Łukasz Przywarty 171018 Data: 20.12.2010 r.

Grupa: WT/N 15:00

Urządzenia peryferyjne

Laboratorium XIV

Bluetooth – komunikacja z telefonem komórkowym

Prowadzący:

1. Zadania do wykonania

Celem ćwiczenia było napisanie programu realizującego komunikację między konkretnym adapterem Bluetooth a urządzeniem korzystającym z tej samej technologii. W ramach realizacji ćwiczenia należało zaimplementować metody wykrywające adaptery Bluetooth podłączone do komputera, szukające w pobliżu aktywnych urządzeń, autoryzujące połączenie między komputerem a wybranym urządzeniem oraz przesyłające pliki do urządzenia Bluetooth.

2. Wstęp teoretyczny

Technologia Bluetooth jest technologią komunikacji krótkiego zasięgu pomiędzy różnymi urządzeniami elektronicznymi takimi jak komputer, telefon komórkowy czy inne urządzenia peryferyjne. Każde urządzenie komunikujące się poprzez Bluetooth posiada specjalny adapter umożliwiający bezprzewodowe porozumiewanie się. Technologia oparta jest na łączu radiowym – działa w paśmie 2,4GHz, wykorzystuje modulację FSK i zapewnia przepustowość do 5MB/s.(Bluetooth 3.1 + High Speed). Urządzenia Bluetooth charakteryzują się stosunkowo małym poborem mocy i ograniczonym zasięgiem (odpowiednio 100m – klasa 1, 10m – klasa II, lub 1m – klasa III - w otwartej przestrzeni).

W projekcie wykorzystywaliśmy funkcje znajdujące się w plikach nagłówkowych:

- `winsock2.h` – biblioteka zawierająca funkcje umożliwiające tworzenie zaawansowanych połączeń sieciowych bez względu na używany protokół sieciowy i warstwę, w której działa aplikacja.
- `BluetoothAPIs.h` – Bluetooth API, zawiera między innymi funkcje zarządzające urządze-niami Bluetooth, wykrywające adaptery oraz autoryzujące urządzenia.
- `ws2bth.h` – rozszerzenia Winsock dla Bluetooth

3. Opis programu

W ramach wykonywania ćwiczenia napisaliśmy jeden program konsolowy realizujący wszystkie zagadnienia wymienione w punkcie 1 niniejszego sprawozdania. Aby uzyskać większą czytelność poniżej zamieścimy tylko najważniejsze fragmenty kodu.

3.1 Wykrywanie adapterów

Pierwszą rzeczą, którą chcieliśmy oprogramować było wykrywanie aktywnych adapterów Bluetooth. Tworzymy więc tablicę uchwytów adapters[MAX_BT_ADAPTERS], gdzie MAX_BT_ADAPTERS jest stałą określającą ile adapterów możemy maksymalnie obsługiwać oraz strukturę BLUETOOTH_FIND_RADIO_PARAMS pozwalającą numerować poszczególne, zainstalowane adaptery Bluetooth. Wykryte urządzenie przechowuje uchwyt `bt_radio_find`. Jeśli jest ona równa NULL to komputer nie znalazł żadnych podłączonych adapterów. Korzystamy z funkcji `BluetoothFindFirstRadio`, która jak sama nazwa wskazuje szuka pierwszego aktywnego urządzenia Bluetooth:

Listing 1: Szukanie pierwszego adaptera Bluetooth

```
BLUETOOTH_FIND_RADIO_PARAMS bt_find_radio_params = {
	sizeof(BLUETOOTH_FIND_RADIO_PARAMS)
};

//uchwyty przechowujące znalezione adaptery i urządzenia
HBLUETOOTH_RADIO_FIND bt_radio_find = NULL;
HBLUETOOTH_DEVICE_FIND bt_dev_find = NULL;

cout << "Szukanie adapterow BT:" << endl;

bt_radio_find = BluetoothFindFirstRadio(&bt_find_radio_params, &adapters[0]);
```

Jeżeli wynik działania funkcji `BluetoothFindFirstRadio` będzie różny od NULL prowadzimy dalsze poszukiwania podłączonych adapterów (jeśli takie istnieją). W tym celu korzystamy z funkcji `BluetoothFindNextRadio`. Każdy znaleziony adapter dodajemy do tablicy `adapters`.

Listing 2: Wykrywanie pozostałych adapterów

```
if (bt_radio_find == NULL) {
	cout << "Brak dostepnych adapterow. Kod bledu: " << GetLastError() << endl;
} else {
	int adapters_found = 1;
	while (BluetoothFindNextRadio(&bt_radio_find, &adapters[adapters_found])) {
		adapters_found++;
		if (adapters_found == MAX_BT_ADAPTERS-1) {
			cout << "Podlaczonych jest wiecej niz 10 adapterow BT do komputera.";
			break;
		}
	}
	for (int i = 0; i < adapters_found; i++)
```

```
            print_adapter_info(adapters[i], i);

      if(!BluetoothFindRadioClose(bt_radio_find))
            cout << "Blad zamykania \"BluetoothFindRadioClose(bt_radio_find)\"." <<
endl;
}
```

Listing 2 zawiera również obsługę błędów a także pętlę drukującą informacje o dostępnych adapterach poprzez wywoływanie metody `print_adapter_info`. Ciało tej funkcji przedstawia Listing 3. Kolejny raz używamy gotowej funkcji, tym razem `BluetoothGetRadioInfo` pozwalającej uzyskać informacje na temat konkretnego adaptera Bluetooth.

Listing 3: Drukowanie informacji o adapterze

```
void print_adapter_info (HANDLE BThandle, int i) {
      BluetoothGetRadioInfo(BThandle, &bt_radio_info);
      wprintf(L"Adapter %d ) ", i);
      wprintf(L"%s ", bt_radio_info.szName);
      wprintf(L"MAC: %02x:%02x:%02x:%02x:%02x:%02x ", bt_radio_info.address.rgBytes[1],
                  bt_radio_info.address.rgBytes[0], bt_radio_info.address.rgBytes[2],
                  bt_radio_info.address.rgBytes[3], bt_radio_info.address.rgBytes[4],
                  bt_radio_info.address.rgBytes[5]);
      wprintf(L"ClassofDevice: 0x%08x ", bt_radio_info.ulClassofDevice);
      wprintf(L"Manufacturer: 0x%04x\n", bt_radio_info.manufacturer);
}
```

Użytkownik otrzymał informacje na temat adapterów, może więc zdecydować, z którego chce korzystać:

Listing 4: Wybór adaptera

```
int wybrany_adapter = 0;
cout << "Wybierz adapter: ";
cin >> wybrany_adapter;
```

3.2 Wykrywanie urządzeń

Za pośrednictwem wybranego adaptera możemy rozpocząć wyszukiwanie urządzeń Bluetooth znajdujących się w pobliżu. Wyszukiwanie urządzeń odbywa się z uwzględnieniem parametrów szukania. Parametry te przechowuje struktura `BLUETOOTH_DEVICE_SEARCH_PARAMS`:

Listing 5: Struktura BLUETOOTH_DEVICE_SEARCH_PARAMS

```
BLUETOOTH_DEVICE_SEARCH_PARAMS bt_dev_search_params = {
	sizeof(BLUETOOTH_DEVICE_SEARCH_PARAMS), //rozmiar struktury
	1, //bool: czy zwracać urządzenia autentykowane
	0, //bool: czy zwracać urządzenia zapamiętane
	1, //bool: czy zwracać urządzenia nieznane
	1, //bool: czy zwracać urządzenia połączone
	1, //bool: czy zwracać kolejne urządzenie
	3, //czas szukania urządzeń - wielokrotność 1.28 sekundy
	NULL
};
```

W tym momencie możemy przejść do właściwej części programu odpowiedzialnej za wyszukiwanie aktywnych urządzeń Bluetooth. Postępujemy podobnie jak w przypadku szukania adapterów. Najpierw wykrywamy pierwsze urządzenie, a gdy takowe się pojawi dalej skanujemy otoczenie w celu odnalezienia kolejnych urządzeń. Informacje o urządzeniach będą przechowywane w tablicy struktur BLUETOOTH_DEVICE_INFO. Użytkownik może dokonać wyboru – tym razem urządzenia, z którym będzie chciał uzyskać połączenie. Wybór ułatwiają informację o danym urządzeniu wyświetlone na ekranie za pośrednictwem funkcji show_device_info, która jest bardzo podobna do wcześniej wymienionej metodą show_adapters_info. Funkcję show_device_info prezentuje Listing 7.

Listing 6: Wykrywanie urządzeń

```
bt_dev_search_params.hRadio = adapters[wybrany_adapter];

cout << endl << "Wyszukiwanie urzadzen BT w poblizu:" << endl;

BLUETOOTH_DEVICE_INFO devices[MAX_BT_DEVICES];
devices[0].dwSize = sizeof(BLUETOOTH_DEVICE_INFO);
bt_dev_find = BluetoothFindFirstDevice(&bt_dev_search_params, &devices[0]);

if (bt_dev_find = NULL) {
	cout << "Nie znaleziono zadnych urzadzen BT." << endl;
	return 0;
} else {
	int devices_found = 1;
```

```
	devices[devices_found].dwSize = sizeof(BLUETOOTH_DEVICE_INFO);
	while(BluetoothFindNextDevice(bt_dev_find, &devices[devices_found])) {
		devices_found++;
		devices[devices_found].dwSize = sizeof(BLUETOOTH_DEVICE_INFO);
	}
	BluetoothFindDeviceClose(bt_dev_find);

	for (int i = 0; i < devices_found; i++)
		print_device_info(devices[i], i);
}

int wybrane_urzadzenie = 0;
cout << "Wybierz urzadzenie:";
cin >> wybrane_urzadzenie;
cout << endl;
```

Listing 7: Drukowanie informacji o urządzeniu Bluetooth

```
void print_device_info (BLUETOOTH_DEVICE_INFO BThandle, int i) {
	wprintf(L"Device %d ) ", i);
	wprintf(L" Name: %s ", BThandle.szName);
	wprintf(L" MAC: %02x:%02x:%02x:%02x:%02x:%02x ", BThandle.Address.rgBytes[1],
				Bthandle.Address.rgBytes[0],
BThandle.Address.rgBytes[2],				Bthandle.Address.rgBytes[3],
BThandle.Address.rgBytes[4],				Bthandle.Address.rgBytes[5]);
	wprintf(L" ClassofDevice: 0x%08x ", BThandle.ulClassofDevice);
	wprintf(L" Connected:%s ", BThandle.fConnected ? L"T" : L"N");
	wprintf(L" Authenticated:%s ", BThandle.fAuthenticated ? L"T" : L"N");
	wprintf(L" Remembered:%s\n", BThandle.fRemembered ? L"T" : L"N");
}
```

3.3 Autoryzacja urządzenia

Implementacja autoryzacji okazała się bardzo łatwa. Korzystamy z gotowej funkcji Bluetooth API: BluetoothAuthenticateDeviceEx, która za parametry przyjmuje między innymi: uchwyt konkretnego adaptera, wskaźnik na urządzenie oraz rodzaj autentykacji. Za każdym razem gdy wywołujemy połączenie adaptera z urządzeniem musimy podać czterocyfrowy kod autoryzacji. Połączenie zakończy się sukcesem gdy kod wpisany w urządzeniu będzie się zgadzał z kodem zadeklarowanym na komputerze.

Listing 8: Autoryzacja urządzenia Bluetooth

```
BluetoothAuthenticateDeviceEx(NULL, adapters[wybrany_adapter],
&devices[wybrane_urzadzenie],                    NULL, MITMProtectionRequired);
```

3.4 Przesyłanie pliku

Aby przesłać plik do urządzenia najpierw musimy się z nim połączyć. Operację łączenia poprzedza utworzenie nowego gniazda Winsock przystosowanego do komunikacji w technologii Bluetooth. W tym celu po pomyślnym utworzeniu gniazda inicjalizujemy strukturę SOCKADDR_BTH. Wpisujemy w nią m.in. adres urządzenia Bluetooth oraz numer portu komunikacyjnego. Następnie wykonujemy próbę połączenia wywołując funkcję connect.

Lisitng 9: Łączenie z urządzeniem Bluetooth

```
WSADATA wsaData;
int result = WSAStartup(MAKEWORD(2,2), &wsaData);
if (result != NO_ERROR) {
	cout << "Blad WSAStartup: " << result;
	return 0;
}

sck = socket(AF_BTH, SOCK_STREAM, BTHPROTO_RFCOMM);

if (sck == INVALID_SOCKET) {
	cout << "Blad przy tworzeniu gniazda. Kod bledu: " << GetLastError() << endl;
	return 0;
} else {
	sadr.addressFamily = AF_BTH;
	sadr.btAddr = devices[wybrane_urzadzenie].Address.ullLong;
	sadr.port = (ULONG)BT_PORT_ANY;
	sadr.serviceClassId = OBEXObjectPushServiceClass_UUID;

	if (0 != connect (sck, (SOCKADDR *) &sadr, sizeof(sadr))) {
		cout << "Blad laczeniu. Kod bledu: " << GetLastError() << endl;
		closesocket( sck );
		WSACleanup();
		return 0;
	}
closesocket(sck);
}
```

Dane można wysyłać na przykład korzystając z protokołu OBEX. Niestety wysyłanie plików nie działało do końca poprawnie, dlatego też pominiemy fragmenty kodu zawierające kompletną implementację. Całość kodu jest dostępna na załączonym do sprawozdania

nośniku.

4. Podsumowanie i wnioski

Realizacja zajęć laboratoryjnych pozwoliła nam zapoznać się z możliwościami pisania programów zapewniających komunikację w technologii Bluetooth między komputerem a odbiornikiem - w naszym przypadku telefonem komórkowym. Korzystanie z gotowych plików nagłówkowych znacznie ułatwiało osiągnięcie niektórych celów, jednak najwięcej problemów sprawiało poprawne wysyłanie plików protokołem OBEX. Wymaga to od dewelopera oprogramowania wszystkich etapów przesyłania pliku – poczynając od sprawdzenia rozmiaru i zawartości pliku a kończąc na przesyłaniu kolejnych pakietów, w tym odpowiednio dobranych paczek z żądaniami `connect` czy `disconnect`.