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. 153504

Прескурел Я.Ю.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](#_Toc146631500)

[Выводы 8](#_Toc146631501)

[Список использованных источников 9](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 10](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является построение системы обмена файлами клиентами через сеть с возможностью выбора и отправки файлов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.[1]

Интерфейс сокетов (или сокет-программирование) представляет собой мощный инструмент для разработки сетевых приложений на платформе Windows и других операционных системах. Сокеты обеспечивают уровень абстракции для взаимодействия между устройствами в сети и позволяют программистам разрабатывать приложения, которые способны обмениваться данными через сеть.

Программисты могут создавать сокеты, устанавливать соединения, отправлять и принимать данные через них. Сокеты могут быть ориентированы как на передачу данных по протоколу TCP (надежное, ориентированное на соединение), так и по протоколу UDP (ненадежное, без соединения).

С использованием сокетов, приложения могут одновременно обслуживать несколько соединений с клиентами. Мультиплексирование позволяет эффективно обрабатывать множество запросов, такие как веб-серверы, чаты или онлайн-игры.

Сокеты широко используются для создания клиент-серверных приложений, где сервер ожидает соединений от клиентов и предоставляет им данные или услуги. Клиенты могут подключаться к серверу и отправлять запросы.

Сокеты позволяют работать с различными сетевыми протоколами, такими как HTTP (для веб-серверов и клиентов), FTP (для передачи файлов), SMTP (для отправки почты) и многими другими. Программисты могут создавать приложения, которые взаимодействуют с этими протоколами.

Сетевое программирование также включает в себя аспекты безопасности, такие как защита от атак и шифрование данных. Программисты должны уделять внимание безопасности своих сетевых приложений, чтобы защитить их от угроз.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

1. Создание сокета: в серверной части сначала создается сокет с помощью socket(). Сокеты – это конечные точки для установления соединения между узлами в сети. В данном случае используется сокет типа SOCK\_STREAM, что означает, что будет установлено надежное TCP-соединение.

2. Привязка к адресу и порту: сервер привязывает свой сокет к определенному IP-адресу и порту с помощью bind(). Это позволяет серверу слушать входящие соединения на указанном порту.

3. Прослушивание соединений: функция listen() используется для настройки серверного сокета на прослушивание входящих соединений. Это позволяет серверу принимать соединения от клиентов.

4. Прием клиентов: в бесконечном цикле сервер ожидает входящие соединения с помощью accept(). Когда клиент подключается, создается новый поток (или потоки) для обработки взаимодействия с этим клиентом.

5. Обработка клиентов: в функции HandleClient происходит обработка команд от клиентов. Клиенты могут загружать файлы на сервер с командой "UPLOAD" и скачивать файлы с сервера с командой "DOWNLOAD". Сам файл передается в бинарном режиме, а данные читаются и записываются в буфер.

6. Управление клиентами: информация о клиентах хранится в векторе clients. Когда клиент отключается или происходит ошибка в соединении, его информация удаляется из вектора.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была построена система передачи файлов клиентами через сеть с возможностью выбора и отправки файлов. Проект поделен на две части:

– серверная часть;

– клиентская часть.

В систему входят три проекта. Проект ServerPart отображает серверную часть, а проекты ClientPart и ClientPart2 – клиентскую часть.

Результат успешного запуска сервера представлен на рисунке 3.1.



Рисунок 3.1 – Результат работы сервера при запуске

Теперь для наглядности работы запустим проекты клиентов. Результат подключения клиентов к серверу представлен на рисунке 3.2.

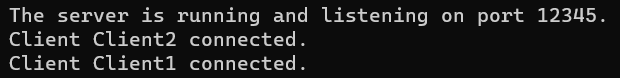


Рисунок 3.2 – Результат подключения клиентов к серверу

Проведем передачу файлов на сервер от Client1. При помощи консольного меню пользователь может выбрать нужную ему функцию. Изображение меню будет представлено на рисунке 3.3.

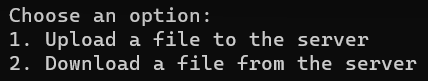


Рисунок 3.3 – Меню пользователя

При выборе первого пункта меню пользователю будет доступна возможность при выборе полного пути до любого файла на устройстве отправить его на сервер. Результат работы данной функции будет представлен на рисунке 3.4.

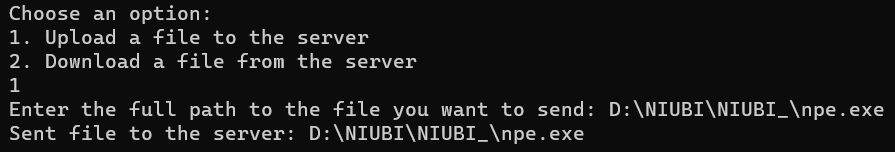


Рисунок 3.4 – Результат отправки на сервер файла от пользователя

На сервере также отображается информация о том, что файл был получен сервером и сохранен. Информация, отображенная на сервере, будет представлена на рисунке 3.5.



Рисунок 3.5 – Результат получения файла от клиента сервером

При выборе второго пункта меню пользователь может получить файл от сервера. Для этого пользователю потребуется ввести название файла. Поиск файлов будет осуществляться по прописанной в коде папке на сервере.

Проведем получение файла от сервера при помощи Client2. Результат получения файла от сервера будет предоставлен на рисунке 3.6.

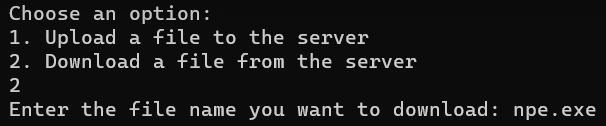


Рисунок 3.6 – Результат получения файла от сервера

На сервере так же отображается информация о том, что файл был отправлен. Информация, отображенная на сервере, будет представлена на рисунке 3.7.

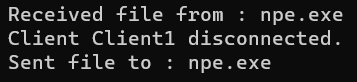


Рисунок 3.7 – Результат отправки файла от сервера

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы была построена система передачи файлов клиентами по сети с возможность выбора и отправки файлов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.
2. [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/procthread/process-and-thread-functions – Дата доступа: 01.11.2023](https://learn.microsoft.com/ru-ru/windows/win32/procthread/process-and-thread-functions%20–%20Дата%20доступа%2024.10.2023)
3. [Электронный ресурс]. – Режим доступа: https://metanit.com/cpp/tutorial/8.3.php – Дата доступа: 30.10.2023

**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

**ServerPart**

**Server.cpp**

#include <winsock2.h>

#include <windows.h>

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

// Структура для взаимодейтсвия с клиентом

struct ClientInfo

{

std::string name;

SOCKET socket;

};

std::vector<ClientInfo> clients;

// Обработка взаимодействий с клиентом

void HandleClient(ClientInfo client)

{

char buffer[1024]; // Буфер для хранения данных, прочитанных из сокета

int bytesRead; // Отслеживание количества байтов, полученных из сокета

while (true)

{

bytesRead = recv(client.socket, buffer, sizeof(buffer), 0); // считывание данных из сокета

if (bytesRead <= 0)

{

// Входим в ветвление, если клиент отключился, произошла ошибка в вводе запроса клиентом

for (auto it = clients.begin(); it != clients.end(); ++it)

{

if (it->socket == client.socket)

{

std::cout << "Client " << it->name << " disconnected." << std::endl;

closesocket(client.socket);

clients.erase(it);

break;

}

}

return;

}

// Определяем тип команды (UPLOAD или DOWNLOAD)

std::string command(buffer, bytesRead);

if (command == "UPLOAD")

{

// Получаем имя файла

bytesRead = recv(client.socket, buffer, sizeof(buffer), 0);

if (bytesRead <= 0)

{

std::cerr << "Error receiving file name." << std::endl;

return;

}

std::string fileName(buffer, bytesRead);

// Получаем и сохраняем файл

std::ofstream receivedFile(fileName, std::ios::out | std::ios::binary);

while ((bytesRead = recv(client.socket, buffer, sizeof(buffer), 0)) > 0)

{

receivedFile.write(buffer, bytesRead);

}

receivedFile.close();

std::cout << "Sent file to " << client.name << ": " << fileName << std::endl;

}

else if (command == "DOWNLOAD")

{

// Получаем имя файла для отправки

bytesRead = recv(client.socket, buffer, sizeof(buffer), 0);

if (bytesRead <= 0)

{

std::cerr << "Error receiving file name." << std::endl;

return;

}

std::string fileName(buffer, bytesRead);

// Отправить файл

std::ifstream fileStream(fileName, std::ios::in | std::ios::binary);

if (fileStream.is\_open())

{

while (!fileStream.eof())

{

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

bytesRead = static\_cast<int>(fileStream.gcount());

if (bytesRead > 0)

{

send(client.socket, buffer, bytesRead, 0);

}

}

fileStream.close();

std::cout << "Sent file to " << client.name << ": " << fileName << std::endl;

}

else

{

std::cerr << "Error opening file for sending: " << fileName << std::endl;

}

}

}

}

DWORD WINAPI HandleClientThread(LPVOID lpParam)

{

ClientInfo\* pClient = reinterpret\_cast<ClientInfo\*>(lpParam);

HandleClient(\*pClient);

return 0;

}

int main()

{

WSADATA wsaData; // Инициализация сокета

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

{

std::cerr << "Error initializing Winsock." << std::endl;

return 1;

}

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

if (serverSocket == INVALID\_SOCKET)

{

std::cerr << "Error creating a socket." << std::endl;

WSACleanup(); // Завершение работы библиотеки

return 1;

}

sockaddr\_in serverAddr; // Адрес сервера для прослушивания входящих сооединений

serverAddr.sin\_family = AF\_INET; // Семейство адресов для сервера

serverAddr.sin\_addr.s\_addr = INADDR\_ANY; // Определение IP адреса

serverAddr.sin\_port = htons(12345); // big-endian сетевой порядок байтов

// Привязка к заданному адресу и порту

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

{

std::cerr << "Error binding the socket to a port." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

// Настройка серверного сокета на прослушивание входящих соединений

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

{ // макс. длина очереди ожидания соединения

std::cerr << "Error listening to the socket." << std::endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "The server is running and listening on port 12345." << std::endl;

while (true)

{

sockaddr\_in clientAddr; // Адрес клиента для прослушивания входящих сообщений

int clientAddrSize = sizeof(clientAddr);

SOCKET clientSocket = accept(serverSocket, (sockaddr\*)&clientAddr, &clientAddrSize);

// заполнение структуры информацией о клиенте (IP адрес и порт)

if (clientSocket != INVALID\_SOCKET)

{

ClientInfo newClient;

newClient.socket = clientSocket;

char nameBuffer[1024];

int nameBytes = recv(clientSocket, nameBuffer, sizeof(nameBuffer), 0);

if (nameBytes > 0)

{

newClient.name.assign(nameBuffer, nameBytes);

}

std::cout << "Client " << newClient.name << " connected." << std::endl;

clients.push\_back(newClient);

HANDLE hThread = CreateThread(NULL, 0, HandleClientThread, &newClient, 0, NULL);

if (hThread == NULL)

{

std::cerr << "Error creating thread." << std::endl;

return 1;

}

}

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

**ClientPart2**

**Client2.cpp**

#include <winsock2.h>

#include <Ws2tcpip.h>

#include <windows.h>

#include <iostream>

#include <fstream>

#include <string>

// Отправка файлов

void SendFile(SOCKET clientSocket, const std::string& filePath)

{

char buffer[1024];

int bytesRead; // Байты, считанные из файла

std::string fileName = filePath.substr(filePath.find\_last\_of("\\/") + 1);

// Отправляем команду "UPLOAD" серверу

std::string uploadCommand = "UPLOAD";

// Команда возвращает -1 в случае ошибки

if (send(clientSocket, uploadCommand.c\_str(), uploadCommand.size(), 0) < 0)

{

std::cerr << "Error sending the upload command." << std::endl;

return;

}

// Отправляем имя файла серверу

if (send(clientSocket, fileName.c\_str(), fileName.size(), 0) < 0)

{

std::cerr << "Error sending the file name." << std::endl;

return;

}

std::ifstream fileStream(filePath, std::ios::in | std::ios::binary);

if (!fileStream.is\_open())

{

std::cerr << "Error opening file for reading: " << filePath << std::endl;

return;

}

while (!fileStream.eof())

{

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

bytesRead = static\_cast<int>(fileStream.gcount());

if (bytesRead > 0)

{

send(clientSocket, buffer, bytesRead, 0);

}

}

fileStream.close();

std::cout << "Sent file to the server: " << filePath << std::endl;

}

void ReceiveFile(SOCKET clientSocket)

{

char buffer[1024];

int bytesRead;

// Отправляем команду "DOWNLOAD" серверу

std::string downloadCommand = "DOWNLOAD";

if (send(clientSocket, downloadCommand.c\_str(), downloadCommand.size(), 0) < 0)

{

std::cerr << "Error sending the download command." << std::endl;

return;

}

std::cout << "Enter the file name you want to download: ";

std::string fileNameToDownload;

std::cin >> fileNameToDownload;

// Отправляем имя файла, который хотим скачать

if (send(clientSocket, fileNameToDownload.c\_str(), fileNameToDownload.size(), 0) < 0)

{

std::cerr << "Error sending the file name to download." << std::endl;

return;

}

std::string savePath = fileNameToDownload;

std::ofstream receivedFile(savePath, std::ios::out | std::ios::binary);

if (!receivedFile.is\_open())

{

std::cerr << "Error opening file for writing: " << savePath << std::endl;

return;

}

else

{

while ((bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0)) > 0)

{

receivedFile.write(buffer, bytesRead);

}

receivedFile.close();

std::cout << "Received and saved file: " << fileNameToDownload << std::endl;

}

}

int main() {

WSADATA wsaData;

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

{

std::cerr << "Error initializing Winsock." << std::endl;

return 1;

}

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

// Протокол TCP, конкретный протокол не указан, что будет использован tcp

if (clientSocket == INVALID\_SOCKET)

{

std::cerr << "Error creating a socket." << std::endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET; // Использование айпиви4

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

serverAddr.sin\_port = htons(12345);

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

{

std::cerr << "Error connecting to the server." << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::string name = "Client2"; // Уникальное имя клиента

send(clientSocket, name.c\_str(), name.size(), 0);

std::cout << "Choose an option:\n1. Upload a file to the server\n2. Download a file from the server\n";

int option;

std::cin >> option;

if (option == 1)

{

std::cout << "Enter the full path to the file you want to send: ";

std::string filePath;

std::cin >> filePath;

SendFile(clientSocket, filePath);

}

else if (option == 2)

{

ReceiveFile(clientSocket);

}

else

{

std::cerr << "Invalid option." << std::endl;

}

closesocket(clientSocket);

WSACleanup();

return 0;

}

return 0;

}