

2016中国数据库技术大会门票申请

# joryers的ChinaUnix博客

暂无签名

首页 | 博文目录 | 关于我



joryers

博客访问： 2598  
博文数量： 4  
博客积分： 0  
博客等级： 民兵  
技术积分： 0  
用户组： 普通用户  
注册时间： 2014-07-10 15:22

加关注

短消息

论坛

加好友

文章分类

全部博文（4）  
未分配的博文（4）

文章存档

2014年（4）

我的朋友

最近访客



amduron



dixiaobi



kingdeko



fengwei1



3346lkl

微信关注

IT168企业级官微

嵌入式Linux环境下视频采集知识(V4L2)

2014-07-10 15:25:19

分类： LINUX  
原文地址： 嵌入式Linux环境下视频采集知识(V4L2)    作者： kjqin

Video for Linux two(Video4Linux2)简称V4L2，是V4L的改进版。V4L2是linux操作系统下用于采集图片、视频和音频数据的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。在远程会议、可视电话、视频监控系统和嵌入式多媒体终端中都有广泛的应用。

### 一、Video for Linux two

在Linux下，所有外设都被看成一种特殊的文件，成为“设备文件”，可以象访问普通文件一样对其进行读写。一般来说，采用V4L2驱动的摄像头设备文件是/dev/v4l/video0。为了通用，可以建立一个到/dev/video0的链接。V4L2支持两种方式来采集图像：内存映射方式(mmap)和直接读取方式(read)。V4L2在include/linux/videodev.h文件中定义了一些重要的数据结构，在采集图像的过程中，就是通过对这些数据的操作来获得最终的图像数据。Linux系统V4L2的能力可在Linux内核编译阶段配置，默认情况下都有此开发接口。V4L2从Linux 2.5.x版本的内核中开始出现。

V4L2规范中不仅定义了通用API元素(Common API Elements)，图像的格式(Image Formats)，输入/输出方法(Input/Output)，还定义了Linux内核驱动处理视频信息的一系列接口(Interfaces)，这些接口主要有：

- 视频采集接口——Video Capture Interface;
- 视频输出接口—— Video Output Interface;
- 视频覆盖/预览接口——Video Overlay Interface;
- 视频输出覆盖接口——Video Output Overlay Interface;
- 编解码接口——Codec Interface。

### 二、应用程序通过V4L2进行视频采集的原理

V4L2支持内存映射方式(mmap)和直接读取方式(read)来采集数据，前者一般用于连续视频数据的采集，后者常用于静态图片数据的采集，本文重点讨论内存映射方式的视频采集。

应用程序通过V4L2接口采集视频数据分为五个步骤：

首先，打开视频设备文件，进行视频采集的参数初始化，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；

其次，申请若干视频采集的帧缓冲区，并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/处理视频数据；

第三，将申请到的帧缓冲区在视频采集输入队列排队，并启动视频采集；

第四，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；

第五，停止视频采集。

#### 1.视频采集的参数初始化


在Linux下，摄像头硬件已经被映射为设备文件“/dev/video0”，用open函数打开这个设备文件，获得其文件描述符fd\_v4l2，然后对这个文件描述符进行参数初始化。

(1) 设置视频的采集窗口参数

设置采集窗口就是在摄像头设备的取景范围之内设定一个视频采集区域。主要是对结构体v4l2\_crop赋值，v4l2\_crop由一个v4l2\_buffer\_type枚举类型的type和v4l2\_rect类型的结构体c构成，来描述视频采集窗口的类型和大小。type设置为视频采集类型



微信号：IT168qiye



系统架构师大会  
微信号：SACC2013

订阅

推荐博文

- vm.swappiness对于数据库的影...
- Linux内核调试技术——进程D...
- Solaris下core dump文件的分...
- 初步理解MySQL的gap锁
- 用FreeBSD抓取PPPoE帐号和密...

热词专题

- lua编译(linux)
- linux+ARM学习路线

V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE。c是表示采集窗口的大小的结构体，它的成员Left和Top分别表示视频采集区域的起始横坐标和纵坐标，width和height分别表示采集图像的宽度和高度。赋值后，用ioctl函数通过这个结构体对fd\_v4l2进行设置。

```
struct v4l2_crop { enum v4l2_buf_type type;

struct v4l2_rect c;

};
```

(2)设置视频点阵格式和点阵大小

主要是对结构体v4l2\_format进行赋值，它由type和联合体fmt构成，来描述视频设备当前行为和数据的格式。

把type赋值为视频采集类型V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE，表示定义了一个视频采集流类型的buffer。fmt中，pix为表示图形格式的v4l2\_pix\_format型结构体。需要设定pix里的几个变量，pixelformat表示采集格式，设置为V4L2\_PIX\_FMT\_YUV420;width、height表示图像的宽度、高度，以字节为单位;sizeimage表示图像所占的存储空间大小，以字节为单位;bytesperline表示每一行的字节数。赋值后，用ioctl函数通过这个结构体对fd\_v4l2进行设置。

```
struct v4l2_format

{ enum v4l2_buf_type type;

union

{ struct v4l2_pix_format pix; // V4L2_BUF_TYPE_VIDEO_CAPTURE

struct v4l2_window win; // V4L2_BUF_TYPE_VIDEO_OVERLAY

__u8 raw_data[200]; // user-defined

} fmt;

};
```

(3)设置视频采集的帧率

结构体v4l2\_streamparm来描述视频流的属性，它由type和联合体parm构成。type同上，由于选的是V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE，所以仅需设定parm中的v4l2\_capture型结构体capture即可。在其中，v4l2\_fract型结构体timeperframe表示平均每一帧所占的时间，由其元素numerator和denominator共同决定，该时长为numerator/denominator;而capturemode则表示采集模式，采集高质量图片值为1，一般设为0。赋值之后，用ioctl函数通过这个结构体对fd\_v4l2进行设置。

```
struct v4l2_streamparm

{ enum v4l2_buf_type type;

union

{ struct v4l2_captureparm capture;

struct v4l2_outputparm output;

__u8 raw_data[200]; /* user-defined */

} parm;

};
```

2.申请并设置视频采集的帧缓冲池

前期初始化完成后，只是解决了一帧视频数据的格式和大小问题，而连续视频帧数据的采集需要用帧缓冲池队列的方式来解决，即要通过驱动程序在内存中申请几个帧缓冲池来存放视频数据。

应用程序通过API接口提供的方法(VIDIOC\_REQBUFS)申请若干个视频数据的帧缓冲池，申请帧缓冲池数量一般不低于3个，每个帧缓冲池存放一帧视频数据，这些帧缓冲池在内核空间。

应用程序通过API接口提供的查询方法(VIDIOC\_QUERYBUF)查询到帧缓冲池在内核空间的长度和偏移量地址。

应用程序再通过内存映射方法(mmap)，将申请到的内核空间帧缓冲池的地址映射到用户空间地址，这样就可以直接处理帧缓冲池的数据。

(1)将帧缓冲池在视频输入队列排队，并启动视频采集

在驱动程序处理视频的过程中，定义了两个队列：视频采集输入队列(incoming queues)和视频采集输出队列(outgoing queues)，前者是等待驱动存放视频数据的队列，后者是驱动程序已经放入了视频数据的队列。如图2所示。

应用程序需要将上述帧缓冲池在视频采集输入队列排队(VIDIOC\_QBUF)，然后可启动视频采集。

## (2)循环往复，采集连续的视频数据

启动视频采集后，驱动程序开始采集一帧数据，把采集的数据放入视频采集输入队列的第一个帧缓冲区，一帧数据采集完成，也就是第一个帧缓冲区存满一帧数据后，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。驱动程序接下来采集下一帧数据，放入第二个帧缓冲区，同样帧缓冲区存满下一帧数据后，被放入视频采集输出队列。

应用程序从视频采集输出队列中取出含有视频数据的帧缓冲区，处理帧缓冲区中的视频数据，如存储或压缩。

最后，应用程序将处理完数据的帧缓冲区重新放入视频采集输入队列,这样可以循环采集，如图1所示。

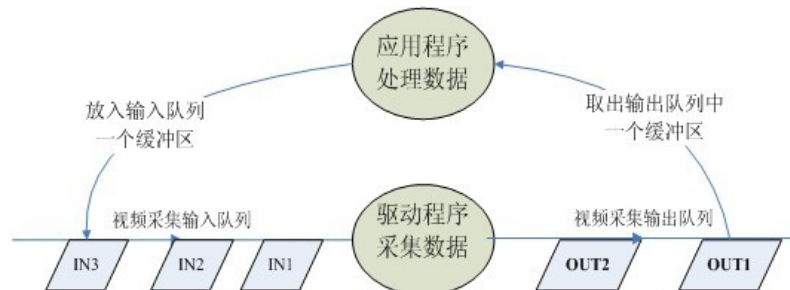


图1 视频采集输入和输出队列示意图

## (3)最终停止采集，释放内存帧缓冲区

### 3.用V4L2采集视频的程序流程和相关API

V4L2采集视频操作基本按照打开视频设备、设置视频格式、启动视频采集，循环处理视频数据、停止视频采集、关闭视频设备，具体操作通过ioctl等函数来实现。一般操作流程如下：

(1)打开视频设备文件。int fd=open("/dev/video0",O\_RDWR);

(2)查询视频设备的能力，比如是否具有视频输入,或者音频输入输出等。ioctl(fd\_v4l, VIDIOC\_QUERYCAP, &cap)

(3)设置视频采集的参数

设置视频的制式，制式包括PAL/NTSC，使用ioctl(fd\_v4l, VIDIOC\_S\_STD, &std\_id)

设置视频图像的采集窗口的大小，使用ioctl(fd\_v4l, VIDIOC\_S\_CROP, &crop)

设置视频帧格式，包括帧的点阵格式，宽度和高度等，使用ioctl(fd\_v4l, VIDIOC\_S\_FMT, &fmt)

设置视频的帧率，使用ioctl(fd\_v4l, VIDIOC\_S\_PARM, &parm)

设置视频的旋转方式，使用ioctl(fd\_v4l, VIDIOC\_S\_CTRL, &ctrl)

(4)向驱动申请视频流数据的帧缓冲区

请求/申请若干个帧缓冲区，一般为不少于3个,使用ioctl(fd\_v4l, VIDIOC\_REQBUFS, &req)

查询帧缓冲区在内核空间中的长度和偏移量 ioctl(fd\_v4l, VIDIOC\_QUERYBUF, &buf)

(5)应用程序通过内存映射，将帧缓冲区的地址映射到用户空间，这样就可以直接操作采集到的帧了，而不必去复制。

buffers[i].start = mmap (NULL, buffers[i].length, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_v4l, buffers[i].offset);

(6)将申请到的帧缓冲全部放入视频采集输出队列，以便存放采集的数据。ioctl (fd\_v4l, VIDIOC\_QBUF, &buf)

(7)开始视频流数据的采集。ioctl (fd\_v4l, VIDIOC\_STREAMON, &type)

(8)驱动将采集到的一帧视频数据存入输入队列第一个帧缓冲区，存完后将该帧缓冲区移至视频采集输出队列。

(9)应用程序从视频采集输出队列中取出已含有采集数据的帧缓冲区。ioctl (fd\_v4l, VIDIOC\_DQBUF, &buf)，应用程序处理该帧缓冲区的原始视频数据。

(10)处理完后，应用程序的将该帧缓冲区重新排入输入队列,这样便可以循环采集数据。ioctl (fd\_v4l, VIDIOC\_QBUF, &buf)

重复上述步骤8到10，直到停止采集数据。

(11)停止视频的采集。ioctl (fd\_v4l, VIDIOC\_STREAMOFF, &type)

(12)释放申请的帧缓冲区unmap，关闭视频设备文件close(fd\_v4l)。

以上的程序流程, 包含了视频设备采集连续的视频数据的逻辑关系。而在实际运用中, 往往还要加入对视频数据进行处理(如压缩编码)的工作, 否则, 视频流数据量相当大, 需要很大的存储空间和传输带宽。

上述过程中, 每一个帧缓冲区都有一个对应的状态标志变量, 其中每一个比特代表一个状态

V4L2\_BUF\_FLAG\_UNMAPPED 0B0000

V4L2\_BUF\_FLAG\_MAPPED 0B0001

V4L2\_BUF\_FLAG\_ENQUEUED 0B0010

V4L2\_BUF\_FLAG\_DONE 0B0100

缓冲区的状态转化如图2所示。

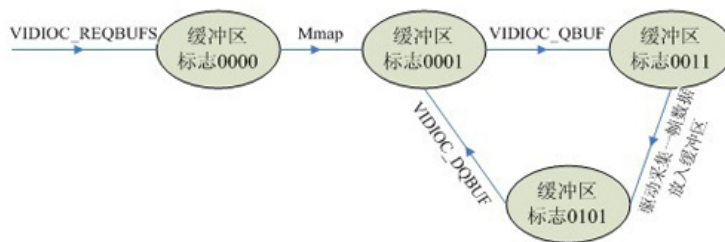


图2 缓冲区的状态标志转化图

### 三、结束语

V4L2是Linux环境下开发视频采集设备驱动程序的一套规范(API), 它为驱动程序的编写提供统一的接口, 并将所有的视频采集设备的驱动程序都纳入其的管理之中。V4L2不仅给驱动程序编写者带来极大的方便, 同时也方便了应用程序的编写和移植, 具有广泛的应用价值。

## V4L2 编程

### 1. 定义

V4L2(Video For Linux Two) 是内核提供给应用程序访问音、视频驱动的统一接口。

### 2. 工作流程:

打开设备 -> 检查和设置设备属性 -> 设置帧格式 -> 设置一种输入输出方法 (缓冲区管理) -> 循环获取数据 -> 关闭设备。

### 3. 设备的打开和关闭:

```
#include
int open(const char *device_name, int flags);

#include
int close(int fd);

例:
int fd=open("/dev/video0",O_RDWR); // 打开设备
close(fd); // 关闭设备
```

注意: V4L2 的相关定义包含在头文件中。

#### 4. 查询设备属性: VIDIOC\_QUERYCAP

相关函数:

```
int ioctl(int fd, int request, struct v4l2_capability *argp);
```

相关结构体:

```
struct v4l2_capability
{
    __u8 driver[16]; // 驱动名字
    __u8 card[32]; // 设备名字
    __u8 bus_info[32]; // 设备在系统中的位置
    __u32 version; // 驱动版本号
    __u32 capabilities; // 设备支持的操作
    __u32 reserved[4]; // 保留字段
};
```

capabilities 常用值:

V4L2\_CAP\_VIDEO\_CAPTURE // 是否支持图像获取

#### 例: 显示设备信息

```
struct v4l2_capability cap;
ioctl(fd,VIDIOC_QUERYCAP,&cap);
printf("Driver Name:%s/nCard Name:%s/nBus info:%s/nDriver
Version:%u.%u.%u/n",cap.driver,cap.card,cap.bus_info,(cap.version>>16)&0xFF,
(cap.version>>8)&0xFF,cap.version&0xFF);
```

#### 5. 帧格式:

VIDIOC\_ENUM\_FMT // 显示所有支持的格式

```
int ioctl(int fd, int request, struct v4l2_fmtdesc *argp);
```

```
struct v4l2_fmtdesc
{
    __u32 index; // 要查询的格式序号, 应用程序设置
    enum v4l2_buf_type type; // 帧类型, 应用程序设置
    __u32 flags; // 是否为压缩格式
    __u8 description[32]; // 格式名称
    __u32 pixelformat; // 格式
    __u32 reserved[4]; // 保留
};
```

#### 例: 显示所有支持的格式

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

fmtdesc.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
printf("Support format:/n");
while(ioctl(fd,VIDIOC_ENUM_FMT,&fmtdesc)!=-1)
{
```

```
printf("/t%d.%s/n",fmtdesc.index+1,fmtdesc.description);
fmtdesc.index++;
}

// 查看或设置当前格式
VIDIOC_G_FMT,VIDIOC_S_FMT

// 检查是否支持某种格式
VIDIOC_TRY_FMT

int ioctl(int fd, int request, struct v4l2_format *argp);
struct v4l2_format
{
enum v4l2_buf_type type;// 帧类型，应用程序设置
union fmt
{
struct v4l2_pix_format pix;// 视频设备使用
struct v4l2_window win;
struct v4l2_vbi_format vbi;
struct v4l2_sliced_vbi_format sliced;
__u8 raw_data[200];
};
};

struct v4l2_pix_format
{
__u32 width; // 帧宽，单位像素
__u32 height; // 帧高，单位像素
__u32 pixelformat; // 帧格式
enum v4l2_field field;
__u32 bytesperline;
__u32 sizeimage;
enum v4l2_colorspace colorspace;
__u32 priv;
};

例：显示当前帧的相关信息

struct v4l2_format fmt;
fmt.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
ioctl(fd,VIDIOC_G_FMT,&fmt);
printf("Current data format
information:/n/twidth:%d/n/theight:%d/n",fmt.fmt.width,fmt.fmt.height);
struct v4l2_fmtdesc fmtdesc;
fmtdesc.index=0;
fmtdesc.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
```

```
while(ioctl(fd,VIDIOC_ENUM_FMT,&fmtdesc)!=-1)
{
if(fmtdesc.pixelformat & fmt.fmt.pixelformat)
{
printf("/tformat:%s/n",fmtdesc.description);
break;
}
fmtdesc.index++;
}
}
```

**例：检查是否支持某种帧格式**

```
struct v4l2_format fmt;
fmt.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.pixelformat=V4L2_PIX_FMT_RGB32;
if(ioctl(fd,VIDIOC_TRY_FMT,&fmt)==-1)
if(errno==EINVAL)
printf("not support format RGB32!/n");
```

## 6. 图像的缩放

VIDIOC\_CROPCAP

```
int ioctl(int fd, int request, struct v4l2_cropcap *argp);
struct v4l2_cropcap
{
enum v4l2_buf_type type;// 应用程序设置
struct v4l2_rect bounds;// 最大边界
struct v4l2_rect defrect;// 默认值
struct v4l2_fract pixelaspect;
};
```

// 设置缩放

VIDIOC\_G\_CROP,VIDIOC\_S\_CROP

```
int ioctl(int fd, int request, struct v4l2_crop *argp);
int ioctl(int fd, int request, const struct v4l2_crop *argp);
struct v4l2_crop
{
enum v4l2_buf_type type;// 应用程序设置
struct v4l2_rect c;
}
```

7. 申请和管理缓冲区，应用程序和设备有三种交换数据的方法，直接 read/write ，内存映射(memory mapping) ，用户指针。这里只讨论 memory mapping.

// 向设备申请缓冲区

```
VIDIOC_REQBUFS

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

struct v4l2_requestbuffers
{
    __u32 count; // 缓冲区内缓冲帧的数目
    enum v4l2_buf_type type; // 缓冲帧数据格式
    enum v4l2_memory memory; // 区别是内存映射还是用户指针方式
    __u32 reserved[2];
};
```

```
enum v4l2_memoy {V4L2_MEMORY_MMAP,V4L2_MEMORY_USERPTR};
//count,type,memory 都要应用程序设置
```

例：申请一个拥有四个缓冲帧的缓冲区

```
struct v4l2_requestbuffers req;
req.count=4;
req.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory=V4L2_MEMORY_MMAP;
ioctl(fd,VIDIOC_REQBUFS,&req);
```

获取缓冲帧的地址，长度：

VIDIOC\_QUERYBUF

```
int ioctl(int fd, int request, struct v4l2_buffer *argp);

struct v4l2_buffer
{
    __u32 index; //buffer 序号
    enum v4l2_buf_type type; //buffer 类型
    __u32 byteused; //buffer 中已使用的字节数
    __u32 flags; // 区分是MMAP 还是USERPTR
    enum v4l2_field field;
    struct timeval timestamp; // 获取第一个字节时的系统时间
    struct v4l2_timecode timecode;
    __u32 sequence; // 队列中的序号
    enum v4l2_memory memory; //IO 方式，被应用程序设置
    union m
    {
        __u32 offset; // 缓冲帧地址，只对MMAP 有效
        unsigned long userptr;
    };
    __u32 length; // 缓冲帧长度
    __u32 input;
    __u32 reserved;
};
```



mmap，定义一个结构体来映射每个缓冲帧。

```
Struct buffer
{
void* start;
unsigned int length;
}*buffers;

#include
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t
offset);
//addr 映射起始地址，一般为NULL，让内核自动选择
//length 被映射内存块的长度
//prot 标志映射后能否被读写，其值为PROT_EXEC,PROT_READ,PROT_WRITE, PROT_NONE
//flags 确定此内存映射能否被其他进程共享，MAP_SHARED,MAP_PRIVATE
//fd,offset, 确定被映射的内存地址
返回成功映射后的地址，不成功返回MAP_FAILED ((void*)-1);

int munmap(void *addr, size_t length);// 断开映射
//addr 为映射后的地址，length 为映射后的内存长度
```

**例：将四个已申请到的缓冲帧映射到应用程序，用buffers 指针记录。**

```
buffers = (buffer*)calloc (req.count, sizeof (*buffers));
if (!buffers) {
fprintf (stderr, "Out of memory/n");
exit (EXIT_FAILURE);
}
// 映射
for (unsigned int n_buffers = 0; n_buffers < req.count; ++n_buffers) {
struct v4l2_buffer buf;
memset(&buf,0,sizeof(buf));
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = n_buffers;
// 查询序号为n_buffers 的缓冲区，得到其起始物理地址和大小
if (-1 == ioctl (fd, VIDIOC_QUERYBUF, &buf))
exit(-1);
buffers[n_buffers].length = buf.length;
// 映射内存
buffers[n_buffers].start =mmap (NULL,buf.length,PROT_READ|PROT_WRITE ,MAP_SHARED,fd,
buf.m.offset);
if (MAP_FAILED == buffers[n_buffers].start)
```

```
exit(-1);  
}
```

8. 缓冲区处理好之后，就可以开始获取数据了

// 启动/ 停止数据流

```
VIDIOC_STREAMON,VIDIOC_STREAMOFF
```

```
int ioctl(int fd, int request, const int *argp);
```

//argp 为流类型指针，如V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE.

在开始之前，还应当把缓冲帧放入缓冲队列：

```
VIDIOC_QBUF// 把帧放入队列
```

```
VIDIOC_DQBUF// 从队列中取出帧
```

```
int ioctl(int fd, int request, struct v4l2_buffer *argp);
```

例：把四个缓冲帧放入队列，并启动数据流

```
unsigned int i;  
enum v4l2_buf_type type;  
// 将缓冲帧放入队列  
for (i = 0; i < 4; ++i)  
{  
    struct v4l2_buffer buf;  
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;  
    buf.memory = V4L2_MEMORY_MMAP;  
    buf.index = i;  
    ioctl (fd, VIDIOC_QBUF, &buf);  
}  
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;  
ioctl (fd, VIDIOC_STREAMON, &type);
```

// 这有个问题，这些buf 看起来和前面申请的buf 没什么关系，为什么呢？

例：获取一帧并处理

```
struct v4l2_buffer buf;  
CLEAR (buf);  
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;  
buf.memory = V4L2_MEMORY_MMAP;  
// 从缓冲区取出一个缓冲帧  
ioctl (fd, VIDIOC_DQBUF, &buf);  
// 图像处理  
process_image (buffers[buf.index].start);  
// 将取出的缓冲帧放回缓冲区  
ioctl (fd, VIDIOC_QBUF, &buf);
```

阅读(281) | 评论(0) | 转发(0) |

上一篇: v4l2驱动编写篇

下一篇: 用gcc 的-D 参数来调试代码

0

#### 相关热门文章

【Android体系】Android基础知...

嵌入式系统中看门狗的使用总结...

NAND Flash--嵌入式NAND Flash...

嵌入式Linux之我行——按键驱...

QEMU源码分析系列(二)

linux 常见服务端口

xmanager 2.0 for linux配置

【ROOTFS搭建】busybox的httpd...

openwrt中luci学习笔记

什么是shell

linux dhcp peizhi roc

关于Unix文件的软链接

求教这个命令什么意思，我是新...

sed -e "/grep/d" 是什么意思...

谁能够帮我解决LINUX 2.6 10...

给主人留下些什么吧! ~~

#### 评论热议

登录后评论。

[登录](#) [注册](#)

[关于我们](#) | [关于IT168](#) | [联系方式](#) | [广告合作](#) | [法律声明](#) | [免费注册](#)

Copyright 2001-2010 ChinaUnix.net All Rights Reserved 北京皓辰网域网络信息技术有限公司. 版权所有

感谢所有关心和支持过ChinaUnix的朋友们

京ICP证041476号 京ICP证060528号