Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

на тему

Средства обмена данными (Windows). Изучение и использованием средств обмена данными и совместного доступа

Студент: гр.153504

Пригожий К. А.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146635545)

[2 Теоретические сведения 4](#_Toc146635546)

[3 Результат выполнения программы 5](#_Toc146635547)

[Список использованных источников 7](#_Toc146635548)

[Приложение А 8](#_Toc146635549)

# 1 ЦЕЛЬ РАБОТЫ

Создать приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

# Интерфейс сокетов и сетевое программирование в *Windows* предоставляют возможности для программного взаимодействия через сеть с использованием сокетов. Сокеты являются стандартными механизмами обмена данными по сети, а *Win32 API* предоставляет функции и объекты для их создания, настройки и использования.

# Для реализации программы, использующей интерфейс сокетов и основы сетевого программирования в *Windows*, можно использовать следующие подходы и функции:

# Создание сокета – функция *socket* для создания сокета, указав тип (например, `*SOCK\_STREAM*` для потокового соединения) и протокол (например, `*IPPROTO\_TCP*` для *TCP*-соединения).

# Привязка сокета – функцию bind для привязки сокета к определенному адресу и порту. Это позволит программе принимать входящие соединения или отправлять данные по указанному адресу.

# Прослушивание входящих соединений – функция listen для установки сокета в режим прослушивания входящих соединений. Это позволяет программе ожидать входящие запросы на соединение.

# Принятие входящих соединений – функция accept для принятия входящих соединений от клиентов. Функция accept создает новый сокет, который можно использовать для обмена данными с клиентом.

# Отправка и прием данных – функция send и recv для отправки и приема данных по сокету. Эти функции позволяют передавать информацию между клиентом и сервером.

# Реализация сетевых протоколов может быть осуществлена на базе созданных сокетов. Можно разрабатывать собственные протоколы для организации передачи данных между клиентами и сервером, либо использовать стандартные протоколы, такие как *TCP* или *UDP*.

# При разработке собственных протоколов необходимо определить формат сообщений, способы кодирования и декодирования данных, а также логику взаимодействия между клиентом и сервером.

# Использование интерфейса сокетов и сетевого программирования в Windows позволяет реализовать программное взаимодействие через сеть.

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Реализованная программа является клиент-серверным приложением.

Сначала нужно запустить сервер и он выведет сообщение о старте.(рисунок 1).

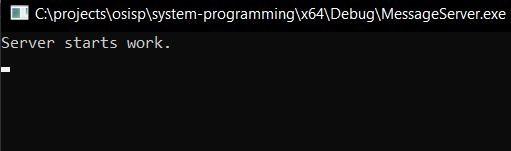


Рисунок 1 – Вывод сервера, сообщающий о старте

После этого нужно запустить программу клиента, которая при успешном подключении к серверу просит ввести имя пользователя (рисунок 2).

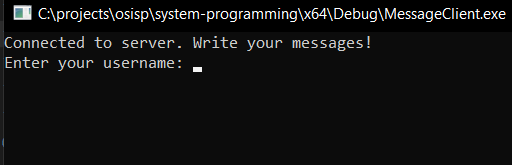


Рисунок 2 – Вывод клиентского приложения при старте

После ввода имени на сервере выводится сообщение об успешной авторизации (рисунок 3).



Рисунок 3 – Вывод сервера при авторизации

После этого пользователь может отсылать сообщения из стандартного ввода, которые будут отображаться на сервере и отправляться всем подключенным пользователям (рисунок 4).

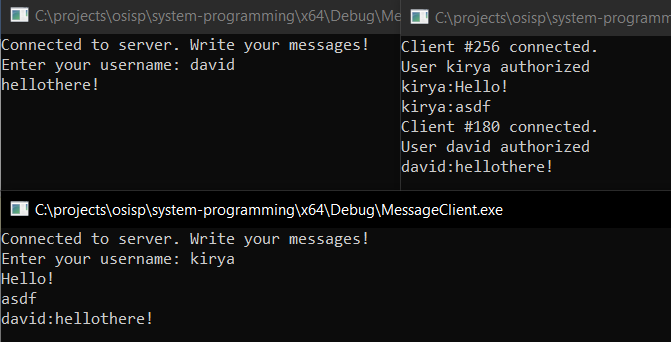


Рисунок 4 – Вывод клиентов и сервера при отправке сообщения

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] WSAStartup function (winsock.h) - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-wsastartup>

[2] getaddrinfo function (ws2tcpip.h) - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/ws2tcpip/nf-ws2tcpip-getaddrinfo>

[3] socket function (winsock2.h) - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-socket>

[4] bind function (winsock.h) - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-bind>

[5] listen function (winsock2.h) - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-listen>

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл MessageClient.main.h**

#pragma once

#include <iostream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <thread>

#include <sstream>

WSAData data;

addrinfo hints;

addrinfo\* serverInfo = nullptr;

SOCKET connectSocket;

std::thread receive\_thread;

void clear(const std::string& message, addrinfo\* info = nullptr, SOCKET\* socket = nullptr);

**Файл MessageClient.main.cpp**

#include "main.h"

#pragma comment(lib, "ws2\_32.lib")

void setHints()

{

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

}

void sendMessages() {

std::string username;

std::cout << "Enter your username: ";

std::cin >> username;

send(connectSocket, username.c\_str(), username.length(), 0);

std::string message;

while (true) {

std::cin >> message;

if (send(connectSocket, message.c\_str(), message.length(), 0) == SOCKET\_ERROR) {

std::cout << "Error occured when seding message to server.";

}

}

}

void receiveMessageFromServer()

{

char receivebuffer[256];

while (true) {

ZeroMemory(receivebuffer, 256);

if (recv(connectSocket, receivebuffer, 256, 0) != SOCKET\_ERROR)

{

std::cout << receivebuffer;

}

}

}

void clear(const std::string& message, addrinfo\* info, SOCKET\* socket)

{

std::cout << message << "\n";

if (info)

freeaddrinfo(info);

if (socket)

closesocket(\*socket);

WSACleanup();

}

int main() {

int res = WSAStartup(MAKEWORD(2, 2), &data);

if (res != 0)

{

std::cout << "Startup failed with code " << res << '\n';

return 1;

}

setHints();

res = getaddrinfo("127.0.0.1", "7777", &hints, &serverInfo);

if (res != 0)

{

clear("getaddrinfo error. Code : " + std::to\_string(res) + '\n');

return false;

}

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

if (connectSocket == INVALID\_SOCKET)

{

clear("Connection ConnectSocket failed.\n", serverInfo);

return 1;

}

res = connect(connectSocket, serverInfo->ai\_addr, serverInfo->ai\_addrlen);

if (res == SOCKET\_ERROR)

{

clear("Unable connect to server.\n", serverInfo, &connectSocket);

return false;

}

std::cout << "Connected to server. Write your messages!\n";

std::thread sendMessagesThread(&sendMessages);

std::thread receiveMessagesThread(&receiveMessageFromServer);

sendMessagesThread.join();

receiveMessagesThread.join();

clear("", serverInfo, &connectSocket);

WSACleanup();

}

**Файл MessageServer.main.h**

#pragma once

#include <iostream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <thread>

#include <sstream>

#include <list>

#include <map>

WSAData data;

addrinfo hints;

addrinfo\* serverInfo = nullptr;

SOCKET listenSocket;

bool itWorks = true;

std::map<SOCKET, std::string> connectionsInfo;

void setHints();

void clear(const std::string& message, addrinfo\* info = nullptr, SOCKET\* socket = nullptr);

void receiveMessageFromClient(SOCKET client);

void sendMessageToClients(const char\* buffer, SOCKET from);

**Файл MessageServer.main.cpp**

#include "main.h"

#pragma comment(lib, "ws2\_32.lib")

int main()

{

int result = WSAStartup(MAKEWORD(2, 2), &data);

if (result != 0) {

std::cout << "Startup failed with code " << result << '\n';

return 1;

}

setHints();

result = getaddrinfo(NULL, "7777", &hints, &serverInfo);

if (result != 0) {

std::stringstream ss;

ss << "getaddrinfo error. Code : " << result << '\n';

clear(ss.str());

return false;

}

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

if (listenSocket == INVALID\_SOCKET) {

clear("Connection ListenSocket failed.\n", serverInfo);

return false;

}

result = bind(listenSocket, serverInfo->ai\_addr, serverInfo->ai\_addrlen);

if (result == SOCKET\_ERROR) {

clear("binding socket failed.\n", serverInfo, &listenSocket);

return false;

}

result = listen(listenSocket, SOMAXCONN);

if (result == SOCKET\_ERROR) {

clear("listen failed.\n", serverInfo, &listenSocket);

return false;

}

std::cout << "Server starts work.\n";

SOCKET socket;

while (true) {

socket = accept(listenSocket, nullptr, nullptr);

if (socket != INVALID\_SOCKET) {

connectionsInfo[socket] = "";

std::cout << "Client #" << socket << " connected.\n";

std::thread read\_thread(&receiveMessageFromClient, socket);

read\_thread.detach();

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(50));

}

for (auto& pair : connectionsInfo) {

closesocket(pair.first);

}

WSACleanup();

}

void setHints()

{

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

}

void clear(const std::string& message, addrinfo\* info, SOCKET\* socket)

{

std::cout << message;

if (info) {

freeaddrinfo(info);

}

if (socket) {

closesocket(\*socket);

}

}

void receiveMessageFromClient(SOCKET client)

{

char receivebuffer[256];

ZeroMemory(receivebuffer, 256);

while (true) {

ZeroMemory(receivebuffer, 256);

if (recv(client, receivebuffer, 256, 0) > 0) {

if (connectionsInfo[client] == "") {

connectionsInfo[client] = receivebuffer;

std::cout << "User " << connectionsInfo[client] << " authorized\n";

}

else {

std::stringstream ss;

ss << connectionsInfo[client] << ':' << receivebuffer << '\n';

std::cout << ss.str();

sendMessageToClients(ss.str().c\_str(), client);

}

}

else {

std::cout << "Connection closed or error occured when receiving from " << connectionsInfo[client] << "\n";

return;

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(50));

}

}

void sendMessageToClients(const char\* message, SOCKET from)

{

for (auto& pair : connectionsInfo) {

if (pair.first != from) {

send(pair.first, message, strlen(message), 0);

}

}

}