// ============================ CLIENT ============================ //

#include <iostream> - библиотека ввода вывода

#include "Winsock2.h" - предоставляет программистам API для создания и управления сокетами (Socket API), которые позволяют обмениваться данными между приложениями, работающими на разных узлах сети.

#include "WS2tcpip.h" - является дополнительной библиотекой для разработки сетевых приложений на операционных системах Windows. Она предоставляет дополнительные функции для работы с сетью, включая расширенную поддержку протоколов и функции для работы с адресами и портами сетевых соединений. В частности, библиотека WS2tcpip.h содержит функции для конвертации адресов IPv4 и IPv6, обеспечивает поддержку различных протоколов, таких как ICMP, IGMP, и IPsec, а также позволяет использовать сокеты в неблокирующем режиме.

#pragma comment(lib, "WS2\_32.lib") - содержит функции, необходимые для работы с сокетами в операционных системах Windows, и является частью Winsock API

using namespace std; - пространство имен std

string GetErrorMsgText(int code); - предназначена для обработки стандартных ошибок библиотеки **WS2\_32.LIB**

string SetErrorMsgText(string msgText, int code); - предназначена для обработки стандартных ошибок библиотеки **WS2\_32.LIB**

int main()

{

setlocale(LC\_ALL, "rus");

try {

WSADATA wsaData; - структура, которая используется для инициализации и настройки сокетов в Windows

SOCKET clientSocket; - тип данных SOCKET используется для создания и идентификации сокетов в приложениях, которые используют Winsock API.

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

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

WSAStartup(), которая вызывается при начале работы с сокетами в приложении. Функция WSAStartup() инициализирует сокеты в Windows и заполняет структуру WSADATA информацией о версии сокетов, поддерживаемых протоколах, и других параметрах сокетов.

Данный код используется для инициализации Winsock API в Windows.

Функция WSAStartup() инициализирует Winsock API и определяет версию API, которая будет использоваться в приложении. Функция принимает два аргумента: первый - это версия API в виде значения типа WORD (2-байтовое целое число), которое создается с помощью функции MAKEWORD(); второй - это указатель на структуру WSADATA, которая используется для получения информации о версии Winsock API и других параметрах сокетов.

В данном коде, функция WSAStartup() вызывается с параметрами MAKEWORD(2, 0) и &wsaData. Первый параметр указывает, что используется версия Winsock API 2.0, а второй параметр - указатель на структуру WSADATA, которая будет заполнена информацией о версии API.

Если функция WSAStartup() завершается с ошибкой, то в коде используется исключение throw, чтобы вызвать функцию SetErrorMsgText(), которая формирует текстовую ошибку с помощью функции WSAGetLastError() и передает ее в исключение. Таким образом, если инициализация Winsock API завершается с ошибкой, то в приложении будет выброшено исключение с текстом ошибки.

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

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

Данный код используется для создания нового сокета с использованием Winsock API в Windows.

Функция socket() используется для создания сокета и возвращает новый дескриптор сокета. В данном коде, функция socket() вызывается с тремя параметрами: AF\_INET - указывает на использование протокола IPv4, SOCK\_STREAM - указывает на использование протокола TCP, и NULL - указывает на использование протокола по умолчанию.

Если вызов функции socket() завершается с ошибкой, то значение дескриптора сокета будет INVALID\_SOCKET. В коде используется проверка на INVALID\_SOCKET и, если дескриптор сокета не создан, используется исключение throw, чтобы вызвать функцию SetErrorMsgText(), которая формирует текстовую ошибку с помощью функции WSAGetLastError() и передает ее в исключение.

SOCKET serverSocket;

SOCKADDR\_IN serv;

SOCKADDR\_IN - это структура, используемая в Winsock API для представления адреса сокета с протоколом IPv4.

Структура SOCKADDR\_IN определяет адрес в виде IP-адреса и номера порта, который связан с сокетом. Она содержит следующие поля:

sin\_family - определяет семейство адресов (в данном случае, AF\_INET для протокола IPv4).

sin\_port - определяет номер порта для соединения (в сетевом порядке байт).

sin\_addr - определяет IP-адрес для соединения (в сетевом порядке байт).

sin\_zero - заполняющее поле для выравнивания структуры до размера 8 байт.

Структура SOCKADDR\_IN используется, например, при вызове функций bind() и connect(), которые привязывают сокет к определенному адресу и порту. В этих функциях передается указатель на структуру SOCKADDR\_IN, содержащую адрес и порт, к которому должен быть привязан сокет.

Также структура SOCKADDR\_IN может быть использована для получения адреса и порта, на которые было установлено соединение с помощью функции getpeername(), которая возвращает информацию об удаленном сокете.

serv.sin\_family = AF\_INET; // используется IP-адресация

serv.sin\_port = htons(2000); // TCP-порт 2000

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

Функция inet\_pton() (или "Internet address to presentation" в аббревиатуре) используется для преобразования текстового представления IP-адреса в двоичный формат (в формате сетевого порядка байт).

Функция принимает два аргумента: семейство адресов (например, AF\_INET для протокола IPv4) и строковое представление IP-адреса в формате "ddd.ddd.ddd.ddd". Она также принимает указатель на буфер, в который будет записано представление адреса в двоичном формате.

Функция возвращает значение 1, если преобразование успешно выполнено, 0, если переданный адрес некорректен, или -1, если произошла ошибка.

Функция inet\_pton() может использоваться для проверки корректности введенного пользователем IP-адреса перед его использованием в приложении, а также для преобразования адреса из текстового представления в двоичный формат для передачи его в функции Winsock API, такие как connect() или bind().

//serv.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // адрес сервера

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

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

Данный код пытается установить соединение с удаленным хостом с помощью функции connect(). Он передает три аргумента:

clientSocket - дескриптор сокета, который будет использоваться для соединения с удаленным хостом.

(sockaddr\*)&serv - указатель на структуру sockaddr, которая содержит информацию об адресе и порте, к которому нужно подключиться. В данном случае, это структура SOCKADDR\_IN, которая уже была заполнена адресом и портом сервера.

sizeof(serv) - размер структуры serv.

Функция connect() устанавливает соединение с удаленным хостом, используя дескриптор сокета, указанный в первом аргументе. Второй аргумент содержит указатель на структуру sockaddr, содержащую адрес и порт, к которому нужно подключиться. Третий аргумент указывает на размер структуры sockaddr.

Если соединение не удалось установить, то функция connect() возвращает SOCKET\_ERROR, поэтому условие в данном коде проверяет, была ли возвращена ошибка. Если функция вернула ошибку, то вызывается функция SetErrorMsgText(), которая формирует сообщение об ошибке с помощью функции WSAGetLastError(), и вызывает исключение с сообщением об ошибке. В противном случае соединение с удаленным хостом успешно установлено.

char ibuf[50], //буфер ввода

obuf[50] = "Hello from Client "; //буфер вывода

int libuf = 0, //количество принятых байт

lobuf = 0; //количество отправленных байь

cout << "Введите кол-во сообщений: ";

int count;

cin >> count;

//count = 10000;

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

if ((lobuf = send(clientSocket, obuf, strlen(obuf) + 1, NULL)) == SOCKET\_ERROR)

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

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

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

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

}

Данное сообщение описывает цикл отправки и получения данных между клиентом и сервером. Цикл повторяется count раз. В каждой итерации клиент отправляет данные на сервер с помощью функции send() и получает ответ от сервера с помощью функции recv().

Аргументы функции send():

clientSocket - дескриптор сокета, по которому происходит передача данных.

obuf - указатель на буфер, содержащий данные, которые нужно отправить.

strlen(obuf) + 1 - размер буфера, который нужно отправить. Функция strlen() возвращает длину строки без учета символа конца строки ('\0'), поэтому к длине строки добавляется 1, чтобы учитывать символ конца строки.

NULL - дополнительные флаги, которые могут быть использованы при передаче данных. В данном случае, флаги не заданы.

Если функция send() возвращает SOCKET\_ERROR, то возникает ошибка при передаче данных, и функция SetErrorMsgText() генерирует сообщение об ошибке с помощью функции WSAGetLastError(), которая возвращает код ошибки.

Аргументы функции recv():

clientSocket - дескриптор сокета, по которому происходит прием данных.

ibuf - указатель на буфер, в который будут записаны принятые данные.

sizeof(ibuf) - размер буфера, в который можно записать принятые данные.

NULL - дополнительные флаги, которые могут быть использованы при приеме данных. В данном случае, флаги не заданы.

Если функция recv() возвращает SOCKET\_ERROR, то возникает ошибка при приеме данных, и функция SetErrorMsgText() генерирует сообщение об ошибке с помощью функции WSAGetLastError(), которая возвращает код ошибки.

Таким образом, в данном цикле клиент и сервер обмениваются данными count раз, отправляя и принимая сообщения друг от друга.

if (SOCKET\_ERROR == send(clientSocket, "Конец передачи", sizeof("Конец передачи"), NULL))

{

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

}

Данный код отправляет на сервер сообщение "Конец передачи" и выводит на экран код ошибки в случае неудачной отправки.

Аргументы функции send():

clientSocket - дескриптор сокета, по которому происходит передача данных.

"Конец передачи" - указатель на буфер, содержащий сообщение, которое нужно отправить. В данном случае, это строка "Конец передачи".

sizeof("Конец передачи") - размер буфера, который нужно отправить. Функция sizeof() возвращает размер строки, включая символ конца строки ('\0'), поэтому в данном случае размер буфера будет равен 15.

NULL - дополнительные флаги, которые могут быть использованы при передаче данных. В данном случае, флаги не заданы.

Если функция send() возвращает SOCKET\_ERROR, то сообщение не было отправлено, и код ошибки можно получить с помощью функции GetLastError() и вывести на экран с помощью cout.

Таким образом, данный код отправляет на сервер сообщение о том, что передача данных завершена, и выводит на экран код ошибки в случае неудачной отправки.

if (closesocket(clientSocket) == SOCKET\_ERROR) // закрыть сокет

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

Данный код закрывает сокет, который был использован для установления соединения с сервером. Функция closesocket() используется для освобождения ресурсов, связанных с сокетом, после того, как соединение было закрыто.

Аргумент функции closesocket():

clientSocket - дескриптор сокета, который нужно закрыть.

Если функция closesocket() возвращает SOCKET\_ERROR, то значит произошла ошибка при закрытии сокета. В этом случае, с помощью функции SetErrorMsgText() создается строка с сообщением об ошибке, которая будет брошена как исключение.

Таким образом, данный код закрывает сокет, связанный с соединением с сервером, и в случае ошибки при закрытии выбрасывает исключение с сообщением об ошибке.

if (WSACleanup() == SOCKET\_ERROR) // завершить работу с библиотекой

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

Данный код используется для завершения работы с библиотекой Winsock, которая была инициализирована ранее с помощью функции WSAStartup(). Функция WSACleanup() освобождает ресурсы, выделенные для работы с библиотекой Winsock.

Если функция WSACleanup() возвращает SOCKET\_ERROR, то значит произошла ошибка при завершении работы с библиотекой. В этом случае, с помощью функции SetErrorMsgText() создается строка с сообщением об ошибке, которая будет брошена как исключение.

Аргумент функции WSACleanup() отсутствует.

Таким образом, данный код завершает работу с библиотекой Winsock и в случае ошибки при завершении работы выбрасывает исключение с сообщением об ошибке.

}

catch (string errorMsgText) {

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

}

return 0;

}

string GetErrorMsgText(int code) // Функция позволяет получить сообщение ошибки

{

string msgText;

switch (code)

{

case WSAEINTR: return "WSAEINTR: Работа функции прервана ";

case WSAEACCES: return "WSAEACCES: Разрешение отвергнуто";

case WSAEFAULT: return "WSAEFAULT: Ошибочный адрес";

case WSAEINVAL: return "WSAEINVAL: Ошибка в аргументе";

case WSAEMFILE: return "WSAEMFILE: Слишком много файлов открыто";

case WSAEWOULDBLOCK: return "WSAEWOULDBLOCK: Ресурс временно недоступен";

case WSAEINPROGRESS: return "WSAEINPROGRESS: Операция в процессе развития";

case WSAEALREADY: return "WSAEALREADY: Операция уже выполняется";

case WSAENOTSOCK: return "WSAENOTSOCK: Сокет задан неправильно";

case WSAEDESTADDRREQ: return "WSAEDESTADDRREQ: Требуется адрес расположения";

case WSAEMSGSIZE: return "WSAEMSGSIZE: Сообщение слишком длинное";

case WSAEPROTOTYPE: return "WSAEPROTOTYPE: Неправильный тип протокола для сокета";

case WSAENOPROTOOPT: return "WSAENOPROTOOPT: Ошибка в опции протокола";

case WSAEPROTONOSUPPORT: return "WSAEPROTONOSUPPORT: Протокол не поддерживается";

case WSAESOCKTNOSUPPORT: return "WSAESOCKTNOSUPPORT: Тип сокета не поддерживается";

case WSAEOPNOTSUPP: return "WSAEOPNOTSUPP: Операция не поддерживается";

case WSAEPFNOSUPPORT: return "WSAEPFNOSUPPORT: Тип протоколов не поддерживается";

case WSAEAFNOSUPPORT: return "WSAEAFNOSUPPORT: Тип адресов не поддерживается протоколом";

case WSAEADDRINUSE: return "WSAEADDRINUSE: Адрес уже используется";

case WSAEADDRNOTAVAIL: return "WSAEADDRNOTAVAIL: Запрошенный адрес не может быть использован";

case WSAENETDOWN: return "WSAENETDOWN: Сеть отключена";

case WSAENETUNREACH: return "WSAENETUNREACH: Сеть не достижима";

case WSAENETRESET: return "WSAENETRESET: Сеть разорвала соединение";

case WSAECONNABORTED: return "WSAECONNABORTED: Программный отказ связи";

case WSAECONNRESET: return "WSAECONNRESET: Связь восстановлена";

case WSAENOBUFS: return "WSAENOBUFS: Не хватает памяти для буферов";

case WSAEISCONN: return "WSAEISCONN: Сокет уже подключен";

case WSAENOTCONN: return "WSAENOTCONN: Сокет не подключен";

case WSAESHUTDOWN: return "WSAESHUTDOWN: Нельзя выполнить send : сокет завершил работу";

case WSAETIMEDOUT: return "WSAETIMEDOUT: Закончился отведенный интервал времени";

case WSAECONNREFUSED: return "WSAECONNREFUSED: Соединение отклонено";

case WSAEHOSTDOWN: return "WSAEHOSTDOWN: Хост в неработоспособном состоянии";

case WSAEHOSTUNREACH: return "WSAEHOSTUNREACH: Нет маршрута для хоста";

case WSAEPROCLIM: return "WSAEPROCLIM: Слишком много процессов";

case WSASYSNOTREADY: return "WSASYSNOTREADY: Сеть не доступна";

case WSAVERNOTSUPPORTED: return "WSAVERNOTSUPPORTED: Данная версия недоступна";

case WSANOTINITIALISED: return "WSANOTINITIALISED: Не выполнена инициализация WS2\_32.DLL";

case WSAEDISCON: return "WSAEDISCON: Выполняется отключение";

case WSATYPE\_NOT\_FOUND: return "WSATYPE\_NOT\_FOUND: Класс не найден";

case WSAHOST\_NOT\_FOUND: return "WSAHOST\_NOT\_FOUND: Хост не найден";

case WSATRY\_AGAIN: return "WSATRY\_AGAIN: Неавторизированный хост не найден";

case WSANO\_RECOVERY: return "WSANO\_RECOVERY: Неопределенная ошибка";

case WSANO\_DATA: return "WSANO\_DATA: Нет записи запрошенного типа";

case WSA\_INVALID\_HANDLE: return "WSA\_INVALID\_HANDLE: Указанный дескриптор события с ошибкой";

case WSA\_INVALID\_PARAMETER: return "WSA\_INVALID\_PARAMETER: Один или более параметров с ошибкой";

case WSA\_IO\_INCOMPLETE: return "WSA\_IO\_INCOMPLETE: Объект ввода - вывода не в сигнальном состоянии";

case WSA\_IO\_PENDING: return "WSA\_IO\_PENDING: Операция завершится позже";

case WSA\_NOT\_ENOUGH\_MEMORY: return "WSA\_NOT\_ENOUGH\_MEMORY: Не достаточно памяти";

case WSA\_OPERATION\_ABORTED: return "WSA\_OPERATION\_ABORTED: Операция отвергнута";

case WSASYSCALLFAILURE: return "WSASYSCALLFAILURE: Аварийное завершение системного вызова";

default: return "\*\*ERROR\*\*";

}

//TODO: добавить коды

//return msgText;

}

string SetErrorMsgText(string msgText, int code) // Функция возвращает сообщение ошибки

{

return msgText + GetErrorMsgText(code);

}