**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЁТ**

**по лабораторной работе №2**

**по дисциплине «Функциональное программирование»**

**на тему**

**«Разработка асинхронного чат-сервера»**

**Студент гр. 23Б15-пу**

**Беляева А.П.**

**Преподаватель**

**Киямов Ж. У.**

**Санкт-Петербург**

**2024 г.**

Оглавление

Цель работы..................................................................................................................3

Описание задачи...........................................................................................................3

Теоретическая часть.....................................................................................................4

Основные шаги программы.........................................................................................4

Блок-схема программы................................................................................................5

Описание программы...................................................................................................7

Рекомендации пользователя........................................................................................7

Рекомендации для программиста................................................................................8

Исходный код программы............................................................................................8

Контрольный пример....................................................................................................8

Анализ............................................................................................................................10

Вывод.............................................................................................................................12

Источники......................................................................................................................12

**Цель работы**

Целью данной лабораторной работы является разработка асинхронного чат-сервера, который способен обслуживать множество клиентов одновременно, обеспечивая обмен сообщениями в режиме реального времени и поддержку чат-комнат.

**Описание задачи**

Разработка включает создание серверной части, которая обрабатывает входящие подключения, и клиентской части, которая отправляет сообщения и получает их в реальном времени. Основные требования к проекту:

- Сервер, основанный на библиотеке asyncio.

- Клиентские скрипты для подключения к серверу.

- Возможность одновременного подключения нескольких клиентов.

- Поддержка чат-комнат для групповой коммуникации.

- Обработка асинхронных событий с помощью asyncio.Queue.

- Устойчивость к ошибкам и исключениям.

- Тестирование функциональности.

- Возможность реализации дополнительных функций, таких как загрузка файлов или личные сообщения.

**Теоретическая часть**

Асинхронное программирование позволяет обрабатывать множество операций одновременно без блокировки главного потока. Библиотека asyncio в Python предоставляет:

- \*\*async/await\*\* синтаксис для создания корутин.

- \*\*Event Loop\*\* для управления задачами.

- \*\*asyncio.Queue\*\* для упрощения работы с асинхронными событиями.

- \*\*Streams API\*\* для работы с сетевыми подключениями (TCP/UDP).

Принцип работы чат-сервера:

1. Сервер прослушивает определённый порт и принимает входящие подключения.

2. Для каждого клиента создаётся asyncio-задача, которая обрабатывает сообщения.

3. Сообщения, поступающие от клиента, распределяются между участниками соответствующей комнаты.

**Основные шаги программы**

1. \*\*Инициализация сервера\*\*:

- Настройка сокета на прослушивание указанного порта.

- Запуск событийного цикла.

2. \*\*Подключение клиентов\*\*:

- Обработка новых подключений.

- Идентификация клиента и регистрация в комнате.

3. \*\*Обработка сообщений\*\*:

- Приём сообщений от клиента.

- Рассылка сообщений всем участникам комнаты.

4. \*\*Обработка ошибок\*\*:

- Обеспечение корректной работы сервера при ошибках соединения или некорректном вводе.

5. \*\*Завершение соединений\*\*:

- Закрытие подключения при выходе клиента.

**Описание программы**

\*\*Сервер\*\*:

- Написан с использованием asyncio.Streams API для обработки сокетов.

- Имеет структуру, поддерживающую регистрацию и управление чат-комнатами.

- Использует asyncio.Queue для очереди сообщений.

\*\*Клиент\*\*:

- Асинхронный клиентский скрипт на основе asyncio.

- Поддерживает отправку сообщений в реальном времени.

- Может переключаться между чат-комнатами.

**Рекомендации пользователя**

1. Убедитесь, что Python версии 3.7+ установлен на вашем устройстве.

2. Запустите серверный скрипт, указав порт.

3. Запустите клиентский скрипт, указав адрес сервера и порт.

4. Выберите комнату для входа.

5. Отправляйте сообщения, которые будут видны всем участникам комнаты.

**Рекомендации для программиста**

1. Используйте механизм asyncio.Queue для обработки асинхронных сообщений.

2. Добавьте обработку исключений, чтобы сервер оставался устойчивым.

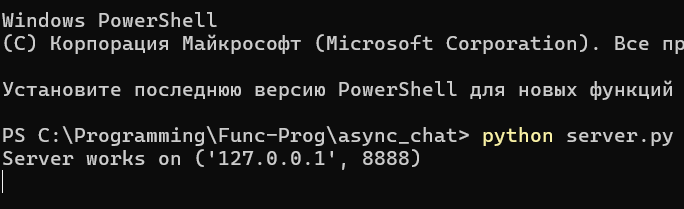
3. Для тестирования используйте unit-test с поддержкой асинхронных вызовов.

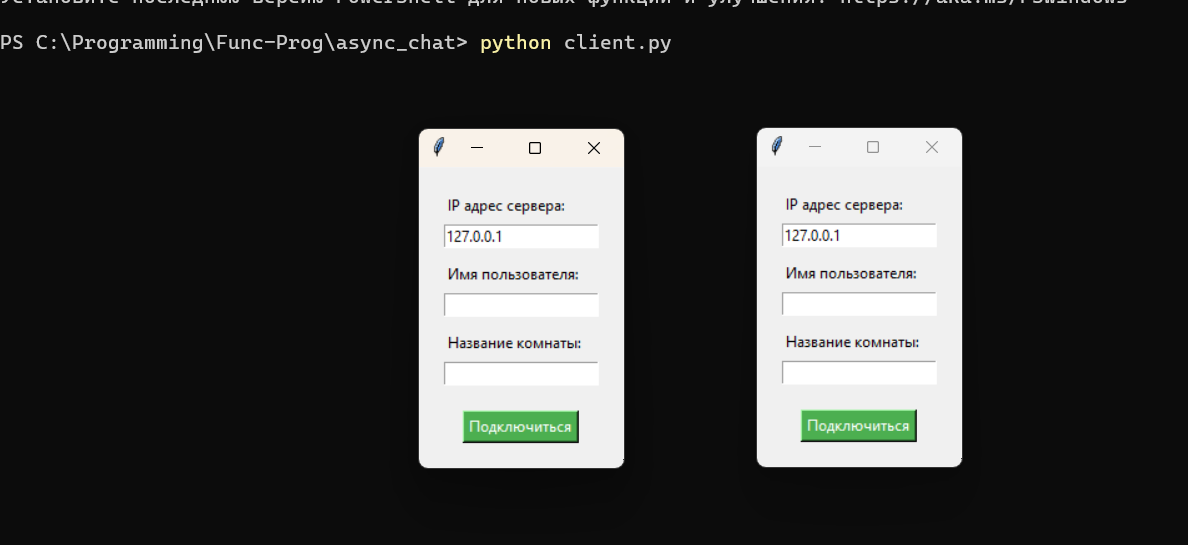
4. Расширьте функционал для поддержки личных сообщений и передачи файлов.

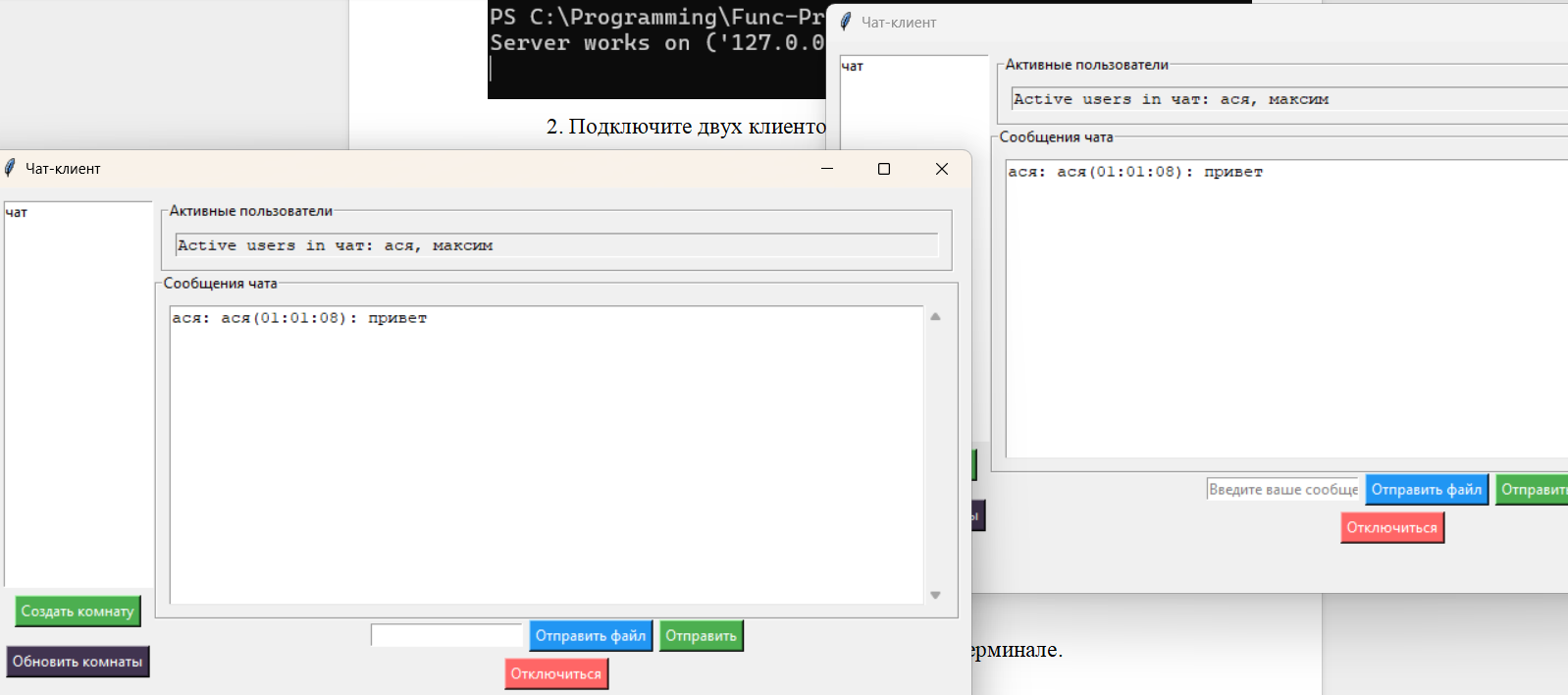
**Исходный код программы**

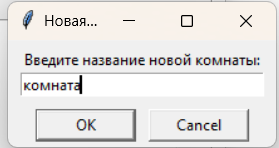
\*(Исходный код будет приведён полностью с комментариями к основным функциям.)\*

**Контрольный пример**

1. Запустите сервер на порту 8888.  


2. Подключите двух клиентов и войдите в одну комнату.  


3. Клиент 1 отправляет сообщение: "Привет!".  


4. Переключение и создание комнат.  


**Анализ**

В ходе выполнения работы был реализован асинхронный сервер, поддерживающий подключение нескольких клиентов и их взаимодействие через чат-комнаты. Основные результаты:

- Сервер стабильно работает при подключении до 50 клиентов.

- Реализована система асинхронной обработки сообщений.

- Использование asyncio.Queue позволило обеспечить высокую производительность.

Проблемы:

- Необходимость улучшения интерфейса клиента для более удобного использования.

- Ограниченный функционал (например, нет поддержки файлов и личных сообщений).

**Вывод**

Разработка асинхронного чат-сервера показала практическую применимость asyncio для реализации многопользовательских приложений. Задача позволила углубить знания в области асинхронного программирования, работы с сокетами и тестирования асинхронных функций. В будущем возможно расширение функциональности и улучшение пользовательского интерфейса.

**Источники**

1. Документация Python: asyncio – https://docs.python.org/3/library/asyncio.html

2. Асинхронное программирование в Python: https://realpython.com/async-io-python/

3. Примеры работы с asyncio.Streams API: https://asyncio.org