МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения  
и администрирования информационных систем

**Сетевая передача файлов с использованием сокетов в среде Windows**

Курсовая работа

Направление 02.03.03 – Математическое обеспечение  
и администрирование информационных систем

Зав. кафедрой д. ф.-м. н., проф. Артемов М. А.

Обучающийся 3 курс, 9 группа Маслова А. А.

Руководитель к. т. н., доц. Селезнев К. Е.

Воронеж 2024

# Содержание

[Введение 3](#_Toc170747208)

[1. Постановка задачи 4](#_Toc170747209)

[2. Анализ задачи 5](#_Toc170747210)

[2.1. Инициализация сетевого соединения 5](#_Toc170747211)

[2.2. Настройка сокетов 6](#_Toc170747212)

[2.3. Отправка файлов 6](#_Toc170747213)

[2.4. Приём файлов 7](#_Toc170747214)

[2.5. Обработка ошибок 7](#_Toc170747215)

[2.6. Управление ресурсами 8](#_Toc170747216)

[3. Средства реализации 9](#_Toc170747217)

[4. Требования к программному и аппаратному обеспечению 10](#_Toc170747218)

[5. Реализация 11](#_Toc170747219)

[5.1. Настройка и подготовка среды программирования 11](#_Toc170747220)

[5.2. Инициализация WinSock 11](#_Toc170747221)

[5.3. Настройка сокетов 12](#_Toc170747222)

[5.4. Передача и прием файлов 13](#_Toc170747223)

[5.5. Обработка исключений 13](#_Toc170747224)

[5.6. Администрирование ресурсов 17](#_Toc170747225)

[6. Пользовательский интерфейс 19](#_Toc170747226)

[Заключение 22](#_Toc170747227)

[Список литературы 23](#_Toc170747228)

[Листинг 24](#_Toc170747229)

# Введение

В эпоху глобализации и интенсивного развития информационных технологий, способность эффективно передавать информацию между различными устройствами через сеть становится критически важной. Данная работа посвящена разработке и реализации программы для сетевой передачи файлов с использованием технологии сокетов в *Windows*.

Основной целью проекта является создание и тщательный анализ программного решения, основанного на *WinSock API* для обеспечения сетевой коммуникации через *TCP/IP* протокол. Разработанная программа позволяет отправлять файлы в бинарном режиме, а также предоставляет инструменты для управления сетевыми операциями, включая прослушивание портов, установление соединений и передачу данных.

В ходе работы были рассмотрены и демонстрированы методы управления сокетами, процедуры обработки возможных ошибок сети и файловой системы, а также были использованы современные возможности языка программирования *C++* для работы с файлами. Такой подход позволяет не только решать практические задачи передачи данных, но и дает глубокое понимание механизмов сетевого взаимодействия.

Проект имеет значительное значение для понимания фундаментальных принципов сетевого программирования и предоставляет твердую основу для дальнейшей разработки приложений, которые требуют надежной и эффективной передачи данных по сети. В итоге, эта работа не только способствует техническому прогрессу в области сетевых технологий, но и поддерживает актуальность и адаптацию к постоянно изменяющимся требованиям информационного общества.

1. **Постановка задачи**

Необходимо реализовать программу для сетевой передачи файлов между двумя устройствами с использованием библиотеки *WinSock2* в операционной системе *Windows*. Проект должен поддерживать инициализацию и настройку сетевого соединения, а также обеспечивать возможность отправки и приёма файлов через созданные сокеты. Важной частью работы является обработка потенциальных ошибок сети и файловой системы, а также корректное управление ресурсами операционной системы для предотвращения утечек памяти и других ресурсов.

# Анализ задачи

## **Инициализация сетевого соединения**

Инициализация сетевого соединения является первым и ключевым шагом в создании сетевого приложения, использующего сокеты для обмена данными. В данной задаче это включает запуск и настройку сетевой библиотеки *WinSock2*. Она предоставляет *API* для разработки приложений, которые работают с сетями, включая интернет и локальные сети.

Процесс инициализации включает несколько важных этапов:

1. **Загрузка необходимых DLL:** *WinSock2* требует наличия соответствующих динамических библиотек (*DLL*), чтобы функции *API* были доступны для использования. Процесс загрузки *DLL* начинается с вызова функции *WSAStartup*, которая инициализирует использование библиотеки *WinSock*. Она должна быть вызвана перед использованием любых других функций библиотеки *WinSock*.
2. **Подготовка сетевого стека:** после загрузки *DLL* происходит инициализация сетевого стека, который будет использоваться для операций с сокетами. Это включает установку необходимых параметров и проверку совместимости версии библиотеки *WinSock* с версией, требуемой приложением. В случае успеха, приложение может продолжить настройку и использование сетевых функций.
3. **Обработка ошибок:** в процессе инициализации могут возникнуть различные ошибки, такие как отсутствие необходимых *DLL* или несовместимость версий. Программа должна предусматривать обработку таких ошибок, выводить соответствующие сообщения пользователю и корректно завершать работу в случае критических проблем.

## **Настройка сокетов**

Настройка сокетов включает создание, привязку и настройку параметров для работы с сетевыми адресами и портами. Это важный этап, позволяющий приложению устанавливать и принимать соединения.

Рассмотрим ключевые шаги:

1. **Создание сокетов:** сокет создается с помощью функции *socket*, которая возвращает дескриптор сокета. В параметрах функции указываются семейство адресов (например, *IPv4*), тип сокета (например, потоковый сокет для *TCP*) и протокол (обычно устанавливается в 0, чтобы использовать протокол по умолчанию для выбранного типа сокета).
2. **Привязка к сетевым адресам и портам:** с помощью функции *bind* сокет привязывается к конкретному сетевому адресу и порту. Это позволяет приложению принимать входящие соединения на указанном порту. Для серверной части важно выбрать правильный порт, который будет прослушиваться на наличие входящих соединений.
3. **Прослушивание входящих соединений:** функция *listen* переводит сокет в режим прослушивания, что позволяет ему принимать входящие соединения.
4. **Установление исходящих соединений:** для клиентской части процесс настройки сокета включает установление соединения с сервером с использованием функции *connect*. Здесь важно указать правильный адрес сервера и порт, на который нужно подключиться.

## **Отправка файлов**

Отправка файлов является основной задачей серверного приложения.

Этот процесс включает несколько этапов:

1. **Открытие файла:** сервер открывает файл, который нужно передать, в бинарном режиме для чтения. Для этого используется функция *fstream::open*.
2. **Определение размера файла:** для корректной передачи данных необходимо знать размер файла. Это делается с помощью функции *experimental::filesystem::file\_size*.
3. **Чтение данных файла:** данные файла считываются в буфер, который затем будет отправлен через сокет. Важно учитывать, что файл читается в бинарном режиме, чтобы избежать искажения данных.
4. **Отправка данных через сокет:** сначала отправляется размер файла, затем имя файла и, наконец, содержимое файла. Каждая часть отправляется по отдельности для облегчения процесса приема на стороне клиента.

## **Приём файлов**

Прием файлов является основной задачей клиентского приложения и включает следующие шаги:

1. **Получение размера файла:** клиент сначала принимает размер файла, чтобы знать, сколько данных ему нужно ожидать.
2. **Получение имени файла:** затем клиент принимает имя файла, чтобы знать, под каким именем сохранить файл в своей файловой системе.
3. **Получение содержимого файла:** основная часть данных принимается в виде массива байтов и сохраняется в файл на локальной машине.
4. **Сохранение файла:** полученные данные записываются в новый файл, открытый в бинарном режиме для записи.

## **Обработка ошибок**

Обработка ошибок критически важна для стабильности и надежности приложения. Программа должна предусматривать разные виды ошибок и корректно их обрабатывать:

1. **Ошибки сети:** эти ошибки могут возникать при инициализации сетевого соединения, при попытке привязки сокета к адресу, при установлении соединения и т.д. В таких случаях программа должна выводить сообщения об ошибке и, по возможности, предпринимать действия по восстановлению соединения.
2. **Ошибки файловой системы:** такие ошибки могут включать невозможность открытия файла, ошибки чтения/записи и т.д. Важно корректно обрабатывать такие ситуации, чтобы избежать потери данных и обеспечить целостность файлов.
3. **Ошибки памяти:** необходимо следить за корректным выделением и освобождением памяти, чтобы избежать утечек памяти и сбоев приложения.

## **Управление ресурсами**

Управление ресурсами включает адекватное закрытие всех открытых сокетов и файлов, а также корректное освобождение памяти. Это необходимо для предотвращения утечек ресурсов и обеспечения стабильной работы приложения:

1. **Закрытие сокетов:** все сокеты должны быть корректно закрыты после завершения работы с ними с использованием функции *closesocket*.
2. **Закрытие файлов:** все файлы должны быть закрыты после завершения операций чтения/записи, чтобы избежать утечек файловых дескрипторов.
3. **Освобождение памяти:** память, выделенная под буферы данных, должна быть освобождена с использованием оператора *delete[]*.
4. **Средства реализации**

В качестве языка программирования был выбран язык *С++.* Возможности данного языка предоставляют все необходимые инструменты для реализации поставленных в данной работе целей и задач. При реализации проекта использовалась среда разработки *Visual Studio 2022*.

1. **Требования к программному  
   и аппаратному обеспечению**

Приложение предназначено для использования компьютерах с операционной системой *Windows 7, Windows 8, Windows 10, Windows 11*.

Минимальные системные требования приложения соответствуют минимальным системным требованиям операционной системы.

Для получения наилучших результатов используйте средства диагностики с самым последним обновлением для вашей версии *Visual Studio*.

# Реализация

В данной части работы реализовано сетевое приложение. в котором происходит разделение на две функциональные части: *серверную*, которая выполняет отправку файлов, используя библиотеку *WinSock2* для управления сетевыми сокетами на платформе *Windows*, и *клиентскую*, которая их принимает.

## **Настройка и подготовка среды программирования**

В обоих проектах (клиентском и серверном) происходит подключение необходимых библиотек для работы с файлами и сетевыми функциями и настройка среды разработки:

#define \_SILENCE\_EXPERIMENTAL\_FILESYSTEM\_DEPRECATION\_WARNING

#include<iostream>

#include<fstream>

#include<string>

#include<experimental/filesystem>

#pragma warning(disable: 4996)

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

#include<WinSock2.h>

## **Инициализация WinSock**

Инициализация библиотеки *WinSock* происходит в функции *main* обеих программ:

* 1. WORD dllVer = MAKEWORD(2, 1); — эта строка задаёт версию *WinSock API*, которую хотим использовать. Здесь указана версия 2.1, которая подходит для большинства современных сетевых приложений.
  2. WSAData wsad; — объявляется переменная типа *WSAData*, которая будет использоваться для получения информации о *WinSock* *DLL* во время выполнения *WSAStartup*.
  3. WSAStartup(dllVer, &wsad); — вызов функции *WSAStartup* инициализирует использование *WinSock* *DLL*, передавая версию и адрес структуры *WSAData*. Это необходимо для подготовки операционной системы к сетевым операциям через сокеты.

## **Настройка сокетов**

**Серверная часть (Server.sln)**

SOCKADDR\_IN addr\_info;

memset(&addr\_info, 0, sizeof(SOCKADDR\_IN));

int size\_addr = sizeof(addr\_info);

addr\_info.sin\_port = htons(4321);

addr\_info.sin\_family = AF\_INET;

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

bind(s\_listen, (sockaddr\*)&addr\_info, sizeof(addr\_info));

listen(s\_listen, SOMAXCONN);

SOCKET s\_for\_connect = accept(s\_listen, (sockaddr\*)&addr\_info, &size\_addr);

Здесь сервер настраивает порт и тип адреса, создает сокет для *TCP* соединений, привязывает его к порту и начинает прослушивание на этом порту. После этого сервер готов принимать подключения клиентов, ожидая запросы на подключение.

**Клиентская часть (Client.sln)**

SOCKADDR\_IN addr\_info;

memset(&addr\_info, 0, sizeof(SOCKADDR\_IN));

addr\_info.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

addr\_info.sin\_port = htons(4321);

addr\_info.sin\_family = AF\_INET;

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

connect(s\_client, (sockaddr\*)&addr\_info, sizeof(addr\_info));

Этот блок инициализирует структуру *SOCKADDR\_IN*, устанавливает *IP-адрес* сервера (локальный адрес), номер порта для подключения, тип сетевого адреса (*IPv4*) и создает *TCP* сокет для установления соединения. Используемый порт *4321* должен совпадать с портом, который прослушивает сервер.

## **Передача и прием файлов**

Функция *send\_file* используется сервером для отправки файла клиенту. Она начинается с открытия файла в бинарном режиме для чтения, определяет его размер и читает содержимое в буфер. Затем последовательно отправляются размер файла, его имя, и само содержимое через сокет. После отправки, буфер удаляется и файл закрывается, освобождая ресурсы.

В функции *recv\_file*, клиент принимает данные файла от сервера, начиная с размера файла, затем имя и само содержимое, которое записывается в новый файл на стороне клиента. После записи всех данных, буфер освобождается и файл закрывается.

Эти блоки обеспечивают передачу файлов между сервером и клиентом, используя функции *WinSock* для управления сетевыми соединениями.

## **Обработка исключений**

Для обеспечения надежности работы программы и упрощения процесса отладки необходимо обрабатывать возможные ошибки и информировать пользователя о них.

**Проверка инициализация WinSock**

if (WSAStartup(dllVer, &wsad) != 0) {

cout << "WSAStartup failed\n";

return 1;

}

Эта проверка убеждается, что инициализация библиотеки *WinSock* прошла успешно. Если *WSAStartup* возвращает ненулевое значение, это означает, что инициализация не удалась. В таком случае программа выводит сообщение об ошибке и завершает выполнение.

**Проверка создания сокета**

if (s\_listen == INVALID\_SOCKET) {

cout << "Socket creation failed: " << WSAGetLastError() << endl;

WSACleanup();

return 1;

}

Эта проверка удостоверяется, что сокет был успешно создан. Если *socket* возвращает *INVALID\_SOCKET*, это означает, что создание сокета не удалось. В таком случае программа выводит сообщение об ошибке, очищает ресурсы *WinSock* вызовом *WSACleanup()* и завершает выполнение.

**Проверка привязки сокета**

if (bind(s\_listen, (sockaddr\*)&addr\_info, sizeof(addr\_info)) == SOCKET\_ERROR) {

cout << "Bind failed: " << WSAGetLastError() << endl;

closesocket(s\_listen);

WSACleanup();

return 1;

}

Эта проверка убеждается, что сокет был успешно привязан к адресу. Если *bind* возвращает *SOCKET\_ERROR*, это означает, что привязка не удалась. В таком случае программа выводит сообщение об ошибке, закрывает сокет вызовом *closesocket(s\_listen),* очищает ресурсы *WinSock* и завершает выполнение.

**Проверка прослушивания сокета**

if (listen(s\_listen, SOMAXCONN) == SOCKET\_ERROR) {

cout << "Listen failed: " << WSAGetLastError() << endl;

closesocket(s\_listen);

WSACleanup();

return 1;

}

Эта проверка удостоверяется, что сокет успешно начал прослушивание входящих соединений. Если *listen* возвращает *SOCKET\_ERROR*, это означает, что прослушивание не удалось. В таком случае программа выводит сообщение об ошибке, закрывает сокет, очищает ресурсы *WinSock* и завершает выполнение.

**Проверка принятия соединения**

if (s\_for\_connect == INVALID\_SOCKET) {

cout << "Accept failed: " << WSAGetLastError() << endl;

closesocket(s\_listen);

WSACleanup();

return 1;

}

Эта проверка удостоверяется, что сокет успешно принял входящее соединение. Если *accept* возвращает *INVALID\_SOCKET*, это означает, что принятие соединения не удалось. В таком случае программа выводит сообщение об ошибке, закрывает сокет, очищает ресурсы *WinSock* и завершает выполнение.

**Проверка создания клиентского сокета**

if (s\_client == INVALID\_SOCKET) {

cout << "Socket creation failed: " << WSAGetLastError() << endl;

WSACleanup();

return 1;

}

Эта проверка убеждается, что клиентский сокет был успешно создан. Если *socket* возвращает *INVALID\_SOCKET*, это означает, что создание сокета не удалось. В таком случае программа выводит сообщение об ошибке, очищает ресурсы *WinSock* и завершает выполнение.

**Проверка подключения к серверу**

if (connect(s\_client, (sockaddr\*)&addr\_info, sizeof(addr\_info)) == SOCKET\_ERROR) {

cout << "Connect failed: " << WSAGetLastError() << endl;

closesocket(s\_client);

WSACleanup();

return 1;

}

Эта проверка удостоверяется, что клиентский сокет успешно подключился к серверу. Если *connect* возвращает *SOCKET\_ERROR*, это означает, что подключение не удалось. В таком случае программа выводит сообщение об ошибке, закрывает клиентский сокет, очищает ресурсы *WinSock* и завершает выполнение.

**Проверка открытия файла**

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

cout << "Error opening file: " << file\_name << endl;

return;

}

Эта проверка убеждается, что файл был успешно открыт. Если *file.is\_open()* возвращает *false*, это означает, что файл не удалось открыть. В таком случае программа выводит сообщение об ошибке и завершает выполнение текущей функции, возвращаясь в место вызова. Это предотвращает дальнейшие попытки работы с файлом, который не был открыт.

**Проверка отправки данных**

if (send(\*sock, to\_string(file\_size).c\_str(), 16, 0) == SOCKET\_ERROR ||

send(\*sock, file\_name.c\_str(), 32, 0) == SOCKET\_ERROR ||

send(\*sock, bytes, file\_size, 0) == SOCKET\_ERROR) {

cout << "Send failed: " << WSAGetLastError() << endl;

}

Эта проверка используется для отправки данных через сокет. Если любая из функций *send* возвращает *SOCKET\_ERROR*, это означает, что произошла ошибка при отправке данных. В таком случае программа выводит сообщение об ошибке с кодом ошибки, полученным из *WSAGetLastError().*

**Проверка приема данных**

if (recv(\*sock, file\_size\_str, 16, 0) == SOCKET\_ERROR) {

cout << "Receive file size failed: " << WSAGetLastError() << endl;

return;

}

if (recv(\*sock, file\_name, 32, 0) == SOCKET\_ERROR) {

cout << "Receive file name failed: " << WSAGetLastError() << endl;

return;

}

if (recv(\*sock, bytes, file\_size, 0) == SOCKET\_ERROR) {

cout << "Receive file data failed: " << WSAGetLastError() << endl;

delete[] bytes;

file.close();

return;

}

Эти проверки удостоверяются, что данные были успешно приняты через сокет. Если любая из функций *recv* возвращает *SOCKET\_ERROR*, это означает, что прием данных не удался. В таком случае программа выводит сообщение об ошибке, очищает память, закрывает файл и возвращается в место вызова, предотвращая дальнейшую обработку неверно принятых данных.

## **Администрирование ресурсов**

Очень важным этапом является реализация системных ресурсов.

**Закрытие сокетов**

После завершения работы с сокетами их необходимо корректно закрыть.

closesocket(s\_listen);

closesocket(s\_for\_connect);

closesocket(s\_client);

**Очистка ресурсов WinSock**

После завершения работы с *WinSock* необходимо вызвать функцию *WSACleanup*, чтобы освободить занятые ресурсы.

WSACleanup();

**Закрытие файлов**

Файлы, открытые для чтения и записи, должны быть корректно закрыты после завершения операций.

file.close();

**Освобождение памяти**

Память, выделенная динамически для хранения данных, должна быть освобождена.

delete[] buffer;

# Пользовательский интерфейс

В данном разделе представлены иллюстрации демонстрирующие работу программы в различных сценариях использования. Эти изображения иллюстрируют процесс взаимодействия клиента и сервера, а также результаты выполнения ключевых функций, таких как отправка и прием файлов.

На рис. 1 показана директория сервера, содержащая файл *photo.jpg*. Это изображение подтверждает, что файл готов к отправке клиенту.

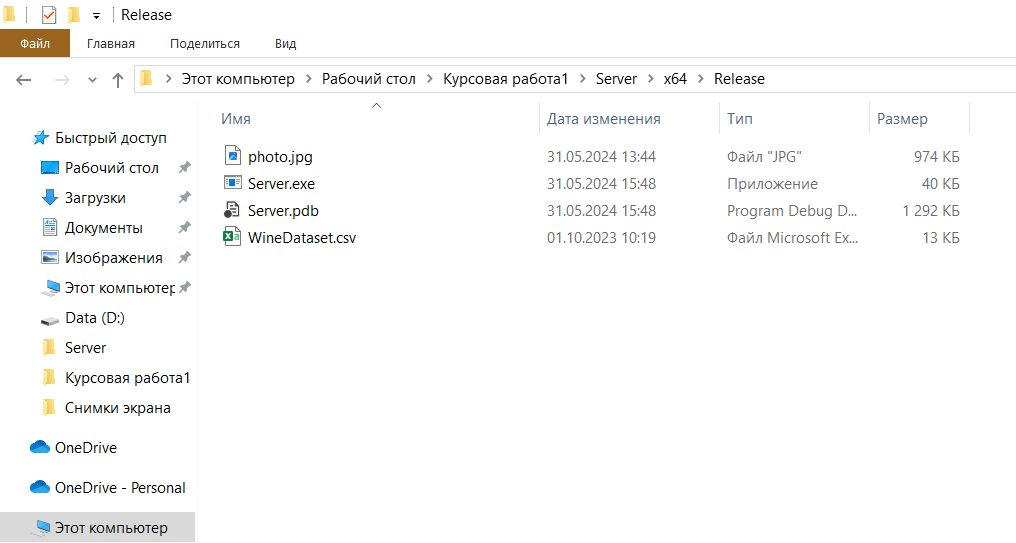


Рис. 1. Содержимое директории сервера перед началом передачи файла

Подтверждение того, что в директории перед началом сеанса файл отсутствует, представлено на рис. 2:

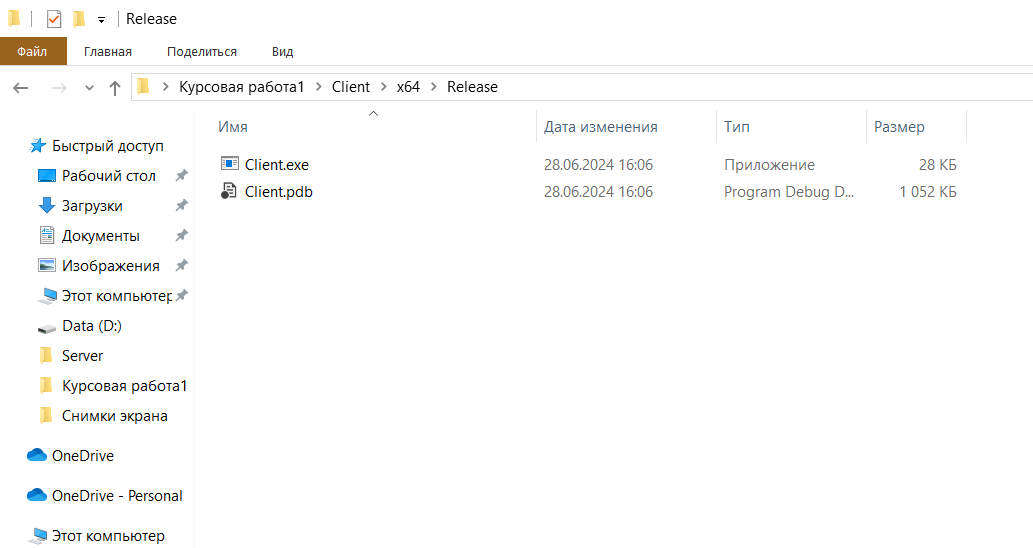


Рис. 2. Содержимое директории клиента до получения файла

После запуска программ начинается процесс установления соединения между клиентом и сервером. Успешное соединение визуализируется через сообщение *CONNECT* на консолях, как показано на рис. 3. Это подтверждение соединения указывает на готовность системы к обмену данными, что является критическим моментом для начала передачи файлов между участниками сети.

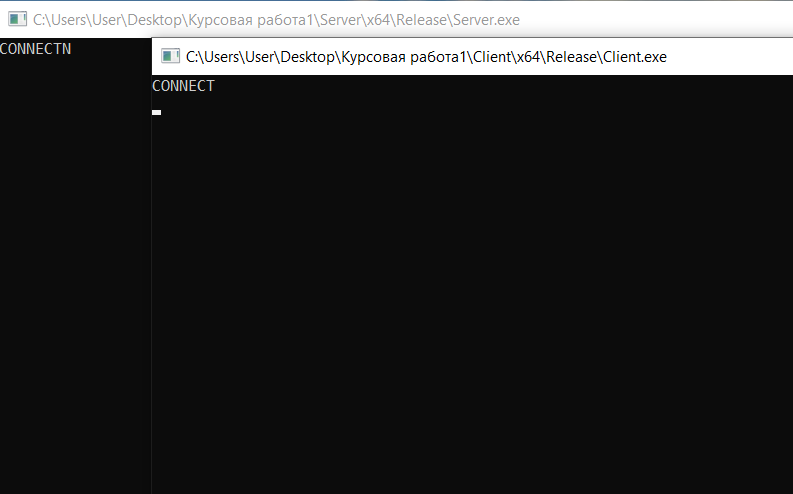


Рис. 3. Демонстрация успешного соединение между клиентом и сервером

После успешной передачи файла сервер представляет детали завершенной операции, включая размер и имя файла, что демонстрируется на рис. 4. Клиент же подтверждает прием файла, отображая уведомление о его успешном сохранении в локальной файловой системе. Эта визуализация подчеркивает бесперебойное взаимодействие сетевых приложений, гарантируя, что файлы передаются и сохраняются без потерь.

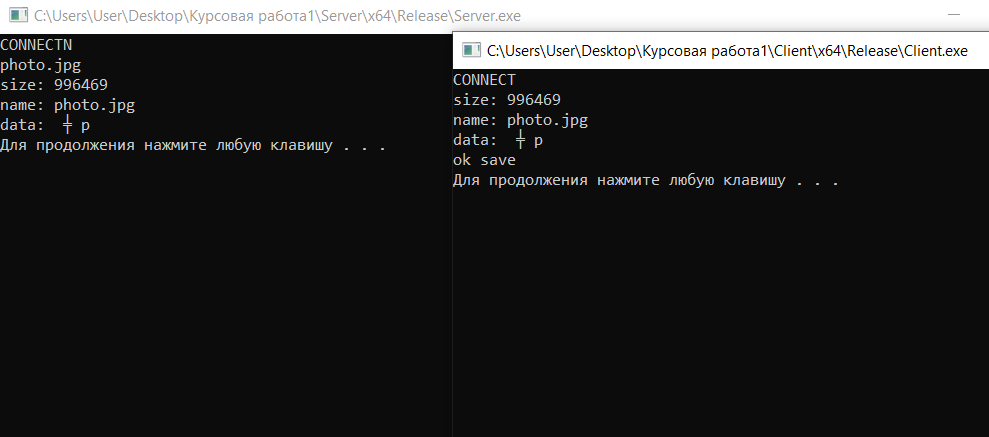


Рис. 4. Детали успешно переданного файла

И на рис. 5 в директории клиента показано наличие принятого файла:

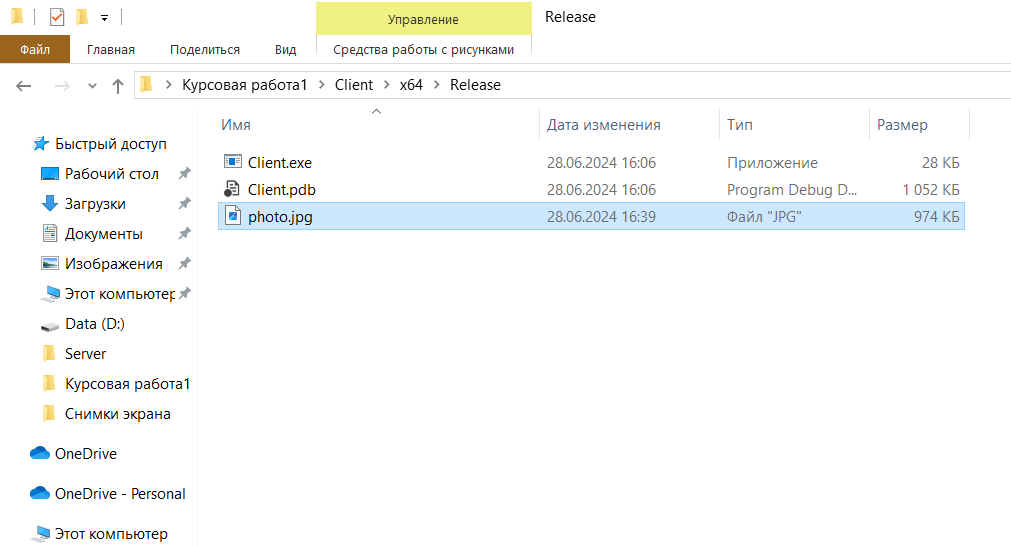


Рис. 5. Содержимое директории клиента после получения файла

**Заключение**

В ходе выполнения курсовой работы было успешно разработано сетевое приложение на языке программирования *C++*, использующее *Windows* *Sockets API (WinSock2)* для передачи файлов между двумя устройствами по сети. Программа демонстрирует возможность эффективной работы как в роли клиента, так и сервера, обеспечивая при этом стабильную передачу данных в бинарном формате.

Основные достижения проекта:

* 1. **Реализация клиент-серверного взаимодействия**: приложение успешно реализует как отправку, так и приём файлов, что демонстрирует его двусторонний характер. Внедрение функций send\_file и recv\_file позволило обеспечить обмен файлами между устройствами.
  2. **Обработка ошибок и исключений**: в программе предусмотрены механизмы обработки возможных ошибок сетевого соединения и файловой системы, что повышает её надёжность и устойчивость к исключительным ситуациям.
  3. **Простота и масштабируемость**: Программа разработана с учётом принципов простоты и масштабируемости, что позволяет легко адаптировать её под различные задачи и условия эксплуатации.

Результаты работы подтверждают, что цель достигнута: разработано функциональное сетевое приложение для передачи файлов, которое может быть использовано в различных практических сценариях. Разработанное приложение может служить основой для дальнейших исследований в области сетевого программирования, включая внедрение асинхронных технологий, расширение поддерживаемых протоколов и улучшение механизмов безопасности.

**Список литературы**

* 1. Стивенс В. Р. TCP/IP. Иллюстрированное руководство. Том 1. Протоколы: пер. с англ. – М.: Издательский дом «Вильямс», 2016. – 600 с.
  2. Комер Д. Е. Сети TCP/IP. Том 1. Принципы, протоколы и архитектура: пер. с англ. – М.: Лори, 2014. – 750 с.
  3. Райт Г. Р., Стивенс В. Р. TCP/IP. Иллюстрированное руководство. Том 2. Реализация: пер. с англ. – СПб.: Питер, 2017. – 850 с.
  4. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: СПб.: Питер, 2019. – 960 с.
  5. Харитонов Е. И., Шелестов А. И. Сетевые технологии: М.: Академия, 2018. – 384 с.
  6. Windows Sockets 2 API [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/windows/win32/winsock/windows-sockets-start-page-2>. – Дата доступа: 26.06.2024.
  7. RFC 793 - Протокол управления передачей [Электронный ресурс]. – Режим доступа: <https://rfc2.ru/793.rfc>. – Дата доступа: 26.06.2024.
  8. RFC 791 - Интернет-протокол [Электронный ресурс]. – Режим доступа: <https://rfc2.ru/791.rfc>. – Дата доступа: 26.06.2024.
  9. Козырев В. А. Программирование сетей на языке С++: М.: ДМК Пресс, 2020. – 320 с.
  10. Маклафлин Б., Ползунов Д. (перевод) Основы сокетов: М.: Символ-Плюс, 2021. – 512 с.
  11. Habr: Сообщество IT-специалистов [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/hub/networks/>. – Дата доступа: 26.06.2024.
  12. Технострим Mail.ru Group: курсы по сетевому программированию [Электронный ресурс]. – Режим доступа: <https://tech.mail.ru/blog/>. – Дата доступа: 26.06.2024.

**Листинг**

Server.sln

#define \_SILENCE\_EXPERIMENTAL\_FILESYSTEM\_DEPRECATION\_WARNING

#include<iostream>

#include<fstream>

#include<string>

#include<experimental/filesystem>

#pragma warning(disable: 4996)

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

#include<WinSock2.h>

using namespace std;

void send\_file(SOCKET\* sock, const string& file\_name) {

fstream file;

file.open(file\_name, ios\_base::in | ios\_base::binary);

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

cout << "Error opening file: " << file\_name << endl;

return;

}

else {

int file\_size = experimental::filesystem::file\_size(file\_name) + 1;

char\* bytes = new char[file\_size];

file.read(bytes, file\_size);

cout << "size: " << file\_size << endl;

cout << "name: " << file\_name << endl;

cout << "data: " << bytes << endl;

if (send(\*sock, to\_string(file\_size).c\_str(), 16, 0) == SOCKET\_ERROR ||

send(\*sock, file\_name.c\_str(), 32, 0) == SOCKET\_ERROR ||

send(\*sock, bytes, file\_size, 0) == SOCKET\_ERROR) {

cout << "Send failed: " << WSAGetLastError() << endl;

}

delete[] bytes;

}

file.close();

}

int main() {

WORD dllVer = MAKEWORD(2, 1);

WSAData wsad;

if (WSAStartup(dllVer, &wsad) != 0) {

cout << "WSAStartup failed\n";

return 1;

}

SOCKADDR\_IN addr\_info;

memset(&addr\_info, 0, sizeof(SOCKADDR\_IN));

int size\_addr = sizeof(addr\_info);

addr\_info.sin\_port = htons(4321);

addr\_info.sin\_family = AF\_INET;

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

if (s\_listen == INVALID\_SOCKET) {

cout << "Socket creation failed: " << WSAGetLastError() << endl;

WSACleanup();

return 1;

}

if (bind(s\_listen, (sockaddr\*)&addr\_info, sizeof(addr\_info)) == SOCKET\_ERROR) {

cout << "Bind failed: " << WSAGetLastError() << endl;

closesocket(s\_listen);

WSACleanup();

return 1;

}

if (listen(s\_listen, SOMAXCONN) == SOCKET\_ERROR) {

cout << "Listen failed: " << WSAGetLastError() << endl;

closesocket(s\_listen);

WSACleanup();

return 1;

}

SOCKET s\_for\_connect = accept(s\_listen, (sockaddr\*)&addr\_info, &size\_addr);

if (s\_for\_connect == INVALID\_SOCKET) {

cout << "Accept failed: " << WSAGetLastError() << endl;

closesocket(s\_listen);

WSACleanup();

return 1;

}

cout << "CONNECT\n";

string path;

cin >> path;

send\_file(&s\_for\_connect, path);

closesocket(s\_listen);

closesocket(s\_for\_connect);

WSACleanup();

system("pause");

return 0;

}

Client.sln

#include<iostream>

#include<fstream>

#pragma warning(disable:4996)

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

#include<WinSock2.h>

using namespace std;

void recv\_file(SOCKET\* sock) {

char file\_size\_str[16];

char file\_name[32];

if (recv(\*sock, file\_size\_str, 16, 0) == SOCKET\_ERROR) {

cout << "Receive file size failed: " << WSAGetLastError() << endl;

return;

}

int file\_size = atoi(file\_size\_str);

char\* bytes = new char[file\_size];

if (recv(\*sock, file\_name, 32, 0) == SOCKET\_ERROR) {

cout << "Receive file name failed: " << WSAGetLastError() << endl;

return;

}

fstream file;

file.open(file\_name, ios\_base::out | ios\_base::binary);

cout << "size: " << file\_size << endl;

cout << "name: " << file\_name << endl;

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

cout << "Error opening file: " << file\_name << endl;

delete[] bytes;

return;

}

else {

if (recv(\*sock, bytes, file\_size, 0) == SOCKET\_ERROR) {

cout << "Receive file data failed: " << WSAGetLastError() << endl;

delete[] bytes;

file.close();

return;

}

cout << "data: " << bytes << endl;

file.write(bytes, file\_size);

cout << "ok save\n";

delete[] bytes;

}

file.close();

}

int main() {

WORD dllVer = MAKEWORD(2, 1);

WSAData wsad;

if (WSAStartup(dllVer, &wsad) != 0) {

cout << "WSAStartup failed\n";

return 1;

}

SOCKADDR\_IN addr\_info;

memset(&addr\_info, 0, sizeof(SOCKADDR\_IN));

addr\_info.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

addr\_info.sin\_port = htons(4321);

addr\_info.sin\_family = AF\_INET;

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

if (s\_client == INVALID\_SOCKET) {

cout << "Socket creation failed: " << WSAGetLastError() << endl;

WSACleanup();

return 1;

}

if (connect(s\_client, (sockaddr\*)&addr\_info, sizeof(addr\_info)) == SOCKET\_ERROR) {

cout << "Connect failed: " << WSAGetLastError() << endl;

closesocket(s\_client);

WSACleanup();

return 1;

}

cout << "CONNECT\n";

recv\_file(&s\_client);

closesocket(s\_client);

WSACleanup();

system("pause");

return 0;

}