**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Функциональное программирование»**

**на тему «****Разработка асинхронного чат-сервера»**

**Студент гр. 23Б16-пу**

**Аникин Р. С.**

**Преподаватель**

**Киямов Ж. У.**

**Санкт-Петербург**

**2024 г.**

**Оглавление**

[**Цель работы** 2](#_Toc185238724)

[**Описание задачи (формализация задачи)** 2](#_Toc185238725)

[**Теоретическая часть** 4](#_Toc185238726)

[**Описание программы** 6](#_Toc185238727)

[**Рекомендации пользователю** 8](#_Toc185238728)

[**Рекомендации программисту** 8](#_Toc185238729)

[**Исходный код программы** 9](#_Toc185238730)

[**Контрольный пример** 9](#_Toc185238731)

[**Вывод** 14](#_Toc185238732)

[**Источники** 15](#_Toc185238733)

# **Цель работы**

Целью данной работы является разработка чат-клиента и серверной части с использованием асинхронного ввода-вывода на основе библиотеки **asyncio** для языка Python. В ходе работы будет реализована возможность обмена сообщениями между пользователями, подключающимися к серверу, а также создание функционала для отображения активных пользователей в комнате. Основное внимание уделяется созданию простого, но эффективного взаимодействия между клиентом и сервером в реальном времени.

# **Описание задачи (формализация задачи)**

Необходимо разработать асинхронный чат-сервер и клиент, позволяющие нескольким пользователям общаться в реальном времени. Серверная часть реализуется с использованием библиотеки asyncio для эффективной обработки нескольких подключений одновременно без блокировки основного потока.

**Функциональность сервера:**

1. **Обработка подключений:**
   * Сервер принимает подключения от клиентов и создаёт асинхронные задачи для их обработки.
   * Каждый клиент передаёт своё имя и выбирает чат-комнату для общения.
2. **Чат-комнаты:**
   * Сервер поддерживает несколько комнат, в которых пользователи могут общаться.
   * При подключении или отключении клиента сервер обновляет список участников комнаты и уведомляет остальных пользователей.
3. **Передача сообщений:**
   * Сообщения, отправленные клиентом, передаются всем пользователям комнаты.
   * Сервер обрабатывает сообщения и отправляет их только в соответствующую комнату.
4. **Обновление списка пользователей:**
   * Сервер отслеживает изменения в комнатах (подключение или выход участников) и отправляет актуальный список активных пользователей всем клиентам.

**Функциональность клиента:**

1. **Подключение к серверу:**
   * Клиент подключается по IP-адресу и порту, передаёт своё имя и выбирает чат-комнату.
2. **Отправка сообщений:**
   * Пользователь может отправлять сообщения в комнату. Каждое сообщение содержит имя отправителя и временную метку.
3. **Получение сообщений:**
   * Клиент получает сообщения от сервера и отображает их в интерфейсе.
4. **Отображение активных пользователей:**
   * Клиент видит список участников комнаты, который обновляется в реальном времени.

Для обеспечения работы программы используется asyncio.Queue, организующая обработку сообщений и предотвращающая блокировку потоков. Этот подход позволяет эффективно обрабатывать несколько подключений одновременно.

# **Теоретическая часть**

#### **Асинхронное программирование:**

Асинхронный подход позволяет выполнять несколько задач параллельно, избегая блокировок. В Python библиотека asyncio обеспечивает:

* **Корутины** — функции с async def, которые могут приостанавливать выполнение с помощью await.
* **Цикл событий** — управляет выполнением корутин и распределяет задачи.
* **Асинхронные задачи** — корутины, которые можно запускать параллельно с помощью asyncio.create\_task().

Асинхронность особенно полезна в сетевых приложениях, так как позволяет обрабатывать множество подключений одновременно, не дожидаясь завершения каждой задачи.

#### **Протокол TCP и сокеты:**

Для взаимодействия между сервером и клиентом используется протокол TCP, обеспечивающий:

* Надёжную доставку данных.
* Сохранение порядка передачи.

Сервер создаёт асинхронные сокеты для обработки подключений, а клиенты устанавливают соединения, отправляют и получают данные через эти сокеты.

#### **Многозадачность с asyncio:**

Асинхронная обработка обеспечивает выполнение нескольких задач параллельно:

* Сервер принимает сообщения, пересылает их в комнаты и обновляет список пользователей.
* Клиент одновременно обрабатывает ввод данных и принимает сообщения от сервера.

Этот подход обеспечивает плавное взаимодействие между пользователями, минимизируя задержки.

#### **Работа с Tkinter:**

Для создания графического интерфейса клиента используется Tkinter:

* Цикл событий asyncio запускается в отдельном потоке, чтобы интерфейс оставался отзывчивым.
* Асинхронные функции обрабатывают отправку и приём сообщений.
* Интерфейс обновляется в реальном времени, чтобы отображать актуальные сообщения и список пользователей.

Использование комбинации asyncio и Tkinter позволяет создать удобный и функциональный чат с поддержкой нескольких пользователей.

**Основные шаги программы**

1. **Запуск серверной части**:
   * Сервер запускается и начинает прослушивать входящие подключения на определенном IP-адресе и порту (в данном случае, 127.0.0.1 и порт 8000).
   * Когда подключается новый клиент, сервер принимает соединение, регистрирует пользователя в указанной комнате и добавляет его в список активных пользователей этой комнаты.
   * Сервер передает всем клиентам в комнате уведомление о присоединении нового участника и обновляет список активных пользователей.
2. **Запуск клиентской части**:
   * Клиентская программа запускается, подключается к серверу, открывается интерфейс на tkinter
   * Пользователь вводит данные.
   * После успешного подключения пользователь получает доступ к чату, а сервер регистрирует его в стандартной комнате.
3. **Обмен сообщениями**:
   * После подключения клиент запускает корутину для получения сообщений от сервера. Все сообщения от других пользователей отображаются в текстовом виджете.
   * Пользователь может вводить сообщения в текстовое поле. При нажатии на кнопку «Отправить» или нажатии Enter сообщение отправляется на сервер.
   * Сервер получает сообщение, пересылает его всем активным пользователям в комнате, и они получают это сообщение в своих окнах чата.
4. **Отключение клиента**:
   * При выходе пользователя клиентская программа завершает соединение с сервером.
5. **Закрытие сервера**:

Сервер продолжает работать, ожидая подключения новых клиентов, пока его не остановят вручную.

# **Описание программы**

Программная реализация написана на языке Python 3.10 с использованием библиотек asyncio, tkinter, threading.

Таблица 1. Описание программы server1.py

|  |  |  |
| --- | --- | --- |
| **Название функции** | **Описание функции** | **Возвращаемое значение** |
| update\_widget() | Обновление интерфейса | Не возвращает |
| handle\_client\_connection () | Установление соединения с клиентом и приём-передача сообщений | Не возвращает |
| join\_room() | Добавление пользователя в комнату | Не возвращает |
| create\_room() | Создание новой комнаты | Не возвращает |
| leave\_room() | Удаление пользователя из комнаты | Не возвращает |
| show\_current\_chat() | Показ пользователю названия текущей комнаты | Не возвращает |
| broadcast\_message() | Передача сообщения всем пользователям, находящимся в этой комнате | Не возвращает |
| send\_private\_message() | Отправка личного сообщения | Не возвращает |
| show\_help() | Отображение меню помощи | Не возвращает |
| start\_server() | Запуск сервера | Не возвращает |
| start\_server\_thread | Запуск сервера в многопоточном режиме | Не возвращает |

Таблица 1. Описание программы client1.py

|  |  |  |
| --- | --- | --- |
| **Название функции** | **Описание функции** | **Возвращаемое значение** |
| get\_messages() | Обработка получения сообщений от сервера | Не возвращает |
| send\_messages() | Отправка сообщений на сервер | Не возвращает |
| get\_input() | Получение ввода из поля ввода | User\_input (string) |
| on\_send\_button\_click() | Обработка нажатия на кнопку «Отправить» | Не возвращает |
| send\_message() | Отправка сообщения (вспомогательная функция) | Не возвращает |
| main() | Запуск клиента и подключение к серверу (вспомогательная) | Не возвращает |
| start\_async\_loop() | Запуск клиента в многопоточном асинхронном режиме | Не возвращает |
| on\_enter\_pressed() | Возможность отправки сообщения по нажатии Enter вместо соответствующей кнопки | Не возвращает |

## **Рекомендации пользователю**

* **Запуск сервера**:
* Сначала запустите **server.py** на устройстве.
* После запуска сервера отобразится сообщение: "Сервер запущен на 127.0.0.1:8000", что означает, что сервер готов принимать подключения.
* **Запуск клиента**:
* Затем запустите **client.py**. Откроется главное окно программы, которое сначала будет свернуто.
* **Подключение к серверу**:
* После запуска клиента автоматически откроется окно для ввода данных подключения.
* Введите: имя пользователя
* Для получения списка доступных команд введите /help
* Для отправки сообщений в комнату, в которой вы находитесь, просто введите текст сообщения и нажмите Enter/«Отправить»

## **Рекомендации программисту**

* Программа использует стандартные библиотеки Python **asyncio** и **tkinter**, которые устанавливаются автоматически с Python и не требуют дополнительной установки.
* Если вы работаете в окружении, где эти библиотеки отсутствуют (например, минимальная установка Python), убедитесь, что tkinter доступен. На Ubuntu и подобных системах его можно установить с помощью команды: sudo apt-get install python3-tk
* Убедитесь, что IP 127.0.0.1 и порт 8000 не заблокированы брандмауэром и доступны для локальных подключений, так как сервер и клиент взаимодействуют через локальный IP.
* Для работы на нескольких устройствах убедитесь, что в **server1.py** указан IP-адрес сервера, доступный в сети, а не 127.0.0.1

# **Исходный код программы**

[**https://github.com/RomanMuteki/func\_prog**](https://github.com/RomanMuteki/func_prog)

# **Контрольный пример**

1. Запуск сервера

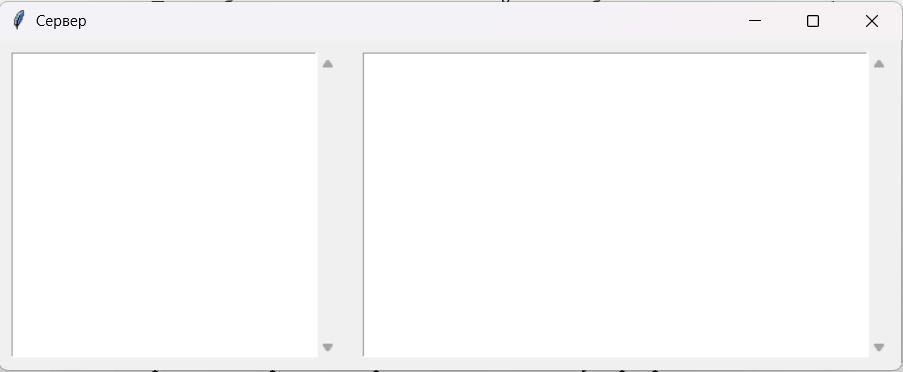


Рис.1. Запущенный сервер

1. Запуск клиента

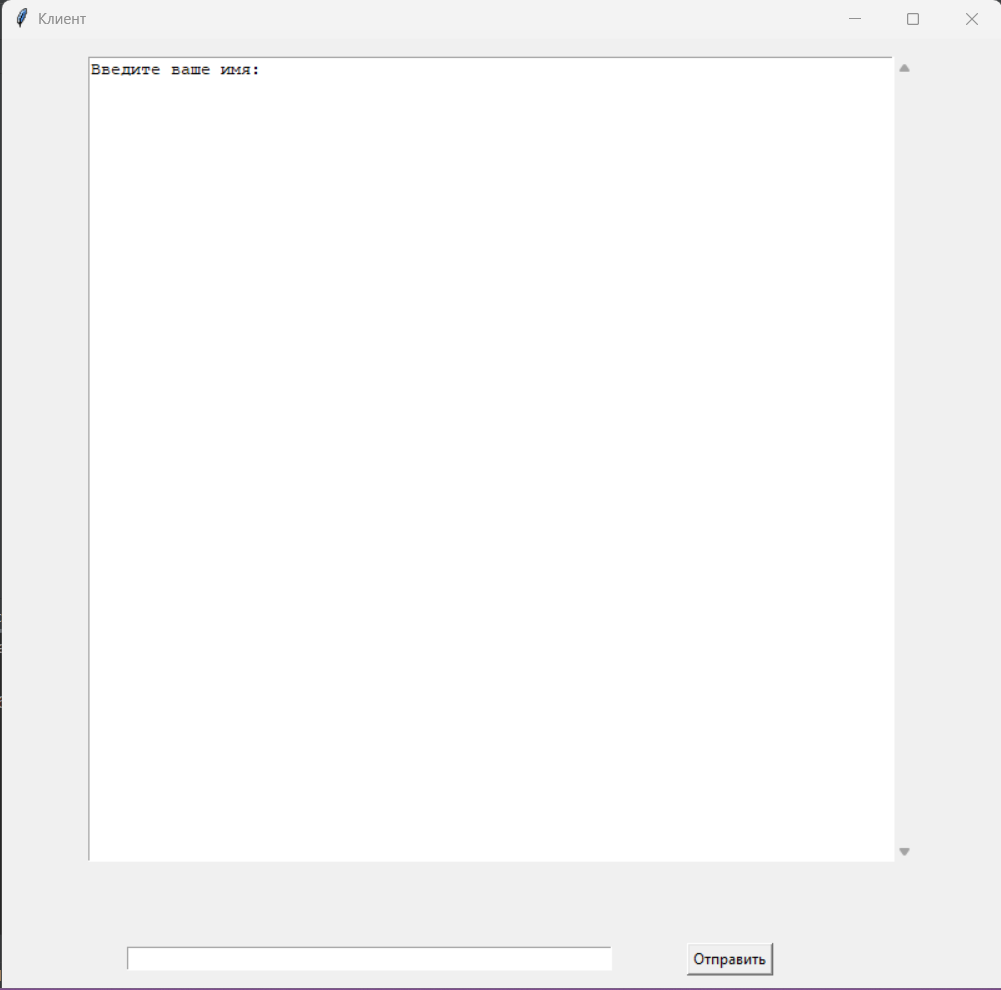


Рис. 2. Запущенный клиент

1. Ввод имени пользователя, создание новой комнаты и подключение к ней

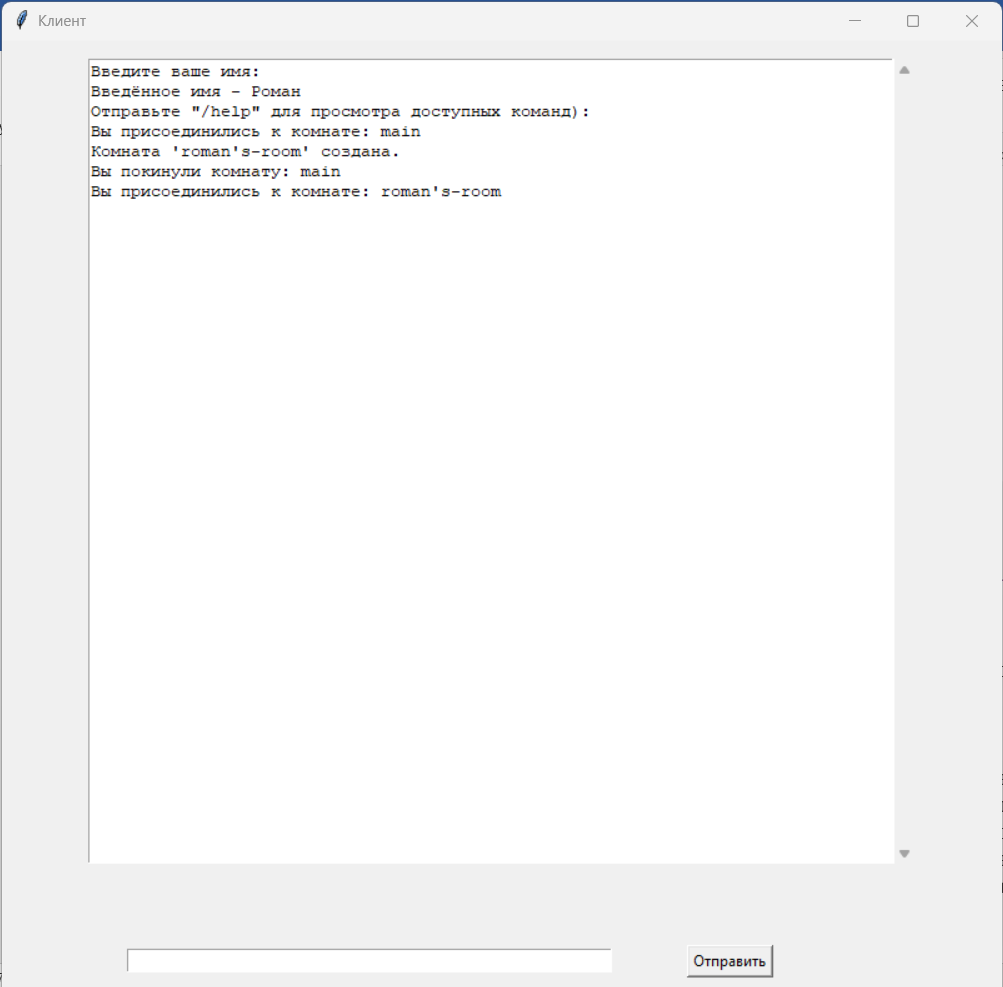


Рис. 3. Вход на сервер, создание новой комнаты и подключение к ней

1. Отправка и получение сообщения в комнате

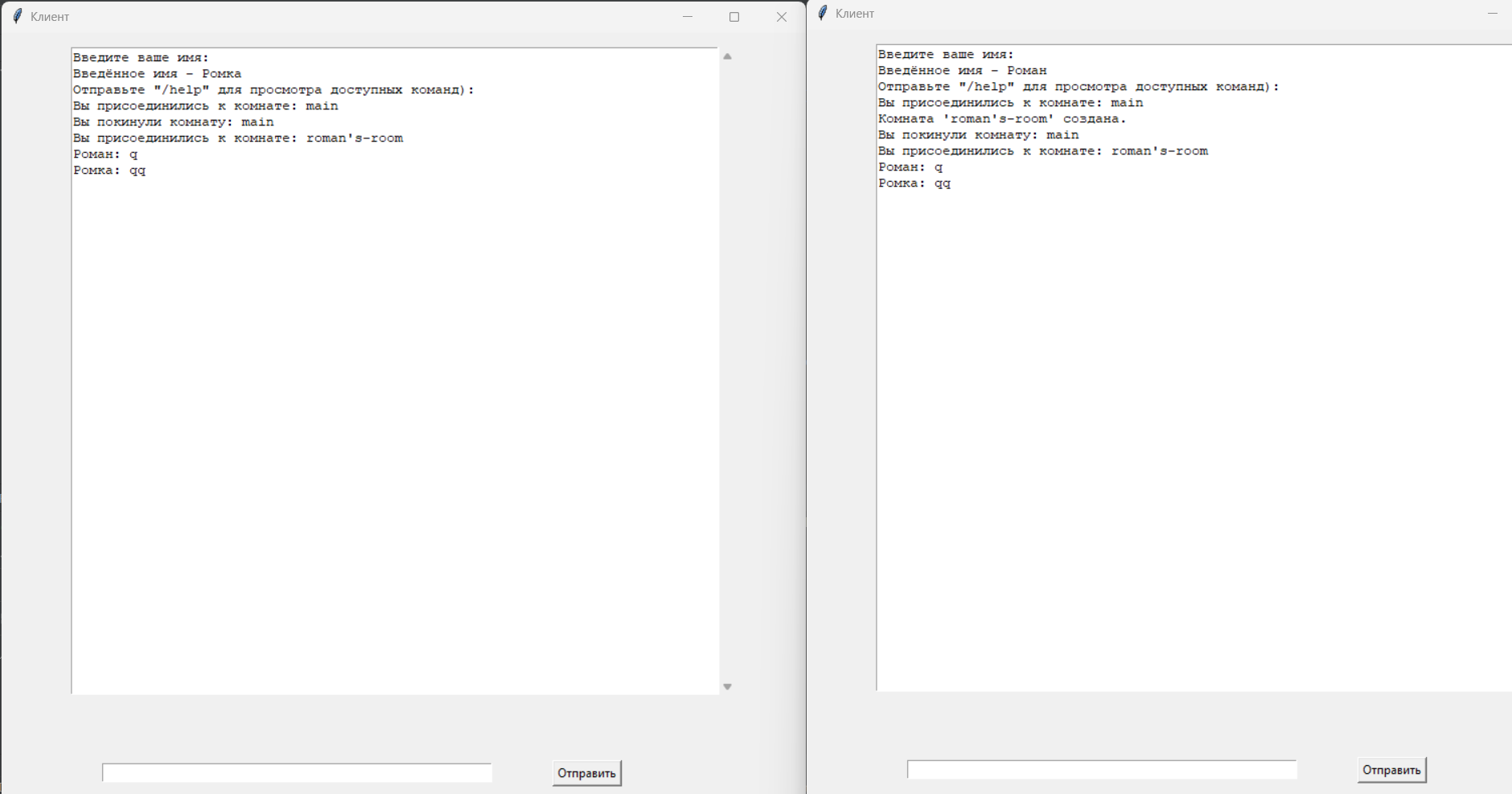


Рис. 4. Отправка и получение комнатных сообщений

1. Отправка и получение личного сообщения

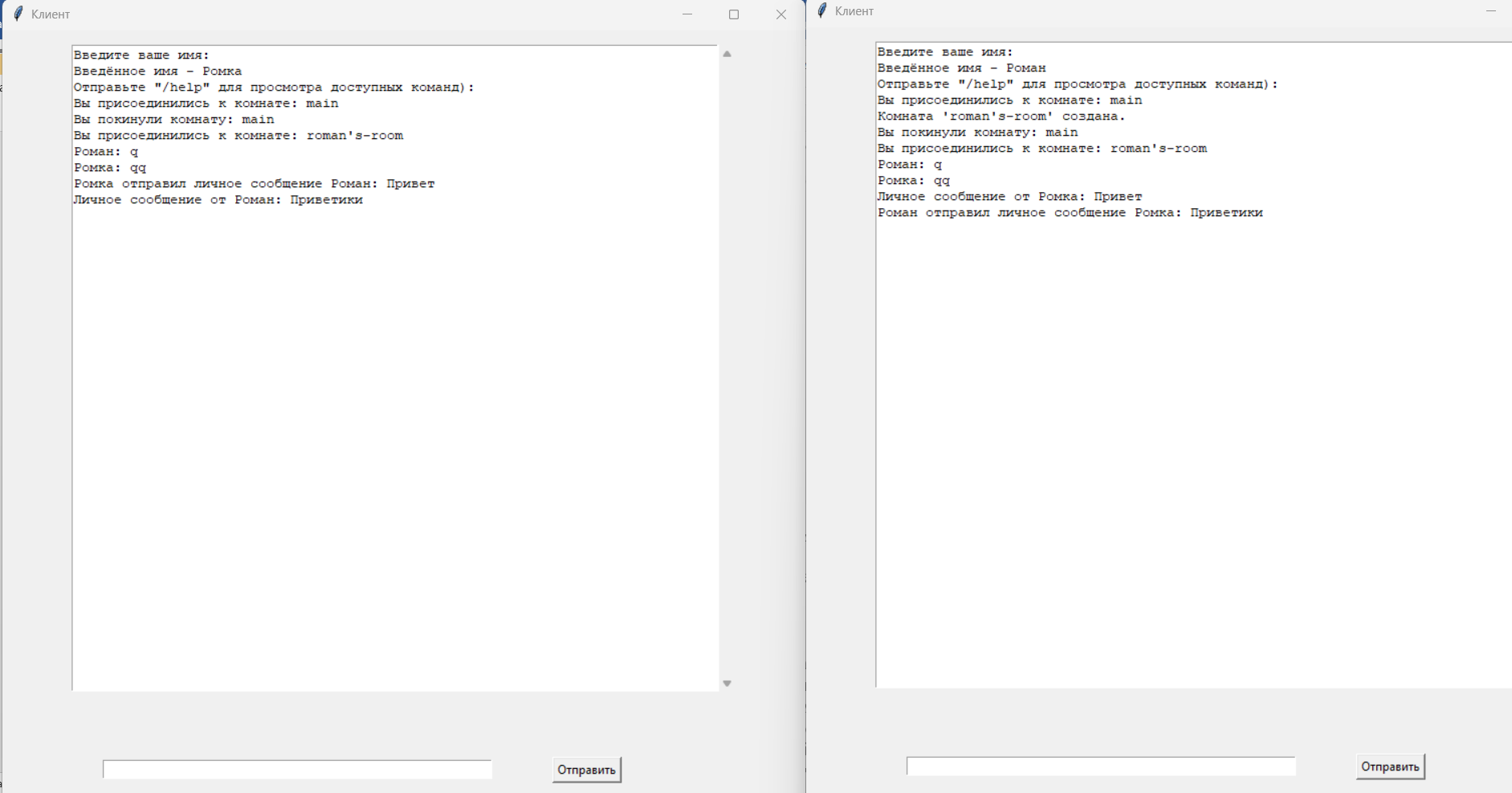


Рис. 5. Отправка и получение личного сообщения

1. Запрос списка доступных команд

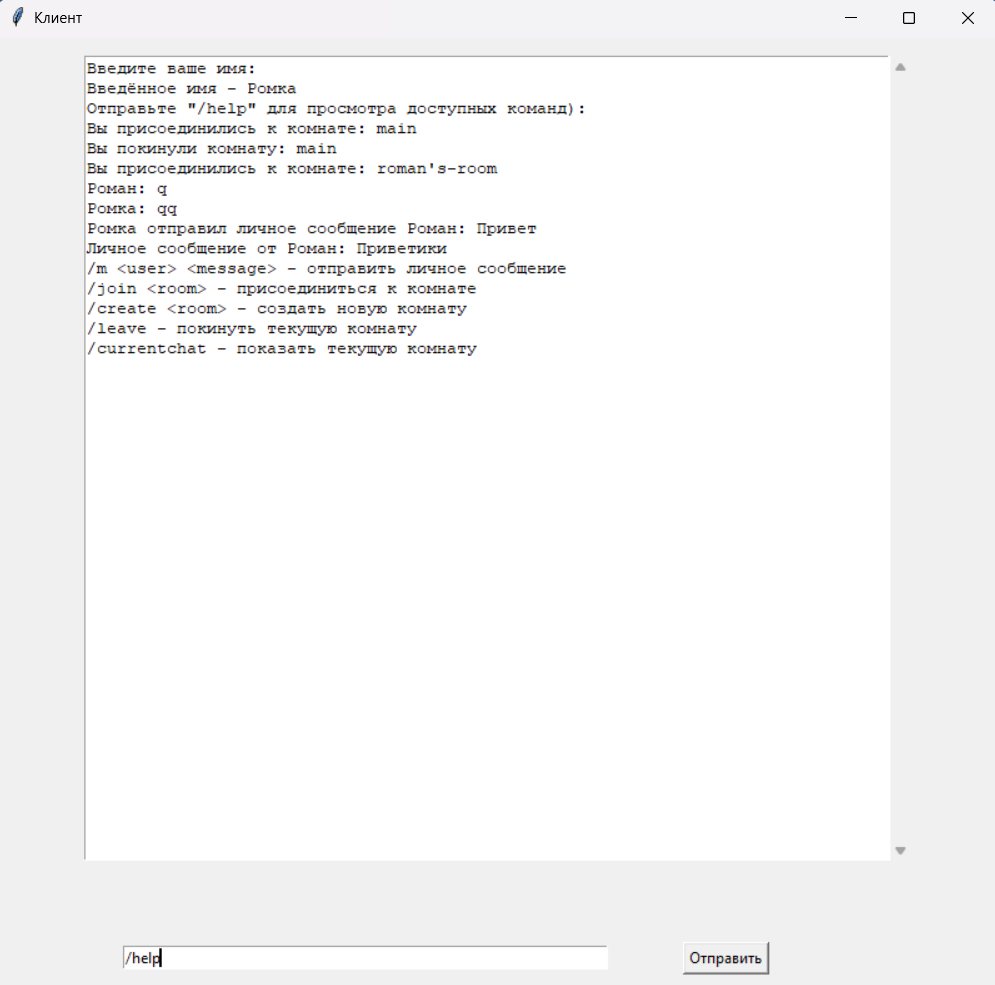


Рис. 6. Получение списка доступных команд посредством /help

# **Вывод**

В ходе выполнения работы была разработана и реализована клиент-серверная программа на языке Python, использующая асинхронные операции для обработки запросов и передачи данных между сервером и клиентом через сокеты. Программа предоставляет простой графический интерфейс с использованием библиотеки tkinter, который позволяет пользователю взаимодействовать с сервером, отправляя запросы и получая ответы в реальном времени.

# **Источники**

* **asyncio**

[*https://docs.python.org/3/library/asyncio.html*](https://docs.python.org/3/library/asyncio.html)

*дата обращения: (21.11.2024)*