**ALSA 0.9.0 HOWTO -- 基本PCM音频**

作者: Matthias Nagorni  
译者: Kevin Lei  
转载请注明出处:http://kevinlei.cublog.cn  
  
1. 介绍  
  
这个HOWTO计划提供一个简短的介绍,使用ALSA 0.9.0写一个简单的音频应用程序.  
  
Section2解释了PCM音频最基本的函数.如果你删除其中的解释文本,那么最后你会得到一个极小的PCM回放程序. Section3简短地讨论了一些PCM截获函数.  
  
在Section4 你将学习怎么为ALSA音序器写一个简单的客户端.这一章节基于seqdemo.c这个例子,一个可以接收MIDI事件并且表示大多数重要的事件类型的程 序.Section5演示ALSA MIDI音序器怎样被用于"路由"MIDI事件,从一个输入端口到一些输出端口.这一段基于midiroute.c这个例子.  
  
Section6 使PCM回放和MIDI输入联结起来,并且解释了迷你合成器--miniFMsynth.c.这一章节引入基于回调的音频回放,就像Paul Davis在linux-audio-dev邮件列表里建议的那样.Section7对基于ALSA音序器队列的MIDI调度提供一个短小的介绍,基于 miniArp.c这个例子.  
  
推荐你也看看doxygen生成的ALSA库函数参考.  
  
编译一个ALSA应用程序:  
  
只要用-lasound参数并且确保你已包含了  
#include <alsa/asoundlib.h>  
  
2. 基本PCM音频  
  
为ALSA 0.9.0写一个简单的PCM应用程序我们首先需要一个PCM设备的句柄.然后我们必须指定同时可供回放或截获的PCM流的方向.我们同样必须提供一些关于我们想要使用的设置选项的信息,比如缓冲区大小,采样率,PCM数据格式等.所以,首先我们声明:  
    
    /\* Handle for the PCM device \*/   
    snd\_pcm\_t \*pcm\_handle;            
  
    /\* Playback stream \*/  
    snd\_pcm\_stream\_t stream = SND\_PCM\_STREAM\_PLAYBACK;  
  
    /\* This structure contains information about    \*/  
    /\* the hardware and can be used to specify the  \*/        
    /\* configuration to be used for the PCM stream. \*/   
    snd\_pcm\_hw\_params\_t \*hwparams;  
    
最 重要的ALSA PCM设备接口是"plughw"和"hw"接口. 如果你使用"plughw"接口,你不需要很在意声卡硬件.如果你的声卡不支持你指定的采样率或采样格式,你的数据将自动被转换.这同样适用于声道的访问 类型和号码.对于"hw"接口,你必须检查你的硬件是否支持你想要使用的设置选项.  
  
    /\* Name of the PCM device, like plughw:0,0          \*/  
    /\* The first number is the number of the soundcard, \*/  
    /\* the second number is the number of the device.   \*/  
    char \*pcm\_name;  
    
然后我们初始化变量并分配一个hwparams结构:  
  
    /\* Init pcm\_name. Of course, later you \*/  
    /\* will make this configurable ;-)     \*/  
    pcm\_name = strdup("plughw:0,0");  
    
  
    /\* Allocate the snd\_pcm\_hw\_params\_t structure on the stack. \*/  
    snd\_pcm\_hw\_params\_alloca(&hwparams);  
    
现在我们可以打开PCM设备:  
  
    /\* Open PCM. The last parameter of this function is the mode. \*/  
    /\* If this is set to 0, the standard mode is used. Possible   \*/  
    /\* other values are SND\_PCM\_NONBLOCK and SND\_PCM\_ASYNC.       \*/   
    /\* If SND\_PCM\_NONBLOCK is used, read / write access to the    \*/  
    /\* PCM device will return immediately. If SND\_PCM\_ASYNC is    \*/  
    /\* specified, SIGIO will be emitted whenever a period has     \*/  
    /\* been completely processed by the soundcard.                \*/  
    if (snd\_pcm\_open(&pcm\_handle, pcm\_name, stream, 0) < 0) {  
      fprintf(stderr, "Error opening PCM device %s\n", pcm\_name);  
      return(-1);  
    }  
    
在我们可以往声卡写PCM数据之前,我们必须指定访问类型,采样格式,采样率,声道号码,周期数目以及周期大小.首先,我们以声卡的全部设置选项空间来初始化hwparams结构.  
  
    /\* Init hwparams with full configuration space \*/  
    if (snd\_pcm\_hw\_params\_any(pcm\_handle, hwparams) < 0) {  
      fprintf(stderr, "Can not configure this PCM device.\n");  
      return(-1);  
    }  
    
Information about possible configurations can be obtained with a set of functions named  
关于合适的设置选项的信息,能以函数命名的一个集合获得.  
  
    snd\_pcm\_hw\_params\_can\_<capability>  
    snd\_pcm\_hw\_params\_is\_<property>  
    snd\_pcm\_hw\_params\_get\_<parameter>  
    
绝大多数重要的参数的可用性,换句话说,访问类型,缓冲区大小,声道号码,采样格式,采样率,以及周期数目,可以以函数命名的一个集合来检验.  
  
    snd\_pcm\_hw\_params\_test\_<parameter>   
  
如果"hw"接口被使用,这些查询函数尤其重要.设置选项空间能以一个函数命名的集合限制在某一设置选项  
  
    snd\_pcm\_hw\_params\_set\_<parameter>  
    
例如,我们假设声卡可以被设置为16位,Little Endian数据的立体声回放,44100Hz采样率.相应地,我们限制设置选项空间匹配于这个设置选项:  
  
    int rate = 44100; /\* Sample rate \*/  
    int exact\_rate;   /\* Sample rate returned by \*/  
                      /\* snd\_pcm\_hw\_params\_set\_rate\_near \*/   
    int dir;          /\* exact\_rate == rate --> dir = 0 \*/  
                      /\* exact\_rate < rate  --> dir = -1 \*/  
                      /\* exact\_rate > rate  --> dir = 1 \*/  
    int periods = 2;       /\* Number of periods \*/  
    snd\_pcm\_uframes\_t periodsize = 8192; /\* Periodsize (bytes) \*/  
    
访 问类型指定了哪一个多声道数据储存在缓冲区的方法.对于交错访问,缓冲区里的每一个帧为声道容纳连续的采样数据.对于16位立体声数据,这意味着缓冲区以 字为单位为左右声道交错地容纳采样数据.对于非交错访问,每一个周期为第一个声道容纳所有采样数据接着是第二个声道的采样数据.  
    
    /\* Set access type. This can be either    \*/  
    /\* SND\_PCM\_ACCESS\_RW\_INTERLEAVED or       \*/  
    /\* SND\_PCM\_ACCESS\_RW\_NONINTERLEAVED.      \*/  
    /\* There are also access types for MMAPed \*/  
    /\* access, but this is beyond the scope   \*/  
    /\* of this introduction.                  \*/  
    if (snd\_pcm\_hw\_params\_set\_access(pcm\_handle, hwparams, SND\_PCM\_ACCESS\_RW\_INTERLEAVED) < 0) {  
      fprintf(stderr, "Error setting access.\n");  
      return(-1);  
    }  
    
    /\* Set sample format \*/  
    if (snd\_pcm\_hw\_params\_set\_format(pcm\_handle, hwparams, SND\_PCM\_FORMAT\_S16\_LE) < 0) {  
      fprintf(stderr, "Error setting format.\n");  
      return(-1);  
    }  
  
    /\* Set sample rate. If the exact rate is not supported \*/  
    /\* by the hardware, use nearest possible rate.         \*/   
    exact\_rate = rate;  
    if (snd\_pcm\_hw\_params\_set\_rate\_near(pcm\_handle, hwparams, &exact\_rate, 0) < 0) {  
      fprintf(stderr, "Error setting rate.\n");  
      return(-1);  
    }  
    if (rate != exact\_rate) {  
      fprintf(stderr, "The rate %d Hz is not supported by your hardware.\n   
                       ==> Using %d Hz instead.\n", rate, exact\_rate);  
    }  
  
    /\* Set number of channels \*/  
    if (snd\_pcm\_hw\_params\_set\_channels(pcm\_handle, hwparams, 2) < 0) {  
      fprintf(stderr, "Error setting channels.\n");  
      return(-1);  
    }  
  
    /\* Set number of periods. Periods used to be called fragments. \*/   
    if (snd\_pcm\_hw\_params\_set\_periods(pcm\_handle, hwparams, periods, 0) < 0) {  
      fprintf(stderr, "Error setting periods.\n");  
      return(-1);  
    }  
  
缓冲区尺寸的单元依赖于函数.一些时候是字节,一些时候是必须指定的帧的数目.一个帧是对所有声道的采样数据数组.对于16位立体声数据,一个帧的长度是4个字节.  
  
    /\* Set buffer size (in frames). The resulting latency is given by \*/  
    /\* latency = periodsize \* periods / (rate \* bytes\_per\_frame)     \*/  
    if (snd\_pcm\_hw\_params\_set\_buffer\_size(pcm\_handle, hwparams, (periodsize \* periods)>>2) < 0) {  
      fprintf(stderr, "Error setting buffersize.\n");  
      return(-1);  
    }  
    
如 果你的硬件不支持2的N次方的缓冲区大小,你可以使用snd\_pcm\_hw\_params\_set\_buffer\_size\_near函数.这个函数工作 起来与snd\_pcm\_hw\_params\_set\_rate\_near相似.现在我们为PCM设备申请由pcm\_handle指向的设置选项.这同样也 将准备好PCM设备.  
    
    /\* Apply HW parameter settings to \*/  
    /\* PCM device and prepare device  \*/  
    if (snd\_pcm\_hw\_params(pcm\_handle, hwparams) < 0) {  
      fprintf(stderr, "Error setting HW params.\n");  
      return(-1);  
    }  
    
在PCM设备被设置以后,我们可以开始对其写PCM数据.第一个写访问将开始PCM回放.对于交错写访问,我们使用函数:  
     
    /\* Write num\_frames frames from buffer data to    \*/   
    /\* the PCM device pointed to by pcm\_handle.       \*/  
    /\* Returns the number of frames actually written. \*/  
    snd\_pcm\_sframes\_t snd\_pcm\_writei(pcm\_handle, data, num\_frames);  
    
对于非交错访问,我们将必须使用函数:  
  
    /\* Write num\_frames frames from buffer data to    \*/   
    /\* the PCM device pointed to by pcm\_handle.       \*/   
    /\* Returns the number of frames actually written. \*/  
    snd\_pcm\_sframes\_t snd\_pcm\_writen(pcm\_handle, data, num\_frames);  
    
在PCM回放开始之后,我们必须确保我们的应用程序发送足够的数据到声卡缓冲区.否则,将发生缓冲区欠载.当这样一个缓冲区欠载发生以后,snd\_pcm\_prepare将被调用.一个简单的立体声锯齿波能以这样的方式生成:  
  
    unsigned char \*data;  
    int pcmreturn, l1, l2;  
    short s1, s2;  
    int frames;  
  
    data = (unsigned char \*)malloc(periodsize);  
    frames = periodsize >> 2;  
    for(l1 = 0; l1 < 100; l1++) {  
      for(l2 = 0; l2 < num\_frames; l2++) {  
        s1 = (l2 % 128) \* 100 - 5000;    
        s2 = (l2 % 256) \* 100 - 5000;    
        data[4\*l2] = (unsigned char)s1;  
        data[4\*l2+1] = s1 >> 8;  
        data[4\*l2+2] = (unsigned char)s2;  
        data[4\*l2+3] = s2 >> 8;  
      }  
      while ((pcmreturn = snd\_pcm\_writei(pcm\_handle, data, frames)) < 0) {  
        snd\_pcm\_prepare(pcm\_handle);  
        fprintf(stderr, "<<<<<<<<<<<<<<< Buffer Underrun >>>>>>>>>>>>>>>\n");  
      }  
    }  
  
如果我们想中止回放,我们既可以使用snd\_pcm\_drop,也可以使用snd\_pcm\_drain.第一个函数将立即中止回放并丢弃未回放的帧.后一个函数将在回放完所有帧后中止回放.  
  
    /\* Stop PCM device and drop pending frames \*/  
    snd\_pcm\_drop(pcm\_handle);  
  
    /\* Stop PCM device after pending frames have been played \*/   
    snd\_pcm\_drain(pcm\_handle);  
  
(全文完)  
  
参考链接:  
  
原文  
http://www.suse.de/~mana/alsa090\_howto.html  
  
Linux音频编程指南  
http://www-128.ibm.com/developerworks/cn/linux/l-audio/  
  
A Tutorial on Using the ALSA Audio API  
http://www.equalarea.com/paul/alsa-audio.html  
  
The ALSA library API reference  
http://www.alsa-project.org/alsa-doc/alsa-lib/