V4L2 API 概述

www.linuxtv.org 下，有篇文档详细讲解了 V4L2 相关知识和体系结构。是 V4L2 方面最全面的文档。可以通过它学习 V4L2 的一些思路和想法。

http://www.linuxtv.org/downloads/v4l-dvb-apis/index.html

文档包含的内容主要是 Linux Kernel 对 用户空间使用者提供的 Video 和 Audio 流 Device. 包括 video Cameras，模拟/数字电视接收器卡，AM/FM 接受卡，流捕捉 Device。

对 V4L2 Device 编程，通常包含以下步骤：

1. 打开 Device。
2. 改变 devcie 的特性，选择 video 或者 audio 输入，video 标准，图像亮度等等。
3. 协商数据格式。
4. 协商输入/输入的方法。
5. 真实的输入/输出 Loop。
6. 关闭 Device。

0.打开和关闭设备：

0.1: 设备名：

V4L2 driver 以 Kernel Modules 形态实现，它在被 root 用户手动 insmod 或者在设备首次被插入时被载入。driver 模块将被挂载在 videodev 模块下。

每个 driver 在载入是都会注册一个或多个主设备号为 81，此设备号为 0-255 的 device node. 如何分配次设备号，完全取决于 root 用户(SamInfo：其实不指定的话，系统也会自动分配没有被占用的次设备号)。这个方式的本意是为了解决 device 冲突。

模块的参数可以选择次设备号,此以设备名为前缀的的_nr 中。例如：video_0 表明/dev/video（0+Base）.
其中,video 的 base 为 0。radio 的 base 则为 64。
#insmod mydriver.ko video_nr=0,1 radio_nr=0,1

1. 关联设备：

Device 可以支持几个相关的功能，例如 video Capture，video overlay （注 1）和 VBI capturing 相关联，因为其功能共享。video 输入设备和 tuner 也类似。V4L 和早期版本的 V4L2 对 Video Capture 和 Video Overlay 采用同样的命名方式和同样范围的子设备号，但 VBI 设备则不同。这种方式有很多问题，更严重的是，V4L videodev 禁止多重打开。

2. 设备的多次打开：

通常,V4L2 设备可以被打开多次, 当驱动支持这个功能时,用户可以在其它应用程序捕捉 Video 或者 Audio 时,同时打开设备调节亮度或者音量.

多重打开功能是可选的,Driver 至少应该支持用户在没有数据访问的情况下并发打开.

因为可以多重打开,所以理论上就应该可以设置优先级别(VIDIOC_G_PRIORITY 和 VIDIOC_S_PRIORITY),优先级高的应用程序可以成功修改一些设置，优先级低的设备去修改时，则会报失败 EBUSY。

3. 共享数据流(Shared Data Streams)

V4L2 driver 不支持多个应用程序通过 copy buffer 读写 Device 的同一个数据流。非要这么干，可以使用在用户空间的代理程序。如果驱动支持流共享，那么其实现必须是透明的。V4L2 API 并没有列出产生冲突时要如何来解决。

4. 能力查询：（Querying Capabilites）

V4L2 包含很宽广的使用范围。 所以首先需要查询其设备能力集。

通常，使用 ioctl VIDIOC_QUERYCAP 来查询当前 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，可以通过这个 ioctl 是否成功来判断当前设备和 dirver 是否支持 V4L2 规范。当然，这样同时还能够得到设备足够的能力信息。

```
struct v4l2_capability
{
__u8 driver[16]; //驱动名。
__u8 card[32];  // Device 名
__u8 bus_info[32];  //在 Bus 系统中存放位置
__u32 version;  //driver 版本
__u32 capabilities;  //能力集
__u32 reserved[4];
};
```

能力集中包含：

V4L2_CAP_VIDEO_CAPTURE	**0x00000001**	The device supports the Video Capture interface.
V4L2_CAP_VIDEO_OUTPUT	0x00000002	The device supports the Video Output interface.
V4L2_CAP_VIDEO_OVERLAY	0x00000004	The device supports the Video Overlay interface. A video overlay device typically stores captured images directly in the video memory of a graphics card,with hardware clipping and scaling.
V4L2_CAP_VBI_CAPTURE	0x00000010	The device supports the Raw VBI Capture interface, providing Teletext and Closed Caption data.
V4L2_CAP_VBI_OUTPUT	0x00000020	The device supports the Raw VBI Output interface.
V4L2_CAP_SLICED_VBI_CAPTURE	0x00000040	The device supports the Sliced VBI Capture interface.
V4L2_CAP_SLICED_VBI_OUTPUT	0x00000080	The device supports the Sliced VBI Output interface.
V4L2_CAP_RDS_CAPTURE	0x00000100	[to be defined]

```
#define V4L2_CAP_TUNER 0x00010000
#define V4L2_CAP_AUDIO 0x00020000
#define V4L2_CAP_RADIO 0x00040000

#define V4L2_CAP_READWRITE 0x01000000
#define V4L2_CAP_ASYNCIO 0x02000000
#define V4L2_CAP_STREAMING 0x04000000
```

看起来很熟悉吧，其实就是 Driver 里面的 Type。

5.格式查选和设置：
这里的格式,其实是指抓到的每一帧数据的相关格式.例如: 长,宽,像素,field 等。
5.1：得到当前格式：
VIDIOC_G_FMT。
ioctl(Handle, VIDIOC_G_FMT, &Format)
参数三：
与很多 ioctl 类似， 它是个 in/out 参数。

```
struct v4l2_format
{
enum v4l2_buf_type type;
union
{
struct v4l2_pix_format pix;
struct v4l2_window win;
struct v4l2_vbi_format vbi;
struct v4l2_sliced_vbi_format sliced;
__u8 raw_data[200];
} fmt;
};
```

v4l2_buf_type type; 输入信息，让用户选择是哪种类型的设备。这里又与 Driver 中对应起来了。

```
struct v4l2_pix_format pix;
struct v4l2_pix_format
{
__u32 width;    //抓取桢的宽度
__u32 height;  //桢高度
__u32 pixelformat;  //像素格式。例如：V4L2_PIX_FMT_YUYV，V4L2_PIX_FMT_RGB332 等。
enum v4l2_field field; //image 包含逐行数据还是隔行数据。如果是隔行数据，是奇行还是偶行
__u32 bytesperline; //每行多少字节，通过 width,height,pixelformat 可以算出
__u32 sizeimage;    //每桢多少字节。也可以算出。但需要加入 field 信息才能算出
enum v4l2_colorspace colorspace;
__u32 priv;
};
```

好像首次取 **sizeimage**，可能会取到不正确的值。

5.2：设置 Format:
VIDIOC_S_FMT:
io_rel = ioctl(Handle, VIDIOC_S_FMT, &Format);
在设置之前，需要填写参数 3Format 内容.
例如;
Format.fmt.pix.width = Width;
Format.fmt.pix.height = Height;

Format.fmt.pix.pixelformat= pixelformat;//V4L2_PIX_FMT_YUYV;
Format.fmt.pix.field = V4L2_FIELD_INTERLACED;

6. Streaming 信息得到和设置：
前面 Format 是单帧数据的设置。现在咱们需要设置和流有关的信息。如桢数。
6.1：得到当前 Stream 信息：
利用 ioctl:
memset(&Stream_Parm, 0, sizeof(struct v4l2_streamparm));
Stream_Parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

io_rel = ioctl(Handle, VIDIOC_G_PARM, &Stream_Parm);
与之前类似，关键还是在参数三：
struct v4l2_streamparm
{
enum v4l2_buf_type type;
union
{
struct v4l2_captureparm capture;
struct v4l2_outputparm output;
__u8 raw_data[200];
} parm;
};

type 是个 in/out 参数，咱们是 Camera 捕捉设备，所以选用 V4L2_BUF_TYPE_VIDEO_CAPTURE。

struct v4l2_captureparm capture;
struct v4l2_captureparm
{
__u32 capability; //功能标签。目前为止已经定义的只有一个 **V4L2_CAP_TIMEPERFRAME，**它代表可以改变帧频率
__u32 capturemode; //一个标签的字段:**V4L2_MODE_HIGHQUALITY,**这个标签意在使硬件在高清模式下工作，实现单帧的捕获。这个模式可以做出任何的牺牲（包括支持的格式，曝光时间等），以达到设备可以处理的最佳图片质量。
struct v4l2_fract timeperframe;
__u32 extendedmode;//它在 API 中没有明确的意义
__u32 readbuffers;//read()操作被调用时内核应为输入的帧准备的缓冲区数量
__u32 reserved[4];
};

调节的关键在 struct v4l2_fract timeperframe;
struct v4l2_fract {
__u32 numerator; //FPS 的分子
__u32 denominator; // FPS 的分母
};

例如：numerator=1, denominator= 30. 则表明每秒 30 帧。

6.2：设置 Stream Setting：

```
memset(&Stream_Parm, 0, sizeof(struct v4l2_streamparm));
Stream_Parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

Stream_Parm.parm.capture.timeperframe.denominator =Denominator;;
Stream_Parm.parm.capture.timeperframe.numerator = Numerator;

io_rel = ioctl(Handle, VIDIOC_S_PARM, &Stream_Parm);
```

注 1：
Video Capture(视频捕捉)和 Video overlay 的区别：

vieo **overlay** 不同于 video capture，是指不需要对 video 信号的帧进行 copy，直接将视频信号转化成显卡的 VGA 信号或者将捕获到的视频帧直接存放在显卡的内存中。

二

Camera 的可设置项极多，V4L2 支持了不少。但 Sam 之前对这些设置的用法和涵义都是在看 videodev2.h 中边看边理解，感觉非常生涩。直到写这篇 blog 时，才发现 v4l2 有专门的 SPEC 来说明：
http://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single/v4l2.html

但也基本没有时间仔细看了。先把自己看头文件看出的一些东西记录在这里吧。

以实际设置过程为顺序谈谈 V4L2 设置。

1. 查询 V4L2 功能集：VIDIOC_QUERYCAP

```
struct v4l2_capability cap;
int rel = 0;
ioctl(Handle, VIDIOC_QUERYCAP, &cap);
```

使用 ioctl VIDIOC_QUERYCAP 来查询当前 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，可以通过这个 ioctl 是否成功来判断当前设备和 dirver 是否支持 V4L2 规范。当然，这样同时还能够得到设备足够的能力信息。

```
struct v4l2_capability
{
 __u8 driver[16];   //驱动名。
 __u8 card[32];     // Device 名
 __u8 bus_info[32];  //在 Bus 系统中存放位置
 __u32 version;      //driver 版本
```

```
 __u32 capabilities;  //能力集
 __u32 reserved[4];
};
```

能力集中包含：

V4L2_CAP_VIDEO_CAPTURE 0x00000001 The device supports the Video Capture interface.
V4L2_CAP_VIDEO_OUTPUT 0x00000002 The device supports the Video Output interface.
V4L2_CAP_VIDEO_OVERLAY 0x00000004 The device supports the Video Overlay interface.
A video overlay device typically stores captured images directly in the video memory of a graphics card,with hardware clipping and scaling.
V4L2_CAP_VBI_CAPTURE 0x00000010 The device supports the Raw VBI Capture interface, providing Teletext and Closed Caption data.
V4L2_CAP_VBI_OUTPUT 0x00000020 The device supports the Raw VBI Output interface.
V4L2_CAP_SLICED_VBI_CAPTURE 0x00000040 The device supports the Sliced VBI Capture interface.
V4L2_CAP_SLICED_VBI_OUTPUT 0x00000080 The device supports the Sliced VBI Output interface.
V4L2_CAP_RDS_CAPTURE 0x00000100 [to be defined]

```
#define V4L2_CAP_TUNER 0x00010000
#define V4L2_CAP_AUDIO 0x00020000
#define V4L2_CAP_RADIO 0x00040000

#define V4L2_CAP_READWRITE 0x01000000
#define V4L2_CAP_ASYNCIO 0x02000000
#define V4L2_CAP_STREAMING 0x04000000
```

看起来很熟悉吧，其实就是 Driver 里面的 Type。

```
 __u8 driver[16]; driver 名，通常为：uvcvideo
 __u8 card[32];  设备名：厂商会填写。
 __u8 bus_info[32];  bus,通常为：usb-hiusb-ehci-2.4
 __u32 version;
 __u32 capabilities;  通常为：V4L2_CAP_VIDEO_CAPTURE | V4L2_CAP_STREAMING
 __u32 reserved[4];
```

2. 枚举设备所支持的 image format: VIDIOC_ENUM_FMT

```
struct v4l2_fmtdesc fmtdesc;
fmtdesc.index = 0;
fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(Handle, VIDIOC_ENUM_FMT, &fmtdesc);
```

使用 ioctl VIDIOC_ENUM_FMT 依次询问，type 为：V4L2_BUF_TYPE_VIDEO_CAPTURE。 index 从 0 开始，依次增加，直到返回. Driver 会填充结构体 struct v4l2_fmtdesc 的其它内容，如果 index 超出范围，则返回-1。

```
struct v4l2_fmtdesc
{
 __u32 index;                    // 需要填充，从 0 开始，依次上升。
 enum v4l2_buf_type type;      //Camera，则填写 V4L2_BUF_TYPE_VIDEO_CAPTURE
```

```
 __u32 flags;                // 如果压缩的，则 Driver 填写：V4L2_FMT_FLAG_COMPRESSED，否则为 0
 __u8 description[32];       // image format 的描述，如：YUV 4:2:2 (YUYV)
 __u32 pixelformat;       //所支持的格式。 如：V4L2_PIX_FMT_UYVY
 __u32 reserved[4];
};
```

这样，则知道当前硬件支持什么样的 image format. 下一步，则可以设置 image 了。当然，设置之前，还可以读取当前缺省设置。

3. 得到和设置 Image Format: VIDIOC_G_FMT, VIDIOC_S_FMT:

3.1: 得到当前 **Image Format:**

```
struct v4l2_format Format;
memset(&Format, 0, sizeof(struct v4l2_format));
Format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ioctl(Handle, VIDIOC_G_FMT, &Format);
```

利用 ioctl VIDIOC_G_FMT. 得到当前设置。
因为 Camera 为 CAPTURE 设备，所以需要设置 type 为： V4L2_BUF_TYPE_VIDEO_CAPTURE
然后 Driver 会填充其它内容。

```
struct v4l2_format
{
 enum v4l2_buf_type type;  // Camera，则用户必须填写：V4L2_BUF_TYPE_VIDEO_CAPTURE
 union
 {
 struct v4l2_pix_format pix;     // used by video capture
                                 // and output devices
 struct v4l2_window win;
 struct v4l2_vbi_format vbi;
 struct v4l2_sliced_vbi_format sliced;
 __u8 raw_data[200];
 } fmt;
};
```

因为是 Camera， 所以采用 pix. 现在分析如下：

```
struct v4l2_pix_format
{
 __u32 width;   //Image width in pixels.
 __u32 height;  // Image Height in pixels.
 __u32 pixelformat;  // Image 格式，最常见的有：V4L2_PIX_FMT_YYUV
 enum v4l2_field field; //是否逐行扫描，是否隔行扫描. Sam 通常采用 V4L2_FIELD_NONE, 逐行放置数据 (注 1)
 __u32 bytesperline; //每行的 byte 数
 __u32 sizeimage;    //总共的 byte 数，bytesperline * height
```

```
 enum v4l2_colorspace colorspace;  //This information supplements
the pixelformat and must be set by the driver
 __u32 priv;
};
```

3.2：设置 Image Format:VIDIOC_S_FMT

之前通过 **VIDIOC_ENUM_FMT** 已经知道 Device 支持什么 Format。 所以就不用猜测了，直接设置吧。

设置 Image Format ,利用 iocto VIDIOC_S_FMT.

需要 APPLICATION 填写的 struct 项目有：

```
struct v4l2_format Format;
Format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
Format.fmt.pix.width = Width;
Format.fmt.pix.height = Height;
Format.fmt.pix.pixelformat= pixelformat; //V4L2_PIX_FMT_YUYV;
Format.fmt.pix.field = field;
io_rel = ioctl(Handle, VIDIOC_S_FMT, &Format);
```

SamInfo：之前设置了 Image Format， 是指每一帧的数据格式，但 Stream 的行为呢，也需要设置，这就是下面所说的 Stream 设置了。它就包含帧数设置和修改。

4. 得到和设置 Stream 信息：VIDIOC_G_PARM，VIDIOC_S_PARM

Stream 信息，主要是设置帧数。

4.1：得到 Stream 信息：

```
struct v4l2_streamparm Stream_Parm;
memset(&Stream_Parm, 0, sizeof(struct v4l2_streamparm));
Stream_Parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
io_rel = ioctl(Handle, VIDIOC_G_PARM, &Stream_Parm);
```

用户只需要填充 type 为 V4L2_BUF_TYPE_VIDEO_CAPTURE。 Driver 就会把结构体中其它部分填充好。

```
struct v4l2_streamparm
{
 enum v4l2_buf_type type;
 union
 {
 struct v4l2_captureparm capture;
 struct v4l2_outputparm output;
 __u8 raw_data[200];
 } parm;
};
```

因为是 Camera， 所以使用 capture. 它是 **struct v4l2_captureparm**

```
struct v4l2_captureparm
{
 __u32 capability;   //是否可以被 timeperframe 控制帧数。可以则：V4L2_CAP_TIMEPERFRAME
 __u32 capturemode;  // 是否为高清模式。如果是：则设置为：V4L2_MODE_HIGHQUALITY。
                     //  高清模式会牺牲其它信息。通常设置为 0。
struct v4l2_fract timeperframe;  //帧数。
 __u32 extendedmode;  //定制的。如果不支持，设置为 0
 __u32 readbuffers;
 __u32 reserved[4];
};

 struct v4l2_fract timeperframe; //帧数。
struct v4l2_fract {
 __u32 numerator;  // 分子。 例：1
 __u32 denominator; //分母。 例：30
};
```

4.2：设置帧数：

```
struct v4l2_streamparm Stream_Parm;
memset(&Stream_Parm, 0, sizeof(struct v4l2_streamparm));
Stream_Parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
Stream_Parm.parm.capture.timeperframe.denominator =Denominator;;
Stream_Parm.parm.capture.timeperframe.numerator = Numerator;
io_rel = ioctl(Handle, VIDIOC_S_PARM, &Stream_Parm);
```

请注意，哪怕 ioctl 返回 0。也有可能没设置成功。所以需要再次 Get。
当然，哪怕 Get 发现设置成功。真正抓帧也可能没那么高。

5. 利用 VIDIOC_G_CTRL 得到一些设置：

一些具体的设置，如曝光模式(Exposure Type)，曝光值(Exposure)，增益(Gain),白平衡(WHITE_BALANCE),亮度(BRIGHTNESS)，饱和度(SATURATION)，对比度(CONTRAST)等信息。可以通过 VIDIOC_G_CTRL 得到当前值。

用法：**APP** 填写结构体中的 **id.** 通过调用 **VIDIOC_G_CTRL**，**driver** 会填写结构体中 **value** 项。

```
struct v4l2_control ctrl;
struct v4l2_control
{
 __u32 id;
 __s32 value;
};
```

以曝光模式，曝光，和增益为例;

曝光模式：

```
struct v4l2_control ctrl;
```

ctrl.id = V4L2_CID_EXPOSURE_AUTO;
ret = ioctl(Handle, VIDIOC_G_CTRL, &ctrl);
ctrl.value 则由 Driver 填写。告知当前曝光模式。
有以下几个选择：
enum v4l2_exposure_auto_type {
V4L2_EXPOSURE_AUTO = 0,
V4L2_EXPOSURE_MANUAL = 1,
V4L2_EXPOSURE_SHUTTER_PRIORITY = 2,
V4L2_EXPOSURE_APERTURE_PRIORITY = 3
};

曝光：
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_EXPOSURE_ABSOLUTE;
ret = ioctl(Handle, VIDIOC_G_CTRL, &ctrl);
同样，driver 填写 ctrl.value. 内容为曝光值。

增益：
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_GAIN;
ret = ioctl(Handle, VIDIOC_G_CTRL, &ctrl);
同样，driver 填写 ctrl.value. 内容为增益。

6. 利用 VIDIOC_QUERYCTRL 得到设置具体信息：
在很多情况下，我们并不知道如何设置一些信息，例如，曝光应该设置为多少？Driver 能够接受的范围是多少？最大，最小值是多少？步长是多少？缺省值为多少？
可以通过 VIDIOC_QUERYCTRL 得到。

咱们还是以增益为例：
struct v4l2_queryctrl Setting;
Setting.id = V4L2_CID_GAIN;
ret = ioctl(Handle, VIDIOC_QUERYCTRL, &Setting);
Driver 就会填写结构体中所有信息。

struct v4l2_queryctrl
{
 __u32 id; //用户设置。指定查找的是哪个 ID。
 enum v4l2_ctrl_type type;
 __u8 name[32]; //ID 对应的名字。
 __s32 minimum;
 __s32 maximum;
 __s32 step;   //步长
 __s32 default_value;

```
 __u32 flags;
 __u32 reserved[2];
};
```
这样，就知道设置什么值是合法的了。那么，下一步就是设置了。

7. 利用 VIDIOC_S_CTRL 来设置：

很简单，设置 id 和 value.调用 ioctl 就好。

还是以增益为例：

```
struct v4l2_control ctrl;
ctrl.id = V4L2_CID_GAIN;
ctrl.value = Gain;
ret = ioctl(Handle, VIDIOC_S_CTRL, &ctrl);
```

有时候，硬件设置很奇怪，可以设置某个信息，却无法得到如何设置的信息。例如：**HD-500** 可以设置增益。却无法得到该如何设置。

8. 利用扩展 Ctrl 设置:

焦距(FOUCE);

三

前面主要介绍的是：V4L2 的一些设置接口，如亮度，饱和度，曝光时间，帧数，增益，白平衡等。今天看看 V4L2 得到数据的几个关键 ioctl，Buffer 的申请和数据的抓取。

1. 初始化 Memory Mapping 或 User Pointer I/O.

int ioctl(int fd, int requestbuf, struct v4l2_requestbuffers * argp);

参数一：open（）所产生的句柄。

参数二：**VIDIOC_REQBUFS**

参数三：in/out 结构体。

```
struct v4l2_requestbuffers
{
 __u32 count;
 enum v4l2_buf_type type;
 enum v4l2_memory memory; //Applications set this field
to V4L2_MEMORY_MMAP or V4L2_MEMORY_USERPTR
 __u32 reserved[2];
};
```

注意，有两种方式的 I/O。 Memory Mapping 和 User Pointer。

Memory Mapping 的 Buffer 由 Driver 申请为物理连续的内存空间(Kernel 空间)。 在此 ioctl 调用时被分配，需要早于 mmap()动作将他们映射到用户空间。

1.1：Memory Mapping 模式详解：
在使用 Memory Mapping 模式时，参数三中结构体内每个 field 都需要设置。

__u32 count; //当 memory=V4L2_MEMORY_MMAP 时，此处才有效。表明要申请的 buffer 个数。
enum v4l2_buf_type type; //Stream 或者 Buffer 的类型。此处肯定为 V4L2_BUF_TYPE_VIDEO_CAPTURE
enum v4l2_memory memory; //既然是 Memory Mapping 模式，则此处设置为：V4L2_MEMORY_MMAP

注意：count 是个输入输出函数。 因为你所申请到的 Buffer 个数不一定就是你所输入的 Number。所以在 ioctl 执行后，driver 会将真实申请到的 buffer 个数填充到此 field. 这个数目有可能大于你想要申请的，也可能小与，甚至可能是 0 个。

应用程序可以再次调用 ioctl--VIDIOC_REQBUFS 来修改 buffer 个数。但前提是必须先释放已经 mapped 的 buffer ，可以先 munmap ，然后设置参数 count 为 0 来释放所有的 buffer。

支持 Memory Mapping I/O 方式的前提是：**v4l2_capability** 中支持 V4L2_CAP_STREAMING。
在这个模式下，数据本身不会被 Copy，只是在 Kernel 和用户态之间交换。在应用程序想要访问到这些数据之前，它必须调用 **mmap()**影射到用户态。

同时也要注意，通过 ioctl 申请的内存，是物理内存，无法被交换入 Disk，所以一定要释放：munmap()。

1.2：User Pointer 模式：
User Pointer 模式时，应用程序实现申请。
只需要填充 Type=V4L2_BUF_TYPE_VIDEO_CAPTURE， memory=V4L2_MEMORY_USERPTR

2. 询问 Buffer 状态：

int ioctl(int fd, int request, struct v4l2_buffer* argp);
参数一：open（）所产生的句柄。
参数二：**VIDIOC_QUERYBUF**
参数三：v4l2_buffer 结构体。(IN/OUT 参数)

注意，此 ioctl 是 Memory Mapping 的 I/O 方法之一。User Pointer 模式不需要。在 Buffer 在 ioctl-VIDIOC_REQBUFS 执行时创建后，随时都可以调用此 Ioctl 得到 buffer 信息。

我们首先通过 v4l2_buffer 结构体看看参数三这个输入输出参数需要输入些什么，以及能够得到什么信息。

```
struct v4l2_buffer
{
 __u32 index;
 enum v4l2_buf_type type;
 __u32 bytesused;
```

```
 __u32 flags;
 enum v4l2_field field;
 struct timeval timestamp;
 struct v4l2_timecode timecode;
 __u32 sequence;

 enum v4l2_memory memory;
 union {
 __u32 offset;
 unsigned long userptr;
 } m;
 __u32 length;
 __u32 input;
 __u32 reserved;
};
```

在调用 ioctl--**VIDIOC_QUERYBUF** 时，需要写入的项目有：
enum v4l2_buf_type type; //V4L2_BUF_TYPE_VIDEO_CAPTURE
__u32 index; // 这里需要解释一下，因为在调用 **ioctl-VIDIOC_REQBUFS** 时，建立了 **count** 个 **Buffer**。所以，这里 **index** 的有效范围是：**0** 到 **count-1.**

在调用 ioctl-**VIDIOC_QUERYBUF** 后，Driver 会填充 v4l2_buffer 结构体内所有信息供用户使用。如果一些正常：
1. flags 中：
V4L2_BUF_FLAG_MAPPED, V4L2_BUF_FLAG_QUEUED and V4L2_BUF_FLAG_DONE 被设置。
2. memory 中，**V4L2_MEMORY_MMAP** 被设置。
3．*m.offset* 中， 从将要 **mapping** 的 **device memory** 头到数据头的 **offset.**
4. *length* 中，填充当前 **Buffer** 长度。
5。其它的 **Field** 有可能设置，也有可能不被设置。

这样，**mmap**（ ）想要有的信息就全了。而 **mmap()**之后，**Device Driver** 申请的或者 **Device Memory** 就能映射到用户空间。数据就可以被应用程序使用了。这才是 **ioctl-VIDIOC_QUERYBUF** 的关键作用。

3.和 Driver 交换 buffer:

对 Camera 这样的捕获设备来说，Device 将数据放到 Buffer 中，用户得到数据。Device 再次将数据放到 Buffer 中。那么 Device Driver 怎样知道哪个 Buffer 是可以存放数据的呢？这就用到当前这两个 ioctl-VIDIOC_QBUF, ioctl-VIDIOC_DQBUF.

ioctl-VIDIOC_QBUF: 将指定的 Buffer 放到输入队列中，即向 Device 表明这个 Buffer 可以存放东西。
ioctl-VIDIOC_DQBUF: 将输出队列中的数据 buffer 取出。

在 **driver** 内部管理着两个 **buffer queues** ，一个输入队列，一个输出队列。对于 **capture**

device 来说，当输入队列中的 **buffer** 被塞满数据以后会自动变为输出队列，等待调用 **VIDIOC_DQBUF** 将数据进行处理以后重新调用 **VIDIOC_QBUF** 将 **buffer** 重新放进输入队列**.**

用法：
ioctl--VIDIOC_QBUF：
int ioctl(int fd, int request, struct v4l2_buffer* argp);
参数一：open（）所产生的句柄。
参数二：**VIDIOC_QBUF**
参数三：v4l2_buffer 结构体。(IN/OUT 参数)

参数三是 IN/OUT 参数。需要填充
enum v4l2_buf_type type; //V4L2_BUF_TYPE_VIDEO_CAPTURE
__u32 index; // 这里需要解释一下，因为在调用 **ioctl-VIDIOC_REQBUFS** 时，建立了 **count** 个 **Buffer**。所以，这里 **index** 的有效范围是：**0** 到 **count-1.**
memory: V4L2_MEMORY_MMAP.

则这个结构体指明的 buffer 被送入输出队列，表明此 Buffer 可以被 device 填充数据。

用法：
ioctl--VIDIOC_DQBUF：
int ioctl(int fd, int request, struct v4l2_buffer* argp);
参数一：open（）所产生的句柄。
参数二：**VIDIOC_DQBUF**
参数三：v4l2_buffer 结构体。(IN/OUT 参数)

从输出队列中取出一个有数据的 Buffer。 这个 Buffer 中的数据被处理后，此 Buffer 可以通过 ioctl-**VIDIOC_QBUF** 再次放入输入队列中去。

4. 开始和结束捕获：
ioctl--VIDIOC_STREAMON. ioctl--VIDIOC_STREAMOFF

非常简单的调用。就是开始和结束。