#include <iostream>

#include "Winsock2.h" // Заголовок WS2\_32.dll

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

#pragma warning(disable : 4996)

using namespace std;

string GetErrorMsgText(int code);

string SetErrorMsgText(string msgText, int code);

SOCKET serverSocket;

char name[] = "Hello";

char checkServName[] = "CheckServ";

bool GetRequestFromClient(char\* name, short port, sockaddr\* from, int\* flen)

{

char nameServer[50];

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

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

{

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

}

Создание UDP-сокета для приема запросов от клиентов

SOCKADDR\_IN serv;

serv.sin\_family = AF\_INET;

serv.sin\_port = htons(port);

serv.sin\_addr.s\_addr = INADDR\_ANY;

Заполнение структуры для связывания сокета с адресом и портом

if (bind(serverSocket, (LPSOCKADDR)&serv, sizeof(serv)) == SOCKET\_ERROR)

{

throw SetErrorMsgText("Такой сервер уже есть:", WSAGetLastError());

}

Привязка сокета к адресу и порту

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

{

if (WSAGetLastError() == WSAETIMEDOUT)

{

return false;

}

else

{

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

}

}

Прием сообщения от клиента и запись его в буфер nameServer

if (!strcmp(nameServer, name))

{

std::cout << std::endl << "Позывной сервера совпадает.";

return true;

}

else if (strcmp(nameServer, checkServName) == 0) {

cout << "Найден идентичный сервер!" << endl;

return false;

}

else

{

std::cout << std::endl << "Позывной сервера не совпадает.";

return false;

}

Сравнение позывного сервера с полученным сообщением

}

bool PutAnswerToClient(const char\* name, sockaddr\* to, int\* lto)

{

if ((sendto(serverSocket, name, strlen(name) + 1, NULL, to, \*lto)) == SOCKET\_ERROR)

{

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

}

вызов функции sendto(), которая отсылает сообщение из буфера на удаленный сокет, указанный в аргументах to и lto. В случае возникновения ошибки при отправке сообщения, функция выбрасывает исключение, которое обрабатывается в вызывающей функции.

return true;

}

void CheckServer() {

char checkServ[] = "CheckServ";

Создается строка checkServ, содержащая сообщение, которое будет отправлено в качестве запроса к другим серверам.

int checkServer;

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

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

Создается UDP-сокет с помощью функции socket(). В случае ошибки создания сокета, функция throw SetErrorMsgText() вызывает исключение.

int optval = 1;

if (setsockopt(checkServer, SOL\_SOCKET,

SO\_BROADCAST,

(char\*)&optval,

sizeof(int)) == SOCKET\_ERROR)

{

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

}

Устанавливается опция сокета SO\_BROADCAST, чтобы разрешить отправку широковещательных сообщений. В случае ошибки установки опции, функция throw SetErrorMsgText() вызывает исключение.

Функция setsockopt() используется для установки параметров сокета. В данном случае, эта функция устанавливает параметры для сокета checkServer.

Параметры функции:

- checkServer: дескриптор сокета, для которого устанавливаются параметры.

- SOL\_SOCKET: уровень протокола, для которого устанавливаются параметры. В данном случае устанавливается уровень для работы с сокетами.

- SO\_BROADCAST: параметр, который устанавливает режим сокета для использования широковещательного адреса. Этот параметр позволяет отправлять сообщения широковещательно, то есть на все устройства в локальной сети.

- (char\*)&optval: указатель на переменную, содержащую значение параметра. В данном случае optval равно 1, что означает включение параметра.

- sizeof(int): размер буфера, содержащего значение параметра. В данном случае размер равен размеру типа int.

SOCKADDR\_IN all;

all.sin\_family = AF\_INET;

all.sin\_port = htons(2000);

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

Создается структура SOCKADDR\_IN, содержащая информацию об адресе и порте получателя. Адрес INADDR\_BROADCAST означает, что сообщение будет отправлено всем устройствам в сети.

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

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

Отправляется сообщение checkServ с помощью функции sendto(). Если отправка не удалась, функция throw SetErrorMsgText() вызывает исключение.

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

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

}

int main() {

setlocale(LC\_ALL, "rus");

WSADATA wsaData;

try {

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

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

CheckServer();

while (true) {

SOCKADDR\_IN clnt;

int lc = sizeof(clnt);

Создается структура SOCKADDR\_IN, представляющая клиента, и переменная lc для ее размера.

if (GetRequestFromClient(name, 2000, (sockaddr\*)&clnt, &lc))

{

std::cout << std::endl << PutAnswerToClient(name, (sockaddr\*)&clnt, &lc);

}

else

{

std::cout << std::endl << PutAnswerToClient("\*Error name\*", (sockaddr\*)&clnt, &lc);

}

Функция GetRequestFromClient вызывается с параметрами name, 2000, (sockaddr\*)&clnt, и &lc. Она ожидает запрос от клиента, используя порт 2000 и блокирует приложение до тех пор, пока запрос не будет получен. Если запрос получен, то функция возвращает true.

Если запрос от клиента был получен, то функция PutAnswerToClient вызывается с параметрами name, (sockaddr\*)&clnt, и &lc. Она отправляет ответ клиенту с именем name. Если запрос от клиента не был получен, то функция PutAnswerToClient вызывается с параметрами "Error name", (sockaddr\*)&clnt, и &lc. Она отправляет клиенту ответ с ошибкой.

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

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

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

IP-адрес и порт клиента получаются из структуры SOCKADDR\_IN.

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

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

}

if (WSACleanup() == SOCKET\_ERROR) //

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

}

catch (string errorMsgText) {

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

}

system("pause");

return 0;

}