Санкт-Петербургский государственный университет

Направление ”Большие данные и распределенная цифровая платформа”

Отчёт по проекту
”Разработка асинхронного чат-сервера”

Выполнил:
Зайнуллин Мансур Альбертович
Группа: 23.Б16-пу

Руководитель:
Киямов Жасур Уткирович

Санкт-Петербург
2024

Оглавление

1 Цель работы

Создать асинхронный чат-сервер, способный обслуживать множество клиентов одновременно и обеспечивающий обмен сообщениями в реальном времени.

2 Описание задачи

Необходимо разработать асинхронный чат-сервер на языке Python с использованием библиотеки `asyncio`. Сервер должен выполнять следующие функции:

1. **Прослушивание порта**:
    - Сервер слушает определённый порт для входящих подключений от клиентов.
2. **Обработка клиентов**:
    - Возможность одновременного подключения множества клиентов с использованием asyncio-задач (coroutines).
3. **Обмен сообщениями**:
    - Клиенты могут отправлять и получать сообщения в реальном времени.
4. **Чат-комнаты**:
    - Поддержка создания и присоединения к различным чат-комнатам. Пользователи могут выбирать, в какую комнату войти.
5. **Рассылка сообщений**:
    - Сообщения от одного клиента рассылаются всем участникам текущей комнаты, кроме отправителя.
6. **Асинхронные события**:
    - Использование `asyncio.Queue` для обработки событий, таких как получение нового сообщения.
7. **Обработка ошибок**:
    - Надёжная обработка исключений для предотвращения остановки сервера при неверном поведении клиента.

8. **Тестирование**:

    - Разработка тестов для проверки функциональности сервера, включая тестирование асинхронных функций.

9. **Дополнительные функции**:

    - Возможность загрузки файлов и отправка личных сообщений между пользователями.

Кроме серверной части, необходимо разработать асинхронный клиент, способный подключаться к серверу, отправлять сообщения и выполнять команды, такие как создание комнаты, присоединение к комнате и отправка личных сообщений.

3 Теоретическая часть

3.1 Асинхронное программирование

Асинхронное программирование позволяет выполнять несколько операций одновременно, не блокируя основной поток выполнения. Это достигается за счёт использования корутин, которые могут приостанавливаться и возобновляться, позволяя другим задачам выполняться в промежутках ожидания.

3.2 Библиотека `asyncio`

`asyncio` — стандартная библиотека Python для написания одновременного кода с использованием корутин. Она предоставляет инфраструктуру для управления событиями, корутинами и асинхронными задачами, что делает её подходящей для разработки сетевых приложений, таких как чат-серверы.

3.3 Сокеты

Сокеты обеспечивают механизм для обмена данными между различными процессами через сеть. В контексте чат-сервера сокеты используются для установления соединения между клиентами и сервером, а также для передачи сообщений.

3.4 Асинхронные события

Асинхронные события управляются с помощью очередей, таких как `asyncio.Queue`, которые позволяют эффективно обрабатывать входящие и исходящие данные без блокировки выполнения других задач. Это критически важно для обработки большого количества подключений и сообщений в реальном времени.

3.5 Многозадачность с корутинами

Корутины — это функции, которые могут быть приостановлены и возобновлены, что позволяет реализовать многозадачность внутри одного потока. В `asyncio` каждая задача выполняется как отдельная корутина, что позволяет серверу обрабатывать множество клиентов одновременно без необходимости использования многопоточности или многопроцессности.

3.6 Обработка исключений

Надёжная обработка исключений необходима для обеспечения стабильной работы сервера. Исключения могут возникать при ошибках соединения, некорректных данных от клиентов или внутренних ошибках сервера. Правильная обработка исключений позволяет серверу продолжать работу, несмотря на возникающие ошибки, и предоставляет информативные сообщения об ошибках для администраторов и пользователей.

3.7 Клиент-серверная архитектура

Клиент-серверная модель предполагает разделение системы на сервер, который предоставляет услуги, и клиентов, которые эти услуги используют. В контексте чат-сервера сервер управляет подключениями, обработкой сообщений и распределением их между клиентами, тогда как клиенты отправляют и получают сообщения через сервер.

3.8 Тестирование асинхронных программ

Тестирование асинхронных приложений требует специальных подходов для проверки корректности выполнения корутин и взаимодействия между ними. Использование фреймворков, таких как `pytest` с поддержкой `asyncio`, позволяет писать тесты, которые проверяют функциональность сервера в условиях многопоточности и асинхронности, обеспечивая надёжность и стабильность приложения.

4 Описание программы

4.1 Алгоритм работы

1. **Запуск сервера**:
    - При запуске `server.py` создаётся асинхронный сервер, который прослушивает порт 8888 на локальном хосте (`127.0.0.1`).
    - Сервер запускается в отдельном потоке и интегрируется с графическим интерфейсом на Tkinter для отображения подключённых клиентов, комнат и логов.
2. **Подключение клиента**:
    - Клиент запускает `client.py`, который подключается к серверу через указанный IP и порт.
    - После подключения клиенту предлагается ввести своё имя.
3. **Управление чат-комнатами**:
    - Клиенты могут создавать новые комнаты, присоединяться к существующим, покидать комнаты и просматривать список доступных комнат.
4. **Обмен сообщениями**:
    - Клиенты могут отправлять сообщения в текущую комнату, которые затем рассылаются всем участникам этой комнаты.
    - Также доступна отправка личных сообщений другим пользователям.
5. **Асинхронная обработка**:
    - Сервер и клиент используют асинхронные корутины для обработки подключения, получения и отправки сообщений без блокировки основного потока выполнения.

6. **Обработка исключений**:

    - В обоих приложениях реализована обработка исключений для стабильной работы и предотвращения аварийного завершения при ошибках.

4.2 Основные функции

4.2.1 `server.py`

- **enqueue_log(message)**:
  - Добавляет сообщение в очередь логов и записывает его в лог-файл.
- **enqueue_client_list()**:
  - Обновляет список подключённых клиентов.
- **enqueue_room_list()**:
  - Обновляет список доступных комнат.
- **update_widgets(client_list_widget, room_list_widget, log_widget)**:
  - Обновляет виджеты GUI на основе данных из очередей.
- **get_current_room(writer)**:
  - Получает текущую комнату, в которой находится клиент.
- **broadcast_message(sender_writer, message, room_name)**:
  - Рассылает сообщение всем клиентам в указанной комнате, кроме отправителя.
- **send_private_message(sender_writer, target_name, message)**:
  - Отправляет личное сообщение конкретному пользователю.
- **join_room(writer, room_name)**:

  - Позволяет клиенту присоединиться к указанной комнате.

- **create_room(writer, room_name)**:
  - Создаёт новую комнату.
- **leave_room(writer)**:
  - Позволяет клиенту покинуть текущую комнату.
- **list_rooms(writer)**:
  - Отправляет список доступных комнат клиенту.
- **show_current_chat(writer)**:
  - Показывает клиенту, в какой комнате он находится.
- **list_users(writer)**:
  - Отправляет список подключённых пользователей.
- **show_help(writer)**:
  - Отправляет список доступных команд.
- **upload_file(reader, writer, filename)**:
  - Обрабатывает загрузку файла от клиента.
- **handle_client_connection(reader, writer)**:
  - Обрабатывает подключение нового клиента, включая аутентификацию и обработку сообщений.
- **disconnect_client(writer, client_address)**:
  - Отключает клиента и очищает его данные.
- **start_server()**:
  - Запускает сервер и начинает прослушивание входящих подключений.

- **server_thread()**:
  - Запускает серверный цикл в отдельном потоке.
- **handle_exit(signum, frame)**:
  - Обрабатывает сигналы завершения работы сервера.

4.2.2 `client.py`

- **enqueue_message(message)**:
  - Добавляет сообщение в очередь сообщений и записывает его в лог-файл.
- **enqueue_error(message)**:
  - Добавляет ошибку в очередь ошибок и записывает её в лог-файл.
- **receive_messages(reader)**:
  - Асинхронно получает сообщения от сервера и добавляет их в очередь сообщений.
- **send_message(writer, message)**:
  - Асинхронно отправляет сообщение на сервер.
- **main()**:
  - Основная асинхронная функция для подключения к серверу и отправки имени пользователя.
- **start_async_loop()**:
  - Запускает асинхронный цикл событий в отдельном потоке.
- **handle_exit(signum, frame)**:
  - Обрабатывает сигналы завершения работы клиента.
- **close_connection()**:

  - Асинхронно закрывает соединение с сервером.

- **on_send_button_click()**:
  - Обрабатывает нажатие кнопки ”Отправить” и отправляет сообщение на сервер.

- **update_widgets()**:
  - Обновляет виджеты GUI на основе данных из очередей сообщений и ошибок.

- **get_input(prompt)**:
  - Отображает диалоговое окно для ввода данных пользователем.

- **on_create_room()**:
  - Обрабатывает создание новой комнаты через кнопку.

- **on_join_room()**:
  - Обрабатывает присоединение к комнате через кнопку.

- **on_send_private_message()**:
  - Обрабатывает отправку личного сообщения через кнопку.

- **on_leave_room()**:
  - Обрабатывает покидание комнаты через кнопку.

- **on_list_rooms()**:
  - Обрабатывает запрос списка комнат через кнопку.

- **on_list_users()**:
  - Обрабатывает запрос списка пользователей через кнопку.

- **on_send_command(command)**:
  - Отправляет команду на сервер.

- **prompt_username()**:
  - Запрашивает у пользователя имя при запуске клиента.

5 Рекомендации пользователю

5.1 Инструкция по установке программы

Для установки и настройки программы выполните следующие шаги:

5.1.1 Шаг 1: Установка Python 3.9

1. Перейдите на официальный сайт Python: `https://www.python.org/downloads/release/python-390/`.
2. Скачайте установочный файл Python 3.9 для вашей операционной системы.
3. Запустите установочный файл и следуйте инструкциям. Во время установки обязательно поставьте галочку на опции ”Add Python to PATH”.
4. После установки откройте командную строку (Terminal) и проверьте установку:

```
python3.9 --version
```

   Вы должны увидеть версию Python 3.9.

5.1.2 Шаг 2: Установка Git

1. Перейдите на официальный сайт Git: `https://git-scm.com/downloads`.
2. Скачайте установочный файл Git для вашей операционной системы.
3. Запустите установочный файл и следуйте инструкциям установки с настройками по умолчанию.
4. После установки откройте командную строку и проверьте установку:

```
git --version
```

   Вы должны увидеть версию Git.

5.1.3 Шаг 3: Клонирование репозитория

1. Откройте командную строку.
2. Перейдите в директорию, куда хотите клонировать проект:

```
cd        //
```

3. Клонируйте репозиторий проекта:

```
git clone https://github.com/MansurYa/chat-server.git
```

4. Перейдите в директорию проекта:

```
cd chat-server
```

5.1.4 Шаг 4: Создание виртуального окружения и установка зависимостей

1. Создайте виртуальное окружение:

```
python -m venv venv
```

2. Активируйте виртуальное окружение:
   - На Windows:

```
venv\Scripts\activate
```

   - На Unix или MacOS:

```
source venv/bin/activate
```

3. Установите необходимые пакеты:

```
pip install -r requirements.txt
```

(Если файл `requirements.txt` отсутствует, убедитесь, что все необходимые библиотеки, такие как `asyncio` и `tkinter`,

установлены. Однако `asyncio` и `tkinter` входят в стандартную библиотеку Python.)

5.1.5 Шаг 5: Запуск сервера и клиента

1. Откройте два терминала.
2. В первом терминале запустите сервер:

```
python server.py
```

3. Во втором терминале запустите клиента:

```
python client.py
```

5.2 Инструкция по эксплуатации программы

После установки и настройки программы выполните следующие шаги для её использования:

5.2.1 Шаг 1: Запуск сервера

1. Убедитесь, что Python установлен и активировано виртуальное окружение (если используется).
2. Перейдите в директорию проекта:

```
cd chat-server
```

3. Запустите сервер:

```
python server.py
```

4. Откроется окно сервера с отображением подключённых клиентов, комнат и логов.

5.2.2 Шаг 2: Запуск клиента

1. На компьютере клиента установите Python и Git, если они ещё не установлены.

2. Клонируйте репозиторий, перейдите в папку проекта и установите зависимости (см. раздел 5.1).

3. Запустите клиента:

```
python client.py
```

4. При запуске клиент запросит ввод имени пользователя. Введите уникальное имя.

5. После подключения к серверу, клиентский интерфейс позволит отправлять сообщения, управлять комнатами и выполнять другие команды через кнопки или ввод в поле сообщений.

5.2.3 Шаг 3: Использование функций чата

1. **Отправка сообщений**:

   - Введите сообщение в поле ввода и нажмите ”Отправить”.

2. **Создание комнаты**:

   - Нажмите кнопку ”Создать комнату”, введите название новой комнаты.

3. **Присоединение к комнате**:

   - Нажмите кнопку ”Присоединиться к комнате” и введите название существующей комнаты.

4. **Покидание комнаты**:

   - Нажмите кнопку ”Покинуть комнату” и подтвердите действие.

5. **Отправка личного сообщения**:

- Нажмите кнопку ”Отправить личное сообщение”, введите имя пользователя и сообщение.

6. **Просмотр списка комнат**:
   - Нажмите кнопку ”Список комнат”.
7. **Просмотр списка пользователей**:
   - Нажмите кнопку ”Список пользователей”.
8. **Загрузка файлов**:
   - Используйте команду `/upload <filename>` в поле сообщений для загрузки файла на сервер.

5.2.4 Шаг 4: Завершение работы

1. Для завершения работы сервера закройте окно сервера или нажмите `Ctrl+C` в терминале.
2. Для завершения работы клиента закройте окно клиента или нажмите `Ctrl+C` в терминале.

6 Контрольный пример

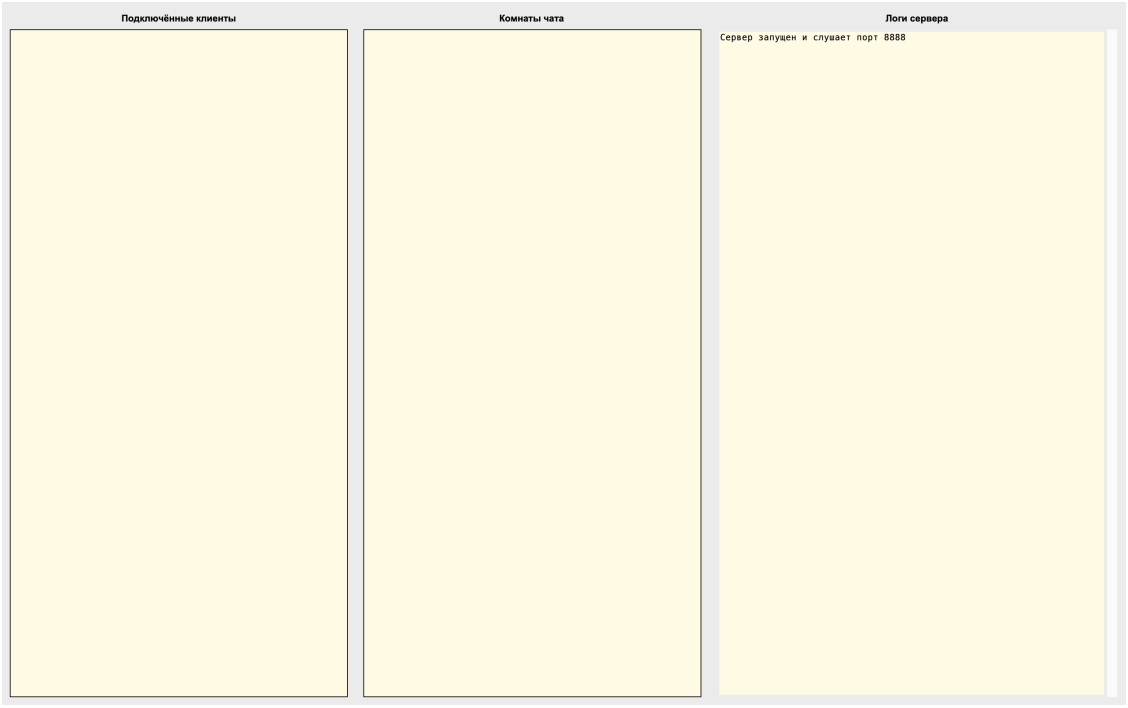

рис. 1 Запуск сервера

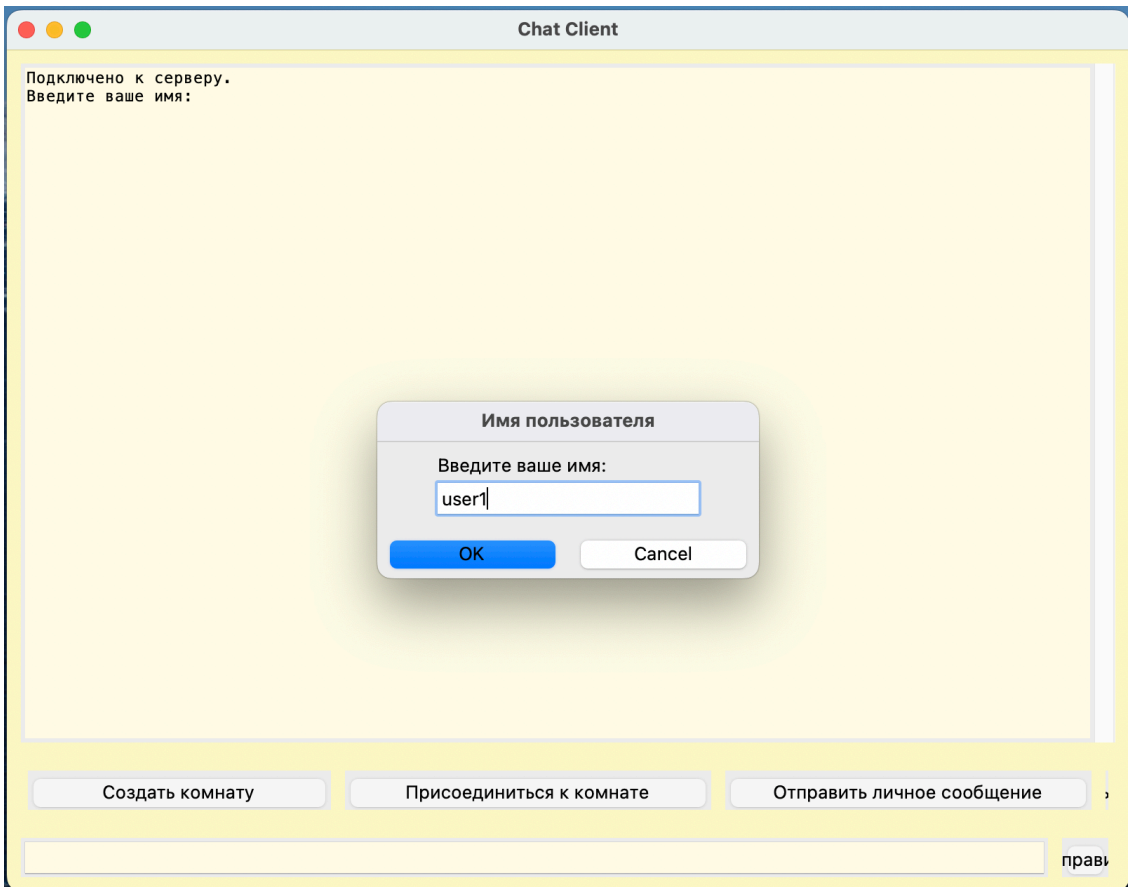

рис. 2 Запуск клиента

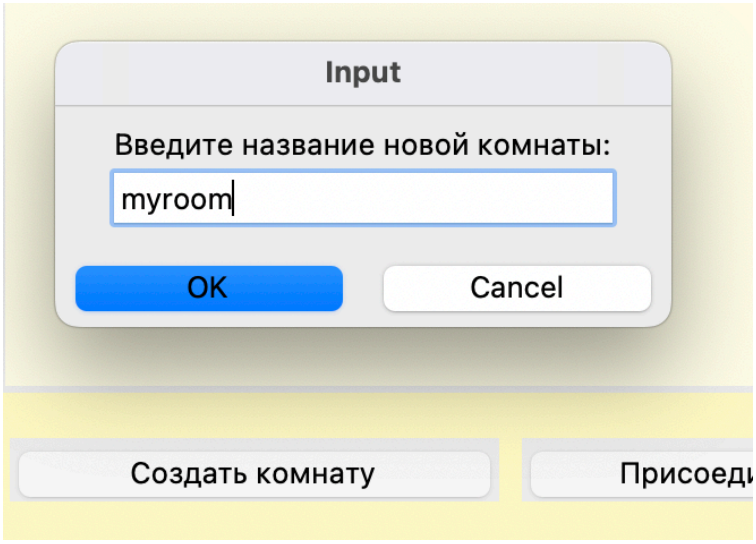

рис. 3 Создание комнаты

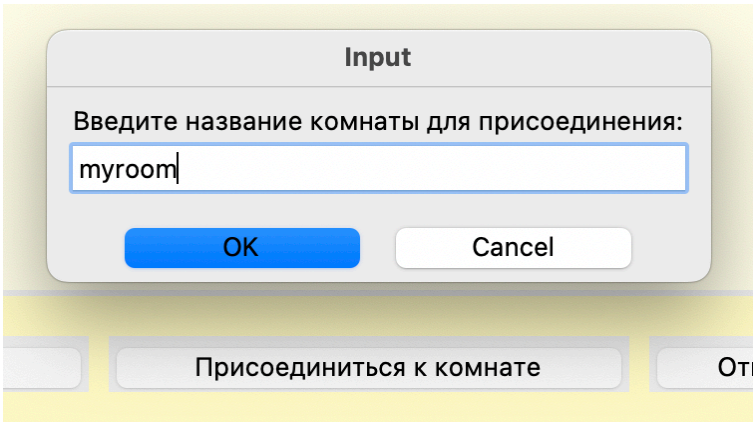

рис. 4 Присоединение к созданной комнате

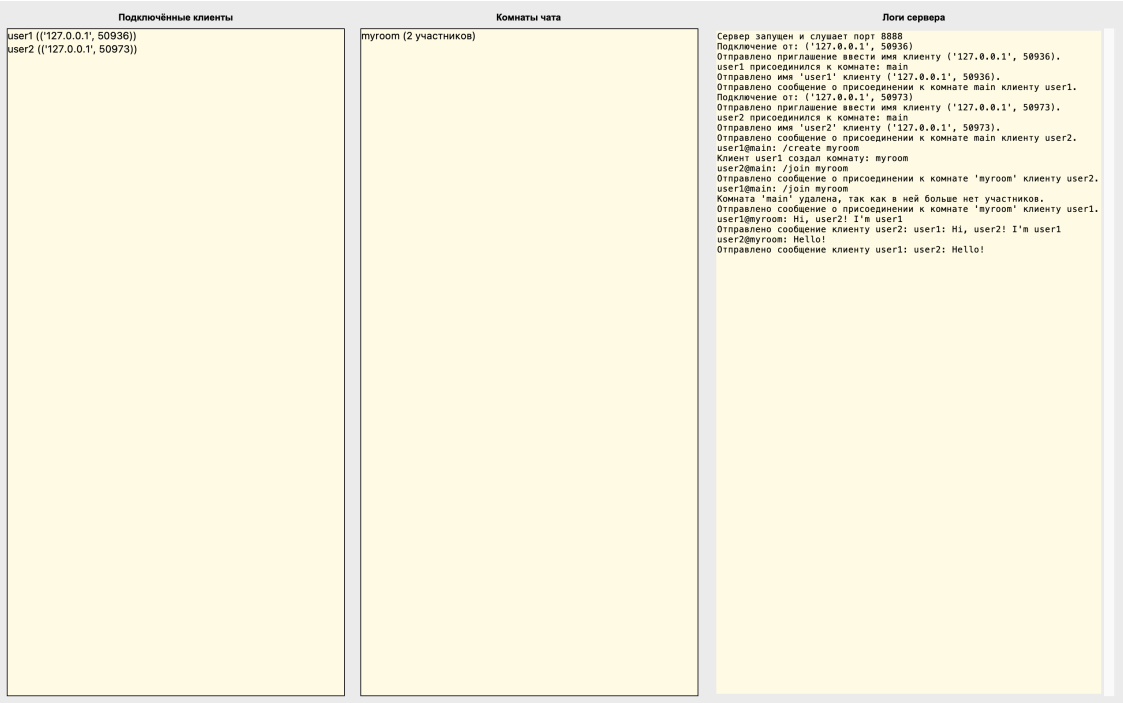

рис. 5 Вывод на части сервера

```
Сервер запущен и слушает порт 8888
Подключение от: ('127.0.0.1', 50936)
Отправлено приглашение ввести имя клиенту ('127.0.0.1', 50936).
user1 присоединился к комнате: main
Отправлено имя 'user1' клиенту ('127.0.0.1', 50936).
Отправлено сообщение о присоединении к комнате main клиенту user1.
Подключение от: ('127.0.0.1', 50973)
Отправлено приглашение ввести имя клиенту ('127.0.0.1', 50973).
user2 присоединился к комнате: main
Отправлено имя 'user2' клиенту ('127.0.0.1', 50973).
Отправлено сообщение о присоединении к комнате main клиенту user2.
user1@main: /create myroom
Клиент user1 создал комнату: myroom
user2@main: /join myroom
Отправлено сообщение о присоединении к комнате 'myroom' клиенту user2.
user1@main: /join myroom
Комната 'main' удалена, так как в ней больше нет участников.
Отправлено сообщение о присоединении к комнате 'myroom' клиенту user1.
user1@myroom: Hi, user2! I'm user1
Отправлено сообщение клиенту user2: user1: Hi, user2! I'm user1
user2@myroom: Hello!
Отправлено сообщение клиенту user1: user2: Hello!
```

рис. 6 Логи сервера

7 Выводы по работе

Разработаны асинхронные сервер и клиент для чат-приложения на Python с использованием библиотеки `asyncio`. Программа поддерживает одновременное подключение множества пользователей, управление чат-комнатами, обмен сообщениями в реальном времени, личные сообщения и загрузку файлов. Реализована стабильная обработка ошибок и проведено тестирование функциональности системы.

8 Полезные ссылки

Ссылка на репозиторий проекта: `https://github.com/MansurYa/chat-server.git`