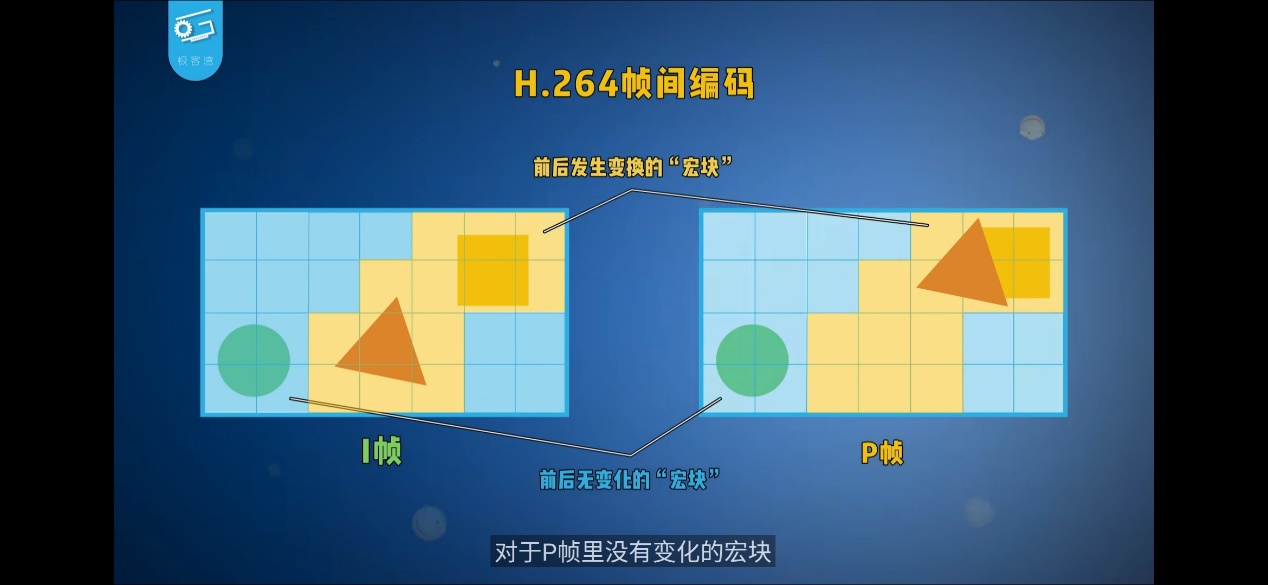




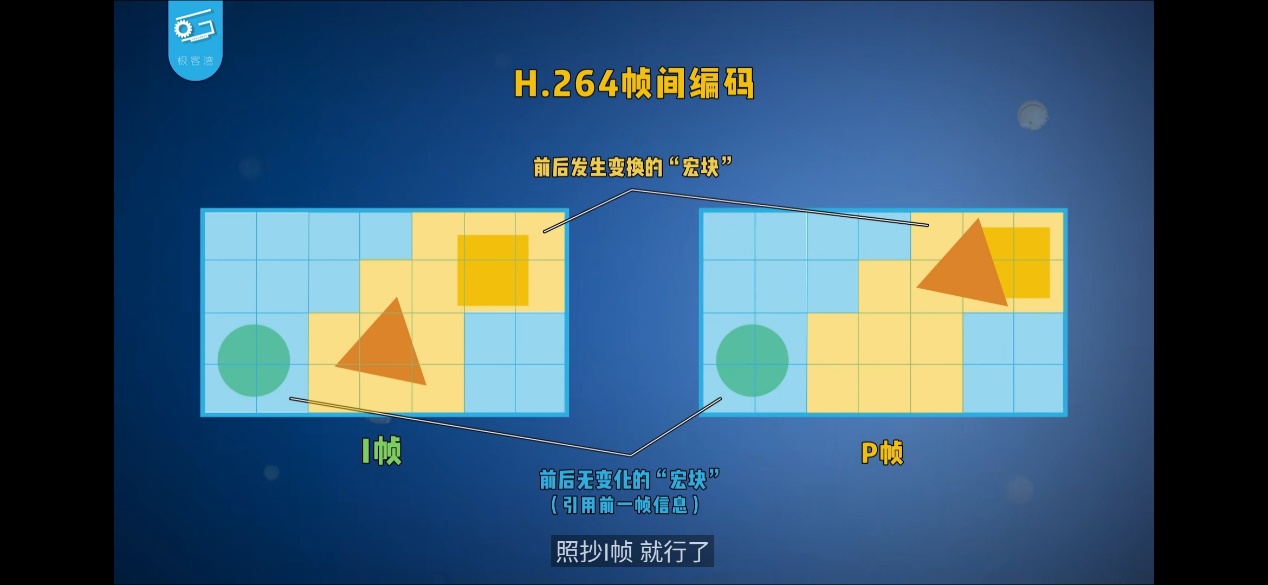








可以直接写上，照抄I帧



而变化的帧，记录下，解码的时候再预测回去，这就是P帧。

B帧是双向预测帧是在I帧和P帧的基础上再进行预测，不仅参考前面的帧，还可以参考后面的帧，所以B帧相比于P帧又节约了一半以上的空间。

**2.3 GOP**

GOP是指每一组IPB帧的序列包含了多少帧，序列就是上面这组变化不大的图像集。

也可以说是I帧后过了多少个帧才会遇到下一个I帧。

相同码率（码率就是指一帧含有的信息量，也叫比特率）下，GOP的值越大，就代表B帧和P帧越多，视频质量越高

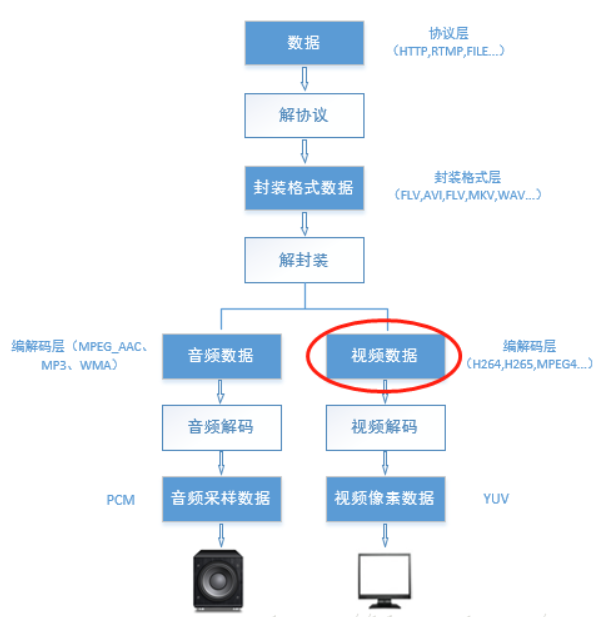
帧速率就是视频是由照片组成，如果8张照片构成的，帧速率就是8帧/秒（fps）

码率有CBR、ABR、VBR

1. CBR：固定码率，稳定，但是浪费空间，因为所有的码率都一样，但是不是所有的都需要。
2. ABR：平均码率，中间选择。
3. VBR：可变码率，灵活，会自动调节，让视频文件更小，画面更清晰。

编码出来的视频格式叫MPEG。简称MPE，MP4就是这个格式。

H.264和FLV后，生成的是F4V，体积小，图像更清晰。



**二、项目需求**

**2.1 产品架构图**

视频采集应用程序

UDP

USB

摄像头驱动

USBUSB

（虚线框内是需要自己完成的）

流程：

1. USB摄像头进行采集数据，由摄像头驱动程序来驱动。
2. 视频采集应用程序通过摄像头驱动从摄像头获取采集到的数据。
3. 采集应用程序通过调用H.264编码库来对采集到的图像进行压缩。
4. 采集应用程序将压缩过的图像通过网络传输（UDP）传给Windows视频显示程序。
5. Windows视频显示程序对图像数据进行解码并显示出来。

**三、系统设计**

1. 根据项目需求分析，该项目需要三个部分的软件开发工作：
2. USB摄像头驱动移植
3. H.264编码库移植
4. 视频采集应用程序开发

2、视频采集应用程序开发流程图

Socket初始化

Bind绑定目标服务器IP地址

初始化H264编码库

把数据包送到Socket进行发送

读取摄像头数据并进行H264编码

读摄像头数据缓冲区检测是否有收到帧数据

初始化摄像头

没有帧时就阻塞

有

* 1. 补充知识----什么是V4L2（video for linux）？

V4L2是V4L的改良版。

V4L2是linux操作系统下用于采集图像的API接口，配合着视频采集应用程序就可以实现图像的采集。

V4L2支持两种方式进行采集：内存映射（用于视频采集）和直接读取（一般用于图片采集）

V4L2在include/linux/video.h中定义了。

**四、源码分析**

1. V4L2编程

1.定义

V4L2(VideoFor Linux Two)是内核提供给应用程序访问视频驱动的统一接口。

2.工作流程：

打开设备－>检查和设置设备属性－>设置帧格式－>设置一种输入输出方法（缓冲区管理）－>循环获取数据－>关闭设备。

3.设备的打开和关闭：

intfd=open(“/dev/video0”,O\_RDWR);//打开设备

close(fd);//关闭设备

**4.查询设备属性：VIDIOC\_QUERYCAP**

相关函数：

int ioctl(int fd, intrequest, struct v4l2\_capability \*argp);

相关结构体：

structv4l2\_capability

{

\_\_u8driver[16]; //驱动名字

\_\_u8card[32]; //设备名字

\_\_u8bus\_info[32]; //设备在系统中的位置

\_\_u32version; //驱动版本号

\_\_u32capabilities; //设备支持的操作

\_\_u32reserved[4]; //保留字段

};

capabilities常用值:

V4L2\_CAP\_VIDEO\_CAPTURE //是否支持图像获取

例：显示设备信息

Struct v4l2\_capability cap;

ioctl(fd,VIDIOC\_QUERYCAP,&cap);

printf(“DriverName:%s/nCardName:%s/nBus info:%s/nDriverVersion:%u.%u.%u/n”,cap.driver,cap.card,cap.bus\_info,(cap.version>>16)&0XFF,(cap.version>>8)&0XFF,cap.version&OXFF);

**5.帧格式：**

VIDIOC\_ENUM\_FMT//显示所有支持的格式

int ioctl(int fd, intrequest, struct v4l2\_fmtdesc \*argp);

struct v4l2\_fmtdesc

{

\_\_u32index; //要查询的格式序号，应用程序设置

enumv4l2\_buf\_type type; //帧类型，应用程序设置

\_\_u32flags; //是否为压缩格式

\_\_u8 description[32]; //格式名称

\_\_u32pixelformat; //格式

\_\_u32reserved[4]; //保留

};

例：显示所有支持的格式

struct v4l2\_fmtdes cfmtdesc;

fmtdesc.index=0;

fmtdesc.type=V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

printf("Supportformat:/n");

while(ioctl(fd,VIDIOC\_ENUM\_FMT,&fmtdesc)!=-1)

{

printf("/t%d.%s/n",fmtdesc.index+1,fmtdesc.description);

fmtdesc.index++;

}

//查看或设置当前格式

VIDIOC\_G\_FMT,VIDIOC\_S\_FMT

//检查是否支持某种格式

VIDIOC\_TRY\_FMT

intioctl(int fd, int request, struct v4l2\_format \*argp);

structv4l2\_format

{

enumv4l2\_buf\_type type;//帧类型，应用程序设置

unionfmt

{

structv4l2\_pix\_format pix;//视频设备使用

structv4l2\_window win;

structv4l2\_vbi\_format vbi;

structv4l2\_sliced\_vbi\_format sliced;

\_\_u8raw\_data[200];

};

};

structv4l2\_pix\_format

{

\_\_u32width; //帧宽，单位像素

\_\_u32height; //帧高，单位像素

\_\_u32pixelformat; //帧格式

enumv4l2\_field field;

\_\_u32bytesperline;

\_\_u32sizeimage;

enumv4l2\_colorspace colorspace;

\_\_u32priv;

};

例：显示当前帧的相关信息

structv4l2\_format fmt;

fmt.type=V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ioctl(fd,VIDIOC\_G\_FMT,&fmt);

printf(“Currentdata formatinformation:/n/twidth:%d/n/theight:%d/n”,fmt.fmt.width,fmt.fmt.height);

struct v4l2\_fmtdescfmtdesc;

fmtdesc.index=0;

fmtdesc.type=V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

while(ioctl(fd,VIDIOC\_ENUM\_FMT,&fmtdesc)!=-1)

{

if(fmtdesc.pixelformat& fmt.fmt.pixelformat)

{

printf(“/tformat:%s/n”,fmtdesc.description);

break;

}

fmtdesc.index++;

}

例：检查是否支持某种帧格式

structv4l2\_format fmt;

fmt.type=V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

fmt.fmt.pix.pixelformat=V4L2\_PIX\_FMT\_RGB32;

if(ioctl(fd,VIDIOC\_TRY\_FMT,&fmt)==-1)

if(errno==EINVAL)

printf(“notsupport format RGB32!/n”);

6.图像的缩放

VIDIOC\_CROPCAP

intioctl(int fd, int request, struct v4l2\_cropcap \*argp);

structv4l2\_cropcap

{

enumv4l2\_buf\_type type;//应用程序设置

structv4l2\_rect bounds;//最大边界

structv4l2\_rect defrect;//默认值

structv4l2\_fract pixelaspect;

};

//设置缩放

VIDIOC\_G\_CROP,VIDIOC\_S\_CROP

intioctl(int fd, int request, struct v4l2\_crop \*argp);

intioctl(int fd, int request, const struct v4l2\_crop \*argp);

structv4l2\_crop

{

enumv4l2\_buf\_type type;//应用程序设置

structv4l2\_rect c;

}

7.申请和管理缓冲区，应用程序和设备有三种交换数据的方法，直接read/write，内存映射(memorymapping)，用户指针。这里只讨论memorymapping.

//向设备申请缓冲区

VIDIOC\_REQBUFS

intioctl(int fd, int request, struct v4l2\_requestbuffers \*argp);

structv4l2\_requestbuffers

{

\_\_u32count; //缓冲区内缓冲帧的数目

enumv4l2\_buf\_type type; //缓冲帧数据格式

enumv4l2\_memory memory; //区别是内存映射还是用户指针方式

\_\_u32reserved[2];

};

enumv4l2\_memoy {V4L2\_MEMORY\_MMAP,V4L2\_MEMORY\_USERPTR};

//count,type,memory都要应用程序设置

例：申请一个拥有四个缓冲帧的缓冲区

structv4l2\_requestbuffers req;

req.count=4;

req.type=V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

req.memory=V4L2\_MEMORY\_MMAP;

ioctl(fd,VIDIOC\_REQBUFS,&req);

获取缓冲帧的地址，长度：

VIDIOC\_QUERYBUF

intioctl(int fd, int request, struct v4l2\_buffer \*argp);

structv4l2\_buffer

{

\_\_u32index; //buffer序号

enumv4l2\_buf\_type type; //buffer类型

\_\_u32byteused; //buffer中已使用的字节数

\_\_u32flags; //区分是MMAP还是USERPTR

enumv4l2\_field field;

structtimeval timestamp;//获取第一个字节时的系统时间

structv4l2\_timecode timecode;

\_\_u32sequence; //队列中的序号

enumv4l2\_memory memory;//IO方式，被应用程序设置

unionm

{

\_\_u32offset;//缓冲帧地址，只对MMAP有效

unsignedlong userptr;

};

\_\_u32length;//缓冲帧长度

\_\_u32input;

\_\_u32reserved;

};

MMAP，定义一个结构体来映射每个缓冲帧。

Structbuffer

{

void\*start;

unsignedint length;

}\*buffers;

#include<sys/mman.h>

void\*mmap(void \*addr, size\_t length, int prot, int flags,int fd, off\_toffset);

//addr映射起始地址，一般为NULL，让内核自动选择

//length被映射内存块的长度

//prot标志映射后能否被读写，其值为PROT\_EXEC,PROT\_READ,PROT\_WRITE,PROT\_NONE

//flags确定此内存映射能否被其他进程共享，MAP\_SHARED,MAP\_PRIVATE

//fd,offset,确定被映射的内存地址

返回成功映射后的地址，不成功返回MAP\_FAILED((void\*)-1);

intmunmap(void \*addr, size\_t length);//断开映射

//addr为映射后的地址，length为映射后的内存长度

例：将四个已申请到的缓冲帧映射到应用程序，用buffers指针记录。

buffers= (buffer\*)calloc (req.count, sizeof (\*buffers));

if(!buffers) {

fprintf(stderr, "Out of memory/n");

exit(EXIT\_FAILURE);

}

//映射

for(unsigned int n\_buffers = 0; n\_buffers < req.count; ++n\_buffers) {

structv4l2\_buffer buf;

memset(&buf,0,sizeof(buf));

buf.type= V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

buf.memory= V4L2\_MEMORY\_MMAP;

buf.index= n\_buffers;

//查询序号为n\_buffers的缓冲区，得到其起始物理地址和大小

if (-1== ioctl (fd, VIDIOC\_QUERYBUF, &buf))

exit(-1);

buffers[n\_buffers].length= buf.length;

//映射内存

buffers[n\_buffers].start=mmap (NULL,buf.length,PROT\_READ | PROT\_WRITE ,MAP\_SHARED,fd,buf.m.offset);

if(MAP\_FAILED == buffers[n\_buffers].start)

exit(-1);

}

8.缓冲区处理好之后，就可以开始获取数据了

//启动/停止数据流

VIDIOC\_STREAMON,VIDIOC\_STREAMOFF

intioctl(int fd, int request, const int \*argp);

//argp为流类型指针，如V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE.

在开始之前，还应当把缓冲帧放入缓冲队列：

VIDIOC\_QBUF//把帧放入队列

VIDIOC\_DQBUF//从队列中取出帧

intioctl(int fd, int request, struct v4l2\_buffer \*argp);

例：把四个缓冲帧放入队列，并启动数据流

unsignedint i;

enumv4l2\_buf\_type type;

//将缓冲帧放入队列

for (i= 0; i < 4; ++i)

{

structv4l2\_buffer buf;

buf.type= V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

buf.memory= V4L2\_MEMORY\_MMAP;

buf.index= i;

ioctl(fd, VIDIOC\_QBUF, &buf);

}

type =V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

ioctl(fd, VIDIOC\_STREAMON, &type);

//这有个问题，这些buf看起来和前面申请的buf没什么关系，为什么呢?

例：获取一帧并处理

structv4l2\_buffer buf;

CLEAR(buf);

buf.type= V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

buf.memory= V4L2\_MEMORY\_MMAP;

//从缓冲区取出一个缓冲帧

ioctl(fd, VIDIOC\_DQBUF, &buf);

//图像处理

process\_image(buffers[buf.index].start);

//将取出的缓冲帧放回缓冲区

ioctl(fd, VIDIOC\_QBUF, &buf);

补充：

常用的IOCTL命令：

VIDIOC\_REQBUFS：分配内存

VIDIOC\_QUERYBUF：把VIDIOC\_REQBUFS中分配的数据缓存转换成物理地址

VIDIOC\_QUERYCAP：查询驱动功能

VIDIOC\_ENUM\_FMT：获取当前驱动支持的视频格式

VIDIOC\_S\_FMT：设置当前驱动的频捕获格式

VIDIOC\_G\_FMT：读取当前驱动的频捕获格式

VIDIOC\_TRY\_FMT：验证当前驱动的显示格式

VIDIOC\_CROPCAP：查询驱动的修剪能力

VIDIOC\_S\_CROP：设置视频信号的边框

VIDIOC\_G\_CROP：读取视频信号的边框

VIDIOC\_QBUF：将空的缓冲帧放入队列

VIDIOC\_DQBUF：将含有数据的缓冲帧从队列中取回

VIDIOC\_STREAMON：开始视频显示函数

VIDIOC\_STREAMOFF：结束视频显示函数

VIDIOC\_QUERYSTD：检查当前视频设备支持的标准，例如PAL或NTSC。

VIDIOC\_G\_STD 获取视频标准VIDIOC\_S\_STD 设置视频标准

1. 服务器代码：

/\* 服务端程序 server.c \*/

//WB

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <sys/ioctl.h>

#include <sys/mman.h>

#include <stdlib.h>

#include <linux/types.h>

#include <linux/videodev.h>

#include <setjmp.h>

#include <string.h>

#include <signal.h>

#include <errno.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include "convert.h"

#include "../avc-src-0.14/avc/common/T264.h"

#define SERVER\_PORT 8888

T264\_t\* m\_t264;

T264\_param\_t m\_param;

char\* m\_pSrc;

char\* m\_pDst;

int\* m\_lDstSize;

char\* m\_pPoolData;

#define USB\_VIDEO "/dev/video0"

int cam\_fd;

struct video\_mmap cam\_mm;/\*视频内存映射\*/

/\*包含摄像头的基本信息，例如设备名称、

支持的最大最小分辨率、信号源信息等\*/

struct video\_capability cam\_cap;

/\*亮度、对比度等和voide\_mmap中的分辨率\*/

struct video\_picture cam\_pic;

struct video\_mbuf cam\_mbuf;/\*摄像头存储缓冲区的帧信息\*/

struct video\_window win;/\* 设备采集窗口参数\*/

char \*cam\_data = NULL;

int nframe;

void read\_video(char \*pixels,int w, int h)

{

int ret;

int frame=0;

cam\_mm.width = w;

cam\_mm.height = h;

/\* 对于单帧采集只需设置 frame=0\*/

cam\_mm.frame = 0;

cam\_mm.format=VIDEO\_PALETTE\_YUV420P; //change by 091215

/\* 若调用成功，则激活设备真正开始一帧图像的截取，是非阻塞的\*/

ret = ioctl(cam\_fd,VIDIOCMCAPTURE,&cam\_mm);

if( ret<0 ) {

printf("ERROR: VIDIOCMCAPTURE\n");

}

/\* 函数判断该帧图像是否截取完毕，成功返回表示截取完毕\*/

ret = ioctl(cam\_fd,VIDIOCSYNC,&frame);

if( ret<0 ) {

printf("ERROR: VIDIOCSYNC\n");

}

}

void config\_vid\_pic()

{

char cfpath[100];

FILE \*cf;

int ret;

if (ioctl(cam\_fd, VIDIOCGPICT, &cam\_pic) < 0) {

printf("ERROR:VIDIOCGPICT\n");

}

/\*图像采集格式,网眼V2000只支持VIDEO\_PALETTE\_YUV420P\*/

cam\_pic.palette = VIDEO\_PALETTE\_YUV420P; //change by 091215

#if 0

cam\_pic.brightness = 30464;

cam\_pic.hue = 36000;

cam\_pic.colour = 0;

cam\_pic.contrast = 43312;

cam\_pic.whiteness = 10312;

cam\_pic.depth = 24;

#endif

cam\_pic.brightness = 30464;

cam\_pic.hue = 111;

cam\_pic.colour = 555;

cam\_pic.contrast = 43312;

cam\_pic.whiteness = 111;

/\*VIDEO\_PALETTE\_YUV420,bpp=12bit\*/

cam\_pic.depth = 12; //bpp == bytes per pixel,change by 091215

/\*设置摄像头缓冲中voideo\_picture信息\*/

ret = ioctl( cam\_fd, VIDIOCSPICT,&cam\_pic );

if( ret<0 ) {

close(cam\_fd);

printf("ERROR: VIDIOCSPICT,Can't set video\_picture format\n");

}

return;

}

void init\_video(int w,int h) /\* bpp == bytes per pixel\*/

{

int ret;

/\*设备的打开\*/

cam\_fd = open( USB\_VIDEO, O\_RDWR );

if( cam\_fd<0 )

printf("Can't open video device\n");

/\* 使用IOCTL命令VIDIOCGCAP，获取摄像头的基本信息，如最大，最小分辨率\*/

ret = ioctl( cam\_fd,VIDIOCGCAP,&cam\_cap ); /\* 摄像头的基本信息\*/

if( ret<0 ) {

printf("Can't get device information: VIDIOCGCAP\n");

}

printf("Device name:%s\nWidth:%d ~ %d\nHeight:%d ~ %d\n",cam\_cap.name, cam\_cap.maxwidth, cam\_cap.minwidth, cam\_cap.maxheight, cam\_cap.minheight);

if( ioctl(cam\_fd,VIDIOCGWIN,&win)<0 ) {

printf("ERROR:VIDIOCGWIN\n");

}

win.x = 0; //windows中的原点坐标

win.y = 0; //windows中的原点坐标

win.width=w; //capture area 宽度

win.height=h; //capture area 高度

/\*使用IOCTL命令VIDIOCSWIN，设置摄像头的基本信息\*/

if (ioctl(cam\_fd, VIDIOCSWIN, &win) < 0) {

printf("ERROR:VIDIOCSWIN\n");

}

/\*设置摄像头voideo\_picture信息\*/

config\_vid\_pic();

/\*使用IOCTL命令VIDIOCGCAP，获取获得摄像头存储缓冲区的帧信息\*/

ret = ioctl(cam\_fd,VIDIOCGMBUF,&cam\_mbuf);

if( ret<0 ) {

printf("ERROR:VIDIOCGMBUF,Can't get video\_mbuf\n");

}

printf("Frames:%d\n",cam\_mbuf.frames);

nframe = cam\_mbuf.frames;

/\*接着把摄像头对应的设备文件映射到内存区\*/

cam\_data = (char\*)mmap(0, cam\_mbuf.size, PROT\_READ|PROT\_WRITE,MAP\_SHARED,cam\_fd,0);

if( cam\_data == MAP\_FAILED ) {

printf("ERROR:mmap\n");

}

printf("Buffer size:%d\nOffset:%d\n",cam\_mbuf.size,cam\_mbuf.offsets[0]);

InitLookupTable();

}

void init\_param(T264\_param\_t\* param, const char\* file)

{

int total\_no;

FILE\* fd;

char line[255];

int32\_t b;

if (!(fd = fopen(file,"r")))

{

printf("Couldn't open parameter file %s.\n", file);

exit(-1);

}

memset(param, 0, sizeof(\*param));

fgets(line, 254, fd); sscanf(line,"%d", &b);

if (b != 4)

{

printf("wrong param file version, expect v4.0\n");

exit(-1);

}

fgets(line, 254, fd); sscanf(line,"%d", &param->width);

fgets(line, 254, fd); sscanf(line,"%d", &param->height);

fgets(line, 254, fd); sscanf(line,"%d", &param->search\_x);

fgets(line, 254, fd); sscanf(line,"%d", &param->search\_y);

fgets(line, 254, fd); sscanf(line,"%d", &total\_no);

fgets(line, 254, fd); sscanf(line,"%d", &param->iframe);

fgets(line, 254, fd); sscanf(line,"%d", &param->idrframe);

fgets(line, 254, fd); sscanf(line,"%d", &param->b\_num);

fgets(line, 254, fd); sscanf(line,"%d", &param->ref\_num);

fgets(line, 254, fd); sscanf(line,"%d", &param->enable\_rc);

fgets(line, 254, fd); sscanf(line,"%d", &param->bitrate);

fgets(line, 254, fd); sscanf(line,"%f", &param->framerate);

fgets(line, 254, fd); sscanf(line,"%d", &param->qp);

fgets(line, 254, fd); sscanf(line,"%d", &param->min\_qp);

fgets(line, 254, fd); sscanf(line,"%d", &param->max\_qp);

fgets(line, 254, fd); sscanf(line,"%d", &param->enable\_stat);

fgets(line, 254, fd); sscanf(line,"%d", &param->disable\_filter);

fgets(line, 254, fd); sscanf(line,"%d", &param->aspect\_ratio);

fgets(line, 254, fd); sscanf(line,"%d", &param->video\_format);

fgets(line, 254, fd); sscanf(line,"%d", &param->luma\_coeff\_cost);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_INTRA16x16) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_INTRA4x4) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_INTRAININTER) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_HALFPEL) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_QUARTPEL) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_SUBBLOCK) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_FULLSEARCH) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_DIAMONDSEACH) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_FORCEBLOCKSIZE) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_FASTINTERPOLATE) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_SAD) \* b;

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_EXTRASUBPELSEARCH) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->flags |= (USE\_SCENEDETECT) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_16x16P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_16x8P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_8x16P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_8x8P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_8x4P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_4x8P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_4x4P) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_16x16B) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_16x8B) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_8x16B) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &b);

param->block\_size |= (SEARCH\_8x8B) \* (!!b);

fgets(line, 254, fd); sscanf(line,"%d", &param->cpu);

fgets(line, 254, fd); sscanf(line, "%d", &param->cabac);

// fgets(line, 254, fd); sscanf(line,"%s", src\_path);

// fgets(line, 254, fd); sscanf(line,"%s", out\_path);

// fgets(line, 254, fd); sscanf(line,"%s", rec\_path);

// param->rec\_name = rec\_path;

fclose(fd);

}

void init\_encoder()

{

//编码准备

const char\* paramfile = "fastspeed.txt";

/\*获取fastspeed.txt文件信息，打开编码库\*/

init\_param(&m\_param, paramfile);

m\_param.direct\_flag = 1;

/\*t264编码的打开\*/

m\_t264 = T264\_open(&m\_param);

m\_lDstSize = m\_param.height \* m\_param.width + (m\_param.height \* m\_param.width >> 1);

/\*分配t264解码后数据存放的内存\*/

m\_pDst = (uint8\_t\*)T264\_malloc(m\_lDstSize, CACHE\_SIZE);

/\*分配内存用于存放一帧数据长度的数据\*/

m\_pPoolData = malloc(m\_param.width\*m\_param.height\*3/2);

}

void udps\_respon(int sockfd,int w,int h)

{

struct sockaddr\_in addrsrc;

struct sockaddr\_in addrdst;

int addrlen,n;

int32\_t iActualLen;

int row\_stride = w\*3\*h/2;

bzero(&addrdst,sizeof(struct sockaddr\_in));

addrdst.sin\_family=AF\_INET;

/\*客户端PC机IP地址\*/

addrdst.sin\_addr.s\_addr=inet\_addr("172.18.23.26");

addrdst.sin\_port=htons(SERVER\_PORT);

while(1)

{

/\*数据的采集\*/

read\_video(NULL,w,h);

/\*对采集到的数据通过H264编码\*/

iActualLen = T264\_encode(m\_t264, cam\_data, m\_pDst, row\_stride);

printf("encoded:%d, %d bytes.\n",row\_stride,iActualLen);

/\*frame\_num：存放帧号\*/

memcpy(m\_pPoolData,&m\_t264->frame\_num,1);

/\*m\_pDst解码后的数据\*/

memcpy(m\_pPoolData+1, m\_pDst, iActualLen);

iActualLen++;

/\*使用UDP协议发送编码后的数据到客服端\*/

sendto(sockfd,m\_pPoolData,iActualLen,0,(struct sockaddr\*)&addrdst,sizeof(struct sockaddr\_in));

}

}

void free\_dev()

{

printf("free device\n");

close(cam\_fd);

}

/\*主函数入口\*/

int main(void)

{

int sockfd;

struct sockaddr\_in addr;

printf("start 2.0...\n");

/\*创建套接字,是UDP\*/

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd<0)

{

printf("0-");

printf("Socket Error\n");

exit(1);

}

bzero(&addr,sizeof(struct sockaddr\_in));

addr.sin\_family=AF\_INET;

addr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

addr.sin\_port=htons(SERVER\_PORT);

/\*套接字绑定\*/

if(bind(sockfd,(struct sockaddr \*)&addr,sizeof(struct sockaddr\_in))<0 )

{

printf(stderr,"Bind Error:%s\n",strerror(errno));

exit(1);

}

/\*该函数完成编码前的准备\*/

init\_encoder();

atexit( &free\_dev );

/\*采集数据前的初始化函数\*/

init\_video(m\_param.width,m\_param.height);

/\*使用UDP协议发送采集到的数据\*/

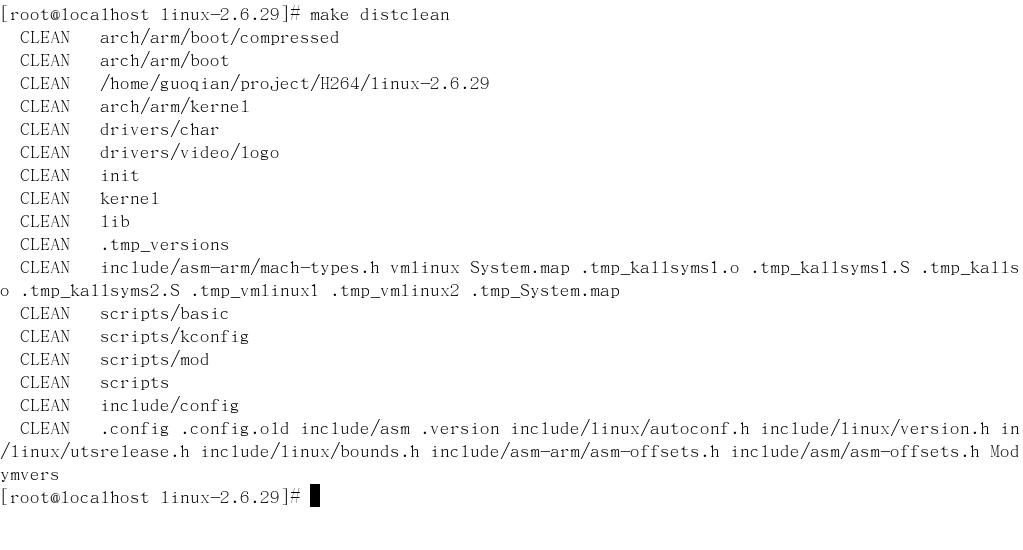
udps\_respon(sockfd,m\_param.width,m\_param.height);

close(sockfd);

}

# 五、项目开发

1. 清除内核中间文件、配置文件

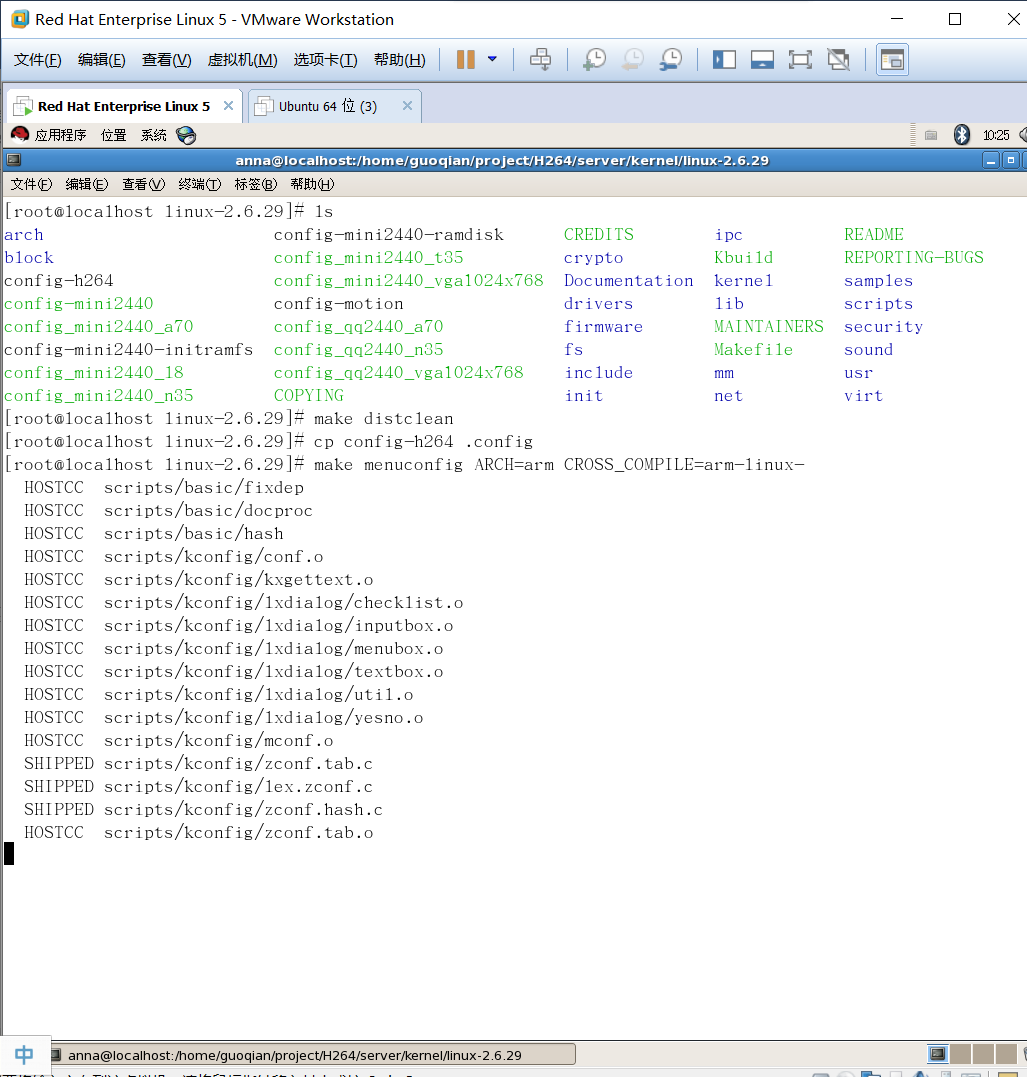


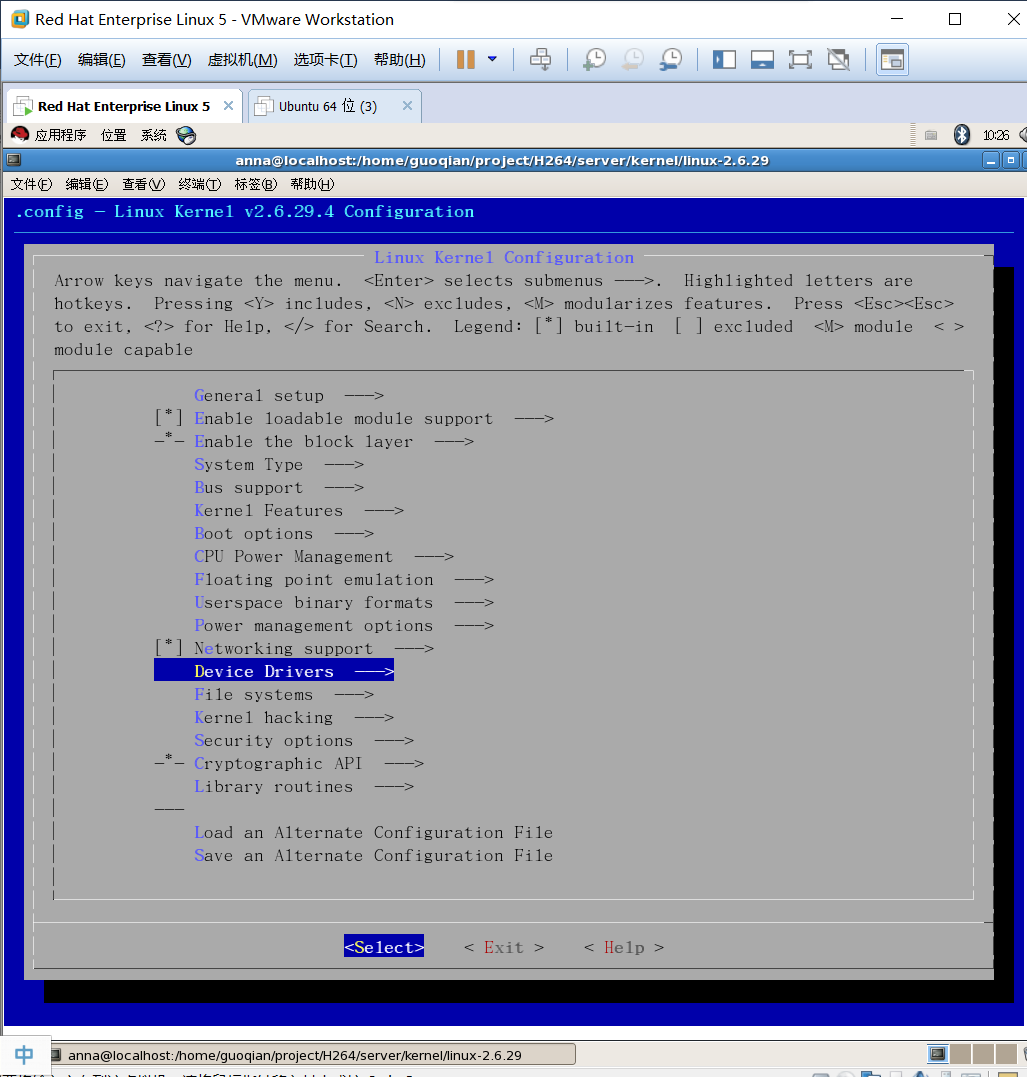
2、选择参考的配置文件，NFS起根文件系统

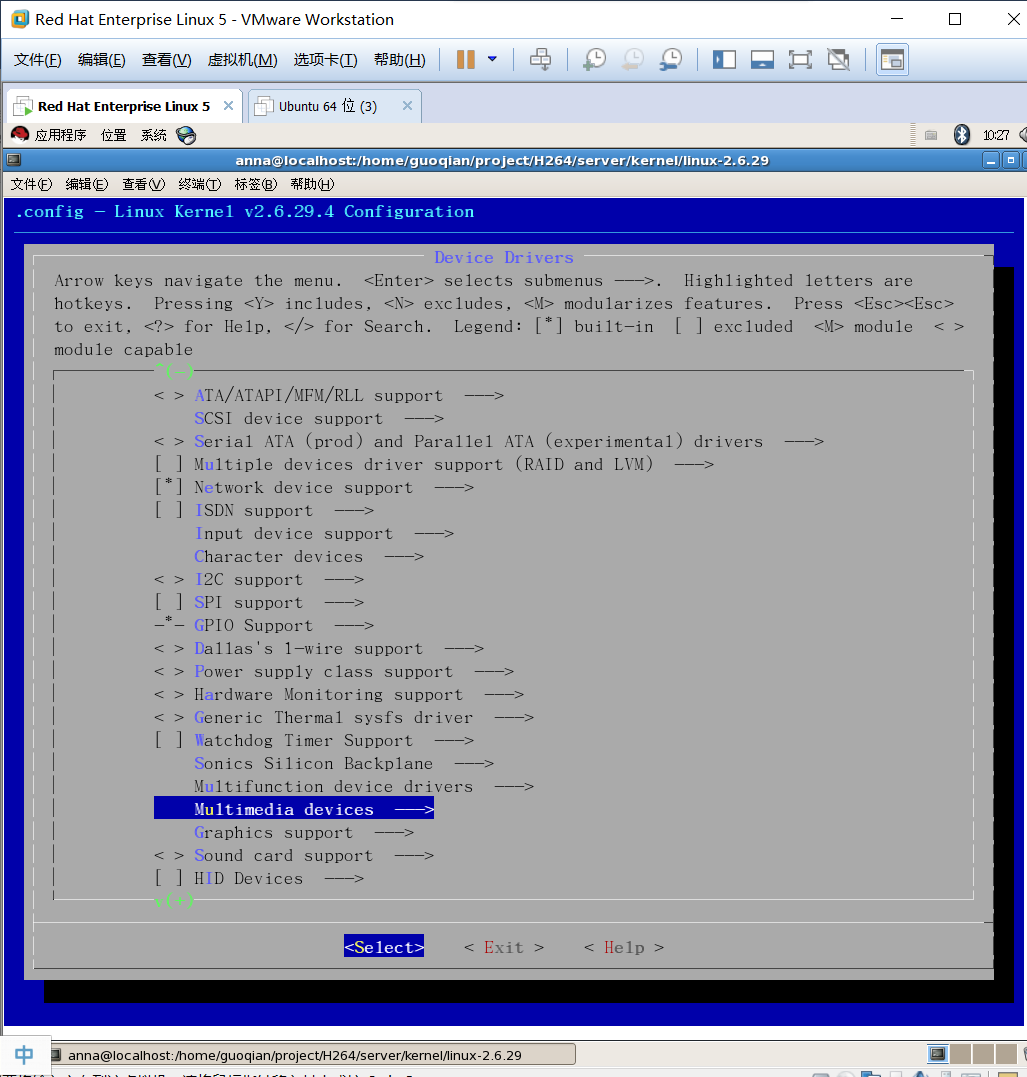
cp config-h264 .config

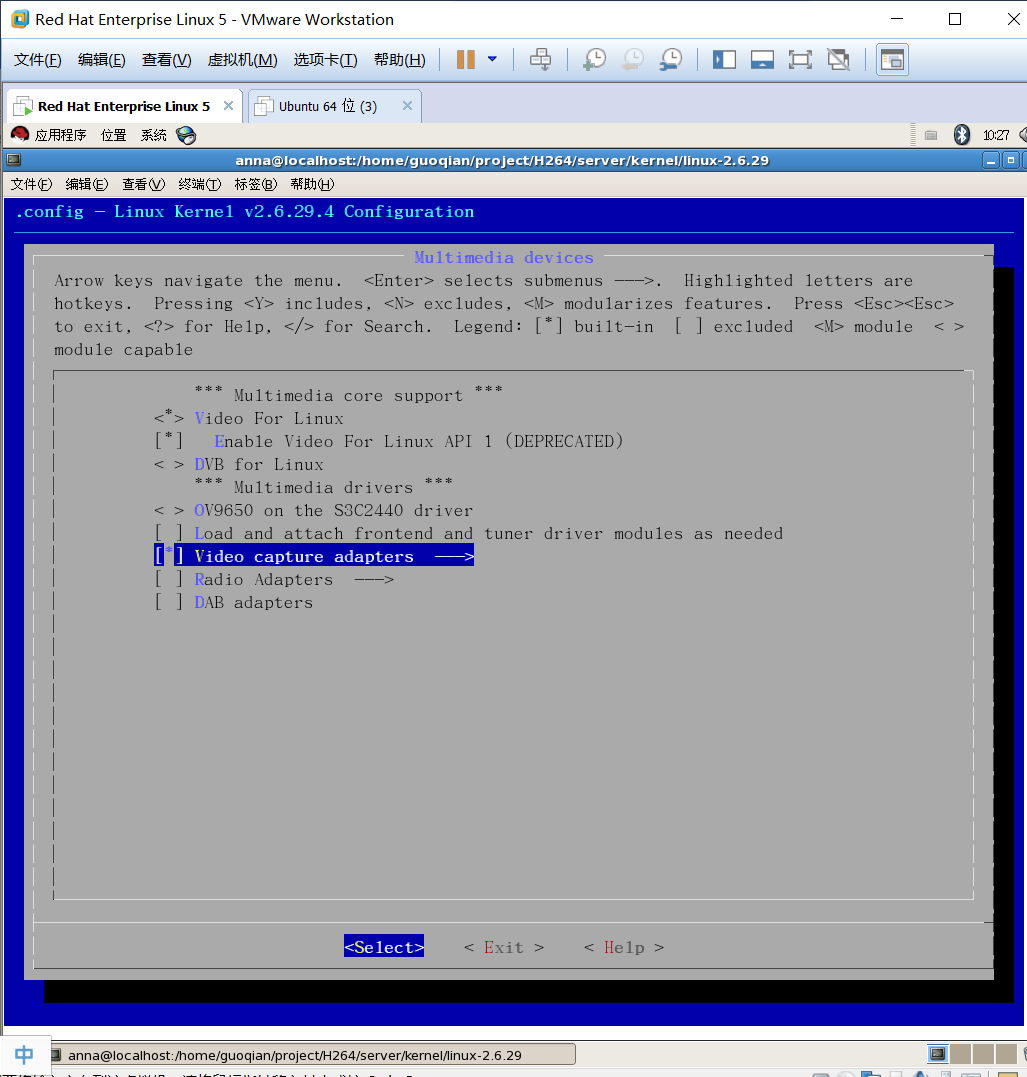
3、配置内核：

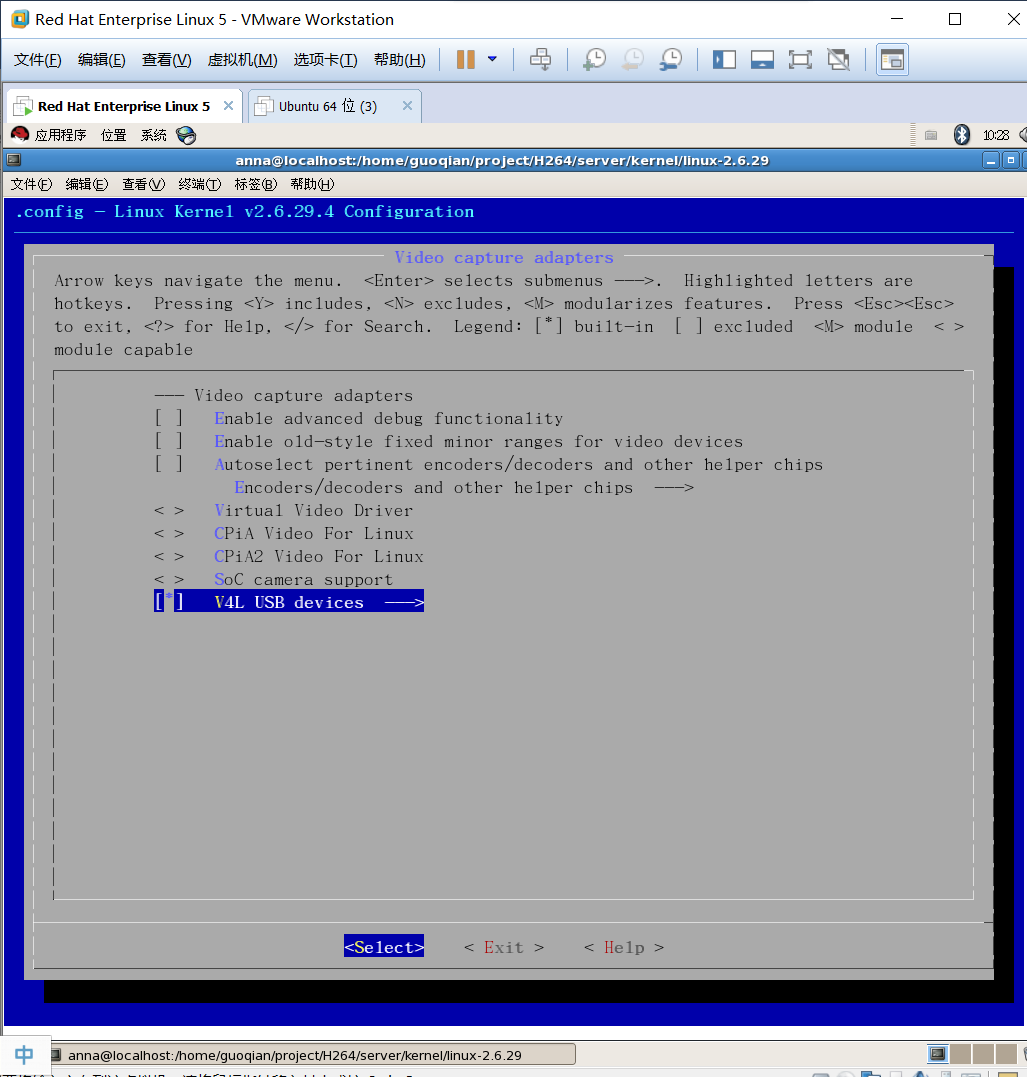
make menuconfig ARCH=arm CROSS\_COMPILE=arm-linux-







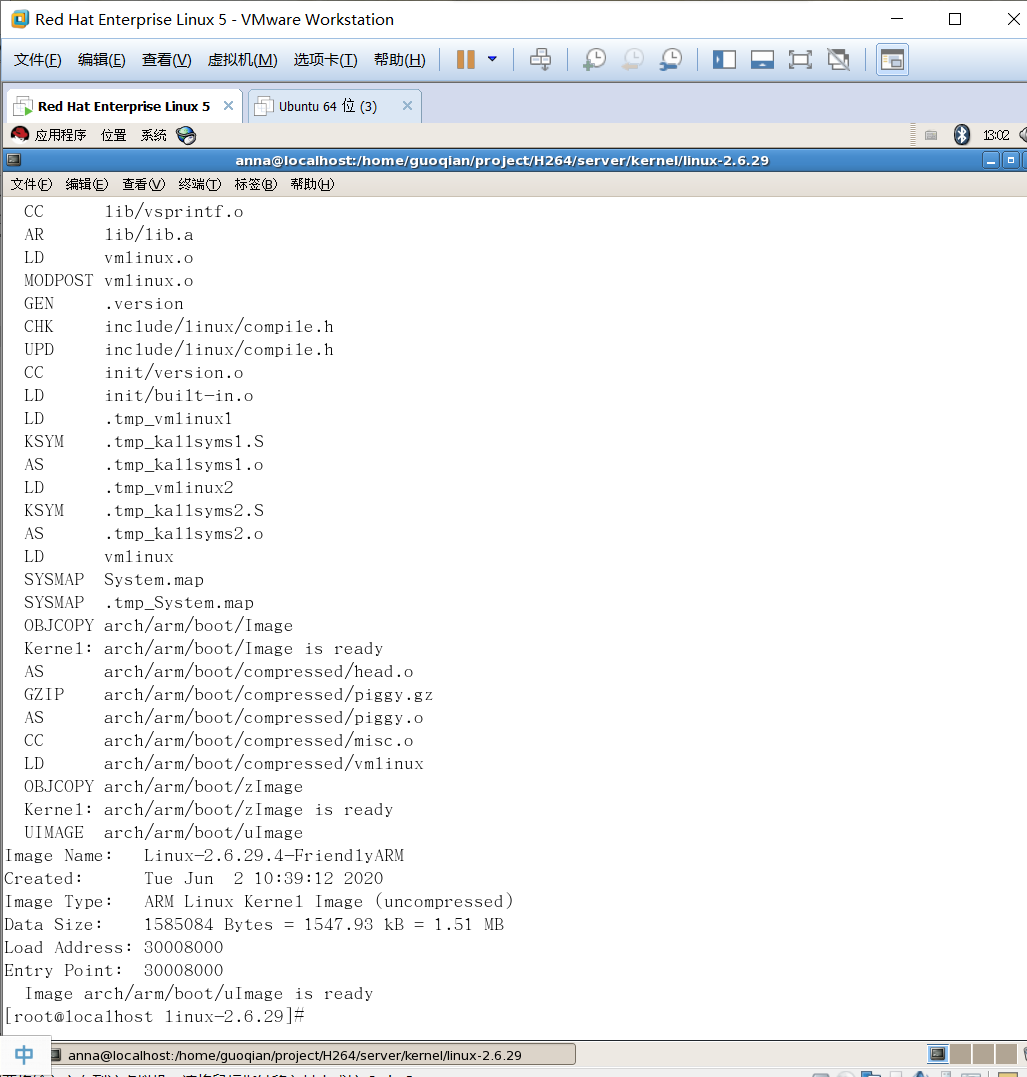




保存，退出

4、编译内核

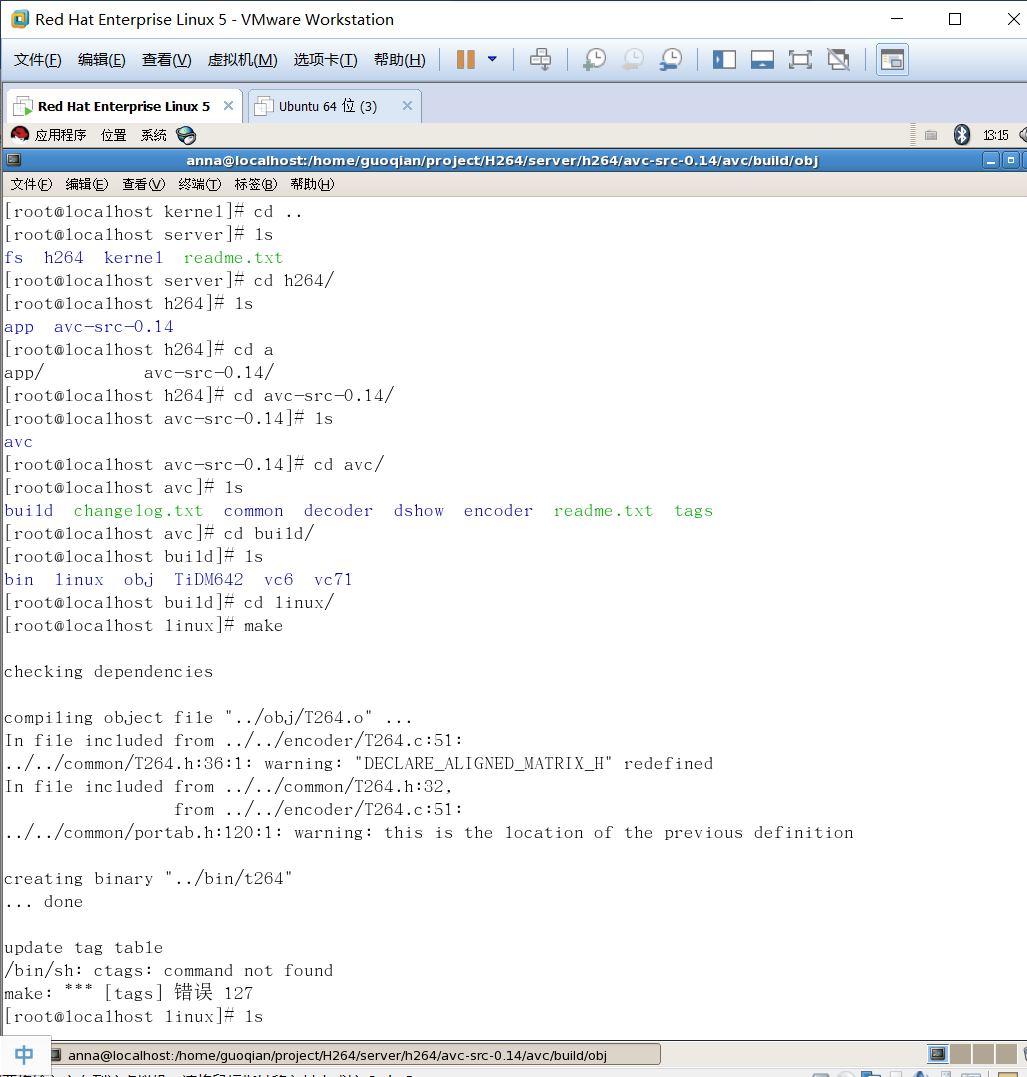
make uImage ARCH=arm CROSS\_COMPILE=arm-linux-

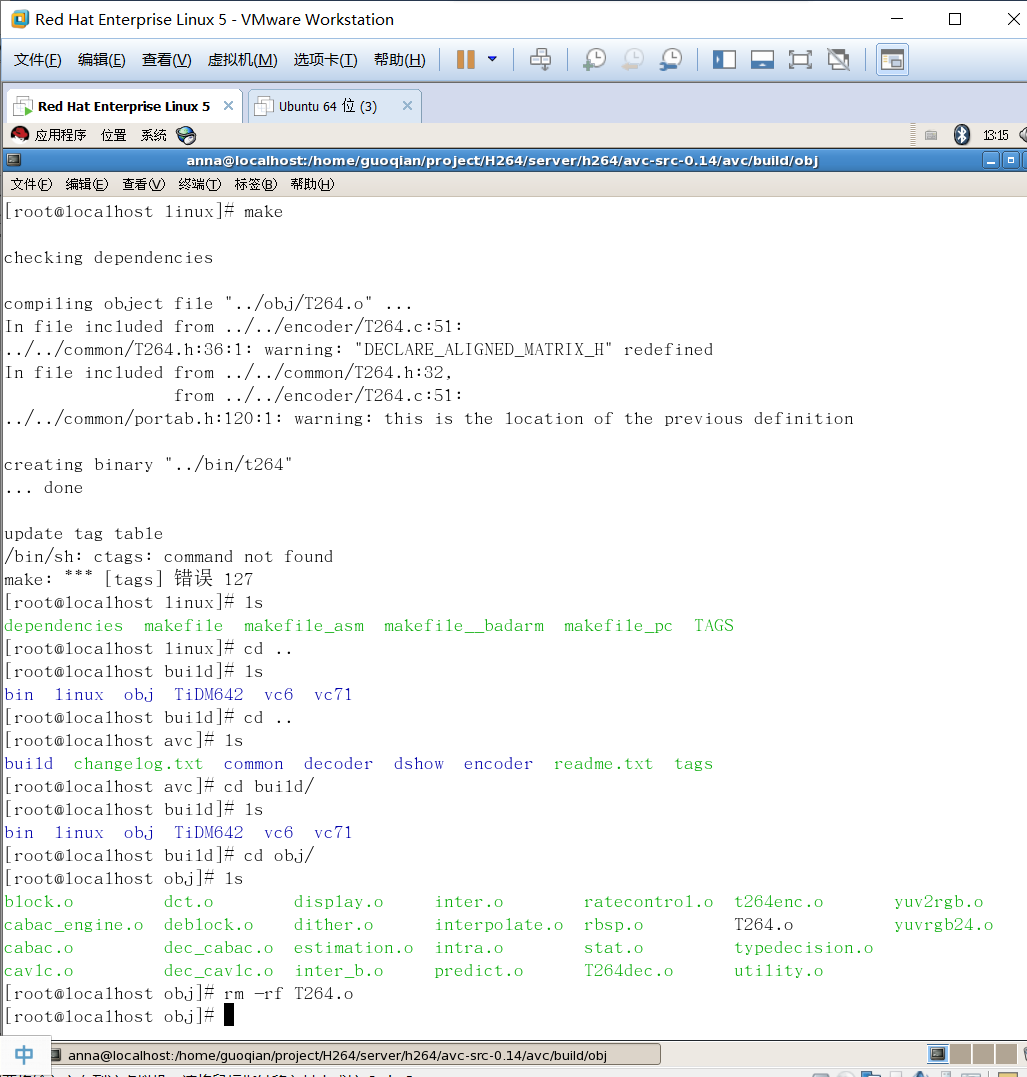


5、把内核映像uImage从arch/arm/boot/下复制到tftp目录下。

cp arch/arm/boot/uImage /tftpboot/uImage-h264

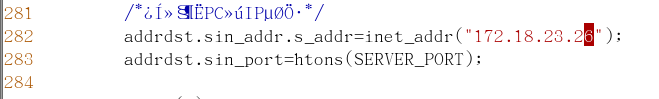
6、H.264编码库移植



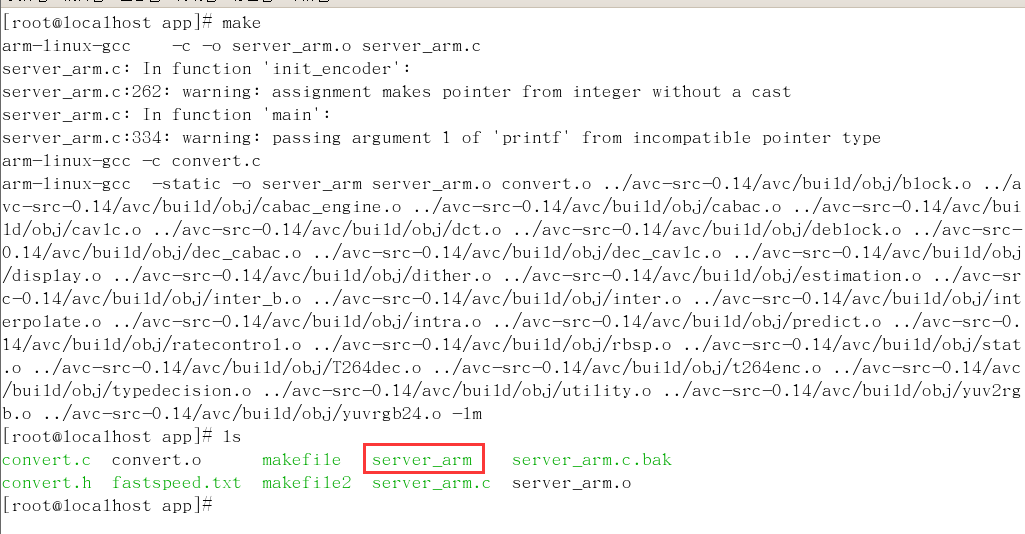


7、编译应用传输程序

需要修改客户端PC机的IP地址



编译



生成可执行程序