9- Практическое задание — Клиент-серверное приложение на C++ с использованием протокола TCP

Table of Contents

Цель

Целью данной лабораторной работы является изучение принципов создания клиент-серверных приложений на языке программирования C++. Сначала, используя предоставленный полный код и объяснения, студенты научатся программировать клиент-серверное приложение на основе протокола TCP. После этого, для задачи самостоятельной реализации и используя знания, полученные в этой и предыдущих лабораторных работах, студентам требуется разработать серверное приложение, которое может одновременно обрабатывать запросы от нескольких клиентов с использованием многопоточности. Сервер будет принимать от клиентов имена файлов и предоставлять их содержимое, тем самым предоставляя возможность изучения файловой системы и сетевого взаимодействия в рамках одного приложения. Такой подход дает студентам возможность приобрести практические навыки разработки программ, которые могут быть использованы в реальных проектах, связанных с обработкой данных и параллельным программированием.

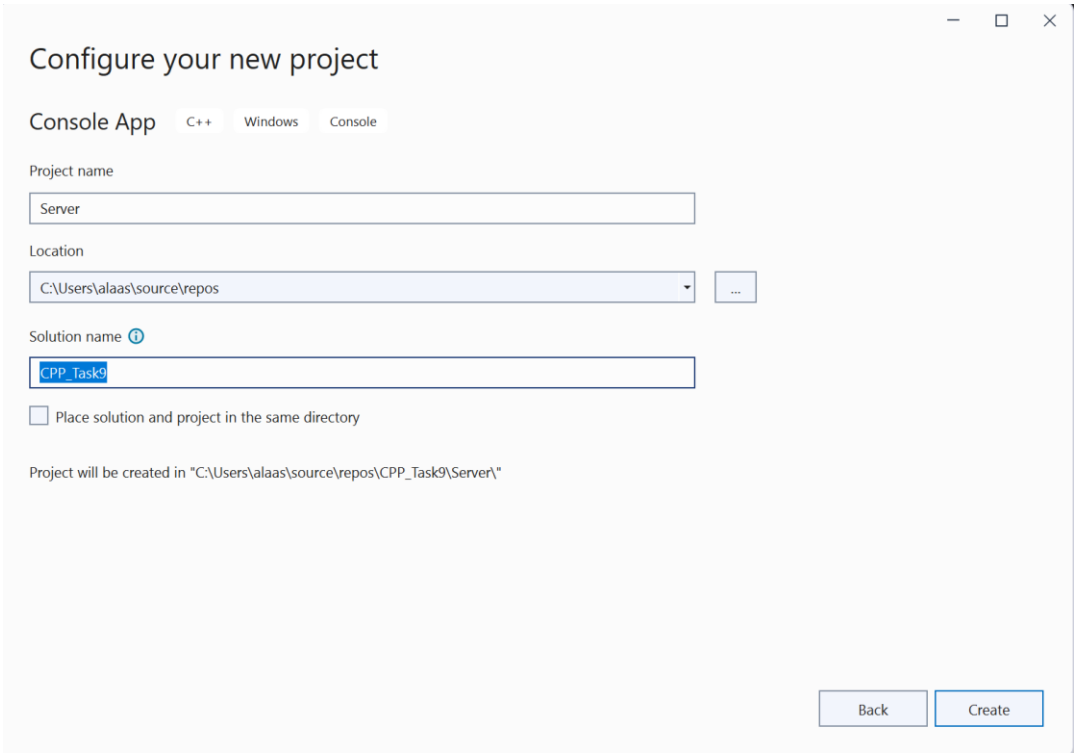
1. Клиент-серверное приложение с использованием протокола TCP

Введение:

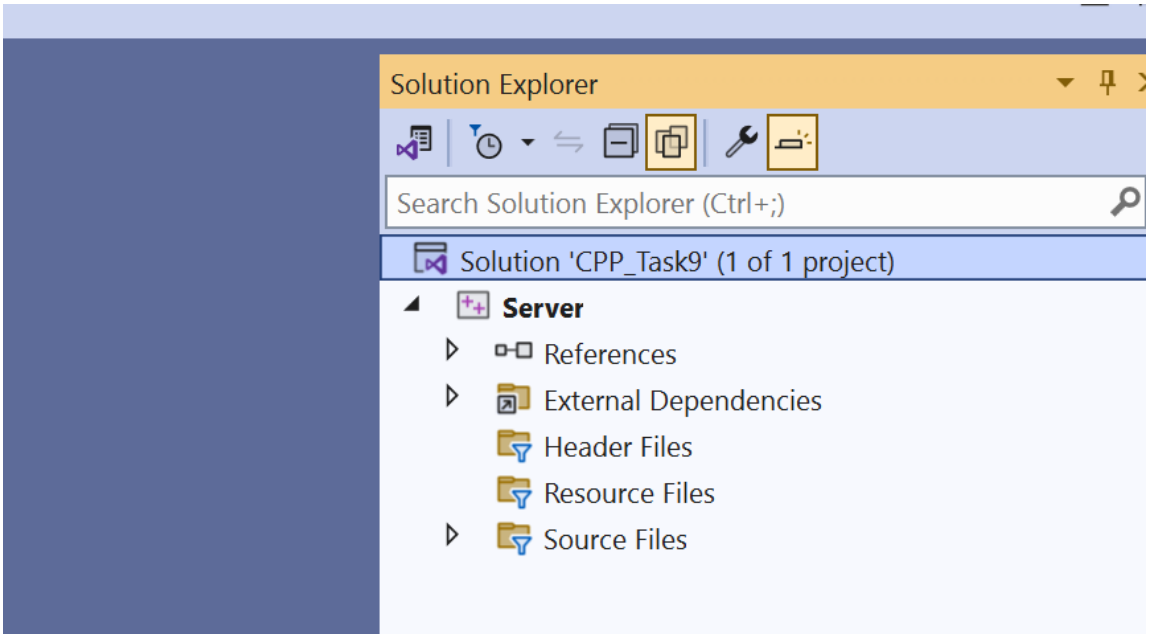
- **Сеть** — это система связанных между собой компонентов (как правило, компьютеров), которые обмениваются данными и ресурсами. Сети могут быть локальными (LAN) и охватывать небольшие пространства, например, офис, или же глобальными (WAN), соединяя устройства по всему миру.
- **Интернет** — это глобальная система взаимосвязанных компьютерных сетей, которая использует стандартизированный набор протоколов (TCP/IP) для связи между миллиардами устройств по всему миру. Это сеть сетей, которая обеспечивает разнообразные информационные и коммуникационные услуги.
- **IP**, или **Интернет-протокол**, — это протокол сетевого уровня стека TCP/IP, который определяет правила и стандарты передачи данных между компьютерами в сети Интернет. IP-адрес представляет собой уникальный адрес, который присваивается каждому устройству, подключенному к сети, для идентификации и возможности обмена данными.
- **TCP** (**Протокол управления передачей**) — это один из основных протоколов передачи данных в сетях, который обеспечивает надежную, упорядоченную и проверяемую доставку данных от отправителя к получателю. TCP устанавливает соединение между двумя хостами и гарантирует, что данные будут доставлены без ошибок и в правильной последовательности.
- **Сокет** — это программный интерфейс (API) для обеспечения обмена данными между процессами через сеть. В контексте сетевого программирования сокет представляет собой конечную точку для отправки или получения данных в сетевом соединении.
- **Сервер** — это компьютерная программа или устройство, предоставляющее данные, ресурсы или услуги другим программам или устройствам, называемым клиентами. В контексте сетевых взаимодействий сервер обычно слушает входящие запросы на определенном порту и обрабатывает их, предоставляя запрошенную информацию или услугу.
- **Клиент** — это программа или устройство, которое запрашивает данные или услуги у сервера. Клиент инициирует соединение и отправляет запрос серверу, ожидая от него ответа или обслуживания. Клиентские приложения включают веб-браузеры, почтовые клиенты и многие другие типы программного обеспечения, которые получают ресурсы от серверов через сеть.

Начало разработки первого клиент-серверного приложения

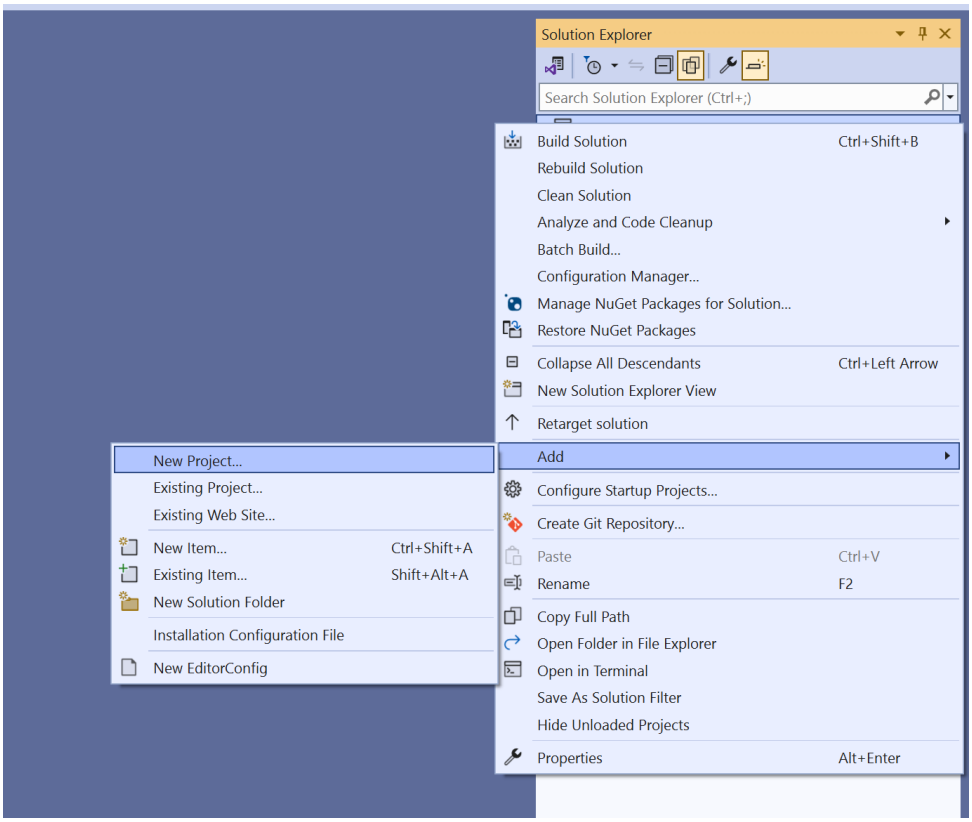
1- Создайте новый консольный проект на C++, обратите внимание, что имя проекта должно быть (Server), в то время как имя решения должно отличаться; например, вы можете использовать (CPP_Task9), как показано на:



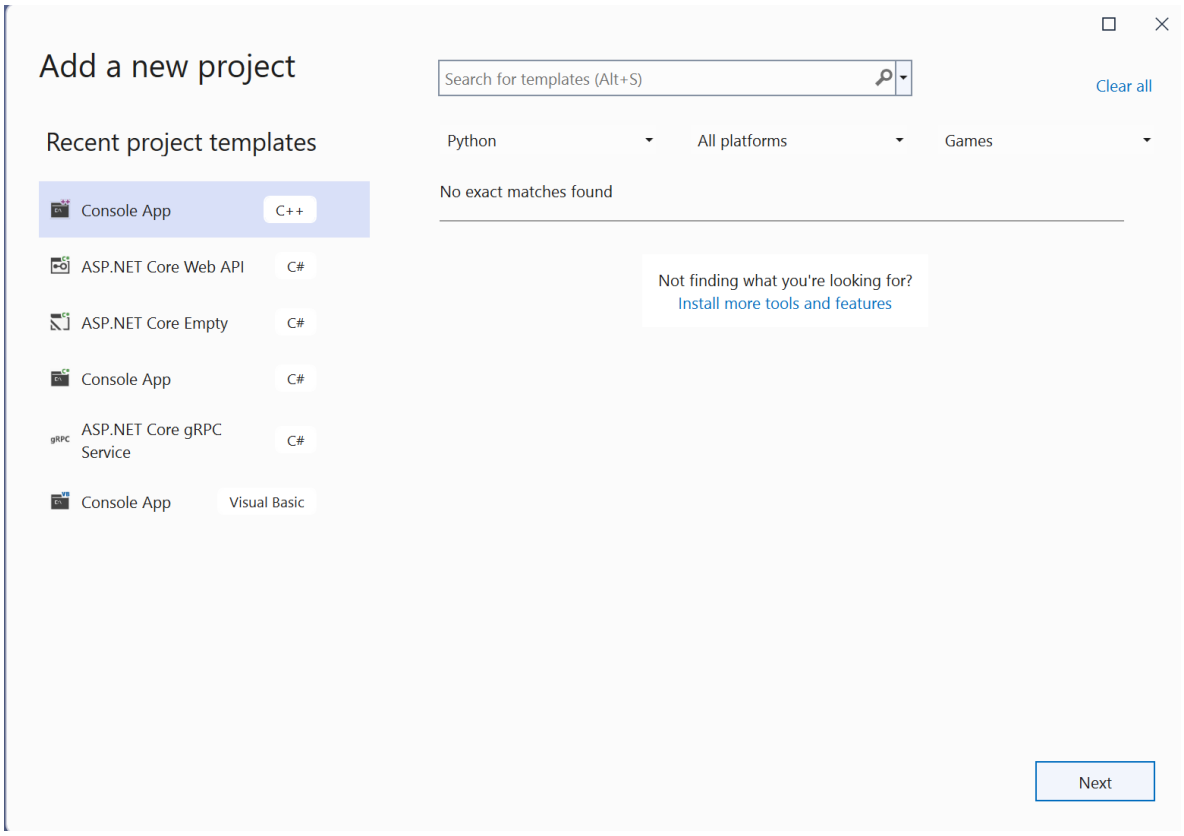
2- В окне (Solution explorer) структура проекта и решения будет примерно следующей, как показано на следующем скриншот:



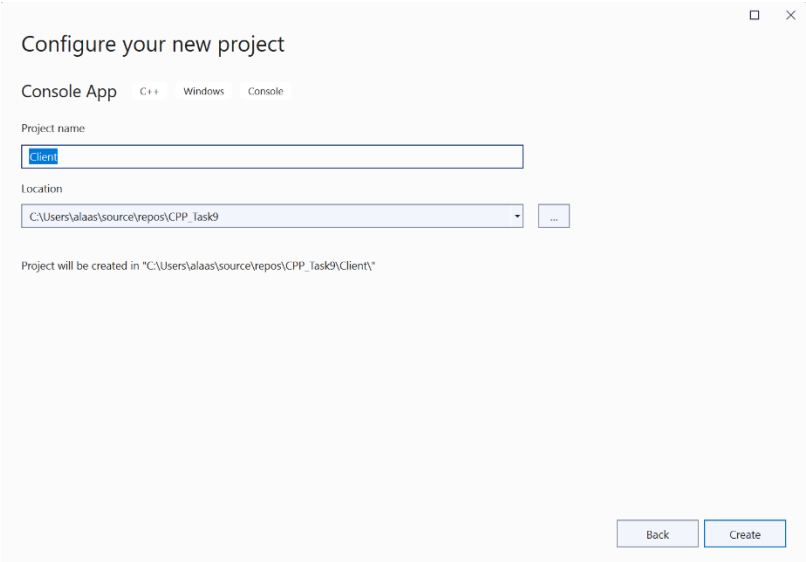
3- Обратите внимание на следующее: в окне (Solution explorer) щелкните правой кнопкой мыши по имени решения (CPP_Task9), а не по проекту внутри решения. Из контекстного меню выберите (add) выберите (Новый проект), как показано на следующем скриншоте:



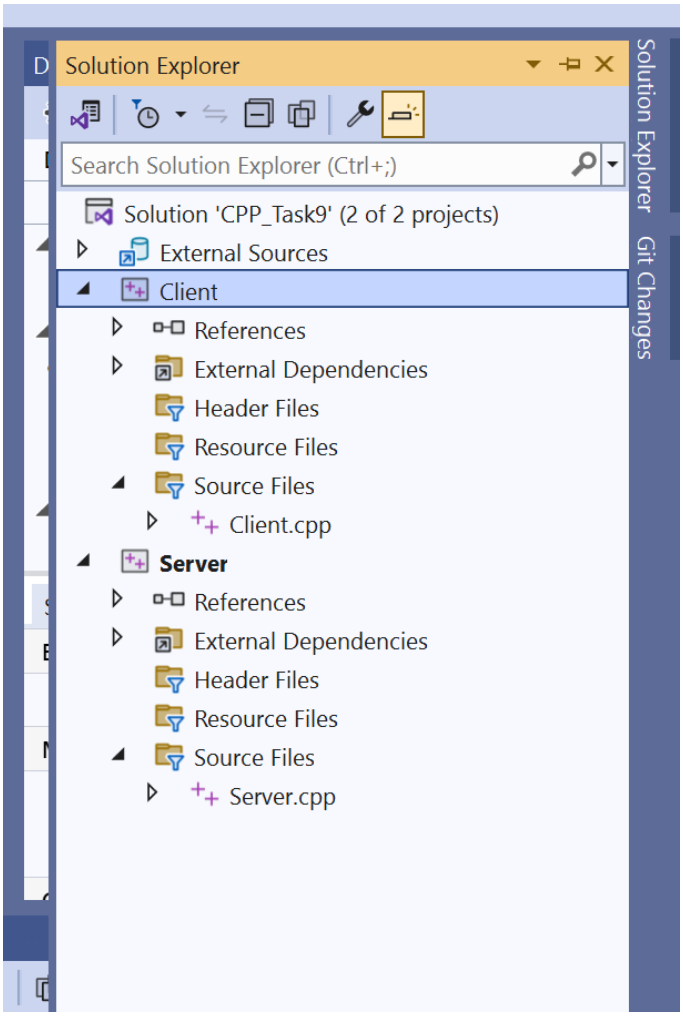
4- Окно (Добавить новый проект) отобразится перед вами, после чего вам нужно выбрать (Консольное приложение) из опций слева, а затем нажать на кнопку (Далее) в нижнем правом углу окна, как показано на следующем скриншоте:



5- Из окна (Настройка вашего нового проекта) выберите имя для добавленного проекта, например, в этом случае он будет называться (Client), а затем нажмите кнопку (Создать) в нижнем правом углу окна, как показано на следующем скриншоте:



6- После предыдущих шагов вы увидите, что в окне (Обозреватель решений) отображается, что наше решение называется (CPP_Task9) и теперь содержит два проекта, один из них - (Server), а другой - (Client). Это разные проекты, но внутри одного решения, и каждый из них может быть запущен независимо. Пожалуйста, посмотрите на следующий скриншот:



Полный код сервера:

```
#include <iostream> // Подключение библиотеки для работы с потоками ввода/вывода.
#include <Winsock2.h> // Подключение библиотеки для работы с сокетами Windows.
#include <Ws2tcpip.h> // Подключение библиотеки для работы с адресами IP.

#pragma comment(lib, "Ws2_32.lib") // Директива компилятору для связывания с библиотекой Ws2_32.lib.

// Объявление класса TCPServer для создания TCP сервера.
class TCPServer {
private:
    SOCKET server_fd; // Сокет сервера.
    struct sockaddr_in address; // Структура адреса для сокета.
    int addrlen = sizeof(address); // Размер структуры адреса.
    const int port = 8080; // Номер порта, который будет слушать сервер.

public:
    // Конструктор класса TCPServer.
    TCPServer() {
        WSADATA wsaData; // Структура для хранения информации о реализации Windows Sockets.
        int opt = 1; // Переменная для установки параметров сокета.

        // Инициализация библиотеки Windows Sockets.
        // WSAStartup - функция для инициализации работы с сокетами в Windows.
        // MAKEWORD(2, 2) - макрос для создания номера версии Winsock, который мы хотим использовать (2.2).
        // &wsaData - указатель на структуру WSADATA, которая получит информацию о реализации сокетов Windows.
        // Функция возвращает 0 при успешной инициализации.
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            std::cerr << "WSAStartup failed with error: " << WSAGetLastError() << std::endl; // Вывод ошибки,
если инициализация не удалась.
            exit(EXIT_FAILURE); // Завершение программы с кодом ошибки.
        }

        // Создание сокета.
        // socket - функция создания сокета.
        // AF_INET - семейство адресов IPv4.
        // SOCK_STREAM - тип сокета для обеспечения надежной, упорядоченной и безошибочной доставки данных
(TCP).
        // 0 - протокол, выбирается автоматически.
        // INVALID_SOCKET - специальное возвращаемое значение, указывающее на ошибку.
        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
            std::cerr << "Socket creation failed with error: " << WSAGetLastError() << std::endl;
            WSACleanup(); // Очистка ресурсов Winsock.
            exit(EXIT_FAILURE);
        }

        // Настройка структуры адресов.
        address.sin_family = AF_INET; // Использование сети IPv4.
        address.sin_addr.s_addr = INADDR_ANY; // Автоматический выбор IP-адреса.
        address.sin_port = htons(port); // Установка порта для прослушивания, конвертация в сетевой порядок
байт.

        // Установка опций сокета.
        // setsockopt - функция для установки опций сокета.
        // server_fd - дескриптор сокета.
        // SOL_SOCKET - уровень опций сокета.
        // SO_REUSEADDR - позволяет повторно использовать локальные адреса.
        // (char*)&opt - указатель на переменную с новым значением опции.
        // sizeof(opt) - размер переменной с опцией.
        // SOCKET_ERROR - возвращаемое значение при ошибке.
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, (char*)&opt, sizeof(opt)) == SOCKET_ERROR) {
            std::cerr << "setsockopt failed with error: " << WSAGetLastError() << std::endl;
            closesocket(server_fd); // Закрытие сокета.
            WSACleanup(); // Очистка ресурсов Winsock.
            exit(EXIT_FAILURE);
        }

        // Привязка сокета к IP-адресу и порту.
        // bind - функция привязывает сокет к конкретному IP-адресу и порту на локальном компьютере.
        // server_fd - дескриптор сокета, который был создан ранее.
```

```
        // (struct sockaddr*)&address - указатель на структуру sockaddr_in, которая содержит адрес, к которому
будет привязан сокет.
        // sizeof(address) - размер структуры адреса. Необходим для корректной работы функции.
        // SOCKET_ERROR - константа, значение которой означает, что операция завершилась с ошибкой.
        if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) == SOCKET_ERROR) {
            std::cerr << "Bind failed with error: " << WSAGetLastError() << std::endl; // Вывод сообщения об
ошибке.
            closesocket(server_fd); // Закрытие сокета.
            WSACleanup(); // Освобождение ресурсов Winsock.
            exit(EXIT_FAILURE); // Завершение программы с кодом ошибки.
        }

        // Перевод сокета в режим прослушивания.
        // listen - функция переводит сокет в состояние прослушивания входящих соединений.
        // server_fd - дескриптор сокета, который будет прослушивать.
        // 3 - максимальное количество одновременных ожидающих соединений в очереди.
        if (listen(server_fd, 3) == SOCKET_ERROR) {
            std::cerr << "Listen failed with error: " << WSAGetLastError() << std::endl; // Сообщение об ошибке
при неудаче.
            closesocket(server_fd); // Закрытие сокета в случае ошибки.
            WSACleanup(); // Освобождение ресурсов Winsock.
            exit(EXIT_FAILURE); // Завершение программы с кодом ошибки.
        }
    }

    // Метод для запуска сервера и обработки входящих подключений.
    void run() {
        std::cout << "Waiting for connections..." << std::endl; // Вывод сообщения о начале ожидания
подключений.

        SOCKET client_socket; // Дескриптор сокета клиента.
        // accept - функция ожидает входящее подключение на сокете.
        // server_fd - дескриптор прослушивающего сокета.
        // (struct sockaddr*)&address - указатель на структуру, куда будет записан адрес подключившегося
клиента.
        // (socklen_t*)&addrlen - указатель на переменную, в которую будет записан размер адресной структуры.
        if ((client_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) ==
INVALID_SOCKET) {
            std::cerr << "Accept failed with error: " << WSAGetLastError() << std::endl; // Сообщение об
ошибке.
            closesocket(server_fd); // Закрытие сокета сервера.
            WSACleanup(); // Освобождение ресурсов Winsock.
            exit(EXIT_FAILURE); // Выход из программы с кодом ошибки.
        }

        char buffer[1024] = { 0 }; // Буфер для приема данных.
        // recv - функция получения данных из сокета.
        // client_socket - дескриптор сокета клиента, от которого принимаются данные.
        // buffer - указатель на буфер, куда будут записаны принятые данные.
        // sizeof(buffer) - размер буфера.
        // 0 - флаги, настройки функции recv (в данном случае флаги не используются).
        int bytes_received = recv(client_socket, buffer, sizeof(buffer), 0); // Получение данных от клиента.
        if (bytes_received == SOCKET_ERROR) {
            std::cerr << "recv failed with error: " << WSAGetLastError() << std::endl; // Сообщение об ошибке
при приеме данных.
            closesocket(client_socket); // Закрытие сокета клиента.
        }
        else {
            std::cout << "Message from client: " << buffer << std::endl; // Вывод сообщения от клиента.
        }

        closesocket(client_socket); // Закрытие сокета клиента после завершения обработки.
    }

    // Деструктор класса.
    ~TCPServer() {
        closesocket(server_fd); // Закрытие сокета сервера.
        WSACleanup(); // Очистка ресурсов Winsock.
    }
};
```

```
int main() {
    TCPServer server; // Создание экземпляра сервера.
    server.run(); // Запуск сервера.
    return 0; // Возвращение кода успешного завершения.
}
```

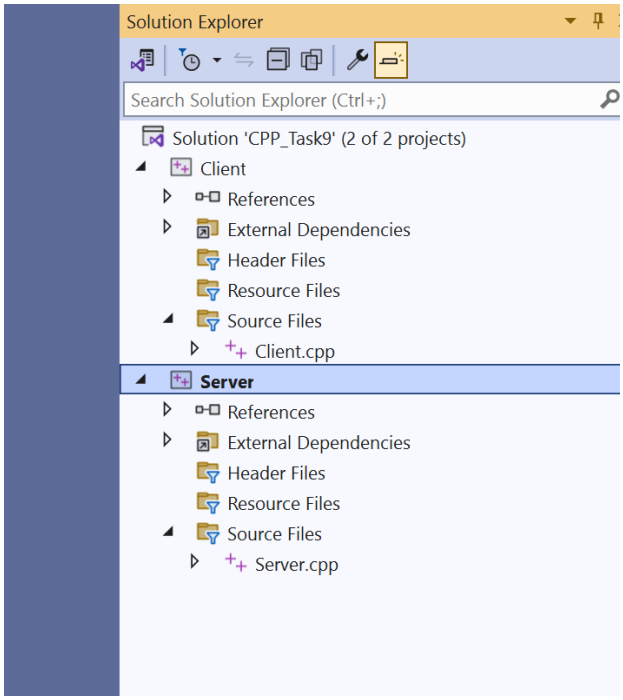
Полный код клиента:

```
#include <iostream> // Включение стандартной библиотеки для ввода и вывода в C++.
#include <Winsock2.h> // Включение библиотеки Winsock2 для работы с сетевыми функциями на Windows.
#include <Ws2tcpip.h> // Включение дополнительной библиотеки для работы с IP адресами.

#pragma comment(lib, "Ws2_32.lib") // Инструкция для компоновщика о необходимости подключения библиотеки
Ws2_32.lib.

// Объявление класса TCPClient, предназначенного для создания TCP клиента.
class TCPClient {
private:
    SOCKET sock; // Объявление сокета для клиента.
    struct sockaddr_in serv_addr; // Структура, содержащая информацию об адресе сервера.
    const char* message = "Hello from client"; // Сообщение, которое будет отправлено серверу.

public:
    // Конструктор класса, принимающий IP адрес и порт сервера.
    TCPClient(const char* address, int port) {
        WSADATA wsaData; // Структура для хранения данных о Windows Sockets.

        // Инициализация библиотеки Winsock.
        // WSAStartup инициализирует использование Winsock DLL со спецификацией версии 2.2.
        // MAKEWORD(2, 2) создает номер версии Winsock, 2.2 в данном случае.
        // &wsaData - указатель на структуру WSADATA, которая будет заполнена информацией о Winsock.
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            std::cerr << "WSAStartup failed." << std::endl; // Вывод сообщения об ошибке при неудачной
инициализации.
            exit(EXIT_FAILURE); // Выход из программы с ошибкой.
        }

        // Создание сокета для клиента.
        // Функция socket создает сокет, который является конечной точкой для общения в сети.
        // AF_INET означает, что используется сеть для коммуникации с IPv4 адресами.
        // SOCK_STREAM указывает на использование протокола TCP, обеспечивающего надежную потоковую передачу
данных.
        // Последний аргумент 0 указывает, что используется протокол по умолчанию для AF_INET, SOCK_STREAM, то
есть TCP.
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
            std::cerr << "Socket creation failed." << std::endl; // Сообщение об ошибке в случае неудачи
создания сокета.
            WSACleanup(); // Вызов WSACleanup для освобождения ресурсов Winsock, занятых приложением.
            exit(EXIT_FAILURE); // Завершение программы с кодом ошибки, указывающим на неудачное выполнение.
        }

        // Задание параметров адреса сервера.
        serv_addr.sin_family = AF_INET; // Указание семейства адресов IPv4.
        // htons конвертирует номер порта из хостового порядка байт в сетевой порядок байт, что необходимо для
корректной работы сетевых функций.
        serv_addr.sin_port = htons(port);

        // Преобразование текстового IP-адреса в двоичный формат.
        // Функция inet_pton преобразует IP-адреса из текстового представления (например, "127.0.0.1") в
числовое (структура in_addr).
        // AF_INET указывает на то, что преобразование производится для IPv4 адреса.
        // В случае успеха inet_pton возвращает 1, в случае ошибки - 0 или -1.
        if (inet_pton(AF_INET, address, &serv_addr.sin_addr) <= 0) {
            std::cerr << "Invalid address: Address not supported." << std::endl; // Вывод сообщения об ошибке
при неверном адресе.
            closesocket(sock); // Закрытие сокета в случае ошибки.
            WSACleanup(); // Освобождение ресурсов Winsock.
```

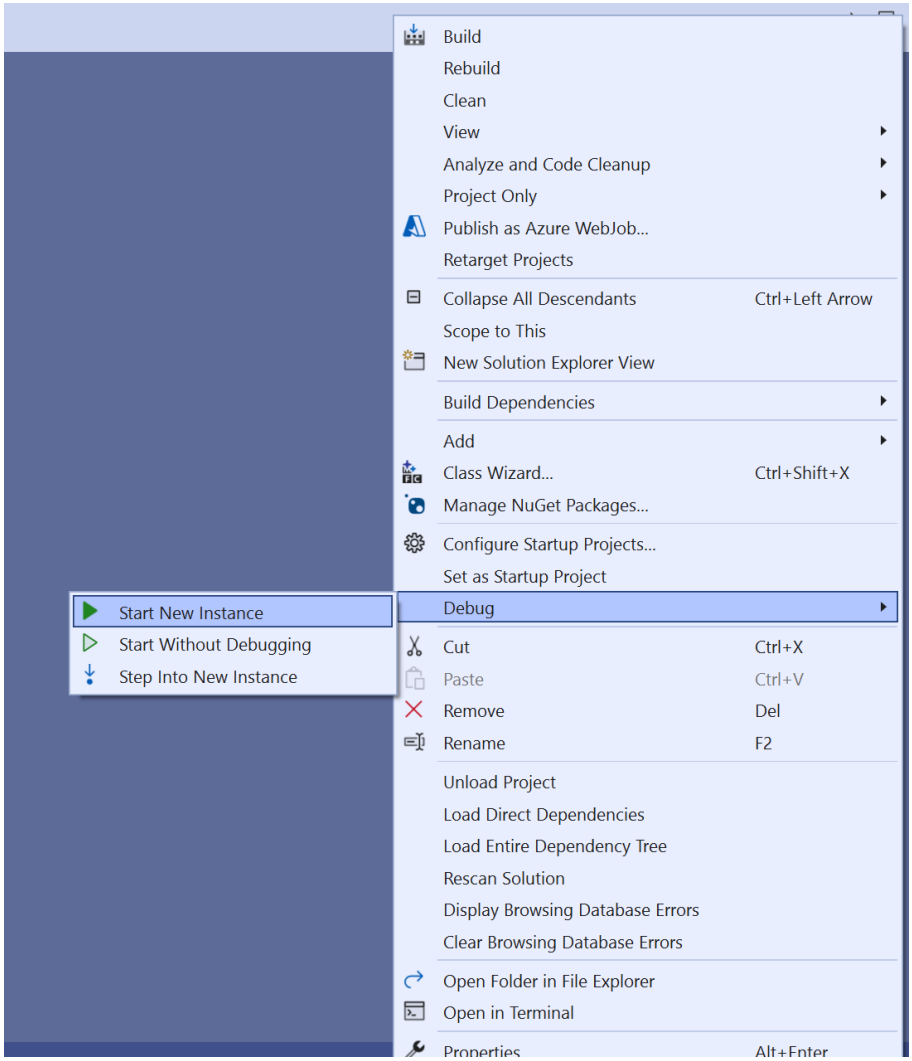
```
            exit(EXIT_FAILURE); // Выход из программы с ошибкой.
        }

        // Установление соединения с сервером.
        // Функция connect инициирует установление соединения на сокете, указывая на сокет клиента, адрес
сервера и размер структуры адреса.
        // Соединение является двусторонним, и после успешного выполнения, клиент может отправлять данные и
получать ответы.
        // Если connect возвращает значение меньше 0, это означает ошибку при попытке соединения.
        if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
            std::cerr << "Connection Failed." << std::endl; // Вывод сообщения о неудаче при попытке
соединения.
            closesocket(sock); // Закрытие сокета в случае неудачи.
            WSACleanup(); // Очистка ресурсов Winsock.
            exit(EXIT_FAILURE); // Завершение работы программы из-за неудачи соединения.
        }


        // Отправка сообщения серверу.
        // send отправляет данные (message) через сокет.
        // strlen(message) - длина сообщения для отправки.
        // 0 - флаги; здесь не используются.
        send(sock, message, strlen(message), 0);
        std::cout << "Message sent" << std::endl; // Подтверждение отправки сообщения.

        closesocket(sock); // Закрытие сокета после отправки сообщения.
        WSACleanup(); // Очистка ресурсов Winsock.
    }
};

int main() {
    TCPClient client("127.0.0.1", 8080); // Создание экземпляра клиента и подключение к серверу на локальном
хосте с портом 8080.
    return 0; // Возвращение 0 указывает на успешное завершение программы.
}
```

Запуск сервера и клиента:

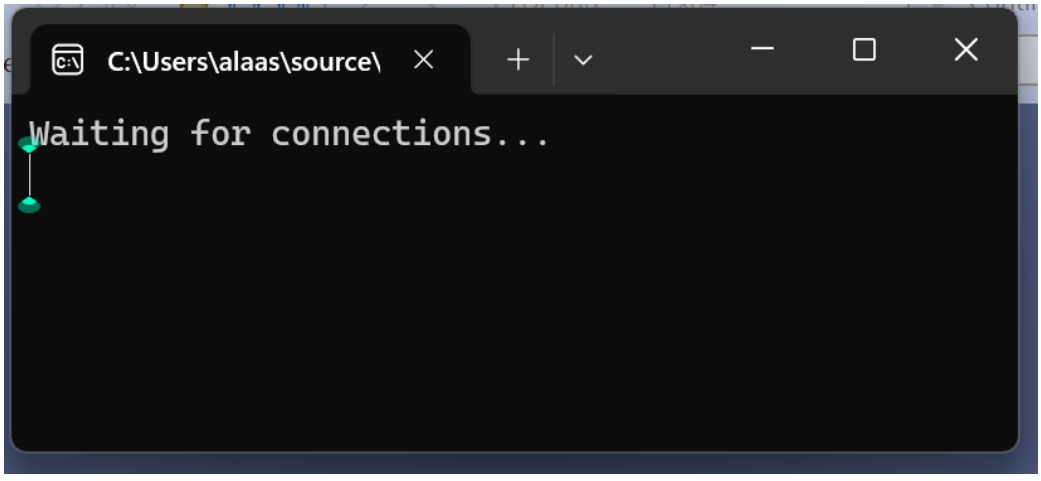
1- Сначала вам нужно запустить сервер независимо. Для этого сначала щелкните по проекту (Server), чтобы выбрать его, как показано на следующем скриншоте:



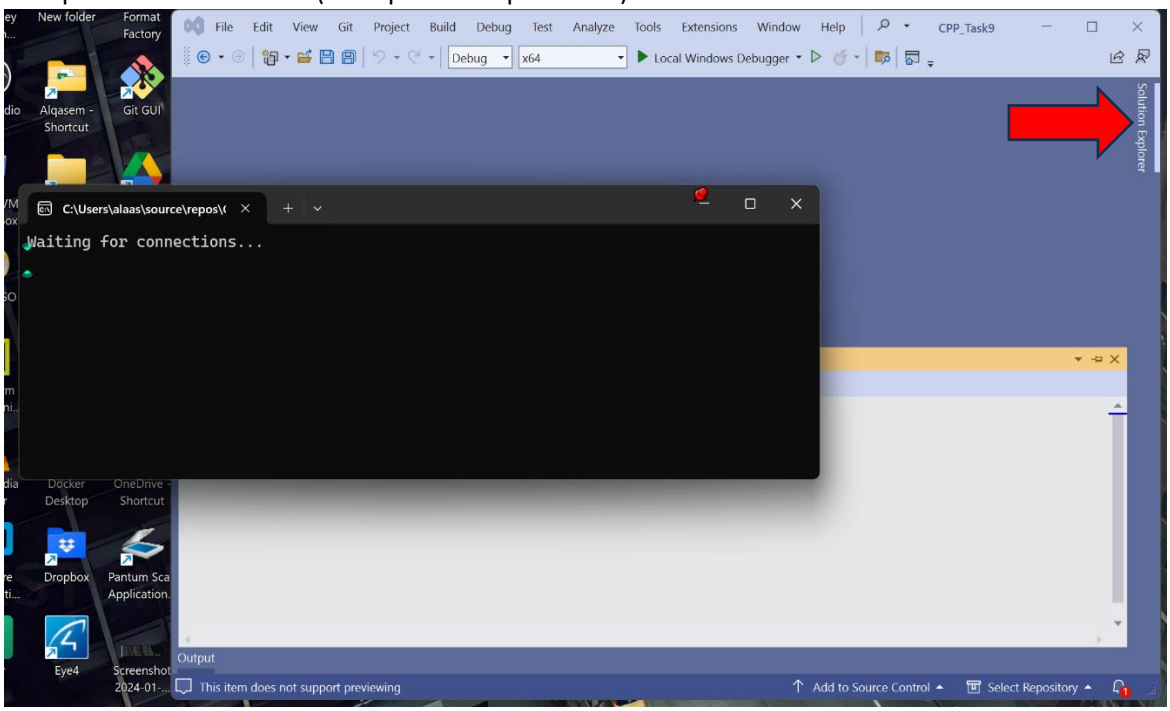
2- После того как вы выбрали проект (Server), вам нужно щелкнуть правой кнопкой мыши по проекту сервера, и из контекстного меню выберите (Отладка), а затем из подменю (Отладка) выберите (Запустить новый экземпляр), как показано на следующем скриншоте:



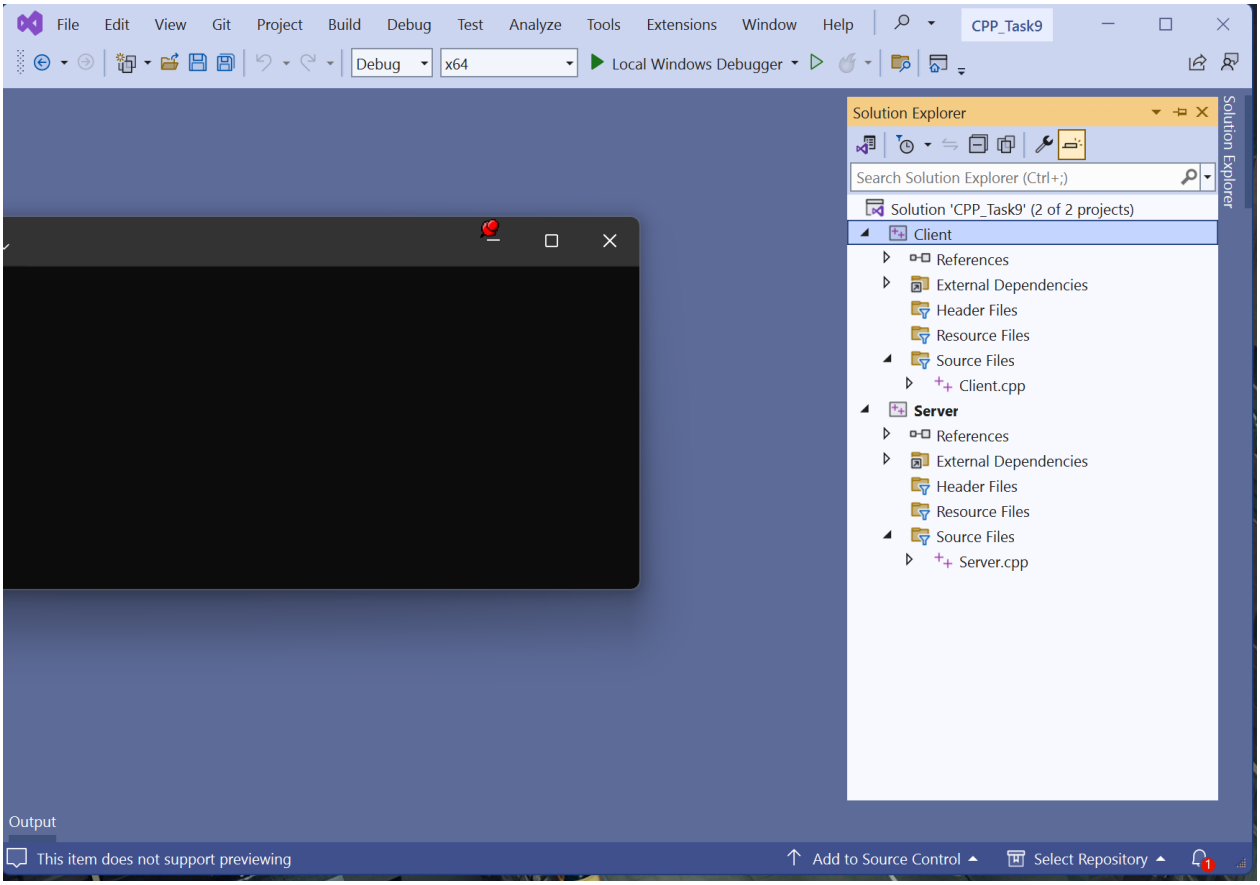
3- После запуска сервера появится консоль выполнения сервера, которая будет ожидать сообщения от клиента, как показано на следующем скриншоте (Пожалуйста, не закрывайте её, потому что нам нужно держать сервер запущенным, чтобы запустить клиент параллельно с сервером):



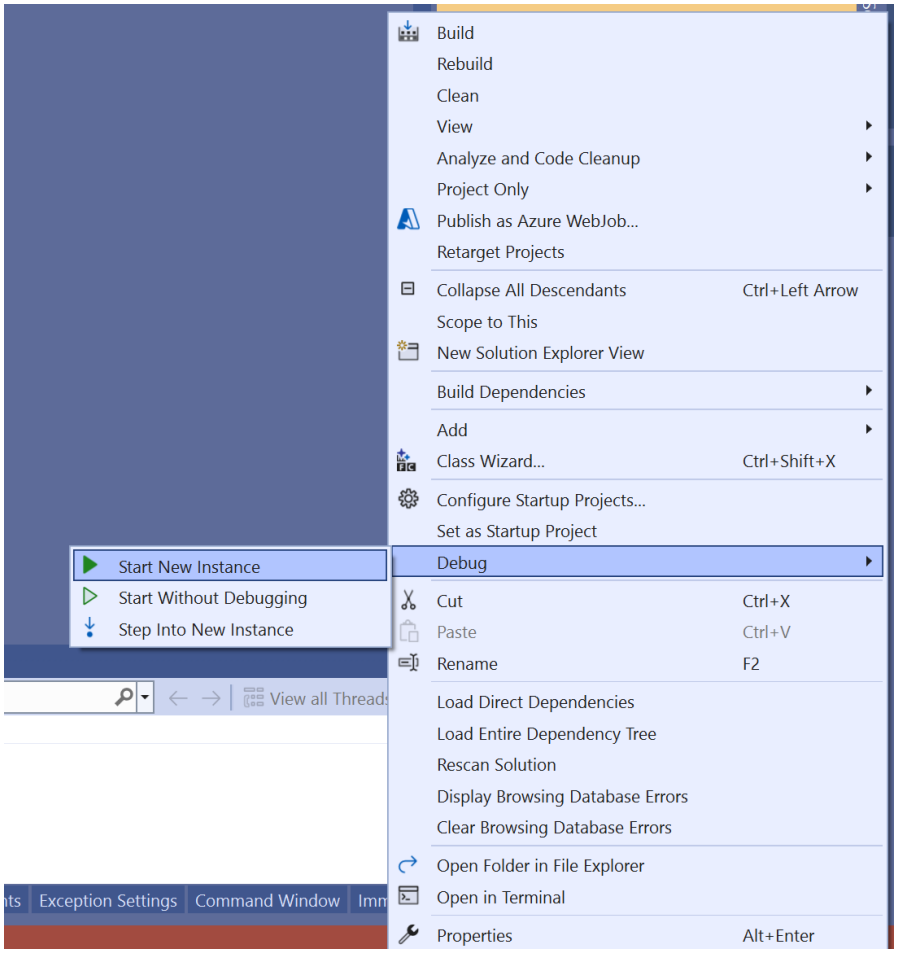
4- Теперь, пока сервер работает, перейдите в главное окно Visual Studio, и из верхнего правого окна Visual Studio, пожалуйста, щелкните по (Обозреватель реше́ний - Solution Explorer), чтобы снова отобразить полное окно (Обозреватель реше́ний).



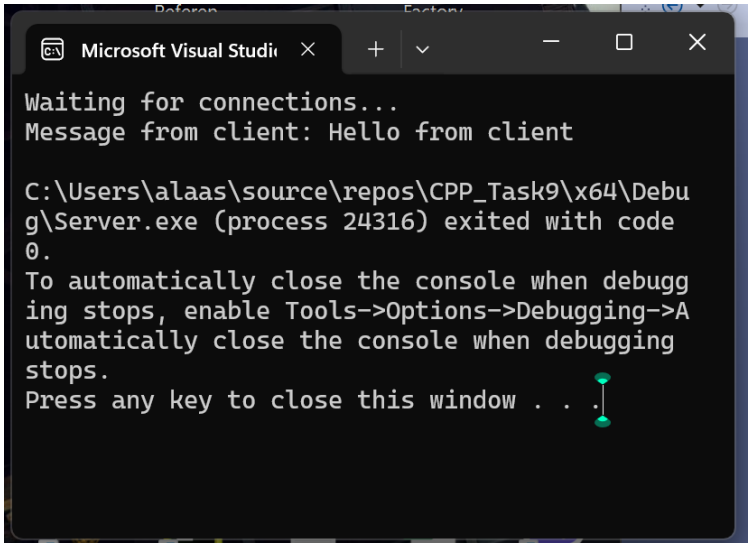
5- Снова, пока сервер работает и окно (Обозреватель решений) отображается, вам нужно выбрать проект (Client), щелкнув по нему, как показано на следующем скриншоте:



6- Опять же, когда сервер работает, и после того, как вы выбрали проект (Клиент), вам нужно щелкнуть правой кнопкой мыши по проекту, и из контекстного меню правой кнопки, пожалуйста, выберите (Отладка), а затем из подменю (Отладка) выберите (Запустить новый экземпляр), как показано на следующем скриншоте:



7- Снова, пока сервер работает, и вы выполнили запуск клиента, как в предыдущем шаге, клиент запрограммирован выполниться один раз, отправить сообщение серверу, и после отправки сообщения завершить выполнение. Но в консоли сервера вы увидите, что сервер уже получил сообщение от клиента, напечатал его в своей консоли и собирается завершить выполнение, как показано на следующем скриншоте. Теперь вы можете закрыть консоль сервера:



2.Самостоятельная задача: Многопоточный сетевой файловый сервер и постоянно работающий клиент на C++

Разработайте усовершенствованное клиент-серверное приложение на C++, где сервер будет обрабатывать запросы от нескольких клиентов одновременно, используя многопоточность. Каждый клиент должен иметь возможность отправлять серверу название файла, который он хочет прочитать. Сервер, получив название файла от клиента, должен искать этот файл в своей файловой системе, читать его содержимое и отправлять обратно клиенту. Клиент, в свою очередь, должен отображать полученное содержимое файла.