Курсовая Работа

по дисциплине «Сетевые Технологии»

на тему «Разработка приложений с сетевым взаимодействием»

Выполнил студент: Гусев Д. А.

Группа: 21ПИ1

Руководитель: Липилин О. В.

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 клиентсерверной 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. Была разработана таблица запросов-ответов между сервером и клиентом. Результат представлен в таблице 1.

Таблица 1 — Запросы — ответы

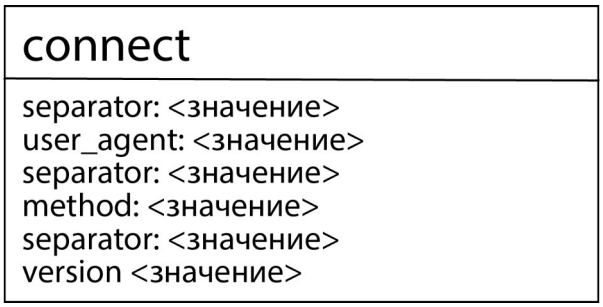
запрос клиент — сервер	ответ сервер — клиент
запрос на подключение	успешное подключение
	ошибка подключения
	неверный запрос
запрос на аутентификацию	успешная аутентификация
	неверный логин или пароль
	неверный запрос
запрос на регистрацию	успешная регистрация
	пользователь уже существует
	неверный запрос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

Были разработаны диаграммы последовательностей сетевого взаимодействия клиента и сервера.

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);

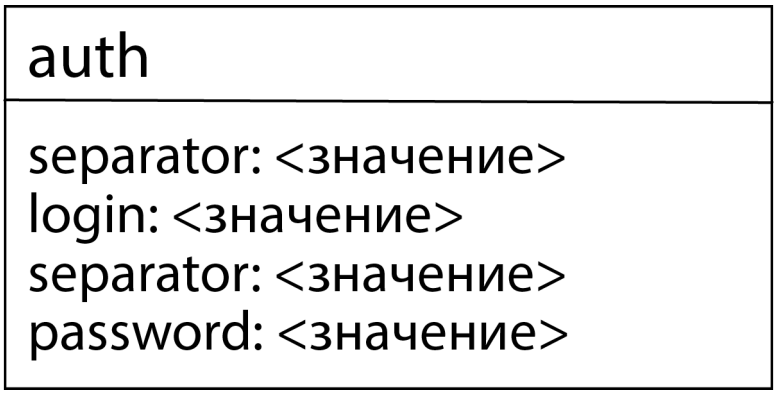
user_agent: <значение> - строка символов в кодировке UTF-8, поле «значение» содержит идентификатор программы-клиента (от 1 до 32 символов английского алфавита), используется для определения сессии каждой программы-клиента;

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;

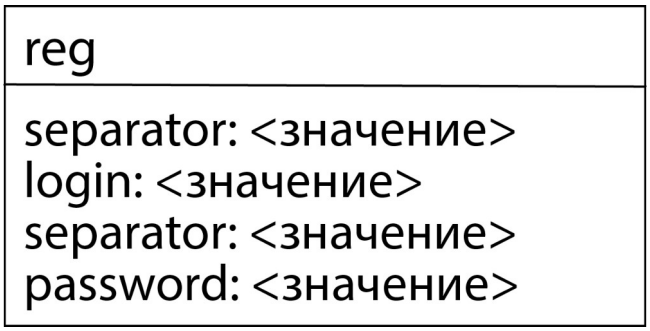
separator: <значение> - строка байт, поле значение содержит разделитель сообщений (4 байта);

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:

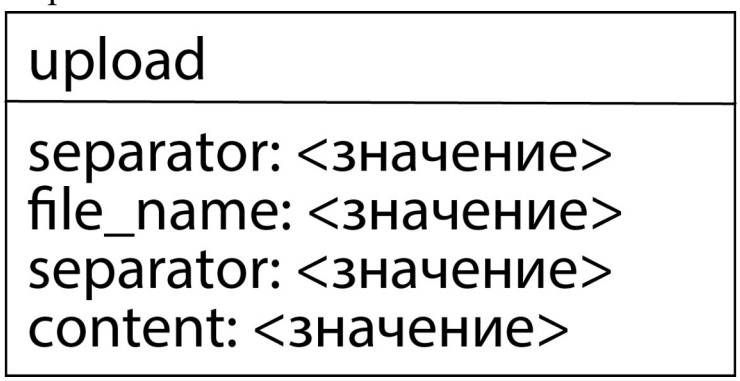
reg - заголовок запроса, строка символов в кодировке UTF-8; separator: <значение> - строка байт, поле значение содержит разделитель сообщений (4 байта);

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.

1) Разарботка общих классов и функций.

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.

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 <*classpickle*> 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 находитсяв репозитории hagithub.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@serуспешное
подключение@end reg@sepuser1@seppass1@end
reg@serуспешная регистрация@end
upload@sepfile.jpg@sep<#^- #JFIF ### # # C #####
#########2!####=,.$2I@LKG@FEPZsbPUmVEFd^mw{js~$t^ C#######;!!;|
SFS|||||||||||||||||||||||||||||||||||||||||||||||||<^ ## #
###" ######^ # ####    ######^ $# ###### # ##
####!A#1#2B"Qq^ ######    ###    #### ###
#####12AQ^# #### ? &##■&&
^^0^#I^#[##«]<*****2><£^^i..;or^F+Z
go^s<^i^3^v^^^H^%z^eB^x<*[^H***H#^83cfG#wi>;<*d Ј#C^ #Sj^V<#n#0<$# #
(#«5#N<*W.q3<fcfcS ^L<$e    *4te«HMg4ltPD& ?#«!#<***K@«I{ □ -
!+<##    4(H4M№##BL#4$end
upload@serуспешная 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@seруспешное подключение@end auth@sepuser@seppassword@end
auth@seруспешная аутентификация@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 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@seрошибка
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@sepHeBepHbrn 3anpoc@end connect@sepPython3 Client
Win64@sepauth@sep1@end connect@seруспешное подключение@end
auth@sepuser2@seppass2@end auth@sepHeBepHbrn логин или пароль@end
auth@sepuser@seppassword@end auth@seруспешная аутентификация@end
upload@sepfile.jpg@sep<$fc^ #JFIF ### # # C #####
#########2!####=,.$2I@LKG@FEPZsbPUmVEFd^mw{^ C#######;!!;|
SFS|||||||||||||||||||||||||||||||||||||||||||||||||<^ ## #
###" ######^ # ####    ######^ $# ###### # ##
####!A#1#2B"Qq^ ######    ###    #### ###    #####12AQ^
## #### ?
gO^s<<^l^3^V^^^}<*e%Z^eB^x<*[^H***H#^3cfG#WI'.;<*d j#C^
#Sj^V<#n#0<$# ^6
(#«5#N<*W.q3<fcfcS ^L<$e    ?#«!#<***K@«I{ □ -
J+^*#    <*^M*M#BL#<#@end
upload@sepHeBepHbrn 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@sepHeBepHbrn 3anpoc@end connect@sepPython3 Client
Win64@sepauth@sep1@end connect@seруспешное подключение@end
auth@sepuser2@seppass2@end auth@sepHeBepHbrn логин или пароль@end
auth@sepuser@seppassword@end auth@seруспешная аутентификация@end
upload@sepfile.jpg@sep<$fc^ #JFIF ### # # C #####
#########2!####=,.$2I@LKG@FEPZsbPUmVEFd^mw{^ C#######;!!;|
SFS||||||||||||||||||||||||||||||||||||||||||||||||<^ ## #
###" ######^ # ####    ######^ $# ###### # ##
####!A#1#2B"Qq^ ######    ###    #### ###    #####12AQ^
## #### ? ^B##^    O^#1ϕ1*[##4E]4***2>4^+
      go^s<^l^3^v^^^H^%z^eB^x<*[^H***H#^83cfG#wi>;<*d J#C^ #Sj^V<0fn#0<$#
^6
(#«5#NN*W.q3<fcfcS ^L<$e    *4te«HMg4ltPD& ?#«!#<***K@«I{ □ -
J+^*#    4(H4MN##BL#4$end
      upload@sepHeBepHbrn тип файла@end
      upload@sepfile.txt@sepEat some more of these soft French rolls, and
drink some tea@end
      upload@seруспешная передача файла@end
```

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.m/pluginfile.php/2566362/mod_resource/content/2/МУ %20КР %20Разработка%20протокола^£ - 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^£ - 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/plugmfile.php/2587836/mod_resош'ce^ontent/0/Лабораторная%20работа%204^£ - Дата доступа: 05.05.2024.