


个人资料



MetalSeed

访问： 267726次

积分： 3744

等级： 

BLOG > 5

排名： 第4880名

原创： 108篇 转载： 22篇

译文： 2篇 评论： 142条

文章搜索

文章分类

处理器Ambarella (1)

嵌入式Android (5)

嵌入式linux (19)

STM32笔记 (14)

ACM回忆 (47)

MCU相关 (27)

随笔 小记 (2)

东西 小站 (6)

文章存档

2015年06月 (1)

2015年05月 (1)

2015年04月 (3)

2015年03月 (4)

2015年02月 (3)

展开

阅读排行

数据结构专题——线段树 (39082)

主席树介绍 (18582)

NE555 + CD4017流水灯

2016软考项目经理实战班 学院周年礼-顶尖课程钜惠呈现 Hadoop英雄会—暨Hadoop 10周年生日大趴 【博客专家】有奖试读—Windows PowerShell实战指南

Video for linux 2 example (v4l2 demo)

2014-10-03 16:46 1342人阅读 评论(3) 收藏 举报

分类： 嵌入式linux (18)

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?)

[+]

V4L2 API讲解附demo

(注：从他人博客整理修正而来)

(看完本文后，更简便的api请移步至[video for linux 2 API](#))

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：

```
[cpp] view plain copy print ? 1
#include <fcntl.h>

int open(const char *device_name, int flags);

#include <unistd.h>

int close(int fd);
```

例：

```
[cpp] view plain copy print ? 1
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<linux/videodev2.h> 中。

4. 查询设备属性： VIDIOC\_QUERYCAP

相关函数：

Android蓝牙串口通信模块	(16998)
两款主流摄像头OV7620	(14330)
锁存器使用总结	(9650)
D/A与A/D转换器	(5400)
STM32F407 Discovery u	(5338)
51单片机导论	(4209)
51单片机导论	(3706)
各种音频视频编码方法	(3579)

评论排行	
数据结构专题——线段树	(36)
主席树介绍	(12)
Android蓝牙串口通信模块	(11)
51单片机进阶	(6)
51单片机导论	(6)
STM32F051 IAP源码分享	(5)
天马行空的ACM现场赛回顾	(5)
基于51的爱心流水灯源码	(5)
C代码优化方案	(4)
悬挂运动控制系统 源代码	(3)

推荐文章	
*一个炫字都不够? ? ! ! ! 手把手带你打造3D自定义view	
*21行python代码实现拼写检查器	
*数据库性能优化之SQL语句优化	
*拉开大变革序幕 (下) : 分布式计算框架与大数据	
*Chromium网页URL加载过程分析	
*Hadoop中止下线操作后大量剩余复制块的解决方案	

最新评论	
主席树介绍	
cww97: 代码有毒	
STM32F051 IAP源码分享	
黄大刀: 思路明确。step2 和step3很关键。	
追梦的故事	
Flaergwe: 哦，还以为是你呢。已经大三。。不过我感觉跟你经历的东西有点相似，我是嵌入式的方向却有acm的回忆。	
天马行空的ACM现场赛回顾	
Tuesday...: 诶看着竟然有点想哭=。=	
追梦的故事	
MetalSeed: @Adrian_1:这个是很久之前的从人转来的 你是大几的小盆友呀	
追梦的故事	
Flaergwe: 话说追梦网好多的妹子	
天马行空的ACM现场赛回顾	
Flaergwe: ACM之后再弄嵌入式	
数据结构专题——线段树	
z1qdhrrdhr: 博主第3块代码有个小错误那个left跟right应该是begin和end吧 ^_^	
数据结构专题——线段树	
newmemory: @Anger_Coder:不过需要注意一点，移位运算符的优先级低，在这里就比“+”低，最后的结果就是...	
数据结构专题——线段树	
清风小竹: 楼主的博客写的很好!赞一个	

```
[cpp] view plain copy print ?
int ioctl(int fd, int request, struct v4l2_capability *argp);
```

相关结构体:

```
[cpp] view plain copy print ?
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

```
[cpp] view plain copy print ?
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.version>>16)&0xFF, (cap.version>>8)&0xFF,cap.version&0xFF);
```

## 5. 设置视频的制式和帧格式

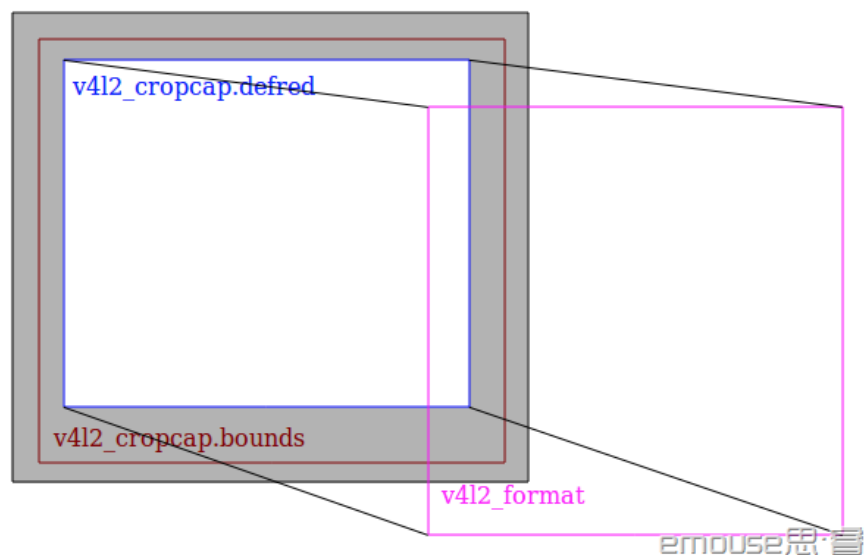
制式包括PAL，NTSC，帧的格式个包括宽度和高度等。  
相关函数:

```
[cpp] view plain copy print ?
int ioctl(int fd, int request, struct v4l2_fmtdesc *argp);

int ioctl(int fd, int request, struct v4l2_format *argp);
```

相关结构体:

v4l2\_cropcap 结构体用来设置摄像头的捕捉能力，在捕捉上视频时应先设置v4l2\_cropcap 的 type 域，再通过 VIDIO\_CROPCAP 操作命令获取设备捕捉能力的参数，保存于 v4l2\_cropcap 结构体中，包括 bounds（最大捕捉方框的左上角坐标和宽高），defrect（默认捕捉方框的左上角坐标和宽高）等。  
v4l2\_format 结构体用来设置摄像头的视频制式、帧格式等，在设置这个参数时应先填好 v4l2\_format 的各个域，如 type（传输流类型），fmt.pix.width(宽)，fmt.pix.height(高)，fmt.pix.field(采样区域，如隔行采样)，fmt.pix.pixelformat(采样类型，如 YUV4:2:2)，然后通过 VIDIO\_S\_FMT 操作命令设置视频捕捉格式。如下图所示:



## 5.1 查询并显示所有支持的格式：VIDIOC\_ENUM\_FMT

相关函数：

```
[cpp] view plain copy print ? ⓘ
int ioctl(int fd, int request, struct v4l2_fmtdesc *argp);
```

相关结构体：

```
[cpp] view plain copy print ? ⓘ
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

```
[cpp] view plain copy print ? ⓘ
struct v4l2_fmtdesc fmdesc; fmdesc.index=0; fmdesc.type=V4L2_BUF_TYPE_VIDEO_CAPTURE; printf("Support\n");
while(ioctl(fd, VIDIOC_ENUM_FMT, &fmdesc) != -1)
{
    printf("%d\t%s\n", fmdesc.index, fmdesc.description);
    fmdesc.index++;
}
```

## 5.2 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

相关函数:

```
[cpp] view plain copy print ? ⓘ
int ioctl(int fd, int request, struct v4l2_format *argp);
```

相关结构体:

```
[cpp] view plain copy print ? ⓘ
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

```
[cpp] view plain copy print ? ⓘ
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.pix.width, fmt.fmt.pix.height);

struct v4l2_fmtdesc fmdesc;

fmdesc.index = 0;

fmdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

while(ioctl(fd,VIDIOC_ENUM_FMT,&fmdesc) != -1)
{
    if(fmdesc.pixelformat & fmt.fmt.pix.pixelformat)
    {
        printf("\tformat:%s\n",fmdesc.description);

        break;
    }
    fmdesc.index++;
}
```

例：检查是否支持某种帧格式

```
[cpp] view plain copy print ?
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

相关函数：

```
[cpp] view plain copy print ?
int ioctl(int fd, int request, struct v4l2_cropcap *argp);

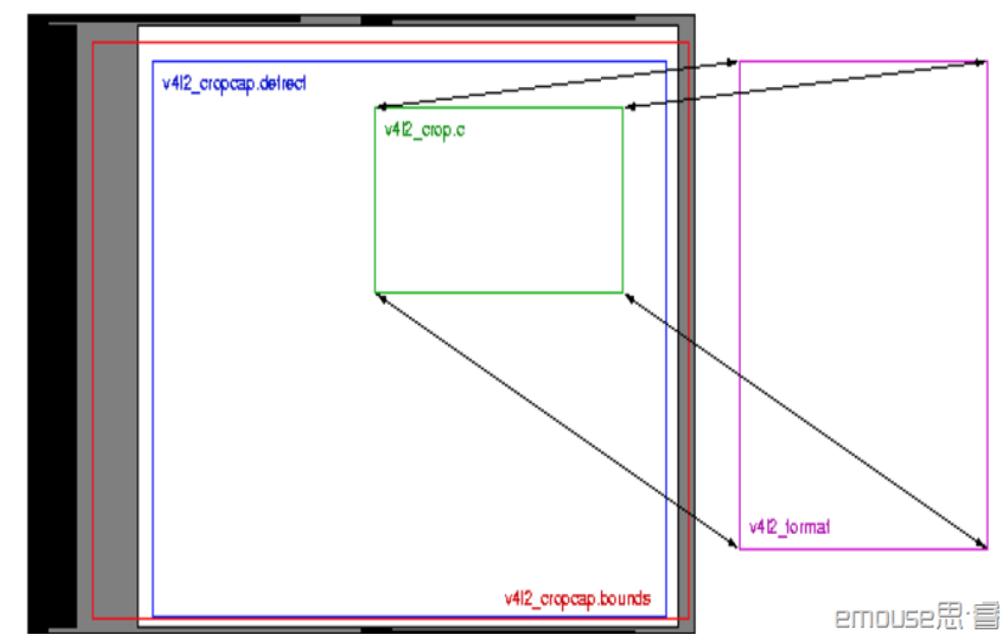
int ioctl(int fd, int request, struct v4l2_crop *argp);

int ioctl(int fd, int request, const struct v4l2_crop *argp);
```

相关结构体：

v4l2\_cropcap 结构体用来设置摄像头的捕捉能力，在捕捉上视频时应先设置v4l2\_cropcap 的 type 域，再通过 VIDIOC\_CROPCAP 操作命令获取设备捕捉能力的参数，保存于 v4l2\_cropcap 结构体中，包括 bounds（最大捕捉方框的左上角坐标和宽高），defrect（默认捕捉方框的左上角坐标和宽高）等。

Cropping 和 scaling 主要指的是图像的取景范围及图片的比例缩放的支持。Crop 就是把得到的数据作一定的裁剪和伸缩，裁剪可以只取样我们可以得到的图像大小的一部分，剪裁的主要参数是位置、长度、宽度。而 scale 的设置是通过 VIDIOC\_G\_FMT 和 VIDIOC\_S\_FMT 来获得和设置当前的 image 的长度，宽度来实现的。看下图



我们可以假设 bounds 是 sensor 最大能捕捉到的图像范围，而 defrect 是设备默认 的最大取样范围，这个可以通过 VIDIOC\_CROPCAP 的 ioctl 来获得设备的 crap 相关的属 性 v4l2\_cropcap，其中的 bounds 就是这个 bounds，其实就是上限。每个设备都有个默 认的取样范围，就是 defrect，就是 default rect 的意思，它比 bounds 要小一些。这个范围也是通过 VIDIOC\_CROPCAP 的 ioctl 来获得的 v4l2\_cropcap 结构中的 defrect 来表示的，我们可以通过 VIDIOC\_G\_CROP 和 VIDIOC\_S\_CROP 来获取和设置设备当前的 crop 设置。

## 6.1 设置设备捕捉能力的参数

相关函数:

```
[cpp] view plain copy print ? ⓘ  
int ioctl(int fd, int request, struct v4l2_cropcap *argp);
```

相关结构体:

```
[cpp] view plain copy print ? ⓘ  
struct v4l2_cropcap  
{  
    enum v4l2_buf_type type; // 数据流的类型, 应用程序设置  
    struct v4l2_rect bounds; // 这是 camera 的镜头能捕捉到的窗口大小的局限  
    struct v4l2_rect defrect; // 定义默认窗口大小, 包括起点位置及长,宽的大小, 大小以像素为单位  
    struct v4l2_fract pixelaspect; // 定义了图片的宽高比  
};
```

## 6.2 设置窗口取景参数 VIDIOC\_G\_CROP 和 VIDIOC\_S\_CROP

相关函数:

```
[cpp] view plain copy print ? ⓘ  
int ioctl(int fd, int request, struct v4l2_crop *argp);  
int ioctl(int fd, int request, const struct v4l2_crop *argp);
```

相关结构体:

```
[cpp] view plain copy print ? ⓘ  
struct v4l2_crop  
{  
    enum v4l2_buf_type type; // 应用程序设置  
    struct v4l2_rect c;  
}
```

## 7.video Inputs and Outputs

VIDIOC\_G\_INPUT 和 VIDIOC\_S\_INPUT 用来查询和选则当前的 input, 一个 video 设备 节点可能对应多个视频源, 比如 saf7113 可以最多支持四路 cvbs 输入, 如果上层想在四个cvbs视频输入间切换, 那么就要调用 ioctl(fd, VIDIOC\_S\_INPUT, &input) 来切换。VIDIOC\_G\_INPUT and VIDIOC\_G\_OUTPUT 返回当前的 video input和output的index.

相关函数:

```
[cpp] view plain copy print ? ⓘ  
int ioctl(int fd, int request, struct v4l2_input *argp);
```

相关结构体:

```
[cpp] view plain copy print ? ⓘ
```

```

struct v4l2_input
{
    __u32 index;    /* Which input */
    __u8 name[32];  /* Label */
    __u32 type;     /* Type of input */
    __u32 audioset; /* Associated audios (bitfield) */
    __u32 tuner;    /* Associated tuner */
    v4l2_std_id std;
    __u32 status;
    __u32 reserved[4];
};

```

我们可以通过VIDIOC\_ENUMINPUT and VIDIOC\_ENUMOUTPUT 分别列举一个input或者 output的信息，我们使用一个v4l2\_input结构体来存放查询结果，这个结构体中有一个 index域用来指定你索要查询的是第几个input/output,如果你所查询的这个input是当前正在使用的，那么在v4l2\_input还会包含一些当前的状态信息，如果所查询的input/output 不存在，那么会返回EINVAL错误，所以，我们通过循环查找，直到返回错误来遍历所有的 input/output. VIDIOC\_G\_INPUT and VIDIOC\_G\_OUTPUT 返回当前的video input和output 的index.

例：列举当前输入视频所支持的视频格式

```

[cpp] view plain copy print ?
struct v4l2_input input;
struct v4l2_standard standard;

memset (&input, 0, sizeof (input));

//首先获得当前输入的 index,注意只是 index, 要获得具体的信息, 就的调用列举操作
if (-1 == ioctl (fd, VIDIOC_G_INPUT, &input.index))
{
    perror ("VIDIOC_G_INPUT");
    exit (EXIT_FAILURE);
}

//调用列举操作, 获得 input.index 对应的输入的具体信息
if (-1 == ioctl (fd, VIDIOC_ENUMINPUT, &input))
{
    perror ("VIDIOC_ENUM_INPUT");
    exit (EXIT_FAILURE);
}

printf ("Current input %s supports:\n", input.name); memset (&standard, 0, sizeof (standard)); standard.index = 0;

//列举所有的所支持的 standard, 如果 standard.id 与当前 input 的 input.std 有共同的
//bit flag, 意味着当前的输入支持这个 standard, 这样将所有驱动所支持的 standard 列举一个
//遍, 就可以找到该输入所支持的所有 standard 了。
while (0 == ioctl (fd, VIDIOC_ENUMSTD, &standard))
{
    if (standard.id & input.std)
        printf ("%s\n", standard.name);
    standard.index++;
}

/* EINVAL indicates the end of the enumeration, which cannot be empty unless this device falls under t
if (errno != EINVAL || standard.index == 0)
{
    perror ("VIDIOC_ENUMSTD");
    exit (EXIT_FAILURE);
}

```

## 8. Video standards

相关函数：

```

[cpp] view plain copy print ?
v4l2_std_id std_id; //这个就是个64bit得数

int ioctl(int fd, int request, struct v4l2_standard *argp);

```

相关结构体:

```
[cpp] view plain copy print ?
typedef u64 v4l2_std_id;

struct v4l2_standard
{
    u32 index;

    v4l2_std_id id;

    u8 name[24];

    struct v4l2_fract frameperiod; /* Frames, not fields */

    u32 framelines;

    u32 reserved[4];
};
```

当然世界上现在有多个视频标准，如NTSC和PAL，他们又细分为好多种，那么我们的设备输入/输出究竟支持什么样的标准呢？我们的当前在使用的输入和输出正在使用的是哪个标准呢？我们怎么设置我们的某个输入输出使用的标准呢？这都是有方法的。

查询我们的输入支持什么标准，首先就得找到当前的这个输入的index，然后查出它的属性，在其属性里面可以得到该输入所支持的标准，将它所支持的各个标准与所有的标准的信息进行比较，就可以获知所支持的各个标准的属性。一个输入所支持的标准应该是一个集合，而这个集合是用bit与的方式用一个64位数字表示。因此我们所查到的是一个数字。

Example: Information about the current video standard v4l2\_std\_id std\_id; //这个就是个64bit得数

```
[cpp] view plain copy print ?
struct v4l2_standard standard;

// VIDIOC_G_STD就是获得当前输入使用的standard，不过这里只是得到了该标准的id
// 即flag，还没有得到其具体的属性信息，具体的属性信息要通过列举操作来得到。
if (-1 == ioctl (fd, VIDIOC_G_STD, &std_id))
{
    perror ("VIDIOC_G_STD");
    exit (EXIT_FAILURE);
//获得了当前输入使用的standard

// Note when VIDIOC_ENUMSTD always returns EINVAL this is no video device
// or it falls under the USB exception, and VIDIOC_G_STD returning EINVAL
// is no error.
}

memset (&standard, 0, sizeof (standard));

standard.index = 0; //从第一个开始列举

// VIDIOC_ENUMSTD用来列举所支持的所有的video标准的信息，不过要先给standard
// 结构的index域制定一个数值，所列举的标准的信息属性包含在standard里面，
// 如果我们所列举的标准和std_id有共同的bit，那么就意味着这个标准就是当前输入
// 所使用的标准，这样我们就得到了当前输入使用的标准的属性信息
while (0 == ioctl (fd, VIDIOC_ENUMSTD, &standard))
{
    if (standard.id & std_id)
    {
        printf ("Current video standard: %s\n", standard.name);
        exit (EXIT_SUCCESS);
    }

    standard.index++;
}

/* EINVAL indicates the end of the enumeration, which cannot be empty unless this device falls under t
if (errno == EINVAL || standard.index == 0)
{
    perror ("VIDIOC_ENUMSTD");
```



```
exit (EXIT_FAILURE);  
}
```

## 9. 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) 和用户指针。这里只讨论内存映射(memory mapping)。

### 9.1 向设备申请缓冲区 VIDIOC\_REQBUFS

相关函数：

```
[cpp] view plain copy print ? ⓘ  
int ioctl(int fd, int request, struct v4l2_requestbuffers *argp);
```

相关结构体：

```
[cpp] view plain copy print ? ⓘ  
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];  
};
```

注：

```
enum v4l2_memory  
{  
    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

```
[cpp] view plain copy print ? ⓘ  
struct v4l2_requestbuffers req;  
req.count = 4;  
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;  
req.memory = V4L2_MEMORY_MMAP;  
ioctl(fd,VIDIOC_REQBUFS,&req);
```

### 9.2 获取缓冲帧的地址，长度：VIDIOC\_QUERYBUF

相关函数：

```
[cpp] view plain copy print ? ⓘ  
int ioctl(int fd, int request, struct v4l2_buffer *argp);
```

相关结构体:

```
[cpp] view plain copy print ?
struct v4l2_buffer
{
    u32 index; //buffer 序号

    enum v4l2_buf_type type; //buffer 类型

    u32 bytesused; //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;
};
```

9.3 内存映射MMAP 及定义一个结构体来映射每个缓冲帧。 相关结构体:

```
[cpp] view plain copy print ?
struct buffer
{
    void* start;

    unsigned int length;
}*buffers;
```

相关函数:

```
[cpp] view plain copy print ?
#include <sys/mman.h>

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)

相关函数:

```
[cpp] view plain copy print ?
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。

```
[cpp] view plain copy print ?
buffers = (buffer*)calloc (req.count, sizeof (*buffers));

if (!buffers)
{
    // 映射
    fprintf (stderr, "Out of memory/n");
    exit (EXIT_FAILURE);
}

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);
    }
}
```

## 10. 缓冲区处理好之后，就可以开始获取数据了

### 10.1 启动 或 停止数据流 VIDIOC\_STREAMON，VIDIOC\_STREAMOFF

```
[cpp] view plain copy print ?
int ioctl(int fd, int request, const int *argp);
```

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

10.2 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

```
[cpp] view plain copy print ?
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

```
[cpp] view plain copy print ?
unsigned int i;

enum v4l2_buf_type type;

for (i = 0; i < 4; ++i) // 将缓冲帧放入队列
{
```

```

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

```

[cpp] view plain copy print ?
struct v4l2_buffer buf; CLEAR (buf);

buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

buf.memory = V4L2_MEMORY_MMAP;

ioctl (fd, VIDIOC_DQBUF, &buf); // 从缓冲区取出一个缓冲帧

process_image (buffers[buf.index].start); //

ioctl (fd,VIDIOC_QBUF&buf); //

```

附官方 v4l2 video capture example

```

[cpp] view plain copy print ?
/*
 * V4L2 video capture example
 *
 * This program can be used and distributed without restrictions.
 *
 * This program is provided with the V4L2 API
 * see http://linuxtv.org/docs.php for more information
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include <getopt.h>          /* getopt_long() */

#include <fcntl.h>          /* low-level i/o */
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

#include <linux/videodev2.h>

#define CLEAR(x) memset(&(x), 0, sizeof(x))

enum io_method {
    IO_METHOD_READ,
    IO_METHOD_MMAP,
    IO_METHOD_USERPTR,
};

struct buffer {
    void *start;
    size_t length;
};

static char *dev_name;
static enum io_method io = IO_METHOD_MMAP;
static int fd = -1;
static struct buffer *buffers;
static unsigned int n_buffers;
static int out_buf;
static int force_format;
static int frame_count = 70;

/*
 *
 *
 *
 *
 */
static void errno_exit(const char *s)
{
    fprintf(stderr, "%s error %d, %s\n", s, errno, strerror(errno));
    exit(EXIT_FAILURE);
}

```

```

static int xioctl(int fh, int request, void *arg)
{
    int r;

    do {
        r = ioctl(fh, request, arg);
    } while (-1 == r && EINTR == errno);

    return r;
}

static void process_image(const void *p, int size)
{
    if (out_buf)
        fwrite(p, size, 1, stdout);

    fflush(stderr);
    fprintf(stderr, ".");
    fflush(stdout);
}

static int read_frame(void)
{
    struct v4l2_buffer buf;
    unsigned int i;

    switch (io) {
    case IO_METHOD_READ:
        if (-1 == read(fd, buffers[0].start, buffers[0].length)) {
            switch (errno) {
                case EAGAIN:
                    return 0;

                case EIO:
                    /* Could ignore EIO, see spec. */

                    /* fall through */

                default:
                    errno_exit("read");
            }
        }

        process_image(buffers[0].start, buffers[0].length);
        break;

    case IO_METHOD_MMAP:
        CLEAR(buf);

        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_MMAP;

        if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
            switch (errno) {
                case EAGAIN:
                    return 0;

                case EIO:
                    /* Could ignore EIO, see spec. */

                    /* fall through */

                default:
                    errno_exit("VIDIOC_DQBUF");
            }
        }

        assert(buf.index < n_buffers);

        process_image(buffers[buf.index].start, buf.bytesused);

        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
            errno_exit("VIDIOC_QBUF");
        break;

    case IO_METHOD_USERPTR:
        CLEAR(buf);

        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_USERPTR;

        if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
            switch (errno) {
                case EAGAIN:
                    return 0;

                case EIO:
                    /* Could ignore EIO, see spec. */

                    /* fall through */

                default:
                    errno_exit("VIDIOC_DQBUF");
            }
        }

        for (i = 0; i < n_buffers; ++i)
            if (buf.m.userptr == (unsigned long)buffers[i].start
                && buf.length == buffers[i].length)
                break;

        assert(i < n_buffers);

        process_image((void *)buf.m.userptr, buf.bytesused);

        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
            errno_exit("VIDIOC_QBUF");
        break;
    }
}

```

```

    }

    return 1;
}

/* two operations
 * step1 : delay
 * step2 : read frame
 */
static void mainloop(void)
{
    unsigned int count;

    count = frame_count;

    while (count-- > 0) {
        for (;;) {
            fd_set fds;
            struct timeval tv;
            int r;

            FD_ZERO(&fds);
            FD_SET(fd, &fds);

            /* Timeout. */
            tv.tv_sec = 2;
            tv.tv_usec = 0;

            r = select(fd + 1, &fds, NULL, NULL, &tv);

            if (-1 == r) {
                if (EINTR == errno)
                    continue;
                errno_exit("select");
            }

            if (0 == r) {
                fprintf(stderr, "select timeout\n");
                exit(EXIT_FAILURE);
            }

            if (read_frame())
                break;
            /* EAGAIN - continue select loop. */
        }
    }
}

/*
 * one operation
 * step1 : VIDIOC_STREAMOFF
 */
static void stop_capturing(void)
{
    enum v4l2_buf_type type;

    switch (io) {
    case IO_METHOD_READ:
        /* Nothing to do. */
        break;

    case IO_METHOD_MMAP:
    case IO_METHOD_USERPTR:
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
            errno_exit("VIDIOC_STREAMOFF");
        break;
    }
}

/* tow operations
 * step1 : VIDIOC_QBUF(insert buffer to queue)
 * step2 : VIDIOC_STREAMOFF
 */
static void start_capturing(void)
{
    unsigned int i;
    enum v4l2_buf_type type;

    switch (io) {
    case IO_METHOD_READ:
        /* Nothing to do. */
        break;

    case IO_METHOD_MMAP:
        for (i = 0; i < n_buffers; ++i) {
            struct v4l2_buffer buf;

            CLEAR(buf);
            buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
            buf.memory = V4L2_MEMORY_MMAP;
            buf.index = i;

            if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                errno_exit("VIDIOC_QBUF");
        }
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
            errno_exit("VIDIOC_STREAMON");
        break;

    case IO_METHOD_USERPTR:
        for (i = 0; i < n_buffers; ++i) {
            struct v4l2_buffer buf;

            CLEAR(buf);
            buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
            buf.memory = V4L2_MEMORY_USERPTR;
            buf.index = i;
            buf.m.userptr = (unsigned long)buffers[i].start;

```

```

        buf.length = buffers[i].length;

        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
            errno_exit("VIDIOC_QBUF");
    }
    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
        errno_exit("VIDIOC_STREAMON");
    break;
}

/* two operations
 * step1 : munmap buffers
 * step2 : free buffers
 */
static void uninit_device(void)
{
    unsigned int i;

    switch (io) {
    case IO_METHOD_READ:
        free(buffers[0].start);
        break;

    case IO_METHOD_MMAP:
        for (i = 0; i < n_buffers; ++i)
            if (-1 == munmap(buffers[i].start, buffers[i].length))
                errno_exit("munmap");

        break;

    case IO_METHOD_USERPTR:
        for (i = 0; i < n_buffers; ++i)
            free(buffers[i].start);

        break;
    }

    free(buffers);
}

static void init_read(unsigned int buffer_size)
{
    buffers = calloc(1, sizeof(*buffers));

    if (!buffers) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }

    buffers[0].length = buffer_size;
    buffers[0].start = malloc(buffer_size);

    if (!buffers[0].start) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }
}

static void init_mmap(void)
{
    struct v4l2_requestbuffers req;

    CLEAR(req);

    req.count = 4;
    req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    req.memory = V4L2_MEMORY_MMAP;

    if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
        if (EINVAL == errno) {
            fprintf(stderr, "%s does not support "
                "memory mapping\n", dev_name);
            exit(EXIT_FAILURE);
        } else {
            errno_exit("VIDIOC_REQBUFS");
        }
    }

    if (req.count < 2) {
        fprintf(stderr, "Insufficient buffer memory on %s\n",
            dev_name);
        exit(EXIT_FAILURE);
    }

    buffers = calloc(req.count, sizeof(*buffers));

    if (!buffers) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }

    for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
        struct v4l2_buffer buf;

        CLEAR(buf);

        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_MMAP;
        buf.index = n_buffers;

        if (-1 == xioctl(fd, VIDIOC_QUERYBUF, &buf))
            errno_exit("VIDIOC_QUERYBUF");

        buffers[n_buffers].length = buf.length;
        buffers[n_buffers].start =
            mmap(NULL /* start anywhere */,
                buf.length,
                PROT_READ | PROT_WRITE /* required */,
                MAP_SHARED /* recommended */);
    }
}

```

```

        fd, buf.m.offset);

        if (MAP_FAILED == buffers[n_buffers].start)
            errno_exit("mmap");
    }
}

static void init_userp(unsigned int buffer_size)
{
    struct v4l2_requestbuffers req;

    CLEAR(req);

    req.count = 4;
    req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    req.memory = V4L2_MEMORY_USERPTR;

    if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
        if (EINVAL == errno) {
            fprintf(stderr, "%s does not support "
                "user pointer i/o\n", dev_name);
            exit(EXIT_FAILURE);
        } else {
            errno_exit("VIDIOC_REQBUFS");
        }
    }

    buffers = calloc(4, sizeof(*buffers));

    if (!buffers) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }

    for (n_buffers = 0; n_buffers < 4; ++n_buffers) {
        buffers[n_buffers].length = buffer_size;
        buffers[n_buffers].start = malloc(buffer_size);

        if (!buffers[n_buffers].start) {
            fprintf(stderr, "Out of memory\n");
            exit(EXIT_FAILURE);
        }
    }
}

/* five operations
 * step1 : cap :query camera's capability and check it(is a video device? is it support read? is it su
 * step2 : cropcap:set cropcap's type and get cropcap by VIDIOC_CROPCAP
 * step3 : set crop parameter by VIDIOC_S_CROP (such as frame type and angle)
 * step4 : set fmt
 * step5 : mmap
 */
static void init_device(void)
{
    struct v4l2_capability cap;
    struct v4l2_cropcap cropcap;
    struct v4l2_crop crop;
    struct v4l2_format fmt;
    unsigned int min;

    if (-1 == xioctl(fd, VIDIOC_QUERYCAP, &cap)) {
        if (EINVAL == errno) {
            fprintf(stderr, "%s is no V4L2 device\n",
                dev_name);
            exit(EXIT_FAILURE);
        } else {
            errno_exit("VIDIOC_QUERYCAP");
        }
    }

    if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
        fprintf(stderr, "%s is no video capture device\n",
            dev_name);
        exit(EXIT_FAILURE);
    }

    switch (io) {
    case IO_METHOD_READ:
        if (!(cap.capabilities & V4L2_CAP_READWRITE)) {
            fprintf(stderr, "%s does not support read i/o\n",
                dev_name);
            exit(EXIT_FAILURE);
        }
        break;

    case IO_METHOD_MMAP:
    case IO_METHOD_USERPTR:
        if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
            fprintf(stderr, "%s does not support streaming i/o\n",
                dev_name);
            exit(EXIT_FAILURE);
        }
        break;
    }

    /* Select video input, video standard and tune here. */

    CLEAR(cropcap);

    cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    /* if device support cropcap's type then set crop */
    if (0 == xioctl(fd, VIDIOC_CROPCAP, &cropcap)) {
        crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        crop.c = cropcap.defrect; /* reset to default */

        if (-1 == xioctl(fd, VIDIOC_S_CROP, &crop)) {
            switch (errno) {

```



```

        case EINVAL:
            /* Cropping not supported. */
            break;
        default:
            /* Errors ignored. */
            break;
    }
} else {
    /* Errors ignored. */
}

CLEAR(fmt);

fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (force_format) {
    fmt.fmt.pix.width      = 640;
    fmt.fmt.pix.height     = 480;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUVV;
    fmt.fmt.pix.field      = V4L2_FIELD_INTERLACED;

    if (-1 == ioctl(fd, VIDIOC_S_FMT, &fmt))
        errno_exit("VIDIOC_S_FMT");

    /* Note VIDIOC_S_FMT may change width and height. */
} else {
    /* Preserve original settings as set by v4l2-ctl for example */
    if (-1 == ioctl(fd, VIDIOC_G_FMT, &fmt))
        errno_exit("VIDIOC_G_FMT");
}

/* Buggy driver paranoia. */
min = fmt.fmt.pix.width * 2;
if (fmt.fmt.pix.bytesperline < min)
    fmt.fmt.pix.bytesperline = min;
min = fmt.fmt.pix.bytesperline * fmt.fmt.pix.height;
if (fmt.fmt.pix.sizeimage < min)
    fmt.fmt.pix.sizeimage = min;

switch (io) {
case IO_METHOD_READ:
    init_read(fmt.fmt.pix.sizeimage);
    break;

case IO_METHOD_MMAP:
    init_mmap();
    break;

case IO_METHOD_USERPTR:
    init_userp(fmt.fmt.pix.sizeimage);
    break;
}
}

/*
 * close (fd)
 */
static void close_device(void)
{
    if (-1 == close(fd))
        errno_exit("close");

    fd = -1;
}

/* three operations
 * step 1 : check dev_name and st_mode
 * step 2 : open(device)
 */
static void open_device(void)
{
    struct stat st;

    if (-1 == stat(dev_name, &st)) {
        fprintf(stderr, "Cannot identify '%s': %d, %s\n",
            dev_name, errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    if (!S_ISCHR(st.st_mode)) {
        fprintf(stderr, "%s is no device\n", dev_name);
        exit(EXIT_FAILURE);
    }

    fd = open(dev_name, O_RDWR /* required */ | O_NONBLOCK, 0);

    if (-1 == fd) {
        fprintf(stderr, "Cannot open '%s': %d, %s\n",
            dev_name, errno, strerror(errno));
        exit(EXIT_FAILURE);
    }
}

static void usage(FILE *fp, int argc, char **argv)
{
    fprintf(fp,
        "Usage: %s [options]\n\n"
        "Version 1.3\n\n"
        "Options:\n"
        "-d | --device name    Video device name [%s]\n\n"
        "-h | --help           Print this message\n\n"
        "-m | --mmap           Use memory mapped buffers [default]\n\n"
        "-r | --read           Use read() calls\n\n"
        "-u | --userp          Use application allocated buffers\n\n"
        "-o | --output         Outputs stream to stdout\n\n"
        "-f | --format         Force format to 640x480 YUVV\n\n"
        "-c | --count          Number of frames to grab [%i]\n\n"
        "",

```

```

        argv[0], dev_name, frame_count);
    }

    static const char short_options[] = "d:hmrufc:";

    static const struct option
    long_options[] = {
        { "device", required_argument, NULL, 'd' },
        { "help", no_argument, NULL, 'h' },
        { "mmap", no_argument, NULL, 'm' },
        { "read", no_argument, NULL, 'r' },
        { "userp", no_argument, NULL, 'u' },
        { "output", no_argument, NULL, 'o' },
        { "format", no_argument, NULL, 'f' },
        { "count", required_argument, NULL, 'c' },
        { 0, 0, 0, 0 }
    };

    int main(int argc, char **argv)
    {
        dev_name = "/dev/video4";

        for (;;) {
            int idx;
            int c;

            c = getopt_long(argc, argv,
                           short_options, long_options, &idx);

            if (-1 == c)
                break;

            switch (c) {
            case 0: /* getopt_long() flag */
                break;

            case 'd':
                dev_name = optarg;
                break;

            case 'h':
                usage(stdout, argc, argv);
                exit(EXIT_SUCCESS);

            case 'm':
                io = IO_METHOD_MMAP;
                break;

            case 'r':
                io = IO_METHOD_READ;
                break;

            case 'u':
                io = IO_METHOD_USERPTR;
                break;

            case 'o':
                out_buf++;
                break;

            case 'f':
                force_format++;
                break;

            case 'c':
                errno = 0;
                frame_count = strtoul(optarg, NULL, 0);
                if (errno)
                    errno_exit(optarg);
                break;

            default:
                usage(stderr, argc, argv);
                exit(EXIT_FAILURE);
            }
        }

        open_device();
        init_device();
        start_capturing();
        mainloop();
        stop_capturing();
        uninit_device();
        close_device();
        fprintf(stderr, "\n");
        return 0;
    }

```