东月之神

在单纯的观念里面，生命就容易变得比较深刻！

个人资料

东月之神

访问： 296186次
积分： 3811
等级： BLOG 5
排名： 第3877名

原创： 119篇
转载： 0篇
译文： 0篇
评论： 207条

个性签名

别驻足，梦想要不停追逐；别认输，熬过黑夜才有日出。要记住，成功就在下一步；路很苦，汗水是最美的书！

和菜鸟一起学linux之V4L2摄像头应用流程

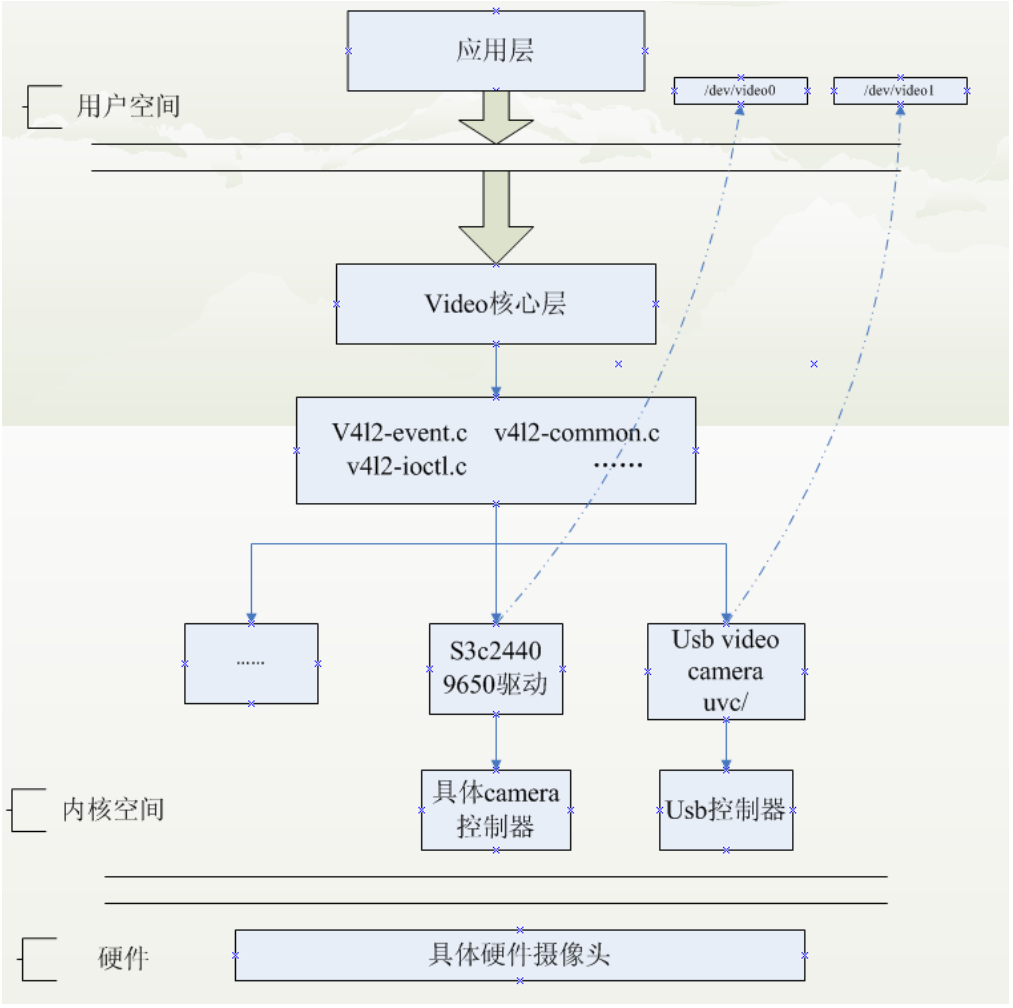
分类： linux 2012-11-16 11:52 7997人阅读 评论(11) 收藏 举报

对于v4l2，上[illegible]收音机驱动的时候用过，其他也就只是用i2c配置一些寄存器就可以了。那时[illegible]了解了，把收音机当作v4l2的设备后会在/dev目录下生成一个radio的节点。[illegible]操作了。后来就没怎么接触了。这周，需要调试下usb的摄像头。因为有问[illegible]要跟进，于是也就要开始学习下linux的v4l2了。看到一篇很不错的文章，下面参考这篇文章，加上自己的一些见解，做一些总结把。

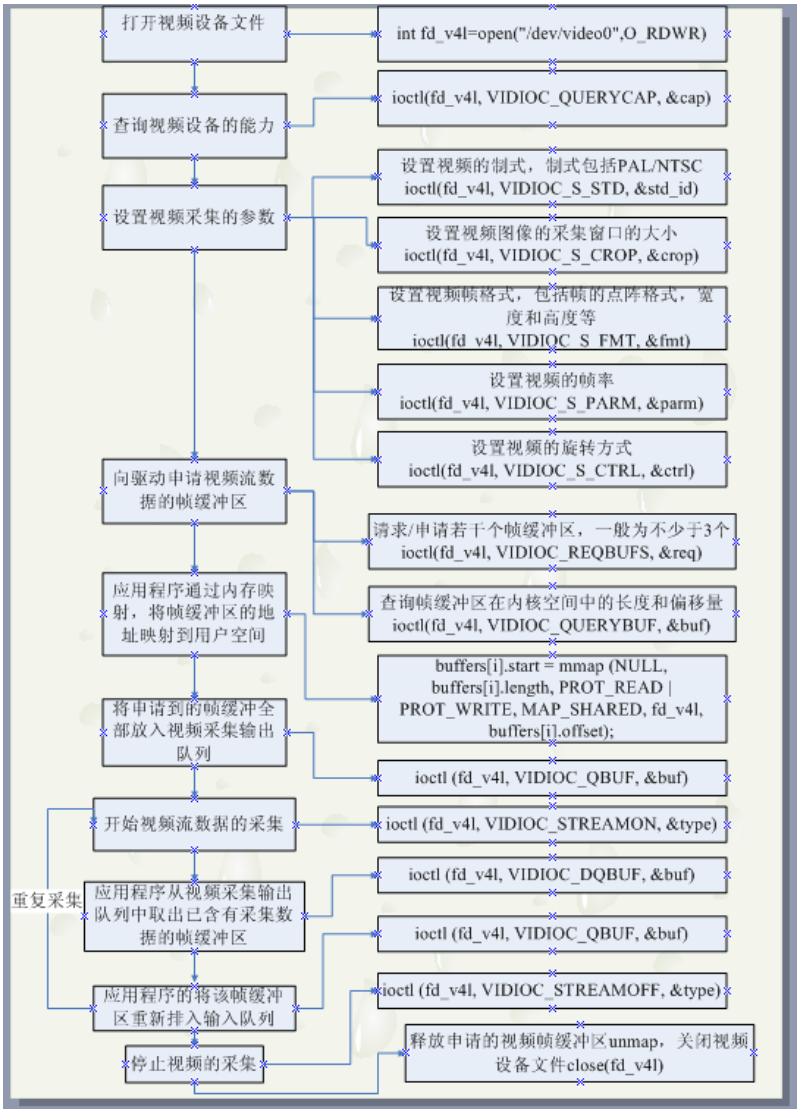
Video for Linu[illegible]4Linux2)简称V4L2，是V4L的改进版。V4L2是linux操作系统下用于采集图片、视频和音频数据的API接口，配合适当的视频采集设备和相应的驱动程序，可以实现图片、视频、音频等的采集。在远程会议、可视电话、视频监控系统和嵌入式多媒体终端中都有广泛的应用。

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/video0。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。

而摄像头所用的主要是capature了，视频的捕捉，具体linux的调用可以参考下图。



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。
具体的程序实现流程可以参考下面的流程图:

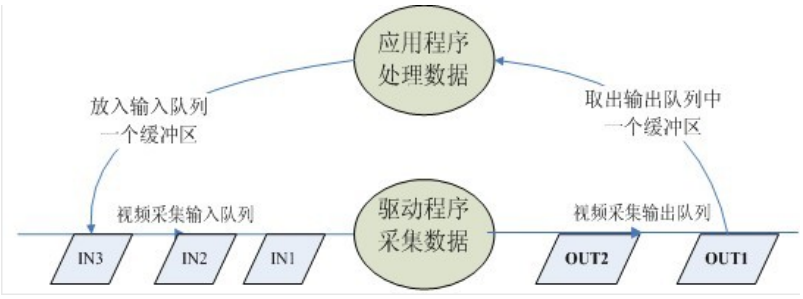


其实其他的都比较简单，就是通过ioctl这个接口去设置一些参数。最主要的就是buf管理。他有一个或者多个输入队列和输出队列。

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，如图所示。



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

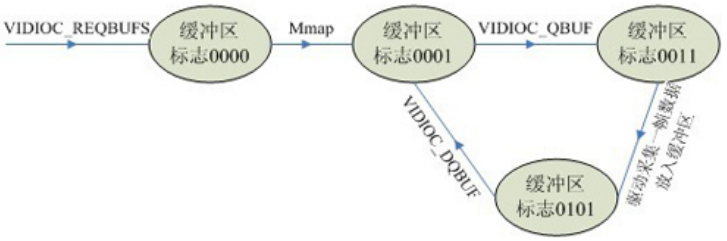
V4L2_BUF_FLAG_UNMAPPED 0B0000

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所示。



下面的程序注释的很好，就拿来参考下：

V4L2 编程
1. 定义
V4L2(Video ForLinux 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<fcntl.h>
int open(constchar *device_name, int flags);

#include <unistd.h>
int close(intfd);
例：

```
[html]
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：

```
[html]
int ioctl(intfd, int request, struct v4l2_capability *argp);
```

相关结构体：

```
[html]
structv4l2_capability
{
__u8 driver[16];      // 驱动名字
__u8 card[32];        // 设备名字
__u8bus_info[32]; // 设备在系统中的位置
__u32 version;        // 驱动版本号
__u32capabilities;  // 设备支持的操作
__u32reserved[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

```
[html]
```

```
structv4l2_capability cap;
ioctl(fd,VIDIOC_QUERYCAP,&cap);
printf("DriverName:%s/nCard Name:%s/nBus info:%s/nDriverVersion:%u.%u.%u/n",cap.driver,
(cap.version>>16)&0XFF,(cap.version>>8)&0XFF,cap.version&OXFF);
```

5. 帧格式：

```
[html]

VIDIOC_ENUM_FMT// 显示所有支持的格式
int ioctl(intfd, int request, struct v4l2_fmtdesc *argp);
structv4l2_fmtdesc
{
__u32 index;   // 要查询的格式序号，应用程序设置
enumv4l2_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
__u32pixelformat; // 格式
__u32reserved[4]; // 保留
};
```

例：显示所有支持的格式

```
[html]

structv4l2_fmtdesc fmtdesc;
fmtdesc.index=0;
fmtdesc.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
printf("Supportformat:/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

```
[html]

VIDIOC_TRY_FMT
int ioctl(intfd, int request, struct v4l2_format *argp);
structv4l2_format
{
enumv4l2_buf_type type;// 帧类型，应用程序设置
union fmt
{
structv4l2_pix_format pix;// 视频设备使用
structv4l2_window win;
structv4l2_vbi_format vbi;
structv4l2_sliced_vbi_format sliced;
__u8raw_data[200];
};
};
```

```
[html]

structv4l2_pix_format
{
__u32 width;  // 帧宽，单位像素
__u32 height;  // 帧高，单位像素
__u32pixelformat; // 帧格式
enum v4l2_fieldfield;
__u32bytesperline;
```

```
__u32 sizeimage;
enumv4l2_colorspace colorspace;
__u32 priv;
};
```

例：显示当前帧的相关信息

```
[html]

structv4l2_format fmt;
fmt.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
ioctl(fd,VIDIOC_G_FMT,&fmt);
printf("Currentdata format information:
/n/twidth:%d/n/theight:%d/n",fmt.fmt.width,fmt.fmt.height);
structv4l2_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

```
[html]

structv4l2_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("notsupport format RGB32!/n");
```

6. 图像的缩放

```
[html]

VIDIOC_CROPCAP
int ioctl(int fd,int request, struct v4l2_cropcap *argp);
structv4l2_cropcap
{
enumv4l2_buf_type type;// 应用程序设置
struct v4l2_rectbounds;//     最大边界
struct v4l2_rectdefrect;// 默认值
structv4l2_fract pixelaspect;
};
```

// 设置缩放

```
[html]

VIDIOC_G_CROP,VIDIOC_S_CROP
int ioctl(intfd, int request, struct v4l2_crop *argp);
int ioctl(intfd, int request, const struct v4l2_crop *argp);
struct v4l2_crop
{
enumv4l2_buf_type type;// 应用程序设置
struct v4l2_rectc;
}
```

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(memorymapping) ，用户指针。这里只讨论 memorymapping.
// 向设备申请缓冲区

```
[html]

VIDIOC_REQBUFS
int ioctl(intfd, int request, struct v4l2_requestbuffers *argp);
structv4l2_requestbuffers
{
__u32 count;  // 缓冲区内缓冲帧的数目
enumv4l2_buf_type type;     // 缓冲帧数据格式
enum v4l2_memorymemory;       // 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

```
[html]

structv4l2_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(intfd, int request, struct v4l2_buffer *argp);

```
[html]

structv4l2_buffer
{
__u32 index;   //buffer 序号
enumv4l2_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_fieldfield;
struct timevaltimestamp;// 获取第一个字节时的系统时间
structv4l2_timecode timecode;
__u32 sequence;// 队列中的序号
enum v4l2_memorymemory;//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 longuserptr;
};
__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。

```
[html]

Struct buffer
{
void* start;
unsigned intlength;
}*buffers;
```

#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);

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。

```
[html]
buffers =(buffer*)calloc (req.count, sizeof (*buffers));
if (!buffers) {
fprintf (stderr,"Out of memory/n");
exit(EXIT_FAILURE);
}
```

// 映射

```
[html]
for (unsignedint n_buffers = 0; n_buffers < req.count; ++n_buffers) {
struct v4l2_bufferbuf;
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, b
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

```
[html]
// 启动/ 停止数据流
VIDIOC_STREAMON,VIDIOC_STREAMOFF
int ioctl(intfd, 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(intfd, int request, struct v4l2_buffer *argp);
```

例：把四个缓冲帧放入队列，并启动数据流

```
[html]
unsigned int i;
enum v4l2_buf_typetype;
// 将缓冲帧放入队列
for (i = 0; i< 4; ++i)
{
structv4l2_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);
```

例：获取一帧并处理

```
[html]

structv4l2_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);
```

至于驱动的实现，可以参考内核中，我是用usb摄像头的，所以，其实现都是好的。主要就是应用程序的实现了。驱动都哦在uvc目录下面，这个待理解。

上一篇 和菜鸟一起学c之gcc编译过程及其常用编译选项
下一篇 和菜鸟一起学电子小玩意之四轴飞行器原理

查看评论

8楼 william9876 2015-05-13 17:23发表

给你赞一个

7楼 zs_lgx1 2014-11-17 15:37发表

楼主请问一下，我的平板是原道的M80，插入了usb摄像头但是在/dev没有生成设备节点videoX呢。连自动摄像头的videoX