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

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

Выполнил: студент группы 153503

Жечко Иван Сергеевич

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[3 Результаты выполнения лабораторной работы 5](#_Toc146728179)

[Заключение 7](#_Toc146728180)

[Список использованных источников 8](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 9](#_Toc146728182)

## 1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API* использующего сокеты *TCP* для передачи данных.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Веб-сокеты - это протокол связи поверх *TCP*-соединения, который позволяет обмениваться сообщениями между браузером и веб-сервером в режиме реального времени. Они позволяют клиенту и серверу инициировать связь друг с другом и отправлять сообщения одновременно. Протокол *WebSocke*t был стандартизирован *IETF* как *RFC* 6455 в 2011 году. Веб-сокеты могут передавать текстовые сообщения, изображения, аудио- и видеофайлы, потоковые данные. Протокол веб-сокет может одновременно и получать, и передавать информацию, так как является двунаправленным (полнодуплексная связь). Веб-сокеты идеально подходят для случаев, когда необходимо обновление данных в режиме реального времени и возможность отправлять сообщения клиенту. Реализация сокетов на *Windows* (*Winsock*) также может использовать *UDP*-протоколы и является более гибкой чем обычный протокол веб-сокетов.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Были реализованы клиент и сервер для текстового чата. Сервер представляет собой консольное приложение, которое выводит информацию о своем состоянии в консоль. Консоль сервера изображена на рисунке 3.1.



Рисунок 3.1 Консоль сервера

Клиентское приложение представляет собой оконное приложение *Win 32*, в котором можно подключиться к серверу и вводить сообщения в чат. На рисунке 3.2 изображено клиентское приложение до подключения к серверу.

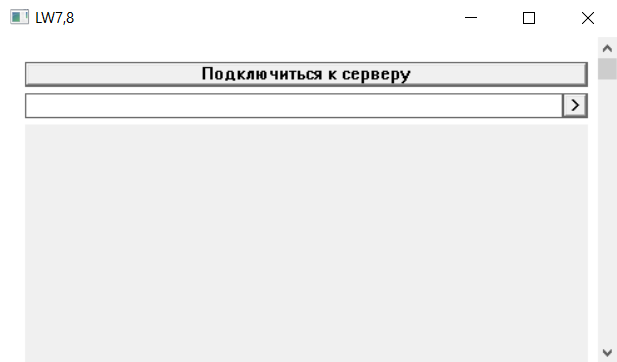


Рисунок 3.2 Клиентское приложение

По нажатию на кнопку подключения к серверу устанавливается соединение с сервером через веб-сокеты. При удачном подключении сервер передает содержимое чата с момента его создания. На рисунке 3.3 изображено клиентское приложение, которое успешно загрузило чат с сервера.

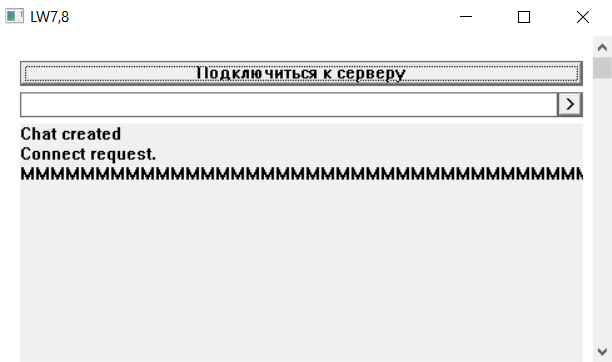


Рисунок 3.3 Приложение с загруженным чатом

При отправке сообщения либо подключения нового клиента, у всех клиентов перезагружается чат с новой информацией. На рисунке 3.4 изображен чат с сообщениями от нескольких пользователей.

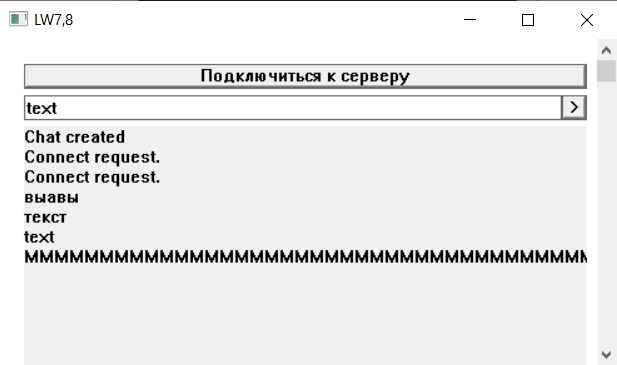


Рисунок 3.4 Приложение с сообщениями пользователей

## ЗАКЛЮЧЕНИЕ

В результате выполнения работы были созданы приложение-сервер и приложение-клиент для текстового чата. Данные приложения используют технологию веб-сокетов для обмена данными между собой в пределах локальной сети.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Programming reference for the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/ – Дата доступа: 07.12.2023.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <winsock2.h>

#include <ws2tcpip.h>

#include <windows.h>

#include <websocket.h>

#include <winhttp.h>

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

#pragma comment(lib, "websocket.lib")

#pragma comment(lib, "winhttp.lib")

#include <tchar.h>

#include <cstring>

#include <string>

#include <cstdlib>

#include <cwchar>

#include <iostream>

#include <thread>

using namespace std;

HWND mainWindow;

HWND hEdit;

HWND chatLabel;

SOCKET ConnectSocket = INVALID\_SOCKET;

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

auto scrollValue = HIWORD(GetDialogBaseUnits());

auto yScrollPos = 0;

void ServerHandler(SOCKET ClientSocket) { //аргумент не используется, нужен из-за отсутствия перегрузки в библиотеке

char buffer[1024];

int bytes;

while (true) {

bytes = recv(ConnectSocket, buffer, sizeof(buffer), 0);

if (bytes <=0) {

continue;

}

wchar\_t wdata[1024];

size\_t converted = 0;

mbstowcs(wdata, buffer, 1024);

std::wstring data(wdata, 1024);

SetWindowText(chatLabel, data.c\_str());

}

closesocket(ConnectSocket);

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

// Window class.

const wchar\_t CLASS\_NAME[] = L"Window";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

const int window\_x = 510;

const int window\_y = 300;

// Window creation

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"LW7,8",

WS\_OVERLAPPEDWINDOW | WS\_VSCROLL,

CW\_USEDEFAULT, CW\_USEDEFAULT, window\_x, window\_y,

NULL,

NULL,

hInstance,

NULL

);

mainWindow = hwnd;

auto connectButton = CreateWindow(

L"BUTTON",

L"Подключиться к серверу",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

20,

20,

450,

20,

hwnd,

(HMENU)1, //id

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL);

auto sendButton = CreateWindow(

L"BUTTON",

L">",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

450,

45,

20,

20,

hwnd,

(HMENU)2, //id

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL);

hEdit = CreateWindowEx(0,

L"EDIT",

NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT,

20, 45, 430, 20,

hwnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL);

chatLabel = CreateWindowEx(0, L"STATIC", L"", WS\_CHILD | WS\_VISIBLE, 20, 70, 450, 5000, hwnd, NULL, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

if (hwnd == NULL)

{

return 0;

}

ShowWindow(hwnd, nCmdShow);

// Message loop.

MSG msg = {};

while (GetMessage(&msg, NULL, 0, 0) > 0)

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_DESTROY: {

PostQuitMessage(0);

return 0;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

HBRUSH brush = CreateSolidBrush(

RGB(255, 255, 255)

);

FillRect(hdc, &ps.rcPaint, brush); //фон

EndPaint(hwnd, &ps);

return 0;

}

case WM\_CREATE:

{

setlocale(LC\_ALL, "");

return 0;

}

case WM\_VSCROLL: { //обработка скролла

int dir = 0;

switch (LOWORD(wParam)) {

case SB\_LINEUP:

yScrollPos -= scrollValue;

dir = 1;

break;

case SB\_LINEDOWN:

yScrollPos += scrollValue;

dir = -1;

break;

case SB\_THUMBPOSITION: {

yScrollPos = HIWORD(wParam);

auto curPos = LOWORD(GetScrollPos(mainWindow, SB\_VERT));

SetScrollPos(mainWindow, SB\_VERT, yScrollPos, TRUE);

ScrollWindow(mainWindow, 0, scrollValue \* (curPos - yScrollPos ), NULL, NULL);

return 0;

}

default:

break;

}

SetScrollPos(mainWindow, SB\_VERT, yScrollPos, TRUE);

ScrollWindow(mainWindow, 0, scrollValue \* dir, NULL, NULL);

return 0;

}

case WM\_COMMAND: { //button click

if (LOWORD(wParam) == 1){ //подключенние к серверу

if (ConnectSocket != INVALID\_SOCKET) {

MessageBox(NULL, L"Подключение завершено", L"", MB\_OK);

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

return 0;

}

// Закрытие сокета

//сокет для сервера

struct addrinfo\* result = NULL, //адрес сервера

\* ptr = NULL,

hints;

// Инициализация Winsock

WSADATA wsaData;

int iResult;

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

MessageBox(NULL, L"Ошибка при инициализации Winsock", L"Ошибка", MB\_OK);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Разрешение адреса сервера и порта

iResult = getaddrinfo("localhost", "8888", &hints, &result);

if (iResult != 0) {

MessageBox(NULL, L"Ошибка при разрешении адреса сервера", L"Ошибка", MB\_OK);

WSACleanup();

return 1;

}

// Попытка подключения к серверу

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

// Создание сокета для подключения к серверу

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

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

MessageBox(NULL, L"Ошибка при создании сокета ", L"Ошибка", MB\_OK);

WSACleanup();

return 1;

}

// Подключение к серверу

iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (iResult == SOCKET\_ERROR) { //неудачное подключение

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

MessageBox(NULL, L"Не удалось подключиться к серверу", L"Ошибка", MB\_OK);

WSACleanup();

return 1;

}

// Отправка данных на сервер

const wchar\_t\* wsendbuf = L"Connect request.\0";

char sendbuf[1024];

wcstombs(sendbuf, wsendbuf, 1024);

iResult = send(ConnectSocket, sendbuf, 1024, 0);

if (iResult == SOCKET\_ERROR) {

MessageBox(NULL, L"Ошибка при отправке данных на сервер", L"Ошибка", MB\_OK);

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

// Получение данных от сервера

char recvbuf[1024];

iResult = recv(ConnectSocket, recvbuf, 1024, 0);

wchar\_t wdata[1024];

size\_t converted = 0;

mbstowcs(wdata, recvbuf, 1024);

std::wstring data(wdata, 1024);

if (iResult > 0) {

// MessageBox(NULL, L"Соединение успешно", L"Данные получены", MB\_OK);

// SetWindowText(chatLabel, data.c\_str());

thread(ServerHandler, ConnectSocket).detach(); //запуск цикла проверки сокета

}

else if (iResult == 0) {

MessageBox(NULL, L"Соединение закрыто сервером", L"Ошибка", MB\_OK);

}

else {

MessageBox(NULL, L"Ошибка при получении данных от сервера", L"Ошибка", MB\_OK);

}

}

else if (LOWORD(wParam) == 2) { //отправка данных

if (ConnectSocket == INVALID\_SOCKET) {

MessageBox(NULL, L"Соединение не было установлено", L"Ошибка", MB\_OK);

}

else {

wchar\_t wsendbuf[1024];

char sendbuf[1024];

GetWindowText((HWND)hEdit, wsendbuf, 1024);

wcstombs(sendbuf, wsendbuf, 1024);

auto iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf) + 1, 0);

}

}

return 0;

}

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}