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

Элементы сетевого программирования

Студент П. С. Павлюткин

Преподаватель С. И. Сиротко

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**

Цель работы изучить и применить основы построения и функционирования сетей, стеки протоколов, программные интерфейсы, а также научиться организовывать взаимодействия между процессами в сетях. Для достижения цели будет создано приложение для интерактивного взаимодействия пользователей (чат).

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Иерархическая модель взаимодействия открытых систем, уровни, протоколы и интерфейсы. Стеки протоколов *OSI*, *TCP/IP* и др. Протокол *IP*. Идентификация абонентов в *IP*-сетях. Протоколы транспортного уровня TCP и *UDP*. Программный интерфейс сокетов. Типы, структуры данных, функции *API*. Порядок организации взаимодействия посредством сокетов (потоковых и датаграмных). Способы организации серверов. Функции (вызовы) *socket* (), *bind* (), *connect* (), *listen* (), *send* (), *recv* (), *send to* (), *recv from* ().

Сетевое программирование — это область программирования, связанная с разработкой приложений, способных обмениваться данными через сети компьютеров. Эти приложения могут выполняться на одном компьютере и взаимодействовать с другими приложениями на этом же компьютере (локальное взаимодействие), а также обмениваться данными с приложениями, работающими на удаленных компьютерах (удаленное взаимодействие).

Сокеты представляют собой абстракцию для обеспечения взаимодействия между процессами через сеть. Они позволяют приложениям отправлять и получать данные по сети.

Существуют два основных типа сокетов — потоковые (*TCP*) и дейтаграммные (*UDP*), обеспечивающие соответственно надежное, установленное соединение и передачу данных без установления соединения. *TCP* (*Transmission Control Protocol*): обеспечивает надежную передачу данных между приложениями. Основан на установлении соединения и гарантирует доставку данных в порядке, в котором они были отправлены. *UDP* (*User* *Datagram Protocol*): Предоставляет более легковесный и быстрый способ передачи данных, но без гарантии надежной доставки. Подходит для приложений, где небольшая задержка более важна, чем надежность. IP-адрес: Идентифицирует уникальный компьютер в сети. IP-адреса используются для указания отправителя и получателя данных. Порт: Идентифицирует конкретное приложение на компьютере. Вместе с IP-адресом образует адрес для входящих и исходящих данных.

API сокетов: Набор функций и процедур, предоставляемых операционной системой для создания, управления и закрытия сокетов. Примеры включают функции `*socket*()`, `*bind*()`, `*listen*()`, `*connect*()`, `*send*()`, `*recv*()` и другие. Модели сетевого взаимодействия: Клиент-Сервер: Один из наиболее распространенных подходов, где одно приложение (клиент) запрашивает услуги у другого (сервера). *Peer-to-Peer*: В этой модели каждый компьютер в сети может одновременно выступать в роли клиента и сервера.

Веб-серверы и клиенты: Программы, обрабатывающие запросы и предоставляющие ресурсы через интернет. Чат-приложения: Позволяют пользователям обмениваться текстовыми сообщениями по сети. Системы удаленного доступа: Позволяют управлять удаленными компьютерами через сеть.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было реализовано приложение для интерактивного взаимодействия пользователей по локальной сети с использованием сокетов (Рисунок 1).

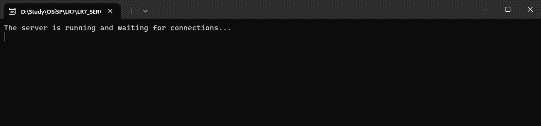


Рисунок 1 – Серверное окно приложения

Также у нас есть клиентское окно приложения, которое позволяет взаимодействовать с сервером (Рисунок 2).

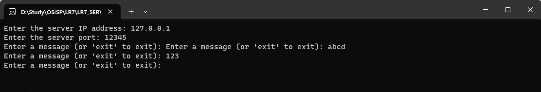


Рисунок 2 – Клиентское окно приложения

Далее будет представлена реализация взаимодействия между клиентом и сервером (Рисунок 3).

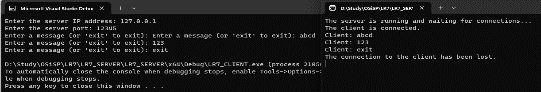


Рисунок 3 – Взаимодействие между серверной и клиентской частью

Наше приложение позволяет пользователям использовать наш самописный сервер для отправки различных сообщений, также наш сервер обрабатывает сразу несколько пользователей, используя многопоточность, для одного пользователя один поток. Сервер и клиент взаимодействуют через сокетов, обеспечивая передачу текстовых сообщений по локальной сети.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены и применены методы взаимодействия между процессами с использованием технологии сокетов в среде Windows и библиотеки WinAPI, основанной на языке программирования C++. Работа включала в себя моделирование взаимодействия нескольких клиентов в упрощенном чате, обеспечивая двусторонний обмен данными по протоколу TCP.

Проект состоит из двух основных компонентов: сервера и клиентов. Сервер создает сокет для приема подключений, обрабатывает входящие сообщения от клиентов и рассылает их всем подключенным участникам. Клиенты взаимодействуют с сервером, передавая пользовательские сообщения и отображая полученные данные.

Для обеспечения взаимодействия между процессами были использованы сокеты, а взаимодействие осуществлялось посредством сетевых протоколов TCP. Особое внимание было уделено обработке ошибок, что позволило создать надежное и стабильное приложение.

Лабораторная работа предоставила практический опыт в реализации сетевого взаимодействия между процессами в операционной системе Windows с использованием библиотеки WinAPI. Этот опыт является ключевым для разработчиков, работающих над распределенными системами и приложениями, где важно обеспечить эффективное общение между отдельными компонентами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. *Win32 API*. Разработка приложений для *Windows*. ─ СПБ: Питер, 2008. ─ 592 с.: ип.
2. Создание классических приложений для *Windows* с использованием *API Win32* [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab7\_SERVER.cpp

#include <iostream>

#include <Winsock2.h>

#include <vector>

#include <thread>

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

void HandleClient(SOCKET clientSocket) {

char buffer[1024];

int bytesReceived;

while (true) {

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

if (bytesReceived <= 0) {

std::cerr << "The connection to the client has been lost." << std::endl;

break;

}

buffer[bytesReceived] = '\0';

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

// Отправка ответа клиенту

send(clientSocket, buffer, bytesReceived, 0);

}

closesocket(clientSocket);

}

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock." << std::endl;

return 1;

}

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

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create socket." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(12345); // Порт сервера

serverAddress.sin\_addr.s\_addr = INADDR\_ANY; // Принимаем подключения на всех доступных интерфейсах

if (bind(serverSocket, (sockaddr\*)&serverAddress, sizeof(serverAddress)) == SOCKET\_ERROR) {

std::cerr << "Failed to bind socket." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

std::cerr << "Failed to listen for connections." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "The server is running and waiting for connections..." << std::endl;

std::vector<std::thread> clientThreads;

while (true) {

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

if (clientSocket == INVALID\_SOCKET) {

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

continue;

}

std::cout << "The client is connected." << std::endl;

clientThreads.emplace\_back(HandleClient, clientSocket);

}

closesocket(serverSocket);

WSACleanup();

for (auto& thread : clientThreads) {

thread.join();

}

return 0;

}

Листинг 2 – Файл Lab7\_Client.cpp

#include <iostream>

#include <Winsock2.h>

#include <string>

#include <WS2tcpip.h>

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

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock." << std::endl;

return 1;

}

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

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create socket." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET;

// Ввод адреса сервера и порта с клавиатуры

std::string serverIP;

int serverPort;

std::cout << "Enter the server IP address: ";

std::cin >> serverIP;

std::cout << "Enter the server port: ";

std::cin >> serverPort;

if (inet\_pton(AF\_INET, serverIP.c\_str(), &serverAddress.sin\_addr) != 1) {

std::cerr << "Invalid IP address." << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

serverAddress.sin\_port = htons(serverPort);

if (connect(clientSocket, (sockaddr\*)&serverAddress, sizeof(serverAddress)) == SOCKET\_ERROR) {

std::cerr << "Failed to connect to the server." << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::string message;

while (true) {

std::cout << "Enter a message (or 'exit' to exit): ";

std::getline(std::cin, message);

send(clientSocket, message.c\_str(), static\_cast<int>(message.size()), 0);

if (message == "exit") {

break;

}

}

closesocket(clientSocket);

WSACleanup();

return 0;

}

Листинг 1 – Файл Lab7\_CLIENT.cpp

#include <iostream>

#include <Winsock2.h>

#include <string>

#include <WS2tcpip.h>

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

int main() {

WSADATA wsaData;

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

std::cerr << "Failed to initialize Winsock." << std::endl;

return 1;

}

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

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Failed to create socket." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET;

// Ввод адреса сервера и порта с клавиатуры

std::string serverIP;

int serverPort;

std::cout << "Enter the server IP address: ";

std::cin >> serverIP;

std::cout << "Enter the server port: ";

std::cin >> serverPort;

if (inet\_pton(AF\_INET, serverIP.c\_str(), &serverAddress.sin\_addr) != 1) {

std::cerr << "Invalid IP address." << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

serverAddress.sin\_port = htons(serverPort);

if (connect(clientSocket, (sockaddr\*)&serverAddress, sizeof(serverAddress)) == SOCKET\_ERROR) {

std::cerr << "Failed to connect to the server." << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::string message;

while (true) {

std::cout << "Enter a message (or 'exit' to exit): ";

std::getline(std::cin, message);

send(clientSocket, message.c\_str(), static\_cast<int>(message.size()), 0);

if (message == "exit") {

break;

}

}

closesocket(clientSocket);

WSACleanup();

return 0;

}