Karta dźwiękowa

Jakub Oszust
241392

Adam Cierniak
241310

22 października 2019

Godzina zajęć: Środa TN, 11:00

Spis treści

1 Zadanie do wykonania

Nasze zadanie polegało na obsłudze karty dźwiękowej, do opracowania mieliśmy poszczególne zagadnienia:

1. Prosty mechanizm odtwarzania dźwięku ✓
   - Komenda PlaySound()
   - Komponent ActiveX 'Windows Media Player'
2. Plik WAV ✓
   - Rodzaje i formaty nagłówka pliku WAV
   - Informacje przechowywane w nagłówku pliku WAV
3. Waveform
   - Komendy typu WaveOut* potrzebne do odtwarzania dźwięku
4. MCI ✓
5. DirectSound
   - DirectX 10.0 Component DirectSound
6. Nagrywanie dźwięku z mikrofonu ✓

Symbolem ✓ zostały oznaczone zadania, które udało nam się zrealizować.

2 Wstęp teoretyczny

Karta dźwiękowa to komputerowa karta rozszerzeń pozwalająca na odtwarzanie, przetwarzanie oraz rejestrację dźwięku. W systemie operacyjnym Windows dostępne są biblioteki systemowe, służące wykonywaniu zadań karty dźwiękowej opisanych wyżej.

MCI (The Media Control Interface) jest interfejsem, który pozwala na wysyłanie prostych komend do obsługi multimediów dla narzędzi obsługujących interfejs, programistycznie najczęściej odbywa się to za pomocą komendy mciSendString, dostępnej w bibliotece winmm.dll.

DirectX to zestaw funkcji API stworzonych do obsługi mulitmediów dla systemu Windows, jego częścią jest DirectSound, który umożliwia dostęp do karty dźwiękowej.

Oprócz tego Windows dostarcza niskopoziomowe biblioteki Waveform audio, za którymi pomocą można obsługiwać funkcje karty dźwiękowej np. za pomocą dostępnej tam funkcji PlaySound()

3 Realizacja ćwiczenia

3.1 Struktura programu

Program został stworzony w języku C#, tak samo jak interfejs graficzny stworzony w Windows Forms. Funkcje oraz zmienne odpowiadające za granie dźwięku za pomocą różnych bibliotek zostały zaimplementowane w klasie SoundManager, która później została zaimportowana do okna aplikacji.

3.2 Algorytm działania

Aplikacja została napisana w ten sposób, aby interakcja z interfejsem użytkownika wywoływała funkcje zawarte w klasie SoundManager. Naciśnięcie przycisków do odtwarzania dźwięków, sprawi że otworzy się okno dialogowe z wyborem pliku dźwiękowego. Gdy plik zostanie wybrany, nazwa pliku zostanie przekazana do funkcji odtwarzającej dźwięk w wybrany sposób. Naciśnięcie przycisku nagrywania, a później naciśnięcie przycisku stop, sprawi, że w folderze w którym został zapisany projekt Visual Studio, pojawi się plik z nagraniem. Wybranie pliku .wav do analizy jego nagłowka, wyświetli na polu tekstowym, to co zawierają poszczególne bity nagłówka, wraz z ich znaczeniem.

Kod klasy Form (okno aplikacji):

```
 using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace UP1Official
{
    public partial class Form1 : Form
    {
        // Nazwa pliku do odtwarzania
        public string fileName { get; set; }

        // Nazwa pliku do nagrywania
        public string recordedFileName { get; set; }
        public Form1()
        {
            InitializeComponent();
            recordedFileName = "recsound";
        }

        // Wybieranie pliku do odtworzenia
        private void ChooseFileButton_Click(object sender, EventArgs
            e)
        {
            var dialog = new OpenFileDialog
```

```
    {
        Title = "Wybierz plik dzwiekowy .wav",
        InitialDirectory = @".",
        Filter = "Wav Files (*.wav)|*.wav",
        FilterIndex = 2,
        RestoreDirectory = true
    };
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        fileName = dialog.FileName;
        fileNameTextbox.Text = fileName;
    }
}

// Przycisk odtwarzajacy dzwiek za pomoca funkcji PlaySound
private void PlaySoundBtn_Click(object sender, EventArgs e)
{
    if (!String.IsNullOrWhiteSpace(fileName))
    {
        SoundManager.PlaySound(fileName, new IntPtr(),
            SoundManager.PlaySoundFlags.SND_ASYNC);
    }
}

// Przycisk zatrzymujacy odtwarzanie dzwieku puszczonego za
    pomoca funkcji PlaySound
private void StopPlaySoundBtn_Click(object sender, EventArgs
    e)
{
    if (!String.IsNullOrWhiteSpace(fileName))
    {
        SoundManager.PlaySound(null, new IntPtr(),
            SoundManager.PlaySoundFlags.SND_SYNC);
    }
}
// Po nacisnieciu tego przycisku pojawia sie dane dotyczace
    naglowka pliku .wav
private void WavDataBtn_Click(object sender, EventArgs e)
{
    if (!String.IsNullOrWhiteSpace(fileName))
    {
        byte[] wavFile = File.ReadAllBytes(fileName);
        byte[] wavHeader = new byte[44];
        for (int i = 0; i < 44; i++)
        {
            wavHeader[i] = wavFile[i];
        }
        string riffDescription = "";
        for (int i = 0; i < 4; i++)
```

```
{
    riffDescription += Encoding.ASCII.GetString(new
        byte[1] { wavHeader[i] });
}
string sizeOfFileBytes = "";
for (int i = 7; i >= 4; i--)
{
    sizeOfFileBytes += Convert.ToString(Convert.
        ToInt32(wavHeader[i]), 2);
}
string waveDescription = "";
for (int i = 8; i < 12; i++)
{
    waveDescription += Encoding.ASCII.GetString(new
        byte[1] { wavHeader[i] });
}
string fmtDescription = "";
for (int i = 12; i < 16; i++)
{
    fmtDescription += Encoding.ASCII.GetString(new
        byte[1] { wavHeader[i] });
}
string lengthOfFormatData = "";
for (int i = 16; i < 20; i++)
{
    lengthOfFormatData += Convert.ToInt32(wavHeader[i
        ]).ToString();
}
string typesOfFormatData = "";
for (int i = 20; i < 22; i++)
{
    typesOfFormatData += Convert.ToInt32(wavHeader[i]).
        ToString();
}
string numbersOfChannel = "";
for (int i = 22; i < 24; i++)
{
    numbersOfChannel += Convert.ToInt32(wavHeader[i]).
        ToString();
}
string sampleRate = "";
for (int i = 24; i < 28; i++)
{
    sampleRate += Convert.ToInt32(wavHeader[i]).
        ToString();
}
string sampleRateWithOtherThings = "";
for (int i = 28; i < 32; i++)
{
```

```
    sampleRateWithOtherThings += Convert.ToInt32(
        wavHeader[i]).ToString();
}
string blockAlign = "";
for (int i = 32; i < 34; i++)
{
    blockAlign += Convert.ToInt32(wavHeader[i]).
        ToString();
}
string bitsPerSample = "";
for (int i = 34; i < 36; i++)
{
    bitsPerSample += Convert.ToInt32(wavHeader[i]).
        ToString();
}
string dataSubChunk = "";
for (int i = 36; i < 40; i++)
{
    dataSubChunk += Convert.ToInt32(wavHeader[i]).
        ToString();
}
string sizeOfDataSection = "";
for (int i = 40; i < 42; i++)
{
    sizeOfDataSection += Convert.ToInt32(wavHeader[i]).
        ToString();
}
List<string> stringsList = new List<string>();
stringsList.Add($"Opis RIFF. (bity 0-3): {
    riffDescription}");
stringsList.Add($"Rozmiar pliku (bity 4- 7): {
    sizeOfFileBytes}");
stringsList.Add($"Opis WAV (bity 8- 11): {
    waveDescription}");
stringsList.Add($"Opis FMT (bity 12-15): {
    fmtDescription}");
stringsList.Add($"łDugosc formatu (bity 16-19): {
    lengthOfFormatData}");
stringsList.Add($"Typy formatu (bity 20-21): {
    typesOfFormatData}");
stringsList.Add($"Liczba łkanaow (bity 22-23): {
    numbersOfChannel}");
stringsList.Add($"Sample rate (bity 24-27): {
    sampleRate}");
stringsList.Add($"Byte rate (bity 28-31): {
    sampleRateWithOtherThings}");
stringsList.Add($"Wyrownanie bloku (bity 32-33): {
    blockAlign}");
stringsList.Add($"Bity na probke (bity 34-35): {
```

```
            bitsPerSample}");
        stringsList.Add($" 'data chunk', rozpoczyna sekcje
            danych w pliku .wav (bity 36-39): {dataSubChunk}")
            ;
        stringsList.Add($" Rozmiar sekcji danych (bity 40-41):
             {sizeOfDataSection}");
        string textboxString = "";
        foreach (string text in stringsList)
        {
            textboxString += text + System.Environment.NewLine
                ;
        }
        wavTextBox.Text = textboxString;
    }
}
// Przycisk uruchamiajacy odtwarzanie dzwieku za pomoca MCI
private void MciPlayBtn_Click(object sender, EventArgs e)
{
    if (!String.IsNullOrWhiteSpace(fileName))
    {
        SoundManager.MciPlay(fileName);
    }
}
// Przycisk zatrzymujacy odtwarzanie MCI
private void StopMCIBtn_Click(object sender, EventArgs e)
{
    if (!String.IsNullOrWhiteSpace(fileName))
    {
        SoundManager.MciStop();
    }
}

// Przycisk rozpoczynajacy nagrywanie za pomoca funkcji
    mciSendString
private void RecordButton_Click(object sender, EventArgs e)
{
    SoundManager.MciRecord(recordedFileName);
}

// Przycisk zatrzymujacy nagrywanie
private void StopRecordBtn_Click(object sender, EventArgs e)
{
    SoundManager.StopRecording();
}

// Przycisk zatrzymujacy odtwarzanie dzwieku za pomoca
    Windows Media Player
private void StopWMPBtn_Click(object sender, EventArgs e)
{
```

```
            SoundManager.StopWMP();
        }

        // Przycisk rozpoczynajacy odtwarzanie dzwieku za pomoca
            Windows Media Player
        private void StartWMPBtn_Click(object sender, EventArgs e)
        {
            SoundManager.StartWMP(fileName);
        }

        // Aktualizowanie wlasciwosci recordedFileName, kiedy zmieni
            sie tekst w textBoxie.
        private void TextBox1_TextChanged(object sender, EventArgs e)
        {
            recordedFileName = textBox1.Text;
        }
    }
}
```

3.3 Najważniejsze użyte funkcje

Najważniejsze użyte funkcje to funkcje dotyczące hardware'u, czyli większość funkcji zawartych w klasie SoundManager. Są tam owrapowane funkcje systemowe z bibliotek Windowsa, służące do obsługi karty dźwiękowej.
Kod klasy SoundManager:

```
  using System;
using System.Text;
using System.Runtime.InteropServices;
using WMPLib;
namespace UP1Official
{
    class SoundManager
    {
        // Funkcja playsound zimportowana z winmm.DLL
        [DllImport("winmm.DLL", EntryPoint = "PlaySound",
            SetLastError = true, CharSet = CharSet.Unicode,
            ThrowOnUnmappableChar = true)]
        public static extern bool PlaySound(string szSound, IntPtr
            hMod, PlaySoundFlags flags);

        // Funkcja mciSendString zimportowana z winmm.DLL
        [DllImport("winmm.dll")]

        private static extern long mciSendString(string command,
            StringBuilder returnValue, int returnLength, IntPtr
            winHandle);
        // Obiekt WindowsMediaPlayer do obslugiwania funkcjonalnosci
        // zwiazanymi z Windows Media Player
```

```
private static WindowsMediaPlayer mediaPlayer = null;
// Wlasciwosc, ktora kiedy jest false to oznacza ze
    mediaPlayer nie gra,
// a kiedy ma wartosc true to gra.
private static bool isPlayerPlaying = false;

// Wlasciwosci wspomagajace dzialanie funkcji
private static string MciFileName { get; set; }
public static string RecordFileName { get; set; }
//Struktura waveHDR przepisana z dokumentacji winmm.dll
public struct WaveHdr
{
    public IntPtr lpData;
    public int dwBufferLength;
    public int dwBytesRecorded;
    public IntPtr dwUser;
    public int dwFlags;
    public int dwLoops;
    public IntPtr lpNext;
    public IntPtr reserved;
}

// Flagi do funkcji PlaySound
[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 zatrzymujaca odtwarzanie WindowsMediaPlayer
public static void StopWMP()
{
    if (mediaPlayer != null)
    {
        mediaPlayer.controls.stop();
    }
    isPlayerPlaying = false;
}
//Funkcja rozpoczynajaca odtwarzanie za pomoca Windows Media
    Player
public static void StartWMP(string fileName)
{
    if (String.IsNullOrWhiteSpace(fileName))
```

```
    {
        if (isPlayerPlaying == true)
        {
            StopWMP();
            mediaPlayer.close();
        }
        mediaPlayer = new WindowsMediaPlayer
        {
            URL = fileName
        };
        mediaPlayer.controls.play();
        isPlayerPlaying = true;
    }
}
// Funkcja odtwarzajaca dzwiek za pomoca funkcji
    mciSendString
public static void MciPlay(string fileName)
{
    mciSendString("open \"" + fileName + "\" type mpegvideo
        alias MediaFile", null, 0, IntPtr.Zero);
    mciSendString("play " + fileName + " from 0", null, 0,
        IntPtr.Zero);
    MciFileName = fileName;
}

// Funkcja zatrzymujaca dzwiek puszczony za pomoca komendy
    mciSendString
public static void MciStop()
{
    if (!String.IsNullOrEmpty(MciFileName))
    {
        mciSendString("play " + MciFileName + " from 0", null,
             0, IntPtr.Zero);
        MciFileName = null;
    }
}

// Funkcja nagrywajaca mikrofon za pomoca mciSendString
public static void MciRecord(string fileName)
{
    mciSendString($"set {fileName} time format ms", null, 0,
        IntPtr.Zero);
    mciSendString($"open new Type waveaudio Alias {fileName}",
         null, 0, IntPtr.Zero);
    mciSendString($"record {fileName}", null, 0, IntPtr.Zero);

    RecordFileName = fileName;
}
// Funkcja zatrzymujaca nagrywanie
```

```
        public static void StopRecording()
        {
            if (!String.IsNullOrWhiteSpace(RecordFileName))
            {
                mciSendString($"save {RecordFileName} C:\\Users\\
                    oszust\\{RecordFileName}.wav", null, 0, IntPtr.
                    Zero);
                mciSendString($"close {RecordFileName}", null, 0,
                    IntPtr.Zero);
                RecordFileName = null;
            }
        }
    }
}
```

3.4 Używane przez nas funkcje związane z hardware'm

private static extern long mciSendString(string command, StringBuilder returnValue, int returnLength, IntPtr winHandle); - komenda mciSendString wysyła komendę do danego urządzenia MCI.
Argumenty:
command- polecenie komendy do wysłania, po jej treści rozpoznawane jest do jakiego urządzenia MCI należy wysłać komendę.
StringBuilder returnValue- wskaźnik do miejsca w pamięci, do której ma zostać wpisana wartość zwrotna z urządzenia MCI, jeżeli programiście nie zależy na wartości zwrotnej, to może zostać nullem.
returnLength- rozmiar wartości zwrotnej mierzony w ilości znaków (char).
IntPtr winHandle- Handle (referencja do obiektu, która używa API systemu operacyjnego) do okna, które ma zostać poinformowane o zakończeniu działania komendy wysłanej w sendString, wykorzystywane tylko wtedy, gdy używamy komendy ze słowem ńotify".
Funkcja zwraca 0, jeżeli jej wykonanie powiodło się, jeżeli zwróci inną wartość, oznacza to błąd.
Funkcję tą w naszym programie użyliśmy do odtwarzania oraz nagrywania dźwięku, nasze wykorzystanie funkcji było dosyć proste, ponieważ używaliśmy tylko pierwszego argumentu.
Do odtwarzania używaliśmy komend ópen"do otwarcia danego pliku oraz play <nazwa pliku> from 0, aby odtworzyć dany plik dźwiękowy od początku. W trakcie tworzenia aplikacji zauważyliśmy, że gdy identyczną komendę play wyślemy do urządzenia MCI w trakcie odtwarzania pliku dźwiękowego, to dźwięk jest przerywany, co zaprogramowaliśmy pod przyciskiem zatrzymującym odtwarzanie.
Do nagrywania użyliśmy komend set, open, record, save i close, które wrzucaliśmy jako string do funkcji mciSendString, tak jak w przypadku nagrywania. Dzięki komendzie set, można ustawić poszczególne opcje dotyczące urzadzenia

MCI, open otwiera plik, do którego następnie nagrywa się dźwięk za pomocą komendy record. W trakcie naciśnięcia przycisku stop, do narzędzia MCI wysyłane są komendy save, która zapisuje plik, w wybranej lokalizacji. Należy pamiętać, że jeżeli nie mamy dostępu do lokalizacji (np. chcemy ją zapisać w strzeżonym miejscu na dysku C: do którego dostęp ma tylko administrator) to plik nie zostanie zapisany a nasze dane zostaną stracone. Funkcja działała nam tylko wtedy, kiedy podawaliśmy bezwzględną lokalizację pliku do zapisania. Po zapisaniu zamykamy plik komendą close, co zwalnia zasoby. Gdybyśmy tego nie zrobili, niemożliwe byłoby ponowne użycie funkcji.
private static extern bool PlaySound(string szSound, IntPtr hMod, PlaySoundFlags flags); - prosta funkcja umożliwiająca odtwarzanie dźwięków, dzięki windowsowemu API.
string szSound- wskazanie na dźwięk jaki ma zostać zagrany, przekazuje się w stringu ścieżkę do pliku, który ma zostać odtworzony. IntPtr hmod- Wskaźnik do zasobu, który ma zostać wczytany, jeżeli nie ma ustawionej flagi SND_RESOURCE to musi mieć wartość null.
fdwSound- Flagi zmieniające to jak ma zostać zagrany dźwięk, w naszym przypadku używaliśmy flagi SND_ASYNC, aby program się nie zacinał po tym jak zaczynamy odtwarzać dźwięk.

4 Literatura

[1] https://docs.microsoft.com/en-us/windows/win32/multimedia/mci

[2] https://pl.wikipedia.org/wiki/DirectX

[3] https://pl.wikipedia.org/wiki/DirectSound

[4] https://docs.microsoft.com/pl-pl/windows/win32/multimedia/about-waveform-audio

[5] https://docs.microsoft.com/en-us/previous-versions/dd757161(v%3Dvs.85)

[6] https://stackoverflow.com/questions/5616988/what-is-a-handle

[7] https://docs.microsoft.com/en-us/previous-versions/dd743680(v%3Dvs.85)