Jakub Batogowski

Grzegorz Oliwa

2016-10-27

Urządzenia peryferyjne

*Obsługa joysticka USB z wykorzystaniem DirectInput*

Zapoznanie się z obsługą urządzeń peryferyjnych w systemie Windows. Obeznanie się z DirectX. Nauka pisania programu obsługującego joystick podłączany za pomocą portu USB. Wyciągnięcie wniosków z odbytego laboratorium.

Urządzenia peryferyjne

Obsługa joysticka USB z wykorzystaniem DirectInput

# **Wstęp:**

Na pierwszych zajęciach laboratoryjnych otrzymaliśmy zadanie polegające na napisaniu programu pozwalającego na prostą obsługę joysticka podłączonego do komputera za pomocą portu USB. Wymagane było, aby program:

* Stwierdzał obecność joysticka podłączonego do komputera
* Umożliwiał używanie joysticka jako myszki
* Przedstawiał obecny stan joysticka (wciśnięte klawisze, koordynaty osi)

Kod naszego programu został napisany w języku C++. Dodatkowo został on w wyczerpujący sposób skomentowany w celu lepszego zrozumienia jego działania.

# **Zagadnienia:**

## HID (Human Interface Device):

Są to urządzenia podłączane do komputera, które wysyłają do niego informacje generowane przez człowieka np. mysz, klawiatura, joystick. Urządzenia HID obecnie produkowane są z myślą, aby móc je podłączać za pomocą portu USB.

## Port USB:

Uniwersalna magistrala szeregowa (Universal Serial Bus) to powszechny port umożliwiający komunikację pomiędzy komputerem, a dowolnym urządzeniem peryferyjnym. Opracowany został między innymi przez Microsoft, IBM i Intel. Urządzenia podłączone za pomocą portu USB mogą być automatycznie wykryte i rozpoznane przez system.

## Szeregowa transmisja danych:

Jeden z rodzajów cyfrowej transmisji danych. Wyróżnia się tym, iż bity informacji przesyłane są kolejno jeden po drugim wraz z danymi pozwalającymi kontrolować transmisję.

## COM (Component Object Model):

Jest to standard definiowania i tworzenia interfejsów opracowany przez firmę Microsoft oraz w oparciu o zestaw adekwatnych bibliotek. Na podstawie standardu COM zostało opracowanych dużo niskopoziomowych API dla produktów firmy Microsoft – między innymi DirectX.

## Biblioteka DirectInput:

Powyższa biblioteka umożliwia obsługę urządzeń wejściowych takich jak mysz, klawiatura czy joystick. Wchodzi ona w skład zestawu funkcji API DirectX firmy Microsoft. Za jej pomocą możliwe jest odczytywanie informacji o podłączanych urządzeniach oraz przypisywanie im określonych akcji (przykładowo po wciśnięciu określonego przycisku). Biblioteka ta najczęściej stosowana jest podczas tworzenia gier komputerowych oraz aplikacji interaktywnych.

# **Kod programu:**

#include <windows.h>

#include <stdio.h>

#include "dinput.h"

#include <iostream>

#include <iomanip>

#include <conio.h>

using namespace std;

/\*================================================================\*/

// Informacja dla konsolidatora mówiąca o tym jakie biblioteki

// ma on przeszukać, aby poniższy program działał prawidłowo

//

#pragma comment(lib, "dinput8.lib")

#pragma comment(lib, "dxguid.lib")

/\*================================================================\*/

// Interfejsy komunikujące się z urządzeniem; niezbedne do:

// - inicjalizacji urządzenia

// - pobierania statusu urządzenia

//

LPDIRECTINPUT8 directInput;

LPDIRECTINPUTDEVICE8 joystick;

/\*================================================================\*/

// Funkcja z wywołaniem zwrotnym posiadająca następujące argumenty:

// - identyfikator urządzenia

// - wskaźnik na raw data

//

BOOL CALLBACK enumCallback(const DIDEVICEINSTANCE\* instance, VOID\* context)

{

// Enumerator przechowujący informacje o błędach (stanach)

HRESULT hresult;

// Tworzenie i inicjalizacja urządzenia

hresult = directInput -> CreateDevice(instance -> guidInstance, &joystick, NULL);

// Nie udało się pobrać joystick'a

if (FAILED(hresult)) return 1;

// Udało się pobrać joystick - SUKCES

return 0;

}

/\*================================================================\*/

// Funkcja z wywołaniem zwrotnym posiadająca następujące argumenty:

// - identyfikator urządzenia

// - wskaźnik na raw data

// (Współpracuje z enumCallback(const DIDEVICEINSTANCE\*, VOID\*))

//

BOOL CALLBACK enumAxesCallback

(const DIDEVICEOBJECTINSTANCE\* instance, VOID\* context)

{

// Rzutowanie raw data na obiekt struktury HWND

HWND hDlg = (HWND)context;

// Tworzenie obiektu przechowującego właściwości urządzenia

DIPROPRANGE propRange;

// Parametrów joysticka

propRange.diph.dwSize = sizeof(DIPROPRANGE);

propRange.diph.dwHeaderSize = sizeof(DIPROPHEADER);

propRange.diph.dwHow = DIPH\_BYID;

propRange.diph.dwObj = instance -> dwType;

propRange.lMin = -500;

propRange.lMax = +500;

// Nie udało się nadpisac ustawień joysticka

if (FAILED(joystick -> SetProperty(DIPROP\_RANGE, &propRange.diph))) return 0;

// Udalo sie nadpisac ustawienia joysticka - SUKCES

return 1;

}

/\*================================================================\*/

// Funkcja przyjmująca jako argument aktualny stan joysticka

//

HRESULT poll(DIJOYSTATE \*js)

{

// Enumerator przechowujacy informacje o błędach (stanach)

HRESULT hr;

// Odczyt eventow joysticka (czy mozemy go dalej używac)

hr = joystick -> Poll();

// Jeżeli makro napotka jeden z następujących błędów

if (FAILED(hr))

{

// Przywrócenie połączenia jeśli strumien input został przerwany

hr = joystick -> Acquire();

// Próba odnowienia polaczenia w przypadku utraty

while (hr == DIERR\_INPUTLOST) hr = joystick -> Acquire();

// Fatal error (drezyny mogą nie przyjechac)

if ((hr == DIERR\_INVALIDPARAM) || (hr == DIERR\_NOTINITIALIZED)) return E\_FAIL;

// Jeśli inna aplikacja ma kontrole nad joystickiem - polączenie zostanie przywrocone gdy joystick sie zwolni

if (hr == DIERR\_OTHERAPPHASPRIO) return S\_OK;

}

// Odczyt stanu urządzenia

if (FAILED(hr = joystick -> GetDeviceState(sizeof(DIJOYSTATE), js))) return hr;

return S\_OK;

}

/\*================================================================\*/

// Główna funkcja programu

//

int main() {

// Zmienna przechowująca informacje na temat stanu podłączonego urządzenia

HRESULT hr;

// Inicjalizacja urzadzenia DirectInput nie powiodla się

if (FAILED(hr = DirectInput8Create(GetModuleHandle(NULL), DIRECTINPUT\_VERSION, IID\_IDirectInput8, (VOID\*\*)&directInput, NULL)))

return 1;

// Nie wykryto joysticka

if (FAILED(hr = directInput->EnumDevices(DI8DEVCLASS\_GAMECTRL, enumCallback, NULL, DIEDFL\_ATTACHEDONLY)))

return 1;

if (joystick == NULL)

{

cout << "Glupcze! Podlacz joysick!\n";

// Wyświetlenie komunikatu o brutalnym odłączeniu joysticka

\_getch();

return E\_FAIL;

}

else cout << "Joystick jest obecny :D\n";

// Wyświetlenie komunikatu o poprawnym wykryciu joysticka

// Struktura przechowujaca zdolnosci / możliwości

(ang. capabilities) urządzenia DirectInput

DIDEVCAPS capabilities;

// Probujemy stwierdzić czy naszym urządzeniem DirectInput jest joystick

if (FAILED(hr = joystick -> SetDataFormat(&c\_dfDIJoystick)))

return 1;

// Probujemy pobrac capabilities joysticka

capabilities.dwSize = sizeof(DIDEVCAPS);

if (FAILED(hr = joystick -> GetCapabilities(&capabilities)))

return 1;

// Probujemy pobrac informacje o osiach joysticka

if (FAILED(hr = joystick -> EnumObjects(enumAxesCallback, NULL, DIDFT\_AXIS)))

return 1;

// Pomocnicza struktura przechowujaca informacje o aktualnym stanie joysticka

DIJOYSTATE state;

// Struktura przechowujaca informacje o pozycji kursora

POINT cursor;

int wcisnietyPrzycisk = -1;

LONG os\_x = 0, os\_y = 0, os\_z = 0;

// Oczekiwanie na dzialanie

cout << "Do dziela!\n";

while (!poll(&state)) {

// Pętla wykonująca się tyle razy ile joystick posiada przycisków

for (int i = 0; i < 11; i++)

{

if (state.rgbButtons[wcisnietyPrzycisk] == 0)

{

// Odkliknięcie lewego przycisku myszy

if (wcisnietyPrzycisk == 0)

{

GetCursorPos(&cursor);

mouse\_event(MOUSEEVENTF\_LEFTUP,

cursor.x, cursor.y, 0, 0);

}

wcisnietyPrzycisk = -1;

}

if (state.rgbButtons[i] && wcisnietyPrzycisk != i)

{

cout << "Klawisz " << i + 1 << " zostal wcisniety!" << endl;

wcisnietyPrzycisk = i;

// Wykrycie wciśnięcia pierwszego przycisku joysticka; przypisanie do

niego lewego przycisku myszy

if (i == 0)

{

GetCursorPos(&cursor);

mouse\_event(MOUSEEVENTF\_LEFTDOWN, cursor.x, cursor.y, 0, 0); }

// Wykrycie wciśnięcia drugiego przycisku joysticka; przypisanie do

niego prawego przycisku myszy

else if (i == 1)

{

GetCursorPos(&cursor);

mouse\_event(MOUSEEVENTF\_RIGHTUP, cursor.x, cursor.y, 0, 0);

}

}

}

if (os\_x != state.lX || os\_y != state.lY || os\_z != state.lZ)

{

// Wyświetlenie informacji o aktualnych osiach joysticka

cout << "X: " << setw(5) << state.lX << " | ";

cout << "Y: " << setw(5) << state.lY << " | ";

cout << "Z: " << setw(5) << state.lZ << endl;

// Aktualizacja osi joysticka

os\_x = state.lX;

os\_y = state.lY;

os\_z = state.lZ;

}

// Aktualizacja pozycji kursora

GetCursorPos(&cursor);

SetCursorPos(cursor.x + state.lX / 100, cursor.y + state.lY / 100);

// Interwał czasowy pomiędzy kolejnymi sprawdzeniami stanu joysticka

Sleep(10);

}

if (joystick)

{

joystick -> Unacquire();

cout << "\nDzojstik jest nieosiagalny!\nNo i po co ruszales przewod?\n";

}

\_getch();

return 0;

}

# **Podsumowanie:**

Podczas pierwszych zajęć laboratoryjnych odświeżyliśmy naszą wiedzę na temat portów szeregowych oraz o tym jak one działają. Zapoznaliśmy się również z podstawami DirectX oraz o tym w jaki komputer rozpoznaje urządzenia, które są do niego podłączane za pomocą portu USB. Napisanie kodu programu oraz analiza przykładów zawartych w różnych źródłach pozwoliła nam lepiej zrozumieć sposób komunikacji pomiędzy komputerem a urządzeniami peryferyjnymi. Wiedza zdobyta na tym laboratorium pozwoliła nam również poznać nowe możliwości języka C++ co niewątpliwie poszerzyło nasze umiejętności programistyczne.