复制位置： <http://blog.csdn.net/reille/article/details/5855859>

感谢原作者：

这里了解一下各个参数的含义以及一些基本概念。

样本长度(sample)：样本是记录音频数据最基本的单位，常见的有8位和16位。

通道数(channel)：该参数为1表示单声道，2则是立体声。

桢(frame)：桢记录了一个声音单元，其长度为样本长度与通道数的乘积。

采样率(rate)：每秒钟采样次数，该次数是针对桢而言。

周期(period)：音频设备一次处理所需要的桢数，对于音频设备的数据访问以及音频数据的存储，都是以此为单位。

交错模式(interleaved)：是一种音频数据的记录方式，在交错模式下，数据以连续桢的形式存放，即首先记录完桢1的左声道样本和右声道样本（假设为立体声格式），再开始桢2的记录。而在非交错模式下，首先记录的是一个周期内所有桢的左声道样本，再记录右声道样本，数据是以连续通道的方式存储。不过多数情况下，我们只需要使用交错模式就可以了。

英文原文：http://www.linuxjournal.com/article/6735  
period(周期):硬件中中断间的间隔时间。它表示输入延时。  
声卡接口中有一个指针来指示声卡硬件缓存区中当前的读写位置。只要接口在运行，这个指针将循环地指向缓存区中的某个位置。  
frame size = sizeof(one sample) \* nChannels  
alsa中配置的缓存(buffer)和周期(size)大小在runtime中是以帧(frames)形式存储的。  
period\_bytes = frames\_to\_bytes(runtime, runtime->period\_size);  
bytes\_to\_frames()  
  
The period and buffer sizes are not dependent on the sample format because they are measured in frames; you do not need to change them.  
  
ALSA声音编程介绍  
ALSA表示高级[**Linux**](http://lib.csdn.net/base/linux)声音体系结构(Advanced Linux Sound Architecture)。它由一系列内核驱动，应用程序编译接口(API)以及支持Linux下声音的实用程序组成。这篇文章里，我将简单介绍ALSA项目的基本框架以及它的软件组成。主要集中介绍PCM接口编程，包括您可以自动实践的程序示例。  
  
您使用ALSA的原因可能就是因为它很新，但它并不是唯一可用的声音API。如果您想完成低级的声音操作，以便能够最大化地控制声音并最大化地提高性能，或者如果您使用其它声音API没有的特性，那么ALSA是很好的选择。如果您已经写了一个音频程序，你可能想要为ALSA声卡驱动添加本地支持。如果您对音频不感兴趣，只是想播放音频文件，那么高级的API将是更好的选择，比如SDL,OpenAL以及那些桌面环境提供的工具集。另外，您只能在有ALSA支持的Linux环境中使用ALSA。  
  
ALSA历史  
ALSA项目发起的起因是Linux下的声卡驱动(OSS/Free drivers)没有得到积极的维护。并且落后于新的声卡技术。Jaroslav Kysela早先写了一个声卡驱动，并由此开始了ALSA项目，随便，更多的开发者加入到开发队伍中，更多的声卡得到支持，API的结构也得到了重组。  
  
Linux内核2.5在开发过程中，ALSA被合并到了官方的源码树中。在发布内核2.6后，ALSA已经内建在稳定的内核版本中并将广泛地使用。  
  
数字音频基础  
声音由变化的气压组成。它被麦克风这样的转换器转换成电子形式。模/数(ADC)转换器将模拟电压转换成离散的样本值。声音以固定的时间间隔被采样，采样的速率称为采样率。把样本输出到数/模(DAC)转换器，比如扩音器，最后转换成原来的模拟信号。  
样本大小以位来表示。样本大小是影响声音被转换成数字信号的精确程度的因素之一。另一个主要的因素是采样率。奈奎斯特(Nyquist)理论中，只要离散系统的奈奎斯特频率高于采样信号的最高频率或带宽，就可以避免混叠现象。  
  
ALSA基础  
ALSA由许多声卡的声卡驱动程序组成，同时它也提供一个称为libasound的API库。应用程序开发者应该使用libasound而不是内核中的ALSA接口。因为libasound提供最高级并且编程方便的编程接口。并且提供一个设备逻辑命名功能，这样开发者甚至不需要知道类似设备文件这样的低层接口。相反，OSS/Free驱动是在内核系统调用级上编程，它要求开发者提供设备文件名并且利用ioctrl来实现相应的功能。为了向后兼容，ALSA提供内核模块来模拟OSS，这样之前的许多在OSS基础上开发的应用程序不需要任何改动就可以在ALSA上运行。另外，libaoss库也可以模拟OSS，而它不需要内核模块。  
ALSA包含插件功能，使用插件可以扩展新的声卡驱动，包括完全用软件实现的虚拟声卡。ALSA提供一系列基于命令行的工具集，比如混音器(mixer)，音频文件播放器(aplay)，以及控制特定声卡特定属性的工具。  
  
ALSA体系结构  
ALSA API可以分解成以下几个主要的接口：  
1 控制接口：提供管理声卡注册和请求可用设备的通用功能  
2 PCM接口：管理数字音频回放(playback)和录音(capture)的接口。本文后续总结重点放在这个接口上，因为它是开发数字音频程序最常用到的接口。  
3 Raw MIDI接口:支持MIDI(Musical Instrument Digital Interface),标准的电子乐器。这些API提供对声卡上MIDI总线的访问。这个原始接口基于MIDI事件工作，由程序员负责管理协议以及时间处理。  
4 定时器(Timer)接口：为同步音频事件提供对声卡上时间处理硬件的访问。  
5 时序器(Sequencer)接口  
6 混音器(Mixer)接口  
  
设备命名  
API库使用逻辑设备名而不是设备文件。设备名字可以是真实的硬件名字也可以是插件名字。硬件名字使用hw:i,j这样的格式。其中i是卡号，j是这块声卡上的设备号。第一个声音设备是hw:0,0.这个别名默认引用第一块声音设备并且在本文示例中一真会被用到。插件使用另外的唯一名字。比如plughw:,表示一个插件，这个插件不提供对硬件设备的访问，而是提供像采样率转换这样的软件特性，硬件本身并不支持这样的特性。  
  
声音缓存和数据传输  
每个声卡都有一个硬件缓存区来保存记录下来的样本。当缓存区足够满时，声卡将产生一个中断。内核声卡驱动然后使用直接内存(DMA)访问通道将样本传送到内存中的应用程序缓存区。类似地，对于回放，任何应用程序使用DMA将自己的缓存区数据传送到声卡的硬件缓存区中。  
这样硬件缓存区是环缓存。也就是说当数据到达缓存区末尾时将重新回到缓存区的起始位置。ALSA维护一个指针来指向硬件缓存以及应用程序缓存区中数据操作的当前位置。从内核外部看，我们只对应用程序的缓存区感兴趣，所以本文只讨论应用程序缓存区。  
  
应用程序缓存区的大小可以通过ALSA库函数调用来控制。缓存区可以很大，一次传输操作可能会导致不可接受的延迟，我们把它称为延时(latency)。为了解决这个问题，ALSA将缓存区拆分成一系列周期(period)(OSS/Free中叫片断fragments).ALSA以period为单元来传送数据。  
一个周期(period)存储一些帧(frames)。每一帧包含时间上一个点所抓取的样本。对于立体声设备，一个帧会包含两个信道上的样本。图1展示了分解过程：一个缓存区分解成周期，然后是帧，然后是样本。图中包含一些假定的数值。图中左右信道信息被交替地存储在一个帧内。这称为交错(interleaved)模式。在非交错模式中，一个信道的所有样本数据存储在另外一个信道的数据之后。  
  
Over and Under Run  
当一个声卡活动时，数据总是连续地在硬件缓存区和应用程序缓存区间传输。但是也有例外。在录音例子中，如果应用程序读取数据不够快，循环缓存区将会被新的数据覆盖。这种数据的丢失被称为overrun.在回放例子中，如果应用程序写入数据到缓存区中的速度不够快，缓存区将会"饿死"。这样的错误被称为"underrun"。在ALSA文档中，有时将这两种情形统称为"XRUN"。适当地设计应用程序可以最小化XRUN并且可以从中恢复过来。  
  
一个典型的声音程序  
使用PCM的程序通常类似下面的伪代码：  
打开回放或录音接口  
设置硬件参数(访问模式，数据格式，信道数，采样率，等等)  
while 有数据要被处理：  
    读PCM数据(录音)  
    或 写PCM数据(回放)  
关闭接口  
  
我们将在下文中看到一些可以工作的代码。我建议您在你的Linux系统上[**测试**](http://lib.csdn.net/base/softwaretest)运行这些代码。查看输出并尝试修改推荐的代码。和本文相关的所有实例清单可以从FTP中获取：ftp.ssc.com/pub/lj/listings/issue126/6735.tgz。  
  
Listing 1. Display Some PCM Types and Formats  
  
#include <alsa/asoundlib.h>  
  
int main() {  
int val;  
  
printf("ALSA library version: %s/n",  
          SND\_LIB\_VERSION\_STR);  
  
printf("/nPCM stream types:/n");  
for (val = 0; val <= SND\_PCM\_STREAM\_LAST; val++)  
    printf(" %s/n",  
      snd\_pcm\_stream\_name((snd\_pcm\_stream\_t)val));  
  
printf("/nPCM access types:/n");  
for (val = 0; val <= SND\_PCM\_ACCESS\_LAST; val++)  
    printf(" %s/n",  
      snd\_pcm\_access\_name((snd\_pcm\_access\_t)val));  
  
printf("/nPCM formats:/n");  
for (val = 0; val <= SND\_PCM\_FORMAT\_LAST; val++)  
    if (snd\_pcm\_format\_name((snd\_pcm\_format\_t)val)  
      != NULL)  
      printf(" %s (%s)/n",  
        snd\_pcm\_format\_name((snd\_pcm\_format\_t)val),  
        snd\_pcm\_format\_description(  
                           (snd\_pcm\_format\_t)val));  
  
printf("/nPCM subformats:/n");  
for (val = 0; val <= SND\_PCM\_SUBFORMAT\_LAST;  
       val++)  
    printf(" %s (%s)/n",  
      snd\_pcm\_subformat\_name((  
        snd\_pcm\_subformat\_t)val),  
      snd\_pcm\_subformat\_description((  
        snd\_pcm\_subformat\_t)val));  
  
printf("/nPCM states:/n");  
for (val = 0; val <= SND\_PCM\_STATE\_LAST; val++)  
    printf(" %s/n",  
           snd\_pcm\_state\_name((snd\_pcm\_state\_t)val));  
  
return 0;  
}  
清单一显示了一些ALSA使用的PCM数据类型和参数。首先需要做的是包括头文件。这些头文件包含了所有库函数的声明。其中之一就是显示ALSA库的版本。  
这个程序剩下的部分的迭代一些PCM数据类型，以流类型开始。ALSA为每次迭代的最后值提供符号常量名，并且提供功能函数以显示某个特定值的描述字符串。你将会看到，ALSA支持许多格式，在我的1.0.15版本里，支持多达36种格式。  
这个程序必须链接到alsalib库，通过在编译时需要加上-lasound选项。有些alsa库函数使用dlopen函数以及浮点操作，所以您可能还需要加上-ldl,-lm选项。  
下面是该程序的Makefile:  
CC=gcc  
TARGET=test  
SRC=$(wildcard \*.c)  
  
OBJECT= ${SRC:.c=.o}  
INCLUDES=-I/usr/include/alsa  
LDFLAGS=-lasound  
  
all:$(TARGET)  
  
$(OBJECT):$(SRC)  
    $(CC) -c $(INCLUDES) $<  
  
$(TARGET):$(OBJECT)  
    $(CC) -o $@ $< $(LDFLAGS)  
  
.PHONY:clean  
  
clean:  
    @rm -rf $(OBJECT) $(TARGET) \*~  
      
  
Listing 2. Opening PCM Device and Setting Parameters  
  
/\*  
  
This example opens the default PCM device, sets  
some parameters, and then displays the value  
of most of the hardware parameters. It does not  
perform any sound playback or recording.  
  
\*/  
  
/\* Use the newer ALSA API \*/  
#define ALSA\_PCM\_NEW\_HW\_PARAMS\_API  
  
/\* All of the ALSA library API is defined  
\* in this header \*/  
#include <alsa/asoundlib.h>  
  
int main() {  
int rc;  
snd\_pcm\_t \*handle;  
snd\_pcm\_hw\_params\_t \*params;  
unsigned int val, val2;  
int dir;  
snd\_pcm\_uframes\_t frames;  
  
/\* Open PCM device for playback. \*/  
rc = snd\_pcm\_open(&handle, "default",  
                    SND\_PCM\_STREAM\_PLAYBACK, 0);  
if (rc < 0) {  
    fprintf(stderr,  
            "unable to open pcm device: %s/n",  
            snd\_strerror(rc));  
    exit(1);  
}  
  
/\* Allocate a hardware parameters object. \*/  
snd\_pcm\_hw\_params\_alloca(&params);  
  
/\* Fill it in with default values. \*/  
snd\_pcm\_hw\_params\_any(handle, params);  
  
/\* Set the desired hardware parameters. \*/  
  
/\* Interleaved mode \*/  
snd\_pcm\_hw\_params\_set\_access(handle, params,  
                      SND\_PCM\_ACCESS\_RW\_INTERLEAVED);  
  
/\* Signed 16-bit little-endian format \*/  
snd\_pcm\_hw\_params\_set\_format(handle, params,  
                              SND\_PCM\_FORMAT\_S16\_LE);  
  
/\* Two channels (stereo) \*/  
snd\_pcm\_hw\_params\_set\_channels(handle, params, 2);  
  
/\* 44100 bits/second sampling rate (CD quality) \*/  
val = 44100;  
snd\_pcm\_hw\_params\_set\_rate\_near(handle,  
                                 params, &val, &dir);  
  
/\* Write the parameters to the driver \*/  
rc = snd\_pcm\_hw\_params(handle, params);  
if (rc < 0) {  
    fprintf(stderr,  
            "unable to set hw parameters: %s/n",  
            snd\_strerror(rc));  
    exit(1);  
}  
  
/\* Display information about the PCM interface \*/  
  
printf("PCM handle name = '%s'/n",  
         snd\_pcm\_name(handle));  
  
printf("PCM state = %s/n",  
         snd\_pcm\_state\_name(snd\_pcm\_state(handle)));  
  
snd\_pcm\_hw\_params\_get\_access(params,  
                          (snd\_pcm\_access\_t \*) &val);  
printf("access type = %s/n",  
         snd\_pcm\_access\_name((snd\_pcm\_access\_t)val));  
  
snd\_pcm\_hw\_params\_get\_format(params, &val);  
printf("format = '%s' (%s)/n",  
    snd\_pcm\_format\_name((snd\_pcm\_format\_t)val),  
    snd\_pcm\_format\_description(  
                             (snd\_pcm\_format\_t)val));  
  
snd\_pcm\_hw\_params\_get\_subformat(params,  
                        (snd\_pcm\_subformat\_t \*)&val);  
printf("subformat = '%s' (%s)/n",  
    snd\_pcm\_subformat\_name((snd\_pcm\_subformat\_t)val),  
    snd\_pcm\_subformat\_description(  
                          (snd\_pcm\_subformat\_t)val));  
  
snd\_pcm\_hw\_params\_get\_channels(params, &val);  
printf("channels = %d/n", val);  
  
snd\_pcm\_hw\_params\_get\_rate(params, &val, &dir);  
printf("rate = %d bps/n", val);  
  
snd\_pcm\_hw\_params\_get\_period\_time(params,  
                                    &val, &dir);  
printf("period time = %d us/n", val);  
  
snd\_pcm\_hw\_params\_get\_period\_size(params,  
                                    &frames, &dir);  
printf("period size = %d frames/n", (int)frames);  
  
snd\_pcm\_hw\_params\_get\_buffer\_time(params,  
                                    &val, &dir);  
printf("buffer time = %d us/n", val);  
  
snd\_pcm\_hw\_params\_get\_buffer\_size(params,  
                         (snd\_pcm\_uframes\_t \*) &val);  
printf("buffer size = %d frames/n", val);  
  
snd\_pcm\_hw\_params\_get\_periods(params, &val, &dir);  
printf("periods per buffer = %d frames/n", val);  
  
snd\_pcm\_hw\_params\_get\_rate\_numden(params,  
                                    &val, &val2);  
printf("exact rate = %d/%d bps/n", val, val2);  
  
val = snd\_pcm\_hw\_params\_get\_sbits(params);  
printf("significant bits = %d/n", val);  
  
snd\_pcm\_hw\_params\_get\_tick\_time(params,  
                                  &val, &dir);  
printf("tick time = %d us/n", val);  
  
val = snd\_pcm\_hw\_params\_is\_batch(params);  
printf("is batch = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_is\_block\_transfer(params);  
printf("is block transfer = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_is\_double(params);  
printf("is double = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_is\_half\_duplex(params);  
printf("is half duplex = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_is\_joint\_duplex(params);  
printf("is joint duplex = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_can\_overrange(params);  
printf("can overrange = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_can\_mmap\_sample\_resolution(params);  
printf("can mmap = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_can\_pause(params);  
printf("can pause = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_can\_resume(params);  
printf("can resume = %d/n", val);  
  
val = snd\_pcm\_hw\_params\_can\_sync\_start(params);  
printf("can sync start = %d/n", val);  
  
snd\_pcm\_close(handle);  
  
return 0;  
}  
  
清单2打开默认的PCM设备，设置一些硬件参数并且打印出最常用的硬件参数值。它并不做任何回放或录音的操作。snd\_pcm\_open打开默认的PCM设备并设置访问模式为PLAYBACK。这个函数返回一个句柄，这个句柄保存在第一个函数参数中。该句柄会在随后的函数中用到。像其它函数一样，这个函数返回一个整数。如果返回值小于0,则代码函数调用出错。如果出错，我们用snd\_errstr打开错误信息并退出。  
为了设置音频流的硬件参数，我们需要分配一个类型为snd\_pcm\_hw\_param的变量。分配用到函数宏snd\_pcm\_hw\_params\_alloca。下一步，我们使用函数snd\_pcm\_hw\_params\_any来初始化这个变量，传递先前打开的PCM流句柄。  
接下来，我们调用API来设置我们所需的硬件参数。这些函数需要三个参数：PCM流句柄，参数类型，参数值。我们设置流为交错模式，16位的样本大小，2个信道，44100bps的采样率。对于采样率而言，声音硬件并不一定就精确地支持我们所定的采样率，但是我们可以使用函数snd\_pcm\_hw\_params\_set\_rate\_near来设置最接近我们指定的采样率的采样率。其实只有当我们调用函数snd\_pcm\_hw\_params后，硬件参数才会起作用。  
程序的剩余部分获得并打印一些PCM流参数，包括周期和缓冲区大小。结果可能会因为声音硬件的不同而不同。  
运行该程序后，做实验，改动一些代码。把设备名字改成hw：0,0,然后看结果是否会有变化。设置不同的硬件参数然后观察结果的变化。

Listing 3. Simple Sound Playback

/\*

This example reads standard from input and writes

to the default PCM device for 5 seconds of data.

\*/

/\* Use the newer ALSA API \*/

#define ALSA\_PCM\_NEW\_HW\_PARAMS\_API

#include <alsa/asoundlib.h>

int main() {

long loops;

int rc;

int size;

snd\_pcm\_t \*handle;

snd\_pcm\_hw\_params\_t \*params;

unsigned int val;

int dir;

snd\_pcm\_uframes\_t frames;

char \*buffer;

/\* Open PCM device for playback. \*/

rc = snd\_pcm\_open(&handle, "default",

SND\_PCM\_STREAM\_PLAYBACK, 0);

if (rc < 0) {

fprintf(stderr,

"unable to open pcm device: %s/n",

snd\_strerror(rc));

exit(1);

}

/\* Allocate a hardware parameters object. \*/

snd\_pcm\_hw\_params\_alloca(&params);

/\* Fill it in with default values. \*/

snd\_pcm\_hw\_params\_any(handle, params);

/\* Set the desired hardware parameters. \*/

/\* Interleaved mode \*/

snd\_pcm\_hw\_params\_set\_access(handle, params,

SND\_PCM\_ACCESS\_RW\_INTERLEAVED);

/\* Signed 16-bit little-endian format \*/

snd\_pcm\_hw\_params\_set\_format(handle, params,

SND\_PCM\_FORMAT\_S16\_LE);

/\* Two channels (stereo) \*/

snd\_pcm\_hw\_params\_set\_channels(handle, params, 2);

/\* 44100 bits/second sampling rate (CD quality) \*/

val = 44100;

snd\_pcm\_hw\_params\_set\_rate\_near(handle, params,

&val, &dir);

/\* Set period size to 32 frames. \*/

frames = 32;

snd\_pcm\_hw\_params\_set\_period\_size\_near(handle,

params, &frames, &dir);

/\* Write the parameters to the driver \*/

rc = snd\_pcm\_hw\_params(handle, params);

if (rc < 0) {

fprintf(stderr,

"unable to set hw parameters: %s/n",

snd\_strerror(rc));

exit(1);

}

/\* Use a buffer large enough to hold one period \*/

snd\_pcm\_hw\_params\_get\_period\_size(params, &frames,

&dir);

size = frames \* 4; /\* 2 bytes/sample, 2 channels \*/

buffer = (char \*) malloc(size);

/\* We want to loop for 5 seconds \*/

snd\_pcm\_hw\_params\_get\_period\_time(params,

&val, &dir);

/\* 5 seconds in microseconds divided by

\* period time \*/

loops = 5000000 / val;

while (loops > 0) {

loops--;

rc = read(0, buffer, size);

if (rc == 0) {

fprintf(stderr, "end of file on input/n");

break;

} else if (rc != size) {

fprintf(stderr,

"short read: read %d bytes/n", rc);

}

rc = snd\_pcm\_writei(handle, buffer, frames);

if (rc == -EPIPE) {

/\* EPIPE means underrun \*/

fprintf(stderr, "underrun occurred/n");

snd\_pcm\_prepare(handle);

} else if (rc < 0) {

fprintf(stderr,

"error from writei: %s/n",

snd\_strerror(rc));

} else if (rc != (int)frames) {

fprintf(stderr,

"short write, write %d frames/n", rc);

}

}

snd\_pcm\_drain(handle);

snd\_pcm\_close(handle);

free(buffer);

return 0;

}

清单3扩展了之前的示例。向声卡中写入了一些声音样本以实现声音回放。在这个例子中，我们从标准输入中读取数据，每个周期读取足够多的数据，然后将它们写入到声卡中，直到5秒钟的数据全部传输完毕。

这个程序的开始处和之前的版本一样---打开PCM设备、设置硬件参数。我们使用由ALSA自己选择的周期大小，申请该大小的缓冲区来存储样本。然后我们找出周期时间，这样我们就能计算出本程序为了能够播放5秒钟，需要多少个周期。

在处理数据的循环中，我们从标准输入中读入数据，并往缓冲区中填充一个周期的样本。然后检查并处理错误，这些错误可能是由到达文件结尾，或读取的数据长度与我期望的数据长度不一致导致的。

我们调用snd\_pcm\_writei来发送数据。它操作起来很像内核的写系统调用，只是这里的大小参数是以帧来计算的。我们检查其返回代码值。返回值为EPIPE表明发生了underrun，使得PCM音频流进入到XRUN状态并停止处理数据。从该状态中恢复过来的标准方法是调用snd\_pcm\_prepare函数，把PCM流置于PREPARED状态，这样下次我们向该PCM流中数据时，它就能重新开始处理数据。如果我们得到的错误码不是EPIPE，我们把错误码打印出来，然后继续。最后，如果写入的帧数不是我们期望的，则打印出错误消息。

这个程序一直循环，直到5秒钟的帧全部传输完，或者输入流读到文件结尾。然后我们调用snd\_pcm\_drain把所有挂起没有传输完的声音样本传输完全，最后关闭该音频流，释放之前动态分配的缓冲区，退出。

我们可以看到这个程序没有什么用，除非标准输入被重定向到了其它其它的文件。尝试用设备/dev/urandom来运行这个程序，该设备产生随机数据：

./example3 </dev/urandom

随机数据会产生5秒钟的白色噪声。  
然后，尝试把标准输入重定向到设备/dev/null和/dev/zero上，并比较结果。改变一些参数，例如采样率和数据格式，然后查看结果的变化。  
  
Listing 4. Simple Sound Recording  
  
  
/\*  
  
This example reads from the default PCM device  
and writes to standard output for 5 seconds of data.  
  
\*/  
  
/\* Use the newer ALSA API \*/  
#define ALSA\_PCM\_NEW\_HW\_PARAMS\_API  
  
#include <alsa/asoundlib.h>  
  
int main() {  
long loops;  
int rc;  
int size;  
snd\_pcm\_t \*handle;  
snd\_pcm\_hw\_params\_t \*params;  
unsigned int val;  
int dir;  
snd\_pcm\_uframes\_t frames;  
char \*buffer;  
  
/\* Open PCM device for recording (capture). \*/  
rc = snd\_pcm\_open(&handle, "default",  
                    SND\_PCM\_STREAM\_CAPTURE, 0);  
if (rc < 0) {  
    fprintf(stderr,  
            "unable to open pcm device: %s/n",  
            snd\_strerror(rc));  
    exit(1);  
}  
  
/\* Allocate a hardware parameters object. \*/  
snd\_pcm\_hw\_params\_alloca(&params);  
  
/\* Fill it in with default values. \*/  
snd\_pcm\_hw\_params\_any(handle, params);  
  
/\* Set the desired hardware parameters. \*/  
  
/\* Interleaved mode \*/  
snd\_pcm\_hw\_params\_set\_access(handle, params,  
                      SND\_PCM\_ACCESS\_RW\_INTERLEAVED);  
  
/\* Signed 16-bit little-endian format \*/  
snd\_pcm\_hw\_params\_set\_format(handle, params,  
                              SND\_PCM\_FORMAT\_S16\_LE);  
  
/\* Two channels (stereo) \*/  
snd\_pcm\_hw\_params\_set\_channels(handle, params, 2);  
  
/\* 44100 bits/second sampling rate (CD quality) \*/  
val = 44100;  
snd\_pcm\_hw\_params\_set\_rate\_near(handle, params,  
                                  &val, &dir);  
  
/\* Set period size to 32 frames. \*/  
frames = 32;  
snd\_pcm\_hw\_params\_set\_period\_size\_near(handle,  
                              params, &frames, &dir);  
  
/\* Write the parameters to the driver \*/  
rc = snd\_pcm\_hw\_params(handle, params);  
if (rc < 0) {  
    fprintf(stderr,  
            "unable to set hw parameters: %s/n",  
            snd\_strerror(rc));  
    exit(1);  
}  
  
/\* Use a buffer large enough to hold one period \*/  
snd\_pcm\_hw\_params\_get\_period\_size(params,  
                                      &frames, &dir);  
size = frames \* 4; /\* 2 bytes/sample, 2 channels \*/  
buffer = (char \*) malloc(size);  
  
/\* We want to loop for 5 seconds \*/  
snd\_pcm\_hw\_params\_get\_period\_time(params,  
                                         &val, &dir);  
loops = 5000000 / val;  
  
while (loops > 0) {  
    loops--;  
    rc = snd\_pcm\_readi(handle, buffer, frames);  
    if (rc == -EPIPE) {  
      /\* EPIPE means overrun \*/  
      fprintf(stderr, "overrun occurred/n");  
      snd\_pcm\_prepare(handle);  
    } else if (rc < 0) {  
      fprintf(stderr,  
              "error from read: %s/n",  
              snd\_strerror(rc));  
    } else if (rc != (int)frames) {  
      fprintf(stderr, "short read, read %d frames/n", rc);  
    }  
    rc = write(1, buffer, size);  
    if (rc != size)  
      fprintf(stderr,  
              "short write: wrote %d bytes/n", rc);  
}  
  
snd\_pcm\_drain(handle);  
snd\_pcm\_close(handle);  
free(buffer);  
  
return 0;  
}  
  
清单4类似于清单3中的程序，除了这里的程序时做声音的抓取(录音)。当打开PCM设备时我们指定打开模式为SND\_PCM\_STREAM\_CPATURE。在主循环中，我们调用snd\_pcm\_readi从声卡中读取数据，并把它们写入到标准输出。同样地，我们检查是否有overrun，如果存在，用与前例中相同的方式处理。  
运行清单4的程序将录制将近5秒钟的声音数据，并把它们发送到标准输出。你也可以重定向到某个文件。如果你有一个麦克风连接到你的声卡，可以使用某个混音程序(mixer)设置录音源和级别。同样地，你也可以运行一个CD播放器程序并把录音源设成CD。尝试运行程序4并把输出定向到某个文件，然后运行程序3播放该文件里的声音数据：  
./listing4 > sound.raw  
./listing3 < sound.raw  
如果你的声卡支持全双工，你可以通过管道把两个程序连接起来，这样就可以从声卡中听到录制的声音：  
./listing4 | ./listing3  
同样地，您可以做实验，看看采样率和样本格式的变化会产生什么影响。  
  
高级特性  
在前面的例子中，PCM流是以阻塞模式操作的，也就是说，直到数据已经传送完，PCM接口调用才会返回。在事件驱动的交互式程序中，这样会长时间阻塞应用程序，通常是不能接受的。ALSA支持以非阻塞模式打开音频流，这样读写函数调用后立即返回。如果数据传输被挂起，调用不能被处理，ALSA就是返回一个EBUSY的错误码。  
许多图形应用程序使用回调来处理事件。ALSA支持以异步的方式打开一个PCM音频流。这使得当某个周期的样本数据被传输完后，某个已注册的回调函数将会调用。  
  
这里用到的snd\_pcm\_readi和snd\_pcm\_writei调用和Linux下的读写系统调用类似。字母i表示处理的帧是交错式(interleaved)的。ALSA中存在非交互模式的对应的函数。Linux下的许多设备也支持mmap系统调用，这个调用将设备内存映射到主内存，这样数据就可以用指针来维护。ALSA也运行以mmap模式打开一个PCM信道，这允许有效的零拷贝(zero copy)方式访问声音数据。  
  
总结  
我希望这篇文章能够激励你尝试编写某些ALSA程序。伴随着2.6内核在Linux发布版本(distributions)中被广泛地使用，ALSA也将被广泛地采用。它的高级特征将帮助Linux音频程序更好地向前发展。  
Jaroslav Kysela和Takashi lwai帮助查阅了本文的草稿并提出了宝贵的意见，在此表示感谢。