Mateusz Kocur 171044

Obsługa karty dźwiękowej

21.12.2010, wtorek TN godz. 15^{15}

1 Zadania do wykonania.

Pierwszą częścią ćwiczenia była odtworzenie dźwięku z pliku typu *wave* za pomocą:

- funkcji PlaySound(),
- komend WaveOut*
- DirectX – DirectSound, wraz z dodatkowymi efektami.

Kolejnym zadaniem było nagranie dźwięku za pomocą komend WaveIn*.

2 Opis programu.

2.1 Funkcja *main()*.

Główna funkcja programu zawiera pętlę nieskończoną, w której wybierane są czynności do wykonania, obejmujące odtwarzanie i zatrzymywanie plików *wave* o wskazanej nazwie. Program oczekuje wyboru, za każdym razem po wyświetleniu pełnego menu przedstawiającego dostępne opcje.

Kombinacja klawiszy *CTRL+C* powoduje wyjście z pętli, zamknięcie portu i zakończenie programu.

Główna pętla w funkcji *main()* przedstawiona jest na listingu 1.

```
int _tmain(int argc, _TCHAR* argv[])
{
	char ch;
	char nazwa[100];
	//char * nazwa = {"a.wav\0"};

	while (true) {
		cout << " == DOSTEPNE FUNKCJE ==\n";
		cout << "\t1) Odtwarzaj plik o wskazanej nazwie (PlaySond())\n";
		cout << "\t2) Stop (PlaySond())\n";
		cout << "\t3) Odtwarzaj plik  o wskazanej nazwie (WaveOut())\n";
		cout << "\t4) Stop (WaveOut())\n";
		cout << "\t5) Pauza (WaveOut())\n";
		cout << "\t6) Odtwarzaj plik  o wskazanej nazwie (DirectSound)\n";
		cout << "\t7) Nagraj dzwiek o dlugosci 5 sekund (WaveIn())\n";
		cout << "\t8) Odtwarzaj nagrany dzwiek (WaveIn())\n";
		cout << "\tCTRL+C) Koniec\n";
		cout << "Wybor: ";
		//pobieranie znaku z klawiatury
```

```
		ch = getch();
		// ewentualne zamykanie programu kombinacja klawiszy CTRL+C
		if (ch == 3)
			break;

		switch (ch) {
		case '1':
			cout << "\nPodaj nazwe pliku (PlaySond()): ";
			cin >> nazwa;
			odtwarzaj((LPCSTR)nazwa);
			cout << endl;
			break;

		case '2':
			cout << "\nStop";
			stop();
			cout << endl << endl;
			break;

		case '3':
			cout << "\nPodaj nazwe pliku (WaveOut()): ";
			cin >> nazwa;
			wczytajWav((LPCSTR)nazwa);
			cout << endl;
			break;

		case '4':
			stopWav();
			cout << endl;
			break;

		case '5':
			pauzaWav();
			if (pauza == true)
				cout << "PAUZA";
			else
				cout << "RESUME";
			cout << endl << endl;
			break;

		case '6':
			cout << "\nPodaj nazwe pliku (DirectSound): ";
			cin >> nazwa;
			odtwarzajDirectSound((LPCSTR)nazwa);
			cout << endl;
			break;
		case '7':
			cout << "\nRozpoczeto nagrywanie - 5 sekund (WaveIn) ";
			nagrajWav();
			cout << endl;
			break;
		case '8':
			cout << "\nOdtwarzanie (WaveIn) ";
			odtworzWav();
			cout << endl;
			break;
		}
	}
	cout << "\nKoniec." << endl;
	return 0;
}
```

Listing 1.

2.2 Odtwarzanie za pomocą *PlaySound()*.

Wykorzystywanie funkcji PlaySound() jest najprostszą metodą na odtwarzanie dźwięków. Na zajęciach odtwarzanie dźwięków za pomocą tej funkcji wykonywane jest wewnątrz funkcji *odtwarzaj()* przedstawionej na listingu 2. Jej argumentem jest nazwa pliku. Wszystkie niezbędne do odtworzenia dźwięku czynności wykonywane są automatycznie.

Funkcja *stop()* przedstawiona jest na listingu 3. Umożliwia ona zatrzymanie odtwarzanego aktualnie dźwięku.

```
void odtwarzaj (LPCSTR nazwa) {
	PlaySound(nazwa, NULL, SND_FILENAME | SND_ASYNC);
}
```

Listing 2.

```
void stop () {
	PlaySound(NULL, NULL, SND_ASYNC);
}
```

Listing 3.

2.3 Odtwarzanie za pomocą *WaveOut**.

Odtwarzanie dźwięków za pomocą komend WaveOut* jest nieco bardziej skomplikowane niż to, opisane w punkcie 2.2 .

Pierwszą, niezbędną czynnością jest otwarcie pliku *wave* i wczytanie wszystkich informacji o zapisanym dźwięku, takich jak liczba kanałów, częstotliwość próbkowania, liczba bitów na próbkę itd. Podstawowe informacje o pliku wyświetlane są w konsoli podczas otwierania. Następnie tworzony jest bufor na próbki i kopiowane są do niego dane audio. Do odtworzenia dźwięku z bufora, należy jeszcze wywołać funkcję *waveOutPrepareHeader()*, przygotowującą blok danych do odtworzenia, obejmującą czasochłonne przetwarzanie bufora i nagłówka raz podczas inicjalizacji. Właściwe odtwarzanie wykonywane jest w momencie wywołania funkcji *waveOutWrite()*. Kod funkcji *wczytajWav()* przedstawiony jest na listingu 4.

Możliwe jest kilkukrotne wywołanie po sobie opisywanej funkcji, co skutkuje nałożeniem się odtwarzanych dźwięków, jednak na zajęciach zostało dodane zabezpieczenie w postaci dodatkowej zmiennej *gra*, która uniemożliwia wykonanie tego typu działania.

```
bool wczytajWav (LPCSTR nazwa) {
	if (gra == true) {
		cout << "BLAD: Dzwiek jest juz odtwarzany." << endl;
		return false;
	}
	pauza = false;

	FILE *plik;
	plik = fopen(nazwa, "rb");
	WAVEFORMATEX wav;
	if (plik) {
		BYTE id[5];
		id[4] = 0;
		DWORD size;
```

```
fread(id, sizeof(BYTE), 4, plik);
if (!strcmp((char *)id, "RIFF")) {
	cout << "ChunkID: " << (char*)id << endl;

	fread(&size, sizeof(DWORD), 1, plik);
	cout << "Chunk Size: " << size << endl;

	// przesuwamy sie na 20 bajt pliku
    fseek(plik,20,SEEK_SET);
	// czytamy format pliku
    fread(&(wav.wFormatTag), 2, 1, plik);
	cout << "Format pliku (kod): " << wav.wFormatTag << endl;
	// czytamy liczbe kanalow
	fread(&(wav.nChannels), 2, 1, plik);
	cout << "Liczba kanalow: " << wav.nChannels << endl;
	// czytamy liczbe probek na sekunde
	fread(&(wav.nSamplesPerSec), 4, 1, plik);
	cout << "Liczba prbek//s: " << wav.nSamplesPerSec << endl;
	// liczba bajtow na sekunde (srednio)
	fread(&(wav.nAvgBytesPerSec), 4, 1, plik);
	// liczba bajtow na probke dla obydwu kanalow
	fread(&(wav.nBlockAlign), 2, 1, plik);
	// liczba bitow na probke
	fread(&(wav.wBitsPerSample), 2, 1, plik);
	cout << "Liczba bitow//probke: " << wav.wBitsPerSample << endl;
	// rozmiar informacji dodatkowych o pliku
	wav.cbSize = 0;

	// otwieramy urzadzenie waveOut
	if(waveOutOpen(&hwaveout, WAVE_MAPPER, &wav, NULL, NULL,
			CALLBACK_NULL) != MMSYSERR_NOERROR) {
		fclose(plik);
		cout << "Blad otwierania waveOutOpen()" << endl;
		return false;
	}

	// 40 bajt pliku
	fseek(plik, 40, SEEK_SET);
	// dlugosc bloku danych
	fread(&(wavhdr.dwBufferLength), 4, 1, plik);

	// alokacja pamieci na bufor z danymi
	wavhdr.lpData = (LPSTR)malloc(wavhdr.dwBufferLength);
	if(wavhdr.lpData == NULL) {
		fclose(plik);
		cout << "Blad alokacji pamieci." << endl;
		return false;
	}

	// kopiowanie danych do bufora
	if(fread(wavhdr.lpData, wavhdr.dwBufferLength, 1, plik) != 1) {
		fclose(plik);
		cout << "Blad kopiowania danych do bufora." << endl;
		return false;
	}

	wavhdr.dwFlags = 0;
	wavhdr.dwLoops = 0;

	// przygtowania
	if(waveOutPrepareHeader(hwaveout,&wavhdr,sizeof(WAVEHDR)) !=
			MMSYSERR_NOERROR) {
		cout << "Blad przygotowywania do odtwarzania.";
		return false;
```

```
                        }

                        // zapisywanie do karty dzwiekowej
                        if(waveOutWrite(hwaveout,&wavhdr,sizeof(WAVEHDR)) !=
                                                        MMSYSERR_NOERROR) {
                                cout << "Blad zapisywania danch." << endl;
                                return false;
                        }

                        gra = true;
                } else
                        cout << "Format pliku inny niz RIFF." << endl;
                fclose (plik);
        } else
                cout << "Blad wczytywania pliku " << nazwa << endl;

        fclose(plik);
        return 0;
}
```

Listing 4.

Dodatkowo do odtwarzania za pomocą WaveOut* utworzone zostały procedury pauzowania i zatrzymywania odtwarzanego pliku. Przedstawione zostały na listingach 5 i 6.

```
void pauzaWav () {
        if (pauza == true) {
                pauza = false;
                waveOutRestart (hwaveout);
        } else {
                if (waveOutPause (hwaveout) != MMSYSERR_NOERROR)
                        cout << "Blad pauzowania." << endl;
                else
                        pauza = true;
        }
}
```

Listing 5.

```
void stop () {
        PlaySound(NULL, NULL, SND_ASYNC);
}
```

Listing 6.

2.4 Odtwarzanie za pomocą *DirectX - DirectSound*.

Możliwe jest również odtwarzanie dźwięków w sposób bardziej zaawansowany z dodatkowymi efektami (np. echo) za pomocą DirectSound, będącego częścią pakietu DirectX.

Aby odtworzyć dźwięk za pomocą DirectSound należy utworzyć interfejs IDirectSound za pomocą metody *DirectSoundCreate()*, następnie stworzyć obiekt DirectSoundBuffer za pomocą funkcji *CreateSoundBuffer()*. Od tego momentu można zapisywać dane do bufora wskazywanego przez wskaźnik zwrócony przez funkcję *Lock()* interfejsu IDirectSound. Po zakończeniu zapisywania danych do bufora i odblokowaniu go w identyczny sposób funkcją *Unlock()* możliwe jest odtworzenie dźwięku za pomocą funkcji *Play()* tego samego interfejsu. Niestety z nieznanych przyczyn wywołanie funkcji *Lock()* kończy się z niewyjaśnionych przyczyn błędem DSERR_PRIOLEVELNEEDED co uniemożliwia poprawne odtworzenie dźwięku.

Listing funkcji *odtwarzajDirectSound()* odtwarzającej dźwięk z pliku o nazwie podanej w parametrze za pomocą DirectSound przedstawiony jest na listingu 7.

```
bool odtwarzajDirectSound (LPCSTR nazwa) {
	HRESULT hr;
	CoInitialize(NULL);

	hr = DirectSoundCreate(NULL, &lpDSO, NULL);

	DSBUFFERDESC bd;
	LPDIRECTSOUNDBUFFER ppdsb;

	memset (&bd, 0, sizeof (DSBUFFERDESC));
	bd.dwSize = sizeof (DSBUFFERDESC);
	bd.dwFlags = DSBCAPS_PRIMARYBUFFER;
	bd.dwBufferBytes = 0;
	bd.lpwfxFormat = NULL;

	HANDLE bufor;
	WAVEFORMATEX * wav;

	lpDSO->SetCooperativeLevel(GetDesktopWindow(), DSSCL_PRIORITY);
	if(DS_OK != lpDSO->CreateSoundBuffer( &bd, &ppdsb, NULL ))
		cout << "Blad CreateSoundBuffer" << endl;

	LPVOID lpvWrite;
	DWORD  dwLength;
	LPVOID lpvWrite2;
	DWORD  dwLength2;

	int tmp;

	tmp = ppdsb->Lock(
		0,
		0,
		&lpvWrite,
		&dwLength,
		NULL,
		NULL,
		DSBLOCK_ENTIREBUFFER);

	if (tmp == DS_OK) {
		wav = wczytajPlik(nazwa, lpvWrite);
		dwLength = wavhdr.dwBufferLength;

		if (bufor == 0)
			cout << "Blad alokacji pamieci.22" << endl;
	} else if (tmp == DSERR_PRIOLEVELNEEDED)
		cout << tmp << endl;

	ppdsb->Unlock(lpvWrite, dwLength, NULL, 0);

	if (DS_OK == lpDSO->CreateSoundBuffer (&bd, &ppdsb, NULL)) {
		ppdsb->SetFormat (wav);
	}

	ppdsb->SetCurrentPosition (0);

	ppdsb->SetVolume(DSBVOLUME_MAX);

	if (ppdsb->Play(0, 0, DSBPLAY_LOOPING) != DS_OK) {
		cout << "BLAD: ppdsb->Play()" << endl;
	}
```

```
        return true;
}
```

Listing 7.

2.5 Nagrywanie za pomocą *WaveIn*.*

Nagrywanie za pomocą WaveIn* zostało zrealizowane tak jak przedstawiono na listingu 8. Na początku należy ustawić parametry nagrywania, takie jak liczba kanałów, częstotliwość próbkowania, rozmiar próbki i rozmiar nagrania (podawany w sekundach). Otwarcie urządzenia wejściowego do nagrywania realizowane jest za pomocą funkcji *waveInOpen()*. Do nagrania dźwięku należy jeszcze przygotować bufor na dane alokując na niego dane oraz wywołując funkcję *waveInPrepareHeader()*. Po wykonaniu opisanych czynności możliwe jest rozpoczęcie właściwego nagrywania funkcją *waveInStart()*. W przypadku niepowodzenia zaalokowana pamięć jest zwalniana.

```
void nagrajWav() {

        unsigned short  Channels         = 1;       //kanałów: 1-mono, 2-stereo
        unsigned long   SamplesPerSecond = 44100;   //probki na sekunde
        unsigned short  BitsPerSample    = 8;       //rozmar probki
        unsigned long   RecordSeconds    = 5;      //dlugosc nagrania

        WaveFormat.wFormatTag      = WAVE_FORMAT_PCM;
        WaveFormat.nChannels       = Channels;
        WaveFormat.nSamplesPerSec  = SamplesPerSecond;
        WaveFormat.wBitsPerSample  = BitsPerSample;
        WaveFormat.nAvgBytesPerSec = SamplesPerSecond * Channels;
        WaveFormat.nBlockAlign     = (Channels*BitsPerSample)/8;
        WaveFormat.cbSize          = 0;

        int Res = waveInOpen(&WaveHandle, WAVE_MAPPER, &WaveFormat, 0, 0,
                                                WAVE_FORMAT_QUERY);
        if (Res == WAVERR_BADFORMAT) return;

        //otwieramy urzadzenie
        Res = waveInOpen(&WaveHandle, WAVE_MAPPER, &WaveFormat, 0, 0, CALLBACK_WINDOW);

        BufferSize = RecordSeconds * (BitsPerSample / 8) * SamplesPerSecond * Channels;
        Buffer = new char [BufferSize];

        //przygotowania
        WaveHeader.dwBufferLength  = BufferSize;
        WaveHeader.dwFlags         = 0;
        WaveHeader.lpData          = Buffer;

        //przygotowujemy naglowek
        Res = waveInPrepareHeader(WaveHandle, &WaveHeader, sizeof(WAVEHDR));
        if(Res) {
                cout << "Blad przygotowania naglowka" << endl;
                if(Buffer)
                        delete Buffer;
                return;
        }

        Res = waveInAddBuffer(WaveHandle, &WaveHeader, sizeof(WAVEHDR));

        //zaczynamy nagrywanie dzwieku
        Res = waveInStart(WaveHandle);
        if(Res!=MMSYSERR_NOERROR){
                cout << "Blad nagrywania dzwieku" << endl;
                if(Buffer)
```

```
                delete Buffer;
            return;
        }
}
```

Listing 8.

Odtwarzanie nagranego dźwięku odbywa się za pomocą funkcji *odtworzWav()* przedstawionej na listingu 9. Odtwarzanie odbywa się za pomocą funkcji *waveOutWrite()*, podobnie jak opisano w punkcie 2.3 z pominięciem fragmentu wczytującego plik wave, ponieważ dane do odtworzenia są już w buforze.

```
void odtworzWav() {
            int Res = waveOutOpen(&WaveOUTHandle, WAVE_MAPPER, &WaveFormat, 0, 0,
                                                    WAVE_FORMAT_QUERY);
            if (Res == WAVERR_BADFORMAT)
                return;

            //otwieramy urzadzenie
            Res = waveOutOpen(&WaveOUTHandle, WAVE_MAPPER, &WaveFormat, 0, 0,
                                                        CALLBACK_WINDOW);
            waveOutPrepareHeader(WaveOUTHandle,&WaveHeader,sizeof(WAVEHDR));
            waveOutWrite(WaveOUTHandle,&WaveHeader,sizeof(WAVEHDR));
}
```

Listing 9.

3 Wnioski.

W systemie Windows dostępnych jest wiele różnych możliwości odtwarzania i nagrywania dźwięku. Wszystkie różnią się oferowanymi możliwościami, jak i stopniem trudności użycia.

Przy odtwarzaniu należy zwrócić uwagę na odpowiednie ustawienie parametrów odtwarzania takich jak częstotliwość próbkowania, liczba bitów na próbkę czy liczba kanałów.