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 «13» февраля 2024 г. Д. А. Гусев

Студент

Руководитель О. В. Липилин

Программная реализация

Код программ клиента и сервера находится на в репозитории на github.com:

E:\Projects\PGU\6 Семестр\Network-Technologies\Coursework\STEP 3\sources

1) Разарботка общих классов и функций. Код представленных классов и функций находится в репозитории на github.com.

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.

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.

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*>.

4) Работа программы представлена на рисунках 1-3.

```
C:\Users\Goose\.venv\Scripts\python.exe C:\Users\Goose\PycharmProjects\Network-Technologies\client.py
Введите метод (<auth> или <reg>): reg
Введите версию (<1> или <2>): 1
успешное подключение
Введите логин: user
Введите пароль: pass
успешная регистрация
Введите путь: "C:\Users\Goose\Downloads\YG8iNRoouJ4.jpg"
LocalERR: файл "C:\Users\Goose\Downloads\YG8iNRoouJ4.jpg" не существует
Введите путь: C:\Users\Goose\Downloads\YG8iNRoouJ4.jpg
неверный тип файла
Введите путь: C:\Users\Goose\Downloads\YG8iNRoouJ4.txt
неверный тип файла
Введите путь: C:\Users\Goose\Downloads\test.txt
успешная передача файла

Process finished with exit code 0
```

Рисунок 1 — Работа программы (Регистрация)

```
Введите метод (<auth> или <reg>): auth
Введите версию (<1> или <2>): 3
неверный запрос
Введите метод (<auth> или <reg>): auth
Введите версию (<1> или <2>): 2
успешное подключение
Введите логин: user
Введите пароль: pass
успешная аутентификация
Введите путь: C:\Users\Goose\Downloads\YG8iNRoouJ4.jpg
успешная передача файла

Process finished with exit code 0
```

Рисуонок 2 — Работа программы (Аутентификация)

```
Введите метод (<auth> или <reg>): фгер
Введите версию (<1> или <2>): 1
ошибка подключения
```

Рисунок 3 — Работа программы (Сервер занят)