#include <iostream>

#include "Winsock2.h"

#include "WS2tcpip.h"

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

using namespace std;

подключение необходимых библиотек для работы с сокетами и объявление пространства имен std.

string GetErrorMsgText(int code);

string SetErrorMsgText(string msgText, int code);

две функции для работы с ошибками в случае возникновения ошибок при работе с сокетами.

int main()

{

setlocale(LC\_ALL, "rus");

try {

WSADATA wsaData;

SOCKET clientSocket;

Функция main - точка входа в программу. Здесь вызывается функция setlocale, которая устанавливает локаль на русский язык. Затем объявляются переменные WSADATA и SOCKET для инициализации библиотеки сокетов и создания клиентского сокета.

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

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

Эта строка вызывает функцию WSAStartup, которая инициализирует библиотеку сокетов. Если функция вернет значение отличное от 0, то произошла ошибка при инициализации библиотеки.

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

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

создается клиентский сокет с помощью функции socket, которая принимает параметры: AF\_INET (адресное семейство), SOCK\_DGRAM (тип сокета), NULL (протокол). Если функция вернет INVALID\_SOCKET, то произошла ошибка при создании сокета.

Функция socket() создает сокет для соединения с другими узлами сети. Она принимает три параметра:

- af (address family) - семейство адресов, используемое для создания сокета. Например, для сетевого протокола IPv4 значение должно быть AF\_INET.

- type - тип создаваемого сокета, который определяет набор правил, используемых для обмена данными. Для UDP-сокета тип должен быть установлен на SOCK\_DGRAM.

- protocol - номер протокола, используемого для сокета. Обычно значение должно быть равным 0, что означает, что операционная система сама выберет подходящий протокол в зависимости от типа сокета и семейства адресов.

В коде, приведенном в вопросе, создается клиентский сокет для обмена сообщениями с сервером по протоколу UDP. Следовательно, в качестве параметров af и type используются AF\_INET и SOCK\_DGRAM соответственно. В качестве protocol используется значение NULL, что соответствует выбору протокола операционной системой по умолчанию. Результатом вызова функции будет дескриптор сокета, который используется для передачи и приема данных.

SOCKET serverSocket;

SOCKADDR\_IN serv;

int lc = sizeof(serv);

serv.sin\_family = AF\_INET;

serv.sin\_port = htons(2000);

inet\_pton(AF\_INET, "127.0.0.1", &(serv.sin\_addr));

Затем объявляется сокет сервера и структура SOCKADDR\_IN, которая содержит адрес и порт сервера. В данном случае, установлен порт 2000 и адрес 127.0.0.1 (localhost).

char ibuf[50],

obuf[50] = "Hello";

буферы сообщений

int libuf = 0,

lobuf = 0;

буферы байт

cout << "Введите кол-во сообщений: ";

int count;

cin >> count;

for (int i = 0; i < count; i++) {

if ((lobuf = sendto(clientSocket, obuf, strlen(obuf) + 1, NULL, (sockaddr\*)&serv, sizeof(serv))) == SOCKET\_ERROR)

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

Эта строка отправляет сообщение от клиента к серверу. clientSocket - идентификатор сокета, obuf - буфер для отправки данных, strlen(obuf) + 1 - размер данных, которые нужно отправить (включая нулевой байт в конце строки), NULL - флаги, которые можно использовать при отправке сообщения, (sockaddr\*)&serv - указатель на структуру sockaddr, содержащую IP-адрес и порт сервера, sizeof(serv) - размер структуры sockaddr. Если функция sendto возвращает SOCKET\_ERROR, то происходит вызов функции SetErrorMsgText с сообщением "send:" и кодом ошибки, полученным из WSAGetLastError(), и бросается исключение.

В данном коде функция sendto() используется для отправки сообщения на удаленный сервер. Параметры функции выглядят следующим образом:

- clientSocket - дескриптор сокета клиента.

- obuf - указатель на буфер с данными для отправки, в данном случае строку "Hello".

- strlen(obuf) + 1 - размер отправляемых данных в байтах, включая символ нуля в конце строки.

- flags - дополнительные опции отправки данных (используется значение по умолчанию, т.е. 0).

- &serv - указатель на структуру с информацией о сокете сервера (IP-адрес и номер порта).

- sizeof(serv) - размер структуры с информацией о сокете сервера.

if ((libuf = recvfrom(clientSocket, ibuf, sizeof(ibuf), NULL, (sockaddr\*)&serv, &lc)) == SOCKET\_ERROR)

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

Эта строка получает сообщение от сервера. clientSocket - идентификатор сокета, ibuf - буфер для хранения полученных данных, sizeof(ibuf) - размер буфера, NULL - флаги, которые можно использовать при получении сообщения, (sockaddr\*)&serv - указатель на структуру sockaddr, содержащую IP-адрес и порт сервера, &lc - указатель на переменную, содержащую размер структуры sockaddr, который изменяется функцией на фактический размер, который получается в ходе выполнения функции. Если функция recvfrom возвращает SOCKET\_ERROR, то происходит вызов функции SetErrorMsgText с сообщением "recv:" и кодом ошибки, полученным из WSAGetLastError(), и бросается исключение.

После отправки сообщения функция recvfrom() принимает ответ от сервера. Она принимает такие же параметры, как и sendto(), за исключением буфера с данными, который здесь является входным параметром. Функция recvfrom() используется для приема данных от сервера. В данном коде она принимает ответ от сервера в буфер ibuf размером sizeof(ibuf) байт и сохраняет его в переменную libuf.

cout << " Сообщение от сервера " << ": " << ibuf << " " << (i + 1) << endl;

}

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

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

if (WSACleanup() == SOCKET\_ERROR) // throw SetErrorMsgText("Cleanup:", WSAGetLastError());

}

catch (string errorMsgText) {

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

}

return 0;

}