#include <iostream>

#include "Winsock2.h"

Подключаем заголовочный файл iostream, который содержит стандартные потоки ввода-вывода (в частности, std::cout) и заголовочный файл Winsock2.h, который содержит объявления для использования библиотеки Winsock 2.0 для работы с сетевыми сокетами в Windows.

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

Эта строка кода указывает компилятору линковаться с библиотекой WS2\_32.lib, которая содержит реализацию функций Winsock 2.0.

#pragma warning(disable : 4996)

Эта строка кода отключает предупреждение "4996", которое связано с использованием функций, которые могут считаться небезопасными.

using namespace std;

Объявляем использование пространства имен std, чтобы не использовать квалификатор std:: перед каждым стандартным идентификатором (например, std::cout).

string GetErrorMsgText(int code);

string SetErrorMsgText(string msgText, int code);

Объявляем две функции GetErrorMsgText и SetErrorMsgText, которые используются для обработки ошибок, связанных с Winsock.

int main() {

setlocale(LC\_ALL, "rus");

try {

WSADATA wsaData;

SOCKET serverSocket;

if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0) throw SetErrorMsgText("Startup:", WSAGetLastError());

Здесь мы объявляем переменную wsaData типа WSADATA, которая содержит информацию о версии Winsock, используемой приложением. Затем мы создаем сокет serverSocket типа SOCKET, который будет использоваться для связи с клиентами. Затем мы вызываем функцию WSAStartup, которая инициализирует библиотеку Winsock с указанной версией (MAKEWORD(2, 0)). Если функция возвращает не нулевое значение, это означает, что произошла ошибка и мы выбрасываем исключение

if ((serverSocket = socket(AF\_INET, SOCK\_DGRAM, NULL)) == INVALID\_SOCKET) throw SetErrorMsgText("socket:", WSAGetLastError());

Функция socket() создает новый сокет, который может использоваться для взаимодействия с другими приложениями через сеть.

Первый аргумент AF\_INET указывает на то, что мы используем IP-протокол версии 4.

Второй аргумент SOCK\_DGRAM указывает на то, что мы используем протокол UDP (User Datagram Protocol), который обеспечивает ненадежную доставку данных.

Третий аргумент в данном случае установлен на NULL, так как для протокола UDP он не используется.

Если функция socket() вернула значение INVALID\_SOCKET, то возникает исключение и функция SetErrorMsgText() вызывается для вывода текста ошибки.

В итоге, переменная serverSocket будет содержать дескриптор созданного сокета.

SOCKADDR\_IN serv;

serv.sin\_family = AF\_INET;

serv.sin\_port = htons(2000);

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

Этот код создает структуру SOCKADDR\_IN, которая используется для определения адреса и порта, на котором сервер будет принимать соединения.

Здесь serv.sin\_family присваивается значение AF\_INET, что указывает на то, что мы будем использовать сокеты интернет-адресации IPv4.

serv.sin\_port устанавливает порт для прослушивания и задает его значение 2000.

serv.sin\_addr.s\_addr устанавливает IP-адрес сервера, которое устанавливается в INADDR\_ANY, что означает, что сервер будет слушать все доступные сетевые интерфейсы.

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

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

Этот код используется для привязки созданного сокета к конкретному IP-адресу и порту на локальной машине. Функция bind используется для связывания сокета с конкретным IP-адресом и портом, который определяется переменной serv. Если операция привязки не удалась, то функция вызывает исключение, содержащее текст ошибки, полученный с помощью функции WSAGetLastError().

SOCKADDR\_IN client;

memset(&client, 0, sizeof(client));

int lc = sizeof(client);

char ibuf[50];

int libuf = 0;

int lobuf = 0;

int t;

int t2;

int res = 0;

while (true) {

int i = 1;

while (true) {

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

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

recvfrom() это функция, которая ожидает приходящие сообщения на серверный сокет serverSocket, затем она сохраняет пришедшие данные в буфер ibuf указанного размера sizeof(ibuf), и адрес отправителя сохраняется в переменную client.

t = clock();

cout << i << " Клиент : ip " << inet\_ntoa(client.sin\_addr) << " Порт: " << htons(client.sin\_port) << " Сообщение: " << ibuf << endl;

i++;

cout это поток вывода в консоль, который выводит информацию о клиенте, который отправил сообщение на сервер, используя функцию inet\_ntoa() для конвертации адреса в удобочитаемый формат и функцию htons() для конвертации порта в удобочитаемый формат.

if ((lobuf = sendto(serverSocket, ibuf, strlen(ibuf) + 1, NULL, (sockaddr\*)&client, sizeof(client))) == SOCKET\_ERROR)

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

Данный код отправляет данные из буфера ibuf клиенту с использованием функции sendto(). Параметры функции:

serverSocket: дескриптор сокета, который принимает данные для отправки.

ibuf: указатель на буфер с данными для отправки.

strlen(ibuf) + 1: размер данных для отправки, включая символ завершения строки.

NULL: флаги для передачи данных. В данном случае флаги не используются.

(sockaddr\*)&client: указатель на структуру sockaddr, которая содержит адрес и порт клиента.

sizeof(client): размер структуры sockaddr.

Если функция sendto() возвращает SOCKET\_ERROR, т.е. произошла ошибка, то генерируется исключение с текстом ошибки, который получается через вызов WSAGetLastError().

t2 = clock();

res += (t2 - t);

}

cout << "Затраченное время: " << res << " секунд";

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

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

if (WSACleanup() == SOCKET\_ERROR)

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

}

}

catch (string errorMsgText) {

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

}

return 0;

}