V4l2应用程序框架

V4L2较V4L有较大的改动，并已成为2.6的标准接口，函盖video\dvb\FM...，多数驱动都在向V4l2迁移。更好地了解V4L2先从应 用入手，然后再深入到内核中结合物理设备／接口的规范实现相应的驱动。本文先就V4L2在视频捕捉或camera方面的应用框架。

V4L2采用流水线的方式，操作更简单直观，基本遵循打开视频设备、设置格式、处理数据、关闭设备，更多的具体操作通过ioctl函数来实现。

1.打开视频设备

在V4L2中，视频设备被看做一个文件。使用open函数打开这个设备：

```
// 用非阻塞模式打开摄像头设备
int cameraFd;
cameraFd = open("/dev/video0", O_RDWR | O_NONBLOCK, 0);
// 如果用阻塞模式打开摄像头设备，上述代码变为：
//cameraFd = open("/dev/video0", O_RDWR, 0);
```

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。

2. 设定属性及采集方式

打开视频设备后，可以设置该视频设备的属性，例如裁剪、缩放等。这一步是可选的。在Linux编程中，一般使用ioctl函数来对设备的I/O通道进行管理：

```
int ioctl (int __fd, unsigned long int __request, .../*args*/) ;
```

在进行V4L2开发中，常用的命令标志符如下(some are optional)：

- VIDIOC_REQBUFS：分配内存
- VIDIOC_QUERYBUF：把VIDIOC_REQBUFS中分配的数据缓存转换成物理地址
- VIDIOC_QUERYCAP：查询驱动功能
- VIDIOC_ENUM_FMT：获取当前驱动支持的视频格式
- VIDIOC_S_FMT：设置当前驱动的频捕获格式
- VIDIOC_G_FMT：读取当前驱动的频捕获格式
- VIDIOC_TRY_FMT：验证当前驱动的显示格式
- VIDIOC_CROPCAP：查询驱动的修剪能力
- VIDIOC_S_CROP：设置视频信号的边框
- VIDIOC_G_CROP：读取视频信号的边框
- VIDIOC_QBUF：把数据从缓存中读取出来
- VIDIOC_DQBUF：把数据放回缓存队列
- VIDIOC_STREAMON：开始视频显示函数
- VIDIOC_STREAMOFF：结束视频显示函数
- VIDIOC_QUERYSTD：检查当前视频设备支持的标准，例如PAL或NTSC。

2.1 检查当前视频设备支持的标准

在亚洲，一般使用PAL（720X576）制式的摄像头，而欧洲一般使用NTSC（720X480），使用VIDIOC_QUERYSTD来检测：

```
v4l2_std_id std;
do {
ret = ioctl(fd, VIDIOC_QUERYSTD, &std);
} while (ret == -1 && errno == EAGAIN);
switch (std) {
case V4L2_STD_NTSC:
//......
case V4L2_STD_PAL:
//......
}
```

2.2 设置视频捕获格式

当检测完视频设备支持的标准后，还需要设定视频捕获格式，结构如下：

```
struct v4l2_format fmt;
memset ( &fmt, 0, sizeof(fmt) );
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
```

```
fmt.fmt.pix.width = 720;
fmt.fmt.pix.height = 576;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
if (ioctl(fd, VIDIOC_S_FMT, &fmt) == -1) {
return -1;
} v4l2_format结
构
如
下
：
struct v4l2_format
{
enum v4l2_buf_type type; // 数据流类型，必须永远是V4L2_BUF_TYPE_VIDEO_CAPTURE
union
{
struct v4l2_pix_format pix;
struct v4l2_window win;
struct v4l2_vbi_format vbi;
__u8 raw_data[200];
} fmt;
};
struct v4l2_pix_format
{
__u32 width; // 宽，必须是16的倍数
__u32 height; // 高，必须是16的倍数
__u32 pixelformat; // 视频数据存储类型，例如是YUV4：2：2还是
RGB
enum v4l2_field field;
__u32 bytesperline;
__u32 sizeimage;
enum v4l2_colorspace colorspace;
__u32 priv;
};
```

2.3 分配内存

接下来可以为视频捕获分配内存：

```
struct v4l2_requestbuffers req;
if (ioctl(fd, VIDIOC_REQBUFS, &req) == -1) {
return -1;
} v4l2_requestbuffers
结
构
如
下
：
struct v4l2_requestbuffers
{
__u32 count; // 缓存数量，也就是说在缓存队列里保持多少张照片
enum v4l2_buf_type type; // 数据流类型，必须永远是V4L2_BUF_TYPE_VIDEO_CAPTURE
enum v4l2_memory memory; // V4L2_MEMORY_MMAP 或 V4L2_MEMORY_USERPTR
__u32 reserved[2];
};
```

2.4 获取并记录缓存的物理空间

使用VIDIOC_REQBUFS，我们获取了req.count个缓存，下一步通过调用VIDIOC_QUERYBUF命令来获
取这些缓存的地址，然后使用mmap函数转换成应用程序中的绝对地址，最后把这段缓存放入缓存队列：

```
typedef struct VideoBuffer {
void *start;
size_t length;
} VideoBuffer;
VideoBuffer* buffers = calloc( req.count, sizeof(*buffers) );
struct v4l2_buffer buf;
```

```
for (numBufs = 0; numBufs < req.count; numBufs++) {
memset( &buf, 0, sizeof(buf) );
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = numBufs;
// 读取缓存
if (ioctl(fd, VIDIOC_QUERYBUF, &buf) == -1) {
return -1;
}
buffers[numBufs].length = buf.length;
// 转换成相对地址
buffers[numBufs].start = mmap(NULL, buf.length, PROT_READ | PROT_WRITE,
MAP_SHARED,fd, buf.m.offset);
if (buffers[numBufs].start == MAP_FAILED) {
return -1;
}
// 放入缓存队列
if (ioctl(fd, VIDIOC_QBUF, &buf) == -1) {
return -1;
}
}
```

2.5 视频采集方式

操作系统一般把系统使用的内存划分成用户空间和内核空间，分别由应用程序管理和操作系统管理。应用程序可以直接访问内存的地址，而内核空间存放的是 供内核访问的代码和数据，用户不能直接访问。v4l2捕获的数据，最初是存放在内核空间的，这意味着用户不能直接访问该段内存，必须通过某些手段来转换地 址。

一共有三种视频采集方式：使用read、write方式；内存映射方式和用户指针模式。

read、write方式，在用户空间和内核空间不断拷贝数据，占用了大量用户内存空间，效率不高。

内存映射方式：把设备里的内存映射到应用程序中的内存控件，直接处理设备内存，这是一种有效的方式。上面的mmap函数就是使用这种方式。

用户指针模式：内存片段由应用程序自己分配。这点需要在v4l2_requestbuffers里将memory字段
设置成V4L2_MEMORY_USERPTR。

2.6 处理采集数据

V4L2有一个数据缓存，存放req.count数量的缓存数据。数据缓存采用FIFO的方式，当应用程序调用缓存数据时，缓存队列将最先采集到的视频数 据缓存送出，并重新采集一张视频数据。这个过程需要用到两个ioctl命令,VIDIOC_DQBUF和VIDIOC_QBUF：

```
struct v4l2_buffer buf;
memset(&buf,0,sizeof(buf));
buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory=V4L2_MEMORY_MMAP;
buf.index=0;
//读取缓存
if (ioctl(cameraFd, VIDIOC_DQBUF, &buf) == -1)
{
return -1;
} //.....视
频
处
理
算
法
//重新放入缓存队列
if (ioctl(cameraFd, VIDIOC_QBUF, &buf) == -1) {
return -1;
}
```

3. 关闭视频设备

使用close函数关闭一个视频设备

```
close(cameraFd)
```

如果使用mmap,最后还需要使用munmap方法。
＝＝＝＝ＥＯＦ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝