参考文献：

<https://blog.csdn.net/qq_53144843/article/details/126593977?sharetype=blog&shareId=126593977&sharerefer=APP&sharesource=qq_67785494&sharefrom=qq> ——Linux之V4L2驱动框架

**Linux之V4L2驱动框架**

目录

一、V4L2简介

二、V4L2操作流程

  1.打开摄像头

2.查询设备的属性/能力/功能

3.获取摄像头支持的格式

4.设置摄像头的采集通道

5.设置/获取摄像头的采集格式和参数

6.申请帧缓冲、内存映射、入队

（1）申请帧缓冲

（2）内存映射

（3）入队

7.开启视频采集

8.读取数据、对数据进行处理

9.结束视频采集

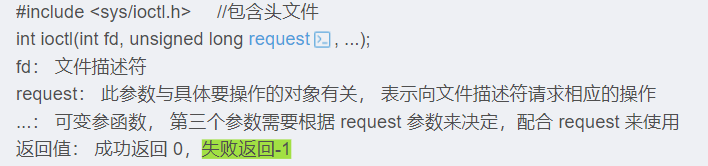
三、应用编程

**一、V4L2简介**

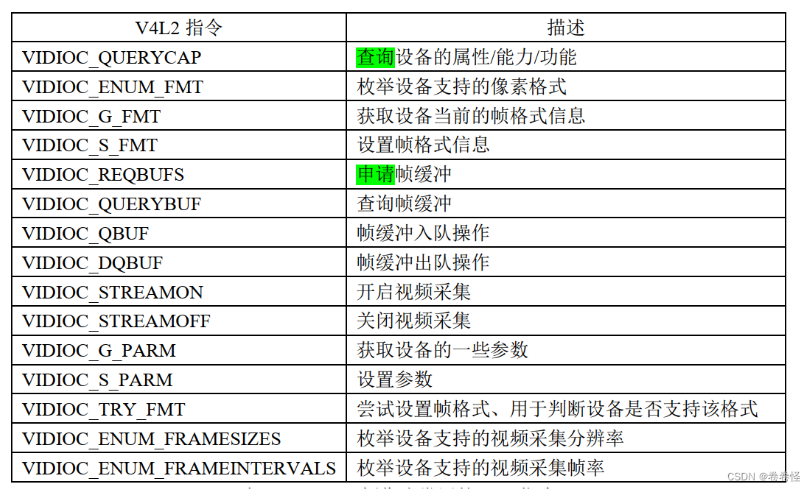
**V4L2**(Video for linux two)是 **Linux 内核中视频类设备的一套驱动框架**，**为视频类设备驱动开发和应用层提供了一套统一的接口规范**。**使用 V4L2 设备驱动框架注册的设备会在 Linux 系统/dev/目录下生成对应的设备节点文件**，**设备节点的名称通常为 videoX**（X 标准一个数字编号:/dev/videox），**每一个 videoX 设备文件就代表一个视频类设备**。**应用程序通过对 videoX 设备文件进行 I/O 操作来配置、使用设备类设备**。

**二、V4L2操作流程**

**V4L2摄像头驱动框架的访问**是**通过系统IO的接口** ------ **ioctl函数**，ioctl专用于**硬件控制的系统IO的接口**。

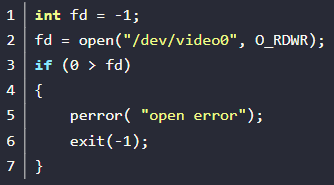


**ioctl()**是一个**文件 IO 操作的杂物箱**，可以处理的事情非常杂、不统一，**一般用于操作特殊文件或硬件外设**，可以**通过 ioctl 获取外设相关信息**。通过 ioctl()来完成，**搭配不同的 V4L2 指令**（request参数）请求不同的操作，这些指令定义在头文件 linux/videodev2.h 中，常用指令如下图：



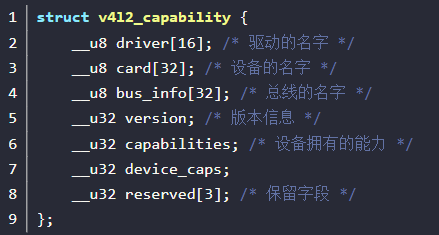
**1.打开摄像头**

**视频类设备**对应的**设备节点为/dev/videoX**， **X 为数字编号**，通常从 0 开始，**调用 open 打开**，得到文件描述符 fd。

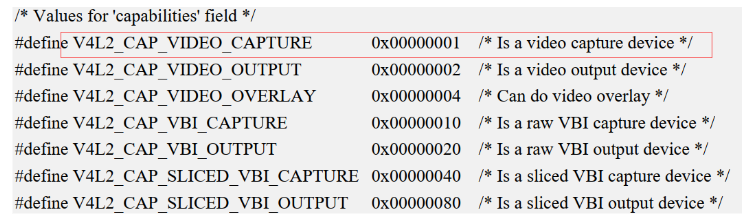


**2.查询设备的属性/能力/功能**

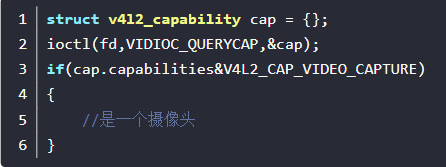
**打开设备后，需要查询设备的属性**，**确定该设备是否是一个视频采集类设备**。**通过 ioctl()**将**获取**到一个 struct **v4l2\_capability** 类型数据， struct v4l2\_capability **数据结构描述了设备的一些属性**，结构体定义如下：



这里关注capabilities 字段，该字段描述了设备拥有的能力，该字段的值如下：

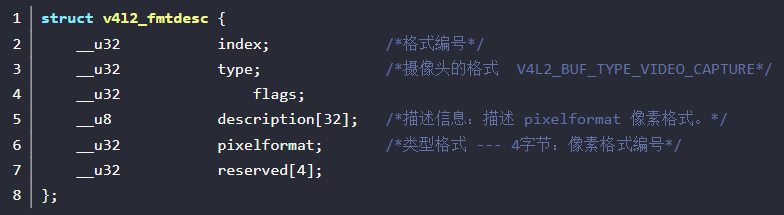


所以**可以通过判断 capabilities字段是否包含** **V4L2\_CAP\_VIDEO\_CAPTURE**、 **来确定它是否是一个摄像头设备**。

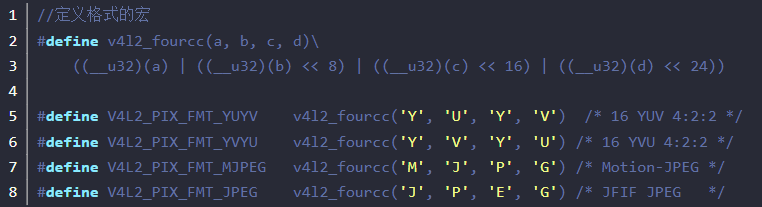


**3.获取摄像头支持的格式**

**获取支持的像素格式使用 VIDIOC\_ENUM\_FMT 指令：**



pixelfoemat像素格式：



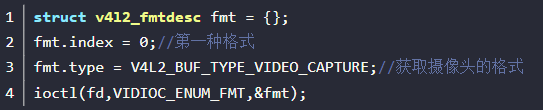
v4l2\_fourcc是 宏定义，通过这个宏以及对应的参数合成的一个u32 类型数据。

**type类型：**

**获取设备**的**哪种功能对应的像素格式**， 因为**有些设备**它**可能既支持视频采集功能**、**又支持视频输出等**其它的**功能**；



**type 字 段** 需 要 **在 调 用 ioctl() 之 前** **设 置 它 的 值** ， **对 于 摄 像 头 ， 需 要 将 type 字 段** 设 置 为**V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE**，**指定将要获取的是视频采集的像素格式**。



**4.设置摄像头的采集通道**

IMG_256

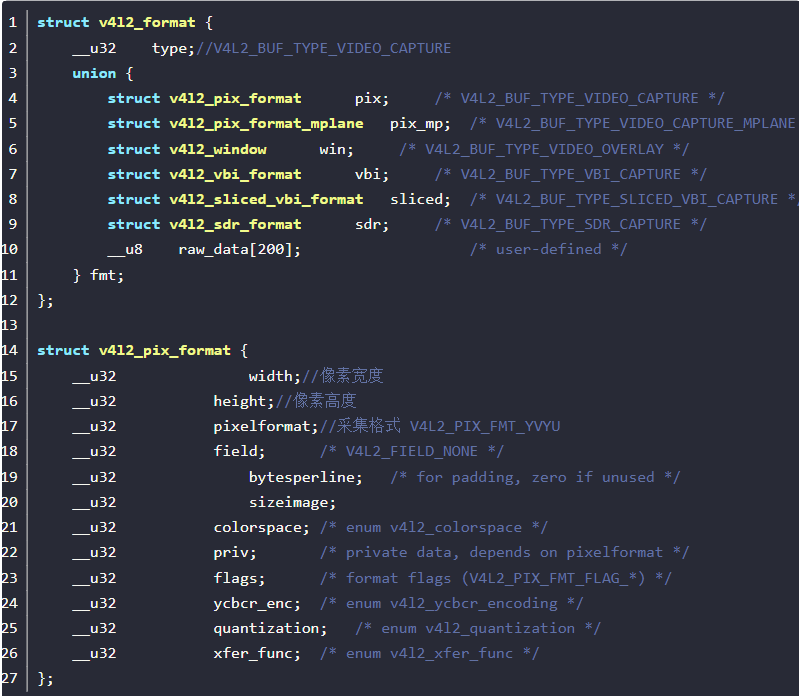
**5.设置/获取摄像头的采集格式和参数**

**使用 VIDIOC\_G\_FMT 指令**查看设备**当期的格式** 和

**使用 VIDIOC\_S\_FMT 指令设置设备的格式**；

IMG_256

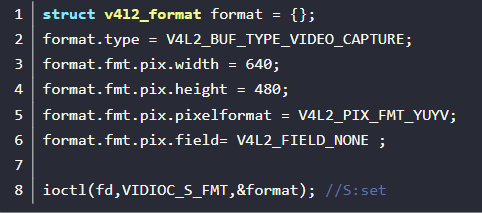
**ioctl()会将获取到的数据写入到** **fmt 指针所指向的对象中**或者会**使用 fmt 所指对象的数据去设置设备的格式**， **struct v4l2\_format 结构体描述了格式相关的信息**。



**type 字段依然与前面介绍的结构体中的 type 字段意义相同**，不管是**获取格式、还是设置格式**都需要在**调用 ioctl()函数之前设置它的值**。

接下来是一个 union 共用体，**当 type 被设置为V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE 时**， **pix 变量生效**，它**是一个 struct v4l2\_pix\_format 类型变量**，**记录了视频帧格式相关的信息**。

**获取当前的格式、并设置格式：**



**6.申请帧缓冲、内存映射、入队**

**（1）申请帧缓冲**

**读取摄像头数据的方式**有两种：

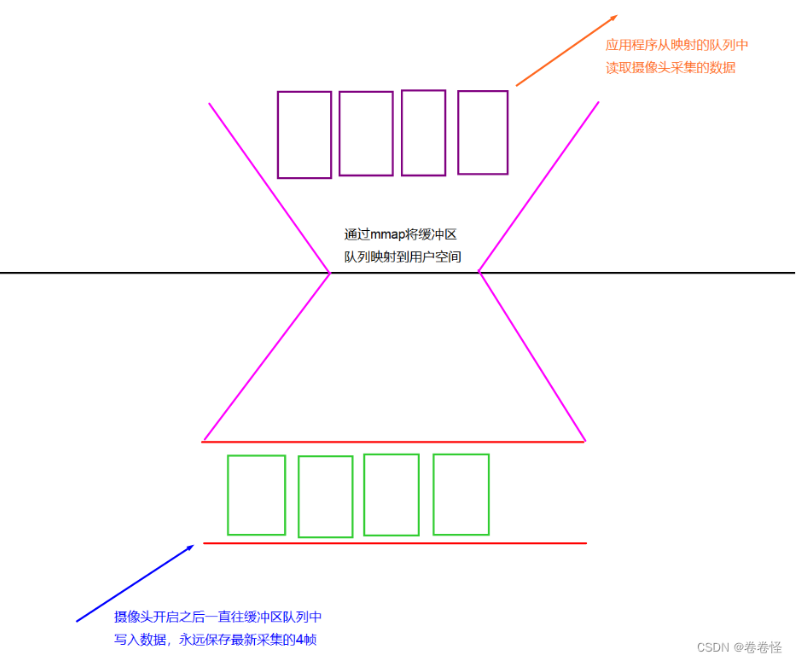
一种是 **read 方式**，直接通过 read()系统调用**读取摄像头采集到的数据**；

另一种是 **streaming 方式**。**使用 VIDIOC\_QUERYCAP 指令**查询**设备的属性**、得到一个 **struct v4l2\_capability 类型数据**， 其中 capabilities 字段记录了设备拥有的能力，

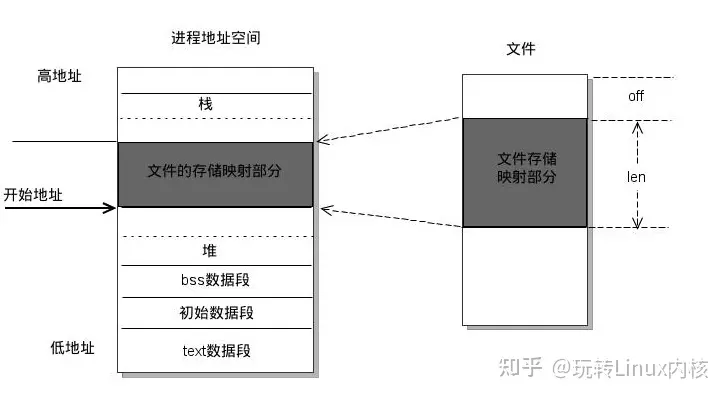
当该字段包含V4L2\_CAP\_READWRITE 时，表示设备支持 read I/O 方式读取数据；

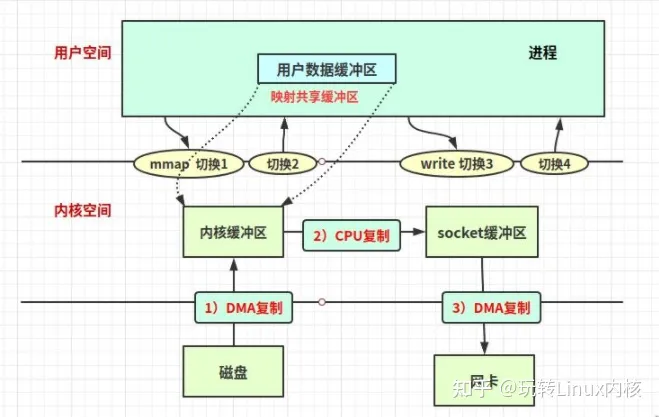
当该字段包含V4L2\_CAP\_STREAMING时，表示设备支持 streaming I/O 方式；

使用streaming I/O 方式，需要向设备申请帧缓冲，并将帧缓冲映射到应用程序进程地址空间中。



（mmap（mmap 即 memory map，也就是内存映射，mmap 是一种[内存映射文件](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98%E6%98%A0%E5%B0%84%E6%96%87%E4%BB%B6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_43778179/article/details/_blank)的方法，即**将一个文件或者其它对象映射到进程的地址空间，实现文件磁盘地址和进程虚拟地址空间中一段虚拟地址的一一对映关系**。实现这样的映射关系后，进程就可以采用指针的方式读写操作这一段内存，而系统会自动回写脏页面到对应的文件磁盘上，即完成了对文件的操作而不必再调用 read、write 等系统调用函数）





详细可见链接：

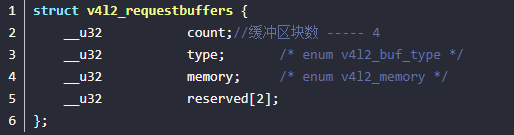
<https://blog.csdn.net/weixin_43778179/article/details/139408868）>

接上：

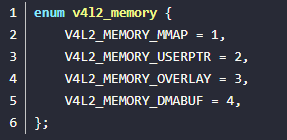
使用 VIDIOC\_REQBUFS 指令可申请帧缓冲：

ioctl(int fd, VIDIOC\_REQBUFS, struct v4l2\_requestbuffers \*reqbuf);

调用 ioctl()需要传入一个 struct v4l2\_requestbuffers \*指针， struct v4l2\_requestbuffers 结构体描述了申请帧缓冲的信息， ioctl()会根据 reqbuf 所指对象填充的信息进行申请。

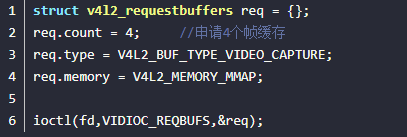


type 字段与前面所提及到的 type 字段意义相同，count 字段用于指定申请帧缓冲的数量。  
memory 字段可取值如下：

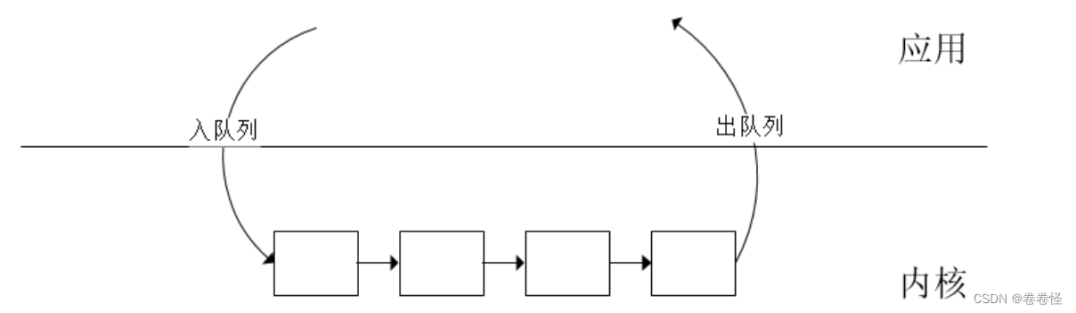


通常将 memory 设置为 V4L2\_MEMORY\_MMAP 。

申请缓存：



streaming I/O 方式会在内核空间中维护一个帧缓冲队列， 驱动程序会将从摄像头读取的一帧数据写入到队列中的一个帧缓冲，接着将下一帧数据写入到队列中的下一个帧缓冲；当应用程序需要读取一帧数据时，需要从队列中取出一个装满一帧数据的帧缓冲，这个取出过程就叫做出队；当应用程序处理完这一帧数据后，需要再把这个帧缓冲加入到内核的帧缓冲队列中，这个过程叫做入队。



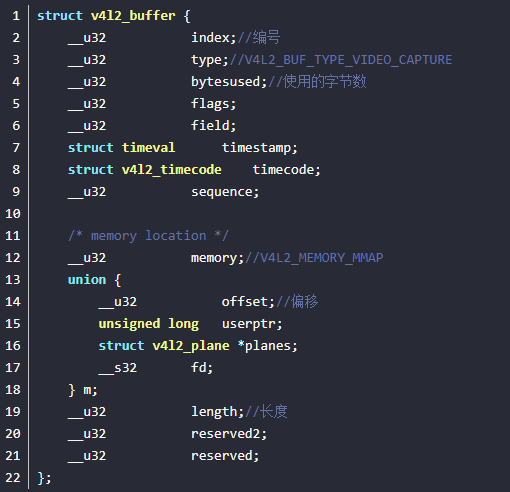
使用 VIDIOC\_REQBUFS 指令申请帧缓冲， 该缓冲区是由内核所维护的，应用程序不能直接读取该缓冲区的数据，需要将其映射到用户空间中，应用程序读取映射区的数据实际上是读取内核维护的帧缓冲中的数据。

**（2）内存映射**

在映射之前，需要查询帧缓冲的信息：帧缓冲的长度、偏移量。使用VIDIOC\_QUERYBUF指令查询：

ioctl(int fd, VIDIOC\_QUERYBUF, struct v4l2\_buffer \*buf);

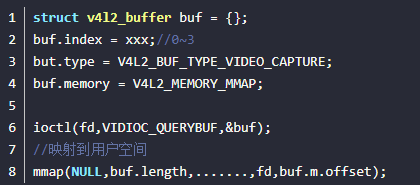
调用 ioctl()需要传入一个 struct v4l2\_buffer \*指针， struct v4l2\_buffer 结构体描述了帧缓冲的信息， ioctl()会将获取到的数据写入到 buf 指针所指的对象中。



index 字段表示一个编号， 申请的多个帧缓冲、 每一个帧缓冲都有一个编号，从 0 开始。一次 ioctl()调用只能获取指定编号对应的帧缓冲的信息，所以要获取多个帧缓冲的信息，需要重复调用多次，每调用一次ioctl()、 index 加 1，指向下一个帧缓冲。

type 字段和memory 字段与前面介绍的一样。length 字段表示帧缓冲的长度，共同体中的 offset 表示帧缓冲的偏移量。因为应用程序通过 VIDIOC\_REQBUFS 指令申请帧缓冲时，内核会向操作系统申请一块内存空间作为帧缓冲区，内存空间的大小就等于申请的帧缓冲数量 \* 每一个帧缓冲的大小，每一个帧缓冲对应到这一块内存空间的某一段，所以它们都有一个地址偏移量。

申请帧缓冲后、调用 mmap()将帧缓冲映射到用户地址空间



**（3）入队**

使用 VIDIOC\_QBUF 指令将帧缓冲放入到内核的帧缓冲队列中，调用 ioctl()之前，需要设置 struct v4l2\_buffer 类型对象的 memory、 type 字段

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

**7.开启视频采集**

使用 VIDIOC\_DQBUF 指令开启视频采集**，**

**ioctl(int fd, VIDIOC\_STREAMON, int \*type); //开启视频采集**

**ioctl(int fd, VIDIOC\_STREAMOFF, int \*type); //停止视频采集**

**enum v4l2\_buf\_type buf\_type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;**

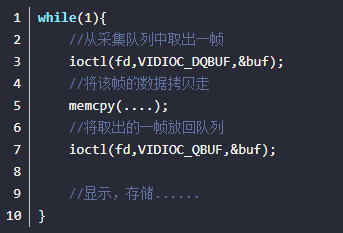
**ioctl(fd,VIDIOC\_STREAMON,&buf\_type);**

**8.读取数据、对数据进行处理**

开启视频采集之后，便可以去读取数据 ，直接读取每一个帧缓冲的在用户空间的映射区即可读取到摄像头采集的每一帧图像数据。在读取数据之前，需要将帧缓冲从内核的帧缓冲队列中取出，这个操作叫做帧缓冲出队。帧缓冲出队，便可读取数据，对数据进行处理：将摄像头采集的图像显示到 LCD屏上；数据处理完成之后，再将帧缓冲入队，往复操作。

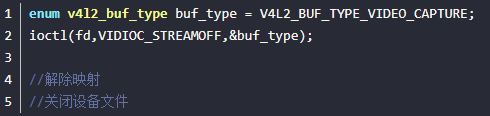
使用 VIDIOC\_DQBUF 指令执行出队操作:

ioctl(int fd, VIDIOC\_DQBUF, struct v4l2\_buffer \*buf);

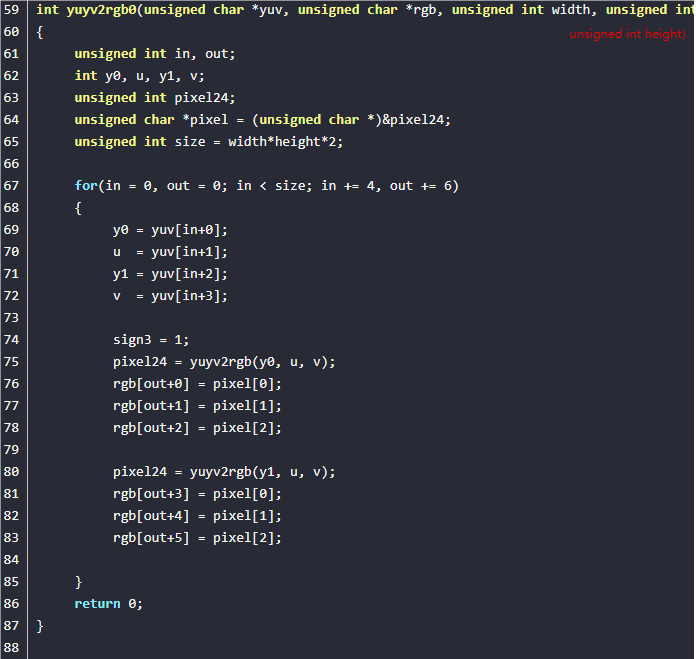
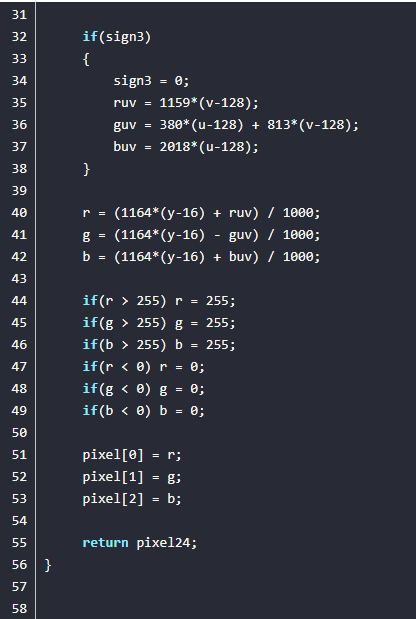
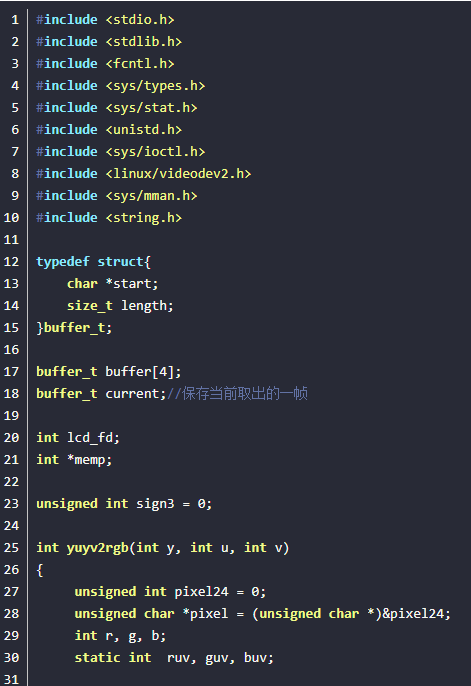


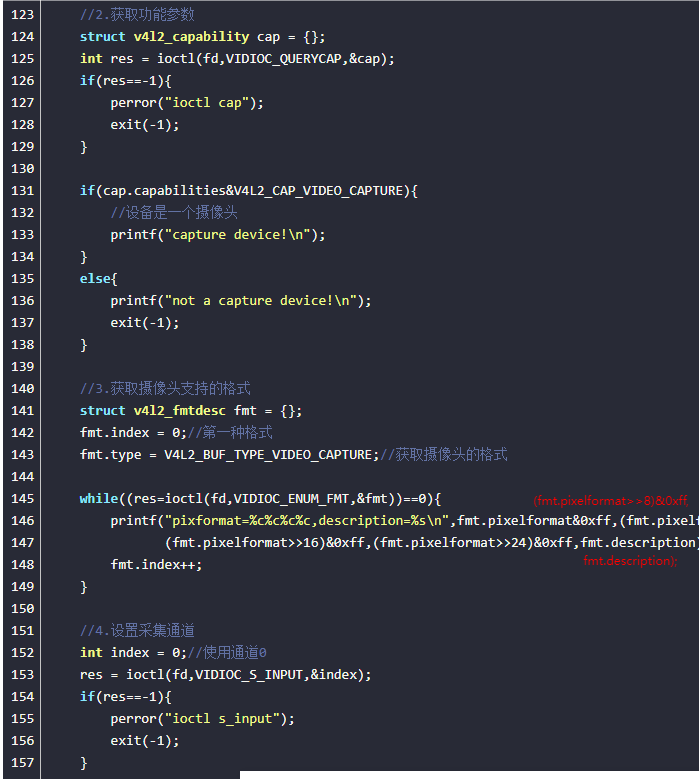
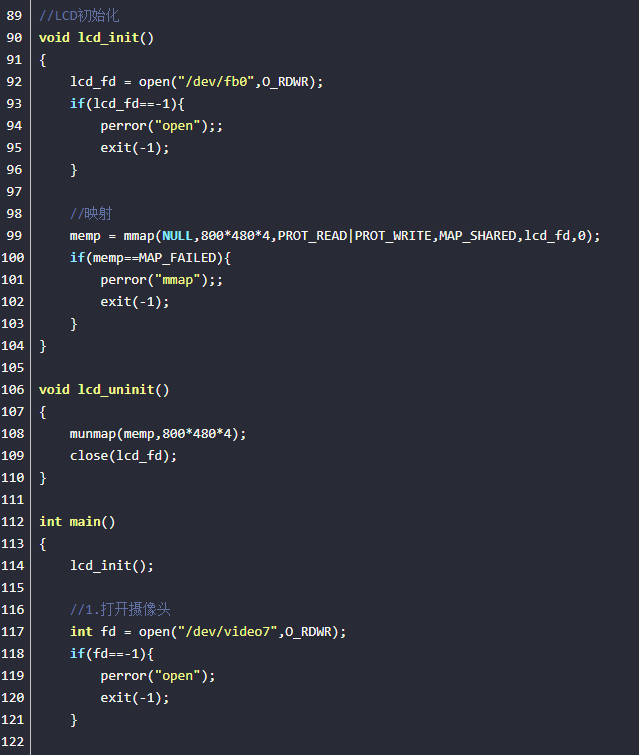
**9.结束视频采集**

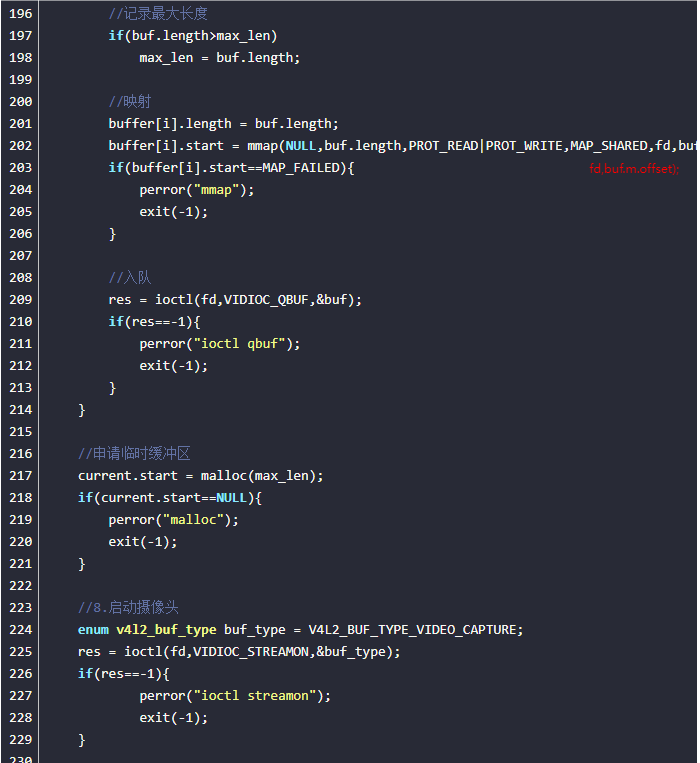
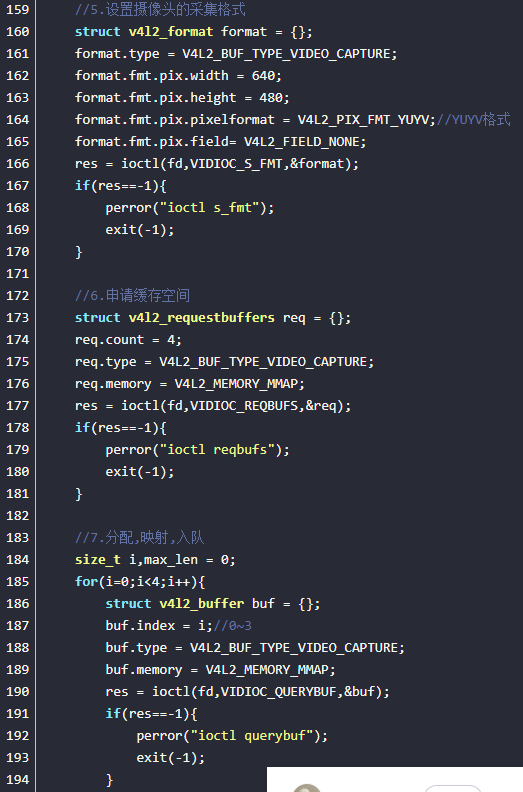
结束视频采集，使用 VIDIOC\_STREAMOFF 指令

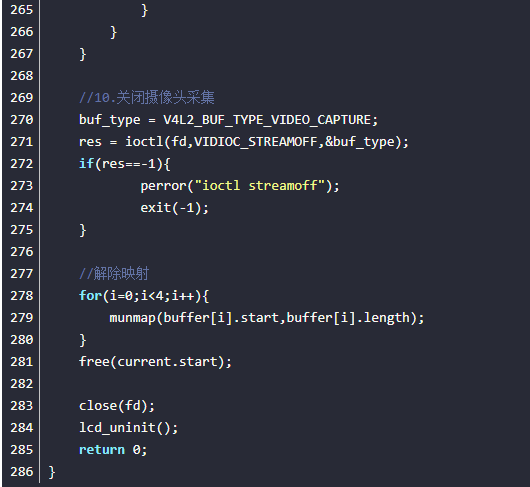
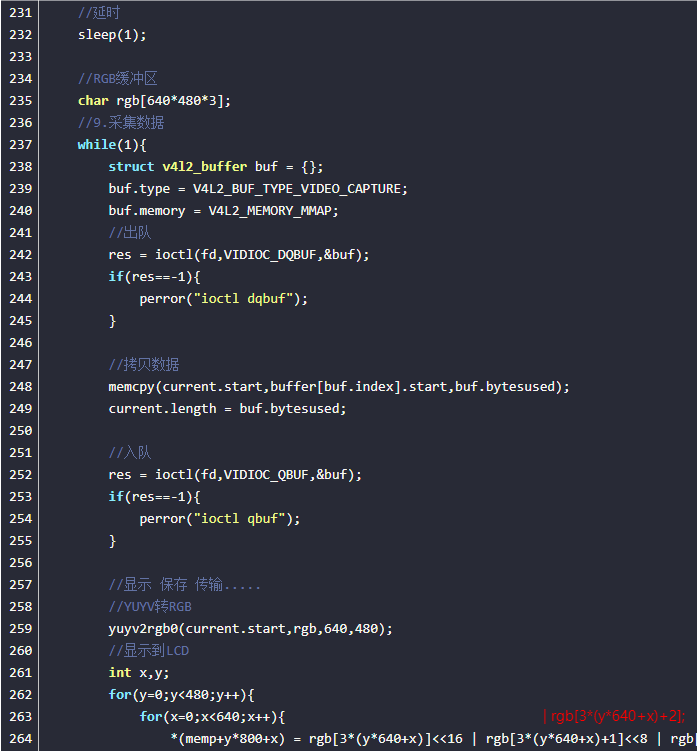


三、应用编程









代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <linux/videodev2.h>

#include <sys/mman.h>

#include <string.h>

typedef struct{

char \*start;

size\_t length;

}buffer\_t;

buffer\_t buffer[4];

buffer\_t current;//保存当前取出的一帧

int lcd\_fd;

int \*memp;

unsigned int sign3 = 0;

int yuyv2rgb(int y, int u, int v)

{

unsigned int pixel24 = 0;

unsigned char \*pixel = (unsigned char \*)&pixel24;

int r, g, b;

static int ruv, guv, buv;

if(sign3)

{

sign3 = 0;

ruv = 1159\*(v-128);

guv = 380\*(u-128) + 813\*(v-128);

buv = 2018\*(u-128);

}

r = (1164\*(y-16) + ruv) / 1000;

g = (1164\*(y-16) - guv) / 1000;

b = (1164\*(y-16) + buv) / 1000;

if(r > 255) r = 255;

if(g > 255) g = 255;

if(b > 255) b = 255;

if(r < 0) r = 0;

if(g < 0) g = 0;

if(b < 0) b = 0;

pixel[0] = r;

pixel[1] = g;

pixel[2] = b;

return pixel24;

}

int yuyv2rgb0(unsigned char \*yuv, unsigned char \*rgb, unsigned int width, unsigned int height)

{

unsigned int in, out;

int y0, u, y1, v;

unsigned int pixel24;

unsigned char \*pixel = (unsigned char \*)&pixel24;

unsigned int size = width\*height\*2;

for(in = 0, out = 0; in < size; in += 4, out += 6)

{

y0 = yuv[in+0];

u = yuv[in+1];

y1 = yuv[in+2];

v = yuv[in+3];

sign3 = 1;

pixel24 = yuyv2rgb(y0, u, v);

rgb[out+0] = pixel[0];

rgb[out+1] = pixel[1];

rgb[out+2] = pixel[2];

pixel24 = yuyv2rgb(y1, u, v);

rgb[out+3] = pixel[0];

rgb[out+4] = pixel[1];

rgb[out+5] = pixel[2];

}

return 0;

}

//LCD初始化

void lcd\_init()

{

lcd\_fd = open("/dev/fb0",O\_RDWR);

if(lcd\_fd==-1){

perror("open");;

exit(-1);

}

//映射

memp = mmap(NULL,800\*480\*4,PROT\_READ|PROT\_WRITE,MAP\_SHARED,lcd\_fd,0);

if(memp==MAP\_FAILED){

perror("mmap");;

exit(-1);

}

}

void lcd\_uninit()

{

munmap(memp,800\*480\*4);

close(lcd\_fd);

}

int main()

{

lcd\_init();

//1.打开摄像头

int fd = open("/dev/video7",O\_RDWR);

if(fd==-1){

perror("open");

exit(-1);

}

//2.获取功能参数

struct v4l2\_capability cap = {};

int res = ioctl(fd,VIDIOC\_QUERYCAP,&cap);

if(res==-1){

perror("ioctl cap");

exit(-1);

}

if(cap.capabilities&V4L2\_CAP\_VIDEO\_CAPTURE){

//设备是一个摄像头

printf("capture device!\n");

}

else{

printf("not a capture device!\n");

exit(-1);

}

//3.获取摄像头支持的格式

struct v4l2\_fmtdesc fmt = {};

fmt.index = 0;//第一种格式

fmt.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;//获取摄像头的格式

while((res=ioctl(fd,VIDIOC\_ENUM\_FMT,&fmt))==0){

printf("pixformat=%c%c%c%c,description=%s\n",fmt.pixelformat&0xff,(fmt.pixelformat>>8)&0xff,

(fmt.pixelformat>>16)&0xff,(fmt.pixelformat>>24)&0xff,fmt.description);

fmt.index++;

}

//4.设置采集通道

int index = 0;//使用通道0

res = ioctl(fd,VIDIOC\_S\_INPUT,&index);

if(res==-1){

perror("ioctl s\_input");

exit(-1);

}

//5.设置摄像头的采集格式

struct v4l2\_format format = {};

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

format.fmt.pix.width = 640;

format.fmt.pix.height = 480;

format.fmt.pix.pixelformat = V4L2\_PIX\_FMT\_YUYV;//YUYV格式

format.fmt.pix.field= V4L2\_FIELD\_NONE;

res = ioctl(fd,VIDIOC\_S\_FMT,&format);

if(res==-1){

perror("ioctl s\_fmt");

exit(-1);

}

//6.申请缓存空间

struct v4l2\_requestbuffers req = {};

req.count = 4;

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

req.memory = V4L2\_MEMORY\_MMAP;

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

if(res==-1){

perror("ioctl reqbufs");

exit(-1);

}

//7.分配,映射,入队

size\_t i,max\_len = 0;

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

struct v4l2\_buffer buf = {};

buf.index = i;//0~3

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

buf.memory = V4L2\_MEMORY\_MMAP;

res = ioctl(fd,VIDIOC\_QUERYBUF,&buf);

if(res==-1){

perror("ioctl querybuf");

exit(-1);

}

//记录最大长度

if(buf.length>max\_len)

max\_len = buf.length;

//映射

buffer[i].length = buf.length;

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

if(buffer[i].start==MAP\_FAILED){

perror("mmap");

exit(-1);

}

//入队

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

if(res==-1){

perror("ioctl qbuf");

exit(-1);

}

}

//申请临时缓冲区

current.start = malloc(max\_len);

if(current.start==NULL){

perror("malloc");

exit(-1);

}

//8.启动摄像头

enum v4l2\_buf\_type buf\_type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

res = ioctl(fd,VIDIOC\_STREAMON,&buf\_type);

if(res==-1){

perror("ioctl streamon");

exit(-1);

}

//延时

sleep(1);

//RGB缓冲区

char rgb[640\*480\*3];

//9.采集数据

while(1){

struct v4l2\_buffer buf = {};

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

buf.memory = V4L2\_MEMORY\_MMAP;

//出队

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

if(res==-1){

perror("ioctl dqbuf");

}

//拷贝数据

memcpy(current.start,buffer[buf.index].start,buf.bytesused);

current.length = buf.bytesused;

//入队

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

if(res==-1){

perror("ioctl qbuf");

}

//显示 保存 传输.....

//YUYV转RGB

yuyv2rgb0(current.start,rgb,640,480);

//显示到LCD

int x,y;

for(y=0;y<480;y++){

for(x=0;x<640;x++){

\*(memp+y\*800+x) = rgb[3\*(y\*640+x)]<<16 | rgb[3\*(y\*640+x)+1]<<8 | rgb[3\*(y\*640+x)+2];

}

}

}

//10.关闭摄像头采集

buf\_type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

res = ioctl(fd,VIDIOC\_STREAMOFF,&buf\_type);

if(res==-1){

perror("ioctl streamoff");

exit(-1);

}

//解除映射

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

munmap(buffer[i].start,buffer[i].length);

}

free(current.start);

close(fd);

lcd\_uninit();

return 0;

}

——————————————————————————————————

/\*

2025-2-5-1：17

模型部署参考:

"D:\BaiduNetdiskDownload\【正点原子】RK3568\【正点原子】RK3568开发板资料（A盘）-基础资料\10、用户手册\10、用户手册\01、测试文档\05【正点原子】ATK-DLRK3568\_Linux5.10\_AI例程测试手册v1.1.pdf"

"D:\BaiduNetdiskDownload\【正点原子】RK3568\【正点原子】RK3568开发板资料（A盘）-基础资料\10、用户手册\10、用户手册\01、测试文档\03【正点原子】ATK-DLRK3568\_AI例程测试手册V1.0.pdf"

摄像头参考：

"D:\BaiduNetdiskDownload\【正点原子】RK3568\【正点原子】RK3568开发板资料（A盘）-基础资料\10、用户手册\10、用户手册\03、辅助文档\30【正点原子】基于Buildroot系统Camera应用开发手册V1.0.pdf"

MIPI屏幕以及对设备树的了解见：

"D:\BaiduNetdiskDownload\【正点原子】RK3568\【正点原子】RK3568开发板资料（A盘）-基础资料\10、用户手册\10、用户手册\03、辅助文档\07【正点原子】ATK-DLRK3568三屏显示参考手册V1.0.pdf"

\*/