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 № 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 гр. 153503

Татаринов В.В.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 6](#_Toc146836469)

[Выводы 7](#_Toc146836470)

[Список использованных источников 8](#_Toc146836471)

[Приложение А (обязательное) листинг кода 9](#_Toc146836472)

# **1 ЦЕЛЬ РАБОТЫ**

Изучить интерфейсы сокетов и основы сетевого программирования. Создать 2 приложения, позволяющие обмениваться информацией между собой как клиент и сервер с использованием интерфейса сокетов. Использовать собственные или стандартные сетевые протоколы для передачи данных.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

Сетевое программирование — это область программирования, связанная с разработкой приложений, способных взаимодействовать и обмениваться данными через компьютерные сети. Эта область охватывает создание программ, работающих в распределенной среде, где различные компьютеры могут общаться между собой, отправлять и принимать данные.

Сеть представляет собой инфраструктуру, позволяющую различным устройствам обмениваться информацией. Это может быть локальная сеть (*LAN*) в пределах одного офиса или дома, глобальная сеть, такая как Интернет, или виртуальная сеть между программами на одном компьютере.

Сокет — это виртуальная конструкция из *IP*-адреса и номера порта. Сокет служит для того, чтобы было проще писать код, а программы могли передавать данные друг другу даже в пределах одного компьютера. Т.к. сокет на сервере один, а программ, которые должны подключаться, много, то сервер копирует сокеты. Когда на сервер поступает запрос на соединение с сокетом, он не устанавливает связь напрямую, а копирует этот сокет и настраивает связь через него. После копирования сервер запоминает, какая копия отвечает за какое соединение, и дальше просто обрабатывает все запросы по очереди. При этом исходный сокет остаётся нетронутым — он не используется для связи напрямую, а служит шаблоном для создания копий [1].

Протоколы представляют собой набор правил и соглашений, определяющих формат, порядок и правила взаимодействия между устройствами или программами в сети. Они служат основой для обмена данными и обеспечивают структурированный способ передачи информации между различными компьютерами, устройствами или системами. Протоколы могут работать на разных уровнях стека сетевых протоколов (например, модель *OSI* или *TCP/IP*), предоставляя структуру для обработки различных аспектов передачи данных.

Сетевая модель *OSI* — сетевая модель стека сетевых протоколов *OSI/ISO*. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Модель определяет различные уровни взаимодействия систем. Каждый уровень выполняет определённые функции при таком взаимодействии [2].

*TCP/IP* — это модель передачи цифровых данных. Протокол передачи *TCP/IP* описывает правила передачи данных, стандарты связи между компьютерами, а также содержит соглашения о маршрутизации и межсетевом взаимодействии [3].

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы были созданы 2 приложения, которые представляют собой клиент и сервер и могут общаться между собой посредством передачи данных изображений и текста.

Интерфейс клиента представляет собой кнопку для загрузки изображения и поле, отображающие результат выполнения передачи данных на сервер (рисунок 1).

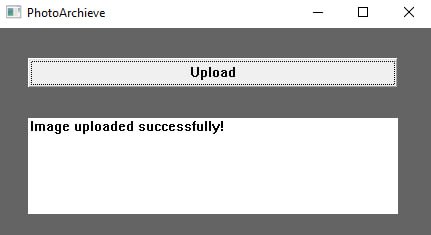


Рисунок 1 – Интерфейс клиента

Интерфейс сервера представляет собой консоль с отчетами о каждом произведенном действии со стороны клиента и сервера (рисунок 2).

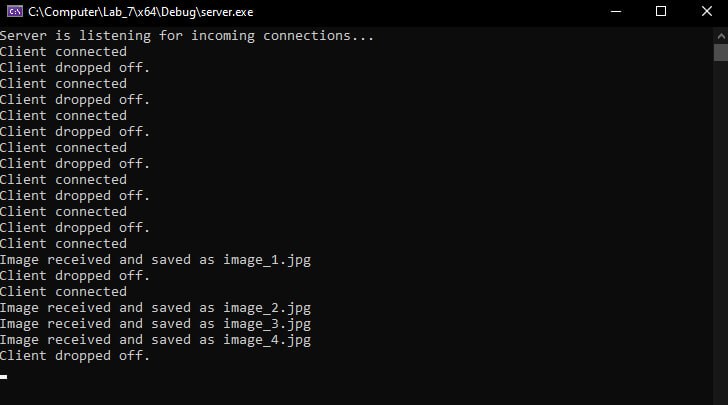


Рисунок 2 – Интерфейс сервера

# **ВЫВОДЫ**

В результате выполнения лабораторной работы были изучены интерфейсы сокетов и основы сетевого программирования. Созданы приложения, представляющие собой клиент и сервер и обменивающиеся данными между собой используя интерфейсы сокетов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Сокет [Электронный ресурс]. – Режим доступа: https://thecode.media/socket/.
2. Сетевая модель *OSI* [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент\_–\_сервер.
3. *TCP/IP* [Электронный ресурс]. – Режим доступа: https://www.nic.ru/help/chto-takoe-tcpip\_11168.html.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *client.cpp*:

#include <Windows.h>

#include <string>

#include <thread>

#include <iostream>

#include <fstream>

#include <list>

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

using namespace std;

#define ID\_BUTTON\_UPLOAD 100

HMENU hMenu;

HHOOK hKeyboardHook = NULL;

HWND hActiveWindow, hListBox;

SOCKET clientSocket;

LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);

void AddMenus(HWND hWnd);

void AddControls(HWND hWnd);

void SendMessage(HWND hWnd);

void ConnectToServer(const char\* serverIP);

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode >= 0) {

if (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN) {

KBDLLHOOKSTRUCT\* pKbStruct = (KBDLLHOOKSTRUCT\*)lParam;

if (pKbStruct->scanCode == 0x1C) {

hActiveWindow = GetForegroundWindow();

HWND hButton = GetDlgItem(hActiveWindow, ID\_BUTTON\_UPLOAD);

if (hButton != NULL) {

SendMessage(hButton, BM\_CLICK, 0, 0);

}

}

}

}

return CallNextHookEx(hKeyboardHook, nCode, wParam, lParam);

}

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR arts, int ncmdshow) {

WNDCLASSW wc = { 0 };

wc.hbrBackground = CreateSolidBrush(RGB(255, 150, 255));

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

wc.hInstance = hInst;

wc.lpszClassName = L"Class";

wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc))

return -1;

CreateWindowW(L"Class", L"PhotoArchieve", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 0, 0, 450, 250, NULL, NULL, hInst, NULL);

HHOOK hKeyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, hInst, 0);

MSG msg = { 0 };

while (GetMessage(&msg, NULL, NULL, NULL)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hKeyboardHook);

return 0;

}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg)

{

case WM\_COMMAND:

switch (wp)

{

case ID\_BUTTON\_UPLOAD: {

SendMessage(hWnd);

break;

}

}

break;

case WM\_CREATE:

AddMenus(hWnd);

AddControls(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

}

void AddMenus(HWND hWnd) {

hMenu = CreateMenu();

SetMenu(hWnd, hMenu);

}

void AddControls(HWND hWnd) {

hListBox = CreateWindowEx(0, L"LISTBOX", NULL, WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 30, 90, 370, 100, hWnd, NULL, NULL, NULL);

CreateWindowEx(0, L"BUTTON", L"Upload", WS\_CHILD | WS\_VISIBLE, 30, 30, 370, 30, hWnd, reinterpret\_cast<HMENU>(ID\_BUTTON\_UPLOAD), NULL, NULL);

ConnectToServer("127.0.0.1");

}

void HandleMessages() {

while (true) {

char buffer[4096];

int bytesRead = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);

if (bytesRead <= 0) {

MessageBox(NULL, L"Error reading messages or server disconnected.", L"Error", MB\_OK | MB\_ICONERROR);

closesocket(clientSocket);

exit(1);

}

buffer[bytesRead] = '\0';

char qwe[] = "Sent";

SendMessageA(hListBox, LB\_ADDSTRING, 0, reinterpret\_cast<LPARAM>(buffer));

}

}

void ConnectToServer(const char\* serverIP) {

WSADATA wsaData;

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

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

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr(serverIP);

serverAddr.sin\_port = htons(12345);

connect(clientSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr));

thread(HandleMessages).detach();

}

void SendMessage(HWND hWnd) {

LPCWSTR title = L"Select Image";

LPCWSTR initialDir = NULL;

LPCWSTR filter = L"Image Files\0\*.jpg;";

WCHAR filename[MAX\_PATH];

filename[0] = '\0';

OPENFILENAME ofn;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrTitle = title;

ofn.lpstrInitialDir = initialDir;

ofn.lpstrFilter = filter;

ofn.nFilterIndex = 1;

ofn.lpstrFile = filename;

ofn.nMaxFile = MAX\_PATH;

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

// Open the File Open dialog

if (!GetOpenFileName(&ofn)) {

return;

}

std::ifstream imageFile(filename, std::ios::binary);

if (!imageFile.is\_open()) {

MessageBox(NULL, L"Image does not exist!", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

char buffer[600000];

while (!imageFile.eof()) {

imageFile.read(buffer, sizeof(buffer));

int bytesRead = imageFile.gcount();

send(clientSocket, buffer, bytesRead, 0);

}

imageFile.close();

}

Листинг 2 **–** Файл *server.cpp*:

#include <iostream>

#include <winsock2.h>

#include <iostream>

#include <winsock2.h>

#include <fstream>

#include <string>

#include <list>

#include <thread>

std::list<SOCKET> clients;

void HandleClient(const SOCKET& clientSocket);

int file\_index = 1;

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

int main() {

// Winsock init

WSADATA wsaData;

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

// Create socket

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

// setup server address

sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

serverAddress.sin\_port = htons(12345);

// bind socket to address

bind(serverSocket, (struct sockaddr\*)&serverAddress, sizeof(serverAddress));

listen(serverSocket, SOMAXCONN);

std::cout << "Server is listening for incoming connections..." << std::endl;

while (true) {

SOCKET clientSocket = accept(serverSocket, NULL, NULL);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to accept connection" << std::endl;

closesocket(serverSocket);

WSACleanup();

continue;

}

clients.push\_back(clientSocket);

std::thread(HandleClient, clientSocket).detach();

}

WSACleanup();

return 0;

}

void HandleClient(const SOCKET& clientSocket) {

std::cout << "Client connected" << std::endl;

char buffer[1000000];

int bytesRead;

do {

bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);

if (bytesRead > 0) {

std::string file\_name = "image\_" + std::to\_string(file\_index);

file\_name += ".jpg";

std::string file\_path = "C:/Computer/Lolka/" + file\_name;

file\_index++;

std::ofstream imageFile(file\_path, std::ios::binary);

imageFile.write(buffer, bytesRead);

imageFile.close();

std::cout << "Image received and saved as " << file\_name << std::endl;

char qwe[4096] = "Image uploaded successfully!";

int bytes = 4096;

send(clientSocket, qwe, bytes, 0);

}

} while (bytesRead > 0);

std::cout << "Client" << " dropped off.\n";

clients.remove\_if([clientSocket](SOCKET s) { return s == clientSocket; });

closesocket(clientSocket);

return;

}