Министерство образования Российской Федерации  
Пензенский государственный университет  
Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №1 - 4  
по курсу «Разработка клиент-серверных приложений»

Выполнили:

студенты группы 19вв2:

Дорофеев А.А.,

Павлов А.А.,

Петров О.М.

Приняли:

Попов К.В.

Зинкин С.А.

2022

**Цель** **работы:**

1. Разработка программы – клиента в архитектуре взаимодействия “клиент - сервер” с использованием семейства протоколов TCP/IP.
2. Разработка программы – сервера в архитектуре взаимодействия “клиент - сервер” с использованием семейства протоколов TCP/IP с добавлением обработки данных на сервере.
3. Использование методов криптографии для закрытия данных, передаваемых между программами клиента и сервера, и их корректное расшифровывание.
4. Комплексная отладка программ клиента и сервера на одной ЛВС и разных ЛВС.



**Алгоритм работы программы-клиента:**

Для того, чтобы установить связь с процессом, выполняющимся на другой ЭВМ, клиентская программа должна создать сокет. Сокет в любой современной системе представляет собой особый вид файла, из которого можно читать и в которой можно записывать двоичные данные. При операциях обмена с сокетом нет никакого контроля типов, эта задача возлагается на приложения. На схеме алгоритма блок “Создание сокета” подразумевает вызов функции socket, который в случае успеха создает сокет – потоковое или дейтаграммное – и семейство протоколов. В данном случае создается потоковый сокет и используется семейство протоколов TCP/IP.

Установка соединения с другим процессом заключается в обмене специальными пакетами и возможно только тогда, когда тот процесс ожидает приема соединений. В противном случае, результат операции будет неудачным и будет получено сообщение о том, что-либо не удалось установить соединение, либо оно было разорвано (зависит от реализации). Для установки соединения требуется указать ЭВМ по IP-адресу или по доменному имени, которое обязательно должно быть преобразовано в IP-адрес, и процесс на этой ЭВМ (по целочисленному идентификатору, называемому портом). Все это реализуется при помощи функции connect. Если соединение успешно установлено, то сразу после вызова этой функции можно вести обмен с гарантированной доставкой пакетов. В противном случае работа невозможна.

Разрыв соединения означает обмен специальными пакетами и может производиться при помощи системного вызова close. Функция close уничтожает сокет, делая его непригодным к использованию (любая операция с ним будет заканчиваться неудачей).

**Алгоритм работы программы-сервера:**

Основной задачей серверной части является обработка. Обмен данными с клиентскими процессами есть важная составляющая часть этой задачи.

Для того, чтобы процессы-клиенты могли связаться с сервером, сервер создает сокет для обмена данными. На схеме алгоритма это представлено блоком “Создание сокета”. Производится так же, как и в клиентской программе.

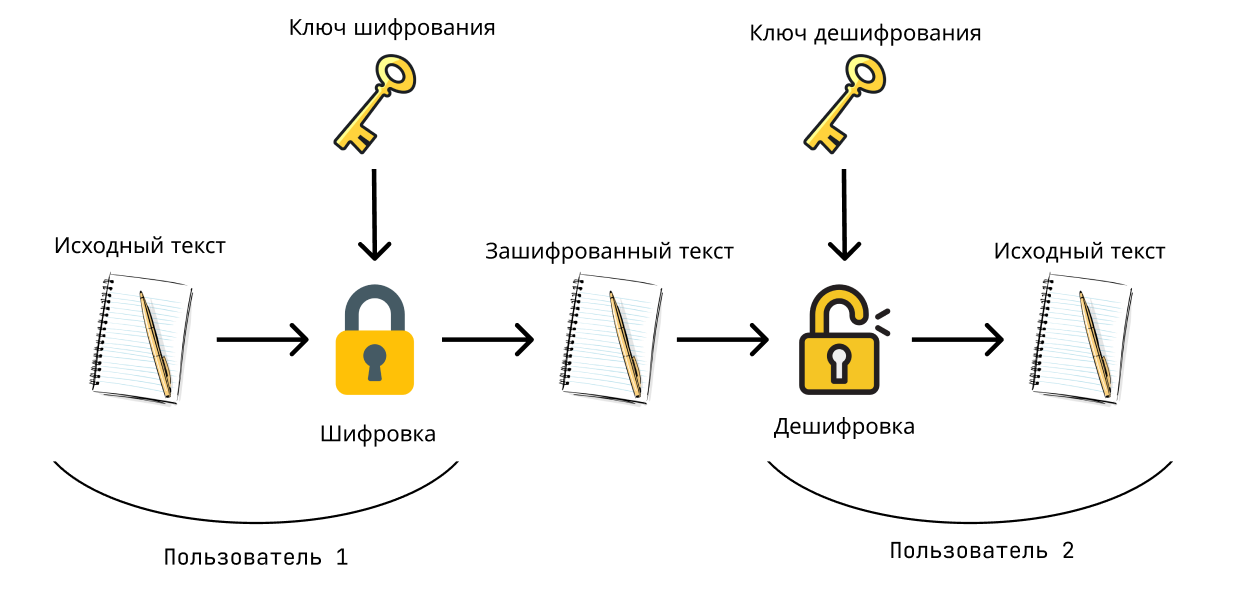
Следующий блок – “Получение локального адреса” – принципиально важен. Он служит для того, чтобы все запросы на соединения, приходящие на данную ЭВМ и обращающиеся к указанному порту, операционная система направляла данному процессу. Операция производится посредством системного вызова bind, в котором указывается созданный ранее сокет, IP–адрес ЭВМ (как правило, это константа 0) и идентификатор процесса, т. е. порт. После этого, разумеется, в случае успеха, программа сервера вызывает функцию listen, которая говорит операционной системе о том, что процесс ожидает поступления запросов на соединение на данный сокет и, что эти запросы нужно ставить в очередь указанной длины (в штуках).

Получение запроса на соединение происходит тогда, когда клиентский процесс вошел в блок “Установка соединения”, т. е. вызвал функцию connect. ОС сервера при этом создает копию сокета, чтобы программа могла на первом экземпляре продолжить работу, а на другом – вести обмен с подключившимся клиентом. Следующий блок – “Создание нового потока” – подразумевает порождения новой параллельной ветки программы, которая будет вести обработку данных. В системах Windows это обычно нить (thread), создаваемая при помощи функции \_beginthread, в UNIX-системах это новый процесс, создаваемый при помощи вызова fork.

По окончании работы с клиентом серверный процесс закрывает свою копию сокета и уничтожается.

**Алгоритм работы закрытия данных:**

Математическое описание шифра перестановки выглядит следующим образом. Пусть длина отрезков, на которые разбивается открытый текст, равна m, а S - взаимно однозначное отображение X = (x1,x2,…xm) в себя. Шифр перестановки преобразует отрезок открытого текста x1,x2,…xm в отрезок шифрованного текста S(x1,x2,…,xm)



Абсолютно стойкие шифры применяются в сетях связи с небольшим объемом передаваемой информации, которые используют, как правило, для передачи особо важной государственной информации. Это объясняется тем, что каждый передаваемый текст должен иметь свой собственный, единственный и неповторимый ключ. Следовательно, перед использованием этого шифра все абоненты должны быть обеспечены достаточным запасом случайных ключей и должна быть исключена возможность их повторного применения.

Задание:

1. Разработать программу клиента, которая должна:

* запрашивать у пользователя адрес программы-сервера;

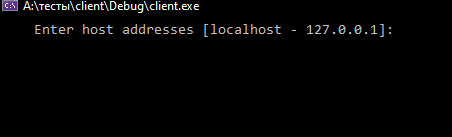


Рисунок 1 ― Запрос адреса сервера

* устанавливать соединение с сервером; (После установки соединения с сервером, выводиться небольшое вспомогательное сообщение, и программа приглашает пользователя ввести сообщение для отправки на сервер)

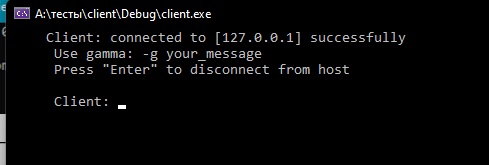


Рисунок 2 ― Соединение с сервером

* передавать на сервер данные;

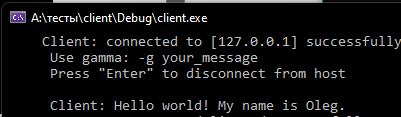


Рисунок 3 ― Передача данных на сервер

* принимать ответ от сервера и выводить его на экран; (При корректной отправке сообщения сервер пришлёт уведомление о том, что сообщение доставлено)

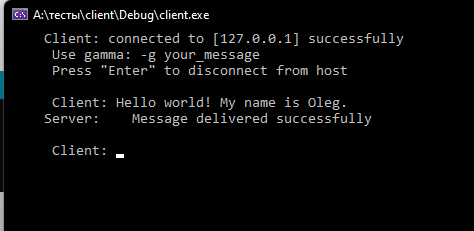


Рисунок 4 ― Ответ от сервера

* закрывать соединение с сервером; (Для закрытия соединения с сервером, необходимо нажать клавишу “Enter” – для отправки пустого сообщения)

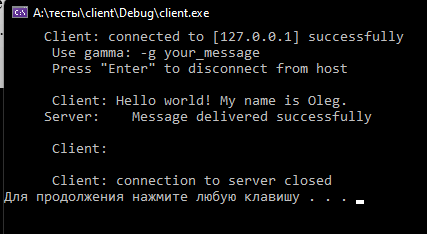


Рисунок 5 ― Закрытие соединения с сервером

1. Разработать программу сервера, которая должна:

* ожидать запросов от программ клиентов на соединение;

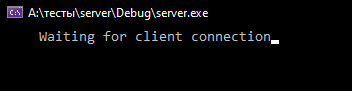


Рисунок 6 ― Ожидание подключения

* устанавливать соединение с клиентами; принимать данные от клиентов и выполнять их обработку;

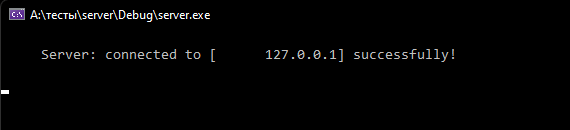


Рисунок 7 ― Установка соединения с клиентом

* пересылать результат обработки клиенту; (Если сообщение было успешно доставлено, сервер посылает ответное сообщение - Message delivered)

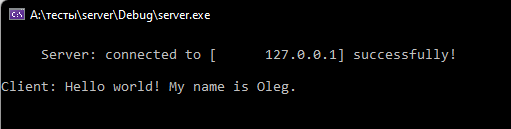


Рисунок 8 ― Вывод сообщения от клиента

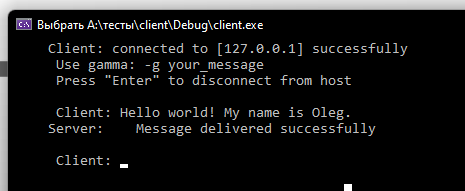


Рисунок 9 ― Сообщение об успешной доставке сообщения

1. Заложить функциональную возможность во взаимодействие “клиент-сервера”, которая должна:

* обеспечить закрытие данных при передаче их по сети; (Для отправки зашифрованного сообщения, необходимо обязательно ввести флаг “-g ” перед желаемым сообщением, после чего ввести желаемое сообщение, этот формат команды приведёт в вспомогательном меню окна клиента. Флаг “-g ” означает использование алгоритма шифрования “гаммирование” при обмене сообщениями между клиентом и сервером. На каждое отправленное сообщение генерируется индивидуальная гамма, с помощью которой можно шифровать/дешифровать сообщение, т.е. клиент производит шифрование сообщения, а сервер дешифрует его, анализируя присутствие флага “-g ” в сообщении)

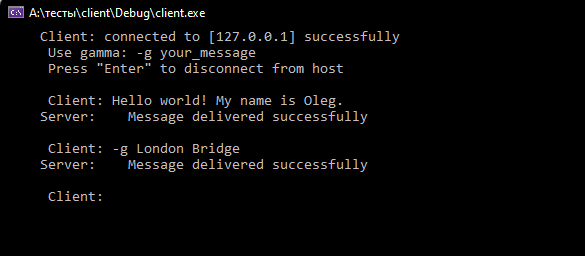


Рисунок 10 ― Ожидание подключения

* обеспечить корректное расшифрование передаваемых данных. (При корректном дешифровании полученного сообщения флага “-g ” очищается и мы получаем чистый текст. Например, при отправки зашифрованного сообщения -g Hello Server на выходе мы получим сообщение без флага => Hello Server)

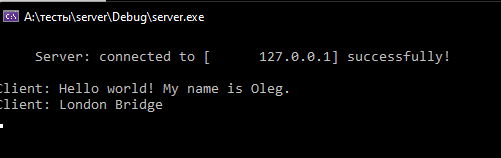


Рисунок 11 ― Ожидание подключения

1. Осуществить проверку работоспособности приложений “клиент-серверной” технологии, включающей:

* обеспечить проверку приложений на одной машине ЛВС;
* обеспечить проверку приложений на разных машинах ЛВС;

**Результат работы программы [на одной машине]:**

**Программа-сервер:**

После запуска сервера, программа автоматически перейдет в режим ожидания соединения от клиента. Никаких действий более выполнять не требуется. Для выхода из программы использовать комбинацию клавиш CTRL+C. После подключения клиента в окне программы отобразится сообщение об успешном подключении. А после отправки сообщения клиентом в окне сервера появится принятая информация.

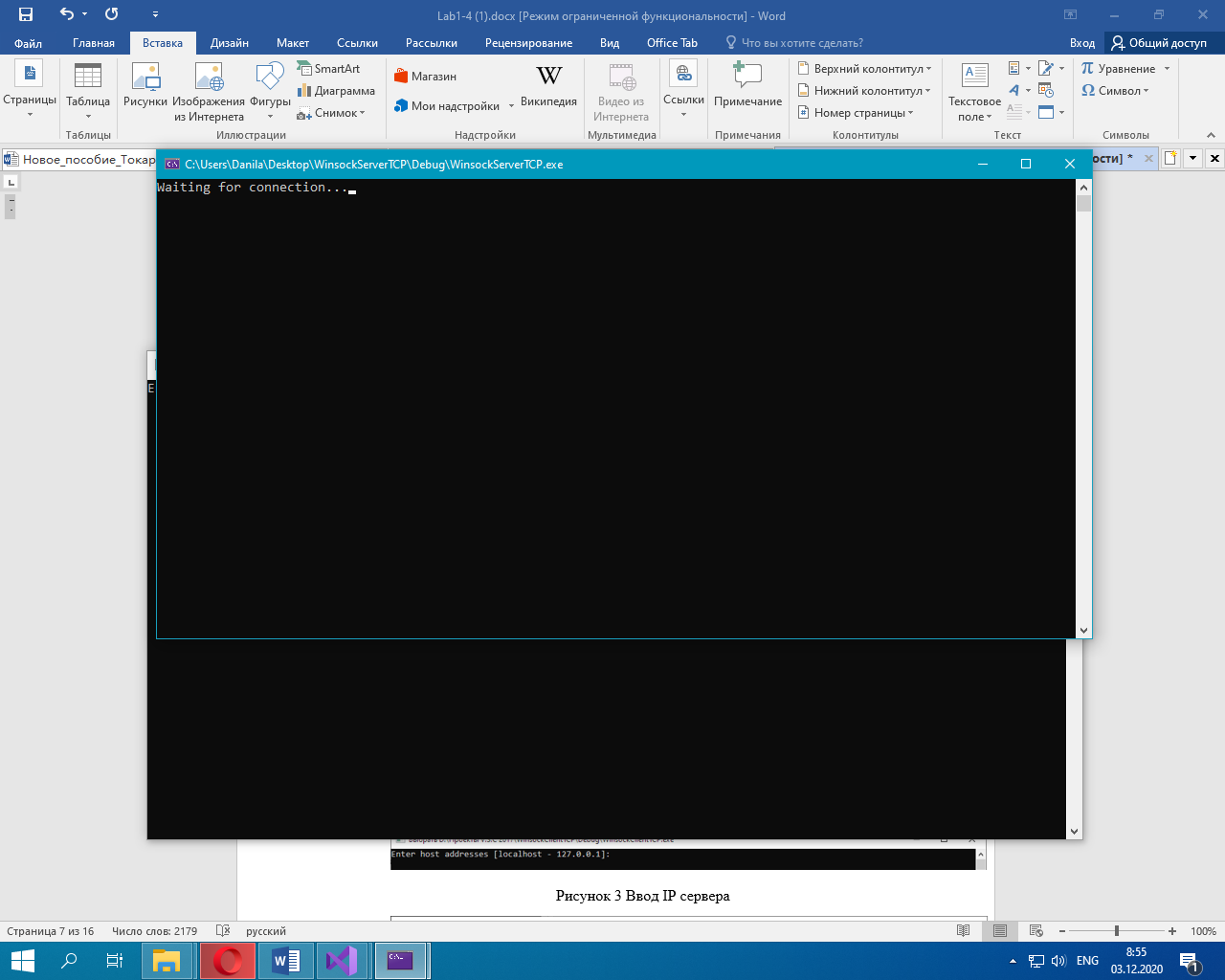


Рисунок 12 ― Ожидание подключения

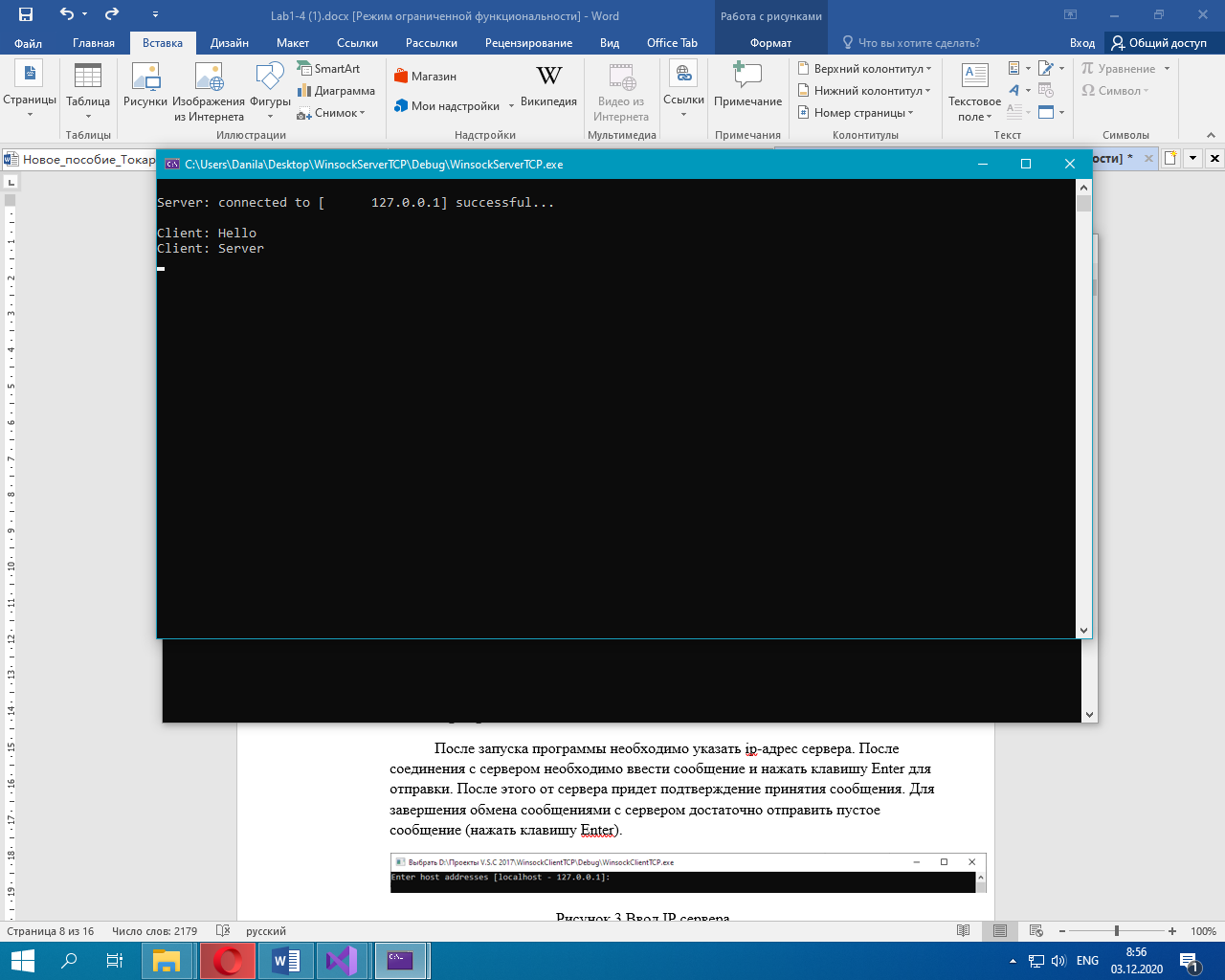


Рисунок 13 ― Обмен данными

**Программа-клиент:**

После запуска программы необходимо указать ip-адрес сервера. После соединения с сервером необходимо ввести сообщение и нажать клавишу Enter для отправки. После этого от сервера придет подтверждение принятия сообщения. Для завершения обмена сообщениями с сервером достаточно отправить пустое сообщение (нажать клавишу Enter).

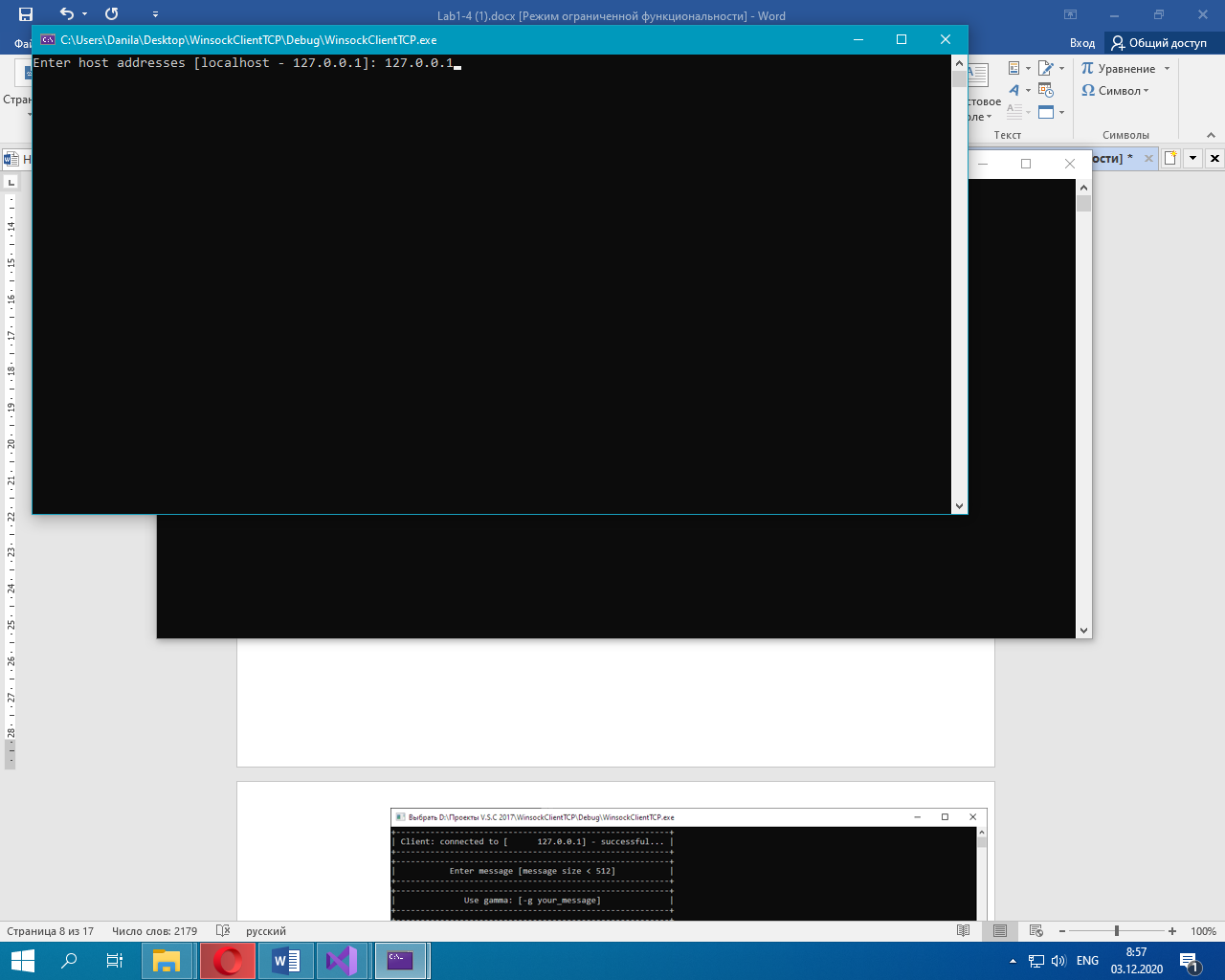


Рисунок 14 ― Ввод IP сервера

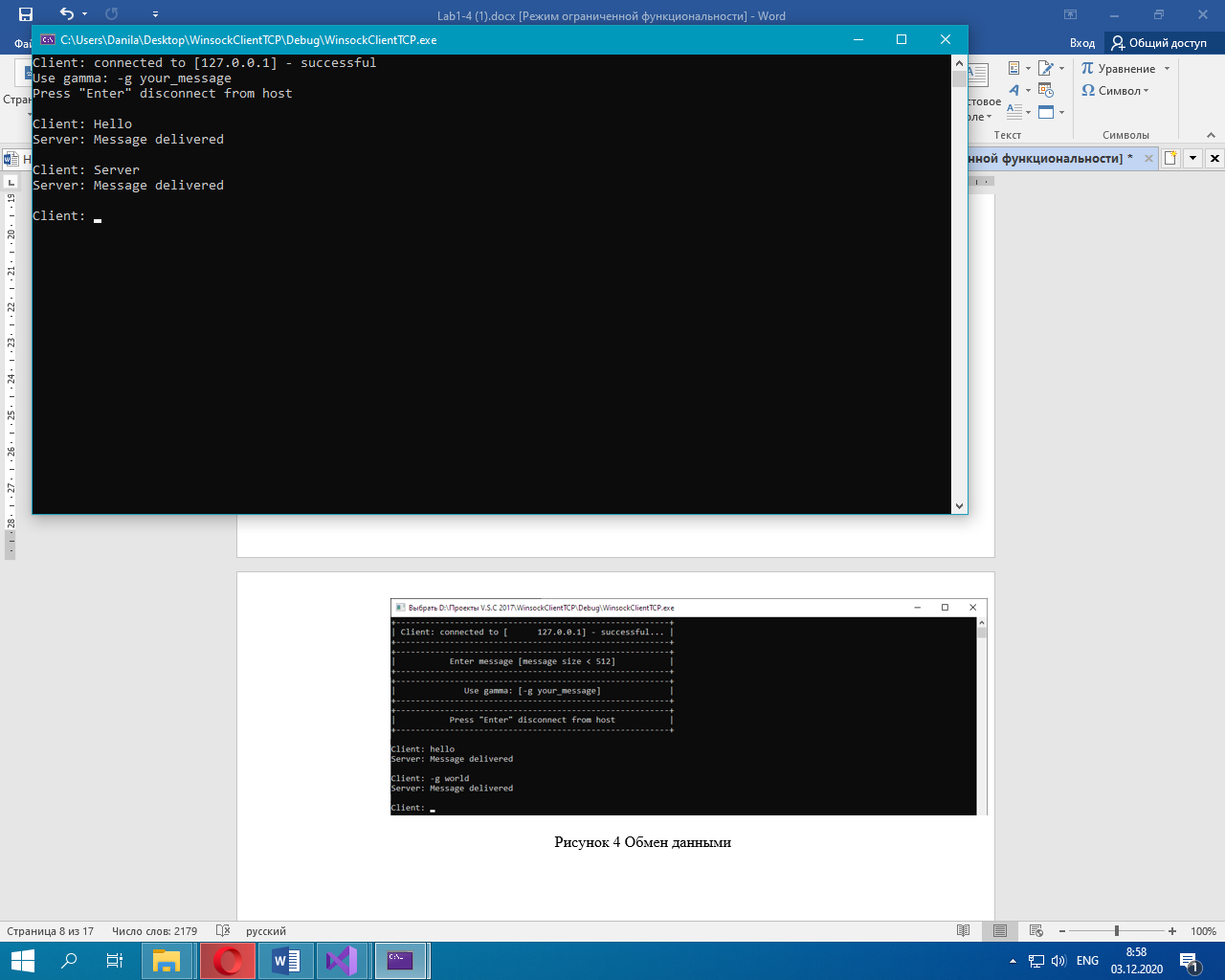


Рисунок 15 ― Обмен данными

**Результат работы программы [на разных машинах]:**

**Программа-сервер:**

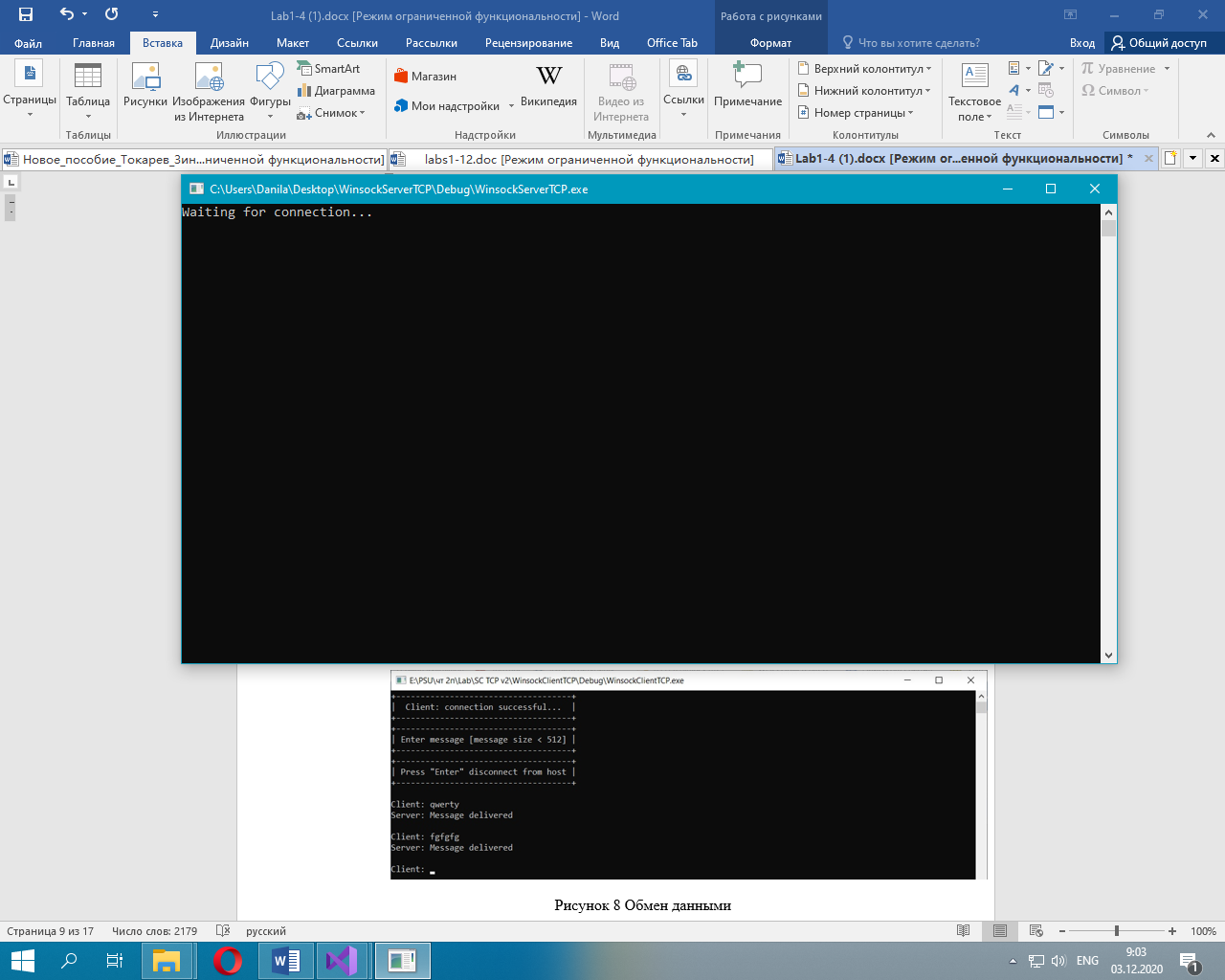


Рисунок 16 ― Ожидание подключения

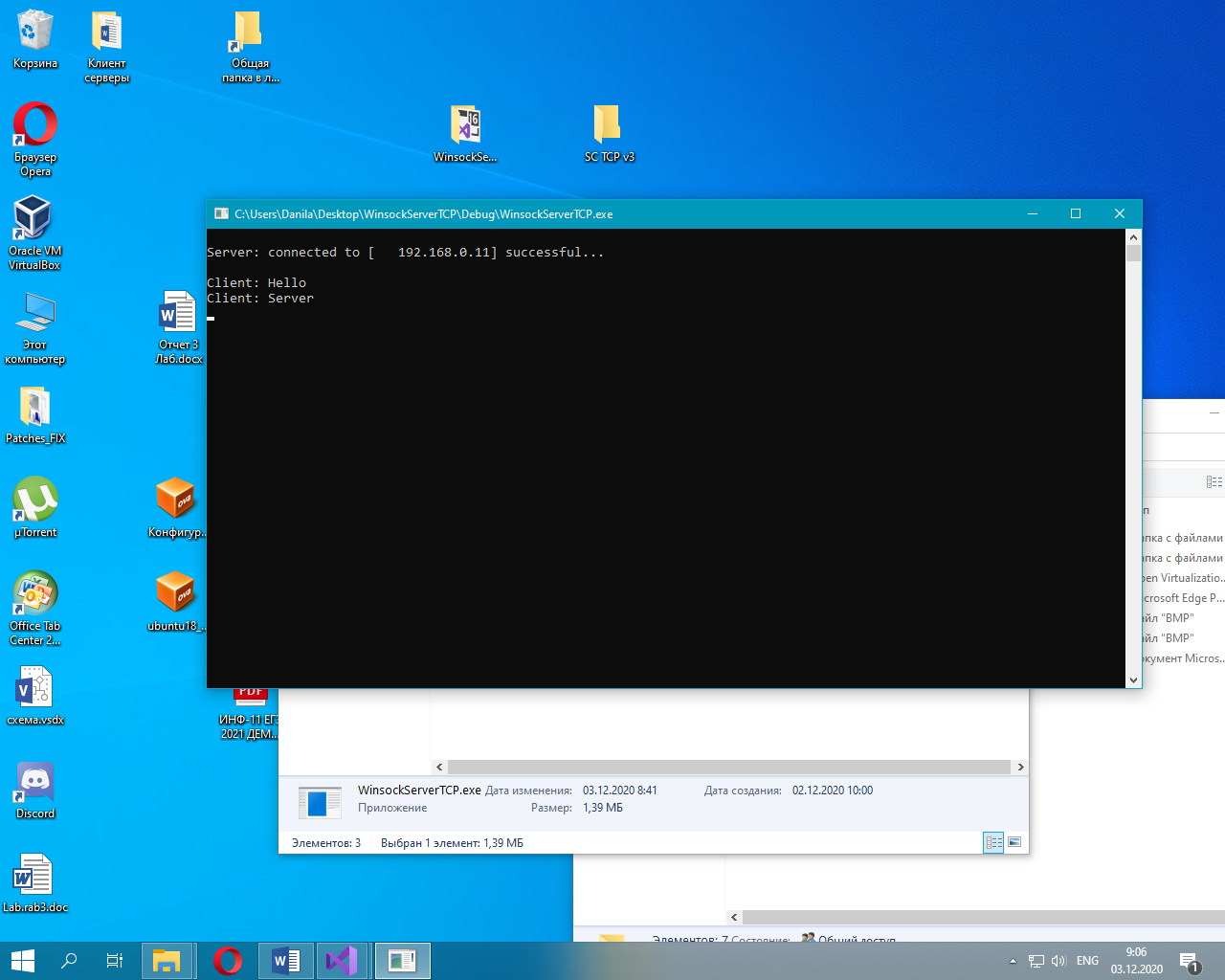


Рисунок 17 ― Обмен данными

**Программа-клиент:**

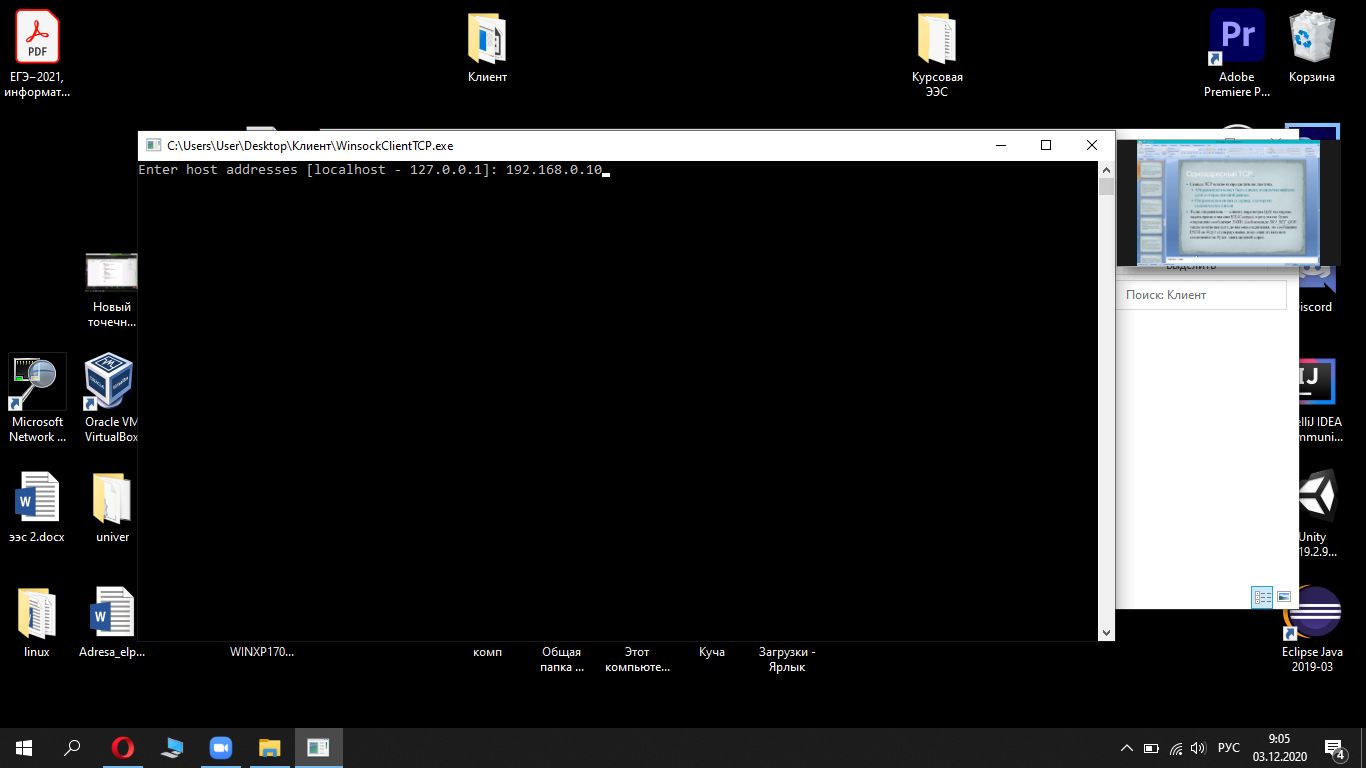


Рисунок 18 ― Ввод IP сервера

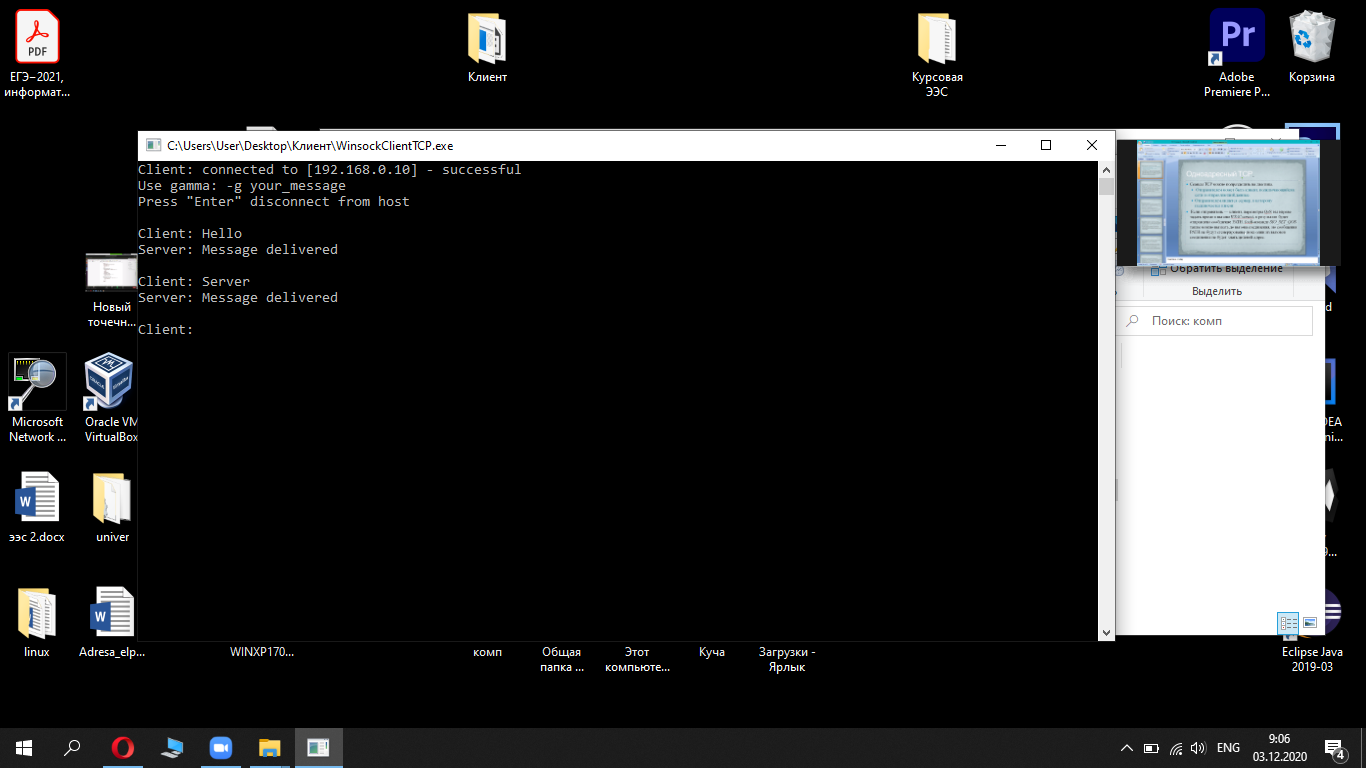


Рисунок 19 ― Обмен данными

**Вывод**

При выполнении данной лабораторной работы мы научились:

1. Разрабатывать программы – клиента в архитектуре взаимодействия “клиент - сервер” с использованием семейства протоколов TCP/IP.
2. Разрабатывать программы – сервера в архитектуре взаимодействия “клиент - сервер” с использованием семейства протоколов TCP/IP.
3. Использовать методы криптографии для закрытия данных, передаваемых между программами клиента и сервера.
4. Комплексной отладке программ клиента и сервера.

**Приложение А**

1. **Файл WinsockServerTCP.cpp**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <iomanip>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include "Gumming.h"

#pragma comment(lib, "Ws2\_32.lib")

int main()

{

using namespace std;

// Initializing Winsock

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData))

{

cerr << "WSAStartup failed";

return 1;

}

struct addrinfo\* result = NULL;

struct addrinfo hints =

{

AI\_PASSIVE, // flags

AF\_INET, // family

SOCK\_STREAM, // socktype

IPPROTO\_TCP // protocol

};

if (getaddrinfo(NULL, "7000", &hints, &result))

{

cerr << "getaddrinfo failed";

WSACleanup();

return 1;

}

// Creating a Socket for the Server

SOCKET listenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (listenSocket == INVALID\_SOCKET)

{

cerr << "socket failed";

freeaddrinfo(result);

WSACleanup();

return 1;

}

// Binding a Socket

if (bind(listenSocket, result->ai\_addr, result->ai\_addrlen) == SOCKET\_ERROR)

{

cerr << "bind failed";

freeaddrinfo(result);

closesocket(listenSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

// Listening on a Socket

if (listen(listenSocket, SOMAXCONN) == SOCKET\_ERROR)

{

cerr << "listen failed";

closesocket(listenSocket);

WSACleanup();

return 1;

}

//Accepting a Connection

cout << "Waiting for connection...";

sockaddr\_in remoteHostInfo;

socklen\_t remoteHostInfoLen = sizeof(remoteHostInfo);

SOCKET clientSocket = accept(listenSocket, (struct sockaddr\*)&remoteHostInfo, &remoteHostInfoLen);

if (clientSocket == INVALID\_SOCKET)

{

cerr << "accept failed";

closesocket(listenSocket);

WSACleanup();

return 1;

}

else

{

system("cls");

cout << "\nServer: connected to [" << setfill(' ') << setw(15) << inet\_ntoa(remoteHostInfo.sin\_addr)

<< "] successful...\n\n";

}

closesocket(listenSocket);

//Receiving and Sending Data on the Server

const unsigned bufferSize = 513;

int gammaBuffer[bufferSize - 1];

char dataBuffer[bufferSize];

int res;

do

{

// Receiving a message

res = recv(clientSocket, dataBuffer, bufferSize, 0);

if (dataBuffer[0] == '-' && dataBuffer[1] == 'g' && dataBuffer[2] == ' ')

{

// Get gamma

recv(clientSocket, (char\*)gammaBuffer, (bufferSize - 1) \* sizeof(int), 0);

// Decrypt message

decryption(dataBuffer, gammaBuffer, res);

}

if (res > 0)

{

// Sending a message

cout << "Client: " << dataBuffer << endl;

const char\* md = "Message delivered";

if (send(clientSocket, md, (int)strlen(md) + 1, 0) == SOCKET\_ERROR)

{

cerr << "send failed";

closesocket(clientSocket);

WSACleanup();

return 1;

}

}

else if (res == 0)

{

cout << "\nServer: connection closed...\n";

}

else

{

cerr << "recv failed";

closesocket(clientSocket);

WSACleanup();

return 1;

}

} while (res > 0);

// Disconnecting the Server

if (shutdown(clientSocket, SD\_SEND))

{

cerr << "shutdown failed";

closesocket(clientSocket);

WSACleanup();

return 1;

}

closesocket(clientSocket);

WSACleanup();

system("pause");

return 0;

}

1. **Файл WinsockClientTCP.cpp**

#include <iostream>

#include <iomanip>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include "Gumming.h"

#pragma comment(lib, "Ws2\_32.lib")

int main()

{

//прежде чем начать работу с сетью нужно загрузить необходимую версию библиотеки,иначе вызов сетевой функции вернёт ошибку

using namespace std;

// Initializing Winsock

WSADATA wsaData;

//Проверка на что?

if (WSAStartup(MAKEWORD(2, 2), &wsaData))//wsastrartup() -функция загрузки библиотеки, makeword()-указываем версию winsock

{

cerr << "WSAStartup failed";

return 1;

}

struct addrinfo\* result = NULL;

struct addrinfo hints =

{

NULL, // flags

AF\_UNSPEC, // family //AF\_UNSPEC - неважно ipv4 или ipv6 //AF\_INET или AF\_INET6

SOCK\_STREAM, // socktype

IPPROTO\_TCP // protocol

};

const unsigned bufferSize = 513;

char dataBuffer[bufferSize];

cout << "Enter host addresses [localhost - 127.0.0.1]: ";

cin.get(dataBuffer, bufferSize).get();

//getaddrinfo((имяхоста/IP-адрес), (Порт), (структура с параметрами), (result указатель на структуру с результатами)

if (getaddrinfo(dataBuffer, "7000", &hints, &result)) // getaddrinfo("127.0.0.1", "7000", &hints, &result)

{

cerr << "getaddrinfo failed";

WSACleanup();

return 1;

}

// Creating a Socket for the Client

SOCKET connectSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (connectSocket == INVALID\_SOCKET)

{

cerr << "socket failed";

freeaddrinfo(result);

WSACleanup();

return 1;

}

// Connecting to a Socket

if (connect(connectSocket, result->ai\_addr, result->ai\_addrlen) == SOCKET\_ERROR)

{

cerr << "connect failed";

freeaddrinfo(result);

closesocket(connectSocket);

WSACleanup();

return 1;

}

else

{

system("cls");

cout << "Client: connected to [" << dataBuffer << "] - successful\n"

<< "Use gamma: -g your\_message\n"

<< "Press \"Enter\" disconnect from host\n\n";

}

freeaddrinfo(result);

// Sending and Receiving Data on the Client

int gammaBuffer[bufferSize - 1];

int res;

do

{

// Sending a message

cout << "Client: ";

cin.get(dataBuffer, bufferSize);

res = strlen(dataBuffer);

while (res && (char)cin.get() != '\n');

if (dataBuffer[0] == '-' && dataBuffer[1] == 'g' && dataBuffer[2] == ' ')

{

// Generating and sending gamma

createGamma(gammaBuffer, res);

// Encrypt message

encryption(dataBuffer, gammaBuffer, res);

}

if (send(connectSocket, dataBuffer, res + (bool)res, 0) == SOCKET\_ERROR)

{

cerr << "send failed";

closesocket(connectSocket);

WSACleanup();

return 1;

}

if (dataBuffer[0] == '-' && dataBuffer[1] == 'g' && dataBuffer[2] == ' ')

{

// Send key

if (send(connectSocket, (char\*)gammaBuffer, (bufferSize - 1) \* sizeof(int), 0) == SOCKET\_ERROR)

{

cerr << "send failed";

closesocket(connectSocket);

WSACleanup();

return 1;

}

}

if (res == 0)

{

cout << "\nClient: connection closed...\n";

break;

}

// Receiving a message

res = recv(connectSocket, dataBuffer, bufferSize, 0);

if (res > 0)

{

cout << "Server: " << dataBuffer << endl << endl;

}

else

{

cerr << "recv failed";

closesocket(connectSocket);

WSACleanup();

return 1;

}

} while (res > 0);

// Disconnecting the Client

if (shutdown(connectSocket, SD\_SEND))

{

cerr << "shutdown failed";

closesocket(connectSocket);

WSACleanup();

return 1;

}

closesocket(connectSocket);

WSACleanup();

system("pause");

return 0;

}

1. **Файл Gumming.cpp**

#include "Gumming.h"

#include <random>

const char\* alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 0123456789";

static const int SP = 3;

int getIndex(char symbol)

{

for (size\_t i = 0; i < strlen(alphabet); i++)

{

if (\*(alphabet + i) == symbol)

return i;

}

return 0;

}

void createGamma(int gamma[], int messageSize)

{

std::random\_device rd;

std::default\_random\_engine dre(rd());

std::uniform\_int\_distribution<int> uid(0, strlen(alphabet));

for (int i = SP; i < messageSize; i++)

{

gamma[i] = uid(dre);

}

}

void encryption(char data[], int gamma[], int messageSize)

{

int alphabetSize = strlen(alphabet) - 1;

int index, \_gamma;

for (int i = SP; i < messageSize; i++)

{

index = getIndex(data[i]);

\_gamma = (gamma[i] + index > alphabetSize ? (gamma[i] + index) % alphabetSize : gamma[i] + index);

data[i] = alphabet[\_gamma];

}

}

void decryption(char data[], int gamma[], int messageSize)

{

int alphabetSize = strlen(alphabet) - 1;

int index, SP;

for (int i = ::SP; i < messageSize - 1; i++)

{

index = getIndex(data[i]);

index = (index + alphabetSize - gamma[i] > alphabetSize ? (index + alphabetSize - gamma[i]) % alphabetSize : index + alphabetSize - gamma[i]);

data[i] = alphabet[index];

}

for (SP = 3; SP < messageSize - 1 - 3; SP += 3)

{

data[SP - 3] = data[SP];

data[SP - 2] = data[SP + 1];

data[SP - 1] = data[SP + 2];

}

for (; SP < messageSize - 1; SP++)

{

data[SP - 3] = data[SP];

}

data[SP - 3] = '\0';

}