个人资料

DroidPhone

访问： 1072927次
积分： 8742
等级： BLOG 6
排名： 第1446名

原创： 51篇 转载： 0篇
译文： 4篇 评论： 536条

Linux ALSA声卡驱动之三：PCM设备的创建

标签： linux playback struct stream file autoload

2011-04-07 21:18 41422人阅读 评论(42) 收藏 举报

分类： Linux设备驱动（19） Linux音频子系统（14）

声明：本博内容均由http://blog.csdn.net/droidphone原创，转载请注明出处，谢谢！

1. PCM是什么

PCM是英文Pulse-code modulation的缩写，中文译名是脉冲编码调制。我们知道在现实生活中，人耳听到的声音是模拟信号，PCM就是要把声音从模拟转换成数字信号的一种技术，他的原理简单地说就是利用一个固定的频率对模拟信号进行采样，采样后的信号在波形上看就像一串连续的幅值不一的脉冲，把这些脉冲的幅值按一定的精度进行量化，这些量化后的数值被连续地输出、传输、处理或记录到存储介质中，所有这些组成了数字音频的产生过程。

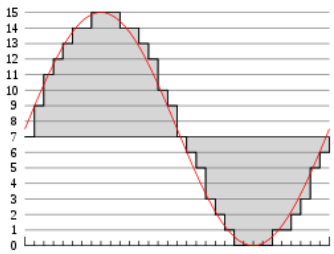

图1.1 模拟音频的采样、量化

PCM信号的两个重要指标是采样频率和量化精度，目前，CD音频的采样频率通常为44100Hz，量化精度是16bit。通常，播放音乐时，应用程序从存储介质中读取音频数据（MP3、WMA、AAC.....），经过解码后，最终送到音频驱动程序中的就是PCM数据，反过来，在录音时，音频驱动不停地把采样所得的PCM数据送回给应用程序，由应用程序完成压缩、存储等任务。所以，音频驱动的两大核心任务就是：

- playback 如何把用户空间的应用程序发过来的PCM数据，转化为人耳可以辨别的模拟音频
- capture 把mic拾取到得模拟信号，经过采样、量化，转换为PCM信号送回给用户空间的应用程序

2. alsa-driver中的PCM中间层

ALSA已经为我们实现了功能强劲的PCM中间层，自己的驱动中只要实现一些底层的需要访问硬件的函数即可。

要访问PCM的中间层代码，你首先要包含头文件<sound/pcm.h>，另外，如果需要访问一些与 hw_param相关的函数，可能也要包含<sound/pcm_params.h>。

Android Audio System 之 (37372)
Linux时间子系统之六：高 (36277)
Linux ALSA声卡驱动之七 (36208)
Linux ALSA声卡驱动之四 (36031)
(31602)

评论排行

Android Audio System 之 (56)
Linux ALSA声卡驱动之三 (42)
Linux ALSA声卡驱动之八 (35)
Linux时间子系统之六：高 (25)
Linux中断（interrupt）子 (24)
Android SurfaceFlinger中 (21)
Linux ALSA声卡驱动之二 (19)
Android Audio System 之 (18)
Linux ALSA声卡驱动之七 (17)
Linux中断（interrupt）子 (17)

推荐文章

* 致JavaScript也将征服的物联网世界
* 从苏宁电器到卡巴斯基：难忘的三年硕士时光
* 作为一名基层管理者如何利用情商管理自己和团队（一）
* Android CircleImageView圆形ImageView
* 高质量代码的命名法则

最新评论

Linux时间子系统之一：clock sou
zhqh100: 1.2 read回调函数时钟源本身不会产生中断，要获得时钟源的当前计数，只能通过主动调用它的rea...

Linux中断（interrupt）子系统之二
liunix61: 大神！必须顶@！！！

Linux时间子系统之三：时间的维
Kevin_Smart: 学习了

Linux时间子系统之六：高精度定
Kevin_Smart: 像楼主说的，原则上，hrtimer是利用一个硬件计数器来实现的，所以精度才可以做到ns级别。硬件的计...

Linux中断（interrupt）子系统之二
12期-马金兴: 恩，虽然这么多字但是我要好好学习一下

已知二叉树的前序遍历和中序遍历
重修月: 很喜欢博主的文章，刚刚用豆约翰博客备份专家备份了您的全部博文。

Linux ALSA声卡驱动之三：PCM
灿哥哥: 学习了

Android Audio System 之三： Au
ss0429: 楼主的文章写的很精炼，多谢分享~

Linux中断（interrupt）子系统之二
KrisFei: 针对这句话有两个问题想讨论下：1. disable_irq()放在中断上半部有会导致死锁。2. 如果...

Linux ALSA声卡驱动之七：ASoC
Wit-Z-Joy: 博主您好，我现在需要解决苹果耳机麦克风不能使用的问题。我的设备也是美标的耳机口，市面上常见的魅族，小...

每个声卡最多可以包含4个pcm的实例，每个pcm实例对应一个pcm设备文件。pcm实例数量的这种限制源于linux设备号所占用的位大小，如果以后使用64位的设备号，我们将可以创建更多的pcm实例。不过大多数情况下，在嵌入式设备中，一个pcm实例已经足够了。

一个pcm实例由一个playback stream和一个capture stream组成，这两个stream又分别有一个或多个substreams组成。

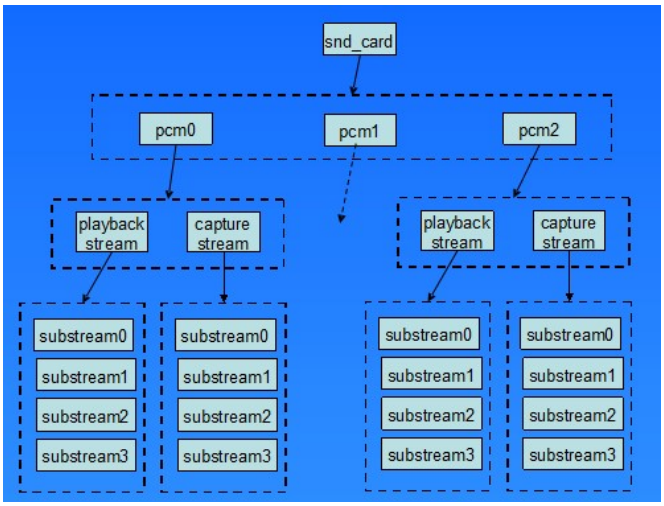

图2.1 声卡中的pcm结构

在嵌入式系统中，通常不会像图2.1中这么复杂，大多数情况下是一个声卡，一个pcm实例，pcm下面有一个playback和capture stream，playback和capture下面各自有一个substream。

下面一张图列出了pcm中间层几个重要的结构，他可以让我们从uml的角度看一看这列结构的关系，理清他们之间的关系，对我们理解pcm中间层的实现方式。

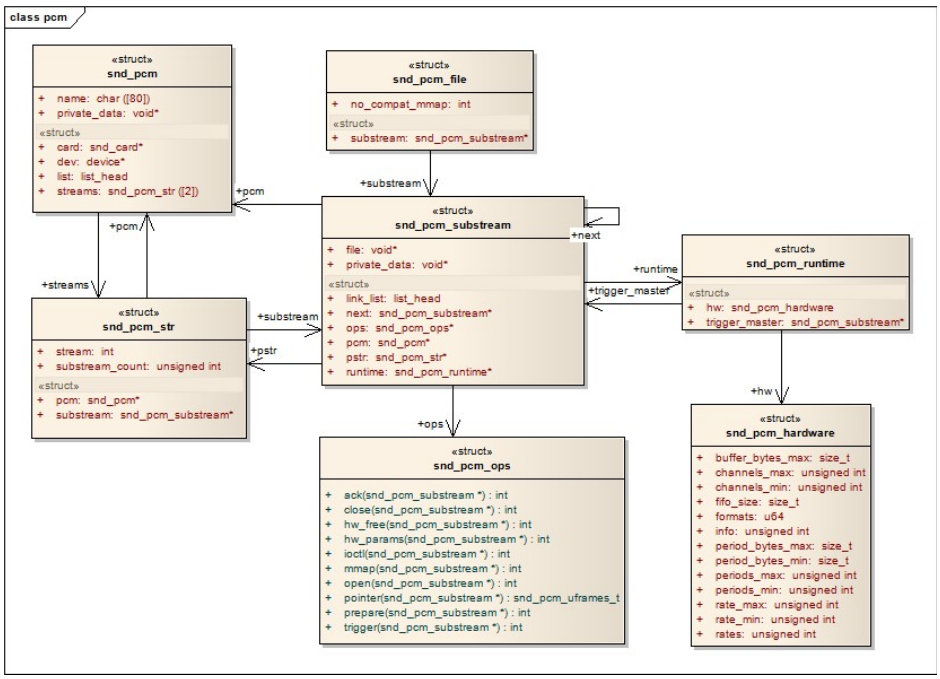

图2.2 pcm中间层的几个重要的结构体的关系图

- snd_pcm是挂在snd_card下面的一个snd_device

- snd_pcm中的字段：streams[2]，该数组中的两个元素指向两个snd_pcm_str结构，分别代表playback stream和capture stream
- snd_pcm_str中的substream字段，指向snd_pcm_substream结构
- snd_pcm_substream是pcm中间层的核心，绝大部分任务都是在substream中处理，尤其是他的ops（snd_pcm_ops）字段，许多user空间的应用程序通过alsa-lib对驱动程序的请求都是由该结构中的函数处理。它的runtime字段则指向snd_pcm_runtime结构，snd_pcm_runtime记录这substream的一些重要的软件和硬件运行环境和参数。

3. 新建一个pcm

alsa-driver的中间层已经为我们提供了新建pcm的api：

```
int snd_pcm_new(struct snd_card *card, const char *id, int device, int playback_count, int capture_count,
                struct snd_pcm ** rpcm);
```

参数device 表示目前创建的是该声卡下的第几个pcm，第一个pcm设备从0开始。

参数playback_count 表示该pcm将会有几个playback substream。

参数capture_count 表示该pcm将会有几个capture substream。

另一个用于设置pcm操作函数接口的api：

```
void snd_pcm_set_ops(struct snd_pcm *pcm, int direction, struct snd_pcm_ops *ops);
```

新建一个pcm可以用下面一张新建pcm的调用的序列图进行描述：

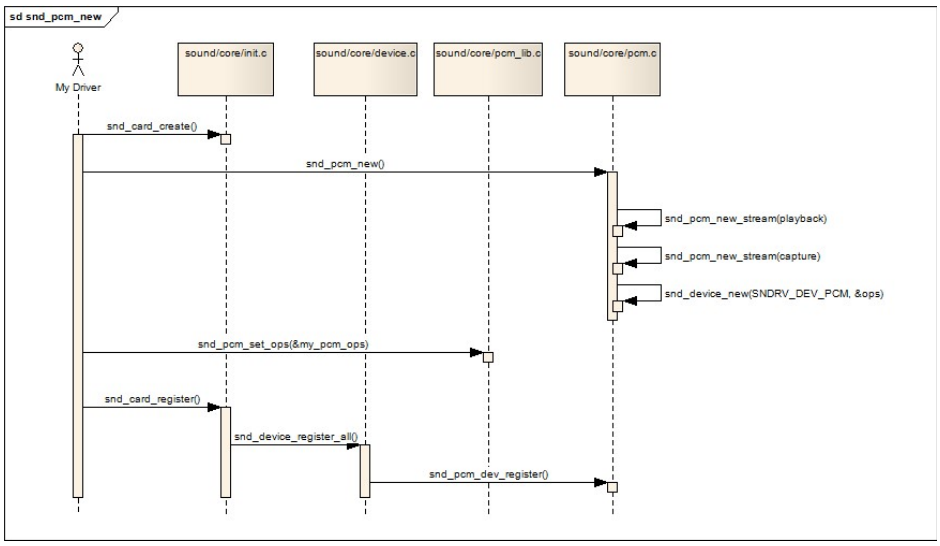

图3.1 新建pcm的序列图

- **snd_card_create**　pcm是声卡下的一个设备（部件），所以第一步是要创建一个声卡

- **snd_pcm_new**　调用该api创建一个pcm，才该api中会做以下事情
  - 如果有，建立playback stream，相应的substream也同时建立
  - 如果有，建立capture stream，相应的substream也同时建立
  - 调用snd_device_new()把该pcm挂到声卡中，参数ops中的dev_register字段指向了函数snd_pcm_dev_register，这个回调函数会在声卡的注册阶段被调用。
- **snd_pcm_set_ops**　设置操作该pcm的控制/操作接口函数，参数中的snd_pcm_ops结构中的函数通常就是我们驱动要实现的函数
- **snd_card_register**　注册声卡，在这个阶段会遍历声卡下的所有逻辑设备，并且调用各设备的注册回调函数，对于pcm，就是第二步提到的snd_pcm_dev_register函数，该回调函数建立了和用户空间应用程序（alsa-lib）通信所用的设备文件节点:/dev/snd/pcmCxxDxxp和/dev/snd/pcmCxxDxxc

4. 设备文件节点的建立（dev/snd/pcmCxxDxxp、pcmCxxDxxc）

4.1 struct snd_minor

每个snd_minor结构体保存了声卡下某个逻辑设备的上下文信息，他在逻辑设备建立阶段被填充，在逻辑设备被使用时就可以从该结构体中得到相应的信息。pcm设备也不例外，也需要使用该结构体。该结构体在include/sound/core.h中定义。

```c-sharp
struct snd_minor {
    int type;           /* SNDRV_DEVICE_TYPE_XXX */
    int card;           /* card number */
    int device;         /* device number */
    const struct file_operations *f_ops;    /* file operations */
    void *private_data;     /* private data for f_ops->open */
    struct device *dev;     /* device for sysfs */
};
```

在sound/sound.c中定义了一个snd_minor指针的全局数组：

```c-sharp
static struct snd_minor *snd_minors[256];
```

前面说过，在声卡的注册阶段（snd_card_register），会调用pcm的回调函数snd_pcm_dev_register()，这个函数里会调用函数snd_register_device_for_dev()：

```c-sharp
static int snd_pcm_dev_register(struct snd_device *device)
{
    ......

    /* register pcm */
    err = snd_register_device_for_dev(devtype, pcm->card,
                          pcm->device,
                  &snd_pcm_f_ops[cidx],
                  pcm, str, dev);
    ......
}
```

我们再进入snd_register_device_for_dev()：

```c-sharp
int snd_register_device_for_dev(int type, struct snd_card *card, int dev,
                const struct file_operations *f_ops,
                void *private_data,
                const char *name, struct device *device)
{
    int minor;
    struct snd_minor *preg;

    if (snd_BUG_ON(!name))
        return -EINVAL;
    preg = kmalloc(sizeof *preg, GFP_KERNEL);
    if (preg == NULL)
        return -ENOMEM;
    preg->type = type;
    preg->card = card ? card->number : -1;
    preg->device = dev;
```

```
    preg->f_ops = f_ops;
    preg->private_data = private_data;
    mutex_lock(&sound_mutex);
#ifdef CONFIG_SND_DYNAMIC_MINORS
    minor = snd_find_free_minor();
#else
    minor = snd_kernel_minor(type, card, dev);
    if (minor >= 0 && snd_minors[minor])
        minor = -EBUSY;
#endif
    if (minor < 0) {
        mutex_unlock(&sound_mutex);
        kfree(preg);
        return minor;
    }
    snd_minors[minor] = preg;
    preg->dev = device_create(sound_class, device, MKDEV(major, minor),
                  private_data, "%s", name);
    if (IS_ERR(preg->dev)) {
        snd_minors[minor] = NULL;
        mutex_unlock(&sound_mutex);
        minor = PTR_ERR(preg->dev);
        kfree(preg);
        return minor;
    }

    mutex_unlock(&sound_mutex);
    return 0;
}
```

- 首先，分配并初始化一个snd_minor结构中的各字段
  - type：SNDRV_DEVICE_TYPE_PCM_PLAYBACK/SNDRV_DEVICE_TYPE_PCM_CAPTURE
  - card: card的编号
  - device：pcm实例的编号，大多数情况为0
  - f_ops：snd_pcm_f_ops
  - private_data：指向该pcm的实例
- 根据type，card和pcm的编号，确定数组的索引值minor，minor也作为pcm设备的此设备号
- 把该snd_minor结构的地址放入全局数组snd_minors[minor]中
- 最后，调用device_create创建设备节点

4.2 设备文件的建立

在4.1节的最后，设备文件已经建立，不过4.1节的重点在于snd_minors数组的赋值过程，在本节中，我们把重点放在设备文件中。

回到pcm的回调函数snd_pcm_dev_register()中：

```
[c-sharp]

static int snd_pcm_dev_register(struct snd_device *device)
{
    int cidx, err;
    char str[16];
    struct snd_pcm *pcm;
    struct device *dev;

    pcm = device->device_data;
         ......
    for (cidx = 0; cidx < 2; cidx++) {
                  ......
        switch (cidx) {
        case SNDRV_PCM_STREAM_PLAYBACK:
            sprintf(str, "pcmC%iD%ip", pcm->card->number, pcm->device);
            devtype = SNDRV_DEVICE_TYPE_PCM_PLAYBACK;
            break;
        case SNDRV_PCM_STREAM_CAPTURE:
            sprintf(str, "pcmC%iD%ic", pcm->card->number, pcm->device);
            devtype = SNDRV_DEVICE_TYPE_PCM_CAPTURE;
```

```
            break;
        }
        /* device pointer to use, pcm->dev takes precedence if
         * it is assigned, otherwise fall back to card's device
         * if possible */
        dev = pcm->dev;
        if (!dev)
            dev = snd_card_get_device_link(pcm->card);
        /* register pcm */
        err = snd_register_device_for_dev(devtype, pcm->card,
                          pcm->device,
                          &snd_pcm_f_ops[cidx],
                          pcm, str, dev);
                  ......
    }
         ......
}
```

以上代码我们可以看出，对于一个pcm设备，可以生成两个设备文件，一个用于playback，一个用于capture，代码中也确定了他们的命名规则：

- playback -- pcmCxDxp，通常系统中只有一各声卡和一个pcm，它就是pcmC0D0p
- capture -- pcmCxDxc，通常系统中只有一各声卡和一个pcm，它就是pcmC0D0c

snd_pcm_f_ops

snd_pcm_f_ops是一个标准的文件系统file_operations结构数组，它的定义在sound/core/pcm_native.c中：

```c-sharp
const struct file_operations snd_pcm_f_ops[2] = {
    {
        .owner =        THIS_MODULE,
        .write =        snd_pcm_write,
        .aio_write =        snd_pcm_aio_write,
        .open =         snd_pcm_playback_open,
        .release =      snd_pcm_release,
        .llseek =       no_llseek,
        .poll =         snd_pcm_playback_poll,
        .unlocked_ioctl =   snd_pcm_playback_ioctl,
        .compat_ioctl =     snd_pcm_ioctl_compat,
        .mmap =         snd_pcm_mmap,
        .fasync =       snd_pcm_fasync,
        .get_unmapped_area =    snd_pcm_get_unmapped_area,
    },
    {
        .owner =        THIS_MODULE,
        .read =         snd_pcm_read,
        .aio_read =     snd_pcm_aio_read,
        .open =         snd_pcm_capture_open,
        .release =      snd_pcm_release,
        .llseek =       no_llseek,
        .poll =         snd_pcm_capture_poll,
        .unlocked_ioctl =   snd_pcm_capture_ioctl,
        .compat_ioctl =     snd_pcm_ioctl_compat,
        .mmap =         snd_pcm_mmap,
        .fasync =       snd_pcm_fasync,
        .get_unmapped_area =    snd_pcm_get_unmapped_area,
    }
};
```

snd_pcm_f_ops作为snd_register_device_for_dev的参数被传入，并被记录在snd_minors[minor]中的字段f_ops中。最后，在snd_register_device_for_dev中创建设备节点：

```c-sharp
snd_minors[minor] = preg;
preg->dev = device_create(sound_class, device, MKDEV(major, minor),
              private_data, "%s", name);
```

4.3 层层深入，从应用程序到驱动层pcm

4.3.1 字符设备注册

在sound/core/sound.c中有alsa_sound_init()函数，定义如下：

```c-sharp
static int __init alsa_sound_init(void)
{
    snd_major = major;
    snd_ecards_limit = cards_limit;
    if (register_chrdev(major, "alsa", &snd_fops)) {
        snd_printk(KERN_ERR "unable to register native major device number %d/n", major);
        return -EIO;
    }
    if (snd_info_init() < 0) {
        unregister_chrdev(major, "alsa");
        return -ENOMEM;
    }
    snd_info_minor_register();
    return 0;
}
```

register_chrdev中的参数major与之前创建pcm设备是device_create时的major是同一个，这样的结果是，当应用程序open设备文件/dev/snd/pcmCxDxp时，会进入snd_fops的open回调函数，我们将在下一节中讲述open的过程。

4.3.2 打开pcm设备

从上一节中我们得知，open一个pcm设备时，将会调用snd_fops的open回调函数，我们先看看snd_fops的定义：

```c-sharp
static const struct file_operations snd_fops =
{
    .owner =    THIS_MODULE,
    .open =     snd_open
};
```

跟入snd_open函数，它首先从inode中取出此设备号，然后以次设备号为索引，从snd_minors全局数组中取出当初注册pcm设备时填充的snd_minor结构（参看4.1节的内容），然后从snd_minor结构中取出pcm设备的f_ops，并且把file->f_op替换为pcm设备的f_ops，紧接着直接调用pcm设备的f_ops->open()，然后返回。因为file->f_op已经被替换，以后，应用程序的所有read/write/ioctl调用都会进入pcm设备自己的回调函数中，也就是4.2节中提到的snd_pcm_f_ops结构中定义的回调。

```c-sharp
static int snd_open(struct inode *inode, struct file *file)
{
    unsigned int minor = iminor(inode);
    struct snd_minor *mptr = NULL;
    const struct file_operations *old_fops;
    int err = 0;

    if (minor >= ARRAY_SIZE(snd_minors))
        return -ENODEV;
    mutex_lock(&sound_mutex);
    mptr = snd_minors[minor];
    if (mptr == NULL) {
        mptr = autoload_device(minor);
        if (!mptr) {
            mutex_unlock(&sound_mutex);
            return -ENODEV;
        }
    }
    old_fops = file->f_op;
    file->f_op = fops_get(mptr->f_ops);
    if (file->f_op == NULL) {
        file->f_op = old_fops;
        err = -ENODEV;
    }
    mutex_unlock(&sound_mutex);
    if (err < 0)
        return err;

    if (file->f_op->open) {
        err = file->f_op->open(inode, file);
        if (err) {
```

```
            fops_put(file->f_op);
            file->f_op = fops_get(old_fops);
        }
    }
    fops_put(old_fops);
    return err;
}
```

下面的序列图展示了应用程序如何最终调用到snd_pcm_f_ops结构中的回调函数：

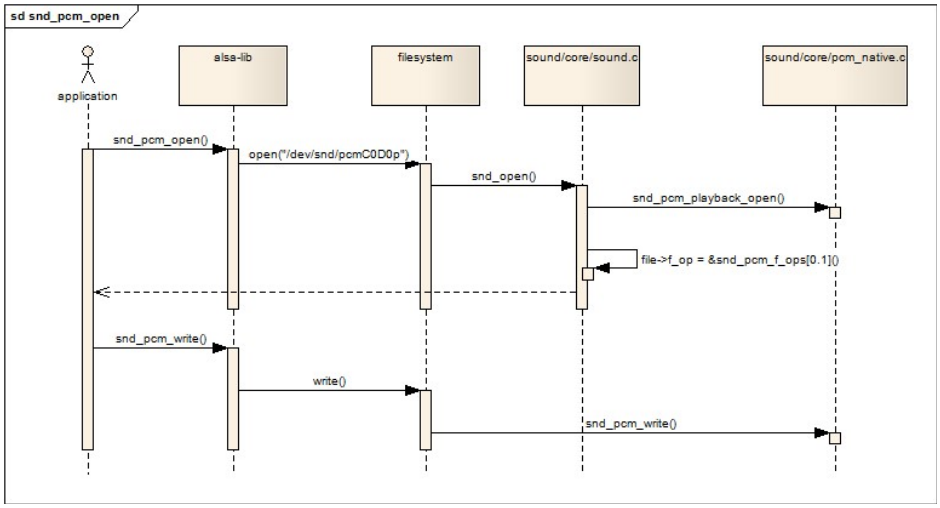

图4.3.2.1　应用程序操作pcm设备

上一篇　Linux ALSA声卡驱动之二：声卡的创建
下一篇　Linux ALSA声卡驱动之四：Control设备的创建

我的同类文章

Linux设备驱动（19）　Linux音频子系统（14）

- Linux SPI总线和设备驱动架… 2014-04-28 阅读 7309
- Linux SPI总线和设备驱动架… 2014-04-18 阅读 7867
- Linux中断（interrupt）子系… 2012-05-01 阅读 20869
- Linux中断（interrupt）子系… 2012-04-24 阅读 13737
- Linux中断（interrupt）子系… 2012-04-12 阅读 29937
- Linux ALSA声卡驱动之八：… 2012-03-13 阅读 31172
- Linux SPI总线和设备驱动架… 2014-04-23 阅读 7288
- Linux SPI总线和设备驱动架… 2014-04-12 阅读 10664
- Linux中断（interrupt）子系… 2012-04-27 阅读 14294
- Linux中断（interrupt）子系… 2012-04-18 阅读 14451
- 自旋锁spin_lock和raw_spin… 2012-03-26 阅读 11824

更多文章

猜你在找

Android底层技术：Linux驱动框架与开发　Linux ALSA声卡驱动之三PCM设备的创建
linux嵌入式开发+驱动开发　Linux ALSA声卡驱动之三PCM设备的创建
从零写Bootloader及移植uboot、linux内核、文件系统和Linux ALSA声卡驱动之三PCM设备的创建
Linux设备驱动开发入门　Linux ALSA声卡驱动之三PCM设备的创建
嵌入式Linux项目实战：三个大项目(数码相框、摄像头驱Linux ALSA声卡驱动之三PCM设备的创建

查看评论

28楼 灿哥哥 2016-04-18 16:36发表

学习了

27楼 saviourxx 2016-02-22 11:23发表

写得不是一般的好～如果还不理解就是自身问题了

26楼 codecoco 2015-04-24 15:08发表

有一种拨云见日的感觉，思路一下子捋顺了，

25楼 bob_fly1984 2015-04-24 15:00发表

思路清晰，不错

24楼 wangwu125 2015-01-06 17:10发表

写的非常好，不得不赞一个。。好奇楼主看内核为什么如此深刻，有什么窍门可用指教下吗？

Re: DroidPhone 2015-01-13 19:01发表

回复wangwu125：一句话，贵在坚持！

23楼 lizhijin48 2014-07-31 10:52发表

楼主写的确实非常好，令我受益匪浅。非常感谢，不过有个地方还想请指教一下：
alsa_sound_init()，这个函数是每注册一个组建(比如pcm)就会执行一次吗？还是每注册一个声卡执行一次？
因为看源码似乎只注册了一次。可是按照你的描述，应该是每个pcm创建应该会对应一次alsa_sound_init()函数的执行。
忘解惑，万分感谢！

Re: DroidPhone 2014-08-01 18:03发表

回复lizhijin48：不会的，alsa_sound_init函数被加上了__init修饰符，只会在初始化阶段执行一次。

22楼 Norton-Linux内核研究 2014-07-14 17:46发表

博主，写的真好，无敌了

21楼 jfh0813 2014-07-07 14:09发表

感谢楼主回复 就是因为应用层不直接跟驱动打交道 导致我搞不清到底ioctl是不是调用的snd_pcm_playback_ioctl 因为3.0以后的UDA1341的驱动在S3C2440上移植后有BUG 但是一直跟踪不到原因 故想搞清楚 望楼主指点

Re: DroidPhone 2014-07-07 15:34发表

回复jfh0813：alsa-lib确实太过庞大和臃肿，以致现在被android放弃了。不过你可以参考一下android源码里的替代实现：tinyalsa，它基本实现了我们想要的功能，想了解应用到底是如何和驱动打交道的话，阅读一下tinyalsa的源码就知道了。

20楼 jfh0813 2014-07-06 15:26发表

楼主 你好
最近在做声卡驱动 你的文章写的太好了 帮了我很大忙 很感谢 但是还有些问题搞不清楚想请教你
应用层是怎么设置采样频率的？因为ALSA现在都基本使用ALSA提供的应用层库 很难跟踪到代码 我个人的理解是对设备文件pcmC0D0p进行ioctl 这样就会调snd_pcm_playback_ioctl
函数 但是 参数是怎么传的呢？比如 我现在的音频文件采样频率是44.1K？
谢谢
另外 很想跟楼主认识 QQ 455695583 期待加我！！！

Re: DroidPhone 2014-07-07 09:42发表

回复jfh0813：应用层一般不会直接和驱动打交道，而是通过alsa-lib和驱动交互，应用层直接使用alsa-lib提供的API进行编程，你可以搜索一下关于alsa-lib相关的使用方法。

19楼 恒赌东道 2014-07-04 12:13发表

楼主您好，在不断的学习中非常感谢您的文章和您的解答，我仍在学习，不过困惑不断，DMA传输这块的内容，楼主可不可以做个详细的分析，还有就是有个结构体snd_pcm_substream是不是继承了很多上层传来的参数？还是对这一块比较模糊！谢谢回复！

18楼 恒赌东道 2014-06-25 18:02发表

您好：请教，当进入snd_pcm_open_substream(pcm, stream, file, &substream); 在调用substream->ops->open(substream);我想问下次打开的应该是对应设备的open了。这个关系不是太清楚，可以详细说下么？谢谢！

Re: shuaishuaio123 2014-06-26 23:46发表

回复WOAICJIA：substream->ops->open(substream);这里的ops在soc_new_pcm这个函数里被赋值了，其实就是调用soc_pcm_open这个函数，你找到soc_new_pcm这个函数，之后跟一下，你就明白了。

Re: DroidPhone 2014-06-26 13:40发表

回复WOAICJIA：你说对"下次打开"是指什么？不太明白你想表达对疑问。

Re: 恒赌东道 2014-06-28 16:08发表

回复DroidPhone：非常感谢，明白了！

17楼 ky0012110 2014-05-19 10:03发表

能帮我做一个软件吗？
主要是 解析PCM 96/24 或者 96/26 的音频软件！
将音频数据恢复成 汉子 或者是 数字！

有会做的 我付钱给你！
QQ3-5--9-7-6-44

Re: 无才顽石 2014-05-29 13:34发表

回复ky0012110：语音识别？

16楼 Elven_lsy 2014-03-24 11:33发表

不知道用字符设备驱动如何能够实现播放和录音功能？
我这边已经能通过AC97读到声卡的ID号，但是后面的数据传输这一块不知道怎么弄，不知道你有什么见解？

15楼 DDR2013 2013-12-25 18:05发表

没有建立抽象模型，没有硬件通路的概念，
估计没几个人能搞懂，只不过是自娱自乐的文章。
写的不过是API的调用关系而已。

Re: edaplayer 2015-12-26 14:39发表

回复DDR2013：你没读过书吧，这么简单的文章都看不懂。

Re: fshh520 2014-09-02 10:24发表

回复DDR2013：只有你没搞明白，其他有耐心的人都看懂了，至少我就受益不少

14楼 BBQLOVEYOU 2013-07-02 10:00发表

牛人！

13楼 裂空 2013-06-17 11:54发表

赞一个！

12楼 fjbwinston 2013-06-04 21:37发表

顶一个！！！

11楼 vito_coleone 2013-05-26 20:20发表

真的讲的很好，十分感谢。

10楼 kuangreng 2012-09-21 15:59发表

第二遍阅读，十分感谢！

9楼 dongshengzou95 2012-08-23 17:27发表

看到这里我已经忍不住要说两句了，楼主能把这么复杂的alsa驱动讲解的这么清楚，思路如此清晰，佩服！我之前好几个疑惑，都在这里能找到精确的答案，感激！

8楼 kuangreng 2012-05-24 12:04发表

正在学习，讲得非常好，十分感谢！！！

7楼 A__VS__Z 2012-03-15 16:13发表

的确好，

6楼 阿曼 2011-12-14 22:41发表

真心佩服博主。不仅自己看懂了，还写出来让更过的人看懂，真心感谢。

5楼 stone10000y 2011-06-22 11:37发表

在你的Linux音频驱动之二
里面找到答案了，不好意思，谢谢

4楼 stone10000y 2011-06-22 11:16发表

赞！ 有个问题请教下：创建好的设备文件为什么是在/dev/snd下，不是直接就在/dev下呢？

3楼 ustcxiangchun 2011-06-08 17:14发表

有个疑问请教一下，后面具体会转到哪个芯片的驱动去又是怎么样的呢？期待楼主也讲讲这个

Re: DroidPhone 2011-06-09 12:37发表

回复 ustcxiangchun：对于嵌入式设备，具体到芯片级的要在soc层实现，以后写到soc时才会涉及。

2楼 ustcxiangchun 2011-06-08 17:12发表

写得非常的好呀，赞一个

1楼 sxjyx2009 2011-04-28 09:17发表

[e03]持续关注中

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap