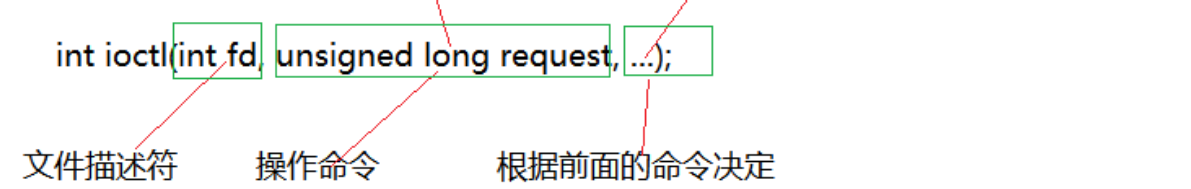
1. 打开设备

```
gec@ubuntu:/mnt/hgfs/online-7/VideoAudio/002/code$ ls /dev/v*
/dev/vcs   /dev/vcs3  /dev/vcs6  /dev/vcsa1  /dev/vcsa4  /dev/vcsa7        /dev/vhost-net   /dev/video1
/dev/vcs1  /dev/vcs4  /dev/vcs7  /dev/vcsa2  /dev/vcsa5  /dev/vga_arbiter  /dev/vhost-vsock  /dev/vmci
/dev/vcs2  /dev/vcs5  /dev/vcsa  /dev/vcsa3  /dev/vcsa6  /dev/vhci         /dev/video0       /dev/vsock
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
int main(void)
{
 //1.打开设备
  int fd = open("/dev/video0", O_RDWR);
  if(fd < 0)
  {
  perror("打开设备失败");
  return -1;
  }
  //9.关闭设备
  close(fd);
  return 0;
}

```

2. 获取支持格式

```c
#define VIDIOC_QUERYCAP          _IOR('V',  0, struct v4l2_capability)
#define VIDIOC_RESERVED          _IO('V',  1)
#define VIDIOC_ENUM_FMT         _IOWR('V',  2, struct v4l2_fmtdesc)
#define VIDIOC_G_FMT            _IOWR('V',  4, struct v4l2_format)
#define VIDIOC_S_FMT            _IOWR('V',  5, struct v4l2_format)
#define VIDIOC_REQBUFS          _IOWR('V',  8, struct v4l2_requestbuffers)
#define VIDIOC_QUERYBUF         _IOWR('V',  9, struct v4l2_buffer)
#define VIDIOC_G_FBUF            _IOR('V', 10, struct v4l2_framebuffer)
#define VIDIOC_S_FBUF            _IOW('V', 11, struct v4l2_framebuffer)
```

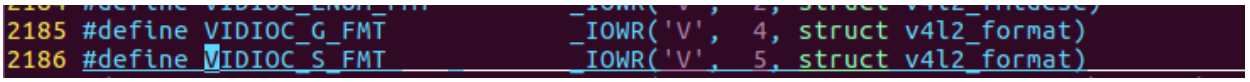


获取摄像头格式VIDIOC_ENUM_FMT--对应存储格式的结构体struct v4l2_fmtdesc

```c
#include <stdio.h>
#include <sys/types.h>
```

```c
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/videodev2.h>
int main(void)
{
 //1.打开设备
 int fd = open("/dev/video0", O_RDWR);
 if(fd < 0)
 {
 perror("打开设备失败");
 return -1;
 }
 //2.获取摄像头支持的格式ioctl(文件描述符， 命令， 与命令对应的结构体)
 struct v4l2_fmtdesc v4fmt;
 v4fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
 int i=0;
 while(1)
 {
 v4fmt.index = i++;
 int ret = ioctl(fd, VIDIOC_ENUM_FMT, &v4fmt);
 if(ret < 0)
 {
 perror("获取失败");
 break;
 }
 printf("index=%d\n", v4fmt.index);
 printf("flags=%d\n", v4fmt.flags);
 printf("description=%s\n", v4fmt.description);
 unsigned char *p = (unsigned char *)&v4fmt.pixelformat;
 printf("pixelformat=%c%c%c%c\n", p[0],p[1],p[2],p[3]);
 printf("reserved=%d\n", v4fmt.reserved[0]);
 }
 //9.关闭设备
 close(fd);
 return 0;
}
```

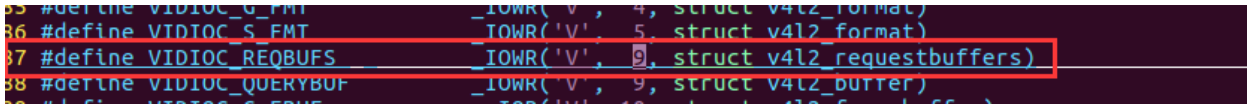
3.配置摄像头采集格式




```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/videodev2.h>
#include <string.h>

int main(void)
{
 //1.打开设备
 int fd = open("/dev/video0", O_RDWR);
 if(fd < 0)
 {
 perror("打开设备失败");
 return -1;
 }

 //3.设置采集格式
 struct v4l2_format vfmt;
 vfmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;//摄像头采集
 vfmt.fmt.pix.width = 640;//设置宽（不能任意）
 vfmt.fmt.pix.height = 480;//设置高
 vfmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;//设置视频采集格式
 int ret = ioctl(fd, VIDIOC_S_FMT, &vfmt);
 if(ret < 0)
 {
 perror("设置格式失败");
 }

 memset(&vfmt, 0, sizeof(vfmt));
 vfmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
 ret = ioctl(fd, VIDIOC_G_FMT, &vfmt);
 if(ret < 0)
```

```
37  {
38  perror("获取格式失败");
39  }
40
41  if(vfmt.fmt.pix.width == 640 && vfmt.fmt.pix.height == 480 &&
42  vfmt.fmt.pix.pixelformat == V4L2_PIX_FMT_YUYV)
43  {
44  printf("设置成功\n");
45  }else
46  {
47  printf("设置失败\n");
48  }
49
50  //9.关闭设备
51  close(fd);
52  return 0;
53 }
```

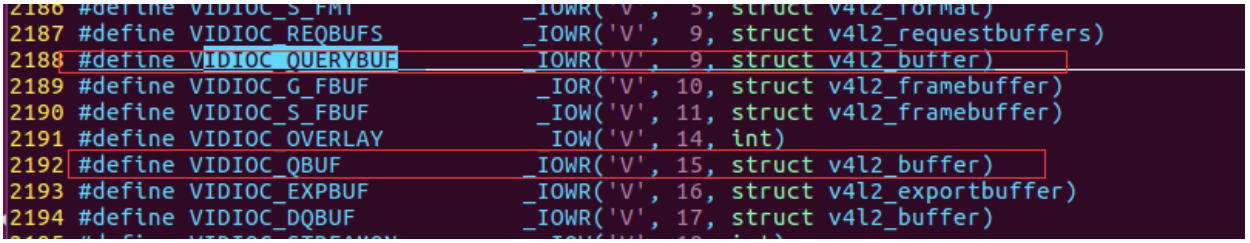
4.申请内核缓冲区队列




```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <sys/ioctl.h>
8  #include <linux/videodev2.h>
9  #include <string.h>
10
11 int main(void)
12 {
13  //1.打开设备
14  int fd = open("/dev/video0", O_RDWR);
15  if(fd < 0)
16  {
17  perror("打开设备失败");
```

```
18  return -1;
19  }
20  //3.设置采集格式
21  struct v4l2_format vfmt;
22  vfmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;//摄像头采集
23  vfmt.fmt.pix.width = 640;//设置宽（不能任意）
24  vfmt.fmt.pix.height = 480;//设置高
25  vfmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;//设置视频采集格式
26  int ret = ioctl(fd, VIDIOC_S_FMT, &vfmt);
27  if(ret < 0)
28  {
29  perror("设置格式失败");
30  }
31  //4.申请内核空间
32  struct v4l2_requestbuffers reqbuffer;
33  reqbuffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
34  reqbuffer.count = 4; //申请4个缓冲区
35  reqbuffer.memory = V4L2_MEMORY_MMAP ;//映射方式
36  ret = ioctl(fd, VIDIOC_REQBUFS, &reqbuffer);
37  if(ret < 0)
38  {
39  perror("申请队列空间失败");
40  }
41  //9.关闭设备
42  close(fd);
43  return 0;
44 }
45
```

5.把内核的缓冲区队列映射到用户空间




```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
```

```c
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/videodev2.h>
#include <string.h>
 #include <sys/mman.h>

 int main(void)
 {
  //1.打开设备
  int fd = open("/dev/video0", O_RDWR);
  if(fd < 0)
  {
  perror("打开设备失败");
  return -1;
  }

  //3.设置采集格式
  struct v4l2_format vfmt;
  vfmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;//摄像头采集
  vfmt.fmt.pix.width = 640;//设置宽（不能任意）
  vfmt.fmt.pix.height = 480;//设置高
  vfmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;//设置视频采集格式
  int ret = ioctl(fd, VIDIOC_S_FMT, &vfmt);
  if(ret < 0)
  {
  perror("设置格式失败");
  }

  //4.申请内核空间
  struct v4l2_requestbuffers reqbuffer;
  reqbuffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
  reqbuffer.count = 4; //申请4个缓冲区
  reqbuffer.memory = V4L2_MEMORY_MMAP ;//映射方式
  ret = ioctl(fd, VIDIOC_REQBUFS, &reqbuffer);
  if(ret < 0)
  {
  perror("申请队列空间失败");
  }

```

```
45  //5.映射
46  unsigned char *mptr[4];//保存映射后用户空间的首地址
47  unsigned int size[4];
48  struct v4l2_buffer mapbuffer;
49  //初始化type, index
50  mapbuffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
51  for(int i=0; i<4; i++)
52  {
53  mapbuffer.index = i;
54  ret = ioctl(fd, VIDIOC_QUERYBUF, &mapbuffer);//从内核空间中查询一个空间做
映射
55  if(ret < 0)
56  {
57  perror("查询内核空间队列失败");
58  }
59  mptr[i] = (unsigned char *)mmap(NULL, mapbuffer.length, PROT_READ|PROT_
WRITE,
60  MAP_SHARED, fd, mapbuffer.m.offset);
61  size[i]=mapbuffer.length;
62
63  //通知使用完毕--‘放回去’
64  ret = ioctl(fd, VIDIOC_QBUF, &mapbuffer);
65  if(ret < 0)
66  {
67  perror("放回失败");
68  }
69  }
70  //9.关闭设备
71  close(fd);
72  return 0;
73 }
74
```

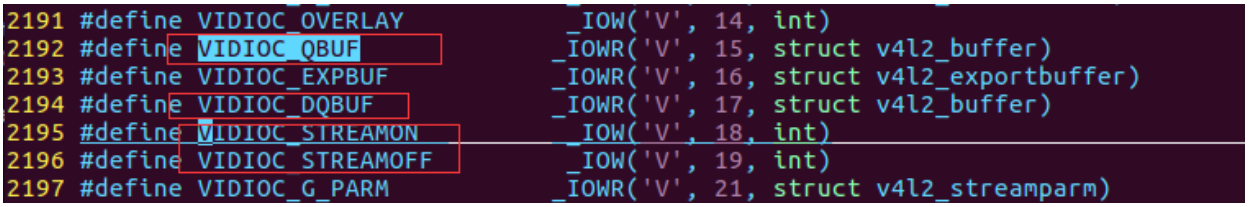
6.开始采集

VIDIOC_STREAMON（开始采集写数据到队列中）

VIDIOC_DQBUF（告诉内核我要某一个数据，内核不可以修改）

VIDIOC_QBUF（告诉内核我已经使用完毕）

VIDIOC_STREAMOFF（停止采集-不在向队列中写数据）




```
//6.开始采集
 int type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
 ret = ioctl(fd, VIDIOC_STREAMON, &type);
 if(ret < 0)
 {
 perror("开启失败");
 }

```

7.采集数据，

```
//从队列中提取一帧数据
 struct v4l2_buffer readbuffer;
 readbuffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
 ret = ioctl(fd, VIDIOC_DQBUF, &readbuffer);
 if(ret < 0)
 {
 perror("提取数据失败");
 }

  FILE *file=fopen("my.jpg", "w+");
  //mptr[readbuffer.index]
  fwrite(mptr[readbuffer.index], readbuffer.length, 1, file);
  fclose(file);

  //通知内核已经使用完毕
  ret = ioctl(fd, VIDIOC_QBUF, &readbuffer);
  if(ret < 0)
  {
  perror("放回队列失败");
  }

```

8. 停止采集

```
 ret = ioctl(fd, VIDIOC_STREAMOFF, &type);
```

9.释放映射

```
for(int i=0; i<4; i)
munmap(mptr[i], size[i]);

```

10.关闭设备

```
close(fd);
```

进入到字符界面 （ctrl+alt + f1）

进入到图形界面 （ctrl+alt +f7）

采集jpeg格式数据并且界面在lcd上显示 （代码参考video_show_jpg.c）

jpeg解码---libjpeg

ubuntu版本

在ubuntu要安装libjpeg8-dev

```
sudo apt install libjpeg8-dev
gcc -o video_show_jpg video_show_jpg.c -ljpeg
```

开发板版本

装备arm版本的libjpeg库 把libjpeg目录拷贝到工程当前目录下

```
arm-linux-gcc -o video_show_jpg video_show_jpg.c -L./libjpeg -I./libjpeg
-ljpeg
```

解码流程

```
int read_JPEG_file (const char *jpegData, char *rgbdata, int size)
{
 struct jpeg_error_mgr jerr;
 struct jpeg_decompress_struct cinfo;
 cinfo.err = jpeg_std_error(&jerr);
 //1创建解码对象并且初始化
 jpeg_create_decompress(&cinfo);
 //2.装备解码的数据
 //jpeg_stdio_src(&cinfo, infile);
  jpeg_mem_src(&cinfo,jpegData, size);
  //3.获取jpeg图片文件的参数
```

```
(void) jpeg_read_header(&cinfo, TRUE);
/* Step 4: set parameters for decompression */
//5.开始解码
(void) jpeg_start_decompress(&cinfo);
//6.申请存储一行数据的内存空间
int row_stride = cinfo.output_width * cinfo.output_components;
unsigned char *buffer = malloc(row_stride);
int i=0;
while (cinfo.output_scanline < cinfo.output_height) {
//printf("****%d\n",i);
(void) jpeg_read_scanlines(&cinfo, &buffer, 1);
memcpy(rgbdata+i*640*3, buffer, row_stride );
i++;
}
//7.解码完成
(void) jpeg_finish_decompress(&cinfo);
//8.释放解码对象
jpeg_destroy_decompress(&cinfo);
return 1;
}
```

1.摄像头采集的数据为YUYV4:2:2格式数据

```
存放的码流为：Y0 U0 Y1 V1 Y2 U2 Y3 V3
映射出像素点为：[Y0 U0 V1] [Y1 U0 V1] [Y2 U2 V3] [Y3 U2 V3]
```

从摄像头采集的一帧数据中读取4个字节Y0 U0 Y1 V1， 把这四个字节转两个像素，[Y0 U0 V1] [Y1 U0 V1]，
在这两个像素通过yuv转rgb公式转换为RGB像素

```
R = 1.164*Y + 1.596 * V - 222.9
G = 1.164*Y - 0.392 * U - 0.823 * V+ 135.6
B = 1.164*Y + 2.017 * U- 276.8
```

码流Y0 U0 Y1 V1 Y2 U2 Y3 V3 --》YUYV像素[Y0 U0 V1] [Y1 U0 V1] [Y2 U2 V3] [Y3 U2 V3]--》套用上面公式把YUYV像素转RGB像素

转换代码：

```
void yuyv_to_rgb(unsigned char *yuyvdata, unsigned char *rgbdata, int w, int h)
{
 //码流Y0 U0 Y1 V1 Y2 U2 Y3 V3 --》YUYV像素[Y0 U0 V1] [Y1 U0 V1] [Y2 U2 V3] [Y3 U2 V3]--》RGB像素
 int r1, g1, b1;
 int r2, g2, b2;
 for(int i=0; i<w*h/2; i++)
 {
 char data[4];
 memcpy(data, yuyvdata+i*4, 4);
  unsigned char Y0=data[0];
  unsigned char U0=data[1];
  unsigned char Y1=data[2];
  unsigned char V1=data[3];
  //Y0U0Y1V1 -->[Y0 U0 V1] [Y1 U0 V1]
  r1 = Y0+1.4075*(V1-128); if(r1>255)r1=255; if(r1<0)r1=0;
  g1 =Y0- 0.3455 * (U0-128) - 0.7169*(V1-128); if(g1>255)g1=255; if(g1<0)g1=0;
  b1 = Y0 + 1.779 * (U0-128); if(b1>255)b1=255; if(b1<0)b1=0;

  r2 = Y1+1.4075*(V1-128);if(r2>255)r2=255; if(r2<0)r2=0;
  g2 = Y1- 0.3455 * (U0-128) - 0.7169*(V1-128); if(g2>255)g2=255; if(g2<0)g2=0;
  b2 = Y1 + 1.779 * (U0-128); if(b2>255)b2=255; if(b2<0)b2=0;

  rgbdata[i*6+0]=r1;
  rgbdata[i*6+1]=g1;
  rgbdata[i*6+2]=b1;
  rgbdata[i*6+3]=r2;
  rgbdata[i*6+4]=g2;
  rgbdata[i*6+5]=b2;
  }
}
```

作业：

用v4l2采集YUYV格式数据，并且在lcd上显示， 注意：图形自己反转，运行结果截图和代码提交