Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и

системное программирование

ОТЧЁТ

к лабораторной работе №8

на тему

Итнтерфейс сокетов и основы сетевого программирования (Windows). Программирование взаимодействия через сеть с использованием интерфейса сокетов. Реализация сетевых протоколов: собственных или стандартных

Выполнила: студентка группы 153502

Шайкова Вероника Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc152903895)

[2 Теоретические сведения 4](#_Toc152903896)

[3 Результаты выполнения лабораторной работы 5](#_Toc152903897)

[Выводы 6](#_Toc152903898)

[Список использованных источников 7](#_Toc152903899)

[Приложение А](#_Toc152903900) [(обязательное)](#_Toc152903901) [Листинг кода 8](#_Toc152903902)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание клиент-серверного приложения на Win32 Api, позволяющее клиенту и серверу обмениваться сообщениями в рамках локальной сети, используя сокеты с протоколом TCP.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

TCP расшифровывается как протокол управления передачей. Это тяжеловесный интернет-протокол, который является более стабильным и надежным в современном онлайн-мире обеспечивает устойчивое, упорядоченное соединение между двумя точками в сети. Применяется в ситуациях, где важна доставка данных без потерь и в правильном порядке.

UDP расшифровывается как User Datagram Protocol. Это облегченный протокол, широко используемый в областях, связанных со средствами массовой информации. Он может обнаруживать поврежденные данные в пакетах данных, но не решает такие проблемы, как потерянные или неупорядоченные пакеты данных. Это причина, по которой UDP также известен как ненадежный протокол передачи данных. Несмотря на то, что он ненадежен, он по-прежнему используется для приложений, требующих много времени, благодаря своей скорости, таких как стриминг и видеоконференции, где скорость важнее точности. [1]

Для работы с сокетами на Win32 Api используется интерфейс Winsock (библиотека winsock2). Работа может быть осуществлена при помощи следующих функций:

WSAStartup ─ инициализация Winsock, вызов запуска сокета.

WSACleanup ─ деинициализация сокетов.

WSAGetLastError ─ возвращает статус ошибки последней неудачной операции Winsock.

Socket ─ создает сокет с указанием типа сокета и протокола.

Closesocket ─ закрытие использованого ранее сокета.

Getaddrinfo ─ передает положение сервиса или его имя и возвращает набор адресов сокетов и другую информацию, необходимую для создания сокета.

Freeaddrinfo ─ освобождает адресную информацию, которую выделяет функция getaddrinfo в структурах addrinfo.

Recv ─ получает данные из подключенного сокета или связанного сокета без подключения. [2]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было создано два консольных приложения, позволяющих производить обмен сообщениями между клиентом (рисунок1) и сервером (рисунок 2).

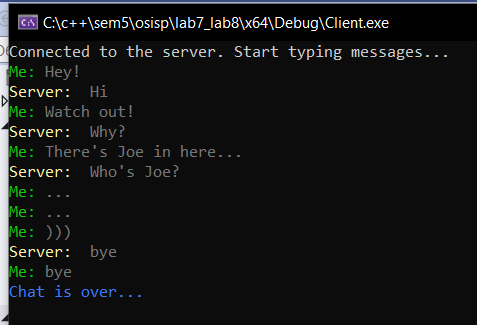


Рисунок 1 ─ чат со стороны клиента

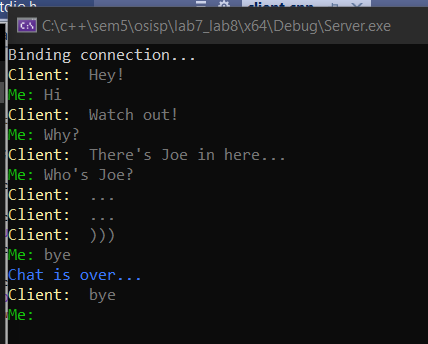


Рисунок 2 ─ чат со стороны сервера

Приложение-сервер использует сокет для установки соединения и сокет для обмена данными с клиентом. Обмен реализован многопоточно, потому в любой момент времени (до завершения чата) доступны как получение, так и отправка сообщения.

Приложение-клиент использует сокет для подключения к серверу и аналогичного обмена данными.

## ВЫВОДЫ

В ходе выполнения лабораторной работы были созданы два консольных приложения: сервер и клиент, которые способны обмениваться сообщениями. Был использован интерфейс сокетов, производящих обмен данными по протоколу TCP.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] TCP vs UDP: Everything you need to know [Электронный ресурс]. ─ Режим доступа: https://humandigitalagency.medium.com/tcp-vs-udp-everything-you-need-to-know-d2a58c8bb105 ─ Дата доступа 05.12.2023

[2] Использование Winsock [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/winsock/using-winsock ─ Дата доступа 04.12.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**server.h**

#pragma once

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdio.h>

#include <thread>

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

#define DEFAULT\_BUFLEN 1000

#define DEFAULT\_PORT "27015"

SOCKET \_socket;

SOCKET \_client\_sock;

HANDLE console\_handle;

WSADATA wsaData;

WORD wgray = 8;

WORD wred = 12;

WORD wgreen = 10;

WORD wyellow = 14;

WORD wwhite = 15;

WORD wblue = 9;

**server.cpp**

#include "server.h"

#include <string>

void log\_error(std::string message) {

SetConsoleTextAttribute(console\_handle, wred);

printf(("\n" + message).c\_str());

SetConsoleTextAttribute(console\_handle, wwhite);

}

void print\_client\_message(char\* message, int res) {

SetConsoleTextAttribute(console\_handle, wyellow);

printf("\rClient:\t");

SetConsoleTextAttribute(console\_handle, wgray);

printf(" %.\*s", res, message);

SetConsoleTextAttribute(console\_handle, wgreen);

printf("Me: ");

SetConsoleTextAttribute(console\_handle, wgray);

}

void get\_message(SOCKET ClientSocket) {

char recvbuf[DEFAULT\_BUFLEN];

int response\_length;

while (true) {

response\_length = recv(ClientSocket, recvbuf, DEFAULT\_BUFLEN, 0);

if (response\_length > 0) {

print\_client\_message(recvbuf, response\_length);

}

else if (response\_length == 0) {

closesocket(ClientSocket);

SetConsoleTextAttribute(console\_handle, wblue);

printf("Client is off...\n");

SetConsoleTextAttribute(console\_handle, wwhite);

break;

}

else {

closesocket(ClientSocket);

log\_error("Error while receiving a message: " + std::to\_string(WSAGetLastError()));

break;

}

};

}

int main() {

console\_handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

int response\_length;

struct addrinfo\* result = NULL;

struct addrinfo hints;

response\_length = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (response\_length != 0) {

log\_error("Error occured while WSA starting: " + std::to\_string(response\_length));

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

response\_length = getaddrinfo(NULL, DEFAULT\_PORT, &hints, &result);

if (response\_length != 0) {

WSACleanup();

log\_error("Error occured while getting addres: " + std::to\_string(response\_length));

return 1;

}

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

if (\_socket == INVALID\_SOCKET) {

log\_error("Invalid Socket: " + std::to\_string(WSAGetLastError()));

freeaddrinfo(result);

WSACleanup();

return 1;

}

response\_length = bind(\_socket, result->ai\_addr, (int)result->ai\_addrlen);

if (response\_length == SOCKET\_ERROR) {

log\_error("Error while binding with the client: " + std::to\_string(WSAGetLastError()));

freeaddrinfo(result);

closesocket(\_socket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

response\_length = listen(\_socket, SOMAXCONN);

if (response\_length == SOCKET\_ERROR) {

log\_error("Error while listening to client: " + std::to\_string(WSAGetLastError()));

closesocket(\_socket);

WSACleanup();

return 1;

}

printf("Binding connection...\n");

\_client\_sock = accept(\_socket, NULL, NULL);

if (\_client\_sock == INVALID\_SOCKET) {

log\_error("Error accepting message: " + std::to\_string(WSAGetLastError()));

closesocket(\_socket);

WSACleanup();

return 1;

}

closesocket(\_socket);

std::thread receiveThread(get\_message, \_client\_sock);

char sendbuf[DEFAULT\_BUFLEN];

while (true) {

SetConsoleTextAttribute(console\_handle, wgreen);

printf("Me: ");

SetConsoleTextAttribute(console\_handle, wgray);

fgets(sendbuf, DEFAULT\_BUFLEN, stdin);

SetConsoleTextAttribute(console\_handle, wwhite);

response\_length = send(\_client\_sock, sendbuf, strlen(sendbuf), 0);

if (response\_length == SOCKET\_ERROR) {

printf("send failed with error: %d\n", WSAGetLastError());

break;

}

if (strcmp(sendbuf, "bye\n") == 0) {

SetConsoleTextAttribute(console\_handle, wblue);

printf("Chat is over...\n");

SetConsoleTextAttribute(console\_handle, wwhite);

//closesocket(\_client\_sock);

break;

}

};

receiveThread.join();

closesocket(\_client\_sock);

WSACleanup();

return 0;

}

**client.h**

#pragma once

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdio.h>

#include <thread>

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

#define DEFAULT\_BUFLEN 1000

#define DEFAULT\_PORT "27015"

HANDLE console\_handle;

WSADATA wsaData;

SOCKET \_socket;

WORD wgray = 8;

WORD wred = 12;

WORD wgreen = 10;

WORD wyellow = 14;

WORD wwhite = 15;

WORD wblue = 9;

**client.cpp**

#include "client.h";

#include <string>

void log\_error(std::string message) {

SetConsoleTextAttribute(console\_handle, wred);

printf(("\n"+message).c\_str());

SetConsoleTextAttribute(console\_handle, wwhite);

}

void print\_server\_message(char\* message, int res) {

SetConsoleTextAttribute(console\_handle, wyellow);

printf("\rServer:\t");

SetConsoleTextAttribute(console\_handle, wgray);

printf(" %.\*s",res,message);

SetConsoleTextAttribute(console\_handle, wgreen);

printf("Me: ");

SetConsoleTextAttribute(console\_handle, wgray);

}

void get\_message(SOCKET ConnectSocket) {

char recvbuf[DEFAULT\_BUFLEN];

int response\_length;

while (true) {

response\_length = recv(ConnectSocket, recvbuf, DEFAULT\_BUFLEN, 0);

if (response\_length > 0) {

print\_server\_message(recvbuf,response\_length);

}

else if (response\_length == 0) {

closesocket(ConnectSocket);

SetConsoleTextAttribute(console\_handle, wblue);

printf("Server is off...\n");

SetConsoleTextAttribute(console\_handle, wwhite);

break;

}

else {

closesocket(ConnectSocket);

log\_error("Error while receiving a message: " + std::to\_string(WSAGetLastError()));

break;

}

};

}

int main() {

int response\_length;

struct addrinfo\* result = NULL, \*ptr = NULL, hints;

console\_handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

response\_length = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (response\_length != 0) {

log\_error("Error occured while WSA starting: " + std::to\_string(response\_length));

//closesocket(\_socket);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

response\_length = getaddrinfo("localhost", DEFAULT\_PORT, &hints, &result);

if (response\_length != 0) {

log\_error("Error occured while getting addres: " + std::to\_string(response\_length));

//closesocket(\_socket);

WSACleanup();

return 1;

}

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

\_socket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (\_socket == INVALID\_SOCKET) {

log\_error("Invalid Socket: " + std::to\_string(WSAGetLastError()));

//closesocket(\_socket);

WSACleanup();

return 1;

}

response\_length = connect(\_socket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (response\_length == SOCKET\_ERROR) {

closesocket(\_socket);

\_socket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (\_socket == INVALID\_SOCKET) {

WSACleanup();

//closesocket(\_socket);

log\_error("Unable to connect to server!\n");

return 1;

}

printf("Connected to the server. Start typing messages...\n");

std::thread receiveThread(get\_message, \_socket);

char sendbuf[DEFAULT\_BUFLEN];

while(true) {

SetConsoleTextAttribute(console\_handle, wgreen);

printf("Me: ");

SetConsoleTextAttribute(console\_handle, wgray);

fgets(sendbuf, DEFAULT\_BUFLEN, stdin);

SetConsoleTextAttribute(console\_handle, wwhite);

response\_length = send(\_socket, sendbuf, strlen(sendbuf), 0);

if (response\_length == SOCKET\_ERROR) {

log\_error("send failed with error: %d\n" + std::to\_string(WSAGetLastError()));

WSACleanup();

//closesocket(\_socket);

break;

}

if (strcmp(sendbuf, "bye\n") == 0) {

SetConsoleTextAttribute(console\_handle, wblue);

printf("Chat is over...\n");

SetConsoleTextAttribute(console\_handle, wwhite);

//closesocket(\_socket);

break;

}

};

receiveThread.join();

closesocket(\_socket);

WSACleanup();

return 0;

}