转 嵌入式Linux下Camera编程--V4L2

发表于2年前(2013-11-13 16:55) 阅读（716）| 评论（0）1人收藏此文章, 我要收藏
赞0
v4l2 ARM Linux Usb Camera

最近有个需求，要在ARM **Linux**上实现USB Camera 拍照功能。

0. 背景知识：

首先要确认的是，Kernel是否支持USB Camera。因为**Linux**下，USB协议除了电气协议和标准，还有很多Class。这些Class就是为了支持和定义某一类设备接口和交互数据格式。只要符合这类标准，则不同厂商的USB设备，不需要特定的driver就能在**Linux**下使用。

例如：USB Input class,则使所有输入设备都可以直接使用。还有类似Audio Class， Pring Class，Mass Storage Class， video class等。

其中Video Class 就是我们常说的UVC（USB Video Class）. 只要USB Camera符合UVC标准。理论上在2.6 Kernel **Linux** 就可以正常使用。

网络上有人谈到怎样判断是否UVC Camera设备：
#lsusb
Bus 001 Device 010: ID **046d:0825** Logitech, Inc.

#lsusb -d 046d:0825 -v | grep "14 Video"
如果出现：

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video

bInterfaceClass 14 Video
则说明是支持UVC.

1. Kernel配置：
Device Drivers ---> <*> Multimedia support ---> **<M> Video For Linux**
Device Drivers ---> <*> Multimedia support ---> [*] Video capture adapters ---> [*] **V4L** USB devices ---> **<M> USB Video Class (UVC)**

--- **V4L** USB devices ：这里还有很多特定厂商的driver.可供选择。

分析：
"USB Video Class (UVC)"：对应的driver是：uvcvideo.ko
"Video For **Linux**": 对应driver是：videodev.ko

安装driver顺序如下：
insmod **v4l**1_compat.ko
insmod videodev.ko
insmod uvcvideo.ko

driver会创建一个或多个主设备号为81，次设备号：0-255的设备。
除了camera会创建为：/dev/videoX 之外，还有VBI设备-/dev/vbiX. Radio设备--/dev/radioX.

2. V4L2一些概念：
2.1：Video Input and Output:
video input and output是指device物理连接。
只有video 和VBI capture拥有input.
Radio设备则没有video input 和output.

2.2: Video Standards:
Video Device支持一个或多个Video 标准。

3. 使用V4L2编程：
使用**V4L**2(Video for **Linux** 2) API的过程大致如下：

Opening the device

Changing device properties, selecting a video and audio input, video standard, picture brightness a. o.

Negotiating a data format

Negotiating an input/output method

The actual input/output loop

Closing the device

3.1：打开设备：
fd = open ("/dev/video0", O_RDWR, 0); //以阻塞模式打开设想头

3.2: 查询设备能力：Querying Capabilities:
因为**V4L**2可以对多种设备编程，所以并不是所有API可以对所有设备编程，哪怕是同类型的设备，使用ioctl--VIDIOC_QUERYCAP去询问支持什么功能。

struct **v4l2_capability** cap;

rel = ioctl(fdUsbCam, VIDIOC_QUERYCAP, &cap);
if(rel != 0)
{

perror("ioctl VIDIOC_QUERYCAP");

return -1;

}

结构体如下：

struct **v4l**2_capability

{

__u8 driver[16];

__u8 card[32];

__u8 bus_info[32];

__u32 version;

__u32 capabilities;

__u32 reserved[4];

};

这里面最重要的是：capabilities:
头文件 **linux**/videodev2.h和kernel头文件 **linux**/videodev2.h中都有描述：

#define **V4L2_CAP_VIDEO_CAPTURE** 0x00000001

#define **V4L**2_CAP_VIDEO_OUTPUT 0x00000002

#define **V4L**2_CAP_VIDEO_OVERLAY 0x00000004

#define **V4L**2_CAP_VBI_CAPTURE 0x00000010

#define **V4L**2_CAP_VBI_OUTPUT 0x00000020

#define **V4L**2_CAP_SLICED_VBI_CAPTURE 0x00000040

#define **V4L**2_CAP_SLICED_VBI_OUTPUT 0x00000080

#define **V4L**2_CAP_RDS_CAPTURE 0x00000100

#define **V4L**2_CAP_VIDEO_OUTPUT_OVERLAY 0x00000200

#define **V4L**2_CAP_HW_FREQ_SEEK 0x00000400

#define **V4L**2_CAP_RDS_OUTPUT 0x00000800

#define **V4L**2_CAP_TUNER 0x00010000

#define **V4L**2_CAP_AUDIO 0x00020000

#define **V4L**2_CAP_RADIO 0x00040000

#define **V4L**2_CAP_MODULATOR 0x00080000

#define **V4L**2_CAP_READWRITE 0x01000000

#define **V4L**2_CAP_ASYNCIO 0x02000000

#define **V4L2_CAP_STREAMING** 0x04000000

这里要说到VBI，Vertical Blanking Interval的缩写。电视信号包括一部分非可视信号，它不传送可视信息，因此被称为VBI(垂直消隐期间)。VBI可以用于传送其他信息，通常是一种专用字幕信号
这和Blog 重显率中所说暗合。

在这里，**V4L2_CAP_VIDEO_CAPTURE** 说明设备是个图像采集设备，**V4L2_CAP_STREAMING** 说明是个Streaming设备。
通常，摄像头都支持以上两个能力。

3.3：查询当前捕获格式：

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

if (ioctl(fdUsbCam, VIDIOC_G_FMT, &fmt) < 0)
 {
   printf("get format failed\n");
   return -1;
 }
```

注意，此处，fmt是个in/out参数。
参见Kernel代码 **v4l**2_ioctl.c中。此ioctl，它会首先判断
fmt.type.
type类型和含义如下：

V4L2_BUF_TYPE_VIDEO_CAPTURE ：vid-cap

V4L2_BUF_TYPE_VIDEO_OVERLAY ：vid-overlay

V4L2_BUF_TYPE_VIDEO_OUTPUT :vid-out

V4L2_BUF_TYPE_VBI_CAPTURE :vbi-cap

V4L2_BUF_TYPE_VBI_OUTPUT : vbi-out

V4L2_BUF_TYPE_SLICED_VBI_CAPTURE : sliced-vbi-cap

V4L2_BUF_TYPE_SLICED_VBI_OUTPUT : sliced-vbi-out

V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY : vid-out-overlay
咱们是使用Video Cam的。所以用**V4L2_BUF_TYPE_VIDEO_CAPTURE**

```
struct v4l2_format
{
 enum v4l2_buf_type type;
 union
 {
 struct v4l2_pix_format pix;
 struct v4l2_window win;
 struct v4l2_vbi_format vbi;
 struct v4l2_sliced_vbi_format sliced;
 __u8 raw_data[200];
 } fmt;
};
```

我们得到的信息在**v4l2_pix_format**中。
你可以看到，宽，高，像素格式。
其中像素格式包括：

```
#define V4L2_PIX_FMT_RGB332 v4l2_fourcc('R','G','B','1')
#define V4L2_PIX_FMT_RGB555 v4l2_fourcc('R','G','B','O')
#define V4L2_PIX_FMT_RGB565 v4l2_fourcc('R','G','B','P')
#define V4L2_PIX_FMT_RGB555X v4l2_fourcc('R','G','B','Q')
#define V4L2_PIX_FMT_RGB565X v4l2_fourcc('R','G','B','R')
#define V4L2_PIX_FMT_BGR24 v4l2_fourcc('B','G','R','3')
#define V4L2_PIX_FMT_RGB24 v4l2_fourcc('R','G','B','3')
#define V4L2_PIX_FMT_BGR32 v4l2_fourcc('B','G','R','4')
#define V4L2_PIX_FMT_RGB32 v4l2_fourcc('R','G','B','4')
#define V4L2_PIX_FMT_GREY v4l2_fourcc('G','R','E','Y')
```

```
#define V4L2_PIX_FMT_YVU410 v4l2_fourcc('Y','V','U','9')
#define V4L2_PIX_FMT_YVU420 v4l2_fourcc('Y','V','1','2')
#define V4L2_PIX_FMT_YUYV v4l2_fourcc('Y','U','Y','V')
#define V4L2_PIX_FMT_UYVY v4l2_fourcc('U','Y','V','Y')
#define V4L2_PIX_FMT_YUV422P v4l2_fourcc('4','2','2','P')
#define V4L2_PIX_FMT_YUV411P v4l2_fourcc('4','1','1','P')
#define V4L2_PIX_FMT_Y41P v4l2_fourcc('Y','4','1','P')

#define V4L2_PIX_FMT_NV12 v4l2_fourcc('N','V','1','2')
#define V4L2_PIX_FMT_NV21 v4l2_fourcc('N','V','2','1')

#define V4L2_PIX_FMT_YUV410 v4l2_fourcc('Y','U','V','9')
#define V4L2_PIX_FMT_YUV420 v4l2_fourcc('Y','U','1','2')
#define V4L2_PIX_FMT_YYUV v4l2_fourcc('Y','Y','U','V')
#define V4L2_PIX_FMT_HI240 v4l2_fourcc('H','I','2','4')
#define V4L2_PIX_FMT_HM12 v4l2_fourcc('H','M','1','2')

#define V4L2_PIX_FMT_SBGGR8 v4l2_fourcc('B','A','8','1')

#define V4L2_PIX_FMT_MJPEG v4l2_fourcc('M','J','P','G')
#define V4L2_PIX_FMT_JPEG v4l2_fourcc('J','P','E','G')
#define V4L2_PIX_FMT_DV v4l2_fourcc('d','v','s','d')
#define V4L2_PIX_FMT_MPEG v4l2_fourcc('M','P','E','G')

#define V4L2_PIX_FMT_WNVA v4l2_fourcc('W','N','V','A')
#define V4L2_PIX_FMT_SN9C10X v4l2_fourcc('S','9','1','0')
#define V4L2_PIX_FMT_PWC1 v4l2_fourcc('P','W','C','1')
#define V4L2_PIX_FMT_PWC2 v4l2_fourcc('P','W','C','2')
#define V4L2_PIX_FMT_ET61X251 v4l2_fourcc('E','6','2','5')
```

fxxk,真TNND多。

请注意，此时取到的宽，高，像素格式均正确。但不知为何，bytesperline却为0。

3.4：设置当前捕获格式

```
fmt.fmt.pix.width =  640;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.pixelformat=V4L2_PIX_FMT_YUYV;
rel = ioctl(fdUsbCam, VIDIOC_S_FMT, &fmt);
if (rel < 0)
{
printf("\nSet format failed\n");
return -1;
}
```

此时，再取当前捕获格式，则一切正常。包括 **bytesperline**

3.5：读取Stream 设置。

```
struct v4l2_streamparm *setfps;
setfps=(struct v4l2_streamparm *) calloc(1, sizeof(struct v4l2_streamparm));
memset(setfps, 0, sizeof(struct v4l2_streamparm));
setfps->type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

 rel = ioctl(fdUsbCam, VIDIOC_G_PARM, setfps);
 if(rel == 0)
 {
   printf("\n  Frame rate:  %u/%u\n",
       setfps->parm.capture.timeperframe.denominator,
       setfps->parm.capture.timeperframe.numerator
       );
 }
 else
 {
   perror("Unable to read out current frame rate");
   return -1;
```

```
}
```

注意： ioctl(fdUsbCam, VIDIOC_G_PARM, setfps); 参数3也是i/o 参数。必须要首先其type.

```
struct v4l2_streamparm
{
 enum v4l2_buf_type type;
 union
 {
 struct v4l2_captureparm capture;
 struct v4l2_outputparm output;
 __u8 raw_data[200];
 } parm;
};
```

type字段描述的是在涉及的操作的类型。对于视频捕获设备，应该为**V4L**2_BUF_TYPE_VIDEO_CAPTURE。对于输出设备应该为**V4L**2_BUF_TYPE_VIDEO_OUTPUT。它的值也可以是**V4L**2_BUF_TYPE_PRIVATE，在这种情况下，raw_data字段用来传递一些私有的，不可移植的，甚至是不鼓励的数据给**内核**。

```
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,
  V4L2_BUF_TYPE_PRIVATE = 0x80,
};
```

咱们当然选用 **V4L2_BUF_TYPE_VIDEO_CAPTURE**

对于捕获设备而言，parm.capture字段是要关注的内容，这个结构体如下：

```
struct v4l2_captureparm
 {
    __u32           capability;
    __u32           capturemode;
    struct v4l2_fract  timeperframe;
    __u32           extendedmode;
    __u32          readbuffers;
    __u32           reserved[4];
 };
```

timeperframe字段用于指定想要使用的帧频率，它又是一个结构体：

```
 struct v4l2_fract {
    __u32  numerator;
    __u32  denominator;
 };
```

numerator 和denominator所描述的系数给出的是成功的帧之间的时间间隔。

numerator 分子， denominator 分母。主要表达每次帧之间时间间隔。 numerator/ denominator秒一帧。

3.6：设置Stream参数。(主要是采集帧数)

```
setfps->parm.capture.timeperframe.numerator=1;
setfps->parm.capture.timeperframe.denominator= 60;
  rel = ioctl(fdUsbCam, VIDIOC_S_PARM, setfps);
if(rel != 0)
  {
    printf("\nUnable to Set FPS");
    return -1;
  }
```

当然，setfps的其它项目，都是之前使用 VIDIOC_G_PARM取得的。

3.7：创建一组缓冲区(buf)

```
struct v4l2_requestbuffers rb;
memset(&rb, 0, sizeof(struct v4l2_requestbuffers));
rb.count = 3;
rb.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
rb.memory = V4L2_MEMORY_MMAP;
rel = ioctl(fdUsbCam, VIDIOC_REQBUFS, &rb);
if (rel < 0)
{
    printf("Unable to allocate buffers: %d.\n", errno);
    return -1;
}
```

其中参数rb为：struct v4l2_requestbuffers:

```
struct v4l2_requestbuffers
{
 __u32 count;
 enum v4l2_buf_type type;
 enum v4l2_memory memory;
 __u32 reserved[2];
};
```

type 字段描述的是完成的I/O操作的类型。通常它的值要么是视频获得设备的

V4L2_BUF_TYPE_VIDEO_CAPTURE ，要么是输出设备的V4L2_BUF_TYPE_VIDEO_OUTPUT

struct v4l2_memory:

```
enum v4l2_memory {
  V4L2_MEMORY_MMAP = 1,
  V4L2_MEMORY_USERPTR = 2,
  V4L2_MEMORY_OVERLAY = 3,
};
```

想要使用内存映谢的缓冲区，它将会把memory字段置为V4L2_MEMORY_MMAP，count置为它想要

使用的缓冲区的数目。

顺便看看USB TO Serail:
Device Drivers ---->[*] USB support ---> <M> USB Serial Converter support ---> <M> USB Prolific 2303 Single Port Serial Driver

USB Prolific 2303 Single Port Serial Driver是指出支持pl2303芯片的USB 2 serial.
pl2303.ko

USB Serial Converter support是基础driver. 对应usbserial.ko

注1：ioctl中常用的cmd.

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。

VIDIOC_G_PARM ：得到Stream信息。如帧数等。
VIDIOC_S_PARM:设置Stream信息。如帧数等。

注2：
如何判断某ioctl cmd所用参数类型：
例如：
ioctl-cmd: VIDIOC_QUERYCAP.
它的返回参数类型ioctl(fd, cmd, 参数)。
首先想到的是从kernel Source **v4l**2_ioctl.c中看。但这比较麻烦，又个简单办法：可以在video2dev.h中看到：
#define VIDIOC_QUERYCAP _IOR ('V', 0, struct **v4l**2_capability)
即使用cmd为 VIDIOC_QUERYCAP 时，参数为 struct **v4l**2_capability

原文地址：http://blog.sina.com.cn/s/blog_602f87700100znq7.html

- « 上一篇
- 下一篇 »

评论*0*

插入： 表情 开源软件 发表评论

关闭插入表情
关闭相关文章阅读

- 2014/04/22 Linux下使用V4L2读取获取拍照（获取...
- 2013/11/05 v4l2视频采集
- 2013/09/12 Porting the Linux Kernel to a Ne...
- 2014/11/14 【解答】arm架构的linux内核，软件...