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

ОТЧЕТ

к лабораторной работе №7

на тему

**СРЕДСТВА ОБМЕНА ДАННЫМИ (WINDOWS).**

**ИЗУЧЕНИЕ И ИСПОЛЬЗОВАНИЕМ СРЕДСТВ**

**ОБМЕНА ДАННЫМИ И СОВМЕСТНОГО ДОСТУПА.**

Студент В. Н. Степанов

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Общие цели лабораторной работы:

1. Изучение и освоение основ работы с сетевыми протоколами и технологиями обмена данными.
2. Получение практических навыков разработки приложений для обмена данными по сети.
3. Развитие навыков работы с различными инструментами и технологиями программирования.
4. Самостоятельное выполнение задания – создать приложение для обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов.
5. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Локальная сеть – это сеть, объединяющая компьютеры и другие устройства, расположенные на небольшом расстоянии друг от друга, например, в одном здании или помещении. Локальные сети используются для обмена информацией между компьютерами, совместного использования ресурсов, таких как принтеры и файлы, а также для доступа к интернету.

Существует несколько типов локальных сетей, включая проводные (Ethernet) и беспроводные (*Wi-Fi*). *Ethernet* является наиболее распространенным типом проводной локальной сети и использует кабели для соединения компьютеров. *Wi-Fi* позволяет создавать беспроводные локальные сети без использования кабелей, но требует наличия точки доступа или маршрутизатора [1].

Исследование и применение методов обмена данными и совместного доступа включает в себя изучение различных методов и технологий, которые позволяют разным системам и приложениям обмениваться информацией и использовать общие ресурсы.

Сетевое программирование является важной частью разработки программного обеспечения, поскольку позволяет создавать приложения, которые могут работать в сети. Это включает создание протоколов, структур данных и алгоритмов для обмена информацией между компьютерами, серверами и другими устройствами в сети.

Модель *OSI* (*Open Systems Interconnection*) является базовой моделью для описания и понимания работы сетевых протоколов. Она состоит из семи уровней: физический, канальный, сетевой, транспортный, сеансовый, уровень представления и прикладной. Каждый уровень выполняет свою функцию и взаимодействует с соседними уровнями для обеспечения работы протокола.

Протоколы *TCP* и *UDP* являются двумя основными протоколами транспортного уровня, которые используются для передачи данных в сетях.

*TCP (Transmission Control Protocol)* – это протокол транспортного уровня, который обеспечивает надежную передачу данных между двумя точками в сети. Он гарантирует, что все данные будут доставлены в правильном порядке, и обеспечивает подтверждение доставки.

*UDP (User Datagram Protocol)* – это другой протокол транспортного уровня, который не гарантирует доставку данных, но обеспечивает более быструю передачу данных. UDP используется в приложениях, где требуется низкая задержка, таких как потоковое видео [2].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы было создано два приложения: сервер и клиент.

Реализация серверного приложения:

1. Создан серверный сокет, привязанный к определенному порту, ожидающий подключений от клиентов.
2. Сокеты клиентов хранятся на стороне клиента до отправки в определённой структуре, после чего отправляются по на сервер и распределятся в такую же структуру.

Реализация клиентского приложения (рисунок q):

1. При нажатии на соответствующую опцию в меню открывается список клиентов, полученный с сервера.
2. Выполнено подключение к серверу по указанному *IP*-адресу и порту.
3. Реализована отправка нескольких сообщений на сервер.
4. Добавлена возможность выборки текстового файла и отправки его в виде пакетов, второй клиент получает уведомление о том, что сообщение пришло и выбирает куда его сохранить.

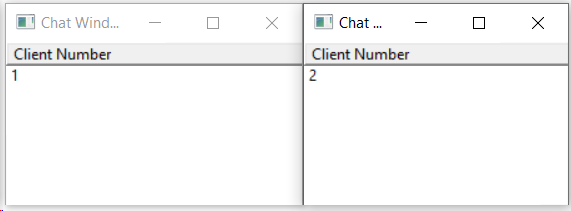


Рисунок 2 – Два запущенных клиента

# ЗАКЛЮЧЕНИЕ

Цели лабораторной работы были достигнуты путем изучения основ работы с сетевыми протоколами и технологиями обмена данными, получения практических навыков разработки приложений для обмена данными по сети и развития навыков работы с инструментами и технологиями программирования. Самостоятельное выполнение задания и решение возникающих вопросов позволило успешно освоить основы сетевого программирования и получить необходимые знания для дальнейшей разработки сетевых приложений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Локальная вычислительная сеть [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Локальная\_вычислительная\_сеть.
2. Сетевая модель OSI [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Сетевая\_модель\_OSI.
3. Использование Winsock [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/winsock/using-winsock>.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – source.cpp

#include "imports.h";

#define RICHEDIT\_CLASS L"RICHEDIT50W"

#define IDM\_OPEN 1001

#define IDM\_SAVE 1002

#define IDM\_EXIT 1003

#define IDM\_NEW 1004

#define IDC\_CLOSE\_BUTTON 1005

#define IDM\_SEARCH 1006

#define IDM\_SEARCH\_IN\_SEARCH\_WINDOW 1008

#define IDM\_CHANGE\_THEME 1009

#define IDM\_CHANGE\_FONT 1010

#define IDM\_OPEN\_BINARY 1011

#define IDM\_SAVE\_BINARY 1012

#define IDC\_START\_COUNT 2

#define IDC\_WORD\_LIST 3

#define WM\_UPDATE\_WORD\_COUNT 6

#define IDC\_STOP\_COUNT 5

#define HOTKEY\_ID 1

#define IDM\_REGISTRY\_ACCESS 1999

#define IDC\_READ\_BUTTON 2999

#define IDC\_RESULT\_EDIT 3999

#define IDC\_CHAT\_LIST 3991

#define OPEN\_CHAT 4999

#define IDC\_LISTVIEW 5999

#define IDC\_SEND\_FILE\_BUTTON 5991

#define IDM\_SEND\_FILE 5191

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK SearchWindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

static int wordsCount = 0;

static HWND hEdit;

static HWND hSearchEdit = nullptr;

static int searchStartPosition = 0;

static HWND hSearchWindow = nullptr;

HWND g\_hwnd;

std::vector<char> g\_content;

HANDLE g\_hFileMapping;

LPVOID g\_pFileData;

DWORD g\_fileSize;

bool g\_bIsWordCountThreadRunning = true;

HANDLE hThread = nullptr;

UINT\_PTR timerId;

HANDLE g\_hFileMutex;

HANDLE g\_hRegistryMutex;

CRITICAL\_SECTION g\_csWordCount;

WSADATA wsaData;

sockaddr\_in serverAddr;

HWND hwndChatWindow;

HWND hChatsListBox;

HANDLE chatThread;

HWND g\_hWndSettings;

SOCKET g\_socket = 0;

enum class Operation

{

Send,

Update,

Close,

};

struct Package {

Operation operation;

int data = -1;

int receiverId = -1;

};

LRESULT CALLBACK RegistryAccessWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

CreateWindow(L"BUTTON", L"Чтение", WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 10, 80, 30, hWnd, (HMENU)IDC\_READ\_BUTTON, GetModuleHandle(NULL), NULL);

CreateWindow(L"EDIT", L"", WS\_VISIBLE | WS\_CHILD | ES\_READONLY | WS\_BORDER,

100, 10, 200, 30, hWnd, (HMENU)IDC\_RESULT\_EDIT, GetModuleHandle(NULL), NULL);

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDC\_READ\_BUTTON:

{

HKEY hKey;

LONG result = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\YourRegistryKey", 0, KEY\_READ, &hKey);

if (result == ERROR\_SUCCESS)

{

WCHAR valueData[256];

DWORD valueSize = sizeof(valueData);

result = RegQueryValueEx(hKey, L"YourRegistryValue", 0, NULL, (LPBYTE)valueData, &valueSize);

if (result == ERROR\_SUCCESS)

{

SetDlgItemText(hWnd, IDC\_RESULT\_EDIT, valueData);

}

else

{

SetDlgItemText(hWnd, IDC\_RESULT\_EDIT, L"Ошибка чтения значения из реестра");

}

RegCloseKey(hKey);

}

else

{

SetDlgItemText(hWnd, IDC\_RESULT\_EDIT, L"Ошибка открытия ключа в реестре");

}

break;

}

}

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

void LoadFontSettings()

{

WaitForSingleObject(g\_hRegistryMutex, INFINITE);

HKEY hKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software\\322EDITOR322\\FontSettings", 0, KEY\_READ, &hKey) == ERROR\_SUCCESS)

{

LOGFONT lf = { 0 };

DWORD dataSize;

DWORD fontUnderline, fontStrikeOut, fontItalic, fontBold;

dataSize = sizeof(DWORD);

RegQueryValueEx(hKey, L"FontUnderline", 0, NULL, (LPBYTE)&fontUnderline, &dataSize);

dataSize = sizeof(DWORD);

RegQueryValueEx(hKey, L"FontStrikeOut", 0, NULL, (LPBYTE)&fontStrikeOut, &dataSize);

dataSize = sizeof(DWORD);

RegQueryValueEx(hKey, L"FontItalic", 0, NULL, (LPBYTE)&fontItalic, &dataSize);

dataSize = sizeof(DWORD);

RegQueryValueEx(hKey, L"FontBold", 0, NULL, (LPBYTE)&fontBold, &dataSize);

dataSize = sizeof(wchar\_t) \* MAX\_PATH;

RegQueryValueEx(hKey, L"FontName", 0, NULL, (LPBYTE)lf.lfFaceName, &dataSize);

dataSize = sizeof(DWORD);

RegQueryValueEx(hKey, L"FontSize", 0, NULL, (LPBYTE)&lf.lfHeight, &dataSize);

RegCloseKey(hKey);

lf.lfUnderline = fontUnderline;

lf.lfStrikeOut = fontStrikeOut;

lf.lfItalic = fontItalic;

lf.lfWeight = fontBold;

HFONT hFont = CreateFontIndirect(&lf);

SendMessage(hEdit, WM\_SETFONT, (WPARAM)hFont, TRUE);

}

ReleaseMutex(g\_hRegistryMutex);

}

void LoadColorSettings()

{

WaitForSingleObject(g\_hRegistryMutex, INFINITE);

HKEY hKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software\\322EDITOR322\\ColorSettings", 0, KEY\_READ, &hKey) == ERROR\_SUCCESS)

{

DWORD dataSize;

DWORD backgroundColor;

dataSize = sizeof(DWORD);

RegQueryValueEx(hKey, L"BackgroundColor", 0, NULL, (LPBYTE)&backgroundColor, &dataSize);

RegCloseKey(hKey);

SendMessage(hEdit, EM\_SETBKGNDCOLOR, FALSE, backgroundColor);

}

ReleaseMutex(g\_hRegistryMutex);

}

struct ThreadParams {

HWND hWnd;

};

void OpenSearchWindow(HWND hWnd)

{

hSearchWindow = CreateWindow(

L"SearchWindowClass",

L"Search Window",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 200,

nullptr,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

if (hSearchWindow != nullptr)

{

hSearchEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 10, 150, 30,

hSearchWindow,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

HWND hSearchButton = CreateWindow(

L"BUTTON",

L"Search",

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

170, 10, 80, 30,

hSearchWindow,

reinterpret\_cast<HMENU>(IDM\_SEARCH\_IN\_SEARCH\_WINDOW),

GetModuleHandle(nullptr),

nullptr

);

if (hSearchEdit == nullptr || hSearchButton == nullptr)

{

MessageBox(hWnd, L"Не удалось создать элементы для поиска.", L"Ошибка", MB\_OK | MB\_ICONERROR);

DestroyWindow(hSearchWindow);

}

ShowWindow(hSearchWindow, SW\_SHOWNORMAL);

}

else

{

MessageBox(hWnd, L"Не удалось создать окно поиска.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

DWORD WINAPI WordCountThread(LPVOID lpParam) {

ThreadParams\* params = reinterpret\_cast<ThreadParams\*>(lpParam);

HWND hWnd = params->hWnd;

while (g\_bIsWordCountThreadRunning) {

int textLength = GetWindowTextLength(hEdit);

if (textLength > 0) {

std::wstring text;

text.resize(textLength + 1);

GetWindowText(hEdit, &text[0], textLength + 1);

int wordCount = 0;

bool inWord = false;

EnterCriticalSection(&g\_csWordCount);

for (wchar\_t c : text) {

if (iswspace(c)) {

inWord = false;

}

else {

if (!inWord) {

wordCount++;

inWord = true;

}

}

}

wordsCount = wordCount;

LeaveCriticalSection(&g\_csWordCount);

if (hWnd) {

SendMessage(hWnd, WM\_UPDATE\_WORD\_COUNT, 0, 0);

}

}

Sleep(1000);

}

return 0;

}

void OnHotkey(HWND hWnd, WPARAM wParam, LPARAM lParam)

{

if (wParam == HOTKEY\_ID)

{

OpenSearchWindow(hWnd);

}

}

void SaveColorSettings(COLORREF color)

{

HKEY hKey;

if (RegCreateKeyEx(HKEY\_CURRENT\_USER, L"Software\\322EDITOR322\\ColorSettings", 0, NULL, 0, KEY\_WRITE, NULL, &hKey, NULL) == ERROR\_SUCCESS)

{

RegSetValueEx(hKey, L"BackgroundColor", 0, REG\_DWORD, (LPBYTE)&color, sizeof(DWORD));

RegCloseKey(hKey);

}

}

void OpenColorDialog()

{

CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };

static COLORREF custColors[16] = { 0 };

cc.rgbResult = RGB(255, 255, 255);

cc.lpCustColors = custColors;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc))

{

SendMessage(hEdit, EM\_SETBKGNDCOLOR, FALSE, cc.rgbResult);

SaveColorSettings(cc.rgbResult);

}

}

LRESULT CALLBACK ChatProc(HWND hWindow, UINT msg, WPARAM wParam, LPARAM lParam)

{

switch (msg)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case WM\_CLOSE:

if (hWindow == g\_hWndSettings) {

Package p = {Operation::Close, 0, 0 };

send(g\_socket, reinterpret\_cast<char\*>(&p), sizeof(Package), NULL);

closesocket(g\_socket);

DestroyWindow(hWindow);

}

break;

}

break;

case WM\_DESTROY:

if (hWindow == g\_hWndSettings) {

g\_hWndSettings = nullptr;

}

break;

case WM\_NOTIFY:

{

NMHDR\* nmh = reinterpret\_cast<NMHDR\*>(lParam);

if (nmh->code == NM\_DBLCLK) {

NMLISTVIEW\* pNmlv = reinterpret\_cast<NMLISTVIEW\*>(nmh);

int selectedIndex = pNmlv->iItem;

WCHAR buffer[MAX\_PATH];

LVITEM lvItem;

lvItem.iItem = selectedIndex;

lvItem.iSubItem = 0;

lvItem.pszText = buffer;

lvItem.cchTextMax = MAX\_PATH;

SendMessage(pNmlv->hdr.hwndFrom, LVM\_GETITEMTEXT, selectedIndex, reinterpret\_cast<LPARAM>(&lvItem));

int itemId = std::stoi(buffer);

if (itemId != -1) {

OPENFILENAME ofn;

wchar\_t szFileName[1024] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

ofn.lpstrFilter = L"Text Files\0\*.txt\0All Files\0\*.\*\0";

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetOpenFileName(&ofn) == TRUE)

{

HANDLE hFile = CreateFileW(szFileName, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

DWORD fSize = GetFileSize(hFile, NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

if (fSize == INVALID\_FILE\_SIZE && GetLastError() != NO\_ERROR) {

CloseHandle(hFile);

return false;

}

g\_content.resize(fSize + 1);

DWORD bytesRead;

if (ReadFile(hFile, g\_content.data(), fSize, &bytesRead, NULL)) {

g\_content[bytesRead] = '\0';

}

else {

return false;

}

CloseHandle(hFile);

}

else {

return false;

}

std::wstring name = szFileName;

name = name.substr(name.find\_last\_of('\\') + 1);

LPWSTR fName = const\_cast<LPWSTR>(name.data());

name = fName;

Package pack{ Operation::Send, name.size(), itemId };

send(g\_socket, reinterpret\_cast<char\*>(&pack), sizeof(Package), NULL);

send(g\_socket, reinterpret\_cast<char\*>(const\_cast<wchar\_t\*>(name.c\_str())), name.size() \* sizeof(wchar\_t), NULL);

send(g\_socket, reinterpret\_cast<char\*>(&fSize), sizeof(int), NULL);

send(g\_socket, g\_content.data(), fSize \* sizeof(char), NULL);

}

}

}

else if (nmh->code == NM\_RCLICK) {

NMLISTVIEW\* pNmlv = reinterpret\_cast<NMLISTVIEW\*>(nmh);

int selectedItemIndex = pNmlv->iItem;

if (selectedItemIndex >= 0) {

POINT pt;

GetCursorPos(&pt);

HMENU hPopupMenu = CreatePopupMenu();

AppendMenu(hPopupMenu, MF\_STRING, IDM\_SEND\_FILE, L"Send File");

TrackPopupMenu(hPopupMenu, TPM\_LEFTALIGN | TPM\_TOPALIGN, pt.x, pt.y, 0, hwndChatWindow, NULL);

DestroyMenu(hPopupMenu);

}

}

break;

}

default:

return DefWindowProc(hWindow, msg, wParam, lParam);

}

return 0;

}

void ProcessChatThread()

{

int op;

int sz;

while (true)

{

Package p;

int bytes = recv(g\_socket, reinterpret\_cast<char\*>(&p), sizeof(Package), NULL);

switch (p.operation)

{

case Operation::Send:

{

std::wstring fName;

fName.resize(p.data \* sizeof(wchar\_t));

recv(g\_socket, reinterpret\_cast<char\*>(const\_cast<wchar\_t\*>(fName.data())), p.data \* sizeof(wchar\_t), NULL);

std::wstring msg = L"File received: " + fName;

int result = MessageBox(NULL, msg.c\_str(), L"File received", MB\_OKCANCEL | MB\_ICONQUESTION);

if (result == IDOK) {

BROWSEINFO bi = { 0 };

bi.hwndOwner = hwndChatWindow;

bi.lpszTitle = L"Choose folder:";

bi.ulFlags = BIF\_NEWDIALOGSTYLE | BIF\_RETURNONLYFSDIRS;

LPITEMIDLIST pidl = SHBrowseForFolder(&bi);

if (pidl != NULL)

{

wchar\_t folderPath[MAX\_PATH];

SHGetPathFromIDList(pidl, folderPath);

int size;

std::string data;

recv(g\_socket, reinterpret\_cast<char\*>(&size), sizeof(int), NULL);

data.resize(size);

recv(g\_socket, const\_cast<char\*>(data.data()), size \* sizeof(char), NULL);

std::wstring path = folderPath;

path += L"\\" + fName;

std::vector<char> cont(data.begin(), data.end());

HANDLE hFile = CreateFileW(path.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE)

{

OutputDebugString(L"Unable to open file");

}

LPDWORD byteRead = 0;

if (!WriteFile(hFile, cont.data(), size, byteRead, NULL))

{

OutputDebugString(L"Unable to async write");

CloseHandle(hFile);

break;

}

CloseHandle(hFile);

}

}

}

break;

case Operation::Update:

{

if (p.data > 0)

{

std::vector<int> receivedData;

int currentId;

recv(g\_socket, reinterpret\_cast<char\*>(&currentId), sizeof(int), 0);

std::vector<char> buffer(p.data \* sizeof(int));

int bytesRead = recv(g\_socket, buffer.data(), buffer.size(), 0);

if (bytesRead > 0) {

receivedData.resize(bytesRead / sizeof(int));

memcpy(receivedData.data(), buffer.data(), bytesRead);

}

else {

break;

}

if (hChatsListBox != NULL) {

SendMessage(hChatsListBox, LB\_RESETCONTENT, 0, 0);

}

for (size\_t i = 0; i < receivedData.size(); i++)

{

if (hChatsListBox != NULL && currentId != i) {

LVITEM lvi;

lvi.mask = LVIF\_TEXT;

lvi.iItem = i;

lvi.iSubItem = 0;

std::wstring text = std::to\_wstring(receivedData[i]);

lvi.pszText = const\_cast<LPWSTR>(text.c\_str());

ListView\_InsertItem(hChatsListBox, &lvi);

}

}

}

}

break;

default:

break;

}

}

}

void InitiateConnection(HWND windowHandle)

{

if (WSAStartup(MAKEWORD(2, 1), &wsaData) != 0) {

OutputDebugString(L"WSAStartup failed.");

return;

}

sockaddr\_in serverAddr;

int serverAddrSize = sizeof(serverAddr);

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(1111);

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

g\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (g\_socket == INVALID\_SOCKET) {

OutputDebugString(L"Failed to create socket.");

return;

}

if (connect(g\_socket, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) {

OutputDebugString(L"Connection failed.");

closesocket(g\_socket);

WSACleanup();

return;

}

hwndChatWindow = CreateWindow(L"ChatWindowClass", L"Chat Window", WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 600, nullptr, nullptr, nullptr, nullptr);

hChatsListBox = CreateWindowEx(0, WC\_LISTVIEW, NULL,

WS\_CHILD | WS\_VISIBLE | LVS\_REPORT | LVS\_EDITLABELS,

0, 0, 600, 600, hwndChatWindow, reinterpret\_cast<HMENU>(IDC\_CHAT\_LIST), NULL, NULL);

LVCOLUMN lvColumn;

lvColumn.mask = LVCF\_WIDTH | LVCF\_TEXT;

lvColumn.cx = 600;

lvColumn.pszText = const\_cast<LPWSTR>(L"Номер клиента");

ListView\_InsertColumn(hChatsListBox, 0, &lvColumn);

chatThread = CreateThread(NULL, NULL, reinterpret\_cast<LPTHREAD\_START\_ROUTINE>(ProcessChatThread), NULL, NULL, NULL);

ShowWindow(hwndChatWindow, SW\_SHOWNORMAL);

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR lpCmdLine, int nCmdShow)

{

g\_hFileMutex = CreateMutex(NULL, FALSE, L"FileMutex");

if (g\_hFileMutex == NULL) {

MessageBox(NULL, L"Не удалось создать мьютекс для ресурсов!", L"Error", MB\_ICONERROR | MB\_OK);

return 1;

}

g\_hRegistryMutex = CreateMutex(NULL, FALSE, L"RegistryMutex");

if (g\_hRegistryMutex == NULL) {

MessageBox(NULL, L"Не удалось создать мьютекс для реестра!", L"Error", MB\_ICONERROR | MB\_OK);

return 1;

}

InitializeCriticalSection(&g\_csWordCount);

HWND hWnd{};

MSG msg{};

WNDCLASS wc{ sizeof(WNDCLASS) };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

wc.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wc.hIcon = LoadIcon(nullptr, IDI\_APPLICATION);

wc.hInstance = hInstance;

wc.lpszClassName = L"LAB1!";

wc.lpszMenuName = nullptr;

wc.style = CS\_VREDRAW | CS\_HREDRAW;

WNDCLASS wcSearchWindow{ sizeof(WNDCLASS) };

wcSearchWindow.lpfnWndProc = SearchWindowProc;

wcSearchWindow.hInstance = hInstance;

wcSearchWindow.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

wcSearchWindow.lpszClassName = L"SearchWindowClass";

WNDCLASSEXW wcChatWindow{ sizeof(WNDCLASSEX) };

wcChatWindow.lpfnWndProc = ChatProc;

wcChatWindow.hInstance = hInstance;

wcChatWindow.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

wcChatWindow.lpszClassName = L"ChatWindowClass";

RegisterClassExW(&wcChatWindow);

if (!RegisterClass(&wcSearchWindow) || !RegisterClassW(&wc)) {

CloseHandle(g\_hFileMutex);

return EXIT\_FAILURE;

}

LoadFontSettings();

if (hWnd = CreateWindow(wc.lpszClassName, L"LAB1", WS\_OVERLAPPEDWINDOW | WS\_CLIPCHILDREN, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, nullptr, nullptr, wc.hInstance, nullptr)) {

HMENU hMenu = CreateMenu();

HMENU hFileMenu = CreateMenu();

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFileMenu, L"Файл");

AppendMenu(hFileMenu, MF\_STRING, IDM\_OPEN, L"Открыть");

AppendMenu(hFileMenu, MF\_STRING, IDM\_SAVE, L"Сохранить");

AppendMenu(hFileMenu, MF\_STRING, IDM\_OPEN\_BINARY, L"Открыть бинарный файл");

AppendMenu(hFileMenu, MF\_STRING, IDM\_SAVE\_BINARY, L"Сохранить бинарный файл");

AppendMenu(hFileMenu, MF\_STRING, IDC\_START\_COUNT, L"Начать счетчик");

AppendMenu(hFileMenu, MF\_STRING, IDC\_STOP\_COUNT, L"Закончить счетчик");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_EXIT, L"Выход");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_NEW, L"Новый");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_CHANGE\_THEME, L"Поменять тему");

AppendMenu(hFileMenu, MF\_STRING, IDM\_CHANGE\_FONT, L"Изменить шрифт");

AppendMenu(hFileMenu, MF\_STRING, IDM\_REGISTRY\_ACCESS, L"Открыть реестр");

AppendMenu(hMenu, MF\_STRING, IDM\_SEARCH, L"Поиск");

AppendMenu(hMenu, MF\_STRING, OPEN\_CHAT, L"Открыть чат");

SetMenu(hWnd, hMenu);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

while (GetMessage(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

if (msg.message == WM\_HOTKEY)

{

OnHotkey(msg.hwnd, msg.wParam, msg.lParam);

}

}

}

closesocket(g\_socket);

return EXIT\_SUCCESS;

}

LRESULT CALLBACK SearchWindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDM\_SEARCH\_IN\_SEARCH\_WINDOW:

{

int searchQueryLength = GetWindowTextLength(hSearchEdit);

if (searchQueryLength == 0)

{

MessageBox(hWnd, L"Введите текст.", L"Error", MB\_OK | MB\_ICONERROR);

return 0;

}

wchar\_t\* searchQuery = new wchar\_t[searchQueryLength + 1];

GetWindowText(hSearchEdit, searchQuery, searchQueryLength + 1);

int mainEditLength = GetWindowTextLength(hEdit);

wchar\_t\* mainEditText = new wchar\_t[mainEditLength + 1];

GetWindowText(hEdit, mainEditText, mainEditLength + 1);

wchar\_t\* found = wcsstr(mainEditText + searchStartPosition, searchQuery);

if (found != nullptr)

{

int foundPos = found - mainEditText;

SendMessage(hEdit, EM\_SETSEL, foundPos, foundPos + searchQueryLength);

CHARFORMAT2 cf;

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_BACKCOLOR;

cf.crBackColor = RGB(255, 255, 0);

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

searchStartPosition = foundPos + searchQueryLength;

}

else

{

searchStartPosition = 0;

MessageBox(hWnd, L"No more matches found.", L"Search Complete", MB\_OK | MB\_ICONINFORMATION);

}

delete[] searchQuery;

delete[] mainEditText;

}

break;

case WM\_CLOSE:

if (hWnd == hSearchWindow)

DestroyWindow(hWnd);

break;

}

break;

case WM\_DESTROY:

if (hWnd == hSearchWindow)

hSearchWindow = nullptr;

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

void ResizeEditControl(HWND hWnd, HWND hEdit)

{

RECT clientRect;

GetClientRect(hWnd, &clientRect);

SetWindowPos(hEdit, nullptr, 0, 0, clientRect.right, clientRect.bottom, SWP\_NOZORDER);

}

void OpenBinaryFile(LPCTSTR filePath) {

WaitForSingleObject(g\_hFileMutex, INFINITE);

if (g\_pFileData != NULL) {

UnmapViewOfFile(g\_pFileData);

g\_pFileData = NULL;

}

if (g\_hFileMapping != NULL) {

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

}

HANDLE hFile = CreateFile(filePath, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {

MessageBox(g\_hwnd, \_T("Failed to open the file"), \_T("Error"), MB\_ICONERROR);

return;

ReleaseMutex(g\_hFileMutex);

}

g\_fileSize = GetFileSize(hFile, NULL);

g\_hFileMapping = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, g\_fileSize, NULL);

if (g\_hFileMapping == NULL) {

MessageBox(g\_hwnd, \_T("Failed to create file mapping"), \_T("Error"), MB\_ICONERROR);

CloseHandle(hFile);

ReleaseMutex(g\_hFileMutex);

return;

}

g\_pFileData = MapViewOfFile(g\_hFileMapping, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, g\_fileSize);

if (g\_pFileData == NULL) {

MessageBox(g\_hwnd, \_T("Failed to map the file into memory"), \_T("Error"), MB\_ICONERROR);

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

CloseHandle(hFile);

ReleaseMutex(g\_hFileMutex);

return;

}

ReleaseMutex(g\_hFileMutex);

}

void StopWordCountThread() {

g\_bIsWordCountThreadRunning = false;

if (hThread != nullptr) {

WaitForSingleObject(hThread, INFINITE);

CloseHandle(hThread);

hThread = nullptr;

}

}

void SaveHexToFile(LPCTSTR filePath, const std::wstring& hexString) {

if (g\_hFileMapping != NULL) {

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

HANDLE hFile = CreateFile(filePath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

DWORD bytesWritten;

WriteFile(hFile, hexString.c\_str(), static\_cast<DWORD>(hexString.size() \* sizeof(WCHAR)), &bytesWritten, NULL);

CloseHandle(hFile);

}

}

}

void OpenRegistryAccessWindow(HWND hWnd)

{

WCHAR szClassName[] = L"RegistryAccessWindowClass";

WNDCLASS wc = { 0 };

wc.lpfnWndProc = RegistryAccessWndProc;

wc.hInstance = GetModuleHandle(NULL);

wc.lpszClassName = szClassName;

if (!RegisterClass(&wc))

{

DWORD dwError = GetLastError();

wchar\_t errorMessage[256];

swprintf(errorMessage, 256, L"Ошибка при регистрации класса окна. Код ошибки: %lu", dwError);

MessageBox(NULL, errorMessage, L"Ошибка", MB\_OK | MB\_ICONERROR);

}

else

{

HWND hRegistryAccessWindow = CreateWindow(

szClassName,

L"Registry Access",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 200,

nullptr,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

if (hRegistryAccessWindow != nullptr)

{

ShowWindow(hRegistryAccessWindow, SW\_SHOWNORMAL);

}

else

{

DWORD dwError = GetLastError();

wchar\_t errorMessage[256];

swprintf(errorMessage, 256, L"Произошла ошибка при создании окна. Код ошибки: %lu", dwError);

MessageBox(NULL, errorMessage, L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static wchar\_t currentFileName[MAX\_PATH] = L"";

HWND hButtonContainer = nullptr;

HWND hStartWordCountButton = nullptr;

switch (message)

{

case WM\_CREATE:

{

LoadLibrary(TEXT("Msftedit.dll"));

hEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

RICHEDIT\_CLASS,

L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | WS\_HSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL | ES\_NOHIDESEL,

0, 0, 800, 500,

hWnd,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

LoadFontSettings();

LoadColorSettings();

if (hEdit == nullptr)

{

MessageBox(hWnd, L"Не удалось создать элемент управления Edit.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

break;

}

case WM\_UPDATE\_WORD\_COUNT: {

std::wstring labelText = L"Количество слов: " + std::to\_wstring(wordsCount);

SetWindowText(hWnd, labelText.c\_str());

break;

}

case WM\_SIZE:

ResizeEditControl(hWnd, hEdit);

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case OPEN\_CHAT:

InitiateConnection(hWnd);

break;

case IDM\_REGISTRY\_ACCESS:

OpenRegistryAccessWindow(hWnd);

break;

case IDM\_CHANGE\_THEME:

OpenColorDialog();

break;

case IDM\_CHANGE\_FONT:

{

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf = { 0 };

cf.hwndOwner = hWnd;

cf.lpLogFont = &lf;

cf.Flags = CF\_SCREENFONTS | CF\_EFFECTS | CF\_INITTOLOGFONTSTRUCT;

if (ChooseFont(&cf))

{

HFONT hFont = CreateFontIndirect(&lf);

SendMessage(hEdit, WM\_SETFONT, (WPARAM)hFont, TRUE);

HKEY hKey;

if (RegCreateKeyEx(HKEY\_CURRENT\_USER, L"Software\\322EDITOR322\\FontSettings", 0, NULL, 0, KEY\_WRITE, NULL, &hKey, NULL) == ERROR\_SUCCESS)

{

RegSetValueEx(hKey, L"FontName", 0, REG\_SZ, (LPBYTE)lf.lfFaceName, sizeof(wchar\_t) \* (wcslen(lf.lfFaceName) + 1));

RegSetValueEx(hKey, L"FontSize", 0, REG\_DWORD, (LPBYTE)&lf.lfHeight, sizeof(DWORD));

RegSetValueEx(hKey, L"FontUnderline", 0, REG\_DWORD, (LPBYTE)&lf.lfUnderline, sizeof(DWORD));

RegSetValueEx(hKey, L"FontStrikeOut", 0, REG\_DWORD, (LPBYTE)&lf.lfStrikeOut, sizeof(DWORD));

RegSetValueEx(hKey, L"FontItalic", 0, REG\_DWORD, (LPBYTE)&lf.lfItalic, sizeof(DWORD));

RegSetValueEx(hKey, L"FontBold", 0, REG\_DWORD, (LPBYTE)&lf.lfWeight, sizeof(DWORD));

RegCloseKey(hKey);

}

}

break;

}

case IDM\_OPEN:

{

OPENFILENAME ofn;

wchar\_t szFile[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"C++ файлы (\*.cpp;\*.txt)\0\*.cpp;\*.txt\0Все файлы (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFile;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_FILEMUSTEXIST | OFN\_PATHMUSTEXIST;

if (GetOpenFileName(&ofn))

{

wcscpy\_s(currentFileName, MAX\_PATH, ofn.lpstrFile);

HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD dwFileSize = GetFileSize(hFile, NULL);

wchar\_t\* fileContent = (wchar\_t\*)malloc((dwFileSize + 1) \* sizeof(wchar\_t));

if (fileContent)

{

DWORD bytesRead;

ReadFile(hFile, fileContent, dwFileSize \* sizeof(wchar\_t), &bytesRead, NULL);

fileContent[dwFileSize / sizeof(wchar\_t)] = L'\0';

SetWindowText(hEdit, fileContent);

free(fileContent);

}

else

{

MessageBox(hWnd, L"Не удалось выделить память для содержимого файла.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Не удалось открыть выбранный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case IDM\_SAVE:

{

HANDLE hFile = INVALID\_HANDLE\_VALUE;

if (currentFileName[0] != L'\0')

{

hFile = CreateFile(currentFileName, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

}

if (hFile == INVALID\_HANDLE\_VALUE)

{

OPENFILENAME ofn;

wchar\_t szFile[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"C++ files (\*.cpp;\*.txt)\0\*.cpp;\*.txt\0All files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFile;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

hFile = CreateFile(ofn.lpstrFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

wcscpy\_s(currentFileName, MAX\_PATH, ofn.lpstrFile);

}

else

{

MessageBox(hWnd, L"Не удалось создать выбранный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

}

if (hFile != INVALID\_HANDLE\_VALUE)

{

int textLength = GetWindowTextLength(hEdit);

wchar\_t\* textBuffer = (wchar\_t\*)malloc((textLength + 1) \* sizeof(wchar\_t));

if (textBuffer)

{

GetWindowText(hEdit, textBuffer, textLength + 1);

DWORD bytesWritten;

WriteFile(hFile, textBuffer, textLength \* sizeof(wchar\_t), &bytesWritten, NULL);

free(textBuffer);

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Не удалось выделить память для текста.", L"Ошибка", MB\_OK | MB\_ICONERROR);

CloseHandle(hFile);

}

}

break;

}

case IDM\_NEW:

{

if (MessageBox(hWnd, L"Вы уверены, что хотите создать новый документ? Все несохраненные изменения будут потеряны.", L"Подтвердите действие", MB\_YESNO | MB\_ICONQUESTION) == IDYES)

{

SetWindowText(hEdit, L"");

currentFileName[0] = L'\0';

}

break;

}

case IDM\_OPEN\_BINARY: {

OPENFILENAME ofn = { 0 };

TCHAR filePath[MAX\_PATH] = { 0 };

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = filePath;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_FILEMUSTEXIST | OFN\_PATHMUSTEXIST;

ofn.lpstrFilter = L"Binary Files\0\*.bin\0All Files\0\*.\*\0";

ofn.lpstrDefExt = L"bin";

if (GetOpenFileName(&ofn)) {

OpenBinaryFile(filePath);

int requiredBufferSize = MultiByteToWideChar(CP\_UTF8, 0, (LPCCH)g\_pFileData, -1, NULL, 0);

if (requiredBufferSize > 0) {

WCHAR\* utf16Text = new WCHAR[requiredBufferSize];

MultiByteToWideChar(CP\_UTF8, 0, (LPCCH)g\_pFileData, -1, utf16Text, requiredBufferSize);

LPCWSTR text = utf16Text;

SetWindowText(hEdit, text);

delete[] utf16Text;

}

lstrcpy(currentFileName, filePath);

}

break;

}

case IDM\_SAVE\_BINARY: {

if (g\_hFileMapping != NULL) {

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = currentFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

ofn.lpstrFilter = L"Binary Files\0\*.bin\0All Files\0\*.\*\0";

ofn.lpstrDefExt = L"bin";

if (GetSaveFileName(&ofn)) {

int textLength = GetWindowTextLength(hEdit);

std::wstring text;

if (textLength > 0) {

text.resize(textLength);

GetWindowText(hEdit, &text[0], textLength + 1);

}

std::wstring hexString = text.c\_str();

SaveHexToFile(currentFileName, hexString);

}

}

break;

}

case IDM\_SEARCH:

OpenSearchWindow(hWnd);

break;

case IDC\_START\_COUNT: {

ThreadParams threadParams;

threadParams.hWnd = hWnd;

hThread = CreateThread(nullptr, 0, WordCountThread, &threadParams, CREATE\_SUSPENDED, nullptr);

if (hThread == nullptr) {

MessageBox(hWnd, L"Не удалось запустить поток.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

if (SetThreadPriority(hThread, THREAD\_PRIORITY\_ABOVE\_NORMAL) == 0) {

MessageBox(hWnd, L"Не удалось назначить приоритет потоку.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

ResumeThread(hThread);

break;

}

case IDC\_STOP\_COUNT: {

StopWordCountThread();

break;

}

case WM\_CLOSE: {

if (hThread != nullptr) {

DeleteCriticalSection(&g\_csWordCount);

TerminateThread(hThread, 0);

CloseHandle(hThread);

}

DestroyWindow(hWnd);

break;

}

case IDM\_EXIT:

PostQuitMessage(0);

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}