V4L2 摄像头编程基本步骤

作者：尚观 李志勇

注意：红色文字为系统提供的数据结构，蓝色文字为示例代码，代码中用到的所有 ioctl 命令和结构体都在/usr/include/linux/videodev2.h 文件中实现

什么是 **V4L2**?

V4L 是 Video for Linux 的缩写,它是 Linux 内核中关于视频设备的子系统，它为 Linux 下的视频驱动提供了统一的接口，使得应用程序可以使用统一的 API 函数操作不同的视频设备，极大地简化了视频系统的开发和维护。由于早期的 V4L 有很多缺陷，Bill Dirks 等人对其进行了重新设计，并取名为 Video for Linux 2(V4L2)，最早出现于 Linux2.5.x 版本。V4L2 相比于 V4L 有更好的扩展性和灵活性，并且支持的硬件设备更多。由于历史的原因,V4L2 一般兼容 V4L，所以很多程序可以用 V4L 接口。

V4L2 支持的设备有哪些?

1.视频采集接口(video capture interface):这种应用的设备可以是高频头或者摄像头.V4L2 的最初设计就是应用于这种功能的。
2.视频输出接口(video output interface):可以驱动计算机的外围视频图像设备，比如可以输出电视信号格式的设备。
3.直接传输视频接口(video overlay interface):它的主要工作是把从视频采集设备采集过来的信号直接输出到输出设备之上,而不用经过系统的 CPU。
4.视频间隔消隐信号接口(VBI interface):它可以使应用可以访问传输消隐期的视频信号。
5.收音机接口(radio interface):可用来处理从 AM 或 FM 高频头设备接收来的音频流。

V4L2 处于什么位置?

<table>
<tr><td>用户</td><td colspan="3">/dev/video0 /dev/video1 /dev/video2 .....</td></tr>
<tr><td rowspan="2">内核</td><td colspan="3">V4L2 子系统（只限于 Linux 内核）</td></tr>
<tr><td>UVC 驱动</td><td>收音机驱动</td><td>其他 video 设备驱动</td></tr>
<tr><td>硬件</td><td>符合 UVC 标准的 USB 摄像头</td><td>AM 或 FM 高频头设备</td><td>其他 video 设备</td></tr>
</table>

使用 **V4L2** 接口控制摄像头采集图像基本步骤有哪些?

1.打开 video 设备

在 Linux 下一切皆文件，所以在 Linux 下摄像头也是一个文件，摄像头的设备文件是/dev/videox，x 可以是 0,1,2,.....。
可以按照以下方法来确定哪个设备节点对应摄像头：

第一步，查看一下/dev/下的 video 设备节点 ls /dev/video*。
第二步，把摄像头插好，再查看一下/dev/下的 video 设备节点，多出来那个就是新插摄像头的设备节点。

```
fd = open("/dev/video1", O_RDWR);
```

2.获得 video 设备的信息

```
struct v4l2_capability {
        __u8    driver[16];
        __u8    card[32];
        __u8    bus_info[32];
        __u32   version;
        __u32   capabilities;
        __u32   device_caps;
        __u32   reserved[3];
};
```

driver 字段：驱动信息。
card 字段：设备信息。
bus_info 字段：设备所在总线的信息。
version 字段：版本（一般为内核版本）。
capabilities 字段：video 设备具备的能力，该字段的取值如下。

```
V4L2_CAP_VIDEO_CAPTURE          /* Is a video capture device */
V4L2_CAP_VIDEO_OUTPUT           /* Is a video output device */
V4L2_CAP_VIDEO_OVERLAY          /* Can do video overlay */
V4L2_CAP_VBI_CAPTURE            /* Is a raw VBI capture device */
V4L2_CAP_VBI_OUTPUT             /* Is a raw VBI output device */
V4L2_CAP_SLICED_VBI_CAPTURE     /* Is a sliced VBI capture device */
V4L2_CAP_SLICED_VBI_OUTPUT      /* Is a sliced VBI output device */
V4L2_CAP_RDS_CAPTURE            /* RDS data capture */
V4L2_CAP_VIDEO_OUTPUT_OVERLAY/* Can do video output overlay */
V4L2_CAP_HW_FREQ_SEEK           /* Can do hardware frequency seek  */
```

在这里我们主要是查看该 video 设备是否有视频输入功能（摄像头）。

```
struct v4l2_capability cap;
//获取设备信息
ret = ioctl(fd, VIDIOC_QUERYCAP, &cap);
if(ret < 0){
        perror("perror VIDIOC_QUERYCAP");
        exit(1);
}

printf("driver:%s\ncard:%s\nbusinfo:%s\nversion:%d\ndevice_caps:%d\n", \
                cap.driver, cap.card, cap.bus_info, cap.version, cap.device_caps);
```

```
//判断该设备是否具有视频输入功能（摄像头）
if(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE){
    printf("This is a video capture\n");
}else{
    printf("This is not a video capture\n");
    exit(1);
}
```

3.查询 video 设备支持的所有图像格式

```
struct v4l2_fmtdesc {
    __u32           index;
    __u32           type;
    __u32           flags;
    __u8            description[32];
    __u32           pixelformat;
    __u32           reserved[4];
};
```

index 字段：图像格式索引。
type 字段：类型，V4L2_BUF_TYPE_VIDEO_CAPTURE。
description 字段：图像格式描述，如”JPEG”。
pixelformat 字段：由四个字符组成 pixelformat & 0xff，(pixelformat >> 8) & 0xff, (pixelformat >> 16) & 0xff，(pixelformat >> 24) & 0xff。

一个 video 设备可能会支持若干种图像格式，比如 JPEG 或者 YUV 等，所以在读取数据之前应该先搞明白 video 设备所支持的图像格式。
index 字段和 type 字段需要用户指定，index 表示该 video 设备所支持的图像格式的索引，对于摄像头来说 type 永远都指定为 V4L2_BUF_TYPE_VIDEO_CAPTURE。

```
struct v4l2_fmtdesc fmtdesc;
fmtdesc.index = 0;
fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

while(1){
    //获取格式信息
    ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc);
    if(ret < 0){
        //如果用户指定的 index 不正确则 ioctl 失败
        break;
    }
    printf("type:%c%c%c%c\n", fmtdesc.pixelformat & 0xff, \
    (fmtdesc.pixelformat >> 8) & 0xff, (fmtdesc.pixelformat >> 16) & 0xff,\
    (fmtdesc.pixelformat >> 24) & 0xff);
    printf("des:%s\n", fmtdesc.description);
    //index++以便获取 video 设备支持的下一个图像格式信息
```

```
        fmtdesc.index++;
}
```

4.查看 video 设备当前支持的图像格式

```
struct v4l2_format {
        __u32    type;
    union {
                struct v4l2_pix_format          pix;
                struct v4l2_pix_format_mplane   pix_mp;
                struct v4l2_window              win;
                struct v4l2_vbi_format          vbi;
                struct v4l2_sliced_vbi_format       sliced;
                __u8    raw_data[200];
        } fmt;
};
```

type 字段：对于摄像头指定为 V4L2_BUF_TYPE_VIDEO_CAPTURE。
fmt 字段：对于摄像头来说，我们需要的是 pix 表示的图像的像素信息。

```
struct v4l2_pix_format {
            __u32               width;
            __u32               height;
            __u32               pixelformat;
            __u32               field;
            __u32               bytesperline;
            __u32               sizeimage;
    __u32               colorspace;
    __u32               priv;
};
```

width 字段：图像宽度。
height 字段：图像高度。
pixelformat 字段：图像格式，该字段的取值如下。
V4L2_PIX_FMT_MJPEG
V4L2_PIX_FMT_MJPEG
V4L2_PIX_FMT_MPEG
V4L2_PIX_FMT_YVU420
V4L2_PIX_FMT_GREY
V4L2_PIX_FMT_RGB565
.....
bytesperline 字段：图像每行所占的字节数。
sizeimage 字段：图像大小。
colorspace 字段：色彩空间，该字段的取值如下。
V4L2_COLORSPACE_JPEG
V4L2_COLORSPACE_SRGB
.....

```
struct v4l2_format format;
format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_G_FMT, &format);
if(ret < 0){
        perror("VIDIOC_G_FMT");
        exit(1);
}

printf("width:%d\n", format.fmt.pix.width);
printf("height:%d\n", format.fmt.pix.height);
printf("pixelformat:%x\n", format.fmt.pix.pixelformat);
printf("field:%x\n", format.fmt.pix.field);
printf("bytesperline:%d\n", format.fmt.pix.bytesperline);       printf("sizeimage:%d\n",
format.fmt.pix.sizeimage);
rintf("colorspace:%d\n", format.fmt.pix.colorspace);
```

5.设置 video 设备的图像格式

```
format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
format.fmt.pix.width = 640;
format.fmt.pix.height = 480;
```

//测试代码使用的是中星微 ZC301 摄像头，该摄像头支持 JPEG 图像格式的抓取

```
format.fmt.pix.pixelformat = V4L2_PIX_FMT_JPEG;
ret = ioctl(fd, VIDIOC_S_FMT, &format);
if(ret < 0){
        perror("VIDIOC_S_FMT");
        exit(1);
}
```

6.向 video 驱动申请存放图像的缓存区

```
struct v4l2_requestbuffers {
        __u32           count;
        __u32           type;
        __u32           memory;
        __u32           reserved[2];
};
```

count 字段：缓存区的数量。

type 字段：对于摄像头指定为 V4L2_BUF_TYPE_VIDEO_CAPTURE。

memory 字段：用户访问缓存区的方法，有以下取值。

V4L2_MEMORY_MMAP
V4L2_MEMORY_USERPTR
V4L2_MEMORY_OVERLAY

向内核驱动申请存放图像数据的缓存区，一般不超过 5 个。
count 字段指定需要申请的缓存区的数量，需要用户指定，type 指定为 V4L2_BUF_TYPE_VIDEO_CAPTURE，memory 字段指定用户访问缓存区的方式，一般指定为 V4L2_MEMORY_MMAP，V4L2_MEMORY_MMAP 参数会让用户态和内核态共享内存（缓存区），比其他方式更高效。

```
struct v4l2_requestbuffers reqbuf;
reqbuf.count = 3;
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
ret = ioctl(fd, VIDIOC_REQBUFS, &reqbuf);
if(ret < 0){
    perror("VIDIOC_REQBUFS");
    exit(1);
}
```

7.把申请到的缓存映射到用户空间

```
struct v4l2_buffer {
    __u32            index;
    __u32            type;
    __u32            bytesused;
    __u32            flags;
    __u32            field;
    struct timeval       timestamp;
    struct v4l2_timecode    timecode;
    __u32            sequence;

    /* memory location */
    __u32            memory;
    union {
        __u32        offset;
        unsigned long        userptr;
        struct v4l2_plane    *planes;
        __s32        fd;
    } m;
    __u32            length;
    __u32            reserved2;
    __u32            reserved;
};
```

index 字段：缓存区的编号，从 0 开始。

type 字段：对于摄像头指定为 V4L2_BUF_TYPE_VIDEO_CAPTURE。
memory 字段：用户访问缓存区的方法，有以下取值。
V4L2_MEMORY_MMAP
V4L2_MEMORY_USERPTR
V4L2_MEMORY_OVERLAY
length 字段：缓存区长度。
offset 字段：缓存区偏移。

用 mmap 函数把内核中存放图像的缓存区映射到用户空间，有几个缓存区就需要映射几次。

```
struct bufinfo {
    void *start;
    unsigned int length;
}*bi;

bi = calloc(reqbuf.count, sizeof(struct bufinfo));
if(!bi){
    perror("calloc");
    exit(1);
}

struct v4l2_buffer buf;
int count = 0;
for(count = 0; count < reqbuf.count; count++){
    buf.index = count;
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_MMAP;
    ret = ioctl(fd, VIDIOC_QUERYBUF, &buf);
    if(ret < 0){
        perror("VIDIOC_REQBUFS");
        exit(1);
    }
    bi[buf.index].length = buf.length;
    bi[buf.index].start = mmap(NULL, buf.length, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
buf.m.offset);
    if(!(bi[buf.index].start)){
        error("mmap error");
        exit(1);
    }
}
```

8.把申请到的缓存放入采集队列

需要把申请到缓存加入到 video 驱动的队列中，这样驱动才会把图像数据存放到相应的缓存当中，用户需要指定 index，type，memory 字段，其实就是把第 index 个缓存区加入到队列尾部。

```
struct v4l2_buffer buf1;
int i = 0;
for(i = 0; i < reqbuf.count; i++){
        buf1.index = i;
        buf1.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf1.memory = V4L2_MEMORY_MMAP;
        ret = ioctl(fd, VIDIOC_QBUF, &buf1);
        if(ret < 0){
                perror("VIDIOC_QBUF");
                exit(1);
        }
}
```

9.开始采集数据

```
enum v4l2_buf_type {
        V4L2_BUF_TYPE_VIDEO_CAPTURE        = 1,
        V4L2_BUF_TYPE_VIDEO_OUTPUT         = 2,
        V4L2_BUF_TYPE_VIDEO_OVERLAY        = 3,
        V4L2_BUF_TYPE_VBI_CAPTURE          = 4,
        V4L2_BUF_TYPE_VBI_OUTPUT           = 5,
        V4L2_BUF_TYPE_SLICED_VBI_CAPTURE   = 6,
        V4L2_BUF_TYPE_SLICED_VBI_OUTPUT    = 7,
#if 1
        /* Experimental */
        V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY = 8,
#endif
        V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE = 9,
        V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE  = 10,
        /* Deprecated, do not use */
        V4L2_BUF_TYPE_PRIVATE              = 0x80,
};
```

让 video 驱动开始采集图像，采集到的图像会按顺序存放到队列中的缓存里。

```
enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_STREAMON, &type);
if(ret < 0){
        perror("VIDIOC_STREAMON");
        exit(1);
}
```

10.读取采集到的图像数据

用 select 函数检测 video 设备是否可读，当可读的时候说明缓存中就已经放好了完整的图像，只需要从队列中把缓存区取下，读取缓存中的数据就可以了。

```
fd_set readset;
FD_ZERO(&readset);
FD_SET(fd, &readset);
ret = select(fd + 1, &readset, NULL, NULL, NULL);
if(ret < 0){
	perror("select");
	exit(1);
}

struct v4l2_buffer buf2;
buf2.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf2.memory = V4L2_MEMORY_MMAP;
ret = ioctl(fd, VIDIOC_DQBUF, &buf2);
if(ret < 0){
	perror("VIDIOC_DQBUF");
	exit(1);
}
```

11.把数据写入文件

如果摄像头采集到的是 JPEG 格式的数据，则可以直接写入文件，这个文件就是 JPEG 格式的图片，如果摄像头采集到的是 YUV 格式的数据，则还需要压缩处理才能成为 JPEG 格式的图片。

```
int pic_fd;
pic_fd = open("./test.jpg", O_RDWR | O_CREAT | O_TRUNC, 0666);
if(pic_fd < 0){
	perror("open ./test.jgp");
	exit(1);
}

ret = write(pic_fd, bi[buf2.index].start, bi[buf2.index].length);
if(ret < 0){
	perror("write pic");
	exit(1);
}
close(pic_fd);
```

12.把缓存再次加入队列

把从队列中取下的缓存再次加入到队列，这样可以循环不断的读取图像数据。

```
ret = ioctl(fd, VIDIOC_QBUF, &buf2);
```

```
if(ret < 0){
        perror("VIDIOC_QBUF");
        exit(1);
}
```

13.释放内存

在不需要再读取图像数据的时候，释放由 mmap 映射的虚拟内存。

```
for(count = 0; count < reqbuf.count; count++){
        munmap(bi[buf.index].start, bi[buf.index].length);
}
```

14.停止采集图像

在不需要摄像头采集图像的时候，可以禁止摄像头采集图像。

```
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
ret = ioctl(fd, VIDIOC_STREAMOFF, &type);
if(ret < 0){
        perror("VIDIOC_STREAMOFF");
        exit(1);
}
```

15.关闭设备

```
close(fd);
```