虚拟摄像头分析

  前面我们提到，V4L2 的核心是 v4l2-dev.c 它向上提供统一的文件操作接口 v4l2\_fops ，向下提供 video\_device 注册接口 register\_video\_device ，作为一个具体的驱动，需要做的工作就是分配、设置、注册一个 video\_device.框架很简单，复杂的是视频设备相关众多的 ioctl。

一、vivi 框架分析

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** \_\_init vivi\_init(**void**)
2. {
3. ret = vivi\_create\_instance(i);
4. ...
5. **return** ret;
6. }
7. module\_init(vivi\_init);

  vivi 分配了一个 video\_device 指针，没有去设置而是直接让它指向了一个现成的 video\_device 结构 vivi\_template ，那么全部的工作都将围绕 vivi\_template 展开。

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** \_\_init vivi\_create\_instance(**int** inst)
2. {
3. **struct** vivi\_dev \*dev;
4. **struct** video\_device \*vfd;
5. **struct** v4l2\_ctrl\_handler \*hdl;
6. **struct** vb2\_queue \*q;
8. // 分配一个 vivi\_dev 结构体
9. dev = kzalloc(**sizeof**(\*dev), GFP\_KERNEL);
11. // v4l2\_dev 初始化,并没有什么作用
12. ret = v4l2\_device\_register(NULL, &dev->v4l2\_dev);
14. // 设置 dev 的一些参数，比如图像格式、大小
15. dev->fmt = &formats[0];
16. dev->width = 640;
17. dev->height = 480;
18. dev->pixelsize = dev->fmt->depth / 8;
19. ...
21. // vivi\_dev->vb\_vidq(vb2\_queue) 初始化
22. q = &dev->vb\_vidq;
23. memset(q, 0, **sizeof**(dev->vb\_vidq));
24. q->type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;
25. q->io\_modes = VB2\_MMAP | VB2\_USERPTR | VB2\_DMABUF | VB2\_READ;
26. q->drv\_priv = dev;
27. q->buf\_struct\_size = **sizeof**(**struct** vivi\_buffer);
29. // vivi\_dev->vb\_vidq(vb2\_queue)->ops
30. q->ops     = &vivi\_video\_qops;
32. // vivi\_dev->vb\_vidq(vb2\_queue)->mem\_ops
33. q->mem\_ops = &vb2\_vmalloc\_memops;
35. // 初始化一些锁之类的东西
36. vb2\_queue\_init(q);
38. /\* init video dma queues \*/
39. INIT\_LIST\_HEAD(&dev->vidq.active);
40. init\_waitqueue\_head(&dev->vidq.wq);
42. // 分配一个 video\_device ,这才是重点
43. vfd = video\_device\_alloc();
45. \*vfd = vivi\_template;
46. vfd->debug = debug;
47. vfd->v4l2\_dev = &dev->v4l2\_dev;
48. set\_bit(V4L2\_FL\_USE\_FH\_PRIO, &vfd->flags);
50. vfd->lock = &dev->mutex;
52. // 注册 video\_device ！！！
53. ret = video\_register\_device(vfd, VFL\_TYPE\_GRABBER, video\_nr);
54. // 把 vivi\_dev 放入 video\_device->dev->p->driver\_data ,这个后边经常用到
55. video\_set\_drvdata(vfd, dev);
57. /\* Now that everything is fine, let's add it to device list \*/
58. list\_add\_tail(&dev->vivi\_devlist, &vivi\_devlist);
60. **if** (video\_nr != -1)
61. video\_nr++;
62. // vivi\_dev->vfd(video\_device) =  vfd
63. dev->vfd = vfd;
64. v4l2\_info(&dev->v4l2\_dev, "V4L2 device registered as %s\n",
65. video\_device\_node\_name(vfd));
66. **return** 0;
67. }

  用户空间调用的是 v4l2\_fops ，但是最终会调用到 vivi\_fops ，vivi\_fops 中的 ioctl 调用 video\_ioctl2

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **struct** video\_device vivi\_template = {
2. .name       = "vivi",
3. .fops           = &vivi\_fops,
4. .ioctl\_ops  = &vivi\_ioctl\_ops,
5. .minor      = -1,
6. .release    = video\_device\_release,
8. .tvnorms              = V4L2\_STD\_525\_60,
9. .current\_norm         = V4L2\_STD\_NTSC\_M,
10. };

  video\_register\_device 过程就不详细分析了，前面的文章中分析过，大概就是向核心层注册 video\_device 结构体，核心层注册字符设备并提供一个统一的 fops ，当用户空间 read write ioctl 等，最终还是会跳转到 video\_device->fops ,还有一点就是核心层会把我们注册进来的 video\_device 结构放入一个全局的 video\_device数组。

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **const** **struct** v4l2\_file\_operations vivi\_fops = {
2. .owner      = THIS\_MODULE,
3. .open           = v4l2\_fh\_open,
4. .release        = vivi\_close,
5. .read           = vivi\_read,
6. .poll       = vivi\_poll,
7. .unlocked\_ioctl = video\_ioctl2, /\* V4L2 ioctl handler \*/
8. .mmap           = vivi\_mmap,
9. };

  这里，先看一下 v4l2\_fh\_open 函数

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **int** v4l2\_fh\_open(**struct** file \*filp)
2. {
3. // 前面注册时，我们将 video\_device 结构体放入了全局数组 video\_device ，现在通过     video\_devdata 函数取出来，后面经常用到这种做法
4. **struct** video\_device \*vdev = video\_devdata(filp);
5. // 分配一个 v4l2\_fh 结构，放入file->private\_data 中
6. **struct** v4l2\_fh \*fh = kzalloc(**sizeof**(\*fh), GFP\_KERNEL);
7. filp->private\_data = fh;
8. **if** (fh == NULL)
9. **return** -ENOMEM;
10. v4l2\_fh\_init(fh, vdev);
11. v4l2\_fh\_add(fh);
12. **return** 0;
13. }

  1、我们随时可以通过 video\_devdata 取出我们注册的 video\_device 结构进行操作

  2、我们随时可以通过 file->private\_data 取出 v4l2\_fh 结构，虽然现在还不知道它有啥用

  下面来分析 ioctl ...首先来看一下调用过程

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **long** video\_ioctl2(**struct** file \*file,
2. unsigned **int** cmd, unsigned **long** arg)
3. {
4. **return** video\_usercopy(file, cmd, arg, \_\_video\_do\_ioctl);
5. }

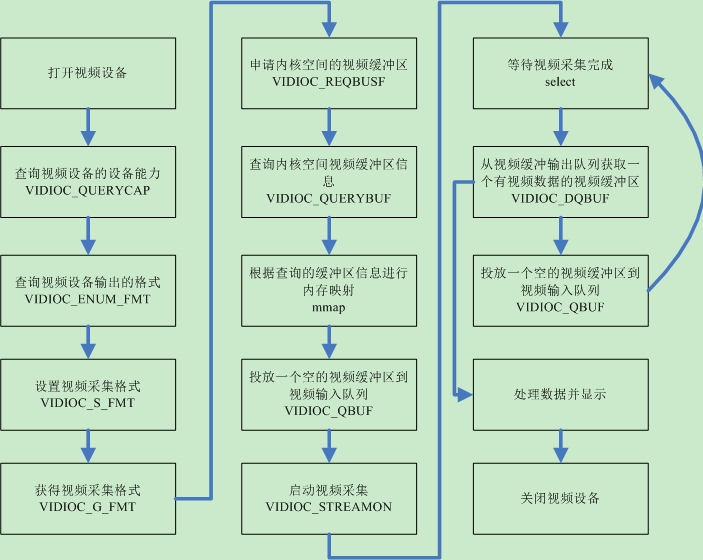
**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **long** \_\_video\_do\_ioctl(**struct** file \*file,
2. unsigned **int** cmd, **void** \*arg)
3. {
4. **struct** video\_device \*vfd = video\_devdata(file);
5. **const** **struct** v4l2\_ioctl\_ops \*ops = vfd->ioctl\_ops;
6. **void** \*fh = file->private\_data;
7. **struct** v4l2\_fh \*vfh = NULL;
8. **int** use\_fh\_prio = 0;
9. **long** ret = -ENOTTY;
11. **if** (ops == NULL) {
12. printk(KERN\_WARNING "videodev: \"%s\" has no ioctl\_ops.\n",
13. vfd->name);
14. **return** ret;
15. }
17. **if** (test\_bit(V4L2\_FL\_USES\_V4L2\_FH, &vfd->flags)) {
18. vfh = file->private\_data;
19. use\_fh\_prio = test\_bit(V4L2\_FL\_USE\_FH\_PRIO, &vfd->flags);
20. }
22. **if** (v4l2\_is\_known\_ioctl(cmd)) {
23. **struct** v4l2\_ioctl\_info \*info = &v4l2\_ioctls[\_IOC\_NR(cmd)];
25. **if** (!test\_bit(\_IOC\_NR(cmd), vfd->valid\_ioctls) &&
26. !((info->flags & INFO\_FL\_CTRL) && vfh && vfh->ctrl\_handler))
27. **return** -ENOTTY;
29. **if** (use\_fh\_prio && (info->flags & INFO\_FL\_PRIO)) {
30. ret = v4l2\_prio\_check(vfd->prio, vfh->prio);
31. **if** (ret)
32. **return** ret;
33. }
34. }
36. **if** ((vfd->debug & V4L2\_DEBUG\_IOCTL) &&
37. !(vfd->debug & V4L2\_DEBUG\_IOCTL\_ARG)) {
38. v4l\_print\_ioctl(vfd->name, cmd);
39. printk(KERN\_CONT "\n");
40. }
42. **switch** (cmd) {
44. /\* --- capabilities ------------------------------------------ \*/
45. **case** VIDIOC\_QUERYCAP:
46. {
47. **struct** v4l2\_capability \*cap = (**struct** v4l2\_capability \*)arg;
49. cap->version = LINUX\_VERSION\_CODE;
50. ret = ops->vidioc\_querycap(file, fh, cap);
51. **if** (!ret)
52. dbgarg(cmd, "driver=%s, card=%s, bus=%s, "
53. "version=0x%08x, "
54. "capabilities=0x%08x, "
55. "device\_caps=0x%08x\n",
56. cap->driver, cap->card, cap->bus\_info,
57. cap->version,
58. cap->capabilities,
59. cap->device\_caps);
60. **break**;
61. }

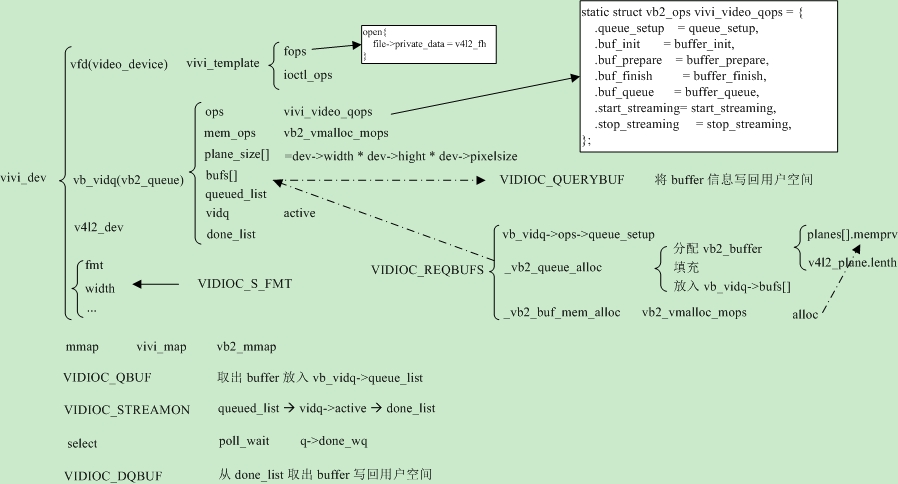
  vivi 驱动就复杂在这些 ioctl 上，下面按照应用层与驱动的交互顺序来具体的分析这些 ioctl 。

二、ioctl 深入分析

  应用空间的一个视频 app 与驱动的交互流程大致如下图所示：



  下面就根据流程，分析每一个 ioctl 在 vivi 中的具体实现。把以上的过程吃透，自己写一个虚拟摄像头程序应该就不成问题了。



 2.1 VIDIOC\_QUERYCAP 查询设备能力

应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **struct** v4l2\_capability {
2. \_\_u8    driver[16]; /\* i.e. "bttv" \*/
3. \_\_u8    card[32];   /\* i.e. "Hauppauge WinTV" \*/
4. \_\_u8    bus\_info[32];   /\* "PCI:" + pci\_name(pci\_dev) \*/
5. \_\_u32   version;        <span style="white-space:pre">    </span>/\* should use KERNEL\_VERSION() \*/
6. \_\_u32   capabilities;   /\* Device capabilities \*/
7. \_\_u32   reserved[4];
8. };
10. **struct** v4l2\_capability cap;
11. ret = ioctl(fd,VIDIOC\_QUERYCAP,&cap);
12. **if** (ret < 0) {
13. LOG("VIDIOC\_QUERYCAP failed (%d)\n", ret);
14. **return** ret;
15. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **void** \*fh = file->private\_data;
2. ops->vidioc\_querycap(file, fh, cap);
3. **static** **int** vidioc\_querycap(**struct** file \*file, **void**  \*priv, **struct** v4l2\_capability \*cap)
4. {
5. **struct** vivi\_fh  \*fh  = priv;
6. **struct** vivi\_dev \*dev = fh->dev;

9. // 这里只是将一些信息写回用户空间而已，非常简单
10. strcpy(cap->driver, "vivi");
11. strcpy(cap->card, "vivi");
12. strlcpy(cap->bus\_info, dev->v4l2\_dev.name, **sizeof**(cap->bus\_info));
13. cap->version =     VIVI\_VERSION; cap->capabilities =V4L2\_CAP\_VIDEO\_CAPTURE |V4L2\_CAP\_STREAMING     | V4L2\_CAP\_READWRITE;**return** 0;}
14. }

  一般我们只关心 capabilities 成员，比如V4L2\_CAP\_VIDEO\_CAPTURE 具有视频捕获能力，其它定义如下：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. /\* Values for 'capabilities' field \*/
2. #define V4L2\_CAP\_VIDEO\_CAPTURE      0x00000001  /\* Is a video capture device \*/
3. #define V4L2\_CAP\_VIDEO\_OUTPUT       0x00000002  /\* Is a video output device \*/
4. #define V4L2\_CAP\_VIDEO\_OVERLAY      0x00000004  /\* Can do video overlay \*/
5. #define V4L2\_CAP\_VBI\_CAPTURE        0x00000010  /\* Is a raw VBI capture device \*/
6. #define V4L2\_CAP\_VBI\_OUTPUT     0x00000020  /\* Is a raw VBI output device \*/
7. #define V4L2\_CAP\_SLICED\_VBI\_CAPTURE 0x00000040  /\* Is a sliced VBI capture device \*/
8. #define V4L2\_CAP\_SLICED\_VBI\_OUTPUT  0x00000080  /\* Is a sliced VBI output device \*/
9. #define V4L2\_CAP\_RDS\_CAPTURE        0x00000100  /\* RDS data capture \*/
10. #define V4L2\_CAP\_VIDEO\_OUTPUT\_OVERLAY   0x00000200  /\* Can do video output overlay \*/
11. #define V4L2\_CAP\_HW\_FREQ\_SEEK       0x00000400  /\* Can do hardware frequency seek  \*/
12. #define V4L2\_CAP\_RDS\_OUTPUT     0x00000800  /\* Is an RDS encoder \*/

 2.2 VIDIOC\_ENUM\_FMT 枚举(查询)设备支持的视频格式

应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **struct** v4l2\_fmtdesc {
2. \_\_u32           index;             /\* Format number      \*/
3. **enum** v4l2\_buf\_type  type;              /\* buffer type        \*/
4. \_\_u32               flags;
5. \_\_u8            description[32];   /\* Description string \*/
6. \_\_u32           pixelformat;       /\* Format fourcc      \*/
7. \_\_u32           reserved[4];
8. };
10. **struct** v4l2\_fmtdesc fmtdesc;
11. fmtdesc.index=0;
12. fmtdesc.type=V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;
13. **while**(ioctl(fd,VIDIOC\_ENUM\_FMT,&fmtdesc)!=-1)
14. {
15. printf("SUPPORT\t%d.%s\n",fmtdesc.index+1,fmtdesc.description);
16. fmtdesc.index++;
17. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **struct** vivi\_fmt formats[] = {
2. {
3. .name     = "4:2:2, packed, YUYV",
4. .fourcc   = V4L2\_PIX\_FMT\_YUYV,
5. .depth    = 16,
6. },
7. ...
8. }
9. **static** **int** vidioc\_enum\_fmt\_vid\_cap(**struct** file \*file, **void**  \*priv,
10. **struct** v4l2\_fmtdesc \*f)
11. {
12. **struct** vivi\_fmt \*fmt;
14. **if** (f->index >= ARRAY\_SIZE(formats))
15. **return** -EINVAL;
17. fmt = &formats[f->index];
19. strlcpy(f->description, fmt->name, **sizeof**(f->description));
20. f->pixelformat = fmt->fourcc;
21. **return** 0;
22. }

  一般一个设备支持多种视频格式，比如 vivi 它所支持的格式存放在 formats 数组中，由于应用层并不知道设备支持多少种格式，也不知道某种格式具体存放在哪个数组项中，因此通过index从0开始尝试，对于驱动层来说就是遍历所有的数组项，返回每一个index对应的视频格式，比如 V4L2\_PIX\_FMT\_YUYV .

 2.3 VIDIOC\_S\_FMT 设置视频格式

应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **struct** v4l2\_format {
2. **enum** v4l2\_buf\_type type;
3. **union** {
4. **struct** v4l2\_pix\_format      pix;     /\* V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE \*/
5. **struct** v4l2\_window      win;     /\* V4L2\_BUF\_TYPE\_VIDEO\_OVERLAY \*/
6. **struct** v4l2\_vbi\_format      vbi;     /\* V4L2\_BUF\_TYPE\_VBI\_CAPTURE \*/
7. **struct** v4l2\_sliced\_vbi\_format   sliced;  /\* V4L2\_BUF\_TYPE\_SLICED\_VBI\_CAPTURE \*/
8. \_\_u8    raw\_data[200];                   /\* user-defined \*/
9. } fmt;
10. };
11. **struct** v4l2\_pix\_format {
12. \_\_u32               width;
13. \_\_u32           height;
14. \_\_u32           pixelformat;
15. **enum** v4l2\_field     field;
16. \_\_u32               bytesperline;   /\* for padding, zero if unused \*/
17. \_\_u32               sizeimage;
18. **enum** v4l2\_colorspace    colorspace;
19. \_\_u32           priv;       /\* private data, depends on pixelformat \*/
20. };
21. **struct** v4l2\_format fmt;
22. memset(&fmt, 0, **sizeof**(fmt));
23. fmt.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;//格式类型
24. fmt.fmt.pix.width //宽度
25. fmt.fmt.pix.height //高度
26. fmt.fmt.pix.pixelformat = VIDEO\_FORMAT;//这一项必须是前面查询出来的某种格式，对应 vivi formats数组
27. fmt.fmt.pix.field       = V4L2\_FIELD\_INTERLACED;//好像是隔行扫描的意思
28. ret = ioctl(fd, VIDIOC\_S\_FMT, &fmt);
29. **if** (ret < 0) {
30. LOG("VIDIOC\_S\_FMT failed (%d)\n", ret);
31. **return** ret;
32. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_s\_fmt\_vid\_cap(**struct** file \*file, **void** \*priv,
2. **struct** v4l2\_format \*f)
3. {
4. **struct** vivi\_dev \*dev = video\_drvdata(file);
5. **struct** vb2\_queue \*q = &dev->vb\_vidq;
7. **int** ret = vidioc\_try\_fmt\_vid\_cap(file, priv, f);
8. //if (fmt->fourcc == f->fmt.pix.pixelformat)返回formats[k]
9. dev->fmt = get\_format(f);
10. dev->pixelsize   = dev->fmt->depth / 8;
11. dev->width       = f->fmt.pix.width;
12. dev->height  = f->fmt.pix.height;
13. dev->field       = f->fmt.pix.field;
15. **return** 0;
16. }
17. **static** **int** vidioc\_try\_fmt\_vid\_cap(**struct** file \*file, **void** \*priv,
18. **struct** v4l2\_format \*f)
19. {
20. **struct** vivi\_dev \*dev = video\_drvdata(file);
21. **struct** vivi\_fmt \*fmt;
22. **enum** v4l2\_field field;
24. fmt = get\_format(f);
26. field = f->fmt.pix.field;
28. **if** (field == V4L2\_FIELD\_ANY) {
29. field = V4L2\_FIELD\_INTERLACED;
30. }
32. f->fmt.pix.field = field;
33. v4l\_bound\_align\_image(&f->fmt.pix.width, 48, MAX\_WIDTH, 2,
34. &f->fmt.pix.height, 32, MAX\_HEIGHT, 0, 0);
35. f->fmt.pix.bytesperline =
36. (f->fmt.pix.width \* fmt->depth) >> 3;
37. f->fmt.pix.sizeimage =
38. f->fmt.pix.height \* f->fmt.pix.bytesperline;
39. **if** (fmt->fourcc == V4L2\_PIX\_FMT\_YUYV ||
40. fmt->fourcc == V4L2\_PIX\_FMT\_UYVY)
41. f->fmt.pix.colorspace = V4L2\_COLORSPACE\_SMPTE170M;
42. **else**
43. f->fmt.pix.colorspace = V4L2\_COLORSPACE\_SRGB;
44. **return** 0;
45. }

  这里将应用层传进来的视频格式简单处理后存放进了一个 vivi\_dev 结构，vivi\_dev 哪里来的呢？，在一开始的时候 vivi\_create\_instance ,我们创建了一个 video\_device 结构代表我们的设备，并设置了一个 vivi\_dev 作为 video\_device->dev->privatedata ，之后 register\_video\_device ，内核会自动将我们的 video\_device 放入全局数组 video\_device[] 中。

 2.4 VIDIOC\_G\_FMT 获得设置好的视频格式

应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. ret = ioctl(fd, VIDIOC\_G\_FMT, &fmt);
2. **if** (ret < 0) {
3. LOG("VIDIOC\_G\_FMT failed (%d)\n", ret);
4. **return** ret;
5. }
6. // Print Stream Format
7. LOG("Stream Format Informations:\n");
8. LOG(" type: %d\n", fmt.type);
9. LOG(" width: %d\n", fmt.fmt.pix.width);
10. LOG(" height: %d\n", fmt.fmt.pix.height);
11. **char** fmtstr[8];
12. memset(fmtstr, 0, 8);
13. memcpy(fmtstr, &fmt.fmt.pix.pixelformat, 4);
14. LOG(" pixelformat: %s\n", fmtstr);
15. LOG(" field: %d\n", fmt.fmt.pix.field);
16. LOG(" bytesperline: %d\n", fmt.fmt.pix.bytesperline);
17. LOG(" sizeimage: %d\n", fmt.fmt.pix.sizeimage);
18. LOG(" colorspace: %d\n", fmt.fmt.pix.colorspace);
19. LOG(" priv: %d\n", fmt.fmt.pix.priv);
20. LOG(" raw\_date: %s\n", fmt.fmt.raw\_data);

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_g\_fmt\_vid\_cap(**struct** file \*file, **void** \*priv,
2. **struct** v4l2\_format \*f)
3. {
4. **struct** vivi\_dev \*dev = video\_drvdata(file);
5. <span style="white-space:pre">    </span>// 把记录在 vivi\_dev 中的参数写回用户空间
6. f->fmt.pix.width        = dev->width;
7. f->fmt.pix.height       = dev->height;
8. f->fmt.pix.field        = dev->field;
9. f->fmt.pix.pixelformat  = dev->fmt->fourcc;
10. f->fmt.pix.bytesperline =
11. (f->fmt.pix.width \* dev->fmt->depth) >> 3;
12. f->fmt.pix.sizeimage =
13. f->fmt.pix.height \* f->fmt.pix.bytesperline;
14. **if** (dev->fmt->fourcc == V4L2\_PIX\_FMT\_YUYV ||
15. dev->fmt->fourcc == V4L2\_PIX\_FMT\_UYVY)
16. f->fmt.pix.colorspace = V4L2\_COLORSPACE\_SMPTE170M;
17. **else**
18. f->fmt.pix.colorspace = V4L2\_COLORSPACE\_SRGB;
19. **return** 0;
20. }

  将我们之前设置的格式返回而已。  
  2.5 VIDIOC\_REQBUFS 请求在内核空间分配视频缓冲区  
    分配的内存位于内核空间,应用程序无法直接访问,需要通过调用mmap内存映射函数,把内核空间的内存映射到用户空间,应用才可以用用户空间地址来访问内核空间。  
应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **struct** v4l2\_requestbuffers {
2. \_\_u32           count;
3. \_\_u32           type;       /\* enum v4l2\_buf\_type \*/
4. \_\_u32           memory;     /\* enum v4l2\_memory \*/
5. \_\_u32           reserved[2];
6. };
7. **struct** v4l2\_requestbuffers reqbuf;
8. reqbuf.type     = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;
9. reqbuf.memory   = V4L2\_MEMORY\_MMAP;
10. reqbuf.count    = BUFFER\_COUNT;
11. ret = ioctl(fd , VIDIOC\_REQBUFS, &reqbuf);
12. **if**(ret < 0) {
13. LOG("VIDIOC\_REQBUFS failed (%d)\n", ret);
14. **return** ret;
15. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_reqbufs(**struct** file \*file, **void** \*priv,
2. **struct** v4l2\_requestbuffers \*p)
3. {
4. **struct** vivi\_dev \*dev = video\_drvdata(file);
5. **return** vb2\_reqbufs(&dev->vb\_vidq, p);    //核心层提供的标准函数
6. }

  vb\_vidq 是 vivi\_dev 的一个成员，前面我们提到它有两个 ops ，一个是 ops 另一个是 mem\_ops

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **struct** vb2\_ops vivi\_video\_qops = {
2. .queue\_setup    = queue\_setup,
3. .buf\_init       = buffer\_init,
4. .buf\_prepare    = buffer\_prepare,
5. .buf\_finish         = buffer\_finish,
6. .buf\_cleanup        = buffer\_cleanup,
7. .buf\_queue      = buffer\_queue,
8. .start\_streaming= start\_streaming,
9. .stop\_streaming     = stop\_streaming,
10. .wait\_prepare       = vivi\_unlock,
11. .wait\_finish        = vivi\_lock,
12. };

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_reqbufs(**struct** file \*file, **void** \*priv,
2. **struct** v4l2\_requestbuffers \*p)
3. {
4. **struct** vivi\_dev \*dev = video\_drvdata(file);
5. **return** vb2\_reqbufs(&dev->vb\_vidq, p);    //核心层提供的标准函数
6. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **int** vb2\_reqbufs(**struct** vb2\_queue \*q, **struct** v4l2\_requestbuffers \*req)
2. {
3. unsigned **int** num\_buffers, allocated\_buffers, num\_planes = 0;
4. **int** ret = 0;
5. // 判断 re->count 是否小于 VIDEO\_MAX\_FRAME
6. num\_buffers = min\_t(unsigned **int**, req->count, VIDEO\_MAX\_FRAME);
7. memset(q->plane\_sizes, 0, **sizeof**(q->plane\_sizes));
8. memset(q->alloc\_ctx, 0, **sizeof**(q->alloc\_ctx));
9. q->memory = req->memory;
11. //(q)->ops->queue\_setup(q,NULL,...)
12. ret = call\_qop(q, queue\_setup, q, NULL, &num\_buffers, &num\_planes,
13. q->plane\_sizes, q->alloc\_ctx);
14. /\* Finally, allocate buffers and video memory \*/
15. ret = \_\_vb2\_queue\_alloc(q, req->memory, num\_buffers, num\_planes);
17. allocated\_buffers = ret;
19. q->num\_buffers = allocated\_buffers;
20. req->count = allocated\_buffers;
21. **return** 0;
22. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** queue\_setup(**struct** vb2\_queue \*vq, **const** **struct** v4l2\_format \*fmt,
2. unsigned **int** \*nbuffers, unsigned **int** \*nplanes,
3. unsigned **int** sizes[], **void** \*alloc\_ctxs[])
4. {
5. **struct** vivi\_dev \*dev = vb2\_get\_drv\_priv(vq);
6. unsigned **long** size;
7. // 每一个buffer 的大小
8. size = dev->width \* dev->height \* dev->pixelsize;
9. **if** (0 == \*nbuffers)
10. \*nbuffers = 32;
11. // 如果申请的buffer过多，导致空间不够减少buffer
12. **while** (size \* \*nbuffers > vid\_limit \* 1024 \* 1024)
13. (\*nbuffers)--;
14. \*nplanes = 1;
15. // 把总大小放入 vivi\_dev->vb\_vidq->plane\_size[0]
16. sizes[0] = size;
17. **return** 0;
18. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** \_\_vb2\_queue\_alloc(**struct** vb2\_queue \*q, **enum** v4l2\_memory memory,
2. unsigned **int** num\_buffers, unsigned **int** num\_planes)
3. {
4. unsigned **int** buffer;
5. **struct** vb2\_buffer \*vb;
6. **int** ret;
7. // 分配多个 vb2\_buffer 填充并放入 vivi\_dev->vb\_vidq->bufs[]
8. **for** (buffer = 0; buffer < num\_buffers; ++buffer) {
9. /\* Allocate videobuf buffer structures \*/
10. vb = kzalloc(q->buf\_struct\_size, GFP\_KERNEL);
12. /\* Length stores number of planes for multiplanar buffers \*/
13. **if** (V4L2\_TYPE\_IS\_MULTIPLANAR(q->type))
14. vb->v4l2\_buf.length = num\_planes;
16. vb->state = VB2\_BUF\_STATE\_DEQUEUED;
17. vb->vb2\_queue = q;
18. vb->num\_planes = num\_planes;
19. vb->v4l2\_buf.index = q->num\_buffers + buffer;
20. vb->v4l2\_buf.type = q->type;
21. vb->v4l2\_buf.memory = memory;
23. /\* Allocate video buffer memory for the MMAP type \*/
24. **if** (memory == V4L2\_MEMORY\_MMAP) {
25. ret = \_\_vb2\_buf\_mem\_alloc(vb);//核心提供的标准函数
26. ret = call\_qop(q, buf\_init, vb);//q->ops->buf\_init
27. }
29. q->bufs[q->num\_buffers + buffer] = vb;
30. }
31. \_\_setup\_offsets(q, buffer);
32. **return** buffer;
33. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** \_\_vb2\_buf\_mem\_alloc(**struct** vb2\_buffer \*vb)
2. {
3. **struct** vb2\_queue \*q = vb->vb2\_queue;
4. **void** \*mem\_priv;
5. **int** plane;
7. /\* num\_planes == 1 \*/
8. **for** (plane = 0; plane < vb->num\_planes; ++plane) {
9. mem\_priv = call\_memop(q, alloc, q->alloc\_ctx[plane],
10. q->plane\_sizes[plane]);
12. /\* Associate allocator private data with this plane \*/
13. vb->planes[plane].mem\_priv = mem\_priv;
14. vb->v4l2\_planes[plane].length = q->[plane];
15. }
17. **return** 0;
18. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **void** \*vb2\_vmalloc\_alloc(**void** \*alloc\_ctx, unsigned **long** size)
2. {
3. **struct** vb2\_vmalloc\_buf \*buf;
5. buf = kzalloc(**sizeof**(\*buf), GFP\_KERNEL);
7. buf->size = size;
8. // 分配空间
9. buf->vaddr = vmalloc\_user(buf->size);
10. buf->handler.refcount = &buf->refcount;
11. buf->handler.put = vb2\_vmalloc\_put;
12. buf->handler.arg = buf;
14. atomic\_inc(&buf->refcount);
15. **return** buf;
16. }

  2.6 VIDIOC\_QUERYBUF 查询分配好的 buffer 信息  
    查询已经分配好的V4L2视频缓冲区的相关信息,包括缓冲区的使用状态、在内核空间的偏移地址、缓冲区长度等,然后应用程序根据这些信息使用mmap把内核空间地址映射到用户空间。  
应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **struct** v4l2\_buffer {
2. \_\_u32                   index;
3. **enum** v4l2\_buf\_type      type;
4. \_\_u32                   bytesused;
5. \_\_u32                   flags;
6. **enum** v4l2\_field         field;
7. **struct** timeval          timest
8. **struct** v4l2\_timecode    timecode;
9. \_\_u32                   sequence;
11. /\* memory location \*/
12. **enum** v4l2\_memory        memory;
13. **union** {
14. \_\_u32               offset;
15. unsigned **long**       userptr;
16. } m;
17. \_\_u32                   length;
18. \_\_u32                   input;
19. \_\_u32                   reserved;
20. };
21. v4l2\_buffer buf;
22. buf.index = i;
23. buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;
24. buf.memory = V4L2\_MEMORY\_MMAP;
25. ret = ioctl(fd , VIDIOC\_QUERYBUF, &buf);
26. **if**(ret < 0) {
27. LOG("VIDIOC\_QUERYBUF (%d) failed (%d)\n", i, ret);
28. **return** ret;
29. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. ops->vidioc\_querybuf(file, fh, p);
2. **static** **int** vidioc\_querybuf(**struct** file \*file, **void** \*priv, **struct** v4l2\_buffer \*p)
3. {
4. **struct** vivi\_dev \*dev = video\_drvdata(file);
5. **return** vb2\_querybuf(&dev->vb\_vidq, p);
6. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **int** vb2\_querybuf(**struct** vb2\_queue \*q, **struct** v4l2\_buffer \*b)
2. {
3. **struct** vb2\_buffer \*vb;
4. // 取出 buf
5. vb = q->bufs[b->index];
6. // 将 buf 信息写回用户空间传递的 b
7. **return** \_\_fill\_v4l2\_buffer(vb, b);
8. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** \_\_fill\_v4l2\_buffer(**struct** vb2\_buffer \*vb, **struct** v4l2\_buffer \*b)
2. {
3. **struct** vb2\_queue \*q = vb->vb2\_queue;
4. **int** ret;
6. /\* Copy back data such as timestamp, flags, input, etc. \*/
7. memcpy(b, &vb->v4l2\_buf, offsetof(**struct** v4l2\_buffer, m));
8. b->input = vb->v4l2\_buf.input;
9. b->reserved = vb->v4l2\_buf.reserved;
11. **if** (V4L2\_TYPE\_IS\_MULTIPLANAR(q->type)) {
12. ret = \_\_verify\_planes\_array(vb, b);
13. **if** (ret)
14. **return** ret;
16. /\*
17. \* Fill in plane-related data if userspace provided an array
18. \* for it. The memory and size is verified above.
19. \*/
20. memcpy(b->m.planes, vb->v4l2\_planes,
21. b->length \* **sizeof**(**struct** v4l2\_plane));
23. **if** (q->memory == V4L2\_MEMORY\_DMABUF) {
24. unsigned **int** plane;
25. **for** (plane = 0; plane < vb->num\_planes; ++plane)
26. b->m.planes[plane].m.fd = 0;
27. }
28. } **else** {
29. /\*
30. \* We use length and offset in v4l2\_planes array even for
31. \* single-planar buffers, but userspace does not.
32. \*/
33. b->length = vb->v4l2\_planes[0].length;
34. b->bytesused = vb->v4l2\_planes[0].bytesused;
35. **if** (q->memory == V4L2\_MEMORY\_MMAP)
36. b->m.offset = vb->v4l2\_planes[0].m.mem\_offset;
37. **else** **if** (q->memory == V4L2\_MEMORY\_USERPTR)
38. b->m.userptr = vb->v4l2\_planes[0].m.userptr;
39. **else** **if** (q->memory == V4L2\_MEMORY\_DMABUF)
40. b->m.fd = 0;
41. }
43. /\*
44. \* Clear any buffer state related flags.
45. \*/
46. b->flags &= ~V4L2\_BUFFER\_STATE\_FLAGS;
48. **switch** (vb->state) {
49. **case** VB2\_BUF\_STATE\_QUEUED:
50. **case** VB2\_BUF\_STATE\_ACTIVE:
51. b->flags |= V4L2\_BUF\_FLAG\_QUEUED;
52. **break**;
53. **case** VB2\_BUF\_STATE\_ERROR:
54. b->flags |= V4L2\_BUF\_FLAG\_ERROR;
55. /\* fall through \*/
56. **case** VB2\_BUF\_STATE\_DONE:
57. b->flags |= V4L2\_BUF\_FLAG\_DONE;
58. **break**;
59. **case** VB2\_BUF\_STATE\_PREPARED:
60. b->flags |= V4L2\_BUF\_FLAG\_PREPARED;
61. **break**;
62. **case** VB2\_BUF\_STATE\_DEQUEUED:
63. /\* nothing \*/
64. **break**;
65. }
67. **if** (\_\_buffer\_in\_use(q, vb))
68. b->flags |= V4L2\_BUF\_FLAG\_MAPPED;
70. **return** 0;
71. }

2.7 mmap  
应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. v4l2\_buffer framebuf[]
2. framebuf[i].length = buf.length;
3. framebuf[i].start = (**char** \*) mmap(
4. NULL,       // 欲指向内存的起始地址，一般为NULL,表示系统自动分配
5. buf.length, //映射长度
6. PROT\_READ|PROT\_WRITE,   //可读可写
7. MAP\_SHARED,     //对映射区的读写会写回内核空间，而且允许其它映射该内核空间地址的进程共享
8. fd,
9. buf.m.offset
10. );
11. **if** (framebuf[i].start == MAP\_FAILED) {
12. LOG("mmap (%d) failed: %s\n", i, strerror(errno));
13. **return** -1;
14. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vivi\_mmap(**struct** file \*file, **struct** vm\_area\_struct \*vma)
2. {
3. **struct** vivi\_dev \*dev = video\_drvdata(file);
4. **int** ret;
5. ret = vb2\_mmap(&dev->vb\_vidq, vma);//核心层提供的函数
6. **return** ret;
7. }

2.8 VIDIOC\_QBUF   
  投放一个空的视频缓冲区到视频缓冲区输入队列，执行成功后，在启动视频设备拍摄图像时，相应的视频数据被保存到视频输入队列相应的视频缓冲区中。  
应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. ret = ioctl(fd , VIDIOC\_QBUF, &buf);
2. **if** (ret < 0) {
3. LOG("VIDIOC\_QBUF (%d) failed (%d)\n", i, ret);
4. **return** -1;

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_qbuf(**struct** file \*file, **void** \*priv, **struct** v4l2\_buffer \*p)
2. {
3. **struct** vivi\_dev \*dev = video\_drvdata(file);
4. **return** vb2\_qbuf(&dev->vb\_vidq, p);
5. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **int** vb2\_qbuf(**struct** vb2\_queue \*q, **struct** v4l2\_buffer \*b)
2. {
3. **struct** rw\_semaphore \*mmap\_sem = NULL;
4. **struct** vb2\_buffer \*vb;
5. **int** ret = 0;
7. vb = q->bufs[b->index];
9. **switch** (vb->state) {
10. **case** VB2\_BUF\_STATE\_DEQUEUED:
11. ret = \_\_buf\_prepare(vb, b);
12. }
13. // 将这个 buffer 挂入 q->queued\_list
14. list\_add\_tail(&vb->queued\_entry, &q->queued\_list);
15. vb->state = VB2\_BUF\_STATE\_QUEUED;
17. **if** (q->streaming)
18. \_\_enqueue\_in\_driver(vb);
20. /\* Fill buffer information for the userspace \*/
21. \_\_fill\_v4l2\_buffer(vb, b);
23. unlock:
24. **if** (mmap\_sem)
25. up\_read(mmap\_sem);
26. **return** ret;
27. }

  实质上就是取出一个 vb2\_buffer 挂入 vivi\_dev->vb\_vidq->queued\_list  
2.9 VIDIOC\_STREAMON  
应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **enum** v4l2\_buf\_type type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;
2. ret = ioctl(fd, VIDIOC\_STREAMON, &type);
3. **if** (ret < 0) {
4. LOG("VIDIOC\_STREAMON failed (%d)\n", ret);
5. **return** ret;
6. }

驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_streamon(**struct** file \*file, **void** \*priv, **enum** v4l2\_buf\_type i)
2. {
3. **struct** vivi\_dev \*dev = video\_drvdata(file);
4. **return** vb2\_streamon(&dev->vb\_vidq, i);
5. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **int** vb2\_streamon(**struct** vb2\_queue \*q, **enum** v4l2\_buf\_type type)
2. {
3. **struct** vb2\_buffer \*vb;
4. **int** ret;
5. vb->state = VB2\_BUF\_STATE\_ACTIVE;
6. // 在 queued\_list 链表中取出 buffer prepare 一下 然后在放入 vidq->active 链表
7. list\_for\_each\_entry(vb, &q->queued\_list, queued\_entry)
8. \_\_enqueue\_in\_driver(vb);
9. //  for (plane = 0; plane < vb->num\_planes; ++plane)
10. //      call\_memop(q, prepare, vb->planes[plane].mem\_priv);
11. //  q->ops->buf\_queue(vb);
12. //q->ops->start\_streaming
13. ret = call\_qop(q, start\_streaming, q, atomic\_read(&q->queued\_count));
15. q->streaming = 1;
16. **return** 0;
17. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **void** buffer\_queue(**struct** vb2\_buffer \*vb)
2. {
3. **struct** vivi\_dev \*dev = vb2\_get\_drv\_priv(vb->vb2\_queue);
4. **struct** vivi\_buffer \*buf = container\_of(vb, **struct** vivi\_buffer, vb);
5. **struct** vivi\_dmaqueue \*vidq = &dev->vidq;
6. unsigned **long** flags = 0;
8. spin\_lock\_irqsave(&dev->slock, flags);
9. // 把 buffer 挂入 vivi\_dev->vidq->active 链表
10. list\_add\_tail(&buf->list, &vidq->active);
11. spin\_unlock\_irqrestore(&dev->slock, flags);
12. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** start\_streaming(**struct** vb2\_queue \*vq, unsigned **int** count)
2. {
3. **struct** vivi\_dev \*dev = vb2\_get\_drv\_priv(vq);
4. dprintk(dev, 1, "%s\n", \_\_func\_\_);
5. **return** vivi\_start\_generating(dev);
6. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vivi\_start\_generating(**struct** vivi\_dev \*dev)
2. {
3. **struct** vivi\_dmaqueue \*dma\_q = &dev->vidq;
5. /\* Resets frame counters \*/
6. dev->ms = 0;
7. dev->mv\_count = 0;
8. dev->jiffies = jiffies;
10. dma\_q->frame = 0;
11. dma\_q->ini\_jiffies = jiffies;
12. // 创建一个内核线程，入口函数 vivi\_thread
13. dma\_q->kthread = kthread\_run(vivi\_thread, dev, dev->v4l2\_dev.name);
15. /\* Wakes thread \*/
16. wake\_up\_interruptible(&dma\_q->wq);
18. **return** 0;
19. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vivi\_thread(**void** \*data)
2. {
3. **struct** vivi\_dev \*dev = data;
5. dprintk(dev, 1, "thread started\n");
7. set\_freezable();
9. **for** (;;) {
10. vivi\_sleep(dev);
12. **if** (kthread\_should\_stop())
13. **break**;
14. }
15. dprintk(dev, 1, "thread: exit\n");
16. **return** 0;
17. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **void** vivi\_sleep(**struct** vivi\_dev \*dev)
2. {
3. **struct** vivi\_dmaqueue \*dma\_q = &dev->vidq;
4. **int** timeout;
5. DECLARE\_WAITQUEUE(wait, current);
7. add\_wait\_queue(&dma\_q->wq, &wait);
8. **if** (kthread\_should\_stop())
9. **goto** stop\_task;
11. /\* Calculate time to wake up \*/
12. timeout = msecs\_to\_jiffies(frames\_to\_ms(1));
14. vivi\_thread\_tick(dev);
16. schedule\_timeout\_interruptible(timeout);
18. stop\_task:
19. remove\_wait\_queue(&dma\_q->wq, &wait);
20. try\_to\_freeze();
21. }

  每次调用 vivi\_sleep 这个线程都被挂入等待队列，调用 vivi\_thread\_tick 填充数据，然后休眠指定的时间自动唤醒，一直循环下去。这样就生成了一帧一帧的视频数据。

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **void** vivi\_thread\_tick(**struct** vivi\_dev \*dev)
2. {
3. **struct** vivi\_dmaqueue \*dma\_q = &dev->vidq;
4. **struct** vivi\_buffer \*buf;
5. unsigned **long** flags = 0;
7. spin\_lock\_irqsave(&dev->slock, flags);
9. buf = list\_entry(dma\_q->active.next, **struct** vivi\_buffer, list);
10. list\_del(&buf->list);
11. spin\_unlock\_irqrestore(&dev->slock, flags);
13. do\_gettimeofday(&buf->vb.v4l2\_buf.timestamp);
15. /\* 填充Buffer \*/
16. vivi\_fillbuff(dev, buf);
18. vb2\_buffer\_done(&buf->vb, VB2\_BUF\_STATE\_DONE);
19. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **void** vb2\_buffer\_done(**struct** vb2\_buffer \*vb, **enum** vb2\_buffer\_state state)
2. {
3. **struct** vb2\_queue \*q = vb->vb2\_queue;
4. unsigned **long** flags;
5. unsigned **int** plane;
7. /\* sync buffers \*/
8. **for** (plane = 0; plane < vb->num\_planes; ++plane)
9. call\_memop(q, finish, vb->planes[plane].mem\_priv);
11. /\* Add the buffer to the done buffers list \*/
12. spin\_lock\_irqsave(&q->done\_lock, flags);
13. vb->state = state;
14. list\_add\_tail(&vb->done\_entry, &q->done\_list);
15. atomic\_dec(&q->queued\_count);
16. #ifdef CONFIG\_SYNC
17. sw\_sync\_timeline\_inc(q->timeline, 1);
18. #endif
19. spin\_unlock\_irqrestore(&q->done\_lock, flags);
21. /\* 应用程序select 时 poll\_wait 里休眠，现在有数据了唤醒 \*/
22. wake\_up(&q->done\_wq);
23. }

  开始的时候我们将以一个 vb\_buffer 挂入 vb\_vidq->queued\_list ，当启动视频传输之后，它被取出挂入 vb\_vidq->vidq->active 队列，然后在内核线程中每一个 tick ，又将它取出填充视频数据之后，再挂入 vb\_vidq->done\_list ，唤醒正在休眠等待视频数据的应用程序。  
2.10 select  
驱动层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. vivi\_poll(**struct** file \*file, **struct** poll\_table\_struct \*wait)
2. {
3. **struct** vivi\_dev \*dev = video\_drvdata(file);
4. **struct** vb2\_queue \*q = &dev->vb\_vidq;
6. **return** vb2\_poll(q, file, wait);
7. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. unsigned **int** vb2\_poll(**struct** vb2\_queue \*q, **struct** file \*file, poll\_table \*wait)
2. {
3. // 挂入休眠队列，是否休眠还要看返回值，大概没有数据就休眠，有数据就不休眠
4. poll\_wait(file, &q->done\_wq, wait);
6. **if** (!list\_empty(&q->done\_list))
7. vb = list\_first\_entry(&q->done\_list, **struct** vb2\_buffer,
8. done\_entry);
9. spin\_unlock\_irqrestore(&q->done\_lock, flags);
11. **if** (vb && (vb->state == VB2\_BUF\_STATE\_DONE
12. || vb->state == VB2\_BUF\_STATE\_ERROR)) {
13. **return** (V4L2\_TYPE\_IS\_OUTPUT(q->type)) ?
14. res | POLLOUT | POLLWRNORM :
15. res | POLLIN | POLLRDNORM;
16. }
17. **return** res;
18. }

  唤醒之后，我们就可以去从视频输出队列中取出buffer，然后根据映射关系，在应用空间取出视频数据了  
2.11 VIDIOC\_DQBUF  
应用层：

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. ret = ioctl(fd, VIDIOC\_DQBUF, &buf);
2. **if** (ret < 0) {
3. LOG("VIDIOC\_DQBUF failed (%d)\n", ret);
4. **return** ret;
5. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** vidioc\_dqbuf(**struct** file \*file, **void** \*priv, **struct** v4l2\_buffer \*p)
2. {
3. **struct** vivi\_dev \*dev = video\_drvdata(file);
4. **return** vb2\_dqbuf(&dev->vb\_vidq, p, file->f\_flags & O\_NONBLOCK);
5. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **int** vb2\_dqbuf(**struct** vb2\_queue \*q, **struct** v4l2\_buffer \*b, **bool** nonblocking)
2. {
3. **struct** vb2\_buffer \*vb = NULL;
4. **int** ret;
5. // 等待在 q->done\_list 取出第一个可用的 buffer
6. ret = \_\_vb2\_get\_done\_vb(q, &vb, nonblocking);
8. ret = call\_qop(q, buf\_finish, vb);
10. /\* 写回buffer的信息到用户空间，应用程序找个这个buffer的mmap之后的地址读数据 \*/
11. \_\_fill\_v4l2\_buffer(vb, b);
12. /\* Remove from videobuf queue \*/
13. list\_del(&vb->queued\_entry);
15. vb->state = VB2\_BUF\_STATE\_DEQUEUED;
16. **return** 0;
17. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** \_\_vb2\_get\_done\_vb(**struct** vb2\_queue \*q, **struct** vb2\_buffer \*\*vb,**int** nonblocking)
2. {
3. unsigned **long** flags;
4. **int** ret;
5. /\*
6. \* Wait for at least one buffer to become available on the done\_list.
7. \*/
8. ret = \_\_vb2\_wait\_for\_done\_vb(q, nonblocking);

11. spin\_lock\_irqsave(&q->done\_lock, flags);
12. \*vb = list\_first\_entry(&q->done\_list, **struct** vb2\_buffer, done\_entry);
13. list\_del(&(\*vb)->done\_entry);
14. spin\_unlock\_irqrestore(&q->done\_lock, flags);
15. **return** 0;
16. }

**[cpp]** [view plain](http://www.itwendao.com/article/detail/63496.html) [copy](http://www.itwendao.com/article/detail/63496.html)

1. **static** **int** buffer\_finish(**struct** vb2\_buffer \*vb)
2. {
3. **struct** vivi\_dev \*dev = vb2\_get\_drv\_priv(vb->vb2\_queue);
4. dprintk(dev, 1, "%s\n", \_\_func\_\_);
5. **return** 0;
6. }