**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**

**Шарабарин М.С.**

**Преподаватель**

**Киямов Ж. У.**

**Санкт-Петербург**

**2024 г.**

Оглавление

1. [Цель работы 3](#_Toc179492837)
2. [Описание задачи (формализация задачи) 4](#_Toc179492838)
3. [Теоретическая часть 6](#_Toc179492839)

**[1.](#_Toc179492840)****[Обработка изображений](#_Toc179492840)** [6](#_Toc179492840)

**[2. Классификация космических объектов](#_Toc179492841)** [6](#_Toc179492841)

**[3.](#_Toc179492842)****[Параллельные вычисления](#_Toc179492842)** [7](#_Toc179492842)

**[4.](#_Toc179492843)****[Методы анализа объектов на изображениях](#_Toc179492843)** [8](#_Toc179492843)

**[5.](#_Toc179492844)****[Хранение и визуализация результатов](#_Toc179492844)** [8](#_Toc179492844)

1. [Основные шаги программы 9](#_Toc179492845)
2. [Описание программы 12](#_Toc179492846)
3. [Рекомендации пользователя 14](#_Toc179492847)

**[Примечания:](#_Toc179492848)** [14](#_Toc179492848)

1. [Рекомендации программиста 15](#_Toc179492849)
2. [Исходный код программы 15](#_Toc179492850)
3. [Контрольный пример 16](#_Toc179492851)
4. [Вывод 18](#_Toc179492852)

# Цель работы

Цель данной работы состоит в создании чат-клиента и серверной части с использованием асинхронного ввода-вывода на базе библиотеки asyncio для Python. В рамках проекта будет реализована возможность обмена сообщениями между пользователями, подключающимися к серверу, а также разработан функционал для отображения списка активных участников. Особое внимание уделяется обеспечению простого и эффективного взаимодействия между клиентом и сервером в режиме реального времени.

# Описание задачи (формализация задачи)

Задача проекта состоит в создании асинхронного чат-сервера и клиента, обеспечивающих обмен сообщениями между множеством пользователей в режиме реального времени. Для серверной части применяется библиотека asyncio, которая позволяет обрабатывать большое число одновременных соединений без блокировки основного потока.

Основные функции сервера:

1. **Управление соединениями:** Принимает подключения от клиентов, используя асинхронные задачи. При подключении клиент передает своё имя и выбирает чат-комнату для общения.
2. **Поддержка чат-комнат:** Обеспечивает работу нескольких комнат, куда может войти каждый клиент для общения с участниками данной комнаты. При выходе клиента или его отключении сервер обновляет список активных пользователей и уведомляет об этом остальных участников.
3. **Трансляция сообщений:** Все сообщения, отправленные пользователем, распределяются среди остальных участников выбранной комнаты. Сервер обрабатывает и перенаправляет их в соответствующую комнату.
4. **Обновление списка активных пользователей:** При каждом подключении или отключении клиента происходит обновление списка присутствующих в каждой комнате, после чего информация рассылается всем участникам.

Основные функции клиента:

1. **Установка соединения с сервером:** Клиент подключается к серверу, вводит своё имя и выбирает нужную чат-комнату.
2. **Отправка сообщений:** Клиент может отправлять сообщения в выбранную комнату, при этом каждое сообщение содержит имя отправителя и время отправки.
3. **Получение сообщений:** Клиент асинхронно принимает сообщения от сервера и выводит их на экран.
4. **Отображение активных пользователей:** Клиент показывает список пользователей, находящихся в данной чат-комнате в данный момент.

Для организации передачи данных применяется асинхронная обработка с использованием очередей asyncio.Queue, что позволяет упорядочивать сообщения и избегать блокировки приложения. Это обеспечивает эффективное взаимодействие между сервером и клиентами при одновременной работе множества подключений без снижения производительности системы.

# Теоретическая часть

### **1. Асинхронное программирование**

Асинхронное программирование – это метод, позволяющий выполнять задачи параллельно без блокировки основного потока. В Python для создания асинхронного кода используется библиотека **asyncio**, предоставляющая средства для обработки операций ввода-вывода, таких как чтение данных из сети, без ожидания завершения каждой задачи. Асинхронность особенно полезна в сетевом программировании, так как позволяет обрабатывать несколько подключений одновременно. Основные понятия **asyncio**:

* **Корутины**: функции, объявленные с помощью async def, могут выполняться параллельно с использованием await для приостановки выполнения.
* **Цикл событий**: центральный компонент asyncio, который управляет выполнением корутин и очередностью задач, распределяя ресурсы между ними.
* **Задачи**: корутины, которые управляются и исполняются циклом событий. Их можно запускать параллельно с помощью asyncio.create\_task().

### **2. Протокол TCP и сокеты**

Для обмена сообщениями между сервером и клиентом используется протокол **TCP** (Transmission Control Protocol). TCP обеспечивает надежную доставку данных, сохраняя порядок и контролируя целостность передачи. В данной работе:

* **Сервер** принимает подключения от клиентов и обрабатывает их с использованием асинхронных сокетов, чтобы избежать блокировки на ожидании данных от каждого клиента.
* **Клиенты** подключаются к серверу по IP-адресу и порту, создавая соединение для передачи данных. В каждом соединении клиент может отправлять и получать сообщения.

### **3. Многозадачность с использованием asyncio**

Для поддержания взаимодействия множества пользователей в реальном времени серверная часть должна обрабатывать несколько подключений одновременно. В **asyncio** используется механизм многозадачности:

* Сервер может выполнять разные задачи (например, прием сообщений, пересылка их другим клиентам, обновление списка активных пользователей) параллельно.
* На стороне клиента **asyncio** позволяет асинхронно обрабатывать ввод и вывод данных, чтобы пользователь мог отправлять сообщения и получать их в реальном времени.

### **4. Работа с интерфейсом на основе Tkinter**

Tkinter предоставляет простые возможности для создания GUI в Python, но не поддерживает асинхронные операции изначально. Для интеграции с **asyncio** используются следующие приемы:

* **Запуск цикла событий**: **asyncio**-цикл запускается в отдельном потоке, чтобы интерфейс оставался отзывчивым.
* **Асинхронные функции для работы с сетью**: отправка и получение сообщений выполняются в асинхронных корутинах, что позволяет не блокировать интерфейс и поддерживать плавность работы.
* **Обновление интерфейса**: данные, полученные от сервера, отображаются в окне чата и в списке активных пользователей, что позволяет пользователям видеть актуальные сообщения и статус участников чата.

В целом, комбинация **asyncio** и **tkinter** позволяет построить асинхронный чат, работающий с несколькими пользователями одновременно, с минимальными задержками и удобным интерфейсом.

# 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 8888).
   * 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, открывается интерфейс на tkinter, запрашивающий у пользователя IP-адрес сервера, имя пользователя и номер комнаты.
   * Пользователь вводит данные, и клиентская программа пытается подключиться к серверу.
   * После успешного подключения пользователь получает доступ к чату, а сервер регистрирует его в указанной комнате.
3. **Обмен сообщениями**:
   * После подключения клиент запускает корутину для получения сообщений от сервера. Все сообщения от других пользователей и обновления списка активных участников отображаются в текстовом виджете.
   * Пользователь может вводить сообщения в текстовое поле. При нажатии на кнопку "Send" 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 (нажатии кнопки "Disconnect") клиентская программа завершает соединение с сервером и отправляет уведомление о выходе из комнаты.
   * Сервер удаляет пользователя из списка активных, обновляет его для других пользователей и рассылает уведомление о том, что участник покинул комнату.
5. **Закрытие сервера**:
   * Сервер продолжает работать, ожидая подключения новых клиентов, пока его не остановят вручную.

# Описание программы

Программная реализация выполнена на языке Python 3.12.7 с использованием библиотек **asyncio** для организации асинхронной связи между клиентом и сервером, а также **tkinter** для создания графического интерфейса клиента. Серверная часть организована в виде асинхронного TCP-сервера, который обрабатывает подключения клиентов, управляет чат-комнатами, пересылает сообщения и обновляет список активных пользователей в комнатах. Клиентская часть реализует подключение к серверу и обеспечивает удобный интерфейс для отправки и получения сообщений в режиме реального времени.

Таблица 1. server.py

| **Функция** | **Описание** | **Параметры функции** | **Описание параметров** |
| --- | --- | --- | --- |
| write\_message | Записывает данные в поток соединения клиента и ждёт завершения передачи (сброса буфера). | writer, data | writer – объект для записи в соединение; data – данные (в виде байтов), которые необходимо отправить. |
| connect\_user | Обрабатывает подключение нового пользователя: получает его имя и группу, проверяет наличие дублирования логина, отправляет сигналы клиенту, регистрирует пользователя и оповещает группу о подключении. | reader, writer | reader – объект для чтения данных от клиента; writer – объект для записи данных клиенту. |
| handle\_chat\_client | Основная функция обслуживания клиента. После подключения пользователя (через connect\_user) непрерывно принимает сообщения, проверяет команду выхода, логгирует полученные сообщения и транслирует их в соответствующую группу, а по завершении работы инициирует отключение пользователя. | reader, writer | reader – объект для получения данных; writer – объект для отправки данных. |
| broadcaster | Осуществляет бесконечное ожидание сообщений из очереди, анализирует их (в том числе обрабатывает специальные команды, такие как @help и @list), формирует итоговое сообщение и отправляет его нужным получателям с учетом личных сообщений или группового чата. | Нет (работает с глобальными переменными queue и ALL\_USERS) | — |
| broadcast\_message | Добавляет сообщение вместе с автором в очередь сообщений для дальнейшей трансляции через функцию broadcaster. | message, author | message – текст сообщения; author – идентификатор (имя) отправителя, используется для определения группы получателей. |
| disconnect\_user | Закрывает соединение с пользователем, удаляет его из списка активных пользователей и оповещает группу об отключении. | name, group, writer | name – имя пользователя; group – название группы, к которой принадлежит пользователь; writer – объект соединения. |
| main | Запускает сервер: создаёт задачу для функции broadcaster, открывает сервер на заданном IP и порту и начинает прослушивание входящих соединений. | Нет | Функция не принимает параметров, настройки (IP, порт) заданы внутри функции. |

**Таблица 2. Функции клиентской части**

| **Функция** | **Описание** | **Параметры функции** | **Описание параметров** |
| --- | --- | --- | --- |
| click\_on\_submit\_button | Обработчик нажатия кнопки отправки сообщения, устанавливающий событие для отправки введённого сообщения. | Нет | — |
| quit\_from\_messanger | Отправляет на сервер команду "QUIT", очищает текстовое поле сообщений, устанавливает флаги выхода и инициирует переход к начальному экрану (фрейму входа). | enter, messages | enter – фрейм для входа; messages – виджет с отображением сообщений. |
| write\_messages | Асинхронно ожидает событие отправки (нажатие кнопки), получает текст из поля ввода, отправляет его на сервер и сбрасывает событие. Если установлен флаг выхода, завершает цикл отправки сообщений. | writer, message\_enter | writer – объект для записи данных в сеть; message\_enter – поле ввода, содержащее текст сообщения. |
| receive\_messages | Постоянно читает данные от сервера, декодирует полученные сообщения и обновляет виджет сообщений, выводя новые сообщения в чат с автопрокруткой. | reader, messages\_widget | reader – объект для чтения данных из сети; messages\_widget – текстовый виджет для отображения чата. |
| main | Основная функция клиента, которая:  • Проверяет корректность введённых логина и названия комнаты; • Устанавливает соединение с сервером; • Отправляет данные для регистрации; • Обрабатывает ответ сервера (сигнал о занятости логина или подтверждение подключения); • Запускает задачи приема и отправки сообщений. | messages, message\_enter, login, room, label\_room, enter, chat | messages – виджет для отображения сообщений; message\_enter – поле ввода для сообщений; login – имя пользователя; room – название комнаты; label\_room – метка для отображения выбранной комнаты; enter, chat – фреймы интерфейса (экран входа и чат). |
| raise\_frame | Асинхронно переключает видимый фрейм в интерфейсе tkinter, поднимая указанный фрейм на передний план. | frame | frame – объект фрейма tkinter, который необходимо сделать активным. |

# Рекомендации пользователя

* **Запуск сервера**:
* Сначала запустите **server.py** на устройстве.
* После запуска сервера отобразится сообщение: "Сервер запущен на 127.0.0.1:8888", 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:
  + **IP сервера**: оставьте 127.0.0.1 для подключения к локальному серверу.
  + **Имя пользователя**: уникальное имя для отображения в чате.
  + **Номер комнаты**: номер комнаты, к которой вы хотите подключиться (например, "1").
* Нажмите кнопку **Enter**.
* **Отправка сообщений**:
* Введите текст сообщения в поле внизу окна.
* Для отправки сообщения нажмите **Enter** или кнопку **Send**.
* **Просмотр активных пользователей**:
* Список активных пользователей комнаты отображается в верхней части окна.
* При присоединении и выходе участников список автоматически обновляется.
* **Отключение от сервера**:
* Чтобы выйти из чата и закрыть программу, нажмите кнопку **Disconnect** в верхней части окна.

# Рекомендации программиста

* **Установка необходимых библиотек**:
* 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:



* **Проверка версии Python**:
* Код протестирован и совместим с Python 3.7 и выше, но рекомендуется использовать Python 3.10 или новее для наилучшей поддержки **asyncio**.
* Убедитесь, что версия Python установлена корректно, запустив:



* **Проверка сети и порта**:
* Убедитесь, что IP 127.0.0.1 и порт 8888 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 **server.py** указан IP-адрес сервера, доступный в сети, а не 127.0.0.1.

# Исходный код программы

https://github.com/neuraCollab/uni/tree/main/functional\_prog/fp/

# Контрольный пример

1. Запуск сервера: для запуска сервера используйте файл **server.py.** Программа должна запустить сервер и вывести в терминал соответствующую надпись. (Рис. 1)



Рис. 1 Запуск сервера

2. Запуск клиента: при запуске клиента откроется окно регистрации в котором необходимо указать некоторые данные (IP, имя, номер чат-комнаты, личный чат или нет). (Рис.2)

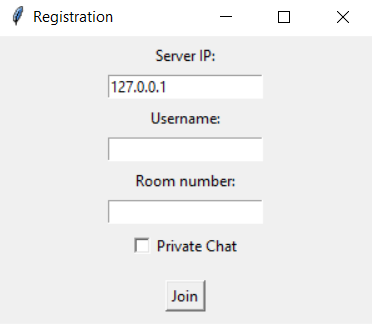


Рис.2 Окно регистрации

3. Вход в чат-комнату: после нажатия: “Enter” откроется выбранная чат-комната (приватная если выбрана галочка), обновится окно активных пользователей, и всех уведомят о том, что вы подключились. (Рис. 3) Приватная комната отличается от обычно тем, что туда могут зайти лишь два пользователя!



Рис. 3 Чат-комната

4. Отправка сообщения: в поле – “Enter message…” вы можете вводить что угодно и при нажатии на “Send” или Enter на клавиатуре отправлять другим пользователям в чат-комнате. (Рис. 4)

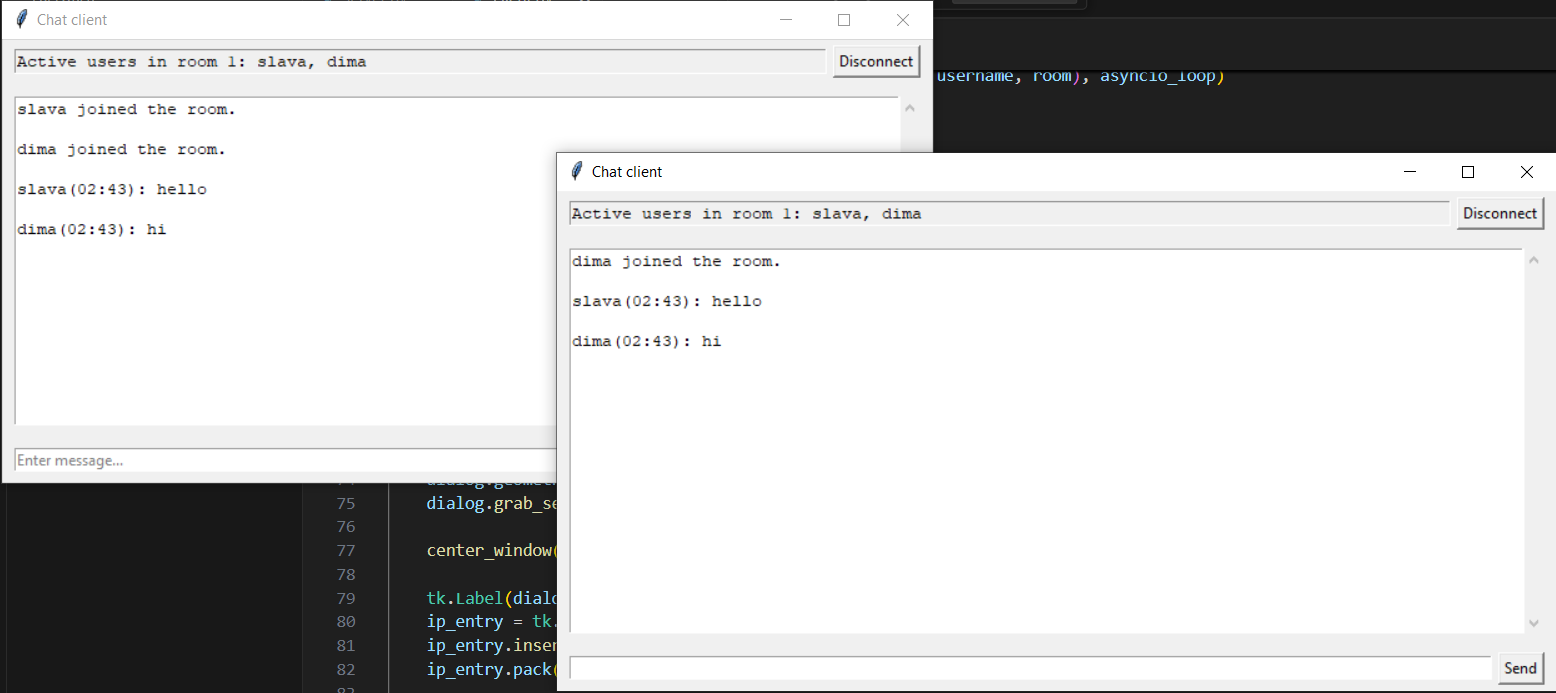


Рис. 4 Отправка сообщений

5. Отключение: при нажатии на – “Disconnect” вы отключитесь от комнаты и об этом уведомят оставшихся пользователей. (Рис. 5)

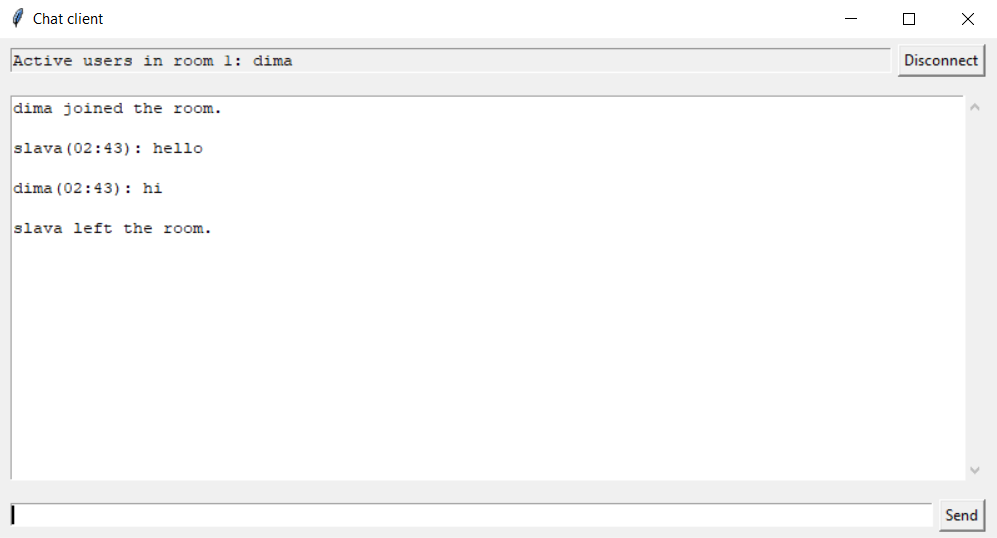


Рис 5. Отключение

# 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.

Серверная часть программы организована для прослушивания входящих соединений на определённом порту и обработки запросов от клиентов с помощью асинхронных функций. Клиентская часть реализует отправку запросов на сервер и вывод полученной информации в удобной форме.

Программа успешно продемонстрировала возможности асинхронного программирования в Python, а также показала, как можно интегрировать сетевые технологии с графическим интерфейсом. Результат работы программы является полезным примером для дальнейшего изучения и применения асинхронных сетевых приложений на Python, а также создания пользовательских интерфейсов для взаимодействия с такими приложениями.

В целом, выполненная работа позволяет сделать вывод, что Python, благодаря своей гибкости и большому количеству доступных библиотек, является отличным инструментом для разработки таких приложений, сочетающих в себе сетевое взаимодействие и графические интерфейсы.

# Источники

* **tkinter**

*[https://docs.python.org/3/library/tkinter.html](https://docs.python.org/3/library/tkinter.html%20)*

*дата обращения: (8.11.2024)*

* **asyncio**

*<https://docs.python.org/3/library/asyncio.html>*

*дата обращения: (11.11.2024)*