Linux ALSA声卡驱动原理分析

-

设备打开过程和数据流程

目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程
- i. 整体分析
- ii. 设备驱动程序insmod流程图
- iii. 应用程序主流程图
- iv. 声卡打开流程图
- v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程
i. 整体分析
ii. 设备驱动程序insmod流程图
iii. 应用程序主流程图
iv. 声卡打开流程图
v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

一、导读

本文主要针对ALSA声卡硬件驱动，从应用程序的角度展示了从用户层到内核层再到硬件驱动程序控制声卡硬件的过程。主要包括insmod、声卡打开、数据写入三个流程。rmmod和声卡关闭的流程与insmod、声卡打开类似，本文没有描述。ALSA的其他部分如控制、录音等，不在本文叙述范围内。其中的insmod是系统初始化时或手动加载，不需要具体的应用程序参与。具体到硬件，使用的是ENS1371芯片，关于最小化的ENS1371芯片驱动程序，可以参考《 Linux ALSA声卡驱动开发最佳实践.pptx》。

目 录

一、导读

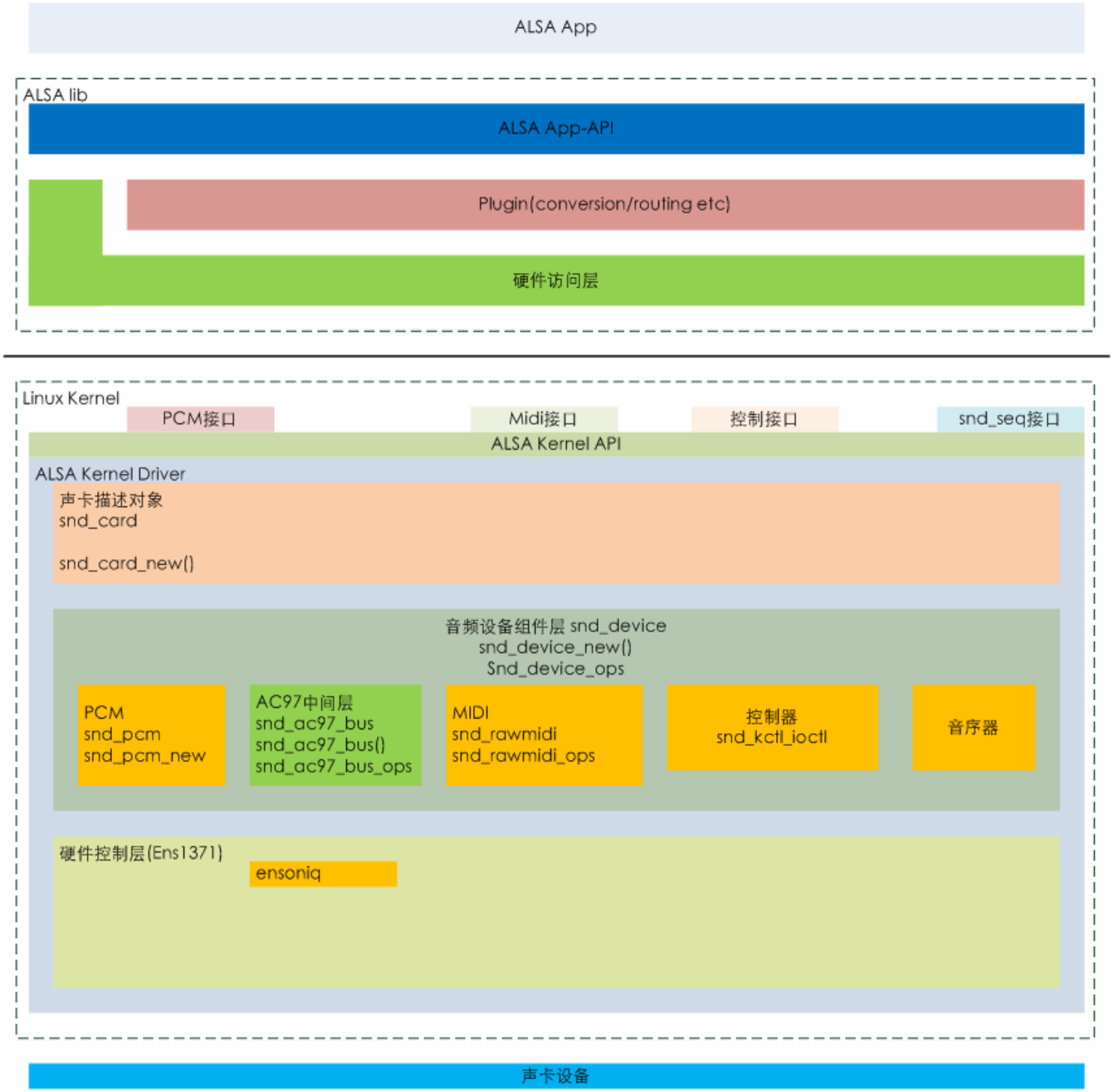
二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程
i. 整体分析
ii. 设备驱动程序insmod流程图
iii. 应用程序主流程图
iv. 声卡打开流程图
v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

二、ALSA架构简介



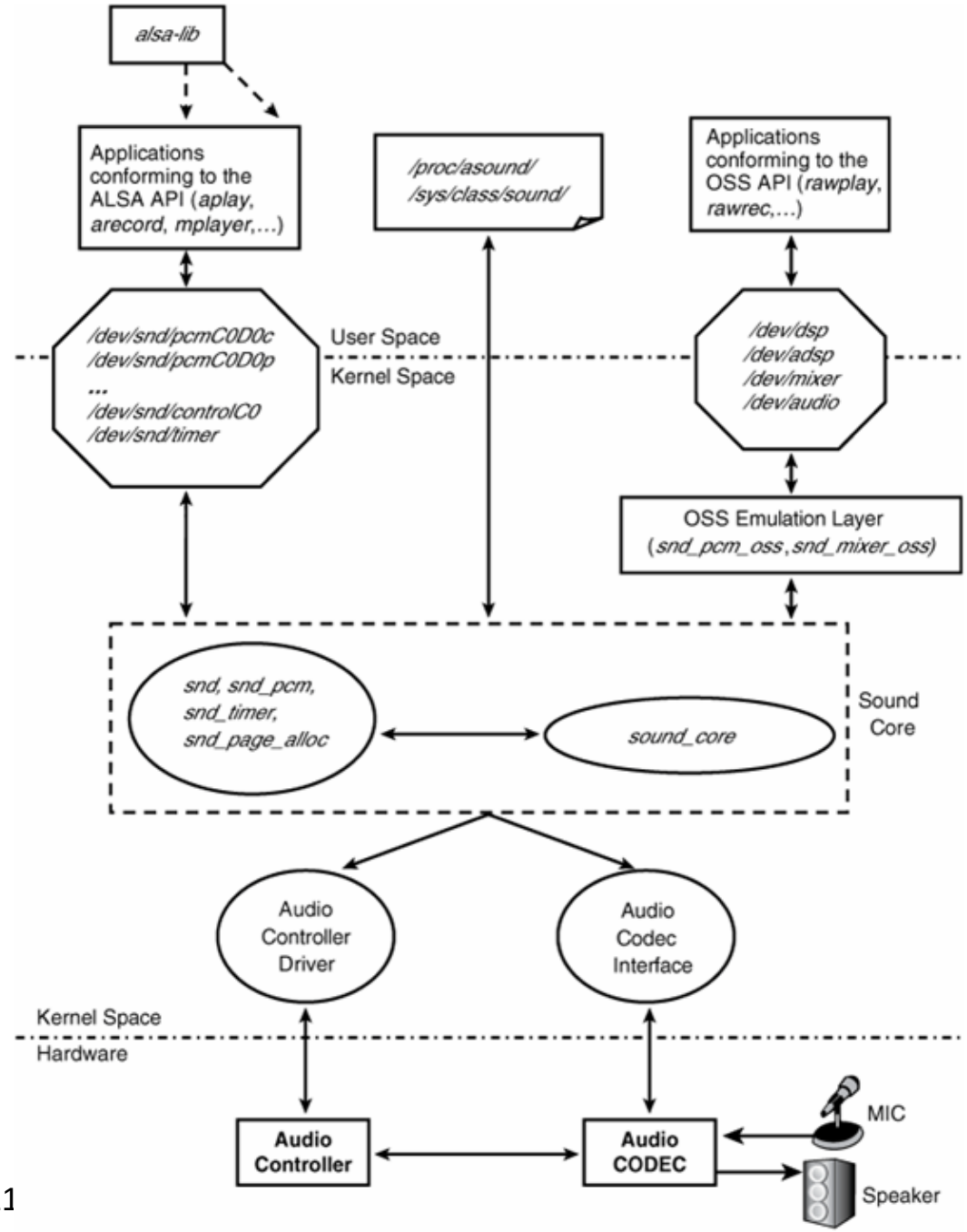
1. ALSA是Advanced Linux Sound Architecture，高级Linux声音架构的简称,它在Linux操作系统上提供了音频和MIDI（Musical Instrument Digital Interface，音乐设备数字化接口）的支持。它包含API 库和工具、内核驱动集合， 对Linux 声音进行支持。 ALSA 包含一系列内核驱动对不同的声卡进行支持，还提供了libasound 的 API 库。用这些进行写程序不需要打开设备等操作，所以编程人员在写程序的时候不会被底层的东西困扰。

2. ALSA自带的应用程序是alsa-utils工具包，包括aplay、alsamixer等。aplay用于在ASLA上播放音频。alsamixer用于改变音频信号的音量。

3. alsa-lib是用户空间的函数库，提供了libasound.so给应用程序使用，应用程序应包含头文件asoundlib.h。这个库通过提供封装函数(ALSA-API)，使ALSA应用程序不需要涉及具体硬件，编写起来更容易。alsa-lib中有control，timer，dmix，pcm等，都是以插件(plugin)的形式存在的。alsa-lib通过硬件访问层的系统调用与内核层进行交互。

4. alsa-driver是音频设备的alsa内核部分的驱动。集成在内核里面，大多是以模块的方式存在。可分为三层。
   (1)最底层是硬件操控层，负责实现硬件操纵访问的功能，这也是声卡驱动程序中用户需实现的主要部分；
   (2)中间层是ASLA驱动的核心部分，它由各种功能的音频设备组件构成，为用户提供了一些预定义组件（如PCM、AC97、音序器和控制器等），另外用户也可以自行定义设备组件；
   (3)驱动的最上层是声卡对象描述层，它是声卡硬件的抽象描述，内核通过这些描述可以得知该声卡硬件的功能、设备组件和操作方法等。

二、ALSA架构简介



左图是从代码的角度体现了alsa-lib和alsa-driver及hardware的交互关系。用户层的alsa-lib通过操作alsa-driver创建的设备文件/dev/snd/pcmC0D0p等对内核层进行访问。内核层的alsa-drivier驱动再经由sound core对硬件声卡芯片进行访问。从而实现了app alsa-lib alsa-driver hardware的操作。

图中右上角OSS相关部分是为了兼容OSS驱动模型而存在的。不是本实践的相关部分。

目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程
i. 整体分析
ii. 设备驱动程序insmod流程图
iii. 应用程序主流程图
iv. 声卡打开流程图
v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

三、准备工作

为了更有效的跟踪ALSA的流程，需要在开始前进行一些准备工作，这包括用户层ALSA-lib库的调试方法和文档生成。alsa-driver的调试方法参见《Linux 基础培训(2)-驱动开发最佳实践-1.pptx》。对调用流程和数据流程进行分析时，这部分内容起到辅助作用。

1. ALSA-lib调试方法

alsa-utils中的aplay程序可以进行应用程序和lib库的调试。

操作如下：

(1) 复制alsa-utils和alsa-lib到linux文件系统下，如：

#cp -rf alsa-utils-1.0.16 /opt/

#cp -rf alsa-lib-1.0.16 /opt/

※红色文字表示shell中输入的命令，具体命令要根据具体环境自己修改。

alsa-utils-1.0.16和alsa-lib-1.0.16从网上下，也可以在光盘debian-506-source-DVD-1.iso中的pool/main/a/文件夹下找到。

(2). 复制音频文件,如复制test_files文件夹到/opt下,

#cp -rf test_files /opt/

(3). 在alsa-lib-1.0.16中依次执行如下：

#cd /opt/alsa-lib-1.0.16

#./configure

#make

#make install

※系统需要安装gcc等工具。

(4). 在alsa-utils-1.0.16 中中依次执行如下：

#cd /opt/alsa-utils-1.0.16

#./configure

#cd aplay

#make

就会生成执行文件aplay。

(5). 执行aplay文件播放.wav声音文件。如：

#./aplay /opt/test_files/pcm.wav

注意命令前面的" ./"如果不加，而系统中又安装了alsa-utils工具，就会执行/usr/bin/aplay，注意不要混了。

(6). 通过gdb可以对alsa-utils的aplay和alsa-lib的libsound.so进行本地调试。

#gdb aplay

(gdb)set args /opt/test_files/pcm.wav

(gdb)b main

(gdb)r

… … … …

三、准备工作

2. ALSA-lib调试方法示例。

```
db55:/opt/alsa-utils-1.0.16/aplay# gdb aplay
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
(gdb) set args /opt/test_files/pcm.wav
(gdb) b main
Breakpoint 1 at 0x80502a4: file aplay.c, line 346.
(gdb) r
Starting program: /opt/alsa-utils-1.0.16/aplay/aplay /opt/test_files/pcm.wav
[Thread debugging using libthread_db enabled]
[New Thread 0xb74dd6b0 (LWP 2831)]
[Switching to Thread 0xb74dd6b0 (LWP 2831)]

Breakpoint 1, main (argc=2, argv=0xbfffd284) at aplay.c:346
346     {
(gdb) b pcm_empty.c : _snd_pcm_empty_open
Breakpoint 2 at 0xb76f8995: file pcm_empty.c, line 82.
(gdb) c
Continuing.
xkm.log--> [aplay.c][390][main()]
xkm.log--> [pcm.c][2072][snd_pcm_open_conf(open_name=_snd_pcm_empty_open)]

Breakpoint 2, _snd_pcm_empty_open (pcmp=0x8053630, name=0x805181c "default",
    root=0x8054890, conf=0x8055100, stream=SND_PCM_STREAM_PLAYBACK, mode=0)
    at pcm_empty.c:82
82              snd_config_for_each(i, next, conf) {
(gdb) bt
#0  _snd_pcm_empty_open (pcmp=0x8053630, name=0x805181c "default",
    root=0x8054890, conf=0x8055100, stream=SND_PCM_STREAM_PLAYBACK, mode=0)
    at pcm_empty.c:82
#1  0xb76d4043 in snd_pcm_open_conf (pcmp=0x8053630, name=0x805181c "default",
    pcm_root=0x8054890, pcm_conf=0x8055100, stream=SND_PCM_STREAM_PLAYBACK,
    mode=0) at pcm.c:2116
#2  0xb76d4815 in snd_pcm_open_noupdate (pcmp=0x8053630, root=0x8054890,
    name=0x805181c "default", stream=SND_PCM_STREAM_PLAYBACK, mode=0, hop=0)
    at pcm.c:2158
#3  0x08050ac7 in main (argc=2, argv=0xbfffd284) at aplay.c:563
(gdb) _
```

图中红色表示为在shell中输入的命令，具体命令要根据具体环境自己修改。

跟踪代码时可以在gdb中使用bt(backtrace)指令跟踪调用栈查看函数调用关系。

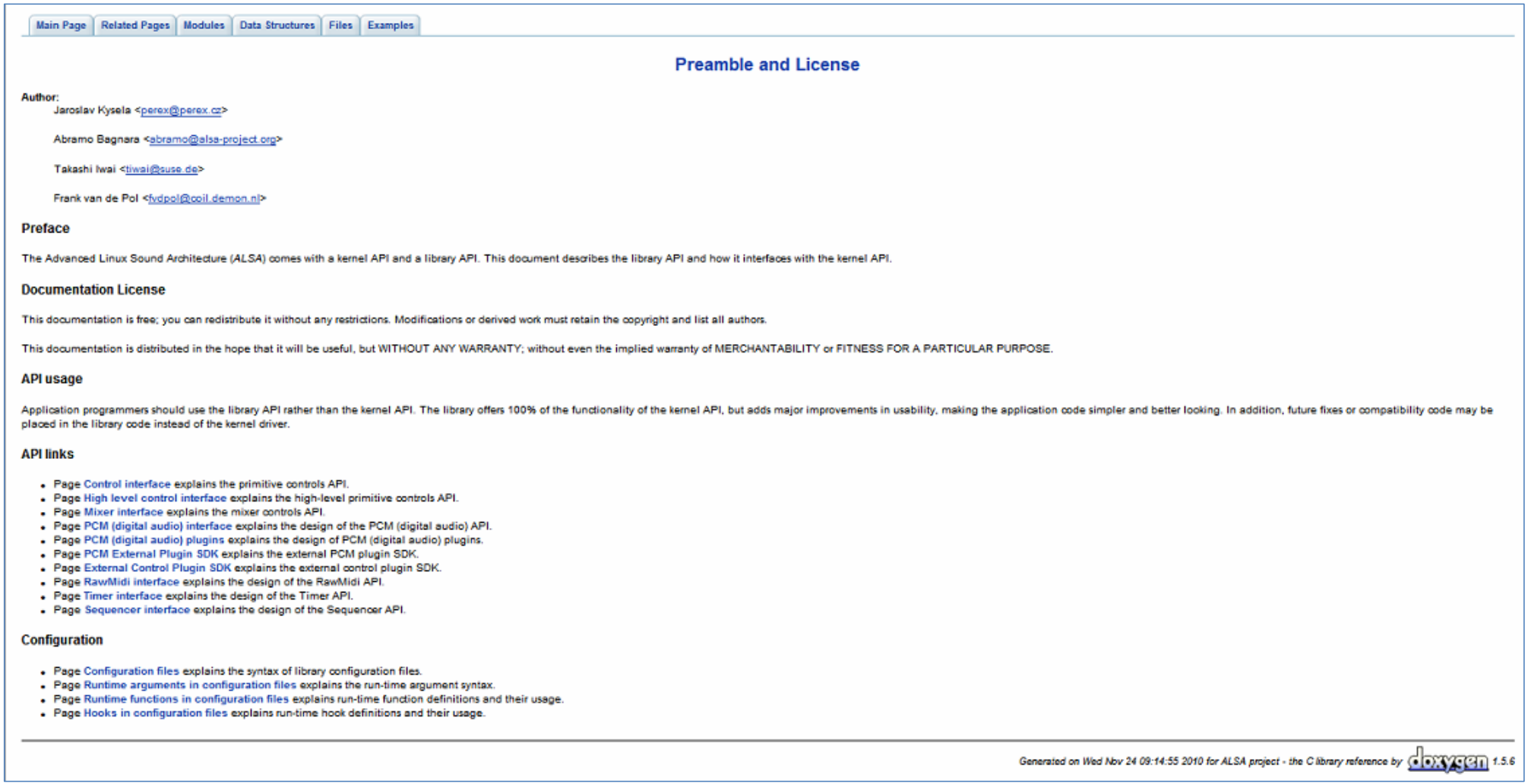
三、准备工作

3. 生成alsa-lib文档

alsa-lib中可以用文档生成工具doxygen生成API及相关说明文档。在alsa-lib-1.0.16中依次执行如下：

```
#cd   /opt/alsa-lib-1.0.16
#cd  doc
#doxygen  doxygen.cfg
```

在新生成的文件夹 doxygen/html中就会有网页形式的文档，首页为*index.html*



Main Page | Related Pages | Modules | Data Structures | Files | Examples

Preamble and License

Author:

Jaroslav Kysela <perex@perex.cz>

Abramo Bagnara <abramo@alsa-project.org>

Takashi Iwai <tiwai@suse.de>

Frank van de Pol <fvdpol@coil.demon.nl>

Preface

The Advanced Linux Sound Architecture (*ALSA*) comes with a kernel API and a library API. This document describes the library API and how it interfaces with the kernel API.

Documentation License

This documentation is free; you can redistribute it without any restrictions. Modifications or derived work must retain the copyright and list all authors.

This documentation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

API usage

Application programmers should use the library API rather than the kernel API. The library offers 100% of the functionality of the kernel API, but adds major improvements in usability, making the application code simpler and better looking. In addition, future fixes or compatibility code may be placed in the library code instead of the kernel driver.

API links

- Page Control interface explains the primitive controls API.
- Page High level control interface explains the high-level primitive controls API.
- Page Mixer interface explains the mixer controls API.
- Page PCM (digital audio) interface explains the design of the PCM (digital audio) API.
- Page PCM (digital audio) plugins explains the design of PCM (digital audio) plugins.
- Page PCM External Plugin SDK explains the external PCM plugin SDK.
- Page External Control Plugin SDK explains the external control plugin SDK.
- Page RawMidi interface explains the design of the RawMidi API.
- Page Timer interface explains the design of the Timer API.
- Page Sequencer interface explains the design of the Sequencer API.

Configuration

- Page Configuration files explains the syntax of library configuration files.
- Page Runtime arguments in configuration files explains the run-time argument syntax.
- Page Runtime functions in configuration files explains run-time function definitions and their usage.
- Page Hooks in configuration files explains run-time hook definitions and their usage.

Generated on Wed Nov 24 09:14:55 2010 for ALSA project - the C library reference by doxygen *1.5.6*

三、准备工作

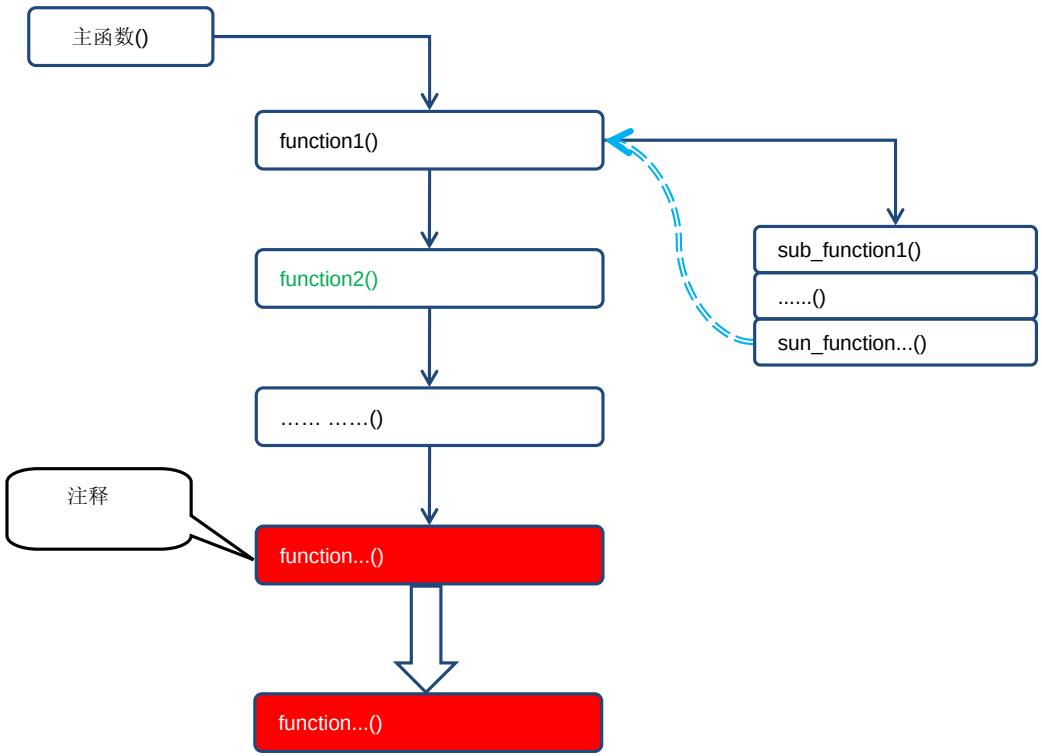
4. 流程图结构说明

(1) 图示为函数的调用关系，向下为同一级调用，向右为函数内部的子函数调用。

(2) 绿色文字函数名（如function2）表示该函数是调用流程中比较关键的点。

(3) 红底白字的函数名（如function…）表示和其它层（如app和lib、lib和driver、alsa-driver和device-driver）的接口函数或kernel的回调函数。

(4) 蓝色双虚线为函数实参等形式的输出值或函数返回值（如sun_function…有输出值到function1）。



目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程

i. 整体分析

ii. 设备驱动程序insmod流程图

iii. 应用程序主流程图

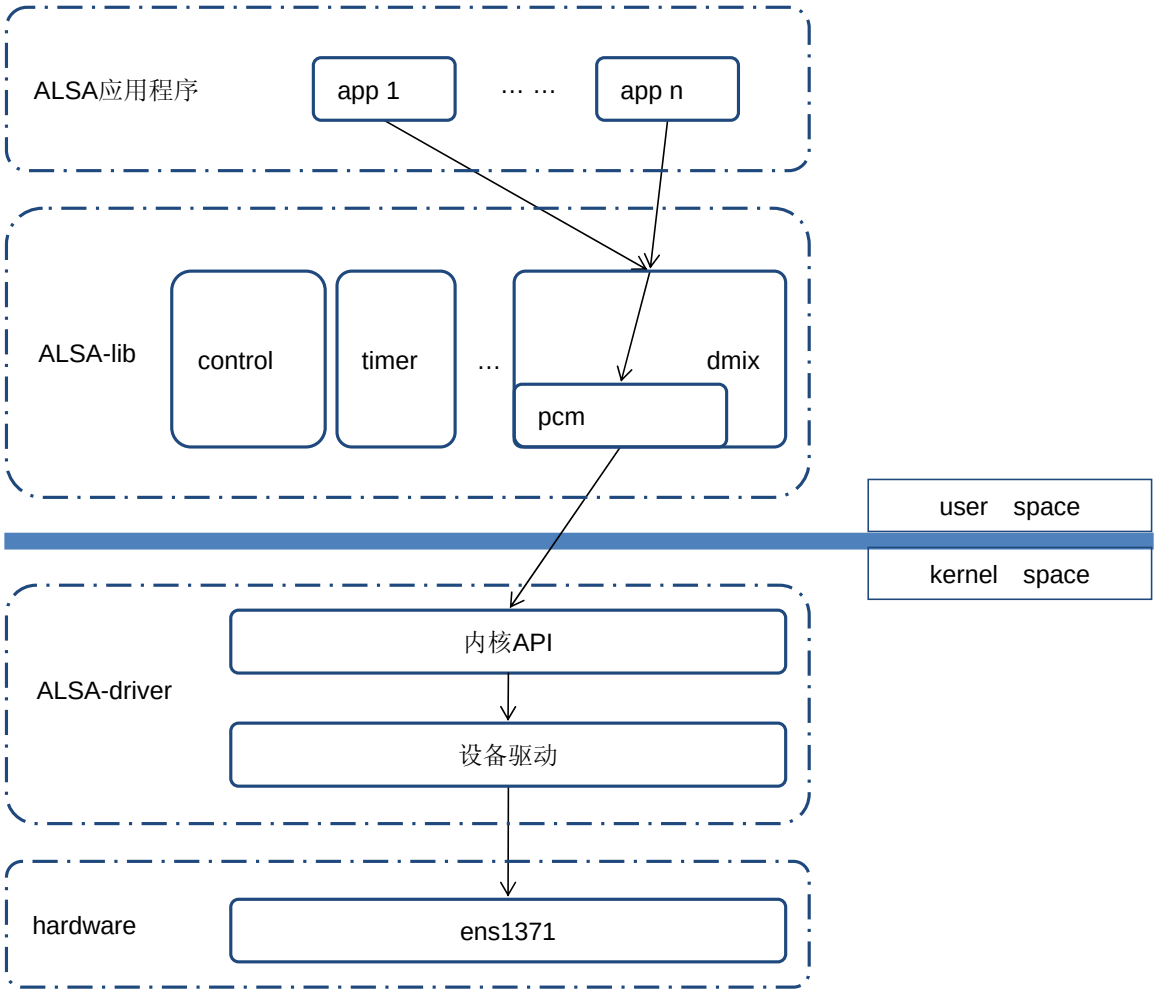
iv. 声卡打开流程图

v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

四、设备打开过程和数据流程

i. 整体分析



1. 声卡芯片的硬件驱动程序实际上是alsa-driver的一部分，但是在本文档中，为了明示层次关系，将这两个分开对待。将硬件驱动程序叫做device-driver(设备驱动)， alsa-driver的其它部分叫做alsa-driver(alsa驱动)。

2. 硬件驱动被insmod后，应用程序就可以调用ALSA-lib的API函数播放声音。左图是从应用程序开始，到我们的编写设备驱动程序的调用过程。从应用程序角度看，ASLA的操作分为两部分，打开及关闭是一部分，写入数据是另一部分。

3. 当声卡name使用”default”时，pcm作为dmix的从设备存在，应用程序直接和dmix的相关函数打交道。dmix混合数据后，直接写到mmap映射的地址中。

4. 关于1371的驱动ens1371-playback.ko，参见《 Linux ALSA声卡驱动开发最佳实践.pptx 》。从设备驱动程序角度看，声卡核心驱动可以分为三个主要部分：
   (1) 模块初始化和退出部分。
   (2) 播放以及停止部分。
   (3) 中断处理部分。
   而从应用程序的角度向下看,这三部分分别在insmod、打开声卡和写入数据时被关联。

目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程

i. 整体分析

ii. 设备驱动程序insmod流程图

iii. 应用程序主流程图

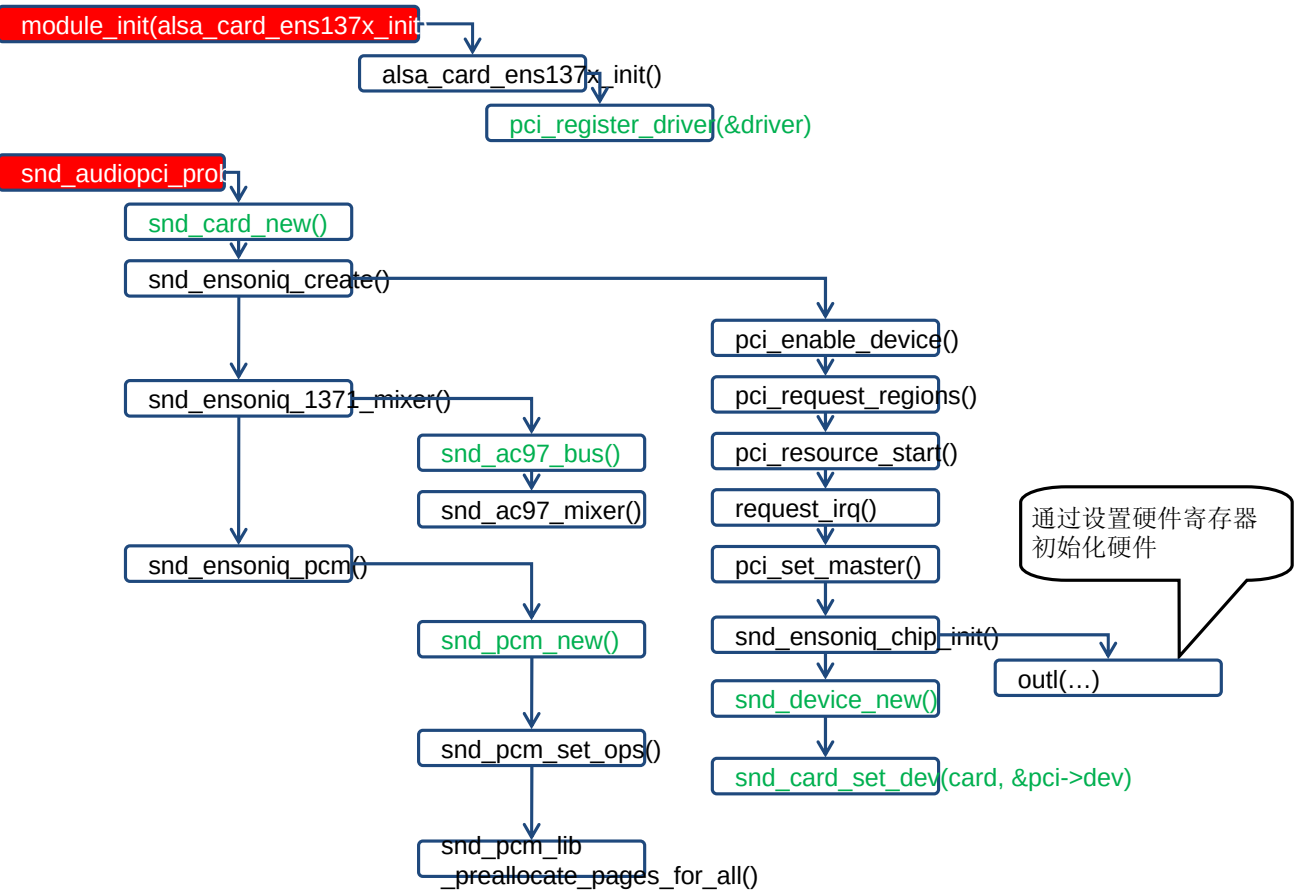
iv. 声卡打开流程图

v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

四、设备打开过程和数据流程

ii. 设备驱动程序insmod流程图



insmod ens1371-playback时，除了snd_ensoniq_chip_init()设置了硬件寄存器使硬件初始化外，其他的都是为了alsa-driver的架构而进行的工作，因为pci总线是层次式结构的，所以insmod设备是pci card device pcm(substream)一层层细化的，

使用inl()/outl()函数读/写内存映射方式的32位寄存器。关于snd_ensoniq_chip_init()时的硬件寄存器配置，参见《ES1371 datasheet.pdf》。例如要使能dac1，根据手册第16页内容，就知道应该设置Interrupt/Chip Select Control Register第6位（从0位开始）,该寄存器的偏移量为0，假设原寄存器内容为ctrl，pci映射的端口为port，则可以用如下的代码实现：

```
ctrl |= (1 << 6);
outl(ctrl, port + 0x00);
```

1. pci_register_driver(): 注册一个pci设备。
   linux-source-2.6.26/include/linux/pci.h : 656
2. snd_card_new(): create and initialize a soundcard structure.
   linux-source-2.6.26/sound/core/init.c : 128
3. 对pci设备进行设置，这些函数分别是：
   pci_enable_device(): Initialize device before it's used by a driver.
   linux-source-2.6.26/drivers/pci/pci.c : 787
   pci_request_regions(): Reserved PCI I/O and memory resources.
   linux-source-2.6.26/drivers/pci/pci.c : 1198
   pci_resource_start():取得设备pci六个内存区其中一个的地址。
   linux-source-2.6.26/include/linux/pci.h : 952
   request_irq(): 注册中断处理函数。
   pci_set_master(): enables bus-mastering for device dev.
   linux-source-2.6.26/drivers/pci/pci.c : 1211
4. snd_device_new(): create an ALSA device component.
   linux-source-2.6.26/sound/core/device.c : 43
5. snd_card_set_dev(): 将声卡和pci设备关联。
   linux-source-2.6.26/include/sound/core.h : 313
6. snd_pcm_new(): create a new PCM instance.
   linux-source-2.6.26/sound/core/pcm.c : 673

ens1371的音效控制、DA和AD转换控制等是符合AC97标准的，所以除了硬件相关部分，由alsa-driver完成。而硬件相关部分是通过注册snd_ac97_bus_ops实现的，这部分可以通过alsa-utils的alsamixer工具调试，如改变音量大小时，alsa-driver会通过我们注册的写函数写相关的硬件寄存器。

目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程

i. 整体分析

ii. 设备驱动程序insmod流程图

iii. 应用程序主流程图

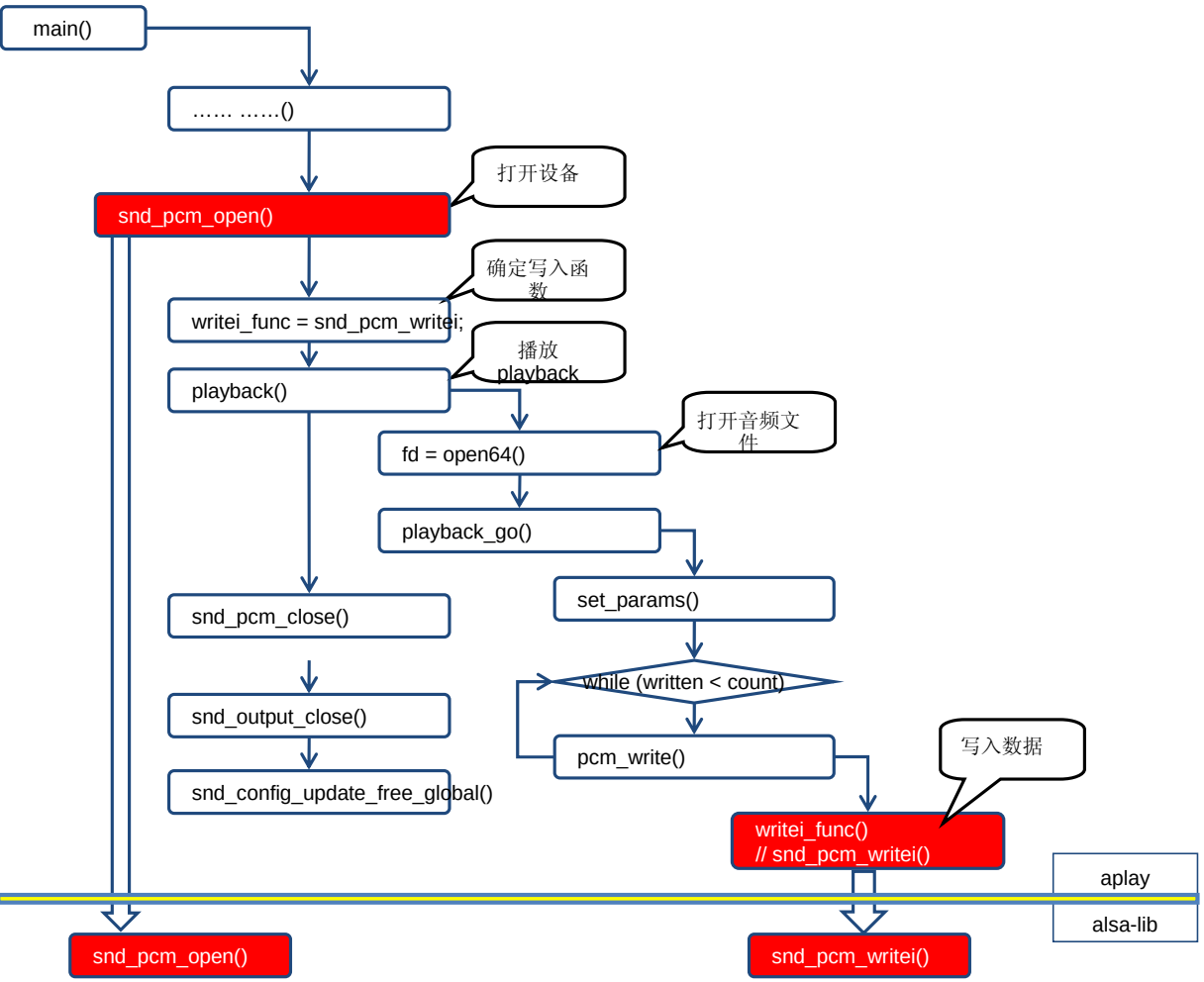
iv. 声卡打开流程图

v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

四、设备打开过程和数据流程

iii. 应用程序主流程图

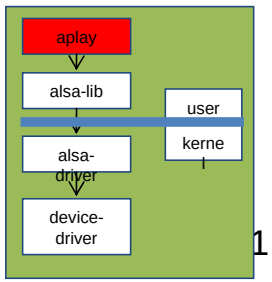


aplay是一个简单的.wav格式播放工具，工作在用户层，通过调用alsa-lib实现音频播放功能。aplay的主要流程为：

1. 调用snd_pcm_open()打开alsa的设备，具体打开对象根据alsa设置有所不同，默认的default递归打开dmix，pcm等设备。
2. 打开文件（open64()）后，循环将数据写入alsa。

主要函数位置：

1. main()
   alsa-utils-1.0.16/aplay/aplay.c : 345
2. snd_pcm_open()
   alsa-lib-1.0.16/src/pcm/pcm.c : 2166
3. snd_pcm_writei()
   alsa-lib-1.0.16/src/pcm/pcm.c : 1175
4. playback(), playback_go(), set_params(), pcm_write()
   alsa-utils-1.0.16/aplay/aplay.c



目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程

i. 整体分析

ii. 设备驱动程序insmod流程图

iii. 应用程序主流程图

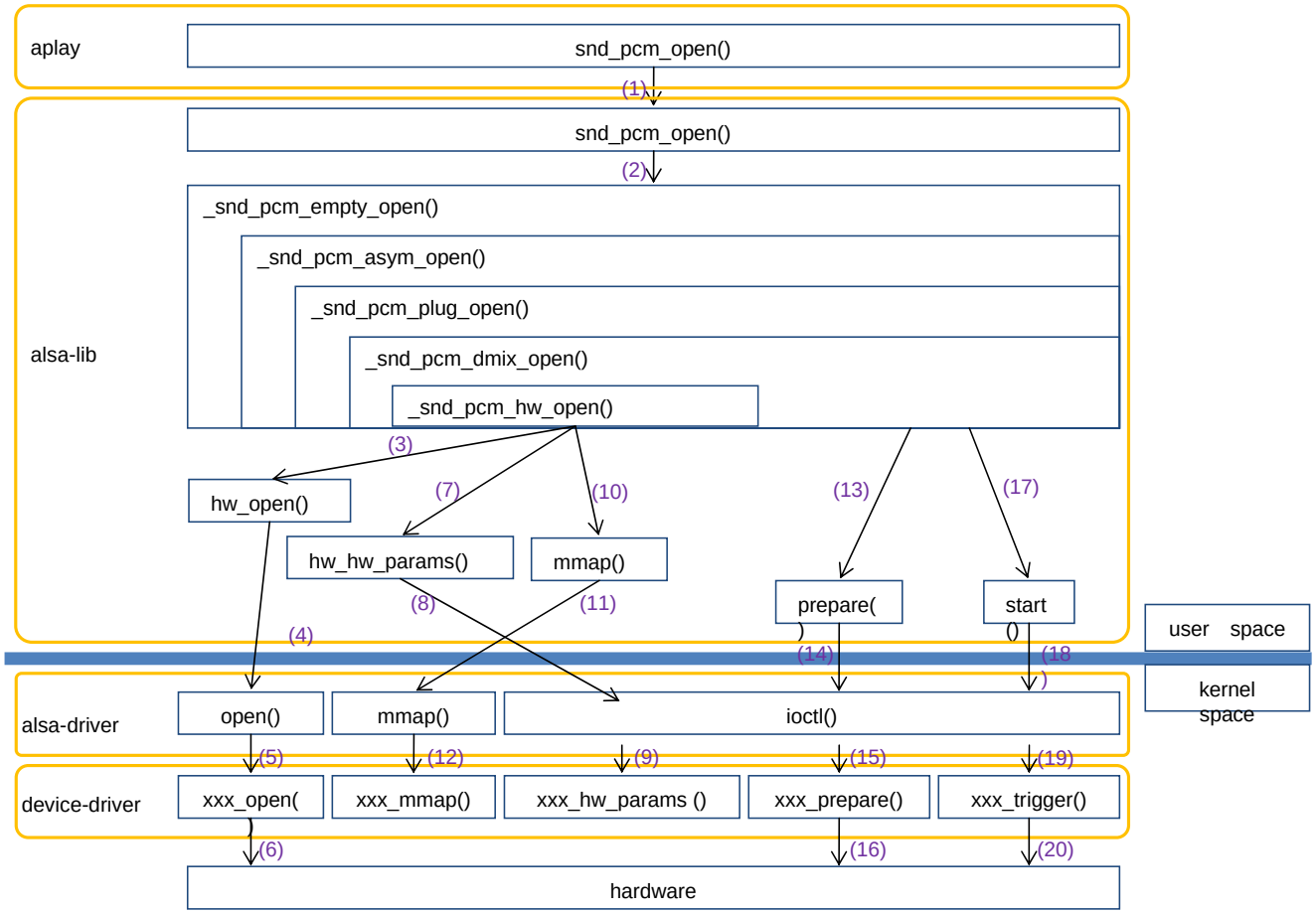
iv. 声卡打开流程图

v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

四、设备打开过程和数据流程

iv. 声卡打开流程图

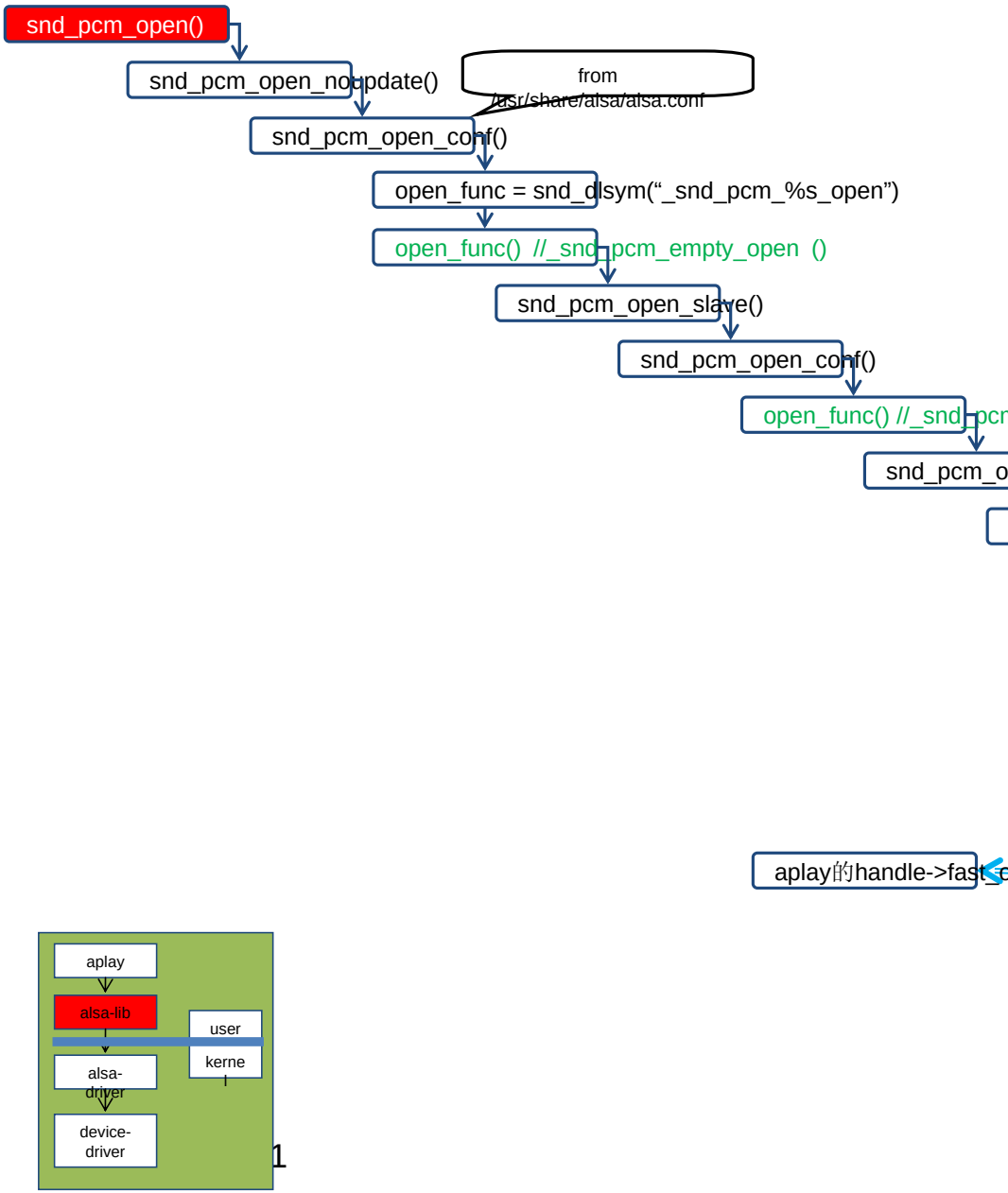


图中是应用程序通过alsa-lib打开声卡的流程。可以通过序号分析。
(1) aplay调用alsa-lib的API函数打开声卡。
(2) alsa-lib通过读取配置文件的信息层层调用依次打开 empty asym plug dmix hw，其中后一个都是前一个的从设备。
(3)(4)(5)(6) hw打开函数调用snd_pcm_hw_open()函数通过alsa-driver层和设备驱动open硬件设备。
(7)(8)(9)打开hw后，设置硬件相关参数。
(10)(11)(12)调用snd_pcm_mmap()函数通过alsa-driver层建立内存映射。
(13)(14)(15)(16) dmix调用snd_pcm_prepare ()函数通过alsa-driver层和设备驱动向硬件中写入参数。
(17)(18)(19)(20) 参数设定后，dmix调用snd_pcm_start ()函数通过alsa-driver层和设备驱动向启动硬件进入工作状态。

详细流程见后续。

四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-lib API snd_pcm_open()流程图



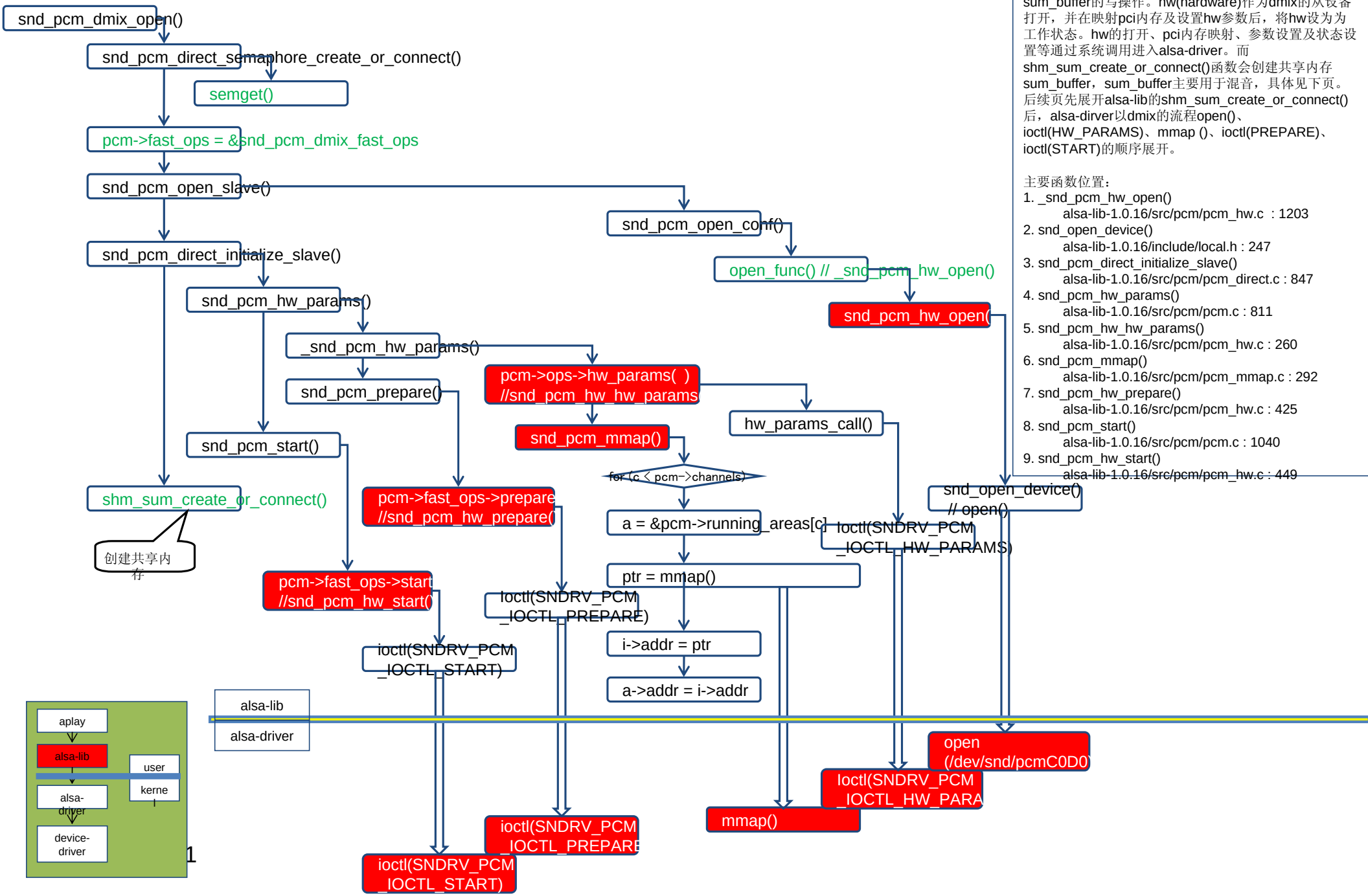
上图对应的是”声卡打开流程图”的(2)
进入alsa-lib后，snd_pcm_open_noupdate()通过层级调用打开配置文件中的设置，以empty asym plug dmix的顺序打开，最后将dmix的fast_ops返回给aplay的handle->fast_ops。dmix的这些方法将会在后面写入数据时以pcm->fast_ops->xxx()的方式调用。

主要函数位置：

1. snd_pcm_open()
   alsa-lib-1.0.16/src/pcm/pcm.c : 2166
2. _snd_pcm_empty_open()
   alsa-lib-1.0.16/src/pcm/pcm_empty.c : 74
3. _snd_pcm_asym_open()
   alsa-lib-1.0.16/src/pcm/pcm_asym.c : 77
4. _snd_pcm_plug_open()
   alsa-lib-1.0.16/src/pcm/pcm_plug.c : 1125
5. _snd_pcm_dmix_open()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 1246
4. snd_pcm_plug_open()
   alsa-lib-1.0.16/src/pcm/pcm_plug.c : 1015
5. snd_pcm_dmix_open()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 921

四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-lib snd_pcm_dmix_open()流程图



上图对应的是"声卡打开流程图"的(3)(4)、(7)(8)、(10)(11)、(13) (14)、(17) (18)

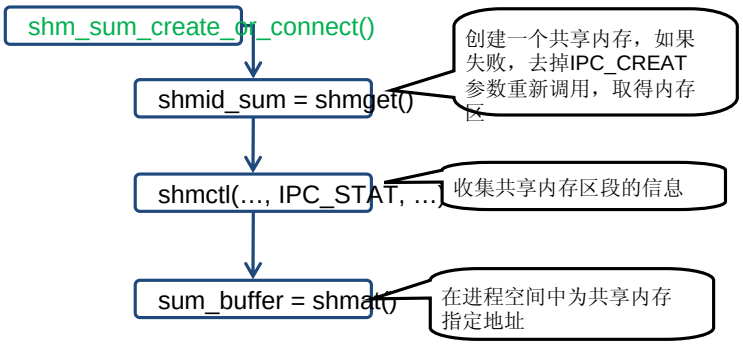
打开dmix时，会创建或获取一个信号量，这个信号量用于多个进程共同访问dmix临界资源时的互斥，如对sum_buffer的写操作。hw(hardware)作为dmix的从设备打开，并在映射pci内存及设置hw参数后，将hw设为为工作状态。hw的打开、pci内存映射、参数设置及状态设置等通过系统调用进入alsa-driver。而shm_sum_create_or_connect()函数会创建共享内存sum_buffer，sum_buffer主要用于混音，具体见下页。后续页先展开alsa-lib的shm_sum_create_or_connect()后，alsa-dirver以dmix的流程open()、ioctl(HW_PARAMS)、mmap ()、ioctl(PREPARE)、ioctl(START)的顺序展开。

主要函数位置：

1. _snd_pcm_hw_open()
   alsa-lib-1.0.16/src/pcm/pcm_hw.c : 1203
2. snd_open_device()
   alsa-lib-1.0.16/include/local.h : 247
3. snd_pcm_direct_initialize_slave()
   alsa-lib-1.0.16/src/pcm/pcm_direct.c : 847
4. snd_pcm_hw_params()
   alsa-lib-1.0.16/src/pcm/pcm.c : 811
5. snd_pcm_hw_hw_params()
   alsa-lib-1.0.16/src/pcm/pcm_hw.c : 260
6. snd_pcm_mmap()
   alsa-lib-1.0.16/src/pcm/pcm_mmap.c : 292
7. snd_pcm_hw_prepare()
   alsa-lib-1.0.16/src/pcm/pcm_hw.c : 425
8. snd_pcm_start()
   alsa-lib-1.0.16/src/pcm/pcm.c : 1040
9. snd_pcm_hw_start()
   alsa-lib-1.0.16/src/pcm/pcm_hw.c : 449

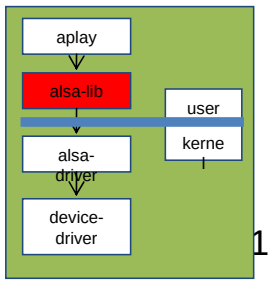
四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-lib shm_sum_create_or_connect()流程图



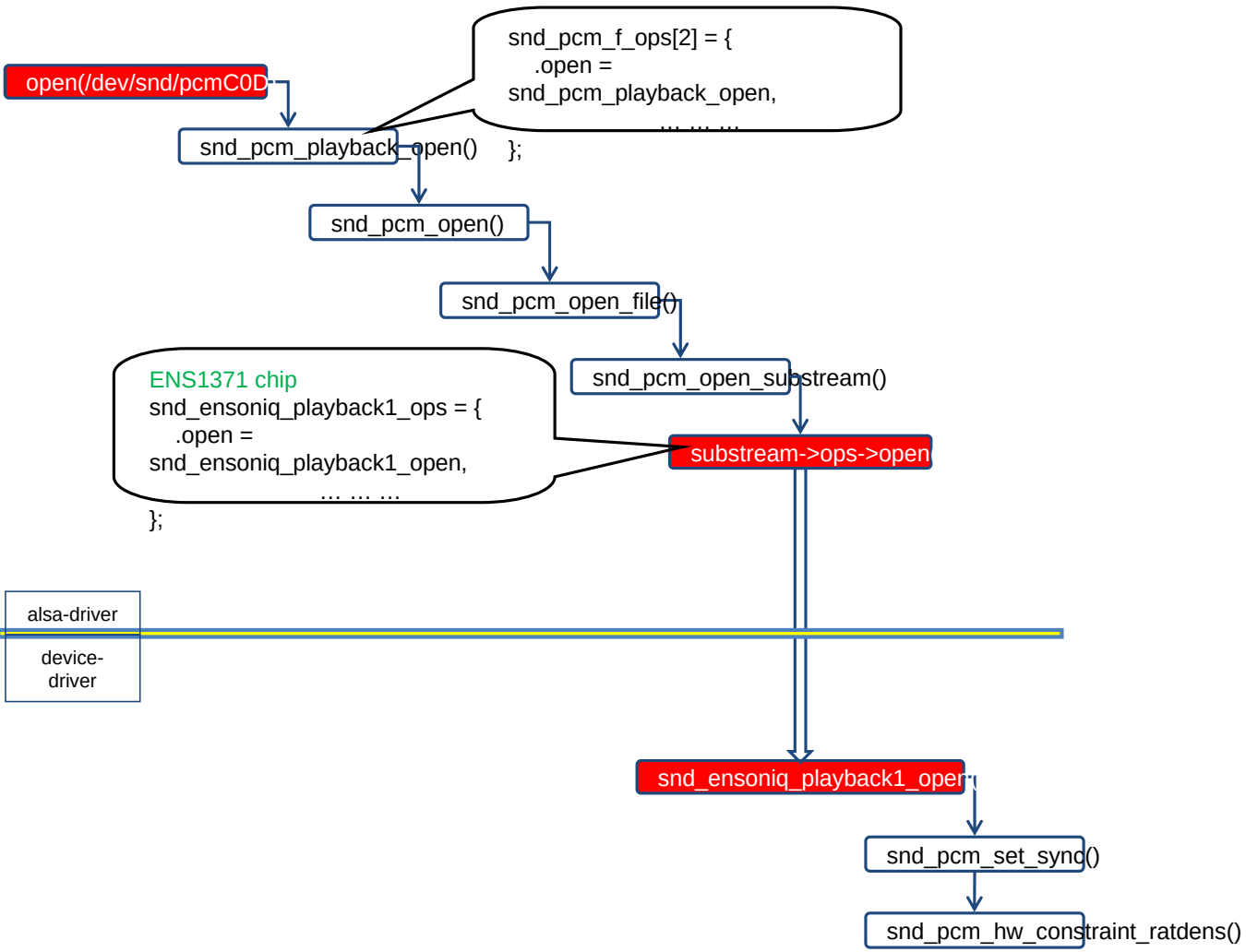
shm_sum_create_or_connect()函数创建共享内存sum_buffer， sum_buffer始和其他进程共享的，这样就可以取得其他进程的数据，并把自己进程的数据和其他进程叠加，完成混音功能。

关于共享内存：共享内存允许两个或多个进程共享同一块内存（这块内存会映射到各自进程独立的地址空间中）从而使这些进程可以相互通信。共享内存块提供了在任意数量的进程之间进行高效双向通信的机制。每个使用者都可以读取写入数据，但是所有程序之间必须达成并遵守一定的协议，以防止诸如在读取信息之前覆写内存空间等竞争状态的出现。
shmget()：分配一个共享内存块。
shmctl()：设置或返回一个共享内存块的相关信息。
shmat()：绑定共享内存到进程地址空间。



四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-driver open()流程图



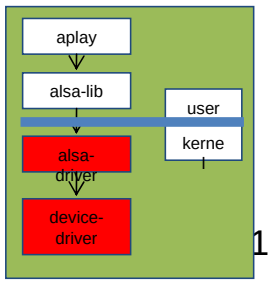
左图上半部alsa-driver对应的是"声卡打开流程图"的(5)

在alsa-driver中open函数对应的snd_pcm_playback_open()逐层调用，打开PCM，设备文件，子流等，在open_substream中通过数指针调用到硬件驱动函数snd_ensoniq_playback1_open()，也就是我们写具体声卡驱动需要注册的open函数。这些函数都已经被添加进内核，调试方法参见《Linux 基础培训(2)-驱动开发最佳实践-1.pptx》。

主要函数位置：
1. snd_pcm_playback_open()
   linux-source-2.6.26/sound/core/pcm_native.c : 2092
2. snd_pcm_open()
   linux-source-2.6.26/sound/core/pcm_native.c : 2110
3. snd_pcm_open_file()
   linux-source-2.6.26/sound/core/pcm_native.c : 2059
4. snd_pcm_open_substream()
   linux-source-2.6.26/sound/core/pcm_native.c : 2017
5. snd_ensoniq_playback1_open()
   linux-source-2.6.26/sound/pci/ens1370.c : 1110

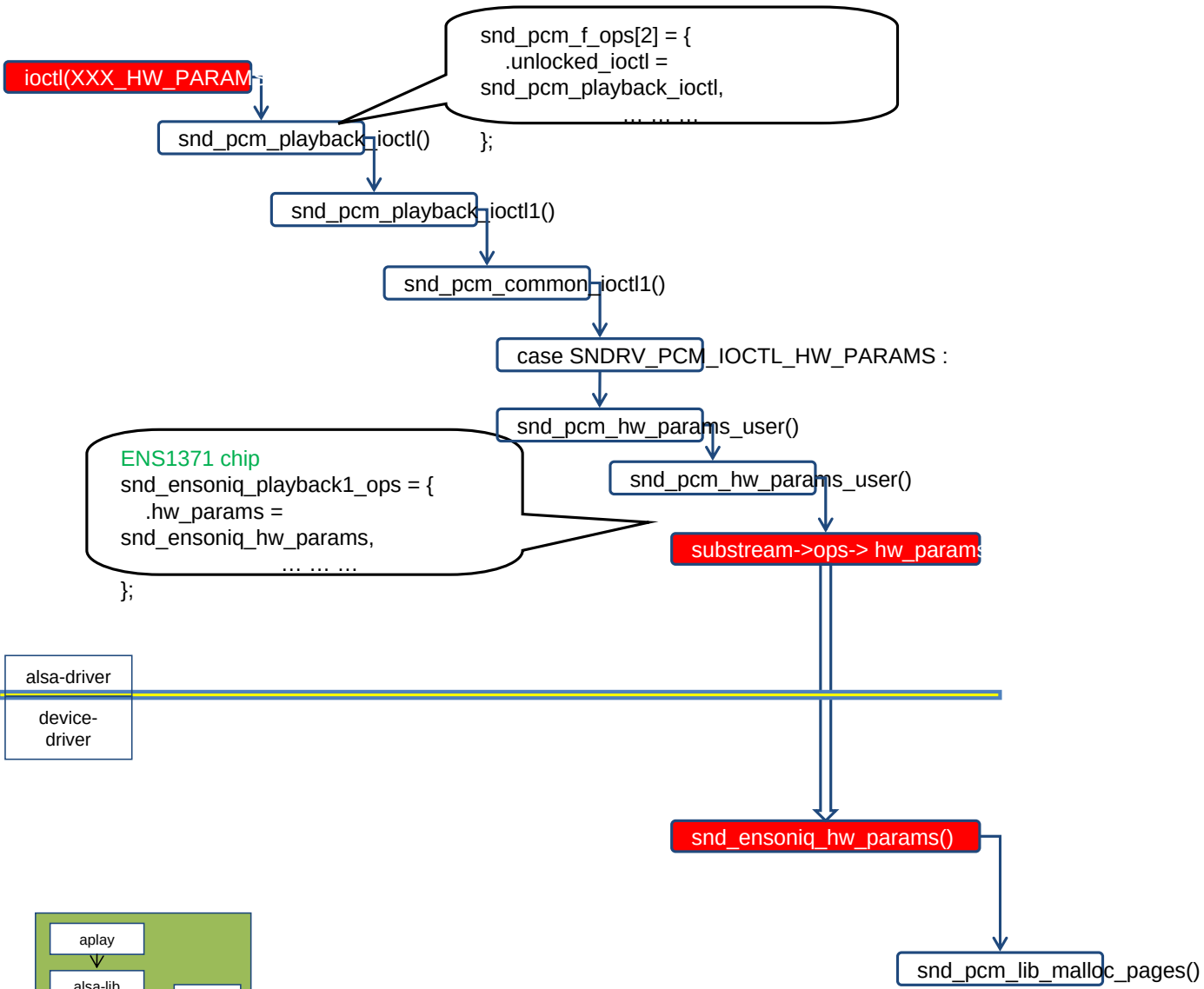
左图下半部alsa-driver对应的是"声卡打开流程图"的(6)

主要函数位置：
1. snd_pcm_set_sync() 设置硬件同步ID
   linux-source-2.6.26/sound/core/pcm_lib.c : 297
2. snd_pcm_hw_constraint_ratdens() 设置硬件性能约束
   linux-source-2.6.26/sound/core/pcm_lib.c : 1068



四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-driver ioctl(HW_PARAMS)流程图



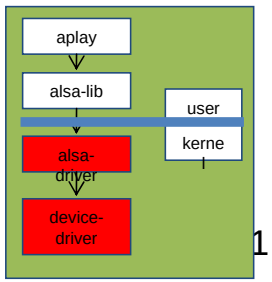
左图上半部alsa-driver对应的是"声卡打开流程图"的(9)

在alsa-driver中ioctl函数对应的snd_pcm_playback_ioctl ()逐层调用，在snd_pcm_common_ioctl1()函数中通过switch-case定位hw_param函数snd_pcm_hw_params_user()，然后逐层向下调用，最后通过函数指针调用到硬件驱动函数snd_ensoniq_hw_params()，也就是我们写具体声卡驱动需要注册的prepare函数。

主要函数位置：
1. snd_pcm_playback_ioctl()
() linux-source-2.6.26/sound/core/pcm_native.c : 2721
2. snd_pcm_common_ioctl1()
( ) linux-source-2.6.26/sound/core/pcm_native.c : 2491
3. snd_pcm_hw_params_user()
linux-source-2.6.26/sound/core/pcm_native.c : 462
4. snd_pcm_hw_params()
linux-source-2.6.26/sound/core/pcm_native.c : 364

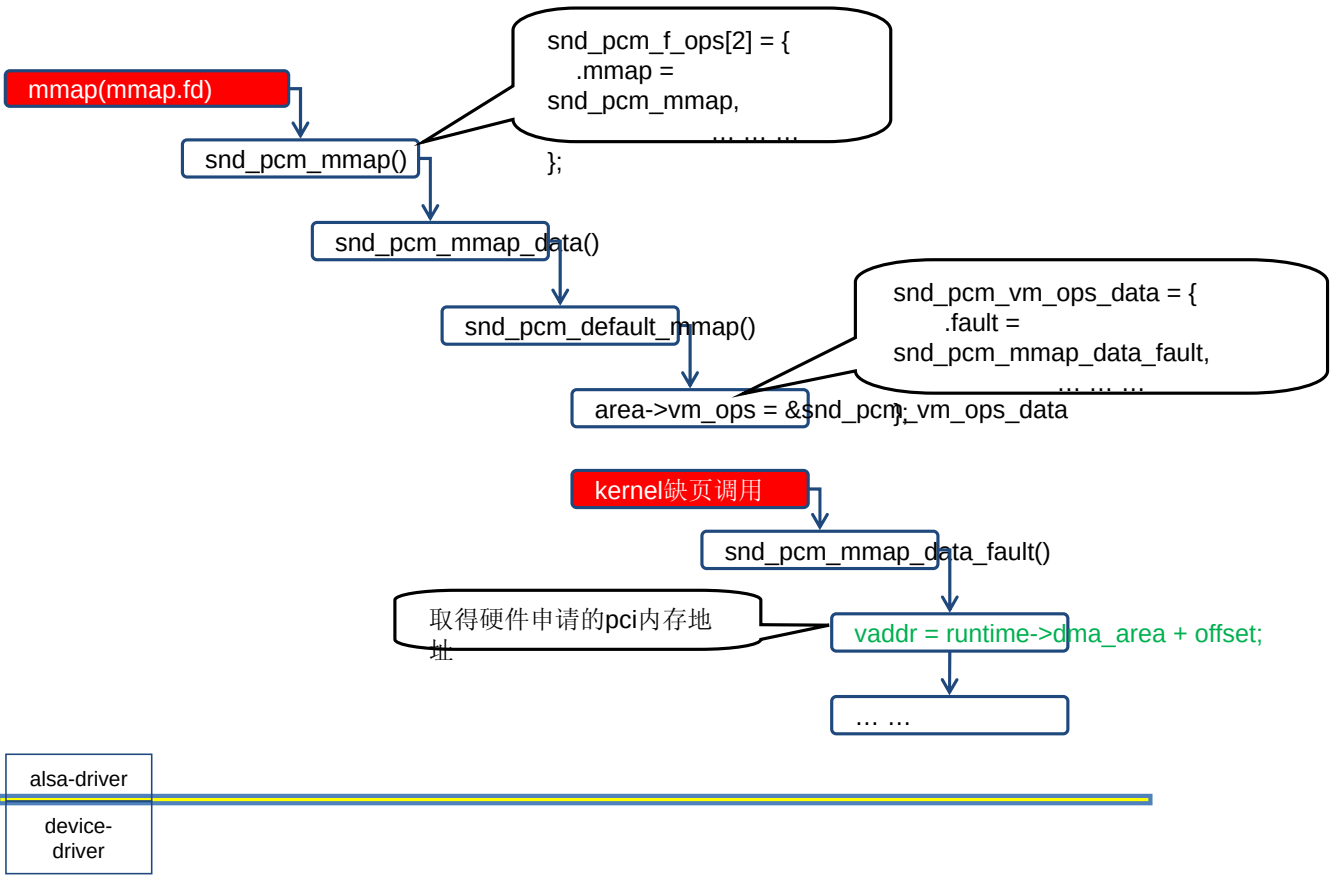
左图下半部alsa-driver对应的是"声卡打开流程图"的(9)

主要函数位置：
1. snd_pcm_lib_malloc_pages() allocate the DMA buffer
linux-source-2.6.26/sound/core/pcm_memory.c : 340



四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-driver mmap()流程图

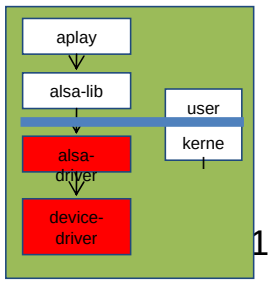


左图alsa-driver对应的是"声卡打开流程图"的(12)

在alsa-driver中mmap函数对应的snd_pcm_mmap ()将dmix的mmap.fd对应的缺页方法注册进内核。通过缺页中断，返回内核虚拟地址runtime->dma_area所在页的描述符指针，完成内存映射。

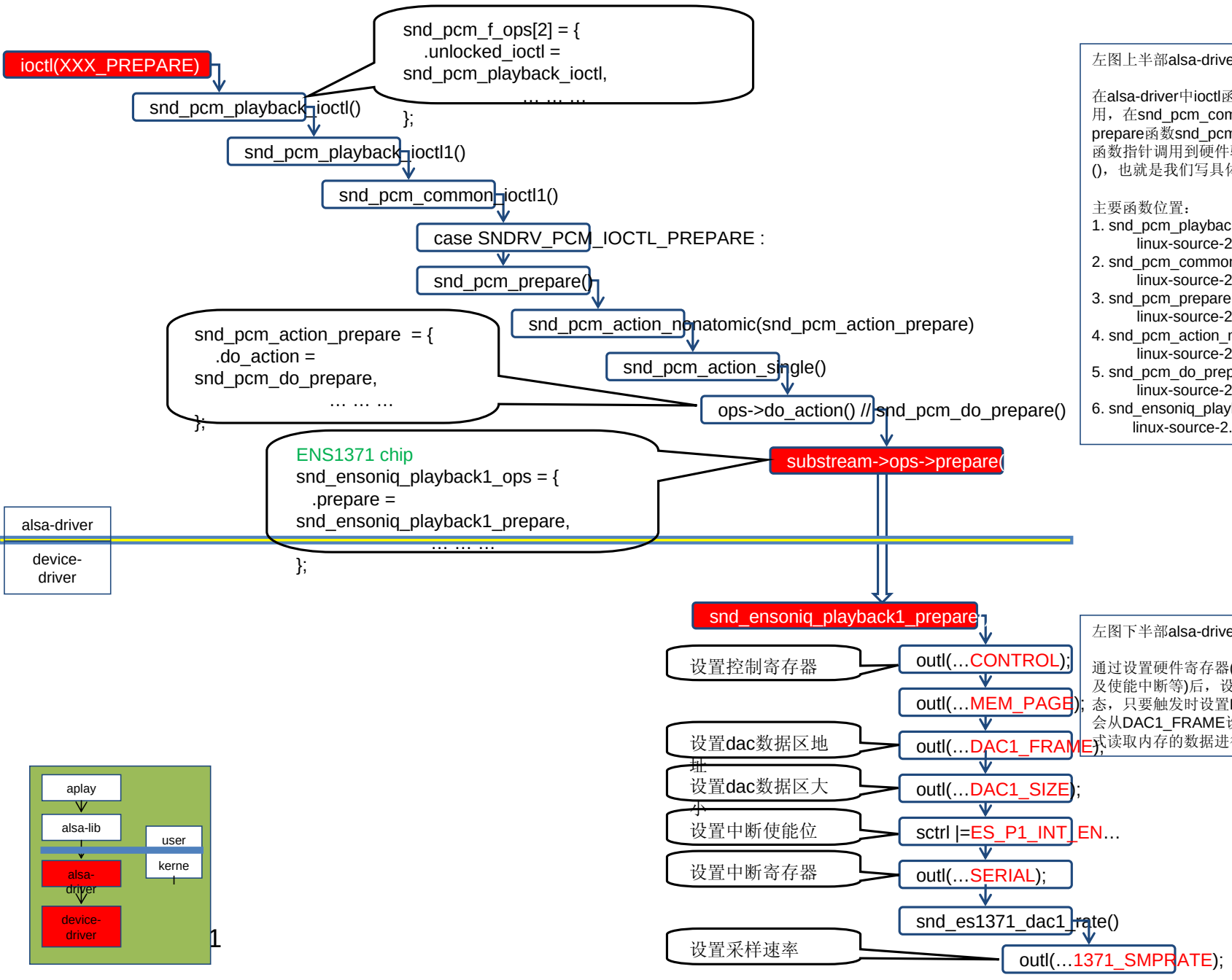
主要函数位置：
1. snd_pcm_mmap()
   linux-source-2.6.26/sound/core/pcm_native.c : 3214
2. snd_pcm_mmap_data()
   linux-source-2.6.26/sound/core/pcm_native.c : 3174
3. snd_pcm_default_mmap()
   linux-source-2.6.26/sound/core/pcm_native.c : 3126
4. snd_pcm_mmap_data_fault()
   linux-source-2.6.26/sound/core/pcm_native.c : 3086

关于mmap：mmap也就是内存映射，对于驱动程序来说，内存映射可以提供给用户直接访问设备内存的能力。映射一个设备意味着将用户空间的一段内存与设备内存关联起来。无论何时当程序分配的地址范围内读写时，实际上访问的就是设备。



四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-driver ioctl(PREPARE)流程图



左图上半部alsa-driver对应的是”声卡打开流程图”的(15)

在alsa-driver中ioctl函数对应的snd_pcm_playback_ioctl ()逐层调用，在snd_pcm_common_ioctl1()函数中通过switch-case定位prepare函数snd_pcm_prepare()，然后逐层向下调用，最后通过函数指针调用到硬件驱动函数snd_ensoniq_playback1_prepare ()，也就是我们写具体声卡驱动需要注册的prepare函数。

主要函数位置：

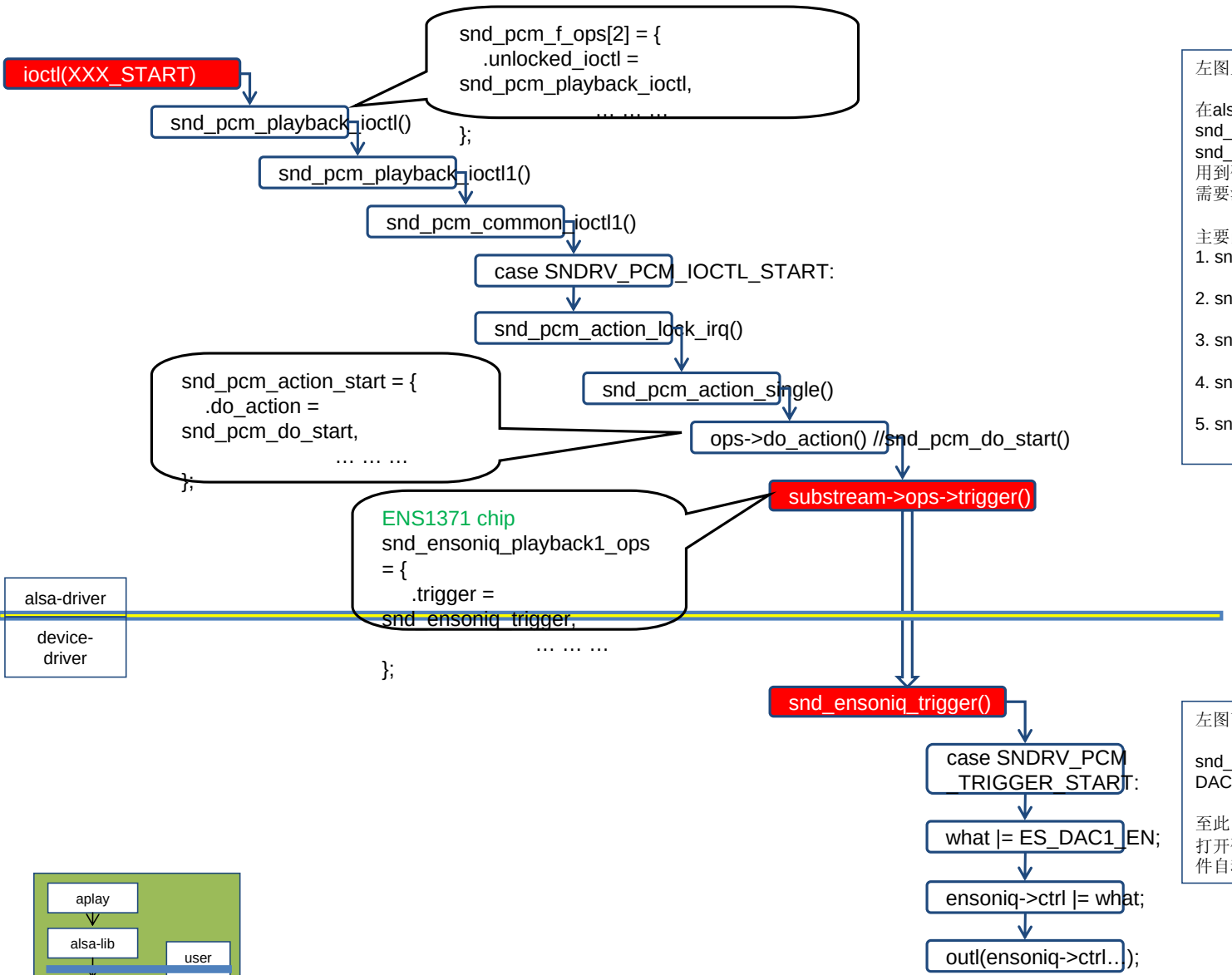
1. snd_pcm_playback_ioctl()
   linux-source-2.6.26/sound/core/pcm_native.c : 2721
2. snd_pcm_common_ioctl1()
   linux-source-2.6.26/sound/core/pcm_native.c : 2491
3. snd_pcm_prepare()
   linux-source-2.6.26/sound/core/pcm_native.c : 1327
4. snd_pcm_action_nonatomic()
   linux-source-2.6.26/sound/core/pcm_native.c : 818
5. snd_pcm_do_prepare()
   linux-source-2.6.26/sound/core/pcm_native.c : 1298
6. snd_ensoniq_playback1_prepare()
   linux-source-2.6.26/sound/pci/ens1370.c : 867

左图下半部alsa-driver对应的是”声卡打开流程图”的(16)

通过设置硬件寄存器(设置控制寄存器及设置内存地址和大小、以及使能中断等)后，设置dac的速率，使ens1371处于工作等待状态，只要触发时设置ES_DAC1_EN位，硬件就可以工作了。硬件会从DAC1_FRAME设置的地址和DAC1_SIZE，以循环缓冲的方式读取内存的数据进行DA转换，并播放声音。

四、设备打开过程和数据流程

iv. 声卡打开流程图 --- alsa-driver ioctl(START)流程图



左图上半部alsa-driver对应的是"声卡打开流程图"的(19)

在alsa-driver中ioctl函数对应的snd_pcm_playback_ioctl ()逐层调用，在snd_pcm_common_ioctl1()函数中通过switch-case定位start函数snd_pcm_action_lock_irq ()，然后逐层向下调用，最后通过函数指针调用到硬件驱动函数snd_ensoniq_trigger()，也就是我们写具体声卡驱动需要注册的trigger函数。

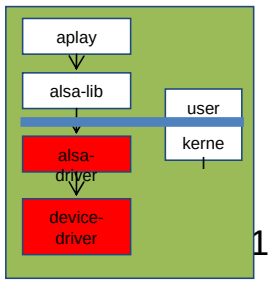
主要函数位置：

1. snd_pcm_playback_ioctl()
   linux-source-2.6.26/sound/core/pcm_native.c : 2721
2. snd_pcm_common_ioctl1()
   linux-source-2.6.26/sound/core/pcm_native.c : 2491
3. snd_pcm_action_lock_irq()
   linux-source-2.6.26/sound/core/pcm_native.c : 794
4. snd_pcm_do_start()
   linux-source-2.6.26/sound/core/pcm_native.c : 848
5. snd_ensoniq_trigger()
   linux-source-2.6.26/sound/pci/ens1370.c : 792

左图下半部alsa-driver对应的是"声卡打开流程图"的(20)

snd_ensoniq_trigger()通过向硬件控制寄存器设置ES_DAC1_EN位，使DAC1使能，进入工作状态。

至此，从用户层应用程序通过alsa-lib到alsa-driver再到硬件驱动程序，打开硬件并开始工作的过程已经完成。下面就要向硬件中发送数据，硬件自动将数据进行DA转换并播放出声音。



目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程

i. 整体分析

ii. 设备驱动程序insmod流程图

iii. 应用程序主流程图

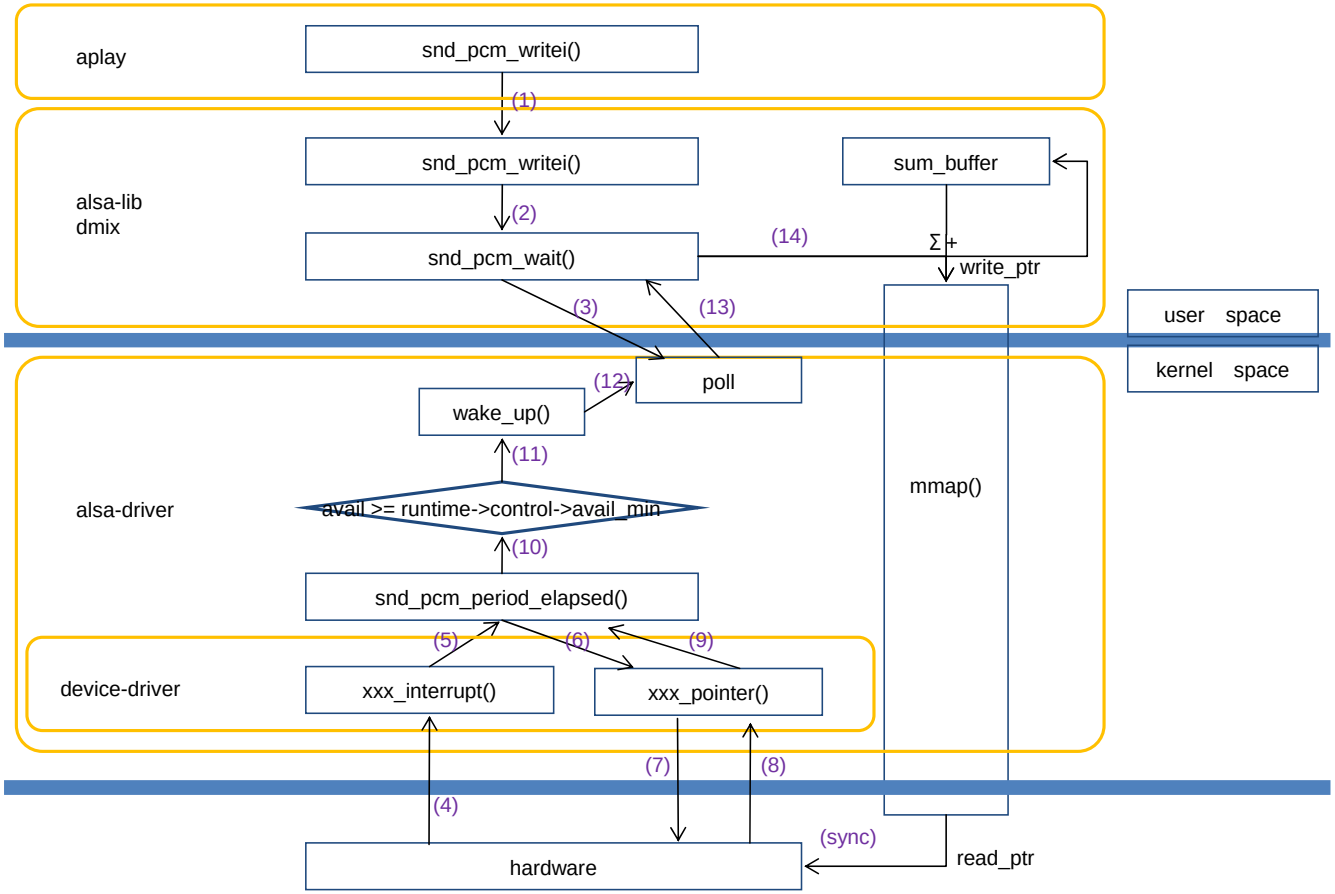
iv. 声卡打开流程图

v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

四、设备打开过程和数据流程

a. 数据写入流程图



图中是应用程序通过alsa-lib写入数据的流程。可以通过序号分析。

(1) aplay调用alsa-lib的API函数写入数据。

(2) alsa-lib调用等待函数等待底层可写。

(3) alsa-lib中是通过poll系统调用进入驱动层，将poll信号加入到sleep队列，阻塞进程。

(4) 硬件发送定时间隔中断触发硬件驱动函数中注册的中断函数。

(5) 中断函数调用alsa-driver中的函数判断是否应该写。

(6) alsa-driver中的函数调用硬件驱动函数获取硬件当前数据大小。

(7) (8) 硬件驱动函数向硬件寄存器读取当前数据大小。

(9) 将从硬件寄存器读取到的size值返回给alsa-driver。

(10) alsa-driver判断空闲数据区大小，

(11) 如果大于设定的最小值，就唤醒sleep队列。

(12) poll信号被唤醒。

(13) 返回alsa-lib继续执行。

(14) alsa-lib中的dmix将数据与其他程序已经写入的数据进行混合后，写入映射内存，同时进行备份已备其他程序写入音频数据时混音。

alsa通过mmap机制将硬件申请的(mod_probe时)pci内存区映射到用户空间，这样，alsa-lib只需要往相应的位置写数据，硬件就能够直接读取了。

sum_buffer可以看做是映射内存的副本，主要是给其他应用程序混音用，不同的程序通过共享内存方式共同访问sum_buffer。

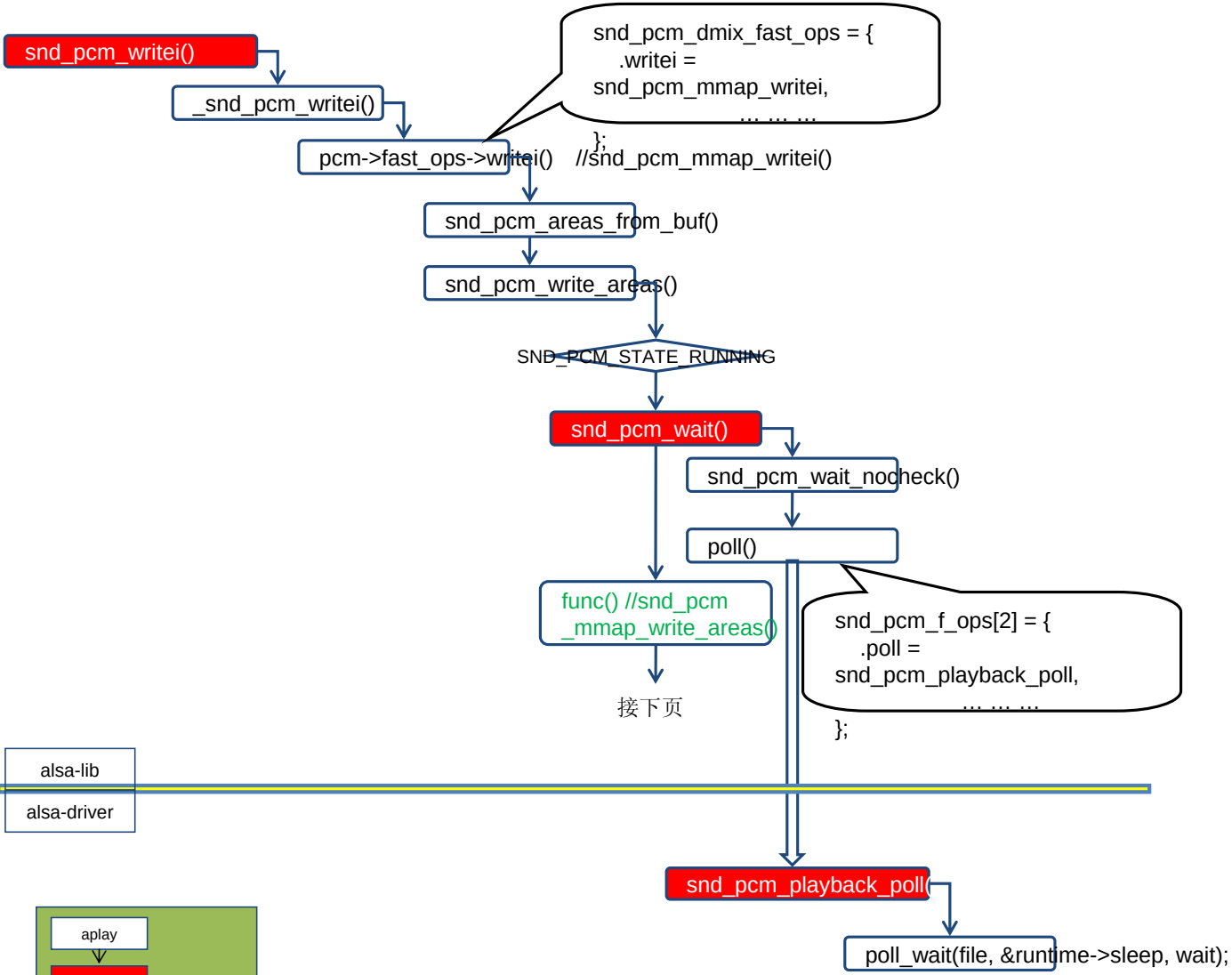
共同访问已经在alsa-lib中实现，具体应用程序不需要考虑。

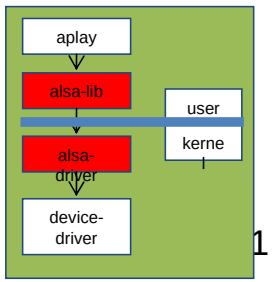
(sync) 在上述(1~14)的同时，如果映射内存中已经存有数据（比如上次写入的），硬件就会读取并进行DA转换（相关寄存器设置后，自动进行，不许软件干预）后播放出来。

详细流程见后续。

四、设备打开过程和数据流程

v. 数据写入流程图 --- alsa-lib API snd_pcm_writei()流程图





左图上半部alsa-driver对应的是"数据写入流程图"的(2)

snd_pcm_writei()函数通过dmix注册的writei调用snd_pcm_mmap_writei()，而snd_pcm_mmap_writei()在等待写入(poll)返回后，调用snd_pcm_write_areas()函数将数据写入dmix。snd_pcm_write_areas()的分析见后续页。

主要函数位置：
1. snd_pcm_writei()　alsa-lib-1.0.16/src/pcm/pcm.c : 1174
2. snd_pcm_mmap_writei()　alsa-lib-1.0.16/src/pcm/pcm_mmap.c : 182
3. snd_pcm_write_areas()　alsa-lib-1.0.16/src/pcm/pcm.c : 6415
4. snd_pcm_wait()　alsa-lib-1.0.16/src/pcm/pcm.c : 2256

左图下半部alsa-driver对应的是"数据写入流程图"的(3)

在alsa-driver中poll函数对应的snd_pcm_playback_poll ()调用poll_wait()函数，写入runtime->sleep队列进入睡眠等待。 runtime->sleep队列的唤醒是由硬件中断触发，alsa-dirver进行唤醒与否的判断后，在需要的情况下，唤醒进程。唤醒机制的详细流程见下页。

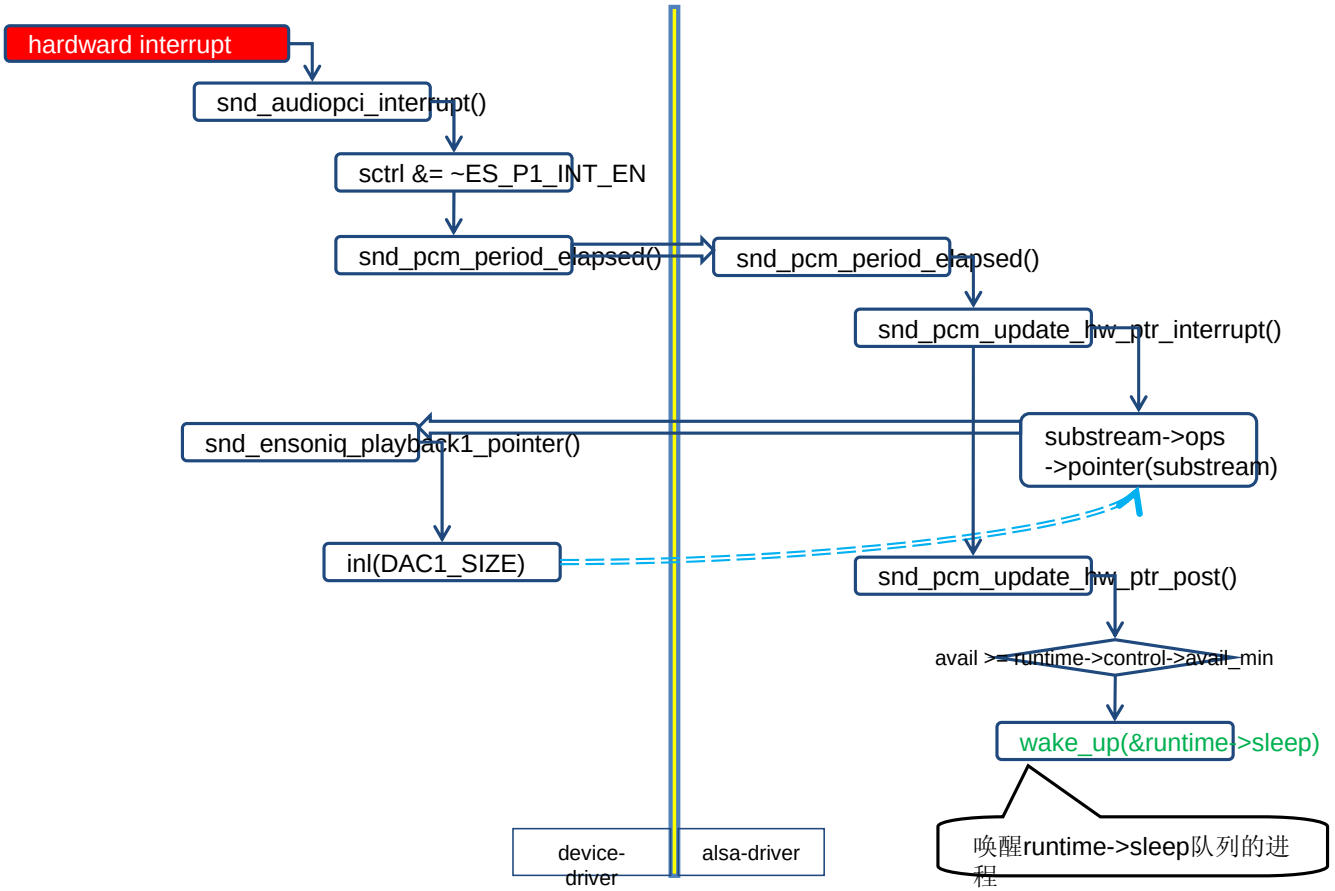
主要函数位置：
1. snd_pcm_playback_poll()
linux-source-2.6.26/sound/core/pcm_native.c : 2899

关于poll：poll允许进程决定是否可以对一个或多个打开的文件做非阻塞的读取或写入。这些调用也会阻塞进程，直到给定的文件描述符集合中的任何一个可以读取或写入。当用户空间程序在驱动程序关联的文件描述符上执行poll系统调用时，该驱动程序方法将被调用。驱动程序方法分为两步处理：

1. 在一个或多个可指示poll状态变化的等待队列上调用poll_wait。
2. 返回一个用来描述操作是否可以立即无阻塞执行的位掩码。
3. poll_wait函数并不阻塞，程序中poll_wait (filp, &outq, wait)这句话的意思并不是说一直等待outq信号量可获得，真正的阻塞动作是上层的select/poll函数中完成的。select/poll会在一个循环中对每个需要监听的设备调用它们自己的poll支持函数以使得当前进程被加入各个设备的等待列表。若当前没有任何被监听的设备就绪，则内核进行调度（调用schedule）让出cpu进入阻塞状态，schedule返回时将再次循环检测是否有操作可以进行，如此反复；否则，若有任意一个设备就绪， select/poll都立即返回。

四、设备打开过程和数据流程

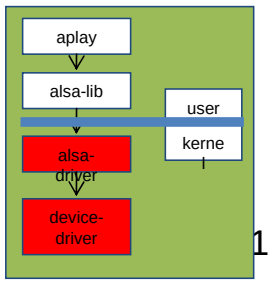
v. 数据写入流程图 --- hardward interrupt 流程图



左图上半部alsa-driver对应的是"数据写入流程图"的(4)(5)(6)(7)(8)(9)(10)(11)(12)

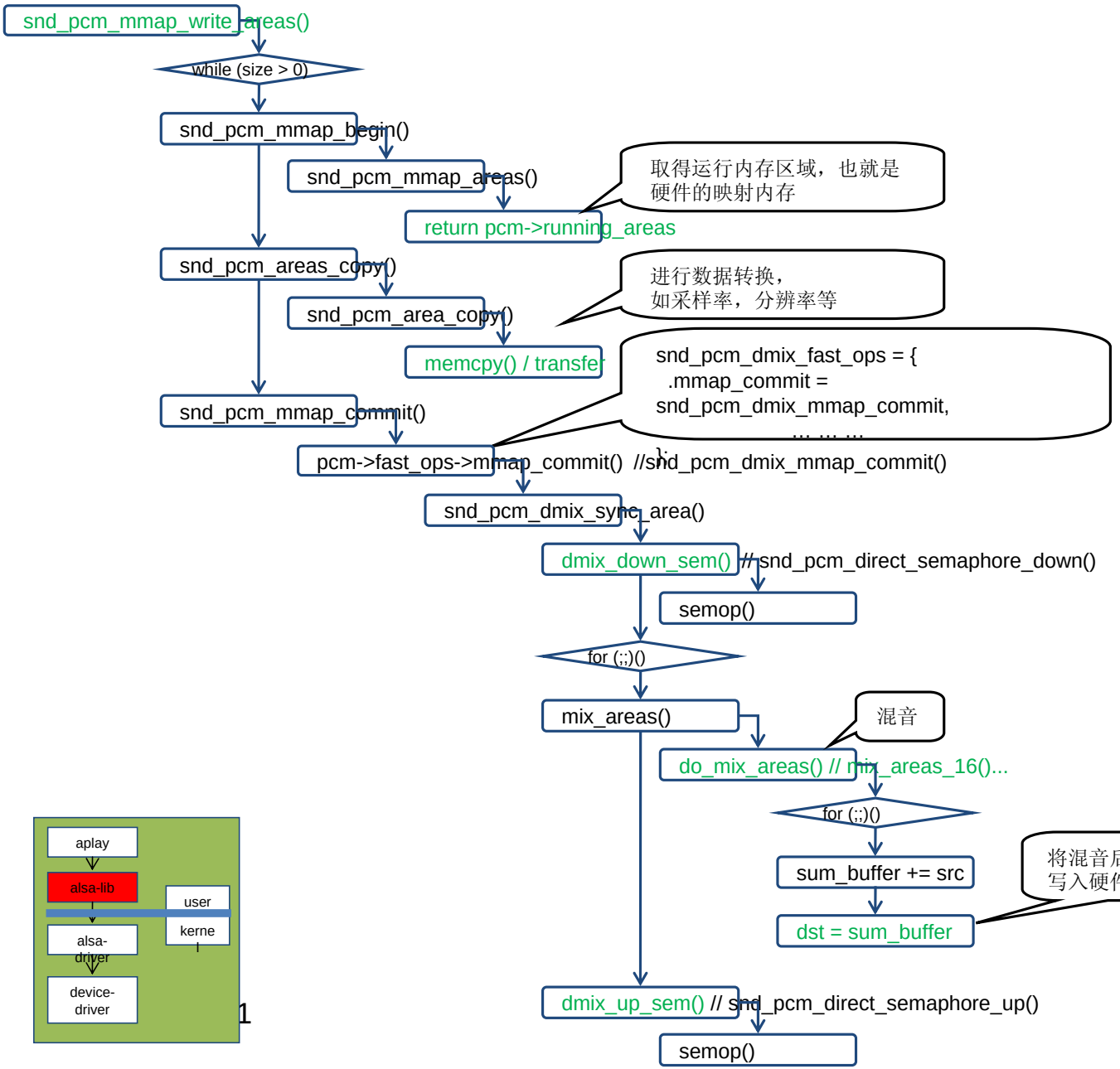
硬件会周期性的发生中断，触发snd_audiopci_interrupt()函数，snd_audiopci_interrupt()清楚硬件中断位后，调用snd_pcm_period_elapsed()函数进入alsa-driver的处理。snd_pcm_period_elapsed()回调snd_ensoniq_playback1_pointer()函数获取硬件为处理的数据size，并计算出硬件的读指针位置，再通过snd_pcm_update_hw_ptr_post()判断如果有效区(读写指针间的空闲区)大于设定的最小值，就调wake_up()唤醒runtime->sleep队列，也就唤醒了poll。使alsa-lib的写入进程继续执行。

主要函数位置：
1. snd_pcm_period_elapsed()
   linux-source-2.6.26/sound/core/ pcm_lib.c : 1464
2. snd_pcm_period_elapsed()
   linux-source-2.6.26/sound/core/ pcm_lib.c : 184
3. snd_pcm_period_elapsed()
   linux-source-2.6.26/sound/core/ pcm_lib.c : 161



四、设备打开过程和数据流程

v. 数据写入流程图 --- alsa-lib snd_pcm_write_areas()流程图



左图上半部alsa-driver对应的是"数据写入流程图"的(13)(14)

snd_pcm_mmap_write_areas()函数循环写入数据，直到数据没有，首先将找到映射内存pcm->running_areas的地址，然后调用snd_pcm_areas_copy()进行数据转换，如采样率，分辨率等，如果源数据和硬件格式一致，就简单地通过memcpy拷贝数据，转换成硬件对应的数据后，调用snd_pcm_mmap_commit()将转换后的数据写入映射内存。写入由snd_pcm_dmix_mmap_commit()完成，在对数据进行混音(do_mix_areas)同时，写入映射内存。

do_mix_areas()是通过软件进行混音的，如上图，将sum_buffer的数据和源数据进行叠加后，写入sum_buffer，同时写入dst。dst就是pcm->running_areas，也就是pci内存的用户层映射。至此，向硬件中写入的流程就全部完成了，硬件会从pci内存中读取出数据进行DA转换后播放出声音。

在这里，多个进程同时对sum_buffer进行写访问，是通过使用信号量函数semop()进行互斥访问的。

主要函数位置：

1. snd_pcm_mmap_areas()
   alsa-lib-1.0.16/src/pcm/pcm_local.h : 458
2. snd_pcm_area_copy()
   alsa-lib-1.0.16/src/pcm/pcm.c : 2527
3. snd_pcm_dmix_mmap_commit()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 773
4. snd_pcm_dmix_sync_area()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 297
5. dmix_down_sem()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 286
6. dmix_up_sem()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 287
7. snd_pcm_direct_semaphore_down()
   alsa-lib-1.0.16/src/pcm/pcm_direct.h : 251
8. snd_pcm_direct_semaphore_up()
   alsa-lib-1.0.16/src/pcm/pcm_direct.h : 257
9. mix_areas()
   alsa-lib-1.0.16/src/pcm/pcm_dmix.c : 152
10. mix_areas_16()
    alsa-lib-1.0.16/src/pcm/pcm_dmix_generic.c : 51

目 录

一、导读

二、ALSA架构简介

三、准备工作

四、设备打开过程和数据流程
- i. 整体分析
- ii. 设备驱动程序insmod流程图
- iii. 应用程序主流程图
- iv. 声卡打开流程图
- v. 数据写入流程图

五、ALSA其它形式的数据写入方法流程图

五、ALSA其它形式的数据写入方法流程图

使用alsa-lib自带的测试工具，可以测试其他形式的数据写入方法。其中测试pcm数据可以通过alsa-lib中的test文件夹中的pcm.c编译出的可执行文件测试。
具体使用方法如下：
#cd alsa-lib-1.0.16/test
#make pcm

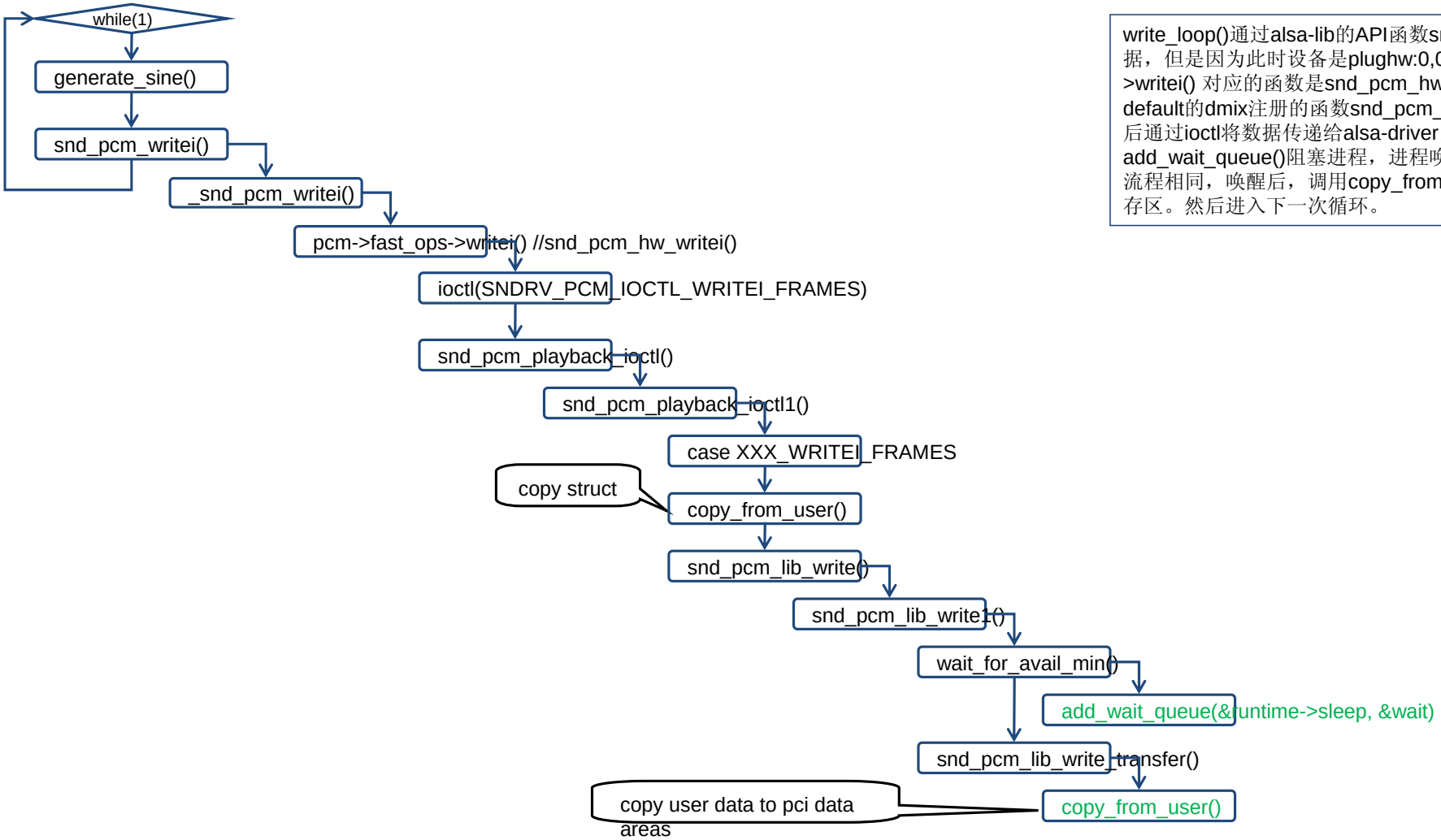
#./pcm -m write
#./pcm -m write_and_poll
#./pcm -m async
#./pcm -m async_direct
#./pcm -m direct_interleaved
#./pcm -m direct_noninterleaved
#./pcm -m direct_write

以上操作绕过alsa-lib中的dmix，直接测试的pcm_hw。
pcm.c中变量device赋“plughw:0,0”值。
可以通过-m设置7种不同的写方法，这些不同大部分体现在写入时的阻塞方式分别有，写入方式不同体现在：write_xxx是通过copy_from_user()将数据写入pci内存区，而direct_xxx是通过mmap()内存映射将数据写入pci内存区。
7种不同的方法分别为：
write：copy_from_user()、阻塞进程。
write_and_poll：copy_from_user()、阻塞poll。
async：copy_from_user()、异步方式。
async_direct：mmap() 、异步方式。
direct_interleaved：mmap() 、交叉存取。
direct_noninterleaved：mmap() 、非交叉存取。
direct_write：mmap() 、阻塞进程。
具体流程参见后续页。

如果将pcm.c中的device 设为“default"
make并执行后流程与前面的aplay类似。可以认为是mmap() 、阻塞poll方式。
修改pcm.c
static char *device = "plughw:0,0 default ";
重新编译，流程同aplay

五、ALSA其它形式的数据写入方法流程图

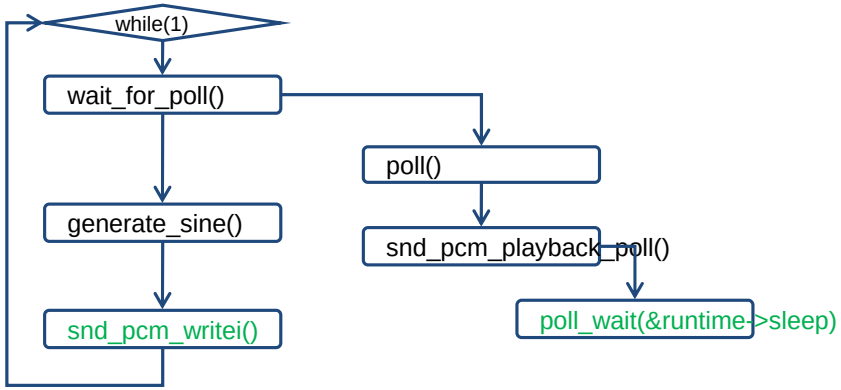
a. write_loop()



write_loop()通过alsa-lib的API函数snd_pcm_writei()写入数据，但是因为此时设备是plughw:0,0，所以pcm->fast_ops->writei() 对应的函数是snd_pcm_hw_writei()，此处与default的dmix注册的函数snd_pcm_mmap_writei()不同，然后通过ioctl将数据传递给alsa-driver，在驱动层中，调用add_wait_queue()阻塞进程，进程唤醒同aplay的硬件中断流程相同，唤醒后，调用copy_from_user()将数据写入pci内存区。然后进入下一次循环。

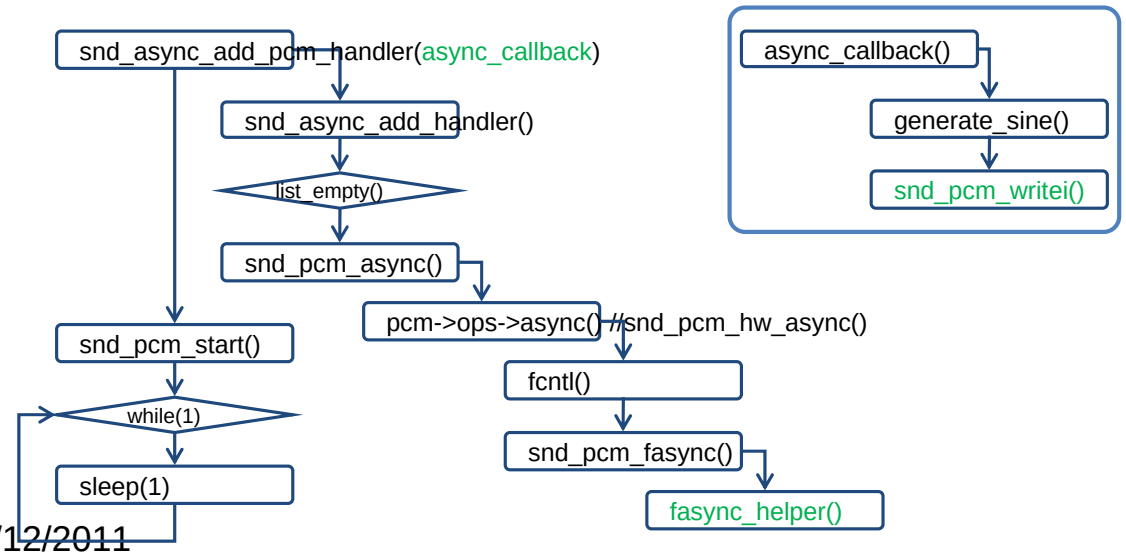
五、ALSA其它形式的数据写入方法流程图

b. write_and_poll_loop()



write_and_poll_loop()与write_loop()的不同体现在用poll的方式阻塞进程，相对于进程而言，poll的量级轻，可以获得较好的系统性能。

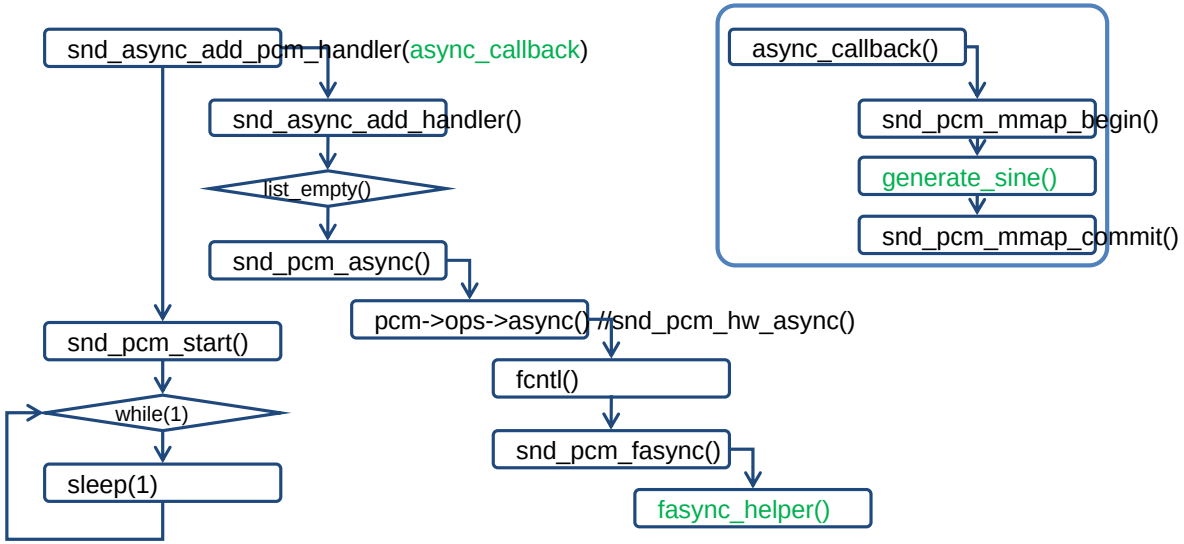
c. async_loop()



async_loop()是通过fcntl()注册回调函数异步的写入数据的，回调函数async_callback()负责写数据，而主循环实际上可以做其他的事情，在这里除了睡眠什么也没做。

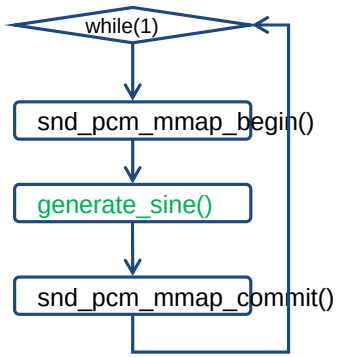
五、ALSA其它形式的数据写入方法流程图

d. async_direct_loop()



async_direct_loop()与async_loop()相同，不同点是回调函数使用mmap的方式写数据。此处的pcm_mmap和前面aplay的dmix_mmap原理相同，使用snd_pcm_mmap_begin()取得映射内存后，在generate_sine()生成正弦波时，同时写入映射内存。

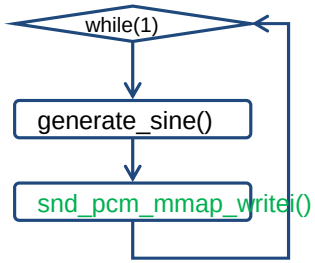
e. direct_loop()



direct_loop是direct_interleaved和direct_noninterleaved共同使用的主循环，direct_loop使用mmap机制，使用snd_pcm_mmap_begin()取得映射内存后，在generate_sine()生成正弦波时，同时写入映射内存。

五、ALSA其它形式的数据写入方法流程图

f. direct_write_loop()



direct_write_loop使用snd_pcm_mmap_writei()函数实现的写入，也是使用的mmap实现的，alsa-lib把begin、copy、commit进行封装提供了一个简单接口。
这也是alsa-lib的default中dmix的writei使用的方式。关于snd_pcm_mmap_writei()的详细流程可以参考aplay的流程。