# 第七章 摄像头V4L2编程

## 7.1 V4L2简介

Video for Linux two(Video4Linux2)简称V4L2，是V4L的改进版。V4L2是linux操作系统下一套用于采集图片、视频和音频数据的通用API接口，配合适当的视频采集设备和相应的驱动程序，可以实现图片、视频、音频等的采集。V4L2像一个优秀的快递员，将视频采集设备的图像数据安全、高效的传递给不同需求的用户。

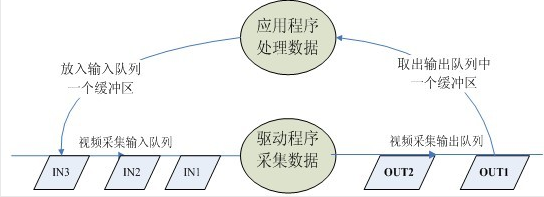
在Linux中，一切皆文件，所有外设都被看成一种特殊的文件，称为“设备文件”。视频设备也不例外，也可以可以看成是设备文件，可以像访问普通文件一样对其进行读写。V4L2驱动的摄像头的设备文件一般是/dev/videoX（X为任意数字，要与自己的设备相对应）。

V4L2支持三种方式来采集图像：内存映射方式(mmap)、直接读取方式(read)和用户指针。内存映射的方式采集速度较快，一般用于连续视频数据的采集，实际工作中的应用概率更高；直接读取的方式相对速度慢一些，所以常用于静态图片数据的采集；用户指针使用较少，如有兴趣可自行研究。由于内存映射方式的应用更广泛，所以本文重点讨论内存映射方式的视频采集。

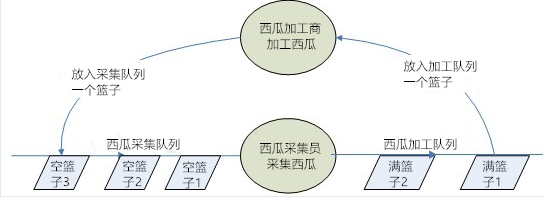
## 7.2 V4L2视频采集原理

在通过V4L2采集图像之前，我们需要做的很多，但是很重要的一步是分配帧缓冲区，并将分配的帧缓冲区从内核空间映射到用户空间，然后将申请到的帧缓冲区在视频采集输入队列排队，剩下的就是等待视频数据的到来。但是，万一视频数据真的来了是怎么个流动过程呢？这个我们有必要了解一下。

当启动视频采集后，驱动程序开始采集一帧图像数据，会把采集的图像数据放入视频采集输入队列的第一个帧缓冲区，一阵图像数据就算采集完成了。第一个帧缓冲区存满一帧图像数据后，驱动程序将该帧缓冲区移至视频采集输出队列，等待应用程序从输出队列取出，应用程序取出图像数据可以对图像数据进行处理或存储操作，然后将帧该缓冲区放入视频采集输入队列的尾部。驱动程序接下来采集下一帧数据，放入第二个缓冲区，同样的帧缓冲区存满一帧数据后，驱动程序将该缓冲区移至视频采集输出队列，应用程序将该帧缓冲区的图像数据取出后又将该帧缓冲区放入视频输入队列尾部，这样循环往复就实现了循环采集。流程如下图所示：



为了更好的理解这个过程，我们可以把“应用程序处理数据”比喻成“西瓜加工商加工西瓜”，“V4L2驱动程序采集数据”比喻成“西瓜采集员采集西瓜”，事先“西瓜加工商”会给“西瓜采集员”准备几个空篮子，然后“西瓜采集员”守着几个空篮子等待“瓜农”（图像采集设备，例如：摄像头）将空篮子装满，当“空篮子1”被“瓜农”装满以后，“西瓜采集员”会将装满西瓜的篮子放到“西瓜加工队列”等待“西瓜加工商”取走加工，当“西瓜加工商”取走装满西瓜的篮子中的西瓜的时候，“西瓜加工商”会将空篮子放回到事先给“西瓜采集员”准备好的西瓜采集队列的尾部。当“瓜农”装满下一个空篮子的时候，“西瓜采集员”同样的将装满西瓜的篮子放到“西瓜加工队列”等待“西瓜加工商”取走加工。这样，整个过程会持续不断的继续下去。



## 7.3 V4L2程序实现流程

使用V4L2进行视频采集，一般分为5个步骤：

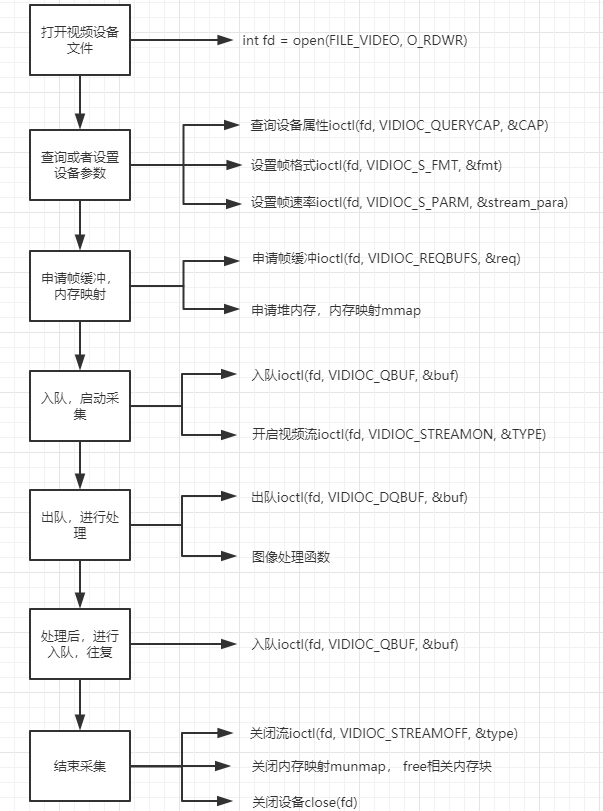
1. 打开设备，进行初始化参数设置，通过V4L2接口设置视频图像的采集窗口、采集的点阵大小和格式；
2. 申请图像帧缓冲，并进行内存映射，将这些帧缓冲区从内核空间映射到用户空间，便于应用程序读取、处理图像数据；
3. 将帧缓冲进行入队操作，启动视频采集；
4. 驱动开始视频数据的采集，应用程序从视频采集输出队列取出帧缓冲区，处理完后，将帧缓冲区重新放入视频采集输入队列，循环往复采集连续的视频数据；
5. 释放资源，停止采集工作。

在进行V4L2开发中，常用的命令标识符如下：

1. VIDIOC\_REQBUFS:分配内存；
2. VIDIOC\_QUERYBUF:把VIDIOC\_REQBUFS中分配的数据缓存转换成物理地址；
3. VIDIOC\_QUERYCAP:查询驱动功能；
4. VIDIOC\_ENUM\_FMT:获取当前驱动支持的视频格式；
5. VIDIOC\_S\_FMT:设置当前驱动的视频捕获格式；
6. VIDIOC\_G\_FMT:读取当前驱动的视频捕获格式；
7. VIDIOC\_TRY\_FMT:验证当前驱动的显示格式；
8. VIDIOC\_CROPCAP:查询驱动的修剪功能；
9. VIDIOC\_S\_CROP:设置视频信号的边框；
10. VIDIOC\_G\_CROP:读取视频信号的边框；
11. VIDIOC\_QBUF:把数据从缓存中读取出来；
12. VIDIOC\_DQBUF:把数据放回缓存队列；
13. VIDIOC\_STREAMOP:开始视频显示函数；
14. VIDIOC\_STREAMOFF:结束视频显示函数；
15. VIDIOC\_QUERYSTD:检查当前视频设备支持的标准，例如PAL或NTSC；

这些IO调用，有些是必须的，有些是可选择的。

具体流程如下图所示：



## 7.4 V4L2程序实例

V4L2的代码主要位于video2lcd/video/v4l2.c文件中，接下来就针对上文 V4L2程序实现流程和流程中使用的重要数据结构，结合v4l2.c文件中的代码进行说明。代码支持内存映射和直接读取两种方式，由于内存映射方式应用更广泛，本文只详细说明内存映射方式，直接读取方式与内存映射方式类似，可自行研究。

### 7.4.1 打开设备

应用程序能够使用阻塞模式或非阻塞模式打开视频设备，如果使用非阻塞模式调用视频设备，即使尚未捕获到信息，驱动依旧会把缓存（DQBUFF）里的东西返回给应用程序。如果使用非阻塞的方式打开摄像头设备，第2行代码中open函数的第二个参数修改为O\_RDWR | O\_NONBLOCK 即可。

70 iFd = open(strDevName, O\_RDWR);

71 if (iFd < 0)

72 {

73 DBG\_PRINTF("can not open %s\n", strDevName);

74 return -1;

75 }

### 7.4.2 查询设备属性

查询设备属性需要使用struct v4l2\_capability结构体，该结构体描述了视频采集设备的driver信息。

01 struct v4l2\_capability

02 {

03 \_\_u8 driver[16];       // 驱动名字

04 \_\_u8 card[32];        // 设备名字

05 \_\_u8 bus\_info[32];    // 设备在系统中的位置

06 \_\_u32 version;        // 驱动版本号

07 \_\_u32 capabilities;    // 设备支持的操作

08 \_\_u32 reserved[4];    // 保留字段

09 };

通过VIDIOC\_QUERYCAP命令来查询driver是否合乎规范。因为V4L2要求所有driver和device都支持这个ioctl。所以，通过VIDIOC\_QUERYCAP命令是否成功来判断当前device和driver是否符合V4L2规范。当然，这个命令执行成功的同时还能够得到设备足够的信息，如struct v4l2\_capability结构体所示内容。86~98行代码检查当前设备是否为capture设备，并检查使用内存映射还是直接读的方式获取图像数据。

78 iError = ioctl(iFd, VIDIOC\_QUERYCAP, &tV4l2Cap);

79 memset(&tV4l2Cap, 0, sizeof(struct v4l2\_capability));

80 iError = ioctl(iFd, VIDIOC\_QUERYCAP, &tV4l2Cap);

81 if (iError) {

82 DBG\_PRINTF("Error opening device %s: unable to query device.\n", strDevName);

83 goto err\_exit;

84 }

85

86 if (!(tV4l2Cap.capabilities & V4L2\_CAP\_VIDEO\_CAPTURE))

87 {

88 DBG\_PRINTF("%s is not a video capture device\n", strDevName);

89 goto err\_exit;

90 }

91

92 if (tV4l2Cap.capabilities & V4L2\_CAP\_STREAMING) {

93 DBG\_PRINTF("%s supports streaming i/o\n", strDevName);

94 }

95

96 if (tV4l2Cap.capabilities & V4L2\_CAP\_READWRITE) {

97 DBG\_PRINTF("%s supports read i/o\n", strDevName);

98 }

### 7.4.3 显示所有支持的格式

显示所有支持的格式需要用到struct v4l2\_fmtdesc结构体，该结构体描述当前camera支持的格式信息。

01 struct v4l2\_fmtdesc

02 {

03 \_\_u32 index; // 要查询的格式序号，应用程序设置

04 enum v4l2\_buf\_type type; // 帧类型，应用程序设置

05 \_\_u32 flags; // 是否为压缩格式

06 \_\_u8 description[32]; // 格式名称

07 \_\_u32 pixelformat; //所支持的格式

08 \_\_u32 reserved[4]; // 保留

09 };

使用VIDIOC\_ENUM\_FMT命令查询当前camera支持的所有格式。struct v4l2\_fmtdesc结构体中index要设置，从0开始；enum v4l2\_buf\_type type也要设置，如果使用的是camera设备，则enum v4l2\_buf\_type type要设置为V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE，因为camera是CAPTURE设备。结构体中的其他内容driver会填充。其中\_\_u32 pixelformat参数在设置图像帧格式时需要使用。

100 memset(&tFmtDesc, 0, sizeof(tFmtDesc));

101 tFmtDesc.index = 0;

102 tFmtDesc.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

103 while ((iError = ioctl(iFd, VIDIOC\_ENUM\_FMT, &tFmtDesc)) == 0) {

104 if (isSupportThisFormat(tFmtDesc.pixelformat))

105 {

106 ptVideoDevice->iPixelFormat = tFmtDesc.pixelformat;

107 break;

108 }

109 tFmtDesc.index++;

110 }

### 7.4.4 设置图像帧格式

设置图像格式需要用到struct v4l2\_format结构体，该结构体描述每帧图像的具体格式，包括帧类型以及图像的长、宽等信息。

01 struct v4l2\_format

02 {

03 enum v4l2\_buf\_type type;          // 帧类型，应用程序设置

04 union fmt

05 {

06 struct v4l2\_pix\_format pix;   // 视频设备使用

07 structv 4l2\_window win;

08 struct v4l2\_vbi\_format vbi;

09 struct v4l2\_sliced\_vbi\_format sliced;

10 \_\_u8 raw\_data[200];

11 };

12 };

struct v4l2\_format结构体需要设置enum v4l2\_buf\_type type和union fmt中的struct v4l2\_pix\_format pix。enum v4l2\_buf\_type type因为使用的是camera设备，camera是CAPTURE设备，所以设置成V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE。struct v4l2\_pix\_format pix设置一帧图像的长、宽和格式等，由于要适配LCD输出，所以长、宽设置为LCD支持的长、宽，如124~125行所示。

119 /\* set format in \*/

120 GetDispResolution(&iLcdWidth, &iLcdHeigt, &iLcdBpp);

121 memset(&tV4l2Fmt, 0, sizeof(struct v4l2\_format));

122 tV4l2Fmt.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

123 tV4l2Fmt.fmt.pix.pixelformat = ptVideoDevice->iPixelFormat;

124 tV4l2Fmt.fmt.pix.width = iLcdWidth;

125 tV4l2Fmt.fmt.pix.height = iLcdHeigt;

126 tV4l2Fmt.fmt.pix.field = V4L2\_FIELD\_ANY;

127

128 /\* 如果驱动程序发现无法某些参数(比如分辨率),

129 \* 它会调整这些参数, 并且返回给应用程序

130 \*/

131 iError = ioctl(iFd, VIDIOC\_S\_FMT, &tV4l2Fmt);

132 if (iError)

133 {

134 DBG\_PRINTF("Unable to set format\n");

135 goto err\_exit;

136 }

### 7.4.5 申请缓冲区

相关结构体如下，该结构体描述申请的缓冲区的基本信息。

01 struct v4l2\_requestbuffers

02 {

03 \_\_u32 count; // 缓冲区内缓冲帧的数目

04 enum v4l2\_buf\_type type; // 缓冲帧数据格式

05 enum v4l2\_memorymemory; // 区别是内存映射还是用户指针方式

06 \_\_u32 reserved[2];

07 };

申请一个拥有四个缓冲帧的缓冲区，\_\_u32 count为缓冲帧的数目；enum v4l2\_buf\_type type和前文一样，同样设置成V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE；enum v4l2\_memorymemory用来区分是内存映射还是用户指针，我们使用内存映射的方式，所以设置成V4L2\_MEMORY\_MMAP。

140 /\* request buffers \*/

141 memset(&tV4l2ReqBuffs, 0, sizeof(struct v4l2\_requestbuffers));

142 tV4l2ReqBuffs.count = NB\_BUFFER;

143 tV4l2ReqBuffs.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

144 tV4l2ReqBuffs.memory = V4L2\_MEMORY\_MMAP;

145

146 iError = ioctl(iFd, VIDIOC\_REQBUFS, &tV4l2ReqBuffs);

147 if (iError)

148 {

149 DBG\_PRINTF("Unable to allocate buffers.\n");

150 goto err\_exit;

151 }

### 7.4.6 将申请的缓冲帧从内核空间映射到用户空间

相关结构体如下，该结构体表示一帧图像数据的基本信息，包含序号、缓冲帧长度和缓冲帧地址等信息。

01 struct v4l2\_buffer

02 {

03 \_\_u32 index; //buffer 序号

04 enum v4l2\_buf\_type type; //buffer 类型

05 \_\_u32 byteused; //buffer 中已使用的字节数

06 \_\_u32 flags; // 区分是MMAP 还是USERPTR

07 enum v4l2\_field field;

08 struct timeval timestamp; // 获取第一个字节时的系统时间

09 struct v4l2\_timecode timecode;

10 \_\_u32 sequence; // 队列中的序号

11 enum v4l2\_memory memory; //IO 方式，被应用程序设置

12 union m

13 {

14 \_\_u32 offset; // 缓冲帧地址，只对MMAP 有效

15 unsigned long userptr;

16 };

17 \_\_u32 length; // 缓冲帧长度

18 \_\_u32 input;

19 \_\_u32 reserved;

20 };

将内核空间的帧缓冲映射到用户空间，需要两个数据接收帧缓冲的长度和地址，我们需要自己定义一个结构体，该结构体位于video2lcd/include/video\_manager.h文件中，其中iVideoBufMaxLen接收帧缓冲的长度，pucVideBuf接收帧缓冲地址。

16 struct VideoDevice {

17 int iFd;

18 int iPixelFormat;

19 int iWidth;

20 int iHeight;

21

22 int iVideoBufCnt;

23 int iVideoBufMaxLen;

24 int iVideoBufCurIndex;

25 unsigned char \*pucVideBuf[NB\_BUFFER];

26

27 /\* 函数 \*/

28 PT\_VideoOpr ptOPr;

29 };

以下代码使用VIDIOC\_QUERYBUF命令和mmap函数将内核空间的缓冲区映射到用户空间。VIDIOC\_QUERYBUF命令的使用需要参数struct v4l2\_buffer结构体，结构体中的type、memory和index参数需要设置，type和memory和前文中的设置一样，分别设置成V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE和 V4L2\_MEMORY\_MMAP，index参数表示申请的缓冲帧的标号，从0开始，包含申请的所有缓冲帧。

mmap函数原形为：

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

参数具体的含义：

1. addr：映射起始地址，一般为NULL，让内核自动选择；
2. length：被映射内存块的长度；
3. prot：标志映射后能否被读写，其值为PROT\_EXEC,PROT\_READ,PROT\_WRITE,PROT\_NONE；
4. flags：确定此内存映射能否被其他进程共享，可设置为MAP\_SHARED或MAP\_PRIVATE；
5. fd：设备文件句柄；
6. offset：确定映射后的内存地址

156 /\* map the buffers \*/

157 for (i = 0; i < ptVideoDevice->iVideoBufCnt; i++)

158 {

159 memset(&tV4l2Buf, 0, sizeof(struct v4l2\_buffer));

160 tV4l2Buf.index = i;

161 tV4l2Buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

162 tV4l2Buf.memory = V4L2\_MEMORY\_MMAP;

163 iError = ioctl(iFd, VIDIOC\_QUERYBUF, &tV4l2Buf);

164 if (iError)

165 {

166 DBG\_PRINTF("Unable to query buffer.\n");

167 goto err\_exit;

168 }

169

170 ptVideoDevice->iVideoBufMaxLen = tV4l2Buf.length;

171 ptVideoDevice->pucVideBuf[i] = mmap(0 /\* start anywhere \*/ ,

172 tV4l2Buf.length, PROT\_READ, MAP\_SHARED, iFd,

173 tV4l2Buf.m.offset);

174 if (ptVideoDevice->pucVideBuf[i] == MAP\_FAILED)

175 {

176 DBG\_PRINTF("Unable to map buffer\n");

177 goto err\_exit;

178 }

179 }

### 7.4.7 将申请的缓冲帧放入队列，并启动数据流

184~194行代码为使用VIDIOC\_QBUF命令，将申请的缓冲帧依次放入缓冲帧输入队列，等待被图像采集设备依次填满；

181 /\* Queue the buffers. \*/

182 for (i = 0; i < ptVideoDevice->iVideoBufCnt; i++)

183 {

184 memset(&tV4l2Buf, 0, sizeof(struct v4l2\_buffer));

185 tV4l2Buf.index = i;

186 tV4l2Buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

187 tV4l2Buf.memory = V4L2\_MEMORY\_MMAP;

188 iError = ioctl(iFd, VIDIOC\_QBUF, &tV4l2Buf);

189 if (iError)

190 {

191 DBG\_PRINTF("Unable to queue buffer.\n");

192 goto err\_exit;

193 }

194 }

### 7.4.8 启动捕捉图像数据

启动捕捉图像数据使用VIDIOC\_STREAMON命令，当该命令执行成功后，便可以等待图像数据的到来。

356 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

357 \* 函数名称：V4l2StartDevice

358 \* 功能描述：开始捕捉图像数据

359 \* 输入参数：ptVideoDevice

360 \* 输出参数：无

361 \* 返 回 值：无

362 \* 修改日期 版本号 修改人 修改内容

363 \* -----------------------------------------------

364 \* 2020/02/16 V1.0 zhenhua 创建

365 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

366 static int V4l2StartDevice(PT\_VideoDevice ptVideoDevice)

367 {

368 int iType = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

369 int iError;

370

371 iError = ioctl(ptVideoDevice->iFd, VIDIOC\_STREAMON, &iType);

372 if (iError)

373 {

374 DBG\_PRINTF("Unable to start capture.\n");

375 return -1;

376 }

377 return 0;

378 }

### 7.4.9 出列采集的帧缓冲，并处理图像数据，然后再将数据帧入列

我们可以使用VIDIOC\_DQBUF命令，等待缓冲帧的到来，当有缓冲帧被放入视频输出缓冲队列，我们便可以采到一帧图像。接收到图像我们可以对图像进行操作，例如保存、压缩或者LCD输出等。

243 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

244 \* 函数名称：V4l2GetFrameForStreaming

245 \* 功能描述：从图像数据流中获取一帧图像数据

246 \* 输入参数：ptVideoDevice

247 ptVideoBuf

248 \* 输出参数：无

249 \* 返 回 值：无

250 \* 修改日期 版本号 修改人 修改内容

251 \* -----------------------------------------------

252 \* 2020/02/16 V1.0 zhenhua 创建

253 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

254 static int V4l2GetFrameForStreaming(PT\_VideoDevice ptVideoDevice, PT\_VideoBuf ptVideoBuf)

255 {

256 struct pollfd tFds[1];

257 int iRet;

258 struct v4l2\_buffer tV4l2Buf;

259

260 /\* poll \*/

261 tFds[0].fd = ptVideoDevice->iFd;

262 tFds[0].events = POLLIN;

263

264 iRet = poll(tFds, 1, -1);

265 if (iRet <= 0)

266 {

267 DBG\_PRINTF("poll error!\n");

268 return -1;

269 }

270

271 /\* VIDIOC\_DQBUF \*/

272 memset(&tV4l2Buf, 0, sizeof(struct v4l2\_buffer));

273 tV4l2Buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

274 tV4l2Buf.memory = V4L2\_MEMORY\_MMAP;

275 iRet = ioctl(ptVideoDevice->iFd, VIDIOC\_DQBUF, &tV4l2Buf);

276 if (iRet < 0)

277 {

278 DBG\_PRINTF("Unable to dequeue buffer.\n");

279 return -1;

280 }

281 ptVideoDevice->iVideoBufCurIndex = tV4l2Buf.index;

282

283 ptVideoBuf->iPixelFormat = ptVideoDevice->iPixelFormat;

284 ptVideoBuf->tPixelDatas.iWidth = ptVideoDevice->iWidth;

285 ptVideoBuf->tPixelDatas.iHeight = ptVideoDevice->iHeight;

286 ptVideoBuf->tPixelDatas.iBpp = (ptVideoDevice->iPixelFormat == V4L2\_PIX\_FMT\_YUYV) ? 16 : \

287 (ptVideoDevice->iPixelFormat == V4L2\_PIX\_FMT\_MJPEG) ? 0 : \

288 (ptVideoDevice->iPixelFormat == V4L2\_PIX\_FMT\_RGB565) ? 16 : \

289 0;

290 ptVideoBuf->tPixelDatas.iLineBytes = ptVideoDevice->iWidth \* ptVideoBuf->tPixelDatas.iBpp / 8;

291 ptVideoBuf->tPixelDatas.iTotalBytes = tV4l2Buf.bytesused;

292 ptVideoBuf->tPixelDatas.aucPixelDatas = ptVideoDevice->pucVideBuf[tV4l2Buf.index];

293 return 0;

294 }

当我们从缓冲帧输出队列取出一个缓冲帧，取出图像数据后我们需要将缓冲帧重新放回到视频输入缓冲队列，该操作还是使用VIDIOC\_QBUF命令，放回缓冲帧输入队列后继续等待被填满。

296 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

297 \* 函数名称：V4l2PutFrameForStreaming

298 \* 功能描述：将取出的帧缓冲重新放回图像输入队列

299 \* 输入参数：ptVideoDevice

300 ptVideoBuf

301 \* 输出参数：无

302 \* 返 回 值：无

303 \* 修改日期 版本号 修改人 修改内容

304 \* -----------------------------------------------

305 \* 2020/02/16 V1.0 zhenhua 创建

306 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

307 static int V4l2PutFrameForStreaming(PT\_VideoDevice ptVideoDevice, PT\_VideoBuf ptVideoBuf)

308 {

309 /\* VIDIOC\_QBUF \*/

310 struct v4l2\_buffer tV4l2Buf;

311 int iError;

312

313 memset(&tV4l2Buf, 0, sizeof(struct v4l2\_buffer));

314 tV4l2Buf.index = ptVideoDevice->iVideoBufCurIndex;

315 tV4l2Buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

316 tV4l2Buf.memory = V4L2\_MEMORY\_MMAP;

317 iError = ioctl(ptVideoDevice->iFd, VIDIOC\_QBUF, &tV4l2Buf);

318 if (iError)

319 {

320 DBG\_PRINTF("Unable to queue buffer.\n");

321 return -1;

322 }

323 return 0;

324 }

### 7.4.10 停止捕捉图像数据

停止采集图像数据，首先使用VIDIOC\_STREAMOFF命令，关闭捕获图像数据。同时要注意取消内存映射和关闭句柄，防止不必要的内存泄漏。代码390~407行为停止捕捉图像数据命令；代码227~241行为取消内存映射和关闭句柄。

380 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

381 \* 函数名称：V4l2StopDevice

382 \* 功能描述：停止捕捉图像数据

383 \* 输入参数：ptVideoDevice

384 \* 输出参数：无

385 \* 返 回 值：无

386 \* 修改日期 版本号 修改人 修改内容

387 \* -----------------------------------------------

388 \* 2020/02/16 V1.0 zhenhua 创建

389 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

390 static int V4l2StopDevice(PT\_VideoDevice ptVideoDevice)

391 {

392 int iType = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

393 int iError;

394

395 iError = ioctl(ptVideoDevice->iFd, VIDIOC\_STREAMOFF, &iType);

396 if (iError)

397 {

398 DBG\_PRINTF("Unable to stop capture.\n");

399 return -1;

400 }

401 return 0;

402 }

403

404 static int V4l2GetFormat(PT\_VideoDevice ptVideoDevice)

405 {

406 return ptVideoDevice->iPixelFormat;

407 }

217 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

218 \* 函数名称：V4l2ExitDevice

219 \* 功能描述：退出采集设备，取消帧缓冲映射和关闭句柄

220 \* 输入参数：ptVideoDevice

221 \* 输出参数：无

222 \* 返 回 值：无

223 \* 修改日期 版本号 修改人 修改内容

224 \* -----------------------------------------------

225 \* 2020/02/16 V1.0 zhenhua 创建

226 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

227 static int V4l2ExitDevice(PT\_VideoDevice ptVideoDevice)

228 {

229 int i;

230 for (i = 0; i < ptVideoDevice->iVideoBufCnt; i++)

231 {

232 if (ptVideoDevice->pucVideBuf[i])

233 {

234 munmap(ptVideoDevice->pucVideBuf[i], ptVideoDevice->iVideoBufMaxLen);

235 ptVideoDevice->pucVideBuf[i] = NULL;

236 }

237 }

238

239 close(ptVideoDevice->iFd);

240 return 0;

241 }