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. 23Б15-пу
Трофимов И.А.

Преподаватель
Киямов Ж. У.

Санкт-Петербург
2024 г.

Оглавление

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

Задача заключается в разработке асинхронного чат-сервера и клиента, которые позволяют нескольким пользователям обмениваться сообщениями в реальном времени. Для реализации серверной части используется асинхронная библиотека **asyncio**[1], что позволяет эффективно обрабатывать множество одновременных подключений без блокировки основного потока.

Сервер должен выполнять следующие функции:

1. **Обработка подключений**: Сервер принимает подключения от клиентов и обрабатывает их с использованием асинхронных задач. Каждый клиент, подключаясь к серверу, передает своё имя и выбирает чат-комнату для общения.
2. **Чат-комнаты**: Сервер поддерживает несколько чат-комнат. Каждый клиент может войти в одну из доступных комнат, где будет обмениваться сообщениями с другими пользователями этой комнаты. Если клиент покидает комнату или отключается, сервер должен обновить список активных пользователей в комнате и уведомить об этом других участников.

3. **Рассылка сообщений**: Все сообщения, отправленные одним пользователем, должны быть отправлены всем другим пользователям выбранной комнаты. Сервер должен эффективно обрабатывать эти сообщения и передавать их в нужную комнату.
4. **Обновление списка активных пользователей**: При каждом подключении или отключении клиента сервер должен обновлять список активных пользователей в каждой комнате и отправлять его всем участникам комнаты.

Клиент должен обеспечивать:

1. **Подключение к серверу**: Клиент подключается к серверу, вводит своё имя и выбирает чат-комнату для общения.
2. **Отправка сообщений**: Клиент может отправлять сообщения в выбранную комнату. Каждое сообщение будет содержать имя пользователя и время отправки.
3. **Получение сообщений**: Клиент должен асинхронно получать сообщения от сервера и отображать их в интерфейсе.
4. **Отображение списка активных пользователей**: Клиент отображает список пользователей, которые в данный момент находятся в выбранной комнате.

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**[1], 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**[1]:

- **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**[1] и **tkinter**[2] 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, 0.0.0.0 и порт 5002).
    - 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 имя пользователя и название 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 "Отправить"общение 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 "Выйти") 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**[1] для организации асинхронной связи между клиентом и сервером, а также **tkinter**[2] 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	Результат
handle_client	Обрабатывает подключение нового клиента, регистрирует его в указанной комнате, пересылает сообщения от клиента другим участникам комнаты.	None
main	Запускает сервер и начинает прослушивание подключений на заданном IP и порту	None

Таблица 2. client.py

Функция	Описание	Результат
ChatClient.__init__	Устанавливает подключение клиента к серверу, отправляет данные о пользователе и подключает его к комнате	None

connect	Подключает клиента к комнате	True, Error
send_message	Отправляет сообщение	None
receive_messages	Проверяет получено ли сообщение	None
disconnect	Отключает клиент от сервера	None
Chat_App._init__	Создает окно чата	None

connect_to_server	Подключает к комнате	Error
send_message	Отправляет сообщение	None
add_message	Показывает сообщение в интерфейсе	None
quit_chat	Завершает работу чата	None

Рекомендации пользователя

Для запуска программы убедитесь, что у вас установлен Python и необходимые библиотеки, такие как tkinter[2]. Код можно запустить в среде разработки или через командную строку, используя консоль для настройки параметров и генерации данных. Запуск программы производится через файл client.py.

Рекомендации программиста

Для поддержания актуальности и работоспособности программы используйте последние версии библиотек, особенно tkinter[2]. Применяйте практики надлежащего именования переменных и функций для улучшения читаемости кода.

Исходный код программы:

Jasur-Labs/fp/chat_GUI at main · hanglider/Jasur-Labs

Контрольный пример

Запустите server.py, client.py и введите свое имя и название комнаты в которую хотите войти (Рис. 1)

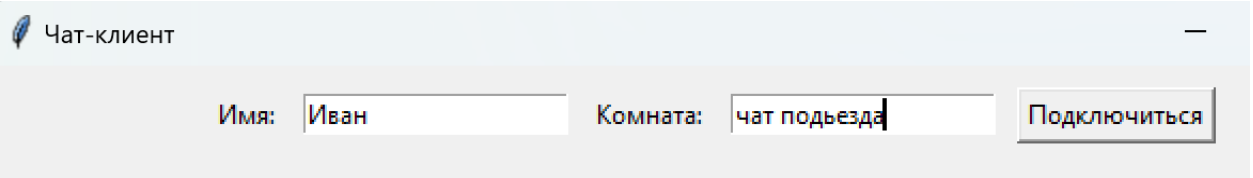

Рис. 1 Запуск сервера

Вход в комнату по нажатию на кнопку “Подключится”. Далее пишите сообщения (Рис.2)

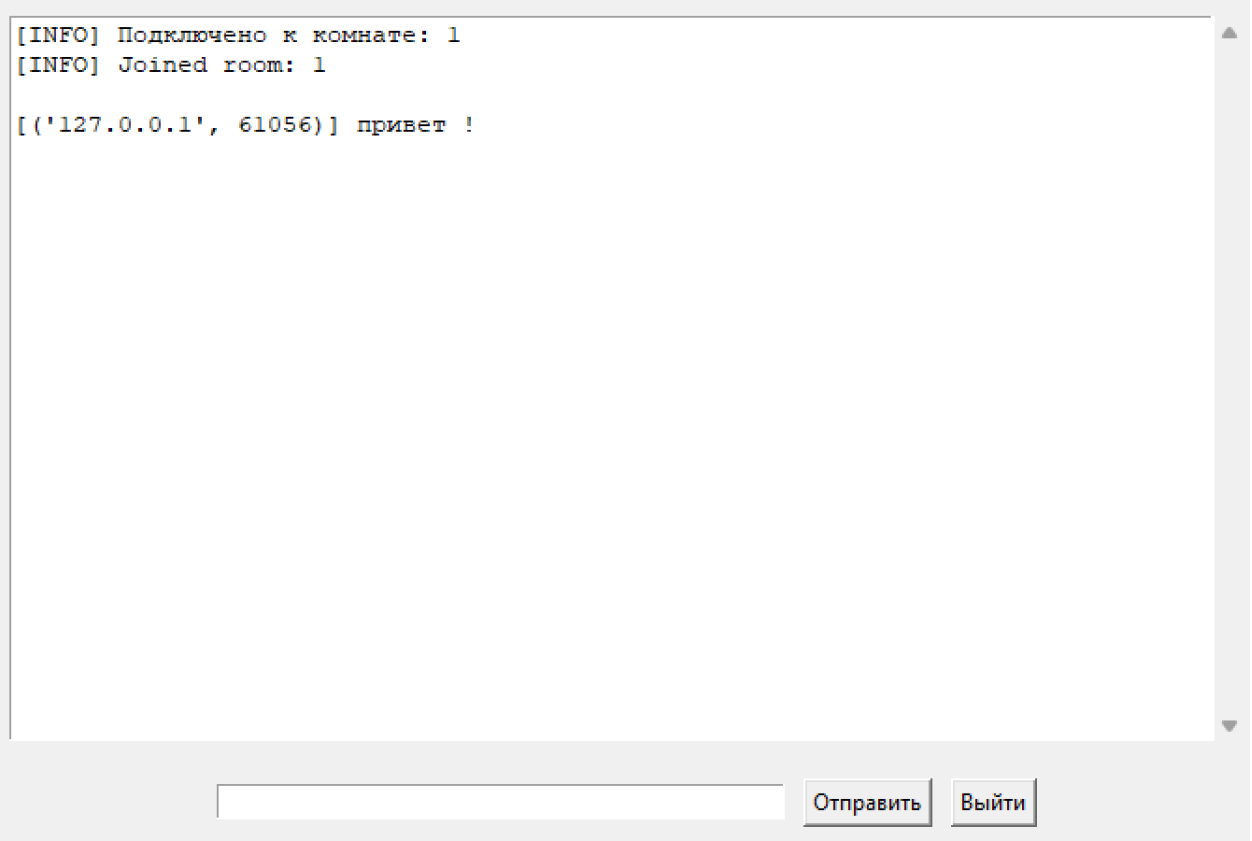

Рис.2 Окно регистрации

Для подключение к новой комнате введите новое название комнаты, при нажатии на – “Выйти” вы отключитесь от сервера и программа завершится.

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

1. **Tkinter** // *https://docs.python.org/3/library/tkinter.html (дата обращения: 8.11.2024)*
2. **asyncio** // *https://docs.python.org/3/library/asyncio.html (дата обращения: 8.11.2024)*