#include <iostream>

#include "Winsock2.h"

Этот код содержит директивы препроцессора для включения заголовочных файлов "iostream" и "Winsock2.h".

Заголовочный файл "iostream" содержит определения для стандартных потоков ввода-вывода (std::cout, std::cin, std::cerr, и т.д.), а также другие вспомогательные функции для работы с потоками ввода-вывода.

Заголовочный файл "Winsock2.h" содержит определения функций, структур и констант, используемых для работы с сокетами в операционных системах семейства Windows. Этот заголовочный файл является необходимым при написании сетевых приложений под Windows, так как он предоставляет необходимые средства для создания, настройки и использования сокетов в Windows.

#pragma comment(lib, "WS2\_32.lib")

Директива pragma указывает компилятору подключить библиотеку WS2\_32.lib. Эта библиотека содержит функции для работы с сокетами в Windows, такие как socket, bind, connect, send, recv, и т.д.

#pragma warning(disable : 4996)

#pragma warning(disable : 4996) - Директива pragma указывает компилятору отключить предупреждение о устаревании функций в стандарте C/C++. В данном случае это предупреждение о том, что функции, такие как inet\_addr, inet\_ntoa, gethostbyname, являются устаревшими и могут быть заменены на более новые функции, такие как inet\_pton и getaddrinfo.

using namespace std;

string GetErrorMsgText(int code);

string SetErrorMsgText(string msgText, int code);

SOCKET clientSocket;

bool GetServer(char\* call, short port, sockaddr\* from, int\* flen)

{

memset(from, 0, sizeof(flen));

устанавливает блок памяти по указанному указателю from нулями. Размер блока определяется по значению, находящемуся по адресу flen.

if ((clientSocket = socket(AF\_INET, SOCK\_DGRAM, NULL)) == INVALID\_SOCKET)

{

throw SetErrorMsgText("socket:", WSAGetLastError());

}

создаёт дейтаграммный (UDP) сокет, используя протокол семейства интернетов (AF\_INET). Если создание сокета прошло неудачно, то генерируется исключение.

int optval = 1;

определяет значение опции для сокета. Значение 1 означает, что опция включена.

if (setsockopt(clientSocket,

SOL\_SOCKET, //уровень д-вия режима

SO\_BROADCAST, //режим сокета для исп-ния широковещ.адреса

(char\*)&optval, //значение режима сокета

sizeof(int)) == SOCKET\_ERROR) //длина буфера оптвал

{

throw SetErrorMsgText("opt:", WSAGetLastError());

}

устанавливает опции сокета. В данном случае, опция SO\_BROADCAST позволяет использовать широковещательный адрес. Если установка опций завершилась неудачно, то генерируется исключение.

SOCKADDR\_IN all;

all.sin\_family = AF\_INET;

all.sin\_port = htons(2000);

all.sin\_addr.s\_addr = INADDR\_BROADCAST;

структура, содержащая информацию о широковещательном адресе. all.sin\_family = AF\_INET - указывает семейство адресов (AF\_INET - IPv4). all.sin\_port = htons(2000) - указывает порт широковещательной передачи, который здесь равен 2000. all.sin\_addr.s\_addr = INADDR\_BROADCAST - определяет IP-адрес, равный широковещательному адресу.

if ((sendto(clientSocket, call, strlen(call) + 1, NULL, (sockaddr\*)&all, sizeof(all))) == SOCKET\_ERROR)

{

throw SetErrorMsgText("sendto:", WSAGetLastError());

}

отправляет данные на указанный адрес. Здесь функция отправляет сообщение call на адрес all. Если отправка не удалась, то генерируется исключение.

char nameServer[50];

создаёт массив символов nameServer размером 50 байт

if ((recvfrom(clientSocket, nameServer, sizeof(nameServer), NULL, from, flen)) == SOCKET\_ERROR)

{

if (WSAGetLastError() == WSAETIMEDOUT)

{

return false;

}

else

{

throw SetErrorMsgText("recv:", WSAGetLastError());

}

}

принимает данные из указанного сокета и сохраняет их в массив nameServer. Здесь from - указатель на структуру, которая будет использована для хранения адреса отправителя. flen - указатель на переменную, которая будет использована для хранения размера структуры отправителя. Если приём не удался, то генерируется исключение.

SOCKADDR\_IN\* addr = (SOCKADDR\_IN\*)&from;

приводит указатель на структуру from к типу SOCKADDR\_IN, чтобы получить доступ к порту и IP-адресу

std::cout << std::endl << "Порт сервера: " << addr->sin\_port;

std::cout << std::endl << "IP-адрес сервера: " << inet\_ntoa(addr->sin\_addr);

if (!strcmp(nameServer, call))

{

std::cout << std::endl << "Сервер с таким именем найден.";

return true;

}

else

{

std::cout << std::endl << "Сервер с таким именем не найден.";

return false;

}

return true;

}

int main() {

setlocale(LC\_ALL, "rus");

try {

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0)

{

throw SetErrorMsgText("Startup:", WSAGetLastError());

}

char call[] = "Hello";

SOCKADDR\_IN clnt;

int lc = sizeof(clnt);

GetServer(call, 2000, (sockaddr\*)&clnt, &lc);

if (closesocket(clientSocket) == SOCKET\_ERROR)

{

throw SetErrorMsgText("closesocket:", WSAGetLastError());

}

if (WSACleanup() == SOCKET\_ERROR)

{

throw SetErrorMsgText("Cleanup:", WSAGetLastError());

}

}

catch (string errorMsgText) {

{ cout << endl << "WSAGetLastError: " << errorMsgText; }

}

}