МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Курсовая работа

по дисциплине «Сетевые Технологии»

на тему «Разработка приложений с сетевым взаимодействием»

ПГУ.100503.С1128.КР.21ПИ108.01.ПЗ

Специальность - 10.05.03 Информационная безопасность автоматизированных систем

Выполнил студент: Гусев Д. А.

Группа: 21ПИ1

Руководитель: Липилин О. В.

Работа защищена с оценкой: _____

Дата защиты: _____

2024

УТВЕРЖДАЮ
Зав. кафедрой ИБСТ
к.т.н., доцент
С.Л.Зефиров
08 02 2024 г.

ЗАДАНИЕ
на курсовую работу

по теме: Разработка приложений с сетевым взаимодействием.
1 Дисциплина: Сетевые технологии.
2 Вариант задания: 8.
3 Студент: Гусев Д. А.
4 Группа: 21ПИ1.
5 Цель: разработка сетевого приложения с архитектурой клиент-сервер.
6 Исходные данные на проектирование:

— требования к сетевому взаимодействию:

— используемый протокол транспортного уровня – TCP;

— в протоколе прикладного уровня должны быть предусмотрены возможности обмена информацией о возникающих ошибках при сетевом взаимодействии;

— требования к программе-клиенту:

— программа-клиент должна обеспечивать передачу текстовых файлов на сервер если версия программы-сервера равна 1;

— программа-клиент должна обеспечивать передачу бинарных файлов на сервер если версия программы-сервера равна 2;

— в программе-клиенте должна быть реализована возможность определения версии программы-сервера;

— в программе-клиенте должна быть предусмотрена возможность регистрации нового пользователя в программе-сервере ;

— в программе-клиенте должна быть предусмотрена обработка ошибок, возникающих при сетевом взаимодействии;

— требования к программе-серверу:

— программа-сервер должна сохранять текстовый файл, полученный от программы клиента, если версия программы-сервера равна 1;

— программа-сервер должна сохранять бинарный файл, полученный от программы клиента, если версия программы-сервера равна 2;

— программа-сервер должна сообщать программе-клиенту об ошибке, если передаваемый файл не соответствует версии программы-сервера;

— в программе-сервере должна выполняться проверка IP-адреса программы-клиента – сохранение файла должно выполняться, если IP-адрес находится в диапазоне разрешенных адресов;

— в программе-сервере должна быть реализована возможность одновременного подключения к серверу нескольких программ-клиентов;

— в программе-сервере должна быть предусмотрена обработка ошибок, возникающих при сетевом взаимодействии;

— максимальное количество одновременно подключенных программ-клиентов – 3;

— при попытке подключения программы-клиента сверх максимального количества, ей должно возвращаться сообщение об ошибке подключения;

— должна быть реализована аутентификация пользователя программы-клиента при подключении к серверу;

— должен быть разработан протокол прикладного уровня, предназначенный для реализации сетевого взаимодействия между программой-клиентом и программой-сервером;

— должна быть проведена проверка работоспособности программ;

— должно быть разработано руководство пользователя.

7 Структура проекта:

7.1 Пояснительная записка (содержание работы):

— разработка протокола прикладного уровня;

— программная реализация клиента и сервера;

— проверка работоспособности разработанных программ;

— разработка руководства пользователя.

7.2 Графическая часть:

UML-диаграммы протокола прикладного уровня.

7.3 Экспериментальная часть:

Тестирование разработанных программ.

8 Календарный план выполнения работы:

—	оформление задания	к 2 неделе семестра;
—	разработка протокола прикладного уровня	к 5 неделе семестра;
—	программная реализация клиента и сервера	к 11 неделе семестра;
—	проверка работоспособности программ	к 13 неделе семестра;
—	разработка руководства пользователя	к 14 неделе семестра;
—	оформление отчета	к 15 неделе семестра.

Задание получил «3» февраля 2024 г. Д. А. Гусев

Студент

Руководитель О. В. Липилин

Оглавление

Введение

В современном мире информационных технологий, где данные являются одним из самых ценных ресурсов, эффективное и безопасное управление файлами становится критически важным. В этом контексте, разработка клиент-серверной системы, которая позволяет загружать файлы с клиента на сервер, представляет собой значительный интерес.

Целью данной курсовой работы является проектирование и реализация клиент-серверной системы, которая обеспечивает возможность загрузки файлов с клиента на сервер. Эта система будет использовать транспортый протокол для передачи данных и прикладной для обмена информацией о состояниях клиента и сервера.

На первом этапе будет разработан протокол для обмена данными между клиентом и сервером. Протокол будет определять формат сообщений, которые будут передаваться между клиентом и сервером, а также процедуры для обработки этих сообщений.

На втором этапе будет будет выполнена программная реализация клиентской и серверной частей системы. Клиентская часть будет отвечать за выбор файлов для загрузки и инициацию процесса загрузки, в то время как серверная часть будет отвечать за прием файлов, их сохранение.

После реализации клиентской и серверной частей системы будет проведено тестирование для проверки их работоспособности. В ходе тестирования будет проверено, что система корректно обрабатывает все возможные сценарии использования.

На заключительном этапе будет подготовлено руководство пользователя, которое будет содержать подробные инструкции по использованию системы, а также информацию о возможных ошибках и способах их устранения.

1 Разработка протокола прикладного уровня

Были разработаны диаграммы состояний сервера и клиента. Диаграммы представлена на рисунках А1 и А2 в приложении А. Была разработана таблица запросов-ответов между сервером и клиентом. Результат представлен в таблице 1.

Таблица 1 — Запросы — ответы

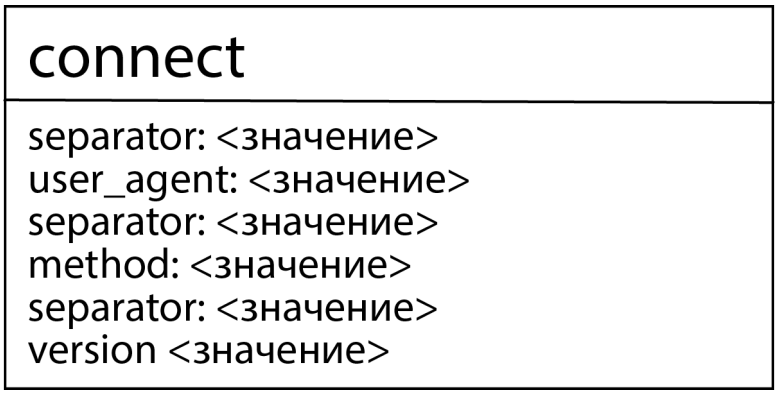
запрос клиент — сервер	ответ сервер — клиент
запрос на подключение	успешное подключение
	ошибка подключения
	неверный запрос
запрос на аутентификацию	успешная аутентификация
	неверный логин или пароль
	неверный запрос
запрос на регистрацию	успешная регистрация
	пользователь уже существует
	неверный запрос
запрос на передачу файла	успешная передача файла
	неверный тип файла
	неверный запрос

Были разработаны диаграммы последовательностей сетевого взаимодействия клиента и сервера. Диаграммы правильного взаимодействия представлены на рисунках А3 и А4 в приложении А. Диаграммы взаимодействия с ошибкой подключения представлены на рисунках А5, А6, А7 в приложении А. Диаграммы взаимодействия с ошибкой аутентификации и регистрации представлены на рисунках А8, А9, А10 и А11 в приложении А. Диаграммы взаимодействия с ошибкой передачи файла представлены на рисунках А12, А13, А14 и А15 в приложении А.

Был разработан формат сообщений.

Название: запрос клиента на подключение к серверу.

Формат:



Значения полей:

connect – заголовок запроса, строка символов в кодировке UTF-8;

separator: <значение> - строка байт, поле значение содержит разделитель сообщений (4 байта);

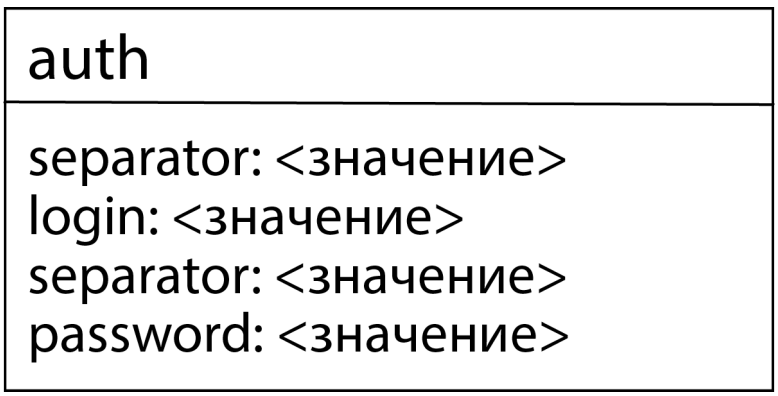
user_agent: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит идентификатор программы-клиента (от 1 до 32 символов английского алфавита), используется для определения сессии каждой программы-клиента;

method: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит информацию о методе входа (от 3 до 4 символов английского алфавита), используется для определения метода входа (регистрация или аутентификация).

version: <значение> - целое число типа int, поле «значение» содержит информацию о версию программы сервера (1 или 2).

Название: запрос клиента на аутентификацию.

Формат:



Значения полей:

auth – заголовок запроса, строка символов в кодировке UTF-8;

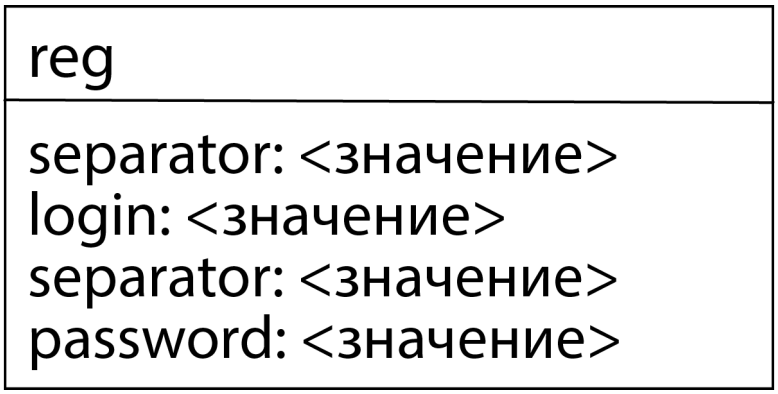
separator: <значение> - строка байт, поле значение содержит разделитель сообщений (4 байта);

login: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит username пользователя (от 4 до 16 символов);

password: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит пароль пользователя (от 6 до 64 символов).

Название: запрос клиента на регистрацию.

Формат:



Значения полей:

reg – заголовок запроса, строка символов в кодировке UTF-8;

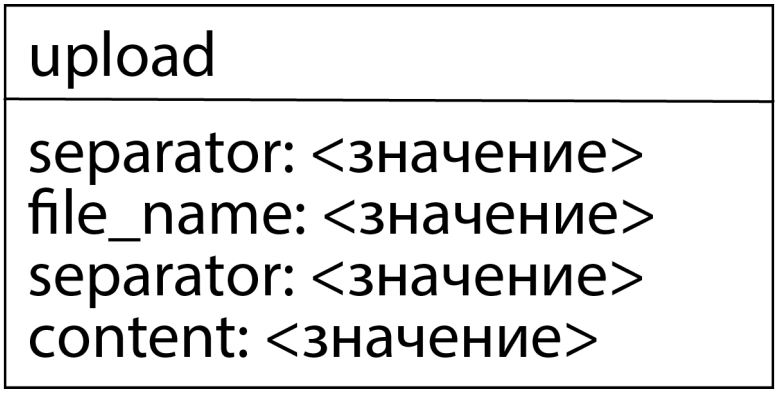
separator: <значение> - строка байт, поле значение содержит разделитель сообщений (4 байта);

login: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит username пользователя (от 4 до 16 символов);

password: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит пароль пользователя (от 6 до 64 символов).

Название: запрос клиента на передачу файла на сервер.

Формат:



Значения полей:

upload – заголовок запроса, строка символов в кодировке UTF-8;

separator: <значение> - строка байт, поле значение содержит разделитель сообщений (4 байта);

fileNmae: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит имя файла (от 1 до 128 символов английского алфавита, кириллицы или цифр от 0 до 9);

content: <значение> - байтовая строка, содержит файл для передачи.

2 Программная реализация

Был разработан код сервера и клиента, а также код классов для удобной работы с socket.socket. Код программ клиента и сервера находится на в репозитории на github.com: *E:\Projects\PGU\6 Семестр\Network-Technologies\Coursework\STEP 3\sources*, а также в приложении Б.

1) Разарботка общих классов и функций. Код представленных классов и функций находится в репозитории на github.com, а также в приложении Б1.

1.1) Был разработан <*class Socket*>, расширяющий возможности <*class socket.socket*>. В класс были добавлены следующие функции:

- <*def receive*>: функция расширяет функционал <socket.socket.recv>.

Параметры функции: <target_len: int> – ожидаемое количесвто сообщений.

Возвращаемое значение: <*Tuple[bytes, ...]*> - Кортеж сообщений, где каждое сообщение строка байт.

Алгоритм работы: функция принимает байты в буфер, до того момента, пока не встретиться строка байт <*self.end*> (Байты, означающие конец сообщения) или пока поток байт не прекратиться. Затем строка разбивается на составные части (разделителем служит атрибут <*self.separator*>). Далее сообщения записываются в кортеж. Если количество сообщений меньше ожидаемого, то «пустые» сообщения записываются в кортеж как <*b`\null`*> - это необходимо для корреткной обработки кортежа полученных сообщений (обрабатывать правильность завпроса), чтобы не приходилось каждый раз проверять длину кортежа сообщений.

- <*def send*>: расширяет функционал <*socket.socket.sendall*>.

Параметры функции: <**msgs: bytes*> – сообщения, которые необходимо передать.

Возвращаемое значение: None – нет возвращаемого значения.

Алгоритм работы: функция «склеивает» все сообщения в один буфер, добавляя разделитель <*self.separator*> между сообщениями, затем отправляет их.

- <*def accept*>: расширяет функционал <socket.socket.accept>.

Параметры функции: не принимает параметров.

Возвращаемое значение: <*Tuple[Socket, Tuple]*> - кортеж, состоящий из клиентского сокета и адреса клиента.

Алгоритм работы: выполняет дейссвия, аналогичные родительской функции, за исключением инициализации <*class Socket*> вместо <*calss socket.socket*> - необходимо для корректного использования <class Socket> при принятии соединения от клиентов.

- В класс были добавлены атрибуты <*self._end: bytes*> и <*self._separator: bytes*> и <*self._chunk_size*>, а также методы их получения(*@property*) и установки(*@***.setter*) — необходимо для корректной работы класса, чтобы нельзя было установить недопустимые значения для атрибутов из вне. Атрибуты отвечают за символы предназначеные для обозначения конца сообщения, разделителя и размера чанка для передачи соответсвенно.

1.2) Был разарботан вспомогательный <*class Hs*>, содержащий аттрибуты типа <*bytes*> - заголовки <*CONN*>, <*AUTH*>, <*UPLOAD*> и <*REG*>. Необходим для удобной проверки правильности запросов.

1.3) Был разарботан вспомогательный <class Ms>, содержащий аттрибуты типа <*bytes*> - сообщения клиент-сервер. Необходим для удобной проверки правильности запросов.

Вышеперечиселнные классы были добавлены в отдельный модуль <*network.py*>, так как они необходимы для работы и сервера, и клиента.

2) Разработка классов и функций для сервера. Код представленных функций находится в репозитории на github.com, а также в приложении Б2.

2.1) Был создан <*class Data*>: расширяет функционал <*class dict*>.

В класс были добавлены следующие функции:

- <*def __init__*>: конструктор класса.

Параметры функции: <*path: str*> - путь к файлу с базой данных сервера (список *«whitelist»*, а также словарь с пользователями *«users»*);

Возвращаемое значение: <*None*> – нет возвращаемого значения;

Алгоритм работы: открывает файл с сериализованными <*class pickle*> данными и передает полученный словарь в <*def super().__init__*>, если файла не существует, создает новый и записывает в него значения базы данных по умолчанию;

- <*def commit*>: функция для записи словаря в файл.

Параметры функции: нет параметров;

Возвращаемое значение: <*None*> – нет возвращаемого значения;

Алгоритм работы: сюреализует данные с помощью <*class pickle*> и записывает их в файл по пути <*self._path*>;

- В класс были добавлен атрибуты <*self._path: str*>— необходим для корректной работы метода commit, содержит путь к базе данных.

2.2) Была разработана <*def file_type*> - функция определения типа файла (текстовый или бинарный).

Параметры функции: <*content: bytes*> - содержимое файла;

Возвращаемое значение: <*bytes*>, возвращает b`1`, если файл текстовый, b`2`, если файл бинарный;

Алгоритм работы: функция декодирует байтовую строку в UTF-8, если возникает ошибка декодирования, функция считает, что файл бинарный и возвращает соответсвующее значение;

2.3) Был разработан <*class Handler*>: расширяе твозможности <*class threading.Thread*> и обрабатывает подключение клиента. В класс были добавлены следующие функции:

- <*def __init__*>: конструктор класса.

Параметры функции: <*socket: Socket*> - клиентский сокет, <*address: Tuple*> - клиентский адрсее, <*data: Data*> - объект бызы данных сервера, <*queue: List['Handler']*> список с текущими обработчиками.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: устанавливает атрибуты <*self._socket*> и <*self._data*>, проверяет находится ли ip клиента в разрешенных, а также есть ли свободные слоты для обработки клиента на сервере. Если условие не выполняется, устанавливает флаг <*self._exitFlag*> в значение <*True*>, а также устанавливает атрибут текущего состояния обработчика <*self._state*> в <*self.connect*> (метод обработки подключения)

- <*def connect*>: обработчик подключения.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: проверяет значение атрибута <*self._exitFlag*>, если флаг находится в состоянии <*True*>, обрабатывает запрос на подключение от клиента и отправляет ему сообщение об ошибке подключения, устанавливает атрибут <self._state> в значение <*None*>. Если флаг установлен в значение <*False*>, обрабатывает подключение, проверяя правильность запроса. По результату проверки отправляет соответсвующее сообщение клиенту. Если запрос правильный, устанавливает значение атрибута <*self._state*> в значение <*self.auth*> или <*self.reg*> (в зависимости от переданного метода), если запрос неправильный, то значение атрибута <*self._state*> остается неизменным.

- <*def auth*>: обработчик аутентификации.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: проверяет правильность завпроса и правильность данных для аутентификации, если параметры не верны, отправляет клиенту соответсвующее сообщение. Если данные верны, отправляет клиенту сообщение об успешной аутентификации и устанавливает значение атрибута <*self._state*> в <*self.upload*> (метод передачи файла).

- <def reg>: обработчик аутентификации.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: проверяет правильность запроса и правильность данных для регистрации, если параметры не верны, отправляет клиенту соответсвующее сообщение. Если данные верны, отправляет клиенту сообщение об успешной регистарции и устанавливает значение атрибута <*self._state*> в <*self.upload*> (метод передачи файла).

- <*def upload*> обработчик получения файла от клиента.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: проверяет правильность запроса и правильность данных файла. Если переданный файл неверного типа или запрос неверный, отправляет клиенту соответвующее сообщение. Если параметры верны, записывает файл в директорию сервера и устанавливает атрибут <*self._state*> в значение <*None*>.

- <*def run*>: функция родительского класса <*Thread*>, которая запускается при вызове метода <*start*>. Является обработчиком состояний сервера.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: Запускает бесконечный цикл, выходом из которого является условие, что атрибут <*self._state*> установлен в значение <*None*>. Если <*self._state*> имеет иное значение, то функция вызывает метод, записанный в атрибут <*self._state*>.

3) Разработка классов и функций для клиента. Код представленных классов и функций находится в репозитории на github.com, а также в приложении Б3.

3.1) Был разработан <*class Handler*>: расширяет твозможности <*class threading.Thread*> и обрабатывает подключение к серверу. В класс были добавлены следующие функции:

- <*def __init__*>: конструктор класса.

Параметры функции: <*socket: Socket*> - сокет.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: устанавливает атрибуты <*self._socket*> и <*self._state*>.

- <*def connect*>: обработчик подключения.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: отправляет серверу запрос на подключение, обрабатывает подключение. Проверяет правильность ответа сервера на запрос клиента. Если сервер ответил на запрос сообщением об успехе, устанавливает значение атрибута <*self._state*> в значение <*self.auth*> или <*self.reg*> (в зависимости от введенного клиентом метода). Если ответ на запрос неудачный, то значение атрибута <*self._state*> остается неизменным. Если сервер возвращает сообщение об ошибке, то значение атрибута <*self._state*> устанавливается на <*None*>.

- <*def auth*>: обработчик аутентификации.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: отправляет серверу запрос на аутентификацию, проверяет успешность ответа на завпрос. Если сервер вернул сообщение о неверном запросе или неверных данных для аутентификации, оставляет атрибут <self._state> неизменным. Если запрос успешен, устанавливает значение атрибута <*self._state*> в <*self.upload*> (метод передачи файла).

- <*def reg*>: обработчик аутентификации.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: отправляет серверу запрос на регистрацию, проверяет успешность ответа на завпросю Если сервер вернул сообщение о неверном запросе или неверных данных для регистрации, оставляет атрибут <self._state> неизменным. Если запрос успешен, устанавливает значение атрибута <*self._state*> в <*self.upload*> (метод передачи файла).

- <*def upload*> обработчик получения файла от клиента.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: отправляет серверу запрос на передачу файла, проверяет успешность ответа на завпрос, если сервер вернул сообщение о неверном запросе или неверном типе файла, ставляет атрибут <self._state> неизменным. Если запрос успешен, устанавливает атрибут <*self._state*> в значение <*None*>.

- <*def run*>: функция родительского класса <*Thread*>, которая запускается при вызове метода <*start*>. Является обработчиком состояний клиента.

Параметры функции: нет параметров.

Возвращаемое значение: <*None*> - нет возвращаемого значения.

Алгоритм работы: Запускает бесконечный цикл, выходом из которого является условие, что атрибут <*self._state*> установлен в значение <*None*>. Если <*self._state*> имеет иное значение, то функция вызывает метод, записанный в атрибут <*self._state*>.

3 Проверка работоспособности

Было составлено описание функций, выполняемых клиентом.

Подключение (connect):

- Описание: Клиент пытается подключиться к серверу. Он выбирает метод (аутентификация или регистрация) и версию протокола. Если сервер подтверждает успешное подключение, клиент переходит к следующему этапу (аутентификация или регистрация).

- Вводимые данные: метод (auth или reg) и версия (1 или 2).

- Сообщение протокола: CONN_SUC для успешного подключения, CONN_ERR для ошибки подключения.

Аутентификация (auth):

- Описание: Клиент отправляет запрос на аутентификацию, вводя логин и пароль. Если сервер подтверждает успешную аутентификацию, клиент переходит к следующему этапу (загрузка файла).

- Вводимые данные: логин и пароль.

- Сообщение протокола: AUTH_SUC для успешной аутентификации, AUTH_ERR для неверного логина или пароля.

Регистрация (reg):

- Описание: Клиент отправляет запрос на регистрацию, вводя логин и пароль. Если сервер подтверждает успешную регистрацию, клиент переходит к следующему этапу (загрузка файла).

- Вводимые данные: логин и пароль.

Сообщение протокола: REG_SUC для успешной регистрации, REG_ERR если пользователь уже существует.

Загрузка файла (upload):

- Описание: Клиент загружает файл на сервер. Он вводит путь к файлу, который затем отправляется на сервер. Если сервер подтверждает успешную передачу файла, процесс завершается.

- Вводимые данные: путь к файлу.

- Сообщение протокола: UP_SUC для успешной передачи файла, UP_ERR для неверного типа файла.

Было составлено описание функций, выполняемых сервером.

Подключение (connect):

- Описание: Сервер обрабатывает запрос на подключение от клиента. Он проверяет заголовок, метод и версию протокола. Если все проверки проходят успешно, сервер подтверждает подключение и переходит к следующему состоянию (аутентификация или регистрация).

- Вводимые данные: заголовок, метод и версия протокола.

- Сообщение протокола: CONN_SUC для успешного подключения, CONN_ERR для ошибки подключения, REQ_ERR для неверного запроса.

Аутентификация (auth):

- Описание: Сервер обрабатывает запрос на аутентификацию от клиента. Он проверяет заголовок и валидность логина и пароля. Если все проверки проходят успешно, сервер подтверждает аутентификацию и переходит к следующему состоянию (загрузка файла).

- Вводимые данные: заголовок, логин и пароль.

- Сообщение протокола: AUTH_SUC для успешной аутентификации, AUTH_ERR для неверного логина или пароля, REQ_ERR для неверного запроса.

Регистрация (reg):

- Описание: Сервер обрабатывает запрос на регистрацию от клиента. Он проверяет заголовок и валидность логина и пароля. Если все проверки проходят успешно, сервер подтверждает регистрацию и переходит к следующему состоянию (загрузка файла).

- Вводимые данные: заголовок, логин и пароль.

- Сообщение протокола: REG_SUC для успешной регистрации, REG_ERR если пользователь уже существует, REQ_ERR для неверного запроса.

Загрузка файла (upload):

- Описание: Сервер обрабатывает запрос на загрузку файла от клиента. Он проверяет заголовок, имя файла и содержимое файла. Если все проверки проходят успешно, сервер сохраняет файл и подтверждает успешную загрузку.

- Вводимые данные: заголовок, имя файла и содержимое файла.

- Сообщение протокола: UP_SUC для успешной загрузки файла, UP_ERR для неверного типа файла, REQ_ERR для неверного запроса.

Было описано взаимодействие клиента и сервера.

Описание взаимодействия клиента и сервера при успешной регистрации:

- Подключение (connect): Клиент (Python3 Client Win64) подключается к серверу. Сервер подтверждает успешное подключение.

- Регистрация (reg): Клиент отправляет запрос на регистрацию с именем пользователя (user1) и паролем (pass1). Сервер подтверждает успешную регистрацию.

- Загрузка файла (upload): Клиент загружает файл (file.jpg) на сервер. Это бинарные данные, которые представляют собой изображение в формате JPEG. Сервер подтверждает успешную передачу файла.

TCP поток представлен ниже:

```
connect@sepPython3 Client Win64@sepreg@sep2@end
connect@seруспешное подключение@end
reg@sepuser1@seppass1@end
reg@seруспешная регистрация@end
upload@sepfile.jpg@sep���  #JFIF ### # #  ﷺ C #####
#########2!####=,.$2I@LKG@FEPZsbPUmVEFd emw{ N` }s~ |
C#######;!!;|SFS||||||
```

```
upload@sepуспешная передача файла@end
```

Описание взаимодействия клиента и сервера при успешной аутентификации:

- Подключение (connect): Клиент (Python3 Client Win64) подключается к серверу. Сервер подтверждает успешное подключение.

- Аутентификация (auth): Клиент отправляет запрос на аутентификацию с именем пользователя (user) и паролем (password). Сервер подтверждает успешную аутентификацию.

- Загрузка файла (upload): Клиент загружает текстовый файл (file.txt) на сервер. Текст файла: “Eat some more of these soft French rolls, and drink some tea”. Сервер подтверждает успешную передачу файла.

TCP поток представлен ниже:

```
connect@sepPython3 Client Win64@sepauth@sep1@end
connect@sepуспешное подключение@end
auth@sepuser@seppassword@end
auth@sepуспешная аутентификация@end
upload@sepfile.txt@sepEat some more of these soft French rolls, and drink some tea@end
upload@sepуспешная передача файла@end
```

Описание взаимодействия клиента и сервера при Ошибка подключения (сервер переполнен):

- Подключение (connect): Клиент (Python3 Client Win64) пытается подключиться к серверу. Однако сервер сообщает об ошибке подключения.

TCP поток представлен ниже:

```
connect@sepPython3 Client Win64@sepauth@sep1@end
connect@sepошибка подключения@end
```

Описание взаимодействия клиента и сервера при неверных запросах при подключении, ошибке при регистрации и ошибке неверного типа файла:

- Подключение (connect): Клиент (Python3 Client Win64) пытается подключиться к серверу с запросом на помощь, но сервер сообщает о неверном

запросе. Затем клиент повторно пытается подключиться к серверу с запросом на аутентификацию, и сервер подтверждает успешное подключение.

- Аутентификация (auth): Клиент отправляет запрос на аутентификацию с именем пользователя (user2) и паролем (pass2), но сервер сообщает о неверном логине или пароле. Затем клиент повторно отправляет запрос на аутентификацию с другим именем пользователя (user) и паролем (password), и сервер подтверждает успешную аутентификацию.

- Загрузка файла (upload): Клиент пытается загрузить файл (file.jpg) на сервер, но сервер сообщает о неверном типе файла. Затем клиент загружает текстовый файл (file.txt) на сервер с текстом “Eat some more of these soft French rolls, and drink some tea”, и сервер подтверждает успешную передачу файла.

TCP поток представлен ниже:

```
connect@sepPython3 Client Win64@sephelp@sep3@end
connect@serневерный запрос@end
connect@sepPython3 Client Win64@sepauth@sep1@end
connect@seруспешное подключение@end
auth@sepuser2@seppass2@end
auth@serневерный логин или пароль@end
auth@sepuser@seppassword@end
auth@seруспешная аутентификация@end
upload@sepfile.jpg@sep���  #JFIF ### # #  ◆◆ C #####
#########2!####=,.$2I@LKG@FEPZsbPUmVEFd◆emw{◆◆◆N`◆◆◆}◆s~◆|◆◆
C#######;!!;|SFS||||||||||||||||||||||||||||||||||||||||||||◆◆ ##
# ###" ######◆◆ #  ####           #######◆◆ $# ###### #      ## ####!
A#1#2B"Qq◆◆ ######          ### ◆◆ #### ###          #####12AQ◆◆
## #### ? ◆##◆◆◆   ◆◆◆dŪ◆◆◆"#I◆m◆#[##◆C]◆◆◆◆◆z2>◆i◆
_◆u.;or◆Y◆T+Z  gO◆#s◆◆◆◆kl◆*3◆`V◆◆m◆Y◆◆r}◆◆e
%Z◆@B◆#x◆<[◆ώ◆◆◆◆Y◆#◆83cfG#WI';◆◆d♪#C◆#Sj◆<V◆#n#0◆◆#◆6
(#◆8#N◆◆w.q3◆◆◆◆S◆L◆◆e◆Y◆◆#◆'m@"◆ ◆#◆#e◆◆P◆◆g◆#P□& ?
#◆B#◆◆◆◆K@◆d{ʔ-J+◆◆#◆◆#◆◆+◆◆ک◆#◆◆<##BL#◆◆@end
upload@serневерный тип файла@end
upload@sepfile.txt@sepEat some more of these soft French rolls, and
drink some tea@end
upload@seруспешная передача файла@end
```

Описание взаимодействия клиента и сервера при неверных запросах при подключении, ошибке при аутентификации и ошибке неверного типа файла:

- Подключение (connect): Клиент (Python3 Client Win64) пытается

подключиться к серверу с запросом на помощь, но сервер сообщает о неверном запросе. Затем клиент повторно пытается подключиться к серверу с запросом на аутентификацию, и сервер подтверждает успешное подключение.

- Аутентификация (auth): Клиент отправляет запрос на аутентификацию с именем пользователя (user2) и паролем (pass2), но сервер сообщает о неверном логине или пароле. Затем клиент повторно отправляет запрос на аутентификацию с другим именем пользователя (user) и паролем (password), и сервер подтверждает успешную аутентификацию.

- Загрузка файла (upload): Клиент пытается загрузить файл (file.jpg) на сервер, но сервер сообщает о неверном типе файла. Затем клиент загружает текстовый файл (file.txt) на сервер с текстом “Eat some more of these soft French rolls, and drink some tea”, и сервер подтверждает успешную передачу файла.

TCP поток представлен ниже:

```
connect@sepPython3 Client Win64@sephelp@sep3@end
connect@serневерный запрос@end
connect@sepPython3 Client Win64@sepauth@sep1@end
connect@serуспешное подключение@end
auth@sepuser2@seppass2@end
auth@serневерный логин или пароль@end
auth@sepuser@seppassword@end
auth@serуспешная аутентификация@end
upload@sepfile.jpg@sep���  #JFIF ### # #  ��  C #####
#########2!####=,.$2I@LKG@FEPZsbPUmVEFd�emw{���N`���}�s~�|��
C#######;!!;|SFS|||||||
```

4 Разработка руководства пользователя

Руководство пользователя программы-клиента.

Описание программы: программа-клиент предназначена для управления соединением и обработки запросов от клиента. Она позволяет пользователю аутентифицироваться или зарегистрироваться на сервере, а также загружать файлы на сервер.

Функционал:

- Установление соединения с сервером – пользователь выбирает метод (аутентификация или регистрация) и версию протокола.

- Аутентификация пользователя – пользователь вводит логин и пароль, которые затем отправляются на сервер для аутентификации.

- Регистрация нового пользователя – пользователь вводит логин и пароль, которые затем отправляются на сервер для регистрации.

- Загрузка файла на сервер – пользователь вводит путь к файлу, который затем отправляется на сервер.

Ограничения: путь к файлу должен быть корректным и файл должен существовать.

Вводимые команды (действия) для выполнения функций программы:

- Установление соединения с сервером – введите метод (<auth> или <reg>) и версию (<1> или <2>).

- Аутентификация пользователя – введите логин и пароль.

- Регистрация нового пользователя – введите логин и пароль.

- Загрузка файла на сервер – введите путь к файлу.

Возможные ошибки и причины их возникновения:

- Ошибка соединения – может возникнуть, если сервер недоступен или если время ожидания соединения истекло.

- Ошибка аутентификации – может возникнуть, если введены неверные учетные данные.

- Ошибка регистрации – может возникнуть, если выбранный логин уже занят.

- Ошибка загрузки файла – может возникнуть, если файл не существует, путь к файлу некорректен или у пользователя нет прав на чтение файла.

- Ошибка сервера – может возникнуть при любых проблемах на стороне сервера.

Руководство пользователя программы-сервера.

Описание программы: программа создает сервер, который обрабатывает клиентские подключения. Она использует класс Handler, который является потоком, для обработки каждого клиентского подключения. Класс Handler имеет различные состояния, такие как connect, auth, reg и upload, которые обрабатывают различные этапы подключения.

Функционал:

- Подключение – Проверяет, является ли клиент допустимым, проверяя его IP-адрес и количество активных подключений.

- Аутентификация – Проверяет, является ли пользователь действительным, проверяя его учетные данные.

- Регистрация – Регистрирует нового пользователя, если его учетные данные уникальны и действительны.

- Загрузка – Обрабатывает загрузку файла от клиента.

- Вводимые команды – Клиенты могут отправлять следующие команды: Headers.CONN (для начала процесса подключения); Headers.AUTH (для аутентификации существующего пользователя); Headers.REG (для регистрации нового пользователя); Headers.UP (для загрузки файла на сервер).

Возможные ошибки и причины их возникновения:

- TimeoutError, ConnectionAbortedError – Эти ошибки могут возникнуть, если произошла проблема с подключением во время обработки запроса.

- Ошибки аутентификации и регистрации – Если учетные данные пользователя недействительны или уже существуют, сервер отправит сообщение об ошибке.

- Ошибки загрузки – Если файл, который пытается загрузить пользователь, имеет недопустимый формат или имя, сервер отправит сообщение об ошибке.

Заключение

В ходе выполнения данной курсовой работы была успешно реализована клиент-серверная система для загрузки файлов. Работа включала в себя несколько ключевых этапов, каждый из которых способствовал достижению общей цели.

На первом этапе был разработан протокол для обмена данными между клиентом и сервером. Это обеспечило основу для взаимодействия между двумя сторонами и позволило обеспечить безопасность и целостность передаваемых файлов.

Затем была выполнена программная реализация клиентской и серверной частей системы. Это включало разработку кода сервера и клиента, а также код классов для удобной работы с socket.socket. Код программ клиента и сервера был размещен в репозитории на GitHub.

После этого было проведено тестирование работоспособности системы. В ходе тестирования было проверено, что система корректно обрабатывает все возможные сценарии использования и обеспечивает безопасность и целостность передаваемых файлов.

Наконец, было подготовлено руководство пользователя, которое содержит подробные инструкции по использованию системы, а также информацию о возможных ошибках и способах их устранения.

Таким образом, в результате выполнения данной работы была создана полноценная система для загрузки файлов с клиента на сервер, которая обеспечивает целостность передаваемых данных.

Список используемых источников

1 Python Software Foundation. Библиотека Python: модуль socket [Электронный ресурс] // Python Software Foundation. - Электрон. дан. - Режим доступа: https://docs.python.org/3/library/socket.html. - Дата доступа: 6.04.2024.

2 Пензенский государственный университет. Методические указания к курсовой работе “Разработка протокола” [Электронный ресурс] // Пензенский государственный университет. - Электрон. дан. - Режим доступа: https://moodle.pnzgu.ru/pluginfile.php/2566362/mod_resource/content/2/МУ%20КР%20Разработка%20протокола.pdf. - Дата доступа: 20.03.2024.

3 Пензенский государственный университет. Методические указания к курсовой работе “Тестирование разработанных программ” [Электронный ресурс] // Пензенский государственный университет. - Электрон. дан. - Режим доступа: https://moodle.pnzgu.ru/pluginfile.php/2566365/mod_resource/content/2/МУ%20КР%20Тестирование%20разработанных%20программ%20Задание.pdf. - Дата доступа: 05.05.2024.

4 Пензенский государственный университет. Лабораторная работа №4 [Электронный ресурс] // Пензенский государственный университет. - Электрон. дан. - Режим доступа: https://moodle.pnzgu.ru/pluginfile.php/2587836/mod_resource/content/0/Лабораторная%20работа%204.pdf. - Дата доступа: 05.05.2024.

Приложение А

(обязательное)

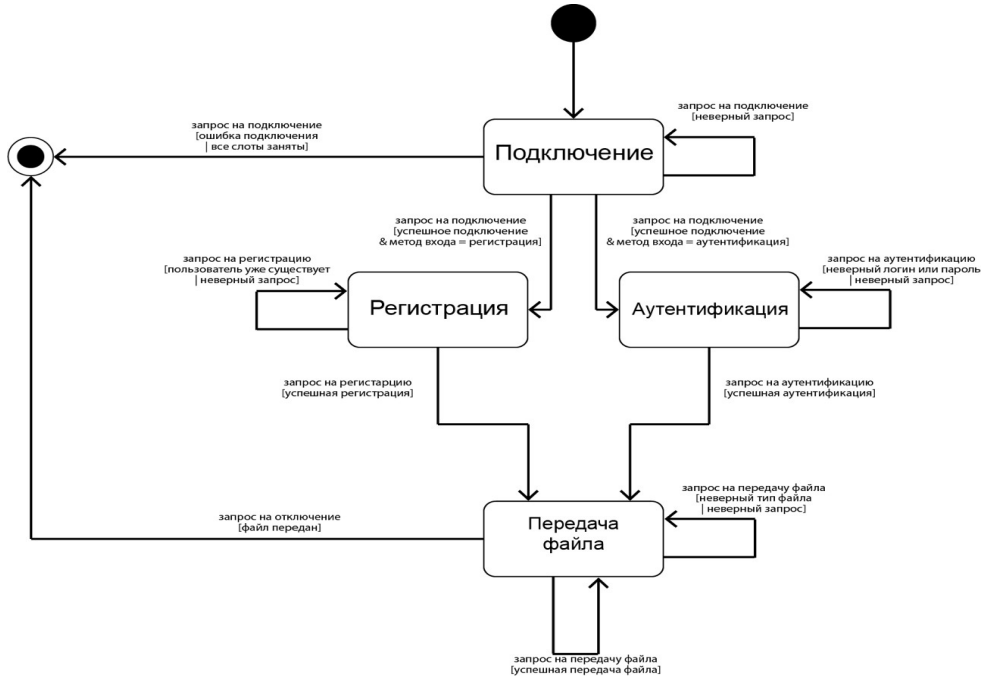


Рисунок А1 — Диаграмма состояний клиента

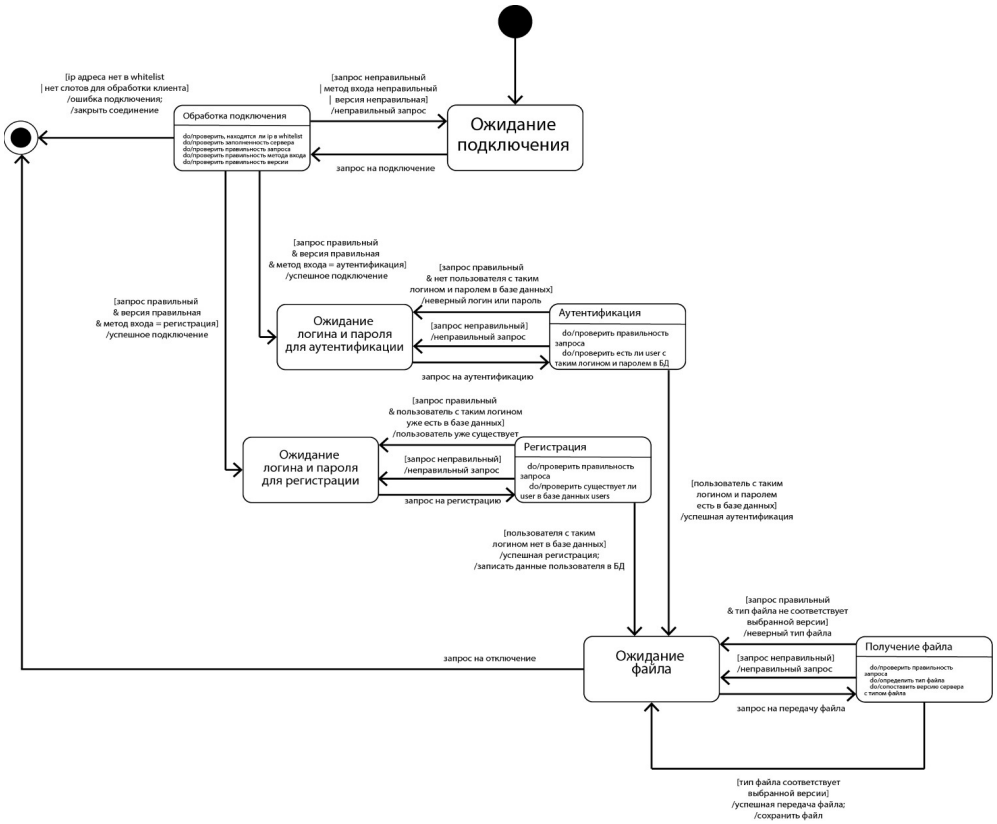


Рисунок А2 — Диаграмма состояний сервера

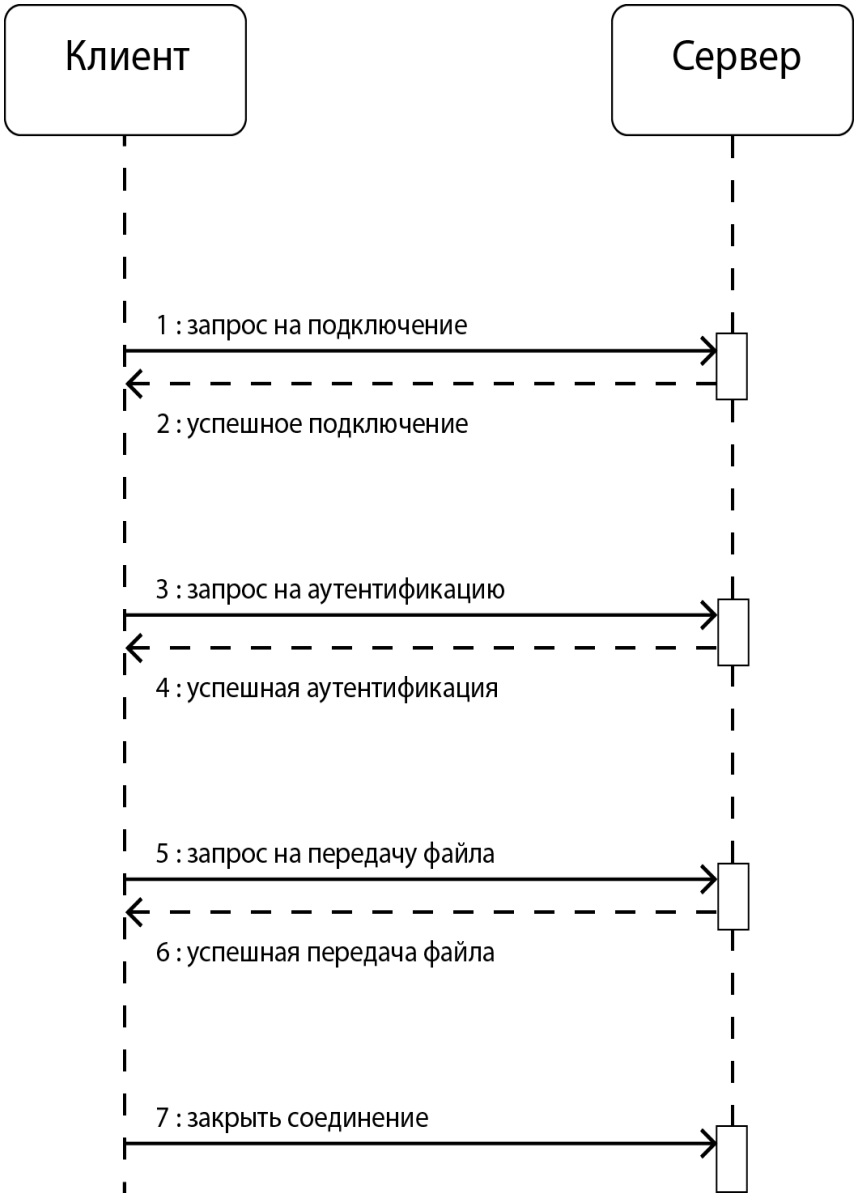


Рисунок А3 — Правильное взаимодействие (аутентификация)

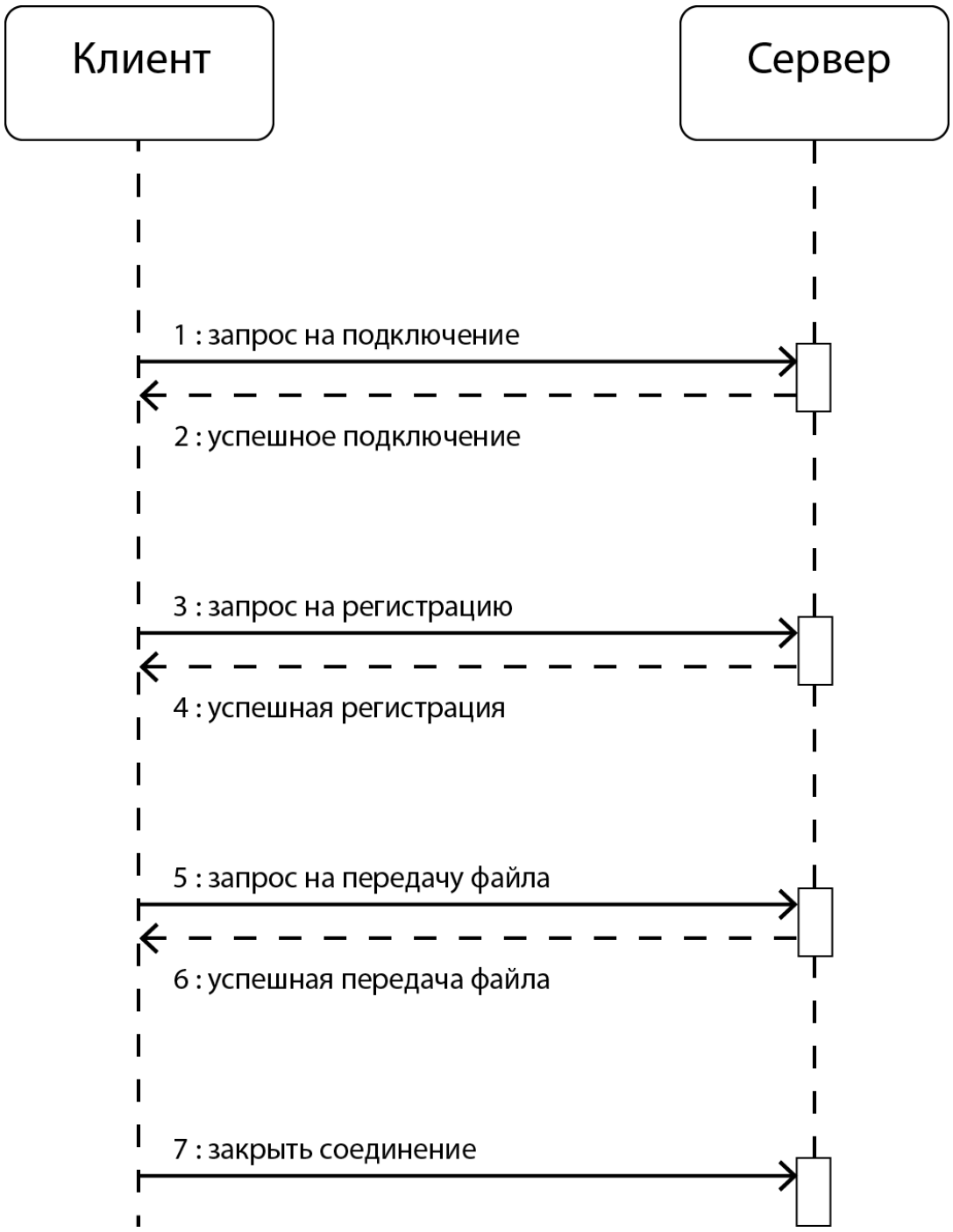


Рисунок А4 — Правильное взаимодействие (регистрация)

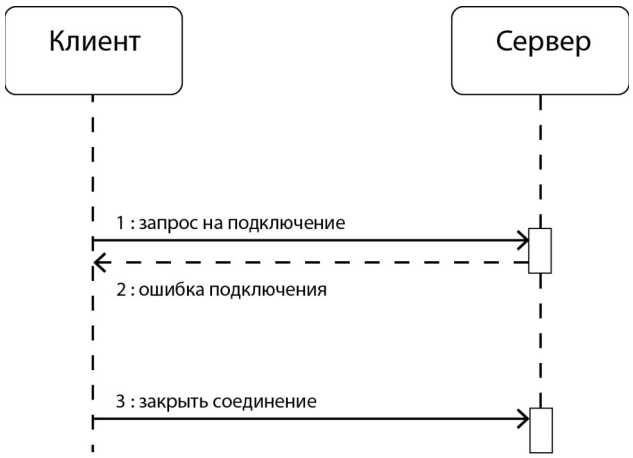


Рисунок А5 — Ошибка подключения

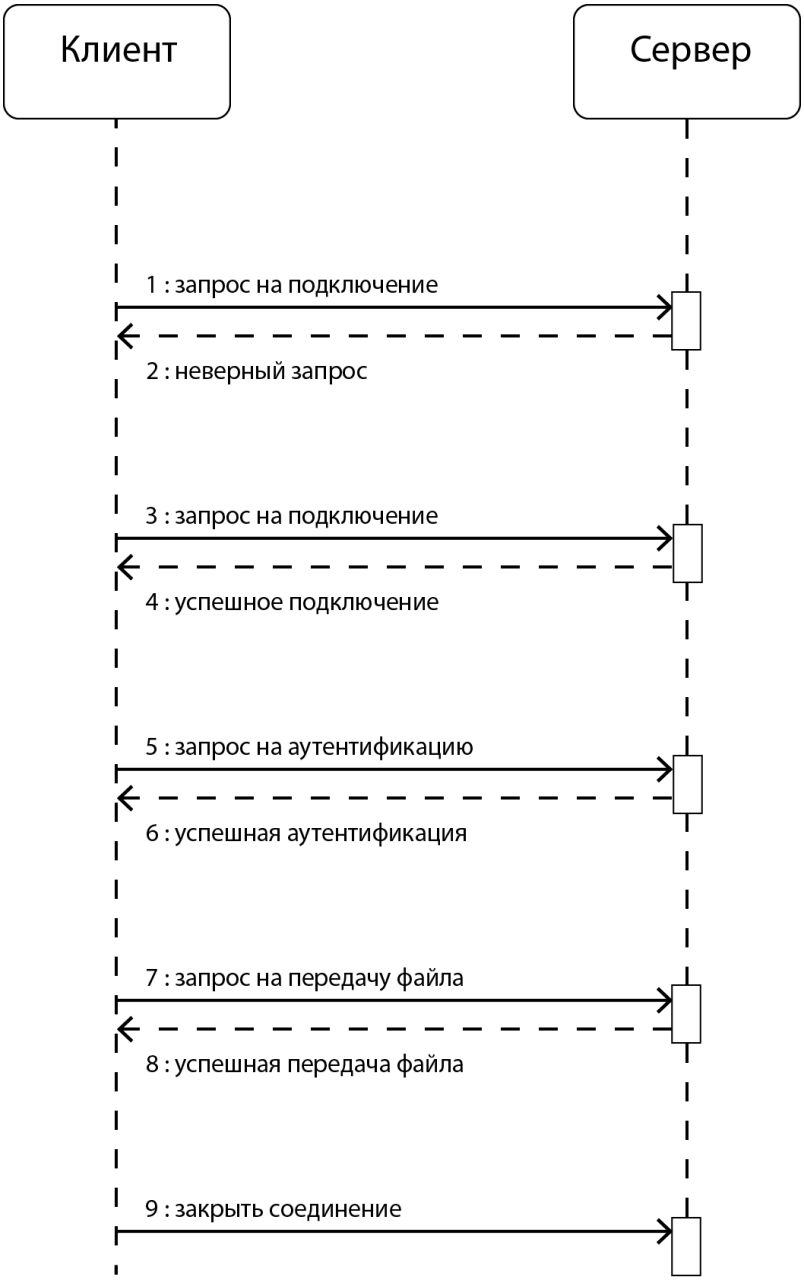


Рисунок А6 — Неверный запрос при подключении (аутентификация)

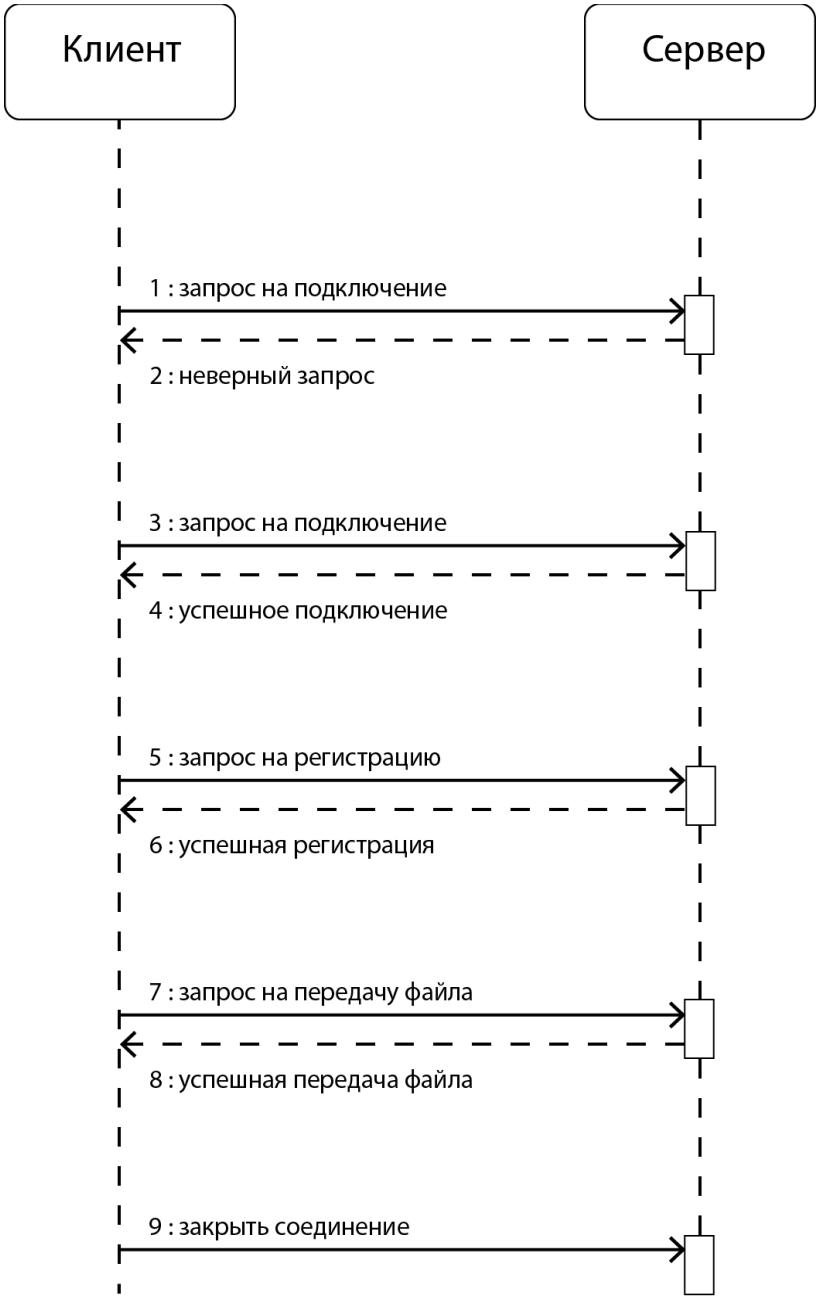


Рисунок А7 — Неверный запрос при подключении (регистрация)

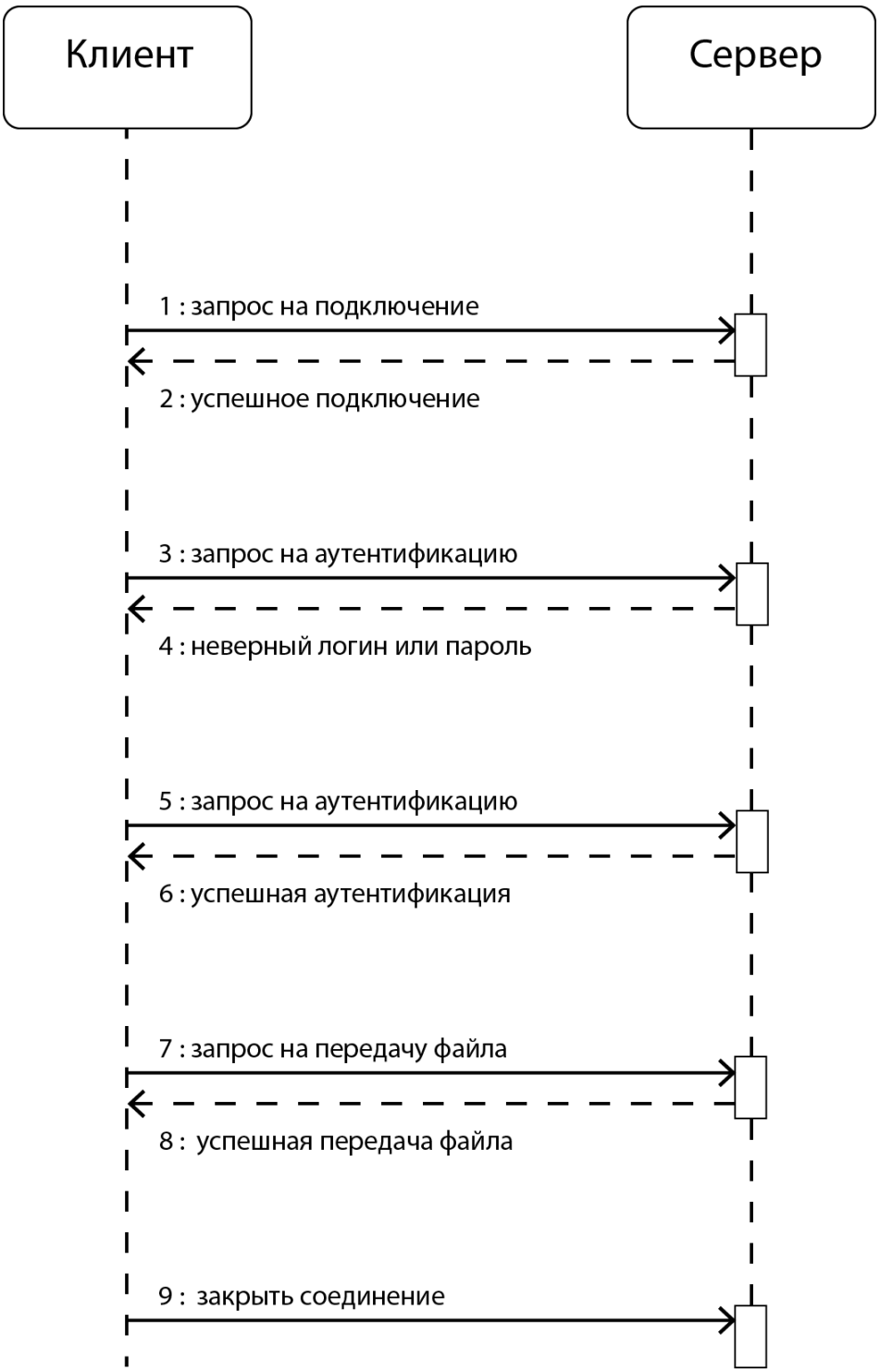


Рисунок А8 — Неверный логин или пароль (аутентификация)

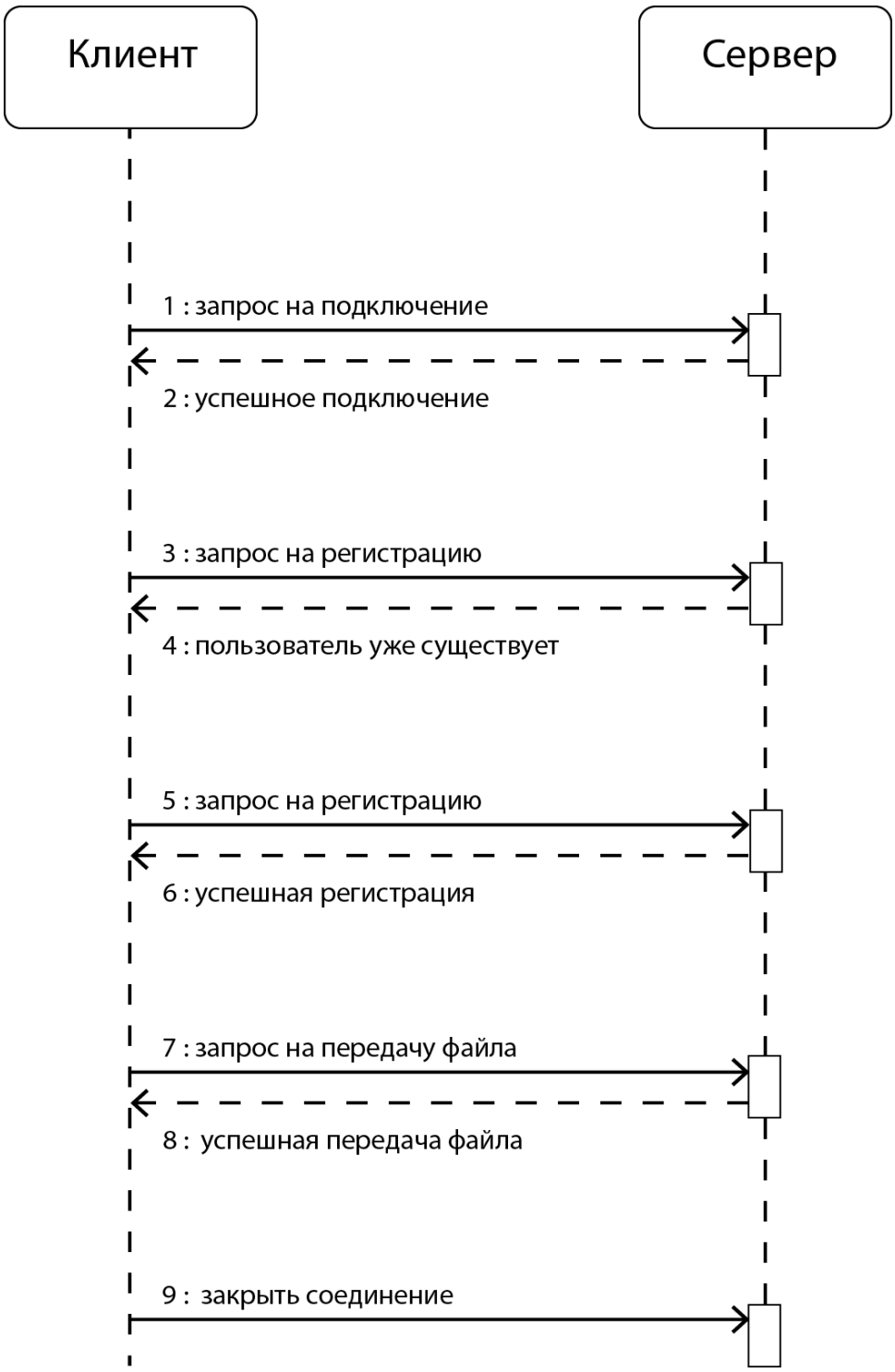


Рисунок А9 — Пользователь уже существует (регистрация)

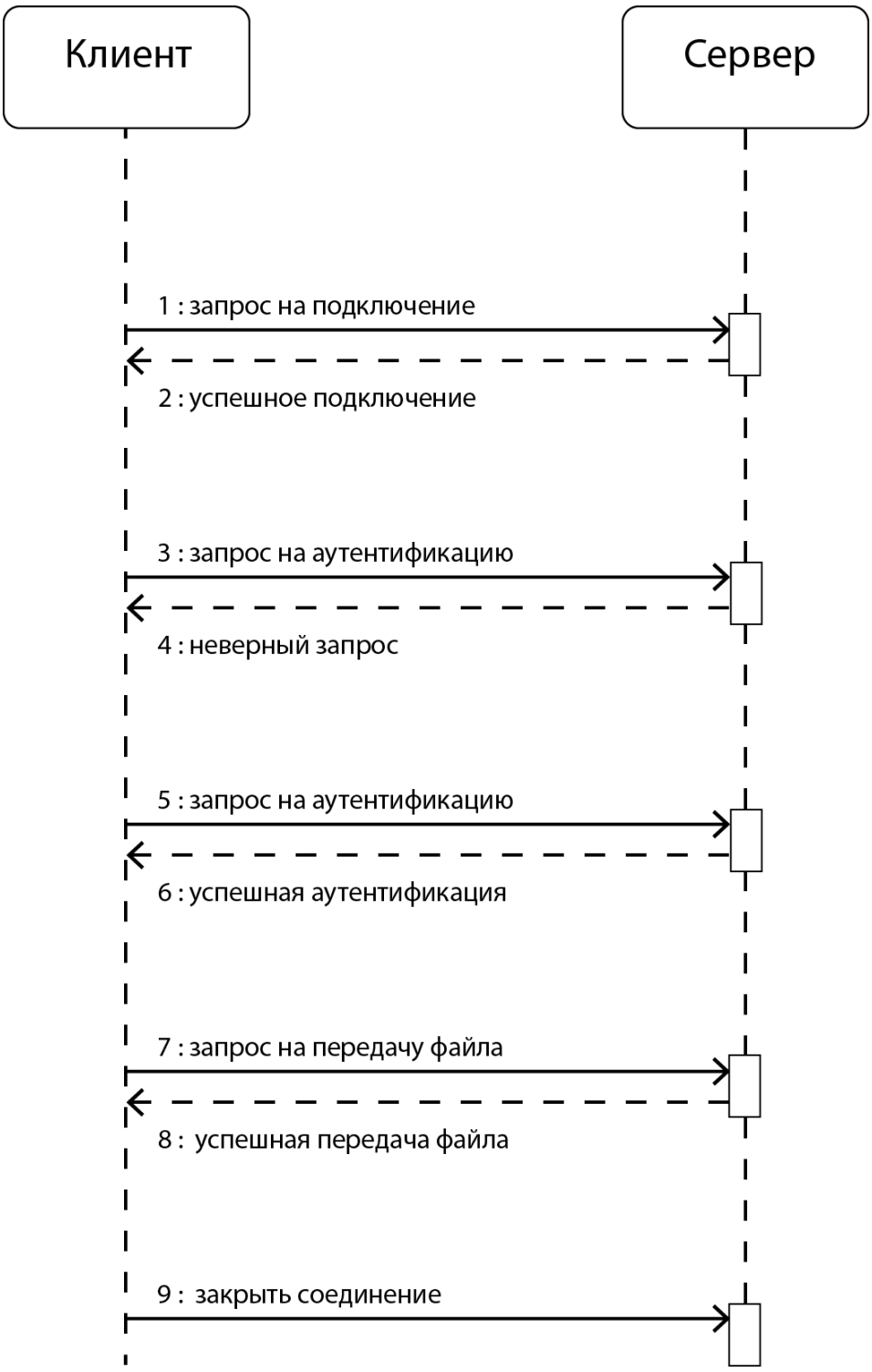


Рисунок А10 — Неверный запрос при аутентификации

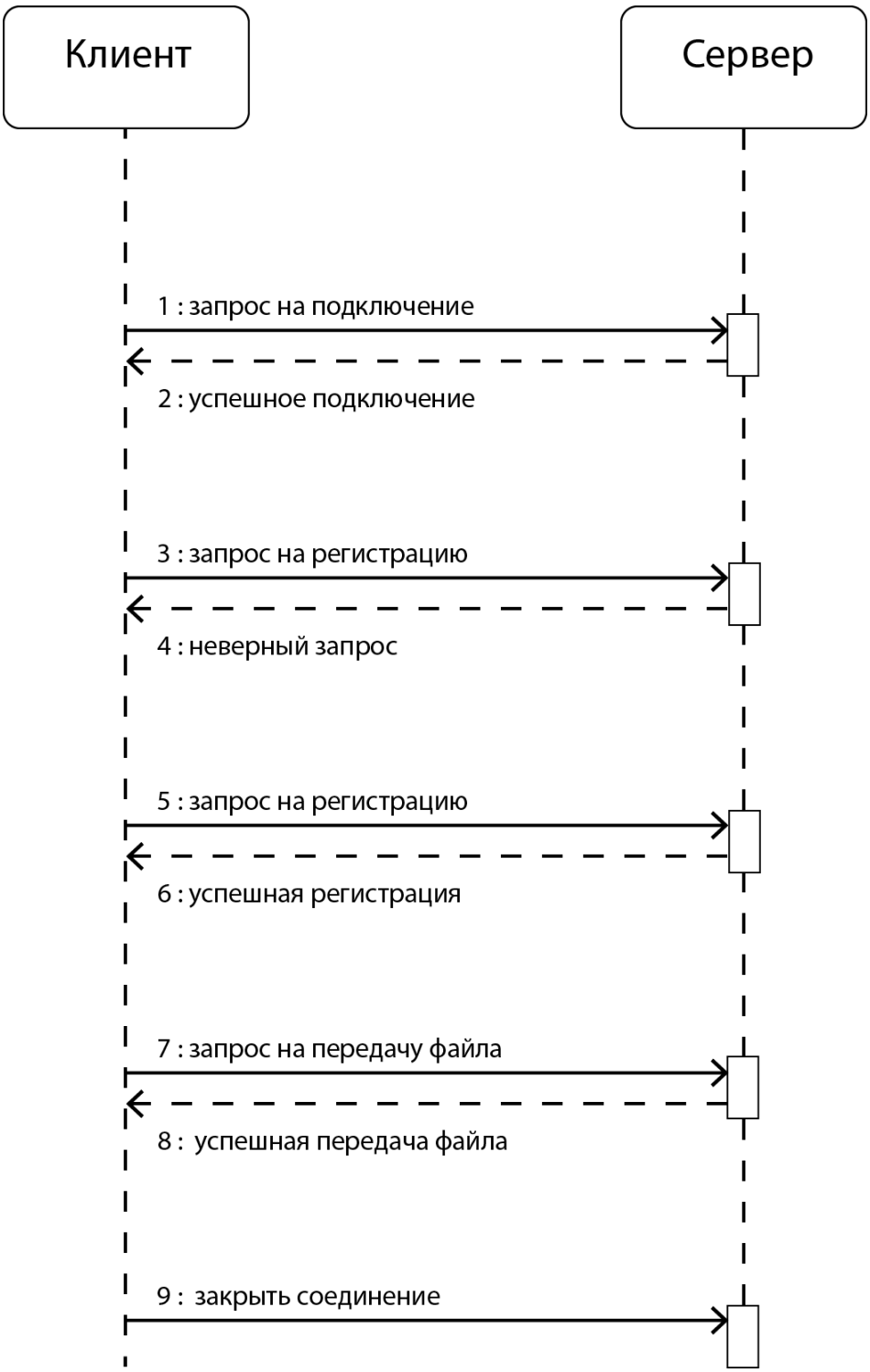


Рисунок А11 — Неверный запрос при регистрации

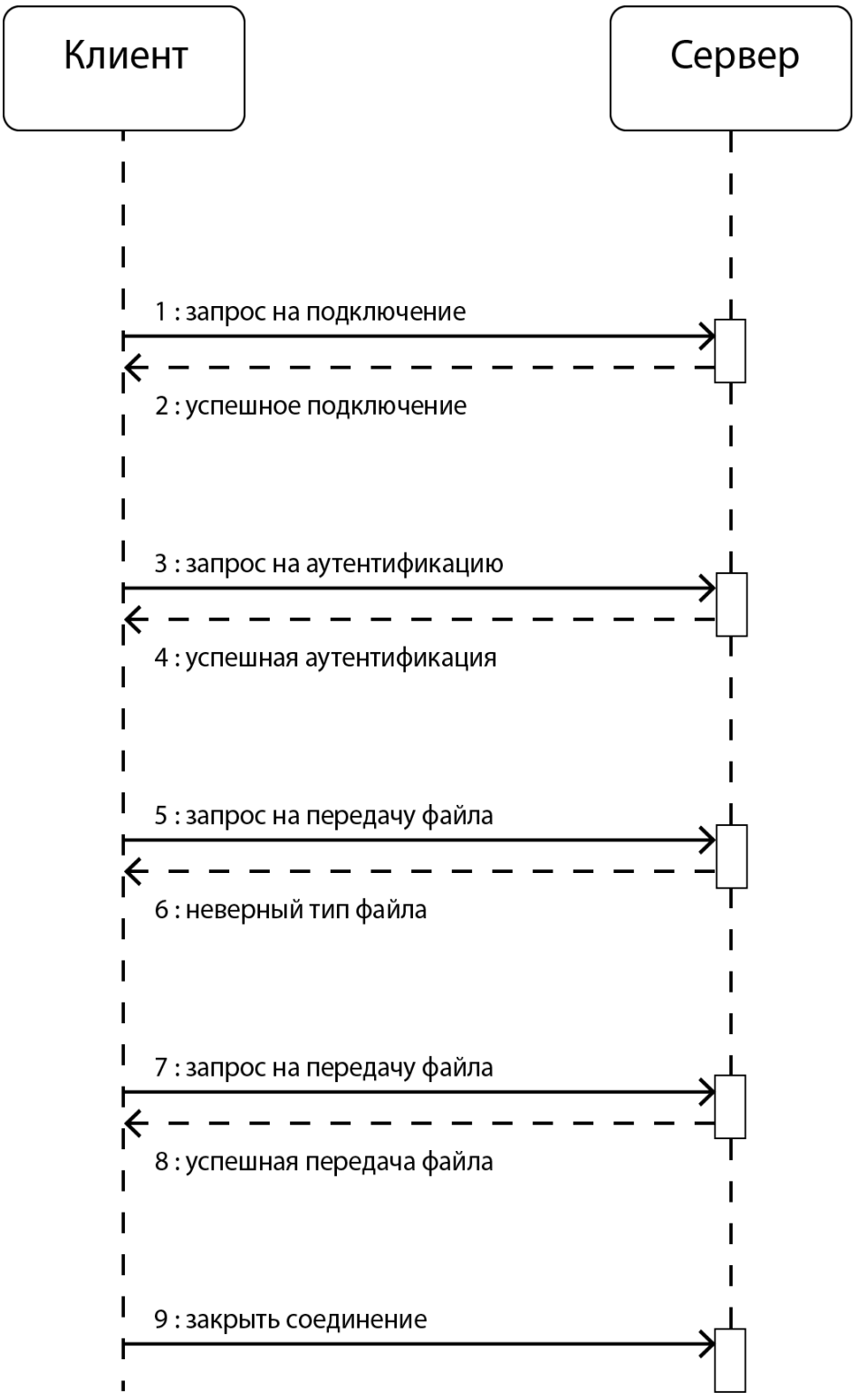


Рисунок А12 — Неверный тип файла (аутентификация)

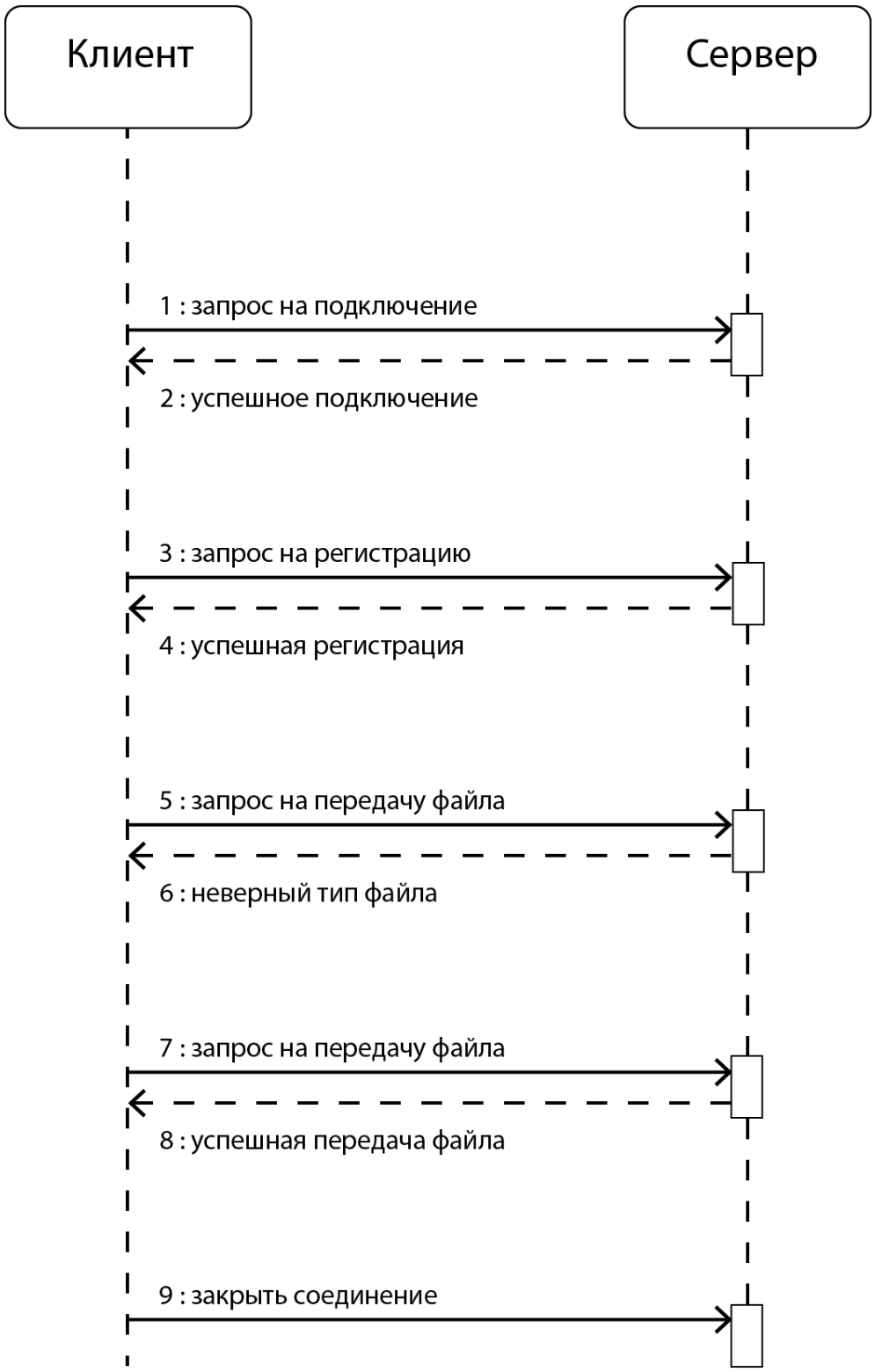


Рисунок А13 — Неверный тип файла (регистрация)

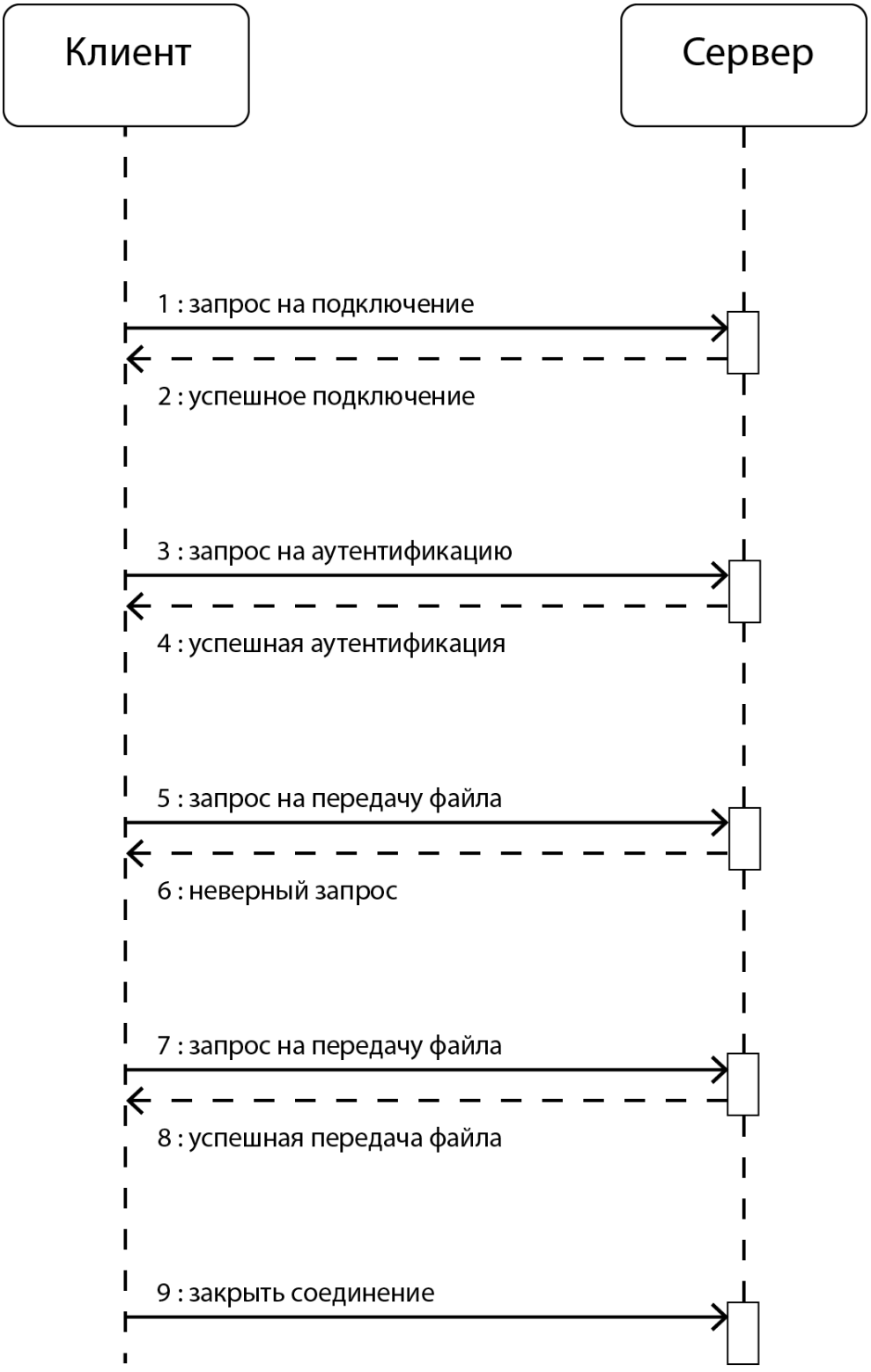


Рисунок А14 — Неверный запрос при передачи файла (аутентификация)

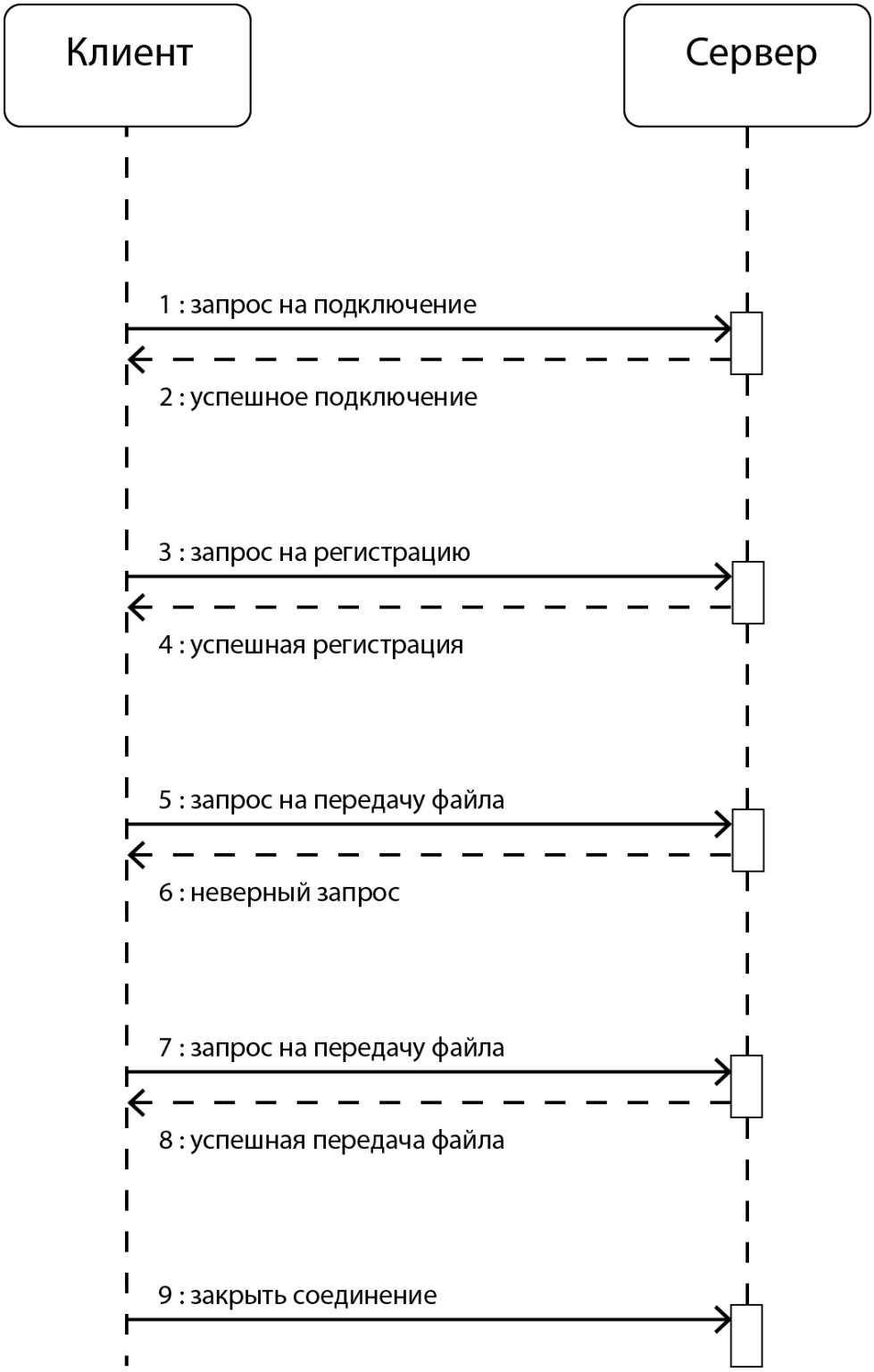


Рисунок А15 — Неверный запрос при передачи файла (регистрация)

Приложение Б

Код программ
(обязательное)

Приложение Б1 — Код для работы сервера и клиента

```
from typing import Tuple

# socket imports
from socket import socket
from _socket import getdefaulttimeout


class Headers:
    """
    Класс, содержащий константы заголовков.

    Attributes:
        CONN: Константа для подключения.
        AUTH: Константа для аутентификации.
        REG: Константа для регистрации.
        UP: константа для загрузки.
    """
    CONN = b'connect'
    AUTH = b'auth'
    REG = b'reg'
    UP = b'upload'


class Messages:
    """
    Класс, содержащий сообщения для обмена данными.

    Attributes:
```

```
        CONN_SUC: Успешное подключение.
        CONN_ERR: Ошибка подключения.
        AUTH_SUC: Успешная аутентификация.
        AUTH_ERR: Неверный логин или пароль.
        REG_SUC: Успешная регистрация.
        REG_ERR: Пользователь уже существует.
        UP_SUC: Успешная передача файла.
        UP_ERR: Неверный тип файла.
        REQ_ERR: Неверный запрос.
    """
    # Сообщения о подключении
    CONN_SUC = 'успешное подключение'.encode("latin-1")
    CONN_ERR = 'ошибка подключения'.encode("latin-1")

    # Сообщения авторизации
    AUTH_SUC = 'успешная аутентификация'.encode("latin-1")
    AUTH_ERR = 'неверный логин или пароль'.encode("latin-1")

    # Регистрационные сообщения
    REG_SUC = 'успешная регистрация'.encode("latin-1")
    REG_ERR = 'пользователь уже существует'.encode("latin-1")

    # Файловые сообщения
    UP_SUC = 'успешная передача файла'.encode("latin-1")
    UP_ERR = 'неверный тип файла'.encode("latin-1")

    # Общие сообщения
    REQ_ERR = 'неверный запрос'.encode("latin-1")


class Socket(socket):
    def __init__(self, *args, **kwargs):
        """Расширение класса socket с дополнительной функциональностью."""
        super().__init__(*args, **kwargs)
        self._separator = b'@sep'
        self._end = b'@end'
```

```
        self._chunk_size = 1024

    # Attribute methods
    @property
    def separator(self) -> bytes:
        return self._separator

    @separator.setter
    def separator(self, value: bytes) -> None:
        self._separator = value

    @property
    def end(self) -> bytes:
        return self._end

    @end.setter
    def end(self, value: bytes):
        self._end = value

    @property
    def chunk_size(self) -> int:
        return self._chunk_size

    @chunk_size.setter
    def chunk_size(self, value: int):
        self._chunk_size = value

    # Socket methods
    def receive(self, target_len: int = 3) -> Tuple[bytes, ...]:
        """
        Получение сообщения до тех пор, пока не будет найден конечный маркер.

        Args:
            target_len (int): Целевая длина для получения. Значение по умолчанию равно
3.
```

```
    Returns:
        Tuple[bytes, ...]: Кортеж полученных сообщений.
    """
    buffer = b''
    while not (self.end in buffer):
        chunk = super().recv(self.chunk_size)
        if not chunk:
            break
        buffer += chunk

    buffer = buffer.replace(self.end, b'')
    buffer = buffer.split(self.separator)[:target_len]
    buffer += [b'@null'] * (target_len - len(buffer))
    return tuple(buffer)

def send(self, *msgs: bytes) -> None:
    """
    Отправка сообщения, соединенные разделителем.

   Args:
       *msgs (bytes): сообщения для отправки.
   """
    buffer = self.separator.join(msgs)

    while buffer:
        chunk = buffer[:self.chunk_size]
        buffer = buffer[self.chunk_size:]
        super().sendall(chunk)
    super().sendall(self.end)

def accept(self) -> Tuple['Socket', Tuple]:
    """
    Принимает входящее соединение и возвращает новый объект сокета и адрес.

    Returns:
        Tuple['Socket', Tuple]: Tuple, содержащий новый объект сокета и адрес.
```

```
        """
        fd, addr = self._accept()
        sock = Socket(self.family, self.type, self.proto, fileno=fd)
        if getdefaulttimeout() is None and self.gettimeout():
            sock.setblocking(True)
        return sock, addr
```

Приложение Б2 — Код программы сервера

```
import ipaddress
from os.path import exists as is_exists
import pickle
from socket import AF_INET, SOCK_STREAM
from threading import Thread
from typing import Tuple, List

from common.network import Socket, Headers, Messages

PATH = './workgroup.bin'
HOST = '127.0.0.1'
PORT = 25565
CONN_LIMIT = 3



def file_type(content: bytes) -> bytes:
    """
    Определить тип файла на основе содержимого.

    Args:
        content (bytes): Содержимое файла.

    Returns:
        bytes: Идентификатор типа файла ('1' для текстового, '2' для двоичного).
    """
    try:
        content.decode()
        return b'1'  # Текстовый тип файла
```

```
    except UnicodeDecodeError:
        return b'2'  # Бинарный тип файла


class Data(dict):
    """
    Пользовательский класс данных для управления серверными данными.
    """

    def __init__(self, path: str = './workgroup.bin'):
        """
        Инициализация объекта Data.

        Args:
            path (str): Путь к файлу данных.
        """
        self._path = path
        if is_exists(path):  # Проверка, существует ли файл
            file = open(path, 'rb')
            try:
                super().__init__(pickle.load(file))  # Загрузка данных из файла
                file.close()
                return
            except EOFError:
                file.close()
        super().__init__({'whitelist': ['127.0.0.1',
                                        ], 'users': {}})  # Данные по умолчанию, если
файл не существует
        with open(path, 'wb+') as file:
            pickle.dump(self, file)  # Сохранение данных в файле

    def commit(self):
        """
        Фиксация данных в файле.
        """
        with open(self._path, 'wb+') as file:
```

```
            pickle.dump(self, file)  # Update and store data to the file


class Handler(Thread):
    """
    Пользовательский класс для обработки клиентских подключений.
    """

    def __init__(self, socket: Socket, address: Tuple, data: Data, queue:
List['Handler']):
        """
        Инициализация объекта обработчика

        Args:
            socket (Socket): Объект socket.
            address (Tuple): Информация об адресе.
            data (Data): Объект данных сервера.
            queue (List): Список с активными обработчиками.
        """
        super().__init__()
        # Внешние атрибуты
        self._socket = socket
        self._data = data

        # Собственные атрибуты
        self._state = self.connect
        self._version = None

        # Setup
        self._exit_flag = not (address[0] in data['whitelist'] and len(queue) <
CONN_LIMIT)

    def run(self):
        """
        Запуск потока обработчика.
        """
```

```
    try:
        while self._state:
            self._state()
    except (TimeoutError, ConnectionAbortedError) as exception:
        print(f"ERR: '{exception}'. Connection close.")
        self._socket.close()

# Состояние подключения
def connect(self) -> None:
    """
    Обработка состояния подключения.
    """
    msgs = self._socket.receive(target_len=4)

    if self._exit_flag:
        self._state = None
        return self._socket.send(Headers.CONN, Messages.CONN_ERR)

    isHeader = (msgs[0] == Headers.CONN)
    isMethod = msgs[2] in (Headers.AUTH, Headers.REG)
    isVersion = msgs[3] in (b'1', b'2')

    if not (isHeader and isVersion and isMethod):
        return self._socket.send(Headers.CONN, Messages.REQ_ERR)

    self._socket.send(Headers.CONN, Messages.CONN_SUC)
    self._state = self.auth if msgs[2] == Headers.AUTH else self.reg
    self._version = msgs[3]

def auth(self):
    """
    Обработка состояния аутентификации.
    """
    msgs = self._socket.receive(target_len=3)

    isHeader = (msgs[0] == Headers.AUTH)
```

```
        isValid = (msgs[1] in self._data['users'] and self._data['users'][msgs[1]] ==
msgs[2])

        if not isHeader:
            return self._socket.send(Headers.AUTH, Messages.REQ_ERR)
        elif not isValid:
            return self._socket.send(Headers.AUTH, Messages.AUTH_ERR)

        self._socket.send(Headers.AUTH, Messages.AUTH_SUC)
        self._state = self.upload

    def reg(self):
        """
        Обработать состояние регистрации нового пользователя.
        """
        msgs = self._socket.receive(target_len=3)

        isHeader = (msgs[0] == Headers.REG)
        isUser = not (msgs[1] in self._data['users'] or msgs[1] == b'@null')
        isPass = not (msgs[2] == b'@null')

        if not isHeader:
            return self._socket.send(Headers.REG, Messages.REQ_ERR)
        elif not (isUser and isPass):
            return self._socket.send(Headers.REG, Messages.REG_ERR)

        self._data['users'][msgs[1]] = msgs[2]
        self._data.commit()

        self._socket.send(Headers.REG, Messages.REG_SUC)
        self._state = self.upload

    def upload(self):
        """
        Обработка состояния загрузки файла.
        """
```

```
        msgs = self._socket.receive(target_len=3)

        isHeader = (msgs[0] == Headers.UP)
        isFileName = not (msgs[1] == b'@null')
        isContent = not (msgs[2] == b'@null')

        if not (isHeader and isFileName and isContent):
            return self._socket.send(Headers.UP, Messages.REQ_ERR)

        fileType = file_type(msgs[2])
        if fileType != self._version:
            return self._socket.send(Headers.UP, Messages.UP_ERR)

        with open(msgs[1].decode(), 'wb+') as file:
            file.write(msgs[2])
        self._socket.send(Headers.UP, Messages.UP_SUC)
        self._state = None


# Установка атрибутов сервера
server_data = Data(PATH)  # Инициализация серверных данных
server_queue = []  # Инициализация пустой очереди для обработчиков

# Добавление ip адресов
start_ip = ipaddress.IPv4Address('192.168.0.128')
end_ip = ipaddress.IPv4Address('192.168.0.196')

for ip_int in range(int(start_ip), int(end_ip) + 1):
    ip = ipaddress.IPv4Address(ip_int)
    if str(ip) not in server_data['whitelist']:
        server_data['whitelist'].append(str(ip))

server_data.commit()
print('ip`s:', server_data['whitelist'])

# Установка сокета
```

```
server_socket = Socket(AF_INET, SOCK_STREAM)  # Создание socket объекта
server_socket.bind((HOST, PORT))  # Привязка сокета к хосту и порту
server_socket.listen()  # Начать прослушивать соединение

# Обработка и прослушивание входящих соединений
while True:
    # Принятие входящего соединения
    client_socket, client_address = server_socket.accept()

    # Обновление очереди, удаление неактивных обработчиков
    server_queue = [_ for _ in server_queue if _.is_alive()]

    # Обработка соединения
    client_socket.settimeout(120)
    handler = Handler(client_socket, client_address, server_data, server_queue)
    handler.start()
    server_queue.append(handler)
```

Приложение Б3 — Код программы клиента

```
import os.path
from os.path import exists as is_exists
from socket import AF_INET, SOCK_STREAM
from threading import Thread

from common.network import Socket, Headers, Messages


class Handler(Thread):
    """
    Класс, представляющий обработчик соединения.

    Класс управляет текущим состоянием соединения и обрабатывает запросы от клиента.
    """

    def __init__(self, socket: Socket):
```

```
        """
        Инициализация объекта Handler.

        Args:
            socket (Socket): Объект socket.
        """
        super().__init__()
        # Внешние атрибуты
        self._socket = socket

        # Внутренние атрибуты
        self._state = self.connect

    def run(self):
        try:
            while self._state:
                self._state()
        except (TimeoutError, ConnectionAbortedError) as exception:
            print(f"ERR: '{exception}'. Connection close.")
            self._socket.close()

    def connect(self) -> None:
        """
        Метод для установления соединения с сервером.

        Пользователь выбирает метод (аутентификация или регистрация) и версию
протокола.
        """
        user_agent = b'Python3 Client Win64'
        method = input('Введите метод (<auth> или <reg>): ').strip().encode("latin-1")
        version = input('Введите версию (<1> или <2>): ').strip().encode("latin-1")

        self._socket.send(Headers.CONN, user_agent, method, version)
        msgs = self._socket.receive(target_len=2)

        # Проверка корректности запроса
```

```
        if msgs[1] == Messages.CONN_ERR:
            self._state = None
            return print(msgs[1].decode())
        elif msgs[1] != Messages.CONN_SUC:
            return print(msgs[1].decode())

        # Определение метода
        if method == Headers.AUTH:
            self._state = self.auth
        elif method == Headers.REG:
            self._state = self.reg
        return print(msgs[1].decode())

    def auth(self):
        """
        Метод для аутентификации пользователя.

        Пользователь вводит логин и пароль, затем отправляет их для аутентификации на сервере.
        """
        login = input('Введите логин: ').strip().encode("latin-1")
        password = input('Введите пароль: ').strip().encode("latin-1")

        self._socket.send(Headers.AUTH, login, password)
        msgs = self._socket.receive(target_len=2)

        if msgs[1] != Messages.AUTH_SUC:
            return print(msgs[1].decode())

        self._state = self.upload
        return print(msgs[1].decode())

    def reg(self):
        """
        Метод для регистрации нового пользователя.
```

```
        Пользователь вводит логин и пароль, затем отправляет их для регистрации на
сервере.
        """
        login = input('Введите логин: ').strip().encode("latin-1")
        password = input('Введите пароль: ').strip().encode("latin-1")

        self._socket.send(Headers.REG, login, password)
        msgs = self._socket.receive(target_len=2)

        if msgs[1] != Messages.REG_SUC:
            return print(msgs[1].decode())

        self._state = self.upload
        return print(msgs[1].decode())

    def upload(self):
        """
        Метод для загрузки файла на сервер.

        Пользователь вводит путь к файлу, который затем отправляется на сервер.
        """
        path = input('Введите путь: ').strip()
        path = path.replace('"', '').replace("'", "")
        if not is_exists(path):
            return print(f"LocalERR: файл {path} не существует")

        fileName = os.path.basename(path).encode("latin-1")
        with open(path, 'rb') as file:
            content = file.read()

        self._socket.send(Headers.UP, fileName, content)
        msgs = self._socket.receive(target_len=2)

        if msgs[1] != Messages.UP_SUC:
            return print(msgs[1].decode())
```

```
        self._state = None
        return print(msgs[1].decode())


client_socket = Socket(AF_INET, SOCK_STREAM)
client_socket.settimeout(120)
client_socket.connect(('127.0.0.1', 25565))
handler = Handler(client_socket)
handler.start()
```