**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Функциональное программирование»**

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

Студент 23.Б16 Юдинцев С.А.

Преподаватель Киямов Ж.У.

**Санкт-Петербург**

**2024 г.**

**Содержание**

**1** [***Цель работы 3***](#_30j0zll)

**2** [***Теоритическая часть: 3***](#_1fob9te)

**3** [***Алгоритм метода:***](#_3znysh7) **4**

**4** [***Описание программы:***](#_2et92p0) **4**

**5** [***Рекомендации пользователю:***](#_tyjcwt) **9**

**6** [***Рекомендации для программиста:***](#_3dy6vkm) **9**

**7** [***Вывод:***](#_4d34og8) **10**

**8** [***Источники:***](#_2s8eyo1) **10**

1. **Цель работы:**

Разработать асинхронный чат-сервер и клиентское приложение, позволяющие пользователям взаимодействовать в реальном времени через текстовые сообщения с поддержкой команд и личных сообщений.

# Теоритическая часть:

**Асинхронное программирование** позволяет обрабатывать множество операций ввода-вывода параллельно, не блокируя выполнение других задач. В рамках разработанного чат-сервера и клиента применяются асинхронные сокеты и библиотеки asyncio, что обеспечивает высокую производительность и масштабируемость системы.

Взаимодействие между сервером и клиентом осуществляется по протоколу TCP. Клиент подключается к серверу, регистрируется в конкретной комнате и получает возможность отправлять и получать сообщения. Асинхронный сервер позволяет обрабатывать запросы от нескольких пользователей одновременно, что делает его эффективным решением для многопользовательского чата.

# Алгоритм метода:

1. **Инициализация сервера**

• Запуск сервера на 127.0.0.1:5004 с использованием asyncio.

• Ожидание входящих подключений.

2. **Регистрация клиента**

• Клиент отправляет команду /join.

• Сервер добавляет клиента в выбранную комнату и рассылает уведомления участникам.

3. **Отправка сообщений**

• Клиент вводит сообщение и отправляет его серверу.

• Сервер пересылает сообщение всем пользователям в комнате.

4. **Команды**

• /quit — выход из комнаты.

• /private <пользователь> <сообщение> — отправка личного сообщения.

5. **Обновление информации о комнатах**

• Сервер регулярно транслирует список активных комнат всем пользователям.

6. **Отключение клиента**

• При разрыве соединения сервер очищает данные о пользователе и уведомляет участников комнаты.

# Описание программы:

**Серверная часть**

1. **Инициализация и запуск сервера**

• Сервер запускается на указанном хосте и порте с использованием asyncio.start\_server.

• Все подключенные пользователи хранятся в словаре clients, где каждому имени пользователя сопоставляется его соединение и комната.

• Информация о комнатах (список участников) сохраняется в rooms.

2. **Подключение и регистрация клиентов**

• При подключении клиент отправляет команду /join <номер комнаты> <имя пользователя>.

• Сервер регистрирует пользователя и добавляет его в выбранную комнату, уведомляя остальных участников комнаты о присоединении нового пользователя.

3. **Обработка сообщений**

• Сообщения отправляются всем участникам комнаты с помощью функции send\_to\_room.

• Реализована поддержка личных сообщений с использованием команды /private <пользователь> <сообщение>.

4. **Отключение клиентов**

• При отправке команды /quit или разрыве соединения сервер удаляет пользователя из списка участников и уведомляет об этом остальных.

5. **Рассылка информации о комнатах**

• С помощью периодической задачи (каждые 3 секунды) сервер отправляет всем клиентам список активных комнат.

**Клиентская часть**

1. **Подключение к серверу**

• Пользователь вводит имя и номер комнаты в графическом интерфейсе (реализованном с использованием tkinter).

• Соединение устанавливается через asyncio.open\_connection.

2. **Отправка сообщений**

• Пользователь отправляет текстовые сообщения или команды через интерфейс. Сообщение передается серверу с использованием асинхронной функции send\_message.

3. **Получение сообщений**

• Клиент запускает фоновую задачу, которая непрерывно читает входящие данные от сервера и отображает их в окне чата.

4. **Отключение**

• При нажатии кнопки “Disconnect” отправляется команда /quit, после чего соединение закрывается.

В таблице ниже представлено описание функций

*Таблица 4.1. Основные функции программы*

| Название функции | Входные переменные | Описание функции |
| --- | --- | --- |
| ChatServer.start() | - | Запускает асинхронный сервер на указанном хосте и порте, ожидает подключений от клиентов. |
| ChatServer.handle\_client() | reader, writer | Обрабатывает подключение клиента, слушает команды и сообщения, управляет входом и выходом. |
| ChatServer.send\_to\_room() | room, message | Отправляет сообщение всем клиентам, находящимся в указанной комнате. |
| ChatServer.send\_private\_message() | sender, recipient, message | Отправляет личное сообщение указанному получателю. |
| ChatServer.send\_to\_client() | username, message | Отправляет сообщение конкретному пользователю. |
| ChatServer.disconnect\_client() | username | Удаляет клиента из комнаты и закрывает его соединение. |
| ChatServer.broadcast\_rooms() | - | Периодически отправляет список активных комнат всем подключенным пользователям. |
| ChatClient.connect() | username, room | Устанавливает соединение с сервером и присоединяет клиента к указанной комнате. |
| ChatClient.send\_message() | message | Отправляет сообщение серверу. |
| ChatClient.receive\_messages() | callback | Получает сообщения от сервера и передает их в функцию обратного вызова для отображения. |
| ChatClient.disconnect() | - | Отправляет серверу команду на отключение и закрывает соединение. |
| ChatApp.on\_connect() | - | Обрабатывает нажатие кнопки “Connect”, подключает клиента к серверу через интерфейс. |
| ChatApp.on\_disconnect() | - | Обрабатывает нажатие кнопки “Disconnect”, завершает соединение с сервером. |
| ChatApp.on\_send\_message() | - | Отправляет введенное пользователем сообщение на сервер. |
| ChatApp.add\_message() | message | Добавляет полученное сообщение в окно чата для отображения пользователю. |
| start\_asyncio\_loop() | - | Запускает цикл событий asyncio в отдельном потоке для клиентского приложения. |

В таблице ниже представлено описание функций переменные

*Таблица 4.2. Основные переменные программы*

| Название переменной | Описание |
| --- | --- |
| self.clients | Словарь для хранения подключенных клиентов, где ключ — имя пользователя, а значение — соединение. |
| self.rooms | Словарь для хранения комнат и их участников (список пользователей в каждой комнате). |
| self.online\_users | Множество имен пользователей, которые в данный момент подключены к серверу. |
| self.host | IP-адрес хоста, на котором запускается сервер. |
| self.port | Порт, на котором слушает сервер. |
| self.username | Имя пользователя клиента. |
| self.room | Название комнаты, к которой подключается клиент. |
| self.reader | Асинхронный поток для чтения данных с сервера. |
| self.writer | Асинхронный поток для записи данных на сервер. |
| self.connected | Логическая переменная, указывающая на состояние подключения клиента (True или False). |
| self.messages | Виджет ScrolledText, который отображает сообщения в интерфейсе клиента. |
| self.username\_entry | Поле ввода для имени пользователя в графическом интерфейсе. |
| self.room\_entry | Поле ввода для номера комнаты в графическом интерфейсе. |
| self.connect\_button | Кнопка для подключения к серверу. |
| self.send\_button | Кнопка для отправки сообщений серверу. |
| results | Примерная переменная для хранения статистики (если понадобится расширение функционала). |

1. **Рекомендации пользователю**

1. **Подключение**

• Введите корректное имя пользователя и номер комнаты.

2. **Основные команды**

• /quit — выход из комнаты.

• /private <имя> <сообщение> — отправить личное сообщение.

3. **Сообщения**

• Текстовые сообщения отображаются в чате в режиме реального времени.

4. **Отключение**

• Для выхода используйте кнопку “Disconnect”.

1. **Рекомендации программисту**

1. **Асинхронность**

• Использование asyncio обеспечивает неблокирующую обработку соединений и сообщений.

• Клиент запускает отдельный поток для обработки цикла событий asyncio.

2. **Обработка ошибок**

• При разрыве соединения или сбое предусмотрена корректная обработка ошибок.

3. **Расширение функционала**

• Добавить сохранение истории сообщений на сервере.

• Реализовать шифрование данных для повышения безопасности.

4. **Отладка**

• Логирование событий на сервере и клиенте помогает выявлять проблемы.

5. **Масштабирование**

• Для деплоя сервера можно использовать Docker-контейнер с пробросом портов:

# Вывод

В результате был разработан асинхронный чат-сервер и клиентское приложение. Сервер эффективно обрабатывает подключения множества пользователей и позволяет им взаимодействовать в реальном времени. Клиентская часть предоставляет простой и удобный интерфейс для отправки и получения сообщений. Реализованы базовые команды, такие как /quit и /private.

Использование асинхронного программирования позволило достичь высокой производительности и масштабируемости системы. В дальнейшем функционал можно расширить добавлением личных чатов, шифрования и аутентификации пользователей.

# Источники:

1. <https://itproger.com/news/razrabotka-onlayn-chata-na-python-i-socket-io>
2. <https://habr.com/ru/companies/skillfactory/articles/599599/>
3. <https://docs.python.org/3/library/asyncio.html>