ADQUISICIÓN DE SEÑALES DE AUDIO A TRAVÉS DE LA TARJETA DE SONIDO

Raúl Fernández Zavala
Cyntia E. Enríquez Ortiz

Referencia de este artículo [1].

La tarjeta de sonido de una computadora personal nos brinda la oportunidad de realizar procesamiento digital de señales en forma económica. Con ella podemos realizar aplicaciones para captura y edición de audio, síntesis de señales, análisis de formas de ondas en el dominio del tiempo o frecuencia, filtrado de señales de audio y en general cualquier forma imaginable de procesamiento digital de señales de audio.

Los sistemas operativos modernos como *Windows y Linux*, cuentan con diferentes interfaces de programación de aplicaciones (*APIs*) que permiten acceder a la tarjeta de sonido, tales como *DirectX (Windows), ALSA (Linux), OSS (Linux)*, entre otras. Incluso podríamos llevar a cabo la programación a bajo nivel, es decir, programar directamente la tarjeta de sonido a través de sus registros internos, sin embargo, esto sólo es recomendable cuando no se cuenta con una *API* para el sistema operativo, como en el caso de *MS-DOS*.

Para poder captura o reproducir audio debemos abrir y configurar la tarjeta de sonido. Una vez abierto el dispositivo, podemos escribir o leer bloques de datos, sin embargo, la forma de realizar esto depende del sistema operativo y de la *API* seleccionada. En este artículo nos enfocaremos en la *API* de multimedia de *Windows* y en particular en la captura de audio.

Para realizar la captura de audio debemos abrir la tarjeta de sonido utilizando la función *waveInOpen()*. Sin embargo Pero, antes de poder utilizar esta función, debemos conocer algunas estructuras de datos necesarios.

Tipo	**Descripción**
WAVEFORMATEX	Estructura que especifica el formato de los datos y configura el dispositivo de audio
WAVEHDR	Estructura utilizada como cabecera de un bloque de datos

En la estructura *WAVEFORMATEX* se especifica el codec, el número de canales utilizados, la frecuencia de muestreo y la cantidad de bits por muestra. Las siguientes líneas de código muestran la inicialización de una estructura *WAVEFORMATEX*, usando un codec *PCM*, una frecuencia de muestreo de 8000 Hz, una resolución de 8 bits y únicamente un canal:

```
WAVEFORMATEX wf;
wf.wFormatTag = WAVE_FORMAT_PCM;
wf.nChannels = 1;
wf.nSamplesPerSec = 8000;
wf.nAvgBytesPerSec = 8000;
wf.nBlockAlign = 1;
wf.wBitsPerSample = 8;
```

La estructura *WAVEHDR* especifica la localización del buffer de datos, su longitud y algunas banderas que proporcionan información del mismo. El siguiente código inicializa una estructura *WAVEHDR* para utilizar un buffer de 400 muestras de 8 bits:

```
WAVEHDR wh;
unsigned char BufferWaveIn[400];

wh.lpData= BufferWaveIn;
wh.dwBufferLength=400;
wh.dwFlags=0;
```

La función *waveInOpen()* abre el dispositivo asociado con el identificador especificado y en un parámetro pasado por referencia regresa un manejador al dispositivo abierto. Es recomendable utilizar el identificador *WAVE_MAPPER* para que el sistema seleccione el dispositivo que mejor se adapte al formato especificado. En la siguiente línea de código se abre la tarjeta de audio en modo de captura y mediante la bandera *CALLBACK_FUNCTION* se indica que la función *waveInProc()* es la encargada de procesar los datos de la señal de audio digitalizada:

```
HWAVEIN hWaveIn;
waveInOpen(  (LPHWAVEOUT)&hWaveIn, WAVE_MAPPER, &wf,
(DWORD)waveInProc, 0L, CALLBACK_FUNCTION  );
```

Después de abrir la tarjeta de audio debemos preparar el buffer con la función waveInPrepareHeader():

```
waveInPrepareHeader( hwIn, &wh, sizeof(WAVEHDR) );
```

Una vez que hemos preparado el buffer, debemos enviarlo a la tarjeta de sonido a través de la función *waveInAddBuffer()*:

```
waveInAddBuffer( hwIn, &wh, sizeof(WAVEHDR));
```

Finalmente podemos iniciar la captura de audio llamando a la función waveInStart():

```
waveInStart(hWaveIn);
```

Cuando el buffer de captura está lleno, se genera un mensaje *WIM_DATA* y la forma en que este se debe atender depende de la opción elegida al abrir la tarjeta. En nuestro caso se utilizó *CALLBACK_FUNCTION* y por lo tanto el mensaje se procesará en la función *waveInProc()*. Esta función copia el buffer que nos regresa la tarjeta de audio y vuelve a agregar el buffer para que la captura continúe, en caso de no hacerlo la captura termina. Dentro de esta función también se envía un mensaje *WM_PAINT* para representar gráficamente la señal de audio digitalizada.

```
void CALLBACK waveInProc( HWAVEIN hwi,UINT uMsg,DWORD dwInstance,DWORD
dwParam1,DWORD dwParam2 )
{
        LPWAVEHDR lpWvHdrIn=(LPWAVEHDR)dwParam1;
        switch(uMsg)
        {
                case WIM_DATA:
                CopyMemory(Buffer,lpWvHdrIn->lpData,BUFFER_SIZE);
                waveInAddBuffer(hWaveIn,lpWvHdrIn,sizeof(WAVEHDR));
                PostMessage(Form1->Handle,WM_PAINT,0,0);
                break;
        }
}
```

Para concluir se presenta una aplicación completa que muestra en una ventana la representación gráfica de una señal de audio. Este programa se desarrolló en Borland C++ Builder y consiste únicamente de un formulario que contiene dos botones y un componente PaintBox de 400x256 pixels. Para poder graficar la señal de audio debemos indicar en la ficha *Events del Object Inspector* que el manejador del evento *OnPaint*, para el componente *FORM1*, es la función *FORMPAINT()*. El código de la aplicación completa se lista a continuación:

```
#include <vcl.h>
```

```
#pragma hdrstop
#include "CapturaAudio.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
#include <mmsystem.h>
#define BUFFER_SIZE 400
#define NUM_BUFFERS 2
WAVEFORMATEX wf;
HWAVEIN hWaveIn;
WAVEHDR WaveHdrIn[NUM_BUFFERS];
unsigned char BufferWaveIn[NUM_BUFFERS*BUFFER_SIZE];
unsigned char Buffer[BUFFER_SIZE]={0};
void InitWaveFormat();
void InitWaveHdr();
void CALLBACK waveInProc( HWAVEIN hwi, UINT uMsg, DWORD dwInstance,
DWORD dwParam1, DWORD dwParam2 );
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
        InitWaveFormat();
        waveInOpen(  (LPHWAVEOUT)&hWaveIn, WAVE_MAPPER, &wf,
        (DWORD)waveInProc, 0L, CALLBACK_FUNCTION  );
        InitWaveHdr();
        for (int k=0;k<NUM_BUFFERS;k++)
        waveInAddBuffer(hWaveIn,&WaveHdrIn[k],sizeof(WAVEHDR));
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        waveInStart(hWaveIn);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
        waveInStop(hWaveIn);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormPaint(TObject *Sender)
{
        unsigned char y;
        PaintBox1->Canvas->Brush->Color = clWhite;
        PaintBox1->Canvas->Rectangle(0, 0, PaintBox1->Width, PaintBox1->Height);
        PaintBox1->Canvas->MoveTo(0,127);
        for (int x=0;x<BUFFER_SIZE;x++)
        {
                y=Buffer[x];
                PaintBox1->Canvas->LineTo(x,y);
        }
}
//---------------------------------------------------------------------------
void InitWaveFormat()
{
        wf.wFormatTag = WAVE_FORMAT_PCM;
        wf.nChannels = 1;
        wf.nSamplesPerSec = 8000;
        wf.nAvgBytesPerSec = 8000;
        wf.nBlockAlign = 1;
        wf.wBitsPerSample = 8;
```

```
}
//---------------------------------------------------------------------------
void InitWaveHdr()
{
        int k;
        for (k=0;k<NUM_BUFFERS;k++)
        {
                WaveHdrIn[k].lpData= BufferWaveIn+k*BUFFER_SIZE;
                WaveHdrIn[k].dwBufferLength=BUFFER_SIZE;
                WaveHdrIn[k].dwFlags=0;
                waveInPrepareHeader(hWaveIn, &WaveHdrIn[k],sizeof(WAVEHDR));
        }
}
//---------------------------------------------------------------------------
void CALLBACK waveInProc( HWAVEIN hwi,UINT uMsg,DWORD dwInstance,DWORD
dwParam1,DWORD dwParam2 )
{
        LPWAVEHDR lpWvHdrIn=(LPWAVEHDR)dwParam1;
        switch(uMsg)
        {
                case WIM_DATA:
                CopyMemory(Buffer,lpWvHdrIn->lpData,BUFFER_SIZE);
                waveInAddBuffer(hWaveIn,lpWvHdrIn,sizeof(WAVEHDR));
                PostMessage(Form1->Handle,WM_PAINT,0,0);
                break;
        }
}
```

La capacidad de procesamiento que ofrece una computadora personal en conjunto con una tarjeta de audio permite realizar diversas aplicaciones en el área de procesamiento digital de señales limitadas solamente por la creatividad del programador, en este artículo sólo se presenta el punto de partida para el desarrollo de aplicaciones más complejas.

Artículo realizado como parte del proyecto CGPI20061141

Referencias

[1] *Microsoft Development Network*, http://msdn.microsoft.com/
[2] Kent Reisdorph, “Aprendiendo Borland C++ Builder 3 en 21 dias”, 1a Edición, Prentice Hall 1999.
[3] Francisco Charte Ojeda, “Programación C++ Builder 2006”, 1a Edición Anaya Multimedia 2006.
[4] *Adavanced Linux Sound Architecture*, http://www.alsa-project.org/
[5] *Open Sound Systems*, http://www.opensound.com/