Embeded Linux

fglswh.cublog.cn

关于 Linux 的视频编程(v4l2 编程)

一.什么是 video4linux
Video4linux2（简称 V4L2),是 linux 中关于视频设备的内核驱动。在 Linux 中，视频设备是设备文件，可以像访问普通文件一样对其进行读写，摄像头在/dev/video0 下。

二、一般操作流程（视频设备）：
1. 打开设备文件。 int fd=open(”/dev/video0″,O_RDWR);
2. 取得设备的 capability，看看设备具有什么功能，比如是否具有视频输入,或者音频输入输出等。VIDIOC_QUERYCAP,struct v4l2_capability
3. 选择视频输入，一个视频设备可以有多个视频输入。VIDIOC_S_INPUT,struct v4l2_input
4. 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。VIDIOC_S_STD,VIDIOC_S_FMT,struct v4l2_std_id,struct v4l2_format

5. 向驱动申请帧缓冲，一般不超过 5 个。struct v4l2_requestbuffers
6. 将申请到的帧缓冲映射到用户空间，这样就可以直接操作采集到的帧了，而不必去复制。mmap
7. 将申请到的帧缓冲全部入队列，以便存放采集到的数据.VIDIOC_QBUF,struct v4l2_buffer
8. 开始视频的采集。VIDIOC_STREAMON
9. 出队列以取得已采集数据的帧缓冲，取得原始采集数据。VIDIOC_DQBUF
10. 将缓冲重新入队列尾,这样可以循环采集。VIDIOC_QBUF
11. 停止视频的采集。VIDIOC_STREAMOFF
12. 关闭视频设备。close(fd);

三、常用的结构体(参见/usr/include/linux/videodev2.h)：

struct v4l2_requestbuffers reqbufs;//向驱动申请帧缓冲的请求，里面包含申请的个数
struct v4l2_capability cap;//这个设备的功能，比如是否是视频输入设备
struct v4l2_input input; //视频输入
struct v4l2_standard std;//视频的制式，比如 PAL，NTSC
struct v4l2_format fmt;//帧的格式，比如宽度，高度等

struct v4l2_buffer buf;//代表驱动中的一帧
v4l2_std_id stdid;//视频制式，例如：V4L2_STD_PAL_B
struct v4l2_queryctrl query;//查询的控制
struct v4l2_control control;//具体控制的值

下面具体说明开发流程（网上找的啦，也在学习么）

打开视频设备

在 V4L2 中，视频设备被看做一个文件。使用 open 函数打开这个设备：

// 用非阻塞模式打开摄像头设备

int cameraFd;

cameraFd = open(“/dev/video0″, O_RDWR | O_NONBLOCK, 0);

// 如果用阻塞模式打开摄像头设备，上述代码变为：

//cameraFd = open(”/dev/video0″, O_RDWR, 0);

关于阻塞模式和非阻塞模式

应用程序能够使用阻塞模式或非阻塞模式打开视频设备，如果使用非阻塞模式调用视频设备，即使尚未捕获到信息，驱动依旧会把缓存（DQBUFF）里的东西返回给应用程序。

设定属性及采集方式

打开视频设备后，可以设置该视频设备的属性，例如裁剪、缩放等。这一步是可选的。在 Linux 编程中，一般使用 ioctl 函数来对设备的 I/O 通道进行管理：

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

__fd：设备的 ID，例如刚才用 open 函数打开视频通道后返回的 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：

1. VIDIOC_REQBUFS：分配内存
2. VIDIOC_QUERYBUF：把 VIDIOC_REQBUFS 中分配的数据缓存转换成物理地址
3. VIDIOC_QUERYCAP：查询驱动功能
4. VIDIOC_ENUM_FMT：获取当前驱动支持的视频格式
5. VIDIOC_S_FMT：设置当前驱动的频捕获格式
6. VIDIOC_G_FMT：读取当前驱动的频捕获格式
7. VIDIOC_TRY_FMT：验证当前驱动的显示格式

8. VIDIOC_CROPCAP：查询驱动的修剪能力
9. VIDIOC_S_CROP：设置视频信号的边框
10. VIDIOC_G_CROP：读取视频信号的边框
11. VIDIOC_QBUF：把数据从缓存中读取出来
12. VIDIOC_DQBUF：把数据放回缓存队列
13. VIDIOC_STREAMON：开始视频显示函数
14. VIDIOC_STREAMOFF：结束视频显示函数
15. 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。

检查当前视频设备支持的标准

在亚洲，一般使用 PAL（720X576）制式的摄像头，而欧洲一般使用 NTSC（720X480），使用 VIDIOC_QUERYSTD 来检测：

```
v4l2_std_id std;

do {

ret = ioctl(fd, VIDIOC_QUERYSTD, &std);

} while (ret == -1 && errno == EAGAIN);

switch (std) {

case V4L2_STD_NTSC:

//……

case V4L2_STD_PAL:

//……

}
```

设置视频捕获格式

当检测完视频设备支持的标准后，还需要设定视频捕获格式：

```
struct v4l2_format    fmt;

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

fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

fmt.fmt.pix.width = 720;

fmt.fmt.pix.height = 576;

fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;

fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;

if (ioctl(fd, VIDIOC_S_FMT, &fmt) == -1) {

return -1;

}
```

v4l2_format 结构体定义如下：

```
struct v4l2_format
{
enum v4l2_buf_type type;    // 数据流类型，必须永远是//V4L2_BUF_TYPE_VIDEO_CAPTURE
union
{
struct v4l2_pix_format    pix;
struct v4l2_window        win;
struct v4l2_vbi_format    vbi;
__u8    raw_data[200];
} fmt;
};
struct v4l2_pix_format
{
__u32                width;        // 宽，必须是 16 的倍数
__u32                height;       // 高，必须是 16 的倍数
__u32                pixelformat;  // 视频数据存储类型，例如是//YUV4：2：2 还是 RGB
enum v4l2_field        field;
__u32                bytesperline;
__u32                sizeimage;
enum v4l2_colorspace    colorspace;
__u32                priv;
};
```

分配内存

接下来可以为视频捕获分配内存：

```
struct v4l2_requestbuffers  req;
if (ioctl(fd, VIDIOC_REQBUFS, &req) == -1) {
return -1;
}
```

v4l2_requestbuffers 定义如下：

```
struct v4l2_requestbuffers
{
```

```
__u32             count;  // 缓存数量，也就是说在缓存队列里保持多少张照片
enum v4l2_buf_type  type;   // 数据流类型，必须永远是 V4L2_BUF_TYPE_VIDEO_CAPTURE
enum v4l2_memory    memory; // V4L2_MEMORY_MMAP 或 V4L2_MEMORY_USERPTR
__u32             reserved[2];
};
```

获取并记录缓存的物理空间

使用 VIDIOC_REQBUFS，我们获取了 req.count 个缓存，下一步通过调用 VIDIOC_QUERYBUF 命令来获取这些缓存的地址，然后使用 mmap 函数转换成应用程序中的绝对地址，最后把这段缓存放入缓存队列：

```
typedef struct VideoBuffer {
void *start;
size_t  length;
} VideoBuffer;
VideoBuffer*         buffers = calloc( req.count, sizeof(*buffers) );
struct v4l2_buffer    buf;
for (numBufs = 0; numBufs < req.count; numBufs++) {
memset( &buf, 0, sizeof(buf) );
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = numBufs;
// 读取缓存
if (ioctl(fd, VIDIOC_QUERYBUF, &buf) == -1) {
return -1;
}
buffers[numBufs].length = buf.length;
// 转换成相对地址
buffers[numBufs].start = mmap(NULL, buf.length,
PROT_READ | PROT_WRITE,
MAP_SHARED,
fd, buf.m.offset);
if (buffers[numBufs].start == MAP_FAILED) {
return -1;
```

```
}

// 放入缓存队列

if (ioctl(fd, VIDIOC_QBUF, &buf) == -1) {

return -1;

}

}
```

关于视频采集方式

操作系统一般把系统使用的内存划分成用户空间和内核空间，分别由应用程序管理和操作系统管理。应用程序可以直接访问内存的地址，而内核空间存放的是 供内核访问的代码和数据，用户不能直接访问。v4l2 捕获的数据，最初是存放在内核空间的，这意味着用户不能直接访问该段内存，必须通过某些手段来转换地 址。

一共有三种视频采集方式：使用 read、write 方式；内存映射方式和用户指针模式。

read、write 方式:在用户空间和内核空间不断拷贝数据，占用了大量用户内存空间，效率不高。

内存映射方式：把设备里的内存映射到应用程序中的内存控件，直接处理设备内存，这是一种有效的方式。上面的 mmap 函数就是使用这种方式。

用户指针模式：内存片段由应用程序自己分配。这点需要在 v4l2_requestbuffers 里将 memory 字段设置成 V4L2_MEMORY_USERPTR。

处理采集数据

V4L2 有一个数据缓存，存放 req.count 数量的缓存数据。数据缓存采用 FIFO 的方式，当应用程序调用缓存数据时，缓存队列将最先采集到的 视频数据缓存送出，并重新采集一张视频数据。这个过程需要用到两个 ioctl 命令,VIDIOC_DQBUF 和 VIDIOC_QBUF：

```
struct v4l2_buffer buf;

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

buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;

buf.memory=V4L2_MEMORY_MMAP;

buf.index=0;

//读取缓存

if (ioctl(cameraFd, VIDIOC_DQBUF, &buf) == -1)

{

return -1;

}

//.....视频处理算法

//重新放入缓存队列

if (ioctl(cameraFd, VIDIOC_QBUF, &buf) == -1) {

return -1;
```

}

关闭视频设备

使用 close 函数关闭一个视频设备

close(cameraFd)

还需要使用 munmap 方法。

发表于： 2009-09-04 ，修改于： 2009-09-04 10:36，已浏览 890 次，有评论 0 条 推荐 投诉

网友评论

发表评论