Universidad Técnica Federico Santa María

Departamento de Electrónica

Programación de Sistemas

API de Sonido OSS/Free

Antonio Martínez Cárdenas

9704011-7

Resumen

El siguiente trabajo trata de los fundamentos de la programación de tarjetas de sonido, mediante la API OSS/Free.

La mayoría de las tarjetas de sonido cuenta con una interfaz MIDI, un digitilalizador de voz, un dispositivo mezclador o mixer y un sintetizador. La comunicación con estos dispositivos generalmente se realiza mediante los archivos ubicados en el filesystem /dev , denominados /dev/mixer, /dev/dsp, /dev/audio, /dev/sequencer y /dev/midi..

La API de sonido esta diseñada para permitir que las aplicaciones escritas con ella sean portables , tanto entre sistemas operativos como entre hardware de sonido.

Es importante mencionar que actualmente esta API no se sigue manteniendo, sin embargo la API (comercial) OSS se basa en gran medida en la API que desarrolla este trabajo, además de esta API (comercial) existe otro proyecto , sin embargo solo para Linux, que se denomina ALSA (Advanced Linux Sound Architecture) que provee drivers para el hardware de sonido y además una API, resulta interesante que esta API también es compatible con OSS.

En el trabajo se describen métodos de grabación y reproducción de sonido, la forma de fijar diversos parámetros como lo son la tasa de muestreo , el formato de muestreo , el número de canales disponibles (mono o estereo).

Además se realiza una descripción de como manipular el mixer, que es un dispositivo de control que supervisa los niveles de volumen de la entrada y la salida, además de conmutar los dispositivos disponibles.

Finalmente se tratan algunos tópicos de la programación MIDI (Musical Instruments Digital Interface) que es un puerto mediante el cual se puede conectar a un PC dispositivos externos, especialmente sintetizadores .

Introducción

Actualmente hasta las denominadas PC para negocios incluyen en la placa proncipal algún tipo de hardware para sonido, la idea de este trabajo es mostrar la API OSS/Free , através de pautas generales para la programación de sonido, esto se realiza mediante la manipulación de dispositivos mezcladores , mostrando como se puede implementar aplicaciones para la reproducción y grabación del sonido y mostrando de forma sencilla la programación para MIDI.

La API de Sonido OSS/Free

Para utilizar la API de sonido es necesario incluir en el código fuente el archivo de encabezado <sys/souncard.h>. Algunos dispositivos soportados por la API son:

/dev/mixer este dispositivo permite acceder a los circuitos mezcladores de la tarjeta , esto hace posible ajustar el volumen de varias fuentes de sonido.

/dev/sndstat este dispositivo es utilizado con propósitos de diagnóstico , a diferencia de los otros, produce salidas en formatos que permiten ser leídos. Ejecutando el comando "cat /dev/sndstat" arroja información sobre la configuración del controlador.

/dev/dsp y /dev/audio , estos son los dispositivos principales para aplicaciones de voz digitalizada.. Ambos dispositvos son muy similares , la diferencia es que el /dev/audio utiliza la ley mu para el formato de muestras, en cambio el /dev/dsp utiliza un formato de 8 bits.

/dev/sequencer este dispositvo es utilizado para aplicaciones de música electrónica, también es utilizado para producir efectos de sonido en juegos. Este dispositivo provee el acceso al sintetizador interno de la tarjeta de sonido , adicionalmente puede usarse para acceder sintetizadores conectados al puerto MIDI .

/dev/midi este dispositivo es un interfaz de bajo nivel para el puerto MIDI , que actúa como un terminal de caracteres.

Antes de comenzar a explicar lo que corresponde a la programación de sonido es necesario conocer la función ioctl , cuyo nombre proviene de input/output control , que es utilizada para manejar dispositivos de caracteres mediante un descriptor de archivo , su prototipo declarado en <ioctl.h>es el siguiente:

int ioctl(int fd , int request,....);

ioctl controla el dispositivo abierto cuyo descriptor de archivo esta indicado por fd y ejecuta el comando contenido en request , un tercer argumento es utilizado según el comando requerido, a veces corresponde a un puntero donde se guarda la salida del comando.

Seleccionando y Abriendo Dispositivos de Sonido.

Un dispositivo de audio debe ser abierto antes de ser utilizado. Existen tres tipos de dispositivos que difieren solo en la codificación de muestreo , /dev/dsp utiliza 8 bits sin signo, /dev/audio utiliza ley mu y /dev/dspw que utiliza 16 bits con signo(litle endian). Se recomienda abrir los dispositivos solo en modo escritura o lectura , el modo lectura-escritura(O_RDWR) debería ser utilizado solo cuando se requiere grabar y reproducir al mismo tiempo.El siguiente fragmento de código muestra como se puede abrir el dispositivo dado por la variable “nombre_dev” , el valor de open_mode puede

ser O_WRONLY,O_RDONLY o O_RDWR, los otros parámetros no están definidos para dispositivos de sonido y pueden no utilizarse.

```
if((audio_fd=open(nombre_dev,open_mode,0))==-1){
/*error al abrir dispositivo*/
perror(nombre_dev);
exit(1);
}
```

Como Grabar.

Escribir una aplicación que lea desde un dispositivo es muy sencillo, cuando la velocidad con que se graba es baja. El programa lo único que requiere es leer los datos desde el dispositivo y luego procesarlos o almacenarlos. El siguiente fragmento de código muestra como se lee información desde un dispositivo de audio.

```
int len;
if((len=read(audio_fd,audio_buffer,count))==-1){
perror("lectura de audio");
exit(1);
}
```

La reproducción funciona de una manera similar, la única diferencia es que la llamada realizada es write.

Parámetros de Muestreo.

Existen tres parámetro que afectan la calidad del sonido, estos son:

- formato de muestreo (llamado número de bits).
- número de canales.
- tasa de muestreo (velocidad).

La API soporta diferentes formatos de muestreo , que son:

AFMT_QUERY No es un formato de audio , pero se utiliza cunado se interroga el formato de audio corriente(SNDCTL_DSP_GETFMTS)

AFMT_MU_LAW Codificación logarítmica Ley μ

AFMT_A_LAW Codificación logarítmica Ley A (no es muy utilizada)

AFMT_IMA_DPCM Codificación ADPCM estándar

AFMT_U8 Codificación estándar de 8 bits sin signo

AFMT_S16_LE Formato de 16 bits con signo little endian (x86)

AFMT_S16_BE Formato de 16 bits con signo big endian (M68K,PPC,Sparc)

AFMT_S8 Formato de 8 bits con signo

AFMT_U16_LE Formato de 16 bits sin signo little endian

AFMT_U16_BE Formato de 16 bits sin signo big endian

AFMT_MPEG Formato audio MPEG.

El formato de muestreo se puede fijar mediante la función ioctl mediante la llamada SNDCTL_DSP_SETFMT. el siguiente segmento de código muestra como se fija el formato de audio AFMT_S16_LE , los otros formatos se fijan de forma similar.

```
int format;
format=AFMT_S16_LE;
if(ioctl(audio_fd,SNDCTL_DSP_SETFMT,&format)==-1){
perror("SNDCTL_DSP_SETFMT");
exit(1);
}
if(format!=SNDCTL_DSP_SETFMT){
```

```
/*el dispositivo no soporta el formato*/
}
```

El programa puede verificar que formatos son soportados por el dispositivo a través de la llamada SNDCTL_DSP_SETFMTS como lo muestro el siguiente ejemplo:

```
int mask ;
if(ioctl(audio_fd, SNDCTL_DSP_SETFMTS,&mask)){
/*capturar el error*/
}
if(mask & AFMT_MPEG){
/*el dispositvo soporta el formato MPEG*/
}
```

La llamada SNDCTL_DSP_SETFMTS retorna solo el formato que actualmente soporta el hardware , es posible que el driver soporte otros formatos utilizando alguna clase s de software . Estos formatos emulados , no son mostrados por la llamada SNDCTL_DSP_SETFMTS, pero la llamada SNDCTL_DSP_SETFMT si los soporta.

Selección del Número de Canales (mono/estereo).

Muchos dispositivos soportan el modo estereo , el modo por defecto es el modo mono, la aplicación puede cambiar el número de canales mediante la llamada SNDCTL_DSP_CHANNELS, tomando como argumento el número de canales, (algunos dispositivos soportan 16 canales).

```
int canales 2; /*mono=1, estereo =2*/
if(ioctl(audio_fd,SNDCTL_DSP_CHANNELS ,&canales)){
/*capturar el error*/
}
if (canales!2){
/*el dispositivo no soporta el modo estereo*/
```

```
}
```

Selección de la Tasa de muestreo.

La tasa de muestreo determina en gran medida la calidad del audio muestreado. La API permite elegir un amplio rango de frecuencias, al mínima frecuencia es de 5[KHz] y la máxima depende del dispositivo, actualmente la mayoría de la s tarjetas soporta 44.1[KHz] (mono) o 22.5[KHz](estereo), en dispositivos mas modernos se llega a 96[KHz] (calidad DVD). El siguiente segmento de código puede ser utilizado para fijar la frecuencia de muestreo.

```
int velocidad=11025;
if (ioctl(audio_fd,SNDCTL_DSP_SPEED,&speed)==-1){
perror("SNDCTL_DSP_SETFMT");
exit(1);
}
```

Programación del Mixer.

Como se especificó anteriormente el mixer es un dispositivo que permite controlar el nivel de volumen. Es posible tener mas de un dispositivo mezclador si es que se tiene más de una tarjeta de sonido , se denominan en el sistema de archivos como : /dev/mixer0, /dev/mixer1 ..etc. siendo /dev/mixer un enlace simbólico a alguno de estos dispositivos. Es posible que no existan mixer presentes en el sistema , existen tarjetas que simplemente no tienen esta funcionalidad. Todos los sistemas tienen el /dev/mixer0 pero la llamada a ioctl retornará un error, y fijara la variable errno a ENXIO. Para el mixer, la API se basa en canales , un canal es un objeto numerado que representa el control físico del mixer. Cada uno de los canales se puede ajustar de forma independiente y puede variara entre 0 (apagado) y 100 (volumen

máximo), adicionalmente se puede controlar el nivel de volumen de la fuente de grabación. Lo primero que se debe realizar es conocer las capacidades del mixer, esto se realiza mediante la función ioctl, la que permite verificar que canales se encuentran presentes y cuales pueden ser utilizados como entradas. Se recomienda que en los programas de audio por ejemplo, el código referente al mixer no se encuentren en la función principal, dado que en algunas tarjetas de sonido la implementación de bajo nivel del mixer puede ser significativamente diferente.

Los canales tiene un numero asociado entre 0 y 30. el archivo <soundcard.h >define algunos nombres para los canales. La macro de la función ioctl , SOUND_MIXER_NRDEVICES otorga el numero de canales conocidos. La siguiente lista muestra los canales mas utilizados :

SOUND_MIXER_VOLUME : nivel maestro de salida.

SOUND_MIXER_BASS : nivel de graves de todos los canales de salida.

SOUND_MIXER_TREBLE : nivel de agudos de todos los canales de salida .

SOUND_MIXER_SYNTH : control de volumen de todas las entradas al sintetizador, tales como el chip FM o la tabla de ondas.

SOUND_MIXER_PCM : nivel de salida de los dispositivos de audio /dev/dsp y /dev/audio

SOUND_MIXER_SPEAKER : nivel de salida del parlante del PC si es que esta conectado directamente a la tarjeta de sonido..

SOUND_MIXER_LINE : nivel de volumen para el conector de entrada de línea.

SOUND_MIXER_MIC : nivel de volumen para la entrada de micrófono.

SOUND_MIXER_CD : nivel de volumen de la entrada de CD de audio.

SOUND_MIXER_ALTPCM : nivel de volumen para dispositivo alternativo de audio .

La interfaz del mixer de la API ha sido implementada de modo que es posible compilar un programa en un sistema y puede ser utilizado en otro sistema con un hardware diferente.

Todas la funciones para verificar el mixer retornan un máscara de bits en una variable entera que es pasada como argumento a la llamada de ioctl, el siguiente segmento de código muestra un método genérico usada para las llamadas descritas posteriormente

```
int mask ;
if(ioctl(mixer_fd, SOUND_MIXER_READ_xxxx,&mask)){
/*el canal no esta presente */
}
```

El test correspondiente al bit del canal del mixer puede realizarse mediante la expresión "mask & (1 <<número_de_canal)".A continuación se tiene una descripción de las llamadas utilizadas para el mixer mediante la función ioctl:

- Canales existentes: para verificar los canales existentes se utiliza la llamada SOUND_MIXER_READ_DEVMASK , que retorna en la variable argumento en este caso en la variable mask, una másacra de bits . Si se desea determinar la presencia de un dispositivo se debe verificar si el bit correspondiente al canal tiene valor uno. Cualquier intento de acceder a un canal no definido mediante la función ioctl retornará un error (ernno se evaluará con EINVAL).

- Grabación: la llamada SOUND_MIXER_READ_RECMASK devuelve una máscara de bits que representa los canales que pueden ser utilizados para grabación.

- Dispositivo mono o estereo : existen dispositivos que funcionan en modo estereo teniendo la posibilidad de fijar independientemente los canales izquierdo y derecho del canal del mixer, aunque algunos canales son solo mono en donde solo se ajusta el canal izquierdo. A través de la llamada SOUND_MIXER_READ_STEREODEVS se obtiene una máscara de bits en que el uno indica que el canal correspondiente soporta estereo.

- Capacidades del mixer : la llamada SOUND_MIXER_READ_CAPS retorna una máscara de bits que describe las capacidades generales del mixer.

Fijar el Volumen.

Una aplicación tiene la posibilidad de leer o escribir el volumen de un dispositivo del mixer mediante la función de ioctl llamando a SOUND_MIXER_READ y SOUND_MIXER_WRITE respectivamente, el canal del mixer es pasado como argumento en la macro. El siguiente ejemplo muestra como se lee el nivel de volumen del a entrada del micrófono:

```
int vol;
if(ioctl(mixer_fd,SOUND_MIXER_READ(SOUND_MIXER_MIC),&vol)==-
1){
/*canal no definido*/
}
```

El volumen de ambos canales estereo es retornado en la misma variable , los bits menos significativos corresponden al canal izquierdo , y los siguientes al canal derecho, para dispositivos mono solo el nivel del canal izquierdo es válido. Como se mencionó anteriormente el volumen de un dispositivo puede ser modificado mediante la función ioctl con la llamada SOUND_MIXER_WRITE, trabaja de forma semejante a la llamada SOUND_MIXER_READ , adicionalmente altera el actual volumen del ca-

nal , cabe destacar que la llamada retorna el nuevo volumen en la variable pasada como argumento.

Seleccion de las Fuentes de grabación

Existen dos llamados para seleccionar las fuentes de grabación , adicionalmente SOUND_MIXER_READ_RECMASK retorna los dispositivos que pueden ser utilizados para grabación.

La ioctl SOUND_MIXER_READ_RECSRC retorna una máscara de bits que coloca un uno en cada recurso activo de grabación.

La SOUND_MIXER_READ_WRITE_RECSRC puede ser utilizada para modificar la selección del dispositivo fuente para la grabación , si es que no existen bits en uno el micrófono es utilizado.

Programación MIDI.

El acrónimo MIDI corresponde a Music Instruments Digital Interface, la interfaz de hardware MIDI utiliza un protocolo serial , asincrónico orientado al byte , similar al RS232 (pero no compatible), la tasa de transmición es de 31250 bits por segundo. La comunicación entre dispositivos midi requiere que el mensaje comience con un byte de estatus , adicionalmente puede llevar otros bytes, el byte de estatus tiene un 1 en el bit más significativo, mientras que los de datos tienen un cero en dicha posición. Los cuatro bits superiores de estatus especifican el tipo de estatus y los últimos cuatro bits indican el número del canal MIDI. Los bytes 0xF0 y 0xFF son reservados para mensajes del sistema.

Existen 16 canales posibles en el cable MIDI, cada uno de ellos puede ser asignado separadamente a diferentes dispositivos físicos. Un archivo MIDI contiene mensajes MIDI y otros datos que pueden ser utilizados por un

secuenciador MIDI y por otras aplicaciones.

Lectura de un Instrumento MIDI

Lo primero que se requiere es una lectura del byte de estatus, posteriormente la lectura desde el controlador de MIDI se obtiene de la siguiente forma:

- Se abre el dispositivo MIDI con la llamada open , esto retorna un descriptor de archivo .
- Leer el byte de estatus , usando la llamada read llamando el descriptor de archivo indicado en el paso anterior.
- Decodificar el byte de estatus según la tabla indicada en el anexo, se leen los datos requeridos según el estatus.
- Realizar las operaciones necesarias con los datos.
- Volver a leer el byte de estatus.

Mensajes consecutivos del mismo tipo pueden omitir el byte de estatus.

Conclusiones

El hardware de sonido presente en los computadores actuales, representa el sonido mediante una secuencia de muestras de una señal de audio, tomadas a intervalos precisamente controlados. Una muestra es el volumen de la señal de audio en el momento en que esta es tomada. La forma más simple del audio digital es el audio sin comprimir , en el cual cada muestra es almacenada tan pronto como es recibida en una secuencia de uno o más bytes. El audio comprimido a su vez permite codificar N bits en N-x bits de modo de ahorrar espacio de disco. Existen diversos formatos , como se mencionó en el desarrollo del trabajo. Estos diversos formatos se pueden manipular mediante la API , fijando los valores y consultando los formatos soportados por el hardware.

Lo que respecta a la grabación y reproducción de sonido , que son situaciones muy similares se deben tener en cuenta los siguientes aspectos:

1. Seleccionar el dispositivo que se desea utilizar.
2. Abrir el dispositivo.
3. Establecer el formato de muestreo del dispositivo.
4. Establecer el número de canales (1 o 2, mono o estereo).
5. Establecer la frecuencia de muestreo para la reproducción.
6. Leer un bloque del archivo que se requiere ejecutar.
7. Escribir dicho bloque al dispositivo de reproducción abierto.
8. Repetir los pasos 4 y 5 hasta encontrarse con EOF (fin de archivo).

9. Cerrar el dispositivo.

Lo que respecta al mixer lo primero que se debe realizar es conocer los dispositivos que soporta el hardware, luego manipular los diferentes canales, con las diferentes llamadas de la función ioctl .

Finalmente cabe destacar la facilidad del acceso a los dispositivos , ya que estos son tratados en forma similar a los archivos , por lo que no es complicado familiarizarse con el ambiente de programación.

A pesar de que esta API ya no se mantiene es importante , ya que actualmente los drivers y los programas aún son compatibles con ella, las otras API se encuentran en desarrollo como es el caso de ALSA, sin embargo la API OSS/free tiene una ventaja sobre esta y es que es compatible con diferente tipo de hardware, la otra API existente es comercial, OSS, pero esta última esta siendo desarrollada por las mismas personas que desarrollaron la API OSS/Free, por lo que esta última es en gran mediada compatible.

Bibliografía

-Programación en Linux con Ejemplos , Kurt Wall, Pentice Hall 2000.

-Open Sound System Programer´s Guide, 4Front Technologies.