**CLIENT**

Давайте подробно разберем приведенный вами код, который реализует клиентское приложение для взаимодействия с сервером через сокеты в Windows.

**Подробное объяснение кода**

1. **Подключение библиотек**:

cpp

Copy

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

Здесь подключаются необходимые библиотеки. winsock2.h и ws2tcpip.h используются для работы с сокетами в Windows, а iostream и string — для ввода и вывода данных.

1. **Подключение библиотеки**:

cpp

Copy

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

Эта строка указывает компилятору, что необходимо подключить библиотеку WS2\_32.lib, которая содержит функции для работы с Winsock.

1. **Объявление функций**:

cpp

Copy

string GetErrorMsgText(int code);

string SetErrorMsgText(string msgText, int code);

Объявляются две функции для обработки ошибок, которые будут использоваться позже в коде.

1. **Функция main**:

cpp

Copy

int main() {

setlocale(LC\_ALL, "rus");

Здесь начинается основная функция программы. Устанавливается локаль для корректного отображения русских символов.

1. **Инициализация Winsock**:

cpp

Copy

WSADATA wsaData;

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

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

}

Здесь происходит инициализация библиотеки Winsock. MAKEWORD(2, 2) указывает, что используется версия 2.2. Если инициализация не удалась, выбрасывается исключение с сообщением об ошибке.

1. **Создание сокета**:

cpp

Copy

SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (clientSocket == INVALID\_SOCKET) {

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

}

Создается сокет с использованием IPv4 (AF\_INET), для TCP-соединений (SOCK\_STREAM). Если сокет не удается создать, выбрасывается исключение.

1. **Настройка адреса сервера**:

cpp

Copy

SOCKADDR\_IN serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(2000);

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

Создается структура SOCKADDR\_IN для хранения информации о сервере. Устанавливается семейство адресов (IPv4), порт (2000) и IP-адрес сервера (127.0.0.1, что означает локальный хост).

1. **Подключение к серверу**:

cpp

Copy

if (connect(clientSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

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

}

Здесь клиент пытается подключиться к серверу. Если подключение не удалось, выбрасывается исключение.

1. **Отправка сообщения:** (одно)

cpp

Copy

const char\* message = "Hello from Client";

send(clientSocket, message, strlen(message) + 1, 0);

cout << "Сообщение отправлено: " << message << endl;

После успешного подключения клиент отправляет сообщение "Hello from Client" на сервер. Используется функция send, которая отправляет данные через сокет.

**Задание количества сообщений**: (несколько)

cpp

Copy

char ibuf[24], obuf[24] = "Hello from Client";

int count;

cout << "Enter number of messages:\n";

cin >> count;

Определяются буферы для входящих и исходящих сообщений. Пользователю предлагается ввести количество сообщений, которые он хочет отправить.

**Отправка и получение сообщений**:

cpp

Copy

int time = clock();

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

if (SOCKET\_ERROR == send(clientSocket, obuf, sizeof(obuf), NULL))

cout << "send:" << GetLastError() << endl;

if (SOCKET\_ERROR == recv(clientSocket, ibuf, sizeof(ibuf), NULL))

cout << "recv:" << GetLastError() << endl;

cout << ibuf << " " << (i + 1) << endl;

}

Запускается цикл, в котором отправляется заданное количество сообщений. Если отправка или получение сообщения завершается ошибкой, выводится сообщение об ошибке. После успешного получения сообщения оно выводится на экран вместе с номером итерации.

1. **Закрытие сокета и завершение Winsock**:

cpp

Copy

closesocket(clientSocket);

WSACleanup();

После отправки сообщения сокет закрывается, и выполняется очистка Winsock.

1. **Обработка ошибок**:  
   Функции GetErrorMsgText и SetErrorMsgText используются для преобразования кодов ошибок в строковые сообщения, которые затем выводятся в консоль.

**Общая суть**

Программа представляет собой клиентское приложение, использующее сокеты для подключения к серверу и отправки сообщения. Она инициализирует библиотеку Winsock, создает TCP-сокет, настраивает адрес сервера, устанавливает соединение и отправляет текстовое сообщение. Программа также включает обработку ошибок, чтобы информировать пользователя о возможных проблемах, таких как сбой инициализации Winsock, ошибки создания сокета или подключения к серверу. В конце программа корректно завершает работу, закрывая сокет и очищая ресурсы Winsock.

**SERVER**

Давайте подробно разберем представленный вами код, который реализует серверное приложение для работы с сокетами в Windows.

**Подробное объяснение кода**

1. **Подключение библиотек**:

cpp

Copy

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

Здесь подключаются необходимые заголовочные файлы:

* + iostream для ввода и вывода,
  + winsock2.h и ws2tcpip.h для работы с сокетами в Windows,
  + string для работы со строками.

1. **Подключение библиотеки**:

cpp

Copy

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

Эта строка сообщает компилятору, что необходимо подключить библиотеку WS2\_32.lib, которая содержит функции для работы с Winsock.

1. **Объявление функций**:

cpp

Copy

string GetErrorMsgText(int code);

string SetErrorMsgText(string msgText, int code);

Объявляются две функции для обработки ошибок, которые будут использоваться позже в коде.

1. **Функция main**:

cpp

Copy

int main() {

setlocale(LC\_ALL, "rus");

Запускается основная функция программы. Устанавливается локаль для корректного отображения русских символов на выводе.

1. **Инициализация Winsock**:

cpp

Copy

WSADATA wsaData;

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

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

}

Здесь происходит инициализация библиотеки Winsock. MAKEWORD(2, 2) указывает, что используется версия 2.2. Если инициализация не удалась, выбрасывается исключение с сообщением об ошибке.

1. **Создание сокета**:

cpp

Copy

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (serverSocket == INVALID\_SOCKET) {

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

}

Создается сокет с использованием IPv4 (AF\_INET) и для TCP-соединений (SOCK\_STREAM). Если сокет не удается создать, выбрасывается исключение.

1. **Настройка адреса сервера**:

cpp

Copy

SOCKADDR\_IN serv;

serv.sin\_family = AF\_INET;

serv.sin\_port = htons(2000);

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

Создается структура SOCKADDR\_IN для хранения информации о сервере:

* + sin\_family указывает на использование IPv4,
  + sin\_port задает порт (2000), который будет использоваться сервером,
  + sin\_addr.s\_addr устанавливается в INADDR\_ANY, что означает, что сервер будет принимать соединения на любом доступном IP-адресе.

1. **Привязка сокета**:

cpp

Copy

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

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

}

Привязывает сокет к указанному адресу и порту. Если привязка не удалась, выбрасывается исключение.

1. **Перевод сокета в режим прослушивания**:

cpp

Copy

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

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

}

Переводит сокет в режим прослушивания, чтобы он мог принимать входящие соединения. SOMAXCONN указывает максимальное количество подключений в очереди. Если перевод не удался, выбрасывается исключение.

1. **Сообщение о начале прослушивания**:

cpp

Copy

cout << "Сервер начал прослушку на порту 2000..." << endl;

Выводится сообщение о том, что сервер начал прослушивание подключений на порту 2000.

1. **Принятие входящего соединения**:

cpp

Copy

SOCKADDR\_IN clientAddr;

int clientAddrSize = sizeof(clientAddr);

SOCKET clientSocket = accept(serverSocket, (sockaddr\*)&clientAddr, &clientAddrSize);

if (clientSocket == INVALID\_SOCKET) {

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

}

Сервер принимает входящее соединение от клиента. Функция accept создает новый сокет для обмена данными с клиентом. Если соединение не удалось, выбрасывается исключение.

1. **Вывод информации о подключенном клиенте**:

cpp

Copy

cout << "Клиент подключен: " << inet\_ntoa(clientAddr.sin\_addr) << ":" << ntohs(clientAddr.sin\_port) << endl;

После успешного подключения выводится IP-адрес и порт клиента, используя inet\_ntoa для преобразования адреса в строковый формат и ntohs для преобразования порта.

1. **Получение сообщения от клиента**:

cpp

Copy

char buffer[256];

int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);

if (bytesReceived > 0) {

cout << "Сообщение от клиента: " << buffer << endl;

}

Сервер ждет, когда клиент отправит сообщение. Используется функция recv для получения данных. Если данные успешно получены, они выводятся на экран.

1. **Закрытие сокетов**:

cpp

Copy

closesocket(clientSocket);

closesocket(serverSocket);

Закрываются сокеты для освобождения ресурсов.

1. **Завершение работы Winsock**:

cpp

Copy

WSACleanup();

Очистка ресурсов Winsock перед завершением программы.

1. **Обработка ошибок**:  
   Функции GetErrorMsgText и SetErrorMsgText используются для преобразования кодов ошибок в строковые сообщения, которые затем выводятся в консоль.

**Общая суть**

Данная программа реализует серверное приложение, использующее сокеты для приема подключений от клиентов. Сервер инициализирует библиотеку Winsock, создает TCP-сокет, настраивает адрес и порт, на которых будет слушать входящие соединения. После успешного подключения к клиенту, сервер принимает сообщение и выводит его на экран. Весь процесс сопровождается обработкой возможных ошибок, что делает программу более надежной. После завершения работы сокеты закрываются, и ресурсы Winsock освобождаются, что гарантирует корректное завершение работы приложения.