Chinaunix首页 | 论坛 | 博客 登录 | 注册

博文

【公布获奖】2-3月原创博文评选

cxw3506的ChinaUnix博客

是该记录点东西了！

首页 | 博文目录 | 关于我

cxw3506

博客访问： 11460

博文数量： 22

博客积分： 359

博客等级： 一等列兵

技术积分： 200

用 户 组： 普通用户

注册时间： 2012-04-08 19:51

加关注 短消息

V4L2驱动程序架构

2012-08-02 10:19:44

分类： 嵌入式

1 V4L2简介

video4linux2(V4L2)是Linux内核中关于视频设备的内核驱动，它为Linux中视频设备访问提供了通用接口，在Linux系统中，V4L2驱动的Video设备节点路径通常/dev/video/中的videoX

V4L2驱动对用户空间提供字符设备，主设备号为81，对于视频设备，其次设备号为0-63。除此之外，次设备号为64-127的Radio设备，次设备号为192-223的是Teletext设备,次设备号为224-255的是VBI设备

V4L2驱动的Video设备在用户空间通过各种ioctl调用进行控制，并且可以使用mmap进行内存映射

1.1 V4L2驱动主要使用的ioctl

命令值如下所示：

点击(此处)折叠或打开

```
#define VIDIOC_QUERYCAP _IOR('V', 0, struct v4l2_capability) /*查询能力*/
#define VIDIO_G_FMT _IOWR('V', 4, struct v4l2_format) /*获得格式*/
#define VIDIOC_S_FMT _IOWR('V', 5, struct v4l2_format) /*设置格式*/
#define VIDIOC_REQBUFS _IOWR('V', 8, strut v4l2_requestbuffers) /*申请内存*/
```

论坛 加好友

文章分类

全部博文（22）

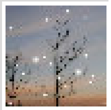
linux（13）

未分配的博文（9）

文章存档

2012年（22）

我的朋友

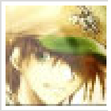


81907343

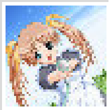
最近访客

幸福软件

Ubuntu_o



rxlinux



bluefish

xkcp0324

complica

```
#define VIDIOC_G_FBUF _IOW('V', 10, struct v4l2_framebuffer) /*获得Framebuffer*/
#define VIDIOC_S_BUF _IOW('V', 11, struct v4l2_framebuffer) /*设置Framebuffer*/
#define VIDIOC_OVERLAY _IOW('V', 14, int) /*设置Overlay*/
#define VIDIOC_QBUF _IOWR('V', 15, struct v4l2_buffer) /*将内存加入队列*/
#define VIDIOC_DQBUF _IOWR('V', 17, strut v4l2_buffer) /*从队列取出内存*/
#define VIDIOC_STREAMON _IOW('V', 18, int) /*开始流*/
#define VIDIOC_STREAMOFF _IOW('V', 19, int) /*停止流*/
#define VIDIOC_G_CTRL _IOWR('V', 27, struct v4l2_control) /*得到控制*/
#define VIDIOC_S_CTRL _IOWR('V', 28, struct v4l2_control) /*设置控制*/
```

1.2 重要结构

头文件 include/linux/videodev2.h

include/media/v4l2-dev.h

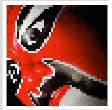
V4L2驱动核心实现文件：driver/media/video/v4l2-dev.c

v4l2-dev.h中定义的video_device是V4L2驱动程序的核心数据结构

```
struct video_device
{
    const struct v4l2_file_operations *fops;
    struct cdev *cdev;//字符设备
    struct device *parent;//父设备
    struct v4l2_device *v4l2_dev;//父v4l2_device
    char name[32];//名称
    int vfl_type;//类型
    int minor;//次设备号
    /*释放回调*/
    void (*release)(struct video_device *vdev);
    /*ioctl回调*/
    const struct v4l2_ioctl_ops *ioctl_ops;
```

huhuwang

rwlcc


xinyuwux

订阅

推荐博文

- 数据库安全--案例篇
- 一次配置DATAGUARD BROKER 的...
- 简单的命令行词典
- Python 数据结构
- Python 调试

热词专题

- rj45 驱动电路
- android 堆栈信息
- SIGSEGVSegmentation fault
- 哈夫曼编码
- xp and dsdm scrum

友情链接

```
}
常用的结构
参见/include/linux/videodev2.h
1)设备能力结构
struct v4l2_capability
{
    __u8 driver[16];//驱动名
    __u8 card[32];//例如Hauppauge winTV
    __u8 bus_info[32];//PCI总线信息
    __u32 version;//内核版本
    __u32 capabilities;//设备能力
    __u32 reserved[4];
};
2)数据格式结构
struct v4l2_format
{
    enum v4l2_buf_type type;//本结构的数据类型
};
3)像素格式结构
struct v4l2_pix_format
{
    __u32   width;//宽度
    __u32   height;//高度
}
4)请求缓冲
struct v4l2_requestbuffers
```

```
{
    __u32   count;//缓存数量
    enum v4l2_buf_type type;//数据流类型
}
5)数据流类型包括V4L2_MEMORY_MMAP和V4L2_MEMORY_USERPTR
enum v4l2_memory{

};
```

点击(此处)折叠或打开

5)数据流类型包括V4L2_MEMORY_MMAP和V4L2_MEMORY_USERPTR enum v4l2_memory{ };

2 V4L2驱动注册 2.1 video_register_device

video4linux2驱动程序的注册drivers/media/video

video_register_device函数用来注册一个v4l驱动程序

```
int video_register_device(struct video_device *vdev, int type, int nr)
{
    return __video_register_device(vdev, type, nr, 1);
}
其中参数type支持的类型如下
#define VFL_TYPE_GRABBER 0//视频
#define VFL_TYPE_VBI    1//从视频消隐的时间取得信息的设备
#define VFL_TYPE_RADIO  2 //广播
#define VFL_TYPE_VTX    3//视传设备
```

```
#define VFL_TYPE_MAX    4//最大值
---------------->返回调用 __video_register_device()
__video_register_device 函数先检查设备类型，接下来
寻找一个可用的子设备号，最后注册相应的字符设备
static int __video_register_device(struct video_device *vdev, int type, int nr, int warn_if_nr_in_use)
{

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;
        nr = devnode_find(vdev, nr == -1 ? 0 : nr, minor_cnt);
    }
    nr = devnode_find(vdev, nr == -1 ? 0 : nr, minor_cnt);
    vdev->cdev->ops = &v4l2_fops;
//注册字符设备
ret = cdev_add(vdev->cdev, MKDEV(VIDEO_MAJOR, vdev->minor), 1);
    ret = device_register(&vdev->dev);
//注册完毕设备信息存储在video_device数组中
    mutex_lock(&videodev_lock);
    video_device[vdev->minor] = vdev;
    mutex_unlock(&videodev_lock);
}
```

```
int video_register_device(struct video_device *vdev, int type, int nr)
{
return __video_register_device(vdev, type, nr, 1);
}
```

其中参数type支持的类型如下

```
#define VFL_TYPE_GRABBER 0//视频
#define VFL_TYPE_VBI 1//从视频消隐的时间取得信息的设备
#define VFL_TYPE_RADIO 2 //广播
#define VFL_TYPE_VTX 3//视传设备
#define VFL_TYPE_MAX 4//最大值
```

----------------->返回调用 __video_register_device() __video_register_device 函数先检查设备类型，接下来寻找一个可用的子设备号，

最后注册相应的字符设备

点击(此处)折叠或打开

2.2 v4l2_fops接口

v4l2_fops为video4linux2设备提供了统一的应用层接口，v4l2_fops定义如下

```
static const struct file_operations v4l2_fops = {
    .owner = THIS_MODULE,
        .read = v4l2_read,
        .write = v4l2_write,
        .open = v4l2_open,
        .get_unmapped_area = v4l2_get_unmapped_area,
        .mmap = v4l2_mmap,
        .unlocked_ioctl = v4l2_ioctl,
        .release = v4l2_release,
        .poll = v4l2_poll,
        .llseek = no_llseek,

};
v4l2_fops中的成员函数最终要调用struct video_device->fops中相应的成员
struct video_device->fops是具体video4linux2摄像头驱动程序必须实现的接口
static ssize_t v4l2_read(struct file *filp, char __user *buf, size_t sz, loff_t *off)
{
    return vdev->fops->read(filp, buf, sz, off);
}
```

2.3 /drivers/media/video/samsung/fimc/s3c_fimc_core.c

驱动探测函数s3c_fimc_probe定义

```
static int s3c_fimc_probe(struct platform_device *dev)
```

```
{
    ctrl = s3c_fimc_register_controller(pdev);

    clk_enable(ctrl->clock);//使能时钟
    //注册V4L2驱动
    ret = video_register_device(ctrl->vd, VFL_TYPE_GRABBER, ctrl->id);
}
s3c_fimc_register_contoller函数主要用来分配资源与申请中断
static struct s3c_fimc_control *s3c_fimc_register_controller(struct platform_device *pdev)
{
    ctrl->vd = &s3c_fimc_video_device[id];
    //申请中断
    ctrl->irq = platform_get_irq(pdev, 0);
    if(request_irq(ctrl->irq, s3c_fimc_irq, IRQF_DISABLED, ctrl->name, ctrl))
};
struct video_device s3c_fimc_video_device[S3C_FIMC_MAX_CTRLS] = {
    [0] = {
        .vfl_type = VID_TYPE_OVERLAY | VID_TYPE_CAPTURE | VID_TYPE_CLIPPING | VID_TYPE_SCALES,
        .fops = &s3c_fimc_fops,
        .ioctl_ops = &s3c_fimc_v4l2_ops,
        .release  = s3c_fimc_vdev_release,

        .name = "sc3_video0",
    },
}
```

s3c_fimc_v4l2_ops,是在drivers/media/video/samsung/fimc中实现的v4l2_ioctl_ops，在用户空间进行ioctl等调用时，要调用到具体实

现的各个函数指针

3 V4L2 操作

3.1 s3c_fimc_open

```
static int s3c_fimc_open(struct file *filp)
{
    struct s3c_fimc_control *ctrl;
    int id, ret;

    id =0;
    ctrl = &s3c_fimc.ctrl[id];
    mutex_lock(&ctrl->lock);
    if (atomic_read(&ctrl->in_use)) {
        ret = -EBUSY;
        goto resource_busy;
    } else {
        atomic_inc(&ctrl->in_use);
        s3c_fimc_reset(ctrl);
        filp->private_data = ctrl;
    }
    mutex_unlock(&ctrl->lock);
    return 0;
resource_busy:
    mutex_unlock(&ctrl->lock);
    return ret;
}
用户空间
打开设备文件
fd = open(dev_name, O_RDWR | O_NONBLOCK, 0);
```

用户空间打开设备文件 fd = open(dev_name, O_RDWR | O_NONBLOCK, 0);

3.2 获取设备的capability,查看设备有什么功能

1）结构体

```
struct v4l2_capability cap;
ret = ioctl(fd, VIDIOC_QUERYCAP, &cap);
/include/linux/videodev2.h
struct v4l2_capability {
    __u8    driver[16]; /* i.e. "bttv" */
    __u8    card[32];   /* i.e. "Hauppauge WinTV" */
    __u8    bus_info[32];   /* "PCI:" + pci_name(pci_dev) */
    __u32   version;        /* should use KERNEL_VERSION() */
    __u32   capabilities;   /* Device capabilities */
    __u32   reserved[4];
};
驱动实现

static int s3c_fimc_v4l2_querycap(struct file *filp, void *fh,
                    struct v4l2_capability *cap)
{
    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
    strcpy(cap->driver, "Samsung FIMC Driver");
    strlcpy(cap->card, ctrl->vd->name, sizeof(cap->card));
    sprintf(cap->bus_info, "FIMC AHB-bus");
    cap->version = 0;
    cap->capabilities = (V4L2_CAP_VIDEO_OVERLAY | \
                V4L2_CAP_VIDEO_CAPTURE | V4L2_CAP_STREAMING);
    return 0;
}
应用层调用
static int video_capability(int fd)
{
    int ret = 0;
    /***********get the device capability********/
    struct v4l2_capability cap;
    ret = ioctl(fd, VIDIOC_QUERYCAP, &cap);
    if (ret < 0) {
        perror("VIDIOC_QUERYCAP failed ");
        return ret;
    }

    printf("\n****Capability informations****\n");
    printf("driver:   %s\n", cap.driver);

    if (cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)
        printf("Capture capability is supported\n");

    if (cap.capabilities & V4L2_CAP_STREAMING)
```

```
        printf("Streaming capability is supported\n");

    if (cap.capabilities & V4L2_CAP_VIDEO_OVERLAY)
        printf("Overlay capability is supported\n");

    return 0;
}
```

3.3 选择视频输入，一个视频设备可以有多个视频输入

```
结构体
struct v4l2_input input;
int index;
得到INPUT
ret = ioctl(fd, VIDIOC_G_INPUT, &index);
input.index = index;
列举INPUT
ret = ioctl(fd, VIDIOC_ENUMINPUT, &input);
设置INPUT
ret = ioctl(fd, VIDIOC_S_INPUT, &index);

struct v4l2_input {
    __u32       index;     /* Which input */
    __u8        name[32];      /* Label */
    __u32       type;      /* Type of input */
    __u32       audioset;      /* Associated audios (bitfield) */
    __u32       tuner;             /* Associated tuner */
    v4l2_std_id  std;
    __u32        status;
    __u32        capabilities;
    __u32        reserved[3];
};

Ioctl: VIDIOC_S_INPUT This IOCTL takes pointer to integer containing index of the input which has to be set. Application will provide the index number as an argument.
    0 - Composite input,
    1 - S-Video input.
驱动
static int s3c_fimc_v4l2_s_input(struct file *filp, void *fh,
                    unsigned int i)
{
    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;

    if (i >= S3C_FIMC_MAX_INPUT_TYPES)
        return -EINVAL;

    ctrl->v4l2.input = &s3c_fimc_input_types[i];

    if (s3c_fimc_input_types[i].type == V4L2_INPUT_TYPE_CAMERA)
        ctrl->in_type = PATH_IN_ITU_CAMERA;
    else
        ctrl->in_type = PATH_IN_DMA;

    return 0;
}
static struct v4l2_input s3c_fimc_input_types[] = {
    {
        .index      = 0,
        .name       = "External Camera Input",
        .type       = V4L2_INPUT_TYPE_CAMERA,
        .audioset   = 1,
        .tuner      = 0,
        .std        = V4L2_STD_PAL_BG | V4L2_STD_NTSC_M,
        .status     = 0,
    },
    {
        .index      = 1,
        .name       = "Memory Input",
        .type       = V4L2_INPUT_TYPE_MEMORY,
        .audioset   = 2,
        .tuner      = 0,
        .std        = V4L2_STD_PAL_BG | V4L2_STD_NTSC_M,
        .status     = 0,
    }
};
static int s3c_fimc_v4l2_enum_input(struct file *filp, void *fh,
                    struct v4l2_input *i)
{
    if (i->index >= S3C_FIMC_MAX_INPUT_TYPES)
```

```
        return -EINVAL;

    memcpy(i, &s3c_fimc_input_types[i->index], sizeof(struct v4l2_input));

    return 0;
}
应用
static int video_input(int fd)
{
    /***********get and set the VIDIO INPUT********/
    int ret = 0;
    struct v4l2_input input;//视频输入信息，对应命令VIDIOC_ENUMINPUT
    int index;
    index = 0;    //0 - Composite input, 1 - S-Video input.

    ret = ioctl (fd, VIDIOC_S_INPUT, &index);
    if (ret < 0) {
        perror ("VIDIOC_S_INPUT");
        return ret;
    }

    input.index = index;
    ret = ioctl (fd, VIDIOC_ENUMINPUT, &input);
    if (ret < 0){
        perror ("VIDIOC_ENUMINPUT");
        return ret;
    }
    printf("\n****input informations****\n");
    printf("name of the input = %s\n", input.name);
    return 0;
}
```

3.4 遍历所有视频格式,查询驱动所支持的格式

```
结构
struct v4l2_fmtdes fmtdes;
ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmtdes);
struct v4l2_fmtdesc {
    __u32           index;             /* Format number      */
    enum v4l2_buf_type  type;              /* buffer type        */
    __u32               flags;
    __u8            description[32];   /* Description string */
    __u32           pixelformat;       /* Format fourcc      */
    __u32           reserved[4];
};
驱动
static int s3c_fimc_v4l2_enum_fmt_vid_cap(struct file *filp, void *fh,
                    struct v4l2_fmtdesc *f)
{
    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
    int index = f->index;

    if (index >= S3C_FIMC_MAX_CAPTURE_FORMATS)
        return -EINVAL;

    memset(f, 0, sizeof(*f));
    memcpy(f, ctrl->v4l2.fmtdesc + index, sizeof(*f));

    return 0;
}
#define S3C_FIMC_MAX_CAPTURE_FORMATS    ARRAY_SIZE(s3c_fimc_capture_formats)
const static struct v4l2_fmtdesc s3c_fimc_capture_formats[] = {
    {
        .index      = 0,
        .type       = V4L2_BUF_TYPE_VIDEO_CAPTURE,
        .flags      = FORMAT_FLAGS_PLANAR,
        .description    = "4:2:0, planar, Y-Cb-Cr",
        .pixelformat    = V4L2_PIX_FMT_YUV420,
    },
    {
        .index      = 1,
        .type       = V4L2_BUF_TYPE_VIDEO_CAPTURE,
        .flags      = FORMAT_FLAGS_PLANAR,
        .description    = "4:2:2, planar, Y-Cb-Cr",
        .pixelformat    = V4L2_PIX_FMT_YUV422P,

    },
    {
        .index      = 2,
        .type       = V4L2_BUF_TYPE_VIDEO_CAPTURE,
        .flags      = FORMAT_FLAGS_PACKED,
        .description    = "4:2:2, packed, YCBYCR",
        .pixelformat    = V4L2_PIX_FMT_YUYV,
    },
    {
        .index      = 3,
        .type       = V4L2_BUF_TYPE_VIDEO_CAPTURE,
        .flags      = FORMAT_FLAGS_PACKED,
        .description    = "4:2:2, packed, CBYCRY",
        .pixelformat    = V4L2_PIX_FMT_UYVY,
    }
};
const static struct v4l2_fmtdesc s3c_fimc_overlay_formats[] = {
    {
        .index      = 0,
        .type       = V4L2_BUF_TYPE_VIDEO_OVERLAY,
        .flags      = FORMAT_FLAGS_PACKED,
        .description    = "16 bpp RGB, le",
        .pixelformat    = V4L2_PIX_FMT_RGB565,
    },
    {
        .index      = 1,
        .type       = V4L2_BUF_TYPE_VIDEO_OVERLAY,
        .flags      = FORMAT_FLAGS_PACKED,
        .description    = "24 bpp RGB, le",
        .pixelformat    = V4L2_PIX_FMT_RGB24,
    },
};
应用层
static int video_fmtdesc(int fd)
{
    /***********Format Enumeration************/
    int ret = 0;
    struct v4l2_fmtdesc fmtdes;
    CLEAR(fmtdes);
    fmtdes.index = 0;
    fmtdes.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    printf("\n**********vidioc enumeration stream format informations:****\n");
    while (1) {

        ret = ioctl(fd, VIDIOC_ENUM_FMT, &fmtdes);
        if (ret < 0)
            break;

         printf("{ pixelformat = %c%c%c%c, description = %s }\n",
                (fmtdes.pixelformat & 0xFF),
                (fmtdes.pixelformat >> 8) & 0xFF,
                (fmtdes.pixelformat >> 16) & 0xFF,
                (fmtdes.pixelformat >> 24) & 0xFF,
                fmtdes.description);

        if (fmtdes.type == V4L2_BUF_TYPE_VIDEO_CAPTURE)
```

```
            printf("video capture type:\n");
        if (fmtdes.pixelformat == V4L2_PIX_FMT_YUYV)
            printf("V4L2_PIX_FMT_YUYV\n");
        fmtdes.index++;
    }
    return 0;
}
```

3.5 设置视频捕获格式（重要）

```
结构体
帧格式包括宽度和高度
struct v4l2_format fmt;
ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
struct v4l2_format {
    enum v4l2_buf_type type;//数据流类型，必须是V4L2_BUF_TYPE_VIDEO_CAPTURE
    union {
        struct v4l2_pix_format      pix;     /* V4L2_BUF_TYPE_VIDEO_CAPTURE */
        struct v4l2_window      win;     /* V4L2_BUF_TYPE_VIDEO_OVERLAY */
        struct v4l2_vbi_format      vbi;     /* V4L2_BUF_TYPE_VBI_CAPTURE */
        struct v4l2_sliced_vbi_format   sliced;  /* V4L2_BUF_TYPE_SLICED_VBI_CAPTURE */
        __u8    raw_data[200];                   /* user-defined */
    } fmt;
};
struct v4l2_pix_format  {
    __u32 pixelformat;//视频数据存储类型，例如是YUV4:2:2还是RGB
}
驱动
```

```
static int s3c_fimc_v4l2_s_fmt_vid_cap(struct file *filp, void *fh,
                  struct v4l2_format *f)
{

    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
    ctrl->v4l2.frmbuf.fmt = f->fmt.pix;

    if (f->fmt.pix.priv == V4L2_FMT_IN)
        s3c_fimc_set_input_frame(ctrl, &f->fmt.pix);
    else
        s3c_fimc_set_output_frame(ctrl, &f->fmt.pix);

    return 0;
}
int s3c_fimc_set_input_frame(struct s3c_fimc_control *ctrl,
              struct v4l2_pix_format *fmt)
{
    s3c_fimc_set_input_format(ctrl, fmt);

    return 0;
}

static void s3c_fimc_set_input_format(struct s3c_fimc_control *ctrl,
                  struct v4l2_pix_format *fmt)
{
    struct s3c_fimc_in_frame *frame = &ctrl->in_frame;

    frame->width = fmt->width;
    frame->height = fmt->height;

    switch (fmt->pixelformat) {
    case V4L2_PIX_FMT_RGB565:
        frame->format = FORMAT_RGB565;
        frame->planes = 1;
        break;

    case V4L2_PIX_FMT_RGB24:
        frame->format = FORMAT_RGB888;
        frame->planes = 1;
        break;

    case V4L2_PIX_FMT_NV12:
        frame->format = FORMAT_YCBCR420;
        frame->planes = 2;
        frame->order_2p = LSB_CBCR;
        break;

    case V4L2_PIX_FMT_NV21:
        frame->format = FORMAT_YCBCR420;
        frame->planes = 2;
        frame->order_2p = LSB_CRCB;
        break;

    case V4L2_PIX_FMT_NV12X:
        frame->format = FORMAT_YCBCR420;
        frame->planes = 2;
        frame->order_2p = MSB_CBCR;
        break;

    case V4L2_PIX_FMT_NV21X:
        frame->format = FORMAT_YCBCR420;
        frame->planes = 2;
        frame->order_2p = MSB_CRCB;
        break;

    case V4L2_PIX_FMT_YUV420:
        frame->format = FORMAT_YCBCR420;
        frame->planes = 3;
        break;

    case V4L2_PIX_FMT_YUYV:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 1;
        frame->order_1p = IN_ORDER422_YCBYCR;
        break;

    case V4L2_PIX_FMT_YVYU:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 1;
        frame->order_1p = IN_ORDER422_YCRYCB;
        break;

    case V4L2_PIX_FMT_UYVY:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 1;
        frame->order_1p = IN_ORDER422_CBYCRY;
        break;

    case V4L2_PIX_FMT_VYUY:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 1;
        frame->order_1p = IN_ORDER422_CRYCBY;
        break;

    case V4L2_PIX_FMT_NV16:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 2;
        frame->order_1p = LSB_CBCR;
        break;

    case V4L2_PIX_FMT_NV61:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 2;
        frame->order_1p = LSB_CRCB;
        break;

    case V4L2_PIX_FMT_NV16X:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 2;
        frame->order_1p = MSB_CBCR;
        break;

    case V4L2_PIX_FMT_NV61X:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 2;
        frame->order_1p = MSB_CRCB;
        break;

    case V4L2_PIX_FMT_YUV422P:
        frame->format = FORMAT_YCBCR422;
        frame->planes = 3;
        break;
    }
}
应用层
static int video_setfmt(int fd)
{
  /***********set Stream data format********/
  int ret = 0;
  struct v4l2_format fmt;
```

```
    CLEAR(fmt);
    fmt.type              =   V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.width     =   640;
    fmt.fmt.pix.height    =   480;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;//for PAL
    fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;

    ret = ioctl(fd, VIDIOC_S_FMT, &fmt);
    if (ret < 0) {
        perror("VIDIOC_S_FMT");
        return ret;
    }

    return 0;
}
```

3.6 视频格式查询

在v4l2中，有两种查询视频格式的方法，一个是遍历所有视频格式的

一个是查询出一种格式的

/*查询出一种格式*/

ret = ioctl(fd, VIDIOC_G_FMT, &fmt);

/*遍历所有视频格式,查询驱动所支持的格式*/

VIDIOC_ENUM_FMT

```
驱动
static int s3c_fimc_v4l2_g_fmt_vid_cap(struct file *filp, void *fh,
                    struct v4l2_format *f)
```

```
{
    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
    int size = sizeof(struct v4l2_pix_format);

    memset(&f->fmt.pix, 0, size);
    memcpy(&f->fmt.pix, &(ctrl->v4l2.frmbuf.fmt), size);

    return 0;
}
应用
static int video_getfmt(int fd)
{
    /***********get Stream data format********/
    int ret= 0;
    struct v4l2_format fmt;
    CLEAR(fmt);
    fmt.type    =   V4L2_BUF_TYPE_VIDEO_CAPTURE;
    ret = ioctl(fd, VIDIOC_G_FMT, &fmt);
    if (ret < 0) {
        perror("VIDIOC_G_FMT");
        return ret;
    }
    printf("/n**********vidioc get stream format informations:****\n");
    if (fmt.fmt.pix.pixelformat == V4L2_PIX_FMT_YUYV)
        printf("8-bit YUYVV pixel format\n");
        printf("Size of the buffer = %d\n", fmt.fmt.pix.sizeimage);
        printf("Line offset = %d\n", fmt.fmt.pix.bytesperline);
    if (fmt.fmt.pix.field == V4L2_FIELD_INTERLACED)
        printf("Storate format is interlaced frame format\n");

    return 0;
}
```

3.7 向驱动申请帧缓冲，内存，一般不超过5个,帧缓冲管理

```
结构体
struct v4l2_requestbuffers req;
ret = ioctl(fd, VIDIOC_REQBUFS, &req);
ret = ioctl(fd, VIDIOC_QUERYBUF, &buf);//读取缓存

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

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;
};
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，最后把这些缓存放入
缓存队列。
The main steps that the application must perform for buffer allocation are:
Allocating Memory
Getting Physical Address
Mapping Kernel Space Address to User Space
驱动支持

static int s3c_fimc_v4l2_reqbufs(struct file *filp, void *fh,
                    struct v4l2_requestbuffers *b)
{
    if (b->memory != V4L2_MEMORY_MMAP) {
        err("V4L2_MEMORY_MMAP is only supported\n");
        return -EINVAL;
    }

    /* control user input */
    if (b->count > 4)
        b->count = 4;
    else if (b->count < 1)
        b->count = 1;

    return 0;
}
static int s3c_fimc_v4l2_querybuf(struct file *filp, void *fh,
                    struct v4l2_buffer *b)
{
    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;

    if (b->type != V4L2_BUF_TYPE_VIDEO_OVERLAY && \
        b->type != V4L2_BUF_TYPE_VIDEO_CAPTURE)
        return -EINVAL;

    if (b->memory != V4L2_MEMORY_MMAP)
        return -EINVAL;

    b->length = ctrl->out_frame.buf_size;

    /*
     * NOTE: we use the m.offset as an index for multiple frames out.
     * Because all frames are not contiguous, we cannot use it as
     * original purpose.
     * The index value used to find out which frame user wants to mmap.
     */
    b->m.offset = b->index * PAGE_SIZE;

    return 0;
}
static int s3c_fimc_v4l2_qbuf(struct file *filp, void *fh,
              struct v4l2_buffer *b)
{
    return 0;
}
应用层
static int video_mmap(int fd)
{
    /*******step 1*****requestbuffers Allocating Memory *******/
    int ret = 0;
    struct v4l2_requestbuffers req;
    CLEAR(req);
    req.count   = 4;
    req.type    = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    req.memory  = V4L2_MEMORY_MMAP;

    ret = ioctl(fd, VIDIOC_REQBUFS, &req);
    if (ret < 0) {
        perror("VIDIOC_REQBUFS");
        return ret;
    }

    if (req.count < 2)
        printf("insufficient buffer memory\n");
        printf("Number of buffers allocated = %d\n", req.count);

    /*******step 2*****Getting Physical Address  *******/
    buffers = calloc(req.count, sizeof(*buffers));
    for (n_buffers = 0; n_buffers < req.count; ++n_buffers)
    {
        struct v4l2_buffer buf;//驱动中的一帧
        CLEAR(buf);
        buf.type    = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory  = V4L2_MEMORY_MMAP;
        buf.index   = n_buffers;

        ret = ioctl(fd, VIDIOC_QUERYBUF, &buf);
        if (ret < 0) {
            perror("VIDIOC_QUERYBUF");
            return ret;
        }

    /*******step 3*****Mapping Kernel Space Address to User Space*******/
        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)
        //perror("mmap failed \n");
    }

    /************requestbuffers in queue***********/
    for (i = 0; i < n_buffers; ++i) {
        struct v4l2_buffer buf;
        CLEAR(buf);

        buf.type    = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory  = V4L2_MEMORY_MMAP;
        buf.index = i;

        ret = ioctl(fd, VIDIOC_QBUF, &buf);//申请的缓冲进入队列
        if (ret < 0) {
            perror("VIDIOC_QBUF");
            return ret;
        }
    }

    return 0;
}
```

3.8 开始捕捉图像数据(重要)

```
<PRE class=csharp name="code">

结构体

<PRE class=csharp name="code">enum v4l2_buf_type type;//开始捕捉图像数据
    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    ret = ioctl(fd, VIDIOC_STREAMON, &type);

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,
#if 1
    /* Experimental */
    V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY = 8,
#endif
```

```
    V4L2_BUF_TYPE_PRIVATE            = 0x80,
};

驱动

static int s3c_fimc_v4l2_streamon(struct file *filp, void *fh,
                    enum v4l2_buf_type i)
{
    struct s3c_fimc_control *ctrl = (struct s3c_fimc_control *) fh;
    if (i != V4L2_BUF_TYPE_VIDEO_CAPTURE)
        return -EINVAL;
printk("s3c_fimc_v4l2_streamon is called\n");
    if (ctrl->in_type != PATH_IN_DMA)
        s3c_fimc_init_camera(ctrl);

    ctrl->out_frame.skip_frames = 0;
    FSET_CAPTURE(ctrl);
    FSET_IRQ_NORMAL(ctrl);
    s3c_fimc_start_dma(ctrl);

    return 0;
}
硬件控制寄存器的配置
应用层
static int video_streamon(int fd)
{
    int ret = 0;

    /************start stream on***********/

    enum v4l2_buf_type types;//开始捕捉图像数据
    types = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    ret = ioctl(fd, VIDIOC_STREAMON, &types);
    if (ret < 0) {
        perror("VIDIOC_STREAMON");
        return ret;
    }

    return 0;
}
```

阅读(4086) | 评论(0) | 转发(8) |

上一篇：嵌入式系统启动例程
下一篇：深入分析 Linux 内核链表

0

相关热门文章

Android Multimedia Framework...

Linux设备驱动之I2C架构分析...

自己学着写lcd入门级驱动程序...

linux守护进程的几个关键地方...

stagefright与opencore对比

嵌入式Linux之我行——u-boot-...

简单字符设备驱动程序...

嵌入式Memo 15 SDRAM On May 6...

自己学着写lcd入门级驱动程序...

网站制作	嵌入式Linux之我行——内核、...	Linux2.6.35下s3c64xx/spi子系...
特别是云计算和大数据技术的快...	android的logcat详细用法	编译vlc+live555,支持rtsp流播...

给主人留下些什么吧！~~

评论热议

请登录后评论。

登录 注册