Android USB Camera(1) : 调试记录

标签： MTK Android USB-Camera UVC V4L2

2016-11-07 19:00 1314人阅读 评论(4) 收藏 举报

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(?) [+]

1. 前言

前段时间调试了一个uvc摄像头，这里做下记录。硬件平台为mt6735，软件平台为**Android** 5.0

2. 底层配置

UVC全称是usb video class，一种usb视频规范。所有遵循uvc协议的摄像头都不需要安装额外的驱动，只需要一个通用驱动即可。**Linux**内核已经集成了uvc驱动，代码路径是kernel-3.10/drivers/media/usb/uvc/

2.1 打开配置

Linux内核需要打开以下配置来支持uvc设备

```
CONFIG_MEDIA_SUPPORT=y
CONFIG_MEDIA_CAMERA_SUPPORT=y
CONFIG_VIDEO_DEV=y
CONFIG_VIDEO_V4L2=y
CONFIG_VIDEOBUF2_CORE=y
CONFIG_VIDEOBUF2_MEMOPS=y
CONFIG_VIDEOBUF2_VMALLOC=y
CONFIG_MEDIA_USB_SUPPORT=y
CONFIG_USB_VIDEO_CLASS=y
```

MTK平台还需要额外打开otg配置

```
CONFIG_USB_MTK_OTG=y
CONFIG_USB_MTK_HDRC=y
CONFIG_USB_MTK_HDRC_HCD=y
```

插入摄像头，如果生成了/dev/video0设备节点，则证明uvc摄像头已经加载成功了。成功生成驱动节点后还需要为它添加权限

2.2 添加权限

在uevent.rc中加入

```
/dev/video0                0666    root        root
```

在system_app.te中加入

```
allow system_app video_device:chr_file { read write open getattr };
```

2.3 Debug

如果没有出现/dev/video0节点，需要先判断是否枚举成功。在shell终端cat相关的节点查询

```
cat /sys/kernel/debug/usb/devices
```

如果该摄像头枚举成功，则能找到对应的设备信息

```
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=1 Spd=480 MxCh=1
D:  Ver=2.00 Cls=00(>ifc) Sub=00 Prot=00 MxPS=64 #Cfgs=1
P:  Vendor=18EC ProdID=3399 Rev=0.00
S:  Manufacturer=ARKMICRO
S:  Product=USB PC CAMERA
```

如果枚举成功则需要判断当前的usb摄像头是不是遵循uvc协议的摄像头。将usb摄像头插到PC上(ubuntu**操作系统**)，通过”lsusb”命令查找是否有视频类接口信息

```
lsusb -d 18ec:3399 -v | grep "14 Video"
```

如果该摄像头遵循UVC协议，则会输出以下类似信息

```
bFunctionClass 14 Video

```

```
bInterfaceClass 14 Video
bInterfaceClass 14 Video
bInterfaceClass 14 Video
```

其中18ec:3399是摄像头的vid和pid，而14 video代表uvc规范

2.4 几个比较有用的调试命令

打开/关闭linux uvc driver log

```
echo 0xffff > /sys/module/uvcvideo/parameters/trace //打开
echo 0 > /sys/module/uvcvideo/parameters/trace //关闭
```

获取详细的usb设备描述符

```
lsusb -d 18ec:3399  - v
```

3. 上层应用

v4l2 - Video for Linux 2，是Linux内核中关于视频设备的内核驱动框架，为上层的访问底层的视频设备提供了统一的接口。同时是针对uvc免驱usb设备的编程框架，主要用于采集usb摄像头等。

MTK标准的Camera并没有采用v4l2框架，所以需要在jni层实现基本的v4l2视频采集流程。

3.1 操作流程

在v4l2编程中，一般使用ioctl函数来对设备进行操作：

```
extern int ioctl (int __fd, unsigned long int __request, …) __THROW;
```

__fd：设备的ID，例如用open函数打开/dev/video0后返回的cameraFd；

__request：具体的命令标志符。

在进行V4L2开发中，一般会用到以下的命令标志符：

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：设置当前驱动的视频格式

VIDIOC_G_FMT：读取当前驱动的视频格式

VIDIOC_TRY_FMT：验证当前驱动的视频格式

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：把数据放回缓存队列

VIDIOC_DQBUF：把数据从缓存中读取出来

VIDIOC_STREAMON：开始视频采集

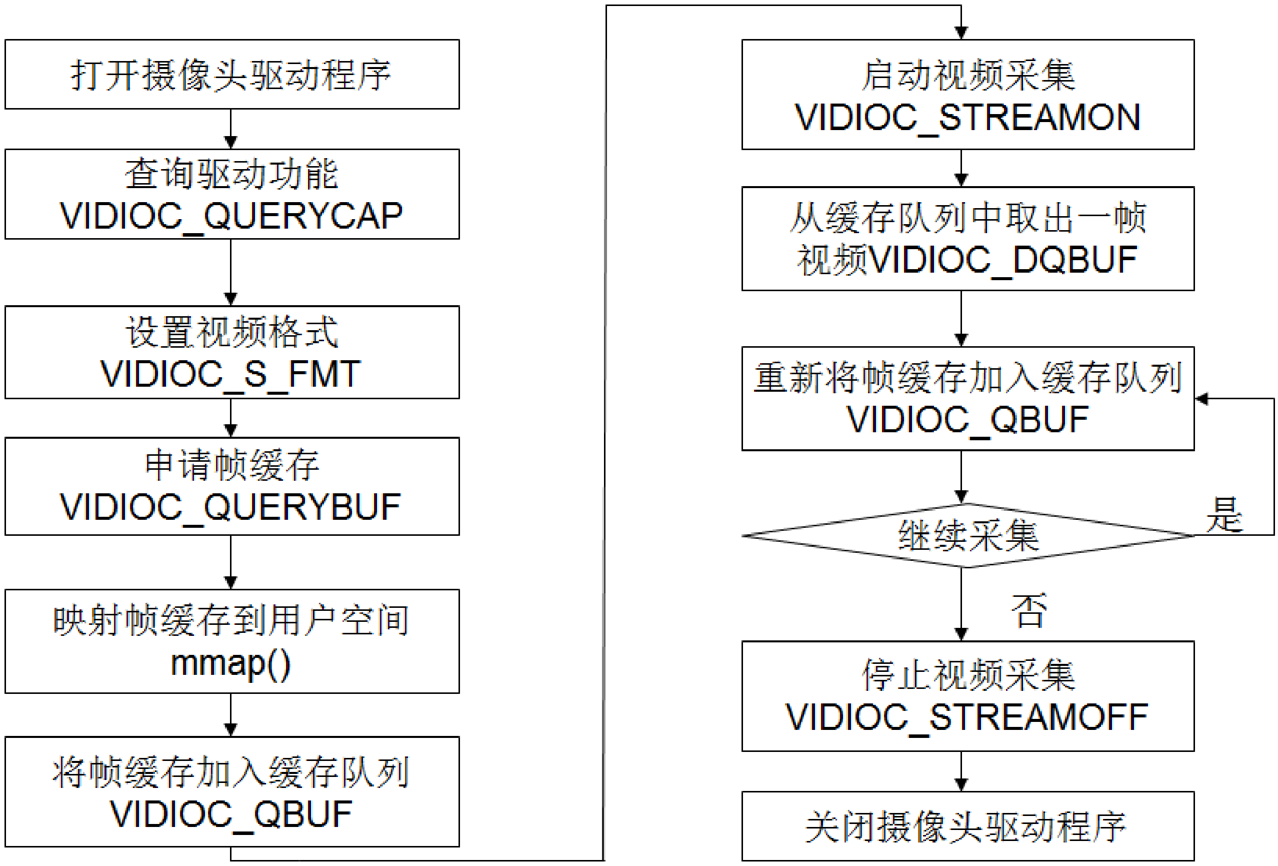
VIDIOC_STREAMOFF：结束视频采集

VIDIOC_QUERYSTD：检查当前视频设备支持的标准，例如PAL或NTSC。

这些IO调用，有些是必须的，有些是可选择的。

在网上有开源的应用simplewebcam，它已经实现了基本的v4l2视频采集流程。大概看下它是怎么做的

操作流程



3.2 具体代码实现

(1) 打开设备驱动节点

```
int opendevice(int i)
{
    struct stat st;

    sprintf(dev_name,"/dev/video%d",i);

    if (-1 == stat (dev_name, &st)) {
        LOGE("Cannot identify '%s': %d, %s", dev_name, errno, strerror (errno));
        return ERROR_LOCAL;
    }

    if (!S_ISCHR (st.st_mode)) {
        LOGE("%s is no device", dev_name);
        return ERROR_LOCAL;
    }

    fd = open (dev_name, O_RDWR);

    if (-1 == fd) {
        LOGE("Cannot open '%s': %d, %s", dev_name, errno, strerror (errno));
        return ERROR_LOCAL;

    }
    return SUCCESS_LOCAL;
}
```

(2) 查询驱动功能

```
int initdevice(void)
{
    struct v4l2_capability cap;
    struct v4l2_format fmt;
    unsigned int min;

    if (-1 == xioctl (fd, VIDIOC_QUERYCAP, &cap)) {
        if (EINVAL == errno) {
            LOGE("%s is no V4L2 device", dev_name);
            return ERROR_LOCAL;
        } else {
            return errnoexit ("VIDIOC_QUERYCAP");
        }
    }

    if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
        LOGE("%s is no video capture device", dev_name);
        return ERROR_LOCAL;
    }

    if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
        LOGE("%s does not support streaming i/o", dev_name);
        return ERROR_LOCAL;
    }


    ......

}
```

(3) 设置视频格式

```
int initdevice(void)
{
    struct v4l2_capability cap;
    struct v4l2_format fmt;

    ......

    CLEAR (fmt);
    fmt.type                = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.width       = IMG_WIDTH;
    fmt.fmt.pix.height      = IMG_HEIGHT;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;

    if (-1 == xioctl (fd, VIDIOC_S_FMT, &fmt))
        return errnoexit ("VIDIOC_S_FMT");

    ......
}
```

(4) 申请帧缓存并映射到用户空间

```
int initmmap(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) {
            LOGE("%s does not support memory mapping", dev_name);
            return ERROR_LOCAL;
        } else {
            return errnoexit ("VIDIOC_REQBUFS");
        }
    }

    if (req.count < 2) {
        LOGE("Insufficient buffer memory on %s", dev_name);
        return ERROR_LOCAL;
    }

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

    if (!buffers) {
        LOGE("Out of memory");
        return ERROR_LOCAL;
    }

    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))
            return errnoexit ("VIDIOC_QUERYBUF");

        buffers[n_buffers].length = buf.length;
        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)
            return errnoexit ("mmap");
    }

    return SUCCESS_LOCAL;
}
```

(5) 将帧缓存加入缓存队列并启动视频采集

```
int startcapturing(void)
{
    unsigned int i;
    struct v4l2_buffer buf;
    enum v4l2_buf_type type;

    for (i = 0; i < n_buffers; ++i) {
        CLEAR (buf);
        buf.type        = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory      = V4L2_MEMORY_MMAP;
        buf.index       = i;

        if (-1 == xioctl (fd, VIDIOC_QBUF, &buf))
            return errnoexit ("VIDIOC_QBUF");
    }

    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (-1 == xioctl (fd, VIDIOC_STREAMON, &type))
        return errnoexit ("VIDIOC_STREAMON");

    return SUCCESS_LOCAL;
}
```

(6) 从缓存队列中取出一帧

```
int readframeonce(void)
{
    for (;;) {
        fd_set fds;
        struct timeval tv;
        int r;

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

        tv.tv_sec = 2;
        tv.tv_usec = 0;

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

        if (-1 == r) {
            if (EINTR == errno)
                continue;

            return errnoexit ("select");
        }

        if (0 == r) {
            LOGE("select timeout");
            return ERROR_LOCAL;

        }

        if (readframe ()==1)
            break;

    }

    return realImageSize;

}
```

```
int readframe(void)
{

    struct v4l2_buffer buf;
    unsigned int i;

    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:
            default:
                return errnoexit ("VIDIOC_DQBUF");
        }
    }

    assert (buf.index < n_buffers);

    convert2JPEG(buffers[buf.index].start, buf.bytesused);

    if (-1 == xioctl (fd, VIDIOC_QBUF, &buf))
        return errnoexit ("VIDIOC_QBUF");

    return 1;
}
```

4. 解码mjpeg格式

我所使用的usb摄像头是mjpeg格式，而从网上下载的simplewebcam应用只支持yuyv格式，所以需要重写解码模块。

4.1 jni层 - 插入huffman表

安卓自带的libjpeg解码库只能解码jpeg格式。而mjpeg格式需要在v4l2读出的帧中找到SOF0（Start Of Frame 0），插入huffman表后就可以用libjpeg库解码成rgb。

```
static int convert2JPEG(const void *p, int size)
{
    char *mjpgBuf = NULL;

    if (pImageBuf == NULL) {
        return errnoexit("pImageBuf isn't initialized in JNI");
    }

    /* Clear pImageBuf and realImageSize */
    memset(pImageBuf, 0, (IMG_WIDTH*IMG_HEIGHT)*2);
    realImageSize = 0;

    /* insert dht data to p, and then save them to pImageBuf */
    realImageSize = insert_huffman(p, size, pImageBuf);

    return SUCCESS_LOCAL;
}

static int insert_huffman(const void *in_buf, int buf_size, void *out_buf)
{
    int pos = 0;
    int size_start = 0;
    char *pcur = (char *)in_buf;
    char *pdeb = (char *)in_buf;
    char *plimit = (char *)in_buf + buf_size;
    char *jpeg_buf = (char *)out_buf;

    /* find the SOF0(Start Of Frame 0) of JPEG */
    while ( (((pcur[0] << 8) | pcur[1]) != 0xffc0) && (pcur < plimit) ){
        pcur++;
    }

    LOGD("pcur: 0x%x, plimit: 0x%x", pcur, plimit);

    /* SOF0 of JPEG exist */
    if (pcur < plimit){
        if (jpeg_buf != NULL)
        {
            /* insert huffman table after SOF0 */
            size_start = pcur - pdeb;
            memcpy(jpeg_buf, in_buf, size_start);
            pos += size_start;
            memcpy(jpeg_buf + pos, dht_data, sizeof(dht_data));
            pos += sizeof(dht_data);
            memcpy(jpeg_buf + pos, pcur, buf_size - size_start);
            pos += buf_size - size_start;
            return pos;
        }
    } else{
        LOGE("SOF0 does not exist");
    }
    return 0;
}

const static unsigned char dht_data[] = {
    0xff, 0xc4, 0x01, 0xa2, 0x00, 0x00, 0x01, 0x05, 0x01, 0x01, 0x01, 0x01,
    0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x02,
    0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x01, 0x00, 0x03,
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
    0x0a, 0x0b, 0x10, 0x00, 0x02, 0x01, 0x03, 0x03, 0x02, 0x04, 0x03, 0x05,
62      0x05, 0x04, 0x04, 0x00, 0x00, 0x01, 0x7d, 0x01, 0x02, 0x03, 0x00, 0x04,
63      0x11, 0x05, 0x12, 0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07, 0x22,
64      0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08, 0x23, 0x42, 0xb1, 0xc1, 0x15,
65      0x52, 0xd1, 0xf0, 0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16, 0x17,
66      0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2a, 0x34, 0x35, 0x36,
67      0x37, 0x38, 0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a,
68      0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5a, 0x63, 0x64, 0x65, 0x66,
69      0x67, 0x68, 0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7a,
70      0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95,
71      0x96, 0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa8,
72      0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2,
73      0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4, 0xd5,
74      0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
75      0xe8, 0xe9, 0xea, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9,
76      0xfa, 0x11, 0x00, 0x02, 0x01, 0x02, 0x04, 0x04, 0x03, 0x04, 0x07, 0x05,
77      0x04, 0x04, 0x00, 0x01, 0x02, 0x77, 0x00, 0x01, 0x02, 0x03, 0x11, 0x04,
78      0x05, 0x21, 0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71, 0x13, 0x22,
79      0x32, 0x81, 0x08, 0x14, 0x42, 0x91, 0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33,
80      0x52, 0xf0, 0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34, 0xe1, 0x25,
81      0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26, 0x27, 0x28, 0x29, 0x2a, 0x35, 0x36,
82      0x37, 0x38, 0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a,
83      0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5a, 0x63, 0x64, 0x65, 0x66,
84      0x67, 0x68, 0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7a,
85      0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x92, 0x93, 0x94,
86      0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
87      0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba,
88      0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
89      0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
90      0xe8, 0xe9, 0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa
91  };
```

第28-31行，找到SOF0所在的位置，并让pcur指向它

第39-47行，在SOF0所在的位置之后插入huffman表，也就是dht_data数组。可被libjpeg解码的图像最终保存在pImageBuf中

4.2 jave层 - 解码并显示

jni层把图像保存在pImageBuf，这个buffer对应Java层的mImageBuffer。Jave层获取到图像之后调用BitmapFactory.decodeByteArray进行解码，并通过Canvas显示图像

```
@Override
public void run() {
    while (true && cameraExists) {
        ......

        imageSize = processCamera();
        if(imageSize == -1 || imageSize == 0)
            continue;

        bmp = BitmapFactory.decodeByteArray(mImageBuffer.array(), mImageBuffer.array
        if(bmp == null)
            continue;

        Canvas canvas = getHolder().lockCanvas();
        if (canvas != null)
        {
            // draw camera bmp on canvas
            canvas.drawBitmap(bmp,null,rect,null);
            getHolder().unlockCanvasAndPost(canvas);
        }
    }
}
```

5. 总结

底层配置，只需要使能otg功能并把uvc相关的配置宏打开，插入设备后生成了/dev/videoX设备节点则说明usb摄像头枚举并初始化成功了

上层应用，采用网上的开源应用simplewebcam，这个应用只支持yuyv格式，所以需要重写解码模块。需要在数据帧中手动插入huffman表之后，才能用android的libjpeg库来解码mjpeg格式

另外，在调试过程中出现了”uvcvideo: Non-zero status (-71) in video completion handler”这样的log，那是因为mt6735平台的usb host controller对iso端点的支持不太好，经常出现丢包现象，这个问题需要打上mtk提供的patch才能解决问题