Camera--V4L2驱动学习记录-conceptcon-ChinaUnix博客

Video for Linux Two
V4L2的是V4L的第二个版本。原来的V4L被引入到Linux内核2.1.x的开发周期后期。
Video4Linux2修正了一些设计缺陷，并开始出现在2.5.X内核。
Video4Linux2驱动程序包括Video4Linux1应用的兼容模式，但实际上，支持是不完整的，并建议V4L2的设备使用V4L2的模式。

要想了解 V4l2 有几个重要的文档是必须要读的，
Documentation/video4linux目录下的V4L2-framework.txt和videobuf、V4L2的官方API文档V4L2 API Specification ，
drivers/media/video目录下的vivi.c（虚拟视频驱动程序 -此代码模拟一个真正的视频设备V4L2 API）。

V4l2可以支持多种设备,它可以有以下几种接口:
1. 视频采集接口(video capture interface):这种应用的设备可以是高频头或者摄像头.V4L2的最初设计就是应用于这种功能的.
2. 视频输出接口(video output interface):可以驱动计算机的外围视频图像设备--像可以输出电视信号格式的设备.
3. 直接传输视频接口(video overlay interface):它的主要工作是把从视频采集设备采集过来的信号直接输出到输出设备之上,而不用经过系统的CPU.
4. 视频间隔消隐信号接口(VBI interface):它可以使应用可以访问传输消隐期的视频信号.
5. 收音机接口(radio interface):可用来处理从AM或FM高频头设备接收来的音频流.

V4L2 驱动核心
V4L2 的驱动源码在 drivers/media/video目录下，主要核心代码有：
v4l2-dev.c　　　　　　　　　　　　　　　　　　　　//linux版本2视频捕捉接口,主要结构体 video_device 的注册
v4l2-common.c　　　　　　　　　　　　//在Linux操作系统体系采用低级别的操作一套设备structures/vectors的通用视频设备接口。
　　　　　　　　　　　　　　　　　　　　　　　　　//此文件将替换videodev.c的文件配备常规的内核分配。
v4l2-device.c　　　　　　　　//V4L2的设备支持。注册v4l2_device
v4l22-ioctl.c　　　　　　　　 //处理V4L2的ioctl命令的一个通用的框架。
v4l2-subdev.c　　　　　　　//v4l2子设备
v4l2-mem2mem.c　//内存到内存为Linux和videobuf视频设备的框架。设备的辅助函数，使用其源和目的地videobuf缓冲区。
头文件linux/videodev2.h、media/v4l2-common.h、media/v4l2-device.h、media/v4l2-ioctl.h、media/v4l2-dev.h、media/v4l2-ioctl.h等。

V4l2相关结构体
1.V4l2_device

```
struct v4l2_device {
        /* dev->driver_data points to this struct.
         Note: dev might be NULL if there is no parent device
         as is the case with e.g. ISA devices. */
        struct device *dev;
        /* used to keep track of the registered subdevs */
        struct list_head subdevs;
        /* lock this struct; can be used by the driver as well if this
         struct is embedded into a larger struct. */
        spinlock_t lock;
        /* unique device name, by default the driver name + bus ID */
        char name[V4L2_DEVICE_NAME_SIZE];
        /* notify callback called by some sub-devices. */
        void (*notify)(struct v4l2_subdev *sd,
                        unsigned int notification, void *arg);
};
```

```
/*
  * Newer version of video_device, handled by videodev2.c
  *         This version moves redundant code from video device code to
  *        the common handler
  */
struct video_device
{
        /* device ops */
        const struct v4l2_file_operations *fops;
        /* sysfs */
        struct device dev;              /* v4l device */
        struct cdev *cdev;              /* character device */
        /* Set either parent or v4l2_dev if your driver uses v4l2_device */
        struct device *parent;               /* device parent */
        struct v4l2_device *v4l2_dev;       /* v4l2_device parent */
        /* device info */
        char name[32];
        int vfl_type;
        /* 'minor' is set to -1 if the registration failed */
        int minor;
        u16 num;
        /* use bitops to set/clear/test flags */
        unsigned long flags;
        /* attribute to differentiate multiple indices on one physical device */
        int index;
        int debug;                        /* Activates debug level*/
        /* Video standard vars */
        v4l2_std_id tvnorms;                /* Supported tv norms */
        v4l2_std_id current_norm;       /* Current tvnorm */
        /* callbacks */
        void (*release)(struct video_device *vdev);
        /* ioctl callbacks */
        const struct v4l2_ioctl_ops *ioctl_ops;
};
```

动态分配：

```
struct video_device *vdev = video_device_alloc();
```

结构体配置：

fops：设置这个v4l2_file_operations结构，file_operations的一个子集。v4l2_dev: 设置这个v4l2_device父设备

name:

ioctl_ops：使用v4l2_ioctl_ops简化的IOCTL，然后设置v4l2_ioctl_ops结构。

lock：如果你想要做的全部驱动程序锁定就保留为NULL。否则你给它一个指针指向一个mutex_lock结构体和任何v4l2_file_operations被调用之前核心应该释放释放锁。

parent：一个硬件设备有多个PCI设备，都共享相同v4l2_device核心时，设置注册使用NULL v4l2_device作为父设备结构。

flags：可选的。设置到V4L2_FL_USE_FH_PRIO如你想让框架处理VIDIOC_G/ S_PRIORITY的ioctl。这就需要您使用结构v4l2_fh。这个标志最终会消失，一旦所有的驱动程序使用的核心优先处理。但现在它必须明确设定。

如果使用v4l2_ioctl_ops，那么你应该设置。unlocked_ioctlvideo_ioctl2在v4l2_file_operations结构。

注册/注销 video_device：

```
/**
 *        video_register_device - register video4linux devices
 *        @vdev: video device structure we want to register
 *        @type: type of device to register
 *        @nr: which device node number (0 == /dev/video0, 1 == /dev/video1, ...
 * -1 == first free)
 *        @warn_if_nr_in_use: warn if the desired device node number
 *         was already in use and another number was chosen instead.
 *
 *        The registration code assigns minor numbers and device node numbers
 *        based on the requested type and registers the new device node with
 *        the kernel.
 *        An error is returned if no free minor or device node number could be
 *        found, or if the registration of the device node failed.
 *
 *        Zero is returned on success.
 *
 *        Valid types are
 *
 *        %VFL_TYPE_GRABBER - A frame grabber
 *
 *        %VFL_TYPE_VTX - A teletext device
 *
 *        %VFL_TYPE_VBI - Vertical blank data (undecoded)
 *
 *        %VFL_TYPE_RADIO - A radio card
 */
static int __video_register_device(struct video_device *vdev, int type, int nr,
                int warn_if_nr_in_use)
{
        int i = 0;
        int ret;
        int minor_offset = 0;
        int minor_cnt = VIDEO_NUM_DEVICES;
        const char *name_base;
        void *priv = video_get_drvdata(vdev);
        /* A minor value of -1 marks this video device as never
         having been registered */
        vdev->minor = -1;
        /* the release callback MUST be present */
        WARN_ON(!vdev->release);
        if (!vdev->release)
                return -EINVAL;
        /* Part 1: check device type */
        switch (type) {
        case VFL_TYPE_GRABBER:
                name_base = "video";
                break;
        case VFL_TYPE_VTX:
                name_base = "vtx";
                break;
        case VFL_TYPE_VBI:
                name_base = "vbi";
                break;
        case VFL_TYPE_RADIO:
                name_base = "radio";
                break;
        default:
                printk(KERN_ERR "%s called with unknown type: %d\n",
                 __func__, type);
                return -EINVAL;
        }
        vdev->vfl_type = type;
        vdev->cdev = NULL;
        if (vdev->v4l2_dev && vdev->v4l2_dev->dev)
                vdev->parent = vdev->v4l2_dev->dev;
        /* Part 2: find a free minor, device node number and device index. */
#ifdef CONFIG_VIDEO_FIXED_MINOR_RANGES
        /* Keep the ranges for the first four types for historical
         * reasons.
         * Newer devices (not yet in place) should use the range
         * of 128-191 and just pick the first free minor there
         * (new style). */
        switch (type) {
        case VFL_TYPE_GRABBER:
                minor_offset = 0;
                minor_cnt = 64;
                break;
        case VFL_TYPE_RADIO:
                minor_offset = 64;
                minor_cnt = 64;
                break;
        case VFL_TYPE_VTX:
                minor_offset = 192;
                minor_cnt = 32;
                break;
        case VFL_TYPE_VBI:
                minor_offset = 224;
                minor_cnt = 32;
                break;
        default:
                minor_offset = 128;
                minor_cnt = 64;
                break;
        }
#endif
        /* Pick a device node number */
        mutex_lock(&videodev_lock);
        nr = devnode_find(vdev, nr == -1 ? 0 : nr, minor_cnt);
        if (nr == minor_cnt)
                nr = devnode_find(vdev, 0, minor_cnt);
        if (nr == minor_cnt) {
                printk(KERN_ERR "could not get a free device node number\n");
                mutex_unlock(&videodev_lock);
                return -ENFILE;
        }
#ifdef CONFIG_VIDEO_FIXED_MINOR_RANGES
        /* 1-on-1 mapping of device node number to minor number */
        i = nr;
#else
        /* The device node number and minor numbers are independent, so
         we just find the first free minor number. */
        for (i = 0; i < VIDEO_NUM_DEVICES; i++)
                if (video_device[i] == NULL)
                        break;
        if (i == VIDEO_NUM_DEVICES) {
                mutex_unlock(&videodev_lock);
                printk(KERN_ERR "could not get a free minor\n");
                return -ENFILE;
        }
#endif
        vdev->minor = i + minor_offset;
        vdev->num = nr;
        devnode_set(vdev);
        /* Should not happen since we thought this minor was free */
        WARN_ON(video_device[vdev->minor] != NULL);
        vdev->index = get_index(vdev);
        mutex_unlock(&videodev_lock);
        /* Part 3: Initialize the character device */
        vdev->cdev = cdev_alloc();
        if (vdev->cdev == NULL) {
                ret = -ENOMEM;
                goto cleanup;
        }
        if (vdev->fops->unlocked_ioctl)
                vdev->cdev->ops = &v4l2_unlocked_fops;
        else
                vdev->cdev->ops = &v4l2_fops;
        vdev->cdev->owner = vdev->fops->owner;
        ret = cdev_add(vdev->cdev, MKDEV(VIDEO_MAJOR, vdev->minor), 1);
        if (ret < 0) {
                printk(KERN_ERR "%s: cdev_add failed\n", __func__);
                kfree(vdev->cdev);
```

```c
                vdev->cdev = NULL;
                goto cleanup;
        }
        /* Part 4: register the device with sysfs */
        memset(&vdev->dev, 0, sizeof(vdev->dev));
        /* The memset above cleared the device's drvdata, so
         put back the copy we made earlier. */
        video_set_drvdata(vdev, priv);
        vdev->dev.class = &video_class;
        vdev->dev.devt = MKDEV(VIDEO_MAJOR, vdev->minor);
        if (vdev->parent)
                vdev->dev.parent = vdev->parent;
        dev_set_name(&vdev->dev, "%s%d", name_base, vdev->num);
        ret = device_register(&vdev->dev);
        if (ret < 0) {
                printk(KERN_ERR "%s: device_register failed\n", __func__);
                goto cleanup;
        }
        /* Register the release callback that will be called when the last
         reference to the device goes away. */
        vdev->dev.release = v4l2_device_release;
        if (nr != -1 && nr != vdev->num && warn_if_nr_in_use)
                printk(KERN_WARNING "%s: requested %s%d, got %s%d\n",
                                __func__, name_base, nr, name_base, vdev->num);
        /* Part 5: Activate this minor. The char device can now be used. */
        mutex_lock(&videodev_lock);
        video_device[vdev->minor] = vdev;
        mutex_unlock(&videodev_lock);
        return 0;
cleanup:
        mutex_lock(&videodev_lock);
        if (vdev->cdev)
                cdev_del(vdev->cdev);
        devnode_clear(vdev);
        mutex_unlock(&videodev_lock);
        /* Mark this video device as never having been registered. */
        vdev->minor = -1;
        return ret;
}
int video_register_device(struct video_device *vdev, int type, int nr)
{
        return __video_register_device(vdev, type, nr, 1);
}
EXPORT_SYMBOL(video_register_device);
/**
 *        video_unregister_device - unregister a video4linux device
 *        @vdev: the device to unregister
 *
 *        This unregisters the passed device. Future open calls will
 *        be met with errors.
 */
void video_unregister_device(struct video_device *vdev)
{
        /* Check if vdev was ever registered at all */
        if (!vdev || vdev->minor < 0)
                return;
        mutex_lock(&videodev_lock);
        set_bit(V4L2_FL_UNREGISTERED, &vdev->flags);
        mutex_unlock(&videodev_lock);
        device_unregister(&vdev->dev);
}
```

```
EXPORT_SYMBOL(video_unregister_device);
```

3.v4l2_subdev

每个子设备驱动程序必须有一个v4l2_subdev结构。这个结构可以独立简单的设备或者如果需要存储更多的状态信息它可能被嵌入在一个更大的结构。由于子设备可以做很多不同的东西，你不想结束一个巨大的OPS结构其中只有少数的OPS通常执行，函数指针进行排序按类别，每个类别都有其自己的OPS结构。顶层OPS结构包含的类别OPS结构，这可能是NULL如果在subdev驱动程序不支持任何从该类别指针。

```
/* Each instance of a subdev driver should create this struct, either
      stand-alone or embedded in a larger struct.
  */
struct v4l2_subdev {
        struct list_head list;
        struct module *owner;
        u32 flags;
        struct v4l2_device *v4l2_dev;
        const struct v4l2_subdev_ops *ops;
        /* name must be unique */
        char name[V4L2_SUBDEV_NAME_SIZE];
        /* can be used to group similar subdevs, value is driver-specific */
        u32 grp_id;
        /* pointer to private data */
        void *priv;
};
```

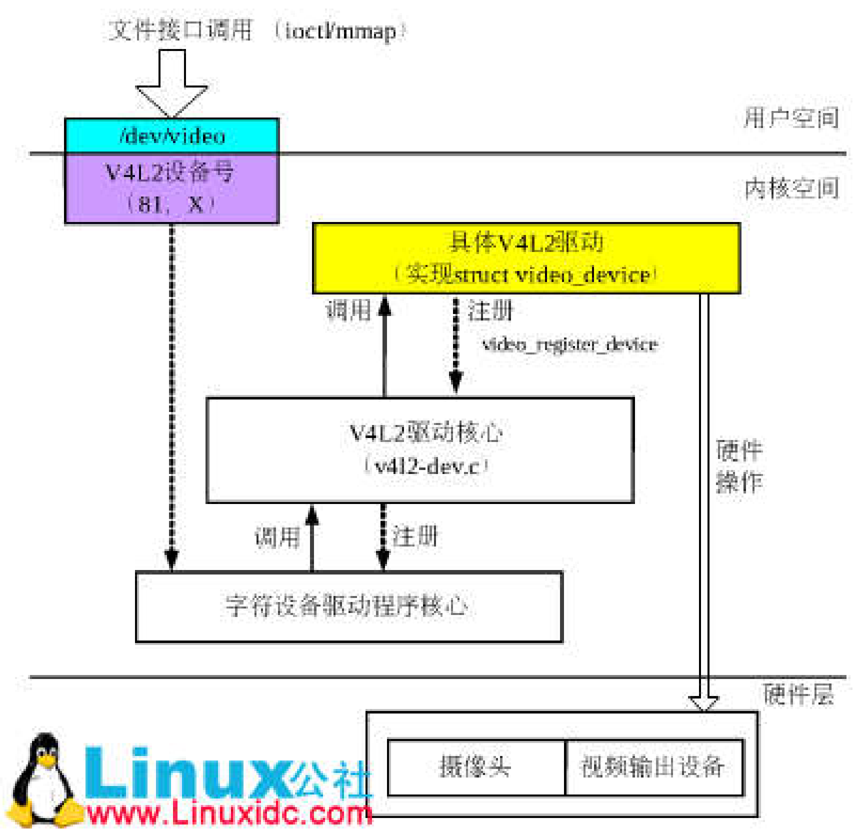
```
struct v4l2_buffer {
        __u32                   index;
        enum v4l2_buf_type type;
        __u32                   bytesused;
        __u32                   flags;
        enum v4l2_field           field;
        struct timeval           timestamp;
        struct v4l2_timecode    timecode;
        __u32                   sequence;
        /* memory location */
        enum v4l2_memory memory;
        union {
                __u32 offset;
                unsigned long userptr;
        } m;
        __u32                   length;
        __u32                   input;
        __u32                   reserved;
};
```

V4L2核心API提供了一套标准方法的用于处理视频缓冲器（称为“videobuf”）。这些方法允许驱动程序以一致的方式来实现read()，mmap()和overlay()。目前使用的设备上的视频缓冲器，支持scatter/gather方法（videobuf-dma-SG），线性存取的DMA的（videobuf-DMA-contig），vmalloc分配的缓冲区，主要用于在USB驱动程序（DMA缓冲区的方法videobuf-vmalloc）。

videobuf层的功能为一种V4L2驱动和用户空间之间的粘合层。它可以处理存储视频帧缓冲区的分配和管理。有一组可用于执行许多标准的POSIX I / O系统调用的功能，包括read（），poll（）的，happily，mmap（）。另一套功能可以用来实现大部分的V4L2的ioctl（）调用相关的流式I/ O的，包括缓冲区分配，排队和dequeueing，流控制。驱动作者使用videobuf规定了一些设计决定，但回收期在驱动器和一个V4L2的用户空间API的贯彻实施在减少代码的形式。

关于videobuf的层的更多信息,请参阅Documentation/video4linux/videobuf

V4l2驱动架构



所有的驱动程序有以下结构：
1) 每个设备包含设备状态的实例结构。
2) 子设备的初始化和命令方式(如果有).
3) 创建V4L2的设备节点 (/dev/videoX, /dev/vbiX and /dev/radioX)和跟踪设备节点的具体数据。
4)文件句柄特定的结构，包含每个文件句柄数据；
5) 视频缓冲处理。

驱动源码分析

vivi.c 虚拟视频驱动程序----- 此代码模拟一个真正的视频设备V4L2 API (位于drivers/media/video目录下)

```
 入口：+int __init vivi_init(void)
                  + vivi_create_instance(i) /*创建设备*//**/。
                        + 分配一个vivi_dev的结构体 /*它嵌套这结构体v4l2_device 和video_device*/
                        + v4l2_device_register(NULL, &dev->v4l2_dev);/*注册vivi_dev中的V4l2_device*/
                        + 初始化视频的DMA队列
                        + 初始化锁
                        + video_device_alloc(); 动态分配video_device结构体
                        + 构建一个video_device结构体 vivi_template 并赋给上面分配的video_device
                              static struct video_device vivi_template = {
                                      .  name            = "vivi",
                                      .fops                = &vivi_fops,
                                      .ioctl_ops        = &vivi_ioctl_ops,
                                      .minor             = -1,
                                      .release       = video_device_release,

                                      .tvnorms                      = V4L2_STD_525_60,
                                      .current_norm              = V4L2_STD_NTSC_M,
                              };
                      + video_set_drvdata(vfd, dev);设置驱动程序专有数据
                      + 所有控件设置为其默认值
                      + list_add_tail(&dev->vivi_devlist, &vivi_devlist);添加到设备列表
            + 构建 v4l2_file_operations 结构体vivi_fops 并实现.open .release .read .poll .mmap函数----- .ioctl 用标准的v4l2控制处理程
序
            + 构建 v4l2_ioctl_ops结构体 vivi_ioctl_ops
                           static const struct v4l2_ioctl_ops vivi_ioctl_ops = {
                                   .vidioc_querycap          = vidioc_querycap,
                                   .vidioc_enum_fmt_vid_cap   = vidioc_enum_fmt_vid_cap,
                                   .vidioc_try_fmt_vid_cap    = vidioc_try_fmt_vid_cap,
                                   .vidioc_s_fmt_vid_cap       = vidioc_s_fmt_vid_cap,
                                   .vidioc_reqbufs           = vidioc_reqbufs,
                                   .vidioc_querybuf          = vidioc_querybuf,
                                   .vidioc_qbuf               = vidioc_qbuf,
                                        .vidioc_dqbuf             = vidioc_dqbuf,
                                        .vidioc_s_std             = vidioc_s_std,
                                        .vidioc_enum_input      = vidioc_enum_input,
                                        .vidioc_g_input          = vidioc_g_input,
                                        .vidioc_s_input          = vidioc_s_input,
                                        .vidioc_queryctrl       = vidioc_queryctrl,
                                        .vidioc_g_ctrl            = vidioc_g_ctrl,
                                        .vidioc_s_ctrl            = vidioc_s_ctrl,
                                        .vidioc_streamon         = vidioc_streamon,
                                        .vidioc_streamoff       = vidioc_streamoff,
                        #ifdef CONFIG_VIDEO_V4L1_COMPAT
                                       .vidiocgmbuf               = vidiocgmbuf,
                      #endif
                  };
+ int vivi_open(struct file *file)
               + vivi_dev *dev = video_drvdata(file);   访问驱动程序专用数据
               + 分配+初始化句柄（vivi_fh）数据
               + 重置帧计数器
               + videobuf_queue_vmalloc_init(); 初始化视频缓冲队列
               + 开启一个新线程用于开始和暂停
+ 实现自定义的v4l2_ioctl_ops 函数
```

这里只是对V4L2的最基本知识

阅读(10259) | 评论(0) | 转发(0) |

0