Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Urządzenia peryferyjne

Obsługa karty muzycznej z wykorzystaniem DirectSound, API i ActiveX

Ćwiczenie nr 11

Prowadzący:

dr inż. Jan Nikodem

Grupa:

Piątek 13:15 TP

Autorzy:

Daniel Glazer, 252743

Paweł Helisz, 252779

Wrocław 05.11.2021

1 Cel ćwiczenia

Celem ćwiczenia było napisanie programu, odtwarzającego dźwięk z wykorzystaniem komendy *PlaySound()* oraz z wykorzystaniem komponentu *ActiveX „Windows Media Player"*. Kolejną funkcją programu było sczytanie i wyświetlenie wartości nagłówka *WAV*(waveform audio format). Następnie należało odtworzyć plik *WAV* za pomocą bibliotek *DirectSound* i *Waveform and Auxiliary Audio*. Dodatkowo pliki w formacie *MP3* wybrane z listy plików należało odtworzyć wykorzystując *Windows Media Player*. Program miał także za zadanie sczytać dane z mikrofonu i zapisać je do pliku w formacie *WAV*. Zadaniem uzupełniającym było rozszerzenie funkcjonalności aplikacji o dodatkowe efekty dźwiękowe (np. echo) - wykorzystując implementację efektów z *DirectSounda*.

2 Opis działania programu

Program został napisany w języku C# i jego rozwiązanie zostało skompilowane do pliku .exe. Po uruchomieniu programu pojawia się okno aplikacji. W górnej części okna występują dwa przyciski. Jeden przycisk, który podpisany jest jako "Wczytaj plik *.wav" odpowiedzialny jest za wczytanie pliku dla odtwarzaczy PlaySound, WaveOut, DirectSound oraz dla efektu echo. Drugi przycisk odpowiada za załadowanie pliku do odtwarzacza *Windows Media Player*. Po załadowaniu pliku, odblokują się pozostałe przyciski, które odpowiadają za podpisane funkcje.

3 Odtwarzanie z wykorzystaniem *PlaySound()*

Funkcja PlaySound została zaimportowana z winmm.dll:

```
[DllImport("winmm.DLL", EntryPoint = "PlaySound", SetLastError = true,
            CharSet = CharSet.Unicode, ThrowOnUnmappableChar = true)]
private static extern bool PlaySound(string pszSound, System.IntPtr hmod,
                                     PlaySoundFlags fdwSound);
```

Jej argumentami są:

- pszSound - string, który zawiera ścieżkę do pliku, który ma zostać odtworzony. Maksymalna długość ścieżki wynosi 256 znaków. Jeśli ten argument przyjmuje wartość NULL, aktualnie odtwarzany plik dźwiękowy zostaje zatrzymany
- hmod - wskaźnik do pliku wykonywalnego, który zawiera zasób do załadowania
- fdwSound - flaga do obsługi odtwarzania dźwięku. Wartości flag zostały przedstawione w typie numerycznym PlaySoundFlags:

```
[System.Flags]
public enum PlaySoundFlags : int{
    SND_SYNC = 0x0000,
    SND_ASYNC = 0x0001,
    SND_NODEFAULT = 0x0002,
    SND_LOOP = 0x0008,
    SND_NOSTOP = 0x0010,
    SND_NOWAIT = 0x00002000,
    SND_FILENAME = 0x00020000,
    SND_RESOURCE = 0x00040004
}
```

Funkcja PlaySound została użyta w programie używając ścieżki do pliku (fileDirectory), System.IntPtr() jako wskaźnik, flaga została ustawiona na tryb asynchroniczny (PlaySoundFlags.SND_ASYNC):

```
string fileDirectory = fileDialog.InitialDirectory + fileDialog.FileName;
PlaySound(fileDirectory, new System.IntPtr(), PlaySoundFlags.SND_ASYNC);
```

Zatrzymanie funkcji PlaySound odbywa sie poprzez ustawenie ścieżki do pliku jako NULL:

```
PlaySound(null, (IntPtr)null, PlaySoundFlags.SND_ASYNC);
```

4 Odtwarzanie za pomocą wykorzystania *Komponentu ActiveX „Windows Media Player"*

Odtwarzacz Windows Media Player został dodany do składników COM projektu. Poniższy kod dodaje ścieżkę pliku do okna Windows Media Player umieszczonego w oknie programu:

```
axWindowsMediaPlayer1.URL = fileMediaDialog.InitialDirectory + fileMediaDialog.FileName;
```

Aby wybrać plik .mp3 lub .wav należało dodać filtr do przeglądanej listy plików:

```
var fileMediaDialog = new OpenFileDialog(); \\stworzenie okna listy plikow
fileMediaDialog.Filter = @"Plik wave (*.wav;*.mp3)|*.wav;*.mp3"; \\dodanie filtra
```

5 Sczytywanie i wyświetlanie nagłówka *WAV*

WAV (ang. waveform audio format) – format plików dźwiękowych stworzony przez Microsoft oraz IBM. WAVE bazuje na formacie RIFF, poszerzając go o informacje o strumieniu audio, takie jak użyty kodek, częstotliwość próbkowania czy liczba kanałów. WAV podobnie jak RIFF został przewidziany dla komputerów IBM PC, toteż wszystkie zmienne zapisywane są w formacie little endian. Odpowiednikiem WAV dla komputerów Macintosh jest AIFF. Pliki WAVE mogą być zapisane przy użyciu dowolnych kodeków audio, ale zazwyczaj stosuje się nieskompresowany format PCM, w którym

pliki zajmują około 172 kB na sekundę dla jakości CD. Ze względu na fakt, iż pole reprezentujące rozmiar pliku jest liczbą 32-bitową, wielkość pliku jest ograniczona do maksymalnie 4 GB. Nagłówek pliku WAV ma długość 44 bajtów. Struktura nagłówka została przedstawiona w poniższej tabeli:

Tabela 1: Struktura pliku WAV

Offset względem początku pliku	Rozmiar w bajtach	Nazwa	Opis zawartości
0 - 3	4	"RIFF"	Początek nagłówka używany do identyfikacji formatu RIFF
4 - 7	4	fileSize	Całkowity rozmiar pliku w bajtach pomniejszony o 8, czyli o "RIFF" i fileSize, równy jednocześnie rozmiarowi danych powiększonemu o 36 (24_h), czyli o fileSize - dataSize
8 - 11	4	"WAVE"	Typ pliku używany do identyfikacji formatu WAV
12 - 15	4	"fmt "	Tzw. format chunk marker (ang.). Ostatni bajt ma wartość 32 (20_h), czyli wartość ASCII dla spacji
16 - 19	4	16 (10_h)	Rozmiar formatu danych w bajtach
20 - 21	2	1	Typ formatu. Wartość 1 oznacza format PCM
22 - 23	2	channels	Liczba kanałów
24 - 27	4	sampleRate	Częstotliwość próbkowania. Wartość standardowa wynosi 44 100 Hz
28 - 31	4	bytesPerSecond	Strumień danych w bajtach na sekundę równy iloczynowi liczby kanałów przez częstotliwość próbkowania i rozdzielczość, podzielonemu przez 8
32 - 33	2	bytesPerSample	Iloczyn liczby kanałów przez rozdzielczość, podzielony przez 8, w bajtach
34 - 35	2	bitsPerSample	Rozdzielczość w bitach na próbkę (i kanał). Wartości standardowe wynoszą 8 lub 16 (10_h)
36 - 39	4	"data"	Początek danych
40 - 43	4	dataSize	Rozmiar danych w bajtach równy iloczynowi strumienia danych przez czas trwania dźwięku w sekundach

Aby sczytać dane z nagłówka użyto klasy FileStream z biblioteki System.IO:

```
using (var fileStream = new FileStream(sciezka, FileMode.Open, FileAccess.Read))
```

Po otworzeniu pliku sczytywano wartości kolejnych bajtów wg powyższej tabeli, używając do tego klasy BinaryReader z biblioteki System.IO:

```
using (var binaryReader = new BinaryReader(fileStream))
```

Dzięki otwarciu pliku i sczytywaniu kolejnych bajtów za pomocą funkcji BitConverter.ToInt16 lub BitConverter.ToInt32 w zależności od liczby bajtów składowych nagłowka (2 lub 4 bajty) oraz funkcji convertBinary pozwalającej na konwersję

bajtów na ciąg znaków, aby możliwe było wyświetlenie ich na ekranie, a także przekazanie danych do zmiennych: formattype, samplepersec, bytespersample, bitespersample. Zmienne te zostaną wykorzystane do odtwarzania dźwięku przy pomocy biblioteki DirectSound. Poniżej znajduje się implementacja całej funkcji:

```
using (var fileStream = new FileStream(sciezka, FileMode.Open, FileAccess.Read))
    using (var binaryReader = new BinaryReader(fileStream)){
    try{
    Console.WriteLine("RIFF: " + convertBinary(binaryReader.ReadBytes(4)));
    Console.WriteLine("SIZE: " + BitConverter.ToInt32(binaryReader.ReadBytes(4), 0));
    Console.WriteLine("WAVE: " + convertBinary(binaryReader.ReadBytes(4)));
    Console.WriteLine("fmt: " + convertBinary(binaryReader.ReadBytes(4)));
    Console.WriteLine("Rozmiar formatu: " +
                      BitConverter.ToInt32(binaryReader.ReadBytes(4), 0));

    formattype = BitConverter.ToInt16(binaryReader.ReadBytes(2), 0);
    Console.WriteLine("Typ formatu: " + formattype);

    channels = BitConverter.ToInt16(binaryReader.ReadBytes(2), 0);
    Console.WriteLine("Liczba kanalow: " + channels);

    samplespersec = BitConverter.ToInt32(binaryReader.ReadBytes(4), 0);
    Console.WriteLine("SampleRate: " + samplespersec);

    Console.WriteLine("BytesPerSecound " +
                      BitConverter.ToInt32(binaryReader.ReadBytes(4), 0));

    bytespersample = BitConverter.ToInt16(binaryReader.ReadBytes(2), 0);
    Console.WriteLine("BytesPerSample: " + bytespersample);

    bitespersample = BitConverter.ToInt16(binaryReader.ReadBytes(2), 0);
    Console.WriteLine("BitsPerSample: " + bitespersample);

    Console.WriteLine("Data: " + convertBinary(binaryReader.ReadBytes(4)));
    Console.WriteLine("DataSize: " + BitConverter.ToInt32((binaryReader.ReadBytes(4)), 0));
    }
    finally{
        binaryReader.Close(); //zamkniecie binaryReader
        fileStream.Close(); //zampkniecie fileStream
    }}
```

6 Odtwarzanie za pomocą *Waveform and Auxiliary Audio*

WaveOut jest interfejsem programowania aplikacji (API) używanym do odtwarzania dźwięków w formacie cyfrowym dla systemów Microsoft Windows. W naszym programie wykorzystaliśmy bibliotekę NAudio, która pozwala na obsługę interfejsu WaveOut w języku C#.Aby odtworzyć dźwięk wykorzystując funkcję waveOutWrite() skorzystaliśmy z klasy AudioFileReader, która w konstruktorze przyjmuje nazwę pliku audio. Uzyskaliśmy w ten sposób strumień zdekodowanych danych audio z pliku WAV.

```
audioFileReader = new NAudio.Wave.AudioFileReader(fileDialog.FileName);
```

Następnie strumień danych audio przekazano jako argument do funkcji:

```
waveOutDevice.Init(audioFileReader);
```

Po wywołaniu metody Play() na obiekcie klasy WaveOut, odtworzono dźwięk z pliku WAV.

```
waveOutDevice.Play();
```

Aby zatrzymać odtwarzanie dźwięku z pliku WAV należało wywołać metodę Stop(), a następnie zamknąć strumień AudioFileReader:

```
waveOutDevice.Stop();
audioFileReader.Close();
```

7 Sczytywanie danych z mikrofonu i zapisywanie ich do pliku

Do sczytania danych z mikrofonu użyto biblioteki NAudio. Utworzono obiekt klasy WaveIn odpowiedzialny za nagrywanie dźwięku. Format pliku przechowującego nagranie został ustawiony za pomocą dwóch parametrów: częstotliwości próbkowania (44100) i liczba kanałów (1).

```
waveIn = new NAudio.Wave.WaveIn();
waveIn.WaveFormat = new NAudio.Wave.WaveFormat(44100, 1);
```

Sczytanie danych polegało na użyciu EventHandlera do obsługi zdarzeń (pobieranie strumienia danych z mikrofonu) oraz utworzenie obiektu waveWriter, który obsługuje bufor zapisu danych do pliku WAV.

```
waveIn.DataAvailable += new EventHandler<NAudio.Wave.WaveInEventArgs>(waveIn_safeRecord);
string path = "test2.wav";
waveWriter = new NAudio.Wave.WaveFileWriter(path, waveIn.WaveFormat);
```

Funkcja waveIn_safeRecord za pomocą bufora waveWriter zapisuje dane do pliku. Jako argumenty funckja waveWriter.Write() przyjmuje jako argumenty tablicę danych, przesunięcie (offset) oraz liczbę nagranych bitów.

```
private void waveIn_safeRecord(object sender, NAudio.Wave.WaveInEventArgs e) {
            if (waveWriter != null){
                waveWriter.Write(e.Buffer, 0, e.BytesRecorded);
                waveWriter.Flush();}}
```

8 Odtwarzanie z wykorzystaniem *DirectSound*

DiectSound to komponent DirectX działający dla systemów Windows. Umożliwia szybki dostęp do karty dźwiękowej i m.in. odtwarzanie i nagrywanie dźwięku. Obsługuje efekty dźwiękowe i dźwięk trójwymiarowy. Do odtworzenia dźwięku z wykorzystaniem DirectSound użyto biblioteki SlimDX, która pozwala na implementację założeń programu w języku C#. Aby odtworzyć dźwięk należało utworzyć obiekt DirectSound, a następnie ustawić flagę kooperacji (CooperativeLevel) tak aby wątek działania sterownika DirectSound mógł współdziałać z elementem graficznym aplikacji.

```
    DirectSound ds = new DirectSound();
//ustawienie watku odtwarzania na priorytetowy
    ds.SetCooperativeLevel(window, CooperativeLevel.Priority);
```

Następnie należało utworzyć obiekt klasy WaveFormat do obsługi nagłówka odtwarzanego pliku WAV. W utworzonym obiekcie ustawiono pola nagłówka na wartości odczytane wcześniej w punkcie 5:

```
WaveFormat format = new WaveFormat();
        format.BitsPerSample = header.getBitespersample(); //rozdzielczosc w bitach
        format.BlockAlignment = header.getBytespersample(); //bajty na probke
        format.Channels = header.getChannels(); //liczba kanalow
        format.FormatTag = WaveFormatTag.Pcm; //typ formatu
        format.SamplesPerSecond = header.getSamplespersec(); //czestotliwosc probkowania
        format.AverageBytesPerSecond = format.SamplesPerSecond * format.BlockAlignment;
        //bajty na sekunde
```

Następnym krokiem było utworzenie buforów. DirectSound obsługuje dwa typy buforów: PrimarySoundBuffer z którego karta dźwiękowa pobiera dane, a programista nie ma nad nim bezpośredniej kontroli oraz SecondarySoundBuffer, który zawiera jedną ścieżkę dźwiękową oraz aktualną pozycję odczytu dźwięku. Sterownik DirectSound pobiera zawartość SecondarySoundBuffer i przekazuje go do PrimarySoundBuffer. Następnie karta dźwiękowa odczytuje zawartość pierwszego bufora i odtwarza dźwięk. Implementacja buforów została pokazana poniżej:

```
    //utworzenie opisu naglowka dla pBuffer
        SoundBufferDescription desc = new SoundBufferDescription();
        desc.Format = format;
        desc.Flags = BufferFlags.GlobalFocus; //flaga bufora ustawiona na odczyt globalny
        desc.SizeInBytes = 8 * format.AverageBytesPerSecond; //ustawienie rozmiaru bufora
    //utworzenie pierwszego bufora
        PrimarySoundBuffer pBuffer = new PrimarySoundBuffer(ds, desc);
    //utworzenie opisu naglowka dla sBuffer1
        SoundBufferDescription desc2 = new SoundBufferDescription();
        desc2.Format = format;
    //flagi ustawione na odczyt globalny, kontrole pozycji oraz obsluge efektow
        desc2.Flags = BufferFlags.GlobalFocus | BufferFlags.ControlPositionNotify
```

```
    |BufferFlags.GetCurrentPosition2 | BufferFlags.ControlEffects;
    desc2.SizeInBytes = 8 * format.AverageBytesPerSecond;
//utworzenie drugiego bufora
    this.sBuffer1 = new SecondarySoundBuffer(ds, desc2);
//utworzenie tablic dla odczytywania pozycji w pliku WAV
    NotificationPosition[] notifications = new NotificationPosition[2];
    notifications[0].Offset = desc2.SizeInBytes / 2 + 1;
    notifications[1].Offset = desc2.SizeInBytes - 1; ;
    notifications[0].Event = new AutoResetEvent(false);
    notifications[1].Event = new AutoResetEvent(false);
    sBuffer1.SetNotificationPositions(notifications);
//utworzenie tablic dla danych pliku WAV
    byte[] bytes1 = new byte[desc2.SizeInBytes / 2];
    byte[] bytes2 = new byte[desc2.SizeInBytes];
```

Kolejnym krokiem jest otworzenie pliku wykorzystując bibliotekę System.IO:

```
stream = File.Open(audioFile, FileMode.Open);
```

Wątek odczytywania danych z pliku WAV i przekazywanie go do SecondarySoundBuffer wygląda następująco:

```
this.fillBuffer = new Thread(() =>{
    int bytesRead;
//wczytywanie danych do bufora
    bytesRead = stream.Read(bytes2, 0, desc2.SizeInBytes);
    sBuffer1.Write<byte>(bytes2, 0, LockFlags.None);
//SecondarySoundBuffor zaczyna odtwarzac plik WAV
    sBuffer1.Play(0, playFlags);
     while (true){
        if (bytesRead == 0) { break; }
        notifications[0].Event.WaitOne();
        bytesRead = stream.Read(bytes1, 0, bytes1.Length);
        sBuffer1.Write<byte>(bytes1, 0, LockFlags.None);
        if (bytesRead == 0) { break; }
        notifications[1].Event.WaitOne();
        bytesRead = stream.Read(bytes1, 0, bytes1.Length);
        sBuffer1.Write<byte>(bytes1, desc2.SizeInBytes / 2, LockFlags.None);}
    stream.Close(); //zamkniecie strumienia danych
    stream.Dispose(); });
```

9 Dodatkowe efekty dźwiękowe - echo

Efekt echo został zaimplementowany przy użyciu efektu z biblioteki DirectSound. Obiekt echo pobiera wartośći z obiektu SoundEffectGuid.StandardEcho, a następnie zapisuje je w tablicy guids, która zostaje podana jako argument metody SetEffects. W ten sposób SecondarySoundBuffer przechowuje dźwięk korzystając z efektu echa:

```
var echo = SoundEffectGuid.StandardEcho;
var guids = new[] { echo };
sBuffer1.SetEffects(guids);
```

10 Wnioski

Zadania przedstawione w instrukcji laboratorium zostały wykonane poprawnie. Program napisany w języku C# wykorzystuje wcześniej opisane biblioteki do odtwarzania i zapisywania dźwięku. Wykonane zadania pozwoliły nam zapoznać się z budową bibliotek do odtwarzania dźwięku oraz opisem nagłówka pliku WAV.