Linux音频编程指南Linux认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022_Linux_E9_9F_B3_E9_A2_c103_645150.htm 虽然目前Linux的优势主要体现在网络服务方面，但事实上同样也有着非常丰富的媒体功能，本文就是以多媒体应用中最基本的声音为对象，介绍如何在Linux平台下开发实际的音频应用程序，同时还给出了一些常用的音频编程框架。 一、数字音频 音频信号是一种连续变化的模拟信号，但计算机只能处理和记录二进制的数字信号，由自然音源得到的音频信号必须经过一定的变换，成为数字音频信号之后，才能送到计算机中作进一步的处理。 数字音频系统通过将声波的波型转换成一系列二进制数据，来实现对原始声音的重现，实现这一步骤的设备常被称为模/数转换器（A/D）。A/D转换器以每秒钟上万次的速率对声波进行采样，每个采样点都记录下了原始模拟声波在某一时刻的状态，通常称之为样本（sample），而每一秒钟所采样的数目则称为采样频率，通过将一串连续的样本连接起来，就可以在计算机中描述一段声音了。对于采样过程中的每一个样本来说，数字音频系统会分配一定存储位来记录声波的振幅，一般称之为采样分辩率或者采样精度，采样精度越高，声音还原时就会越细腻。 数字音频涉及到的概念非常多，对于在Linux下进行音频编程的程序员来说，最重要的是理解声音数字化的两个关键步骤：采样和量化。采样就是每隔一定时间就读一次声音信号的幅度，而量化则是将采样得到的声音信号幅度转换为数字值，从本质上讲，采样是时间上的数字化，而量化则是幅度上的数字化。下面介绍几个在进行音频

编程时经常需要用到的技术指标： 1. 采样频率 采样频率是指将模拟声音波形进行数字化时，每秒钟抽取声波幅度样本的次数。采样频率的选择应该遵循奈奎斯特（Harry Nyquist）采样理论：如果对某一模拟信号进行采样，则采样后可还原的最高信号频率只有采样频率的一半，或者说只要采样频率高于输入信号最高频率的两倍，就能从采样信号系列重构原始信号。正常人听觉的频率范围大约在20Hz~20kHz之间，根据奈奎斯特采样理论，为了保证声音不失真，采样频率应该在40kHz左右。常用的音频采样频率有8kHz、11.025kHz、22.05kHz、16kHz、37.8kHz、44.1kHz、48kHz等，如果采用更高的采样频率，还可以达到DVD的音质。 2. 量化位数 量化位数是对模拟音频信号的幅度进行数字化，它决定了模拟信号数字化以后的动态范围，常用的有8位、12位和16位。量化位越高，信号的动态范围越大，数字化后的音频信号就越可能接近原始信号，但所需要的存贮空间也越大。 3. 声道数 声道数是反映音频数字化质量的另一个重要因素，它有单声道和双声道之分。双声道又称为立体声，在硬件中有两条线路，音质和音色都要优于单声道，但数字化后占据的存储空间的大小要比单声道多一倍。 二、声卡驱动 出于对安全性方面的考虑，Linux下的应用程序无法直接对声卡这类硬件设备进行操作，而是必须通过内核提供的驱动程序才能完成。在Linux上进行音频编程的本质就是要借助于驱动程序，来完成对声卡的各种操作。 对硬件的控制涉及到寄存器中各个比特位的操作，通常这是与设备直接相关并且对时序的要求非常严格，如果这些工作都交由应用程序员来负责，那么对声卡的编程将变得异常复杂而困难起来，驱动程序的作用正是

要屏蔽硬件的这些底层细节，从而简化应用程序的编写。目前Linux下常用的声卡驱动程序主要有两种：OSS和ALSA。 最早出现在Linux上的音频编程接口是OSS（Open Sound System），它由一套完整的内核驱动程序模块组成，可以为绝大多数声卡提供统一的编程接口。OSS出现的历史相对较长，这些内核模块中的一部分（OSS/Free）是与Linux内核源码共同免费发布的，另外一些则以二进制的形式由4Front Technologies公司提供。由于得到了商业公司的鼎力支持，OSS已经成为在Linux下进行音频编程的事实标准，支持OSS的应用程序能够在绝大多数声卡上工作良好。 虽然OSS已经非常成熟，但它毕竟是一个没有完全开放源代码的商业产品，ALSA（Advanced Linux Sound Architecture）恰好弥补了这一空白，它是在Linux下进行音频编程时另一个可供选择的声卡驱动程序。ALSA除了像OSS那样提供了一组内核驱动程序模块之外，还专门为简化应用程序的编写提供了相应的函数库，与OSS提供的基于ioctl的原始编程接口相比，ALSA函数库使用起来要更加方便一些。ALSA的主要特点有：* 支持多种声卡设备 * 模块化的内核驱动程序 * 支持SMP和多线程 * 提供应用开发函数库 * 兼容OSS应用程序 ALSA和OSS最大的不同之处在于ALSA是由志愿者维护的自由项目，而OSS则是由公司提供的商业产品，因此在对硬件的适应程度上OSS要优于 ALSA，它能够支持的声卡种类更多。ALSA虽然不及OSS运用得广泛，但却具有更加友好的编程接口，并且完全兼容于OSS，对应用程序员来讲无疑是一个更佳的选择。 三、编程接口 如何对各种音频设备进行操作是在Linux上进行音频编程的关键，通过内核提供的一组系统调用，应用程序能够访

问声卡驱动程序提供的各种音频设备接口，这是在Linux下进行音频编程最简单也是最直接的方法。 3.1 访问音频设备 无论是OSS还是ALSA，都是以内核驱动程序的形式运行在Linux内核空间中的，应用程序要想访问声卡这一硬件设备，必须借助于Linux 内核所提供的系统调用（system call）。从程序员的角度来说，对声卡的操作在很大程度上等同于对磁盘文件的操作：首先使用open系统调用建立起与硬件间的联系，此时返回的文件描述符将作为随后操作的标识；接着使用read系统调用从设备接收数据，或者使用write系统调用向设备写入数据，而其它所有不符合读/写这一基本模式的操作都可以由ioctl系统调用来完成；最后，使用close系统调用告诉Linux内核不会再对该设备做进一步的处理。 * open系统调用 系统调用open可以获得对声卡的访问权，同时还能为随后的系统调用做好准备，其函数原型如下所示： int open(const char *pathname, int flags, int mode). 参数pathname是将要被打开的设备文件的名称，对于声卡来讲一般是 /dev/dsp。参数flags用来指明应该以什么方式打开设备文件，它可以是O_RDONLY、O_WRONLY或者O_RDWR，分别表示以只读、只写或者读写的方式打开设备文件；参数mode通常是可选的，它只有在指定的设备文件不存在时才会用到，指明新创建的文件应该具有怎样的权限。 如果open系统调用能够成功完成，它将返回一个正整数作为文件标识符，在随后的系统调用中需要用到该标识符。如果open系统调用失败，它将返回-1，同时还会设置全局变量errno，指明是什么原因导致了错误的发生。 * read系统调用 系统调用read用来从声卡读取数据，其函数原型如下所示： int read(int fd, char *buf, size_t count). 参数fd是

设备文件的标识符，它是通过之前的open系统调用获得的；参数 buf是指向缓冲区的字符指针，它用来保存从声卡获得的数据；参数count则用来限定从声卡获得的最大字节数。如果read系统调用成功完成，它将返回从声卡实际读取的字节数，通常情况会比count的值要小一些；如果read系统调用失败，它将返回-1，同时还会设置全局变量errno，来指明是什么原因导致了错误的发生。 * write系统调用 系统调用write用来向声卡写入数据，其函数原型如下所示： size_t write(int fd, const char *buf, size_t count). 系统调用write和系统调用read在很大程度是类似的，差别只在于 write是向声卡写入数据，而read则是从声卡读入数据。参数fd同样是设备文件的标识符，它也是通过之前的open系统调用获得的；参数buf是指向缓冲区的字符指针，它保存着即将向声卡写入的数据；参数count则用来限定向声卡写入的最大字节数。 如果write系统调用成功完成，它将返回向声卡实际写入的字节数；如果read系统调用失败，它将返回-1，同时还会设置全局变量errno，来指明是什么原因导致了错误的发生。无论是 read还是write，一旦调用之后Linux内核就会阻塞当前应用程序，直到数据成功地从声卡读出或者写入为止。 * ioctl系统调用 系统调用ioctl可以对声卡进行控制，凡是对设备文件的操作不符合读/写基本模式的，都是通过ioctl来完成的，它可以影响设备的行为，或者返回设备的状态，其函数原型如下所示： int ioctl(int fd, int request, ...). 参数fd是设备文件的标识符，它是在设备打开时获得的；如果设备比较复杂，那么对它的控制请求相应地也会有很多种，参数request的目的就是用来区分不同的控制请求；通常说来，在对设备进行控制时还需要有

其它参数，这要根据不同的控制请求才能确定，并且可能是与硬件设备直接相关的。 * close系统调用 当应用程序使用完声卡之后，需要用close系统调用将其关闭，以便及时释放占用的硬件资源，其函数原型如下所示： int close(int fd). 参数fd是设备文件的标识符，它是在设备打开时获得的。一旦应用程序调用了close系统调用，Linux内核就会释放与之相关的各种资源，因此建议在不需要的时候尽量及时关闭已经打开的设备。 3.2 音频设备文件 对于Linux应用程序员来讲，音频编程接口实际上就是一组音频设备文件，通过它们可以从声卡读取数据，或者向声卡写入数据，并且能够对声卡进行控制，设置采样频率和声道数目等等。 * /dev/sndstat 设备文件/dev/sndstat是声卡驱动程序提供的最简单的接口，通常它是一个只读文件，作用也仅仅只限于汇报声卡的当前状态。一般说来， /dev/sndstat是提供给最终用户来检测声卡的，不宜用于程序当中，因为所有的信息都可以通过 ioctl系统调用来获得。 Linux提供的cat命令可以很方便地从/dev/sndstat获得声卡的当前状态： [xiaowp@linuxgam sound]$ cat /dev/sndstat * /dev/dsp 声卡驱动程序提供的/dev/dsp是用于数字采样（sampling）和数字录音（recording）的设备文件，它对于Linux下的音频编程来讲非常重要：向该设备写数据即意味着激活声卡上的D/A转换器进行放音，而向该设备读数据则意味着激活声卡上的A/D转换器进行录音。目前许多声卡都提供有多个数字采样设备，它们在Linux下可以通过/dev/dsp1等设备文件进行访问。 DSP是数字信号处理器（Digital Signal Processor）的简称，它是用来进行数字信号处理的特殊芯片，声卡使用它来实现模拟信号和数字信号的转换。声卡中

的DSP设备实际上包含两个组成部分：在以只读方式打开时，能够使用A/D转换器进行声音的输入；而在以只写方式打开时，则能够使用D/A转换器进行声音的输出。严格说来，Linux下的应用程序要么以只读方式打开/dev/dsp输入声音，要么以只写方式打开/dev/dsp输出声音，但事实上某些声卡驱动程序仍允许以读写的方式打开 /dev/dsp，以便同时进行声音的输入和输出，这对于某些应用场合（如IP电话）来讲是非常关键的。 在从DSP设备读取数据时，从声卡输入的模拟信号经过A/D转换器变成数字采样后的样本（sample），保存在声卡驱动程序的内核缓冲区中，当应用程序通过read系统调用从声卡读取数据时，保存在内核缓冲区中的数字采样结果将被复制到应用程序所指定的用户缓冲区中。需要指出的是，声卡采样频率是由内核中的驱动程序所决定的，而不取决于应用程序从声卡读取数据的速度。如果应用程序读取数据的速度过慢，以致低于声卡的采样频率，那么多余的数据将会被丢弃；如果读取数据的速度过快，以致高于声卡的采样频率，那么声卡驱动程序将会阻塞那些请求数据的应用程序，直到新的数据到来为止。 在向DSP设备写入数据时，数字信号会经过D/A转换器变成模拟信号，然后产生出声音。应用程序写入数据的速度同样应该与声卡的采样频率相匹配，否则过慢的话会产生声音暂停或者停顿的现象，过快的话又会被内核中的声卡驱动程序阻塞，直到硬件有能力处理新的数据为止。与其它设备有所不同，声卡通常不会支持非阻塞（non-blocking）的I/O操作。 无论是从声卡读取数据，或是向声卡写入数据，事实上都具有特定的格式（format），默认为8位无符号数据、单声道、8KHz采样率，如果默认值

无法达到要求，可以通过ioctl系统调用来改变它们。通常说来，在应用程序中打开设备文件/dev/dsp之后，接下去就应该为其设置恰当的格式，然后才能从声卡读取或者写入数据。*/dev/audio /dev/audio类似于/dev/dsp，它兼容于Sun工作站上的音频设备，使用的是mu-law编码方式。如果声卡驱动程序提供了对 /dev/audio的支持，那么在Linux上就可以通过cat命令，来播放在Sun工作站上用 mu-law进行编码的音频文件：[xiaowp@linuxgam sound]$ cat audio.au 