**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**

**Абрахин Е.Д.**

**Преподаватель**

**Киямов Ж. У.**

**Санкт-Петербург**

**2024 г.**

Оглавление

1. [Цель работы 3](#_Toc180354448)
2. [Описание задачи (формализация задачи) 4](#_Toc180354449)
3. [Теоретическая часть 6](#_Toc180354450)
4. [Основные шаги программы 9](#_Toc180354451)
5. [Описание программы 12](#_Toc180354452)
6. [Рекомендации пользователя 14](#_Toc180354453)
7. [Рекомендации программиста 15](#_Toc180354454)
8. [Исходный код программы 15](#_Toc180354455)
9. [Контрольный пример 16](#_Toc180354456)
10. [Вывод 19](#_Toc180354457)
11. [Литература 20](#_Toc180354458)

# Цель работы

Целью данной работы является разработка асинхронного чат-сервера, который способен обслуживать множество клиентов одновременно и обеспечивает обмен сообщениями в режиме реального времени. Сервер должен быть эффективным, масштабируемым и устойчивым к ошибкам, обеспечивая комфортное взаимодействие пользователей в рамках чат-комнат.

# Описание задачи (формализация задачи)

Необходимо создать асинхронный чат-сервер с использованием библиотеки asyncio, который будет обрабатывать входящие подключения от клиентов и управлять обменом сообщениями. Основные требования к разработке включают:

1. Асинхронный сервер:

* Реализовать асинхронный сервер, который будет прослушивать определенный порт для входящих подключений.
* Использовать библиотеку asyncio для создания и управления асинхронными задачами, что позволит эффективно обрабатывать множество клиентов одновременно.

1. Асинхронные клиенты:

* Написать асинхронный клиентский скрипт, который может подключаться к серверу и отправлять сообщения.
* Клиенты должны иметь возможность подключаться к серверу, выбирать чат-комнаты и обмениваться сообщениями в реальном времени.

1. Многопоточность и управление задачами:

* Сервер должен обрабатывать одновременные подключения нескольких клиентов, используя asyncio-задачи.

1. Чат-комнаты:

* Реализовать поддержку чат-комнат, позволяя пользователям выбирать, в какую комнату они хотят войти для общения.
* Каждый клиент должен иметь возможность видеть и отправлять сообщения только тем участникам, которые находятся в одной комнате.

1. Рассылка сообщений:

* Сервер должен рассылать сообщения от одного клиента всем участникам текущей чат-комнаты.

1. Исключения и обработка ошибок:

* Обработать возможные исключения, чтобы гарантировать, что неверное поведение клиента не приведет к остановке работы сервера.
* Обеспечить стабильность и надежность работы сервера в условиях реальных пользователей.

# Теоретическая часть

Асинхронный чат-сервер является сетевым приложением, которое позволяет множеству клиентов обмениваться сообщениями в режиме реального времени, находясь в различных чат-комнатах. Разработка такого сервера включает в себя применение асинхронных технологий для обработки одновременных подключений большого числа пользователей, что существенно повышает эффективность и масштабируемость системы. Основной целью данной работы является создание асинхронного чат-сервера, который будет способен обслуживать множество клиентов одновременно, используя библиотеку asyncio в Python.

Асинхронное программирование

Асинхронное программирование представляет собой подход к разработке программного обеспечения, при котором операции выполняются без блокирования выполнения программы. Это особенно важно для приложений, которые работают с сетью и требуют высокой производительности при обработке большого количества одновременных запросов. В Python для этого используется библиотека asyncio, которая поддерживает асинхронные задачи (coroutines), позволяющие параллельно обрабатывать различные события, такие как прием сообщений от клиентов или рассылка сообщений в чат-комнаты.

Асинхронная модель программирования основана на цикле событий, где задачи ставятся в очередь, а выполнение операций происходит по мере их готовности. Это позволяет эффективно использовать ресурсы системы и снижает задержки, связанные с ожиданием ввода-вывода.

Архитектура чат-сервера

Разрабатываемый чат-сервер должен выполнять следующие функции:

* Сервер должен прослушивать определенный порт и принимать новые подключения клиентов. Для каждого клиента создается асинхронная задача, которая обрабатывает запросы от этого клиента.
* Сервер обрабатывает запросы от клиентов асинхронно, без блокирования работы других клиентов. Это позволяет одновременно обрабатывать большое количество подключений.
* Каждый клиент может выбрать чат-комнату, в которую он хочет войти. Сообщения, отправленные в одну комнату, видны только участникам этой комнаты, что обеспечивает разделение потоков сообщений.
* После того как клиент отправляет сообщение, сервер должен переслать это сообщение всем другим клиентам, находящимся в той же комнате.
* Для организации асинхронного обмена данными между клиентами используются очереди сообщений, которые обрабатываются асинхронно. Это обеспечивает быструю передачу данных и минимальные задержки.

Реализация сервера

Основная часть работы по реализации сервера заключается в настройке асинхронного цикла событий, который будет управлять подключениями клиентов и пересылкой сообщений.

Асинхронный сервер: Сервер запускается с помощью asyncio.start\_server, который создает асинхронные задачи для обработки каждого подключения клиента. Эти задачи управляют приемом и передачей данных между клиентами и сервером.

Обработка клиентских запросов: После подключения клиент посылает запросы на сервер, которые могут включать:

* Запрос на создание новой комнаты.
* Запрос на отправку сообщения в комнату.
* Запрос на получение списка существующих комнат.
* Запрос на получение списка сообщений из определенной комнаты.

Реализация клиента

Клиентское приложение также является асинхронным. Оно подключается к серверу, посылает запросы и получает ответы в асинхронном режиме. Основные компоненты клиента:

Подключение к серверу: Клиент подключается к серверу с использованием асинхронного сокета и передает запросы на создание или присоединение к чат-комнатам, а также на отправку сообщений.

Интерфейс пользователя: Используется библиотека tkinter для создания графического интерфейса, который позволяет пользователю выбирать комнаты, отправлять сообщения и просматривать историю чата.

Асинхронная обработка сообщений: Клиент получает новые сообщения от сервера асинхронно и обновляет текстовое окно чата в реальном времени.

Исключения и обработка ошибок

Один из важнейших аспектов разработки сетевых приложений — это обработка ошибок. В данной реализации необходимо обрабатывать возможные ошибки, такие как:

* Потеря соединения с сервером.
* Неверный формат данных, полученных от клиента.
* Ошибки при создании или удалении комнат.

Это обеспечивает стабильную работу сервера даже в условиях непредвиденных ситуаций.

# Основные шаги программы

Server.py

1. Запуск программы

* Программа запускает асинхронный сервер на определённом IP-адресе и порту (по умолчанию "0.0.0.0:5555").
* Сервер начинает прослушивать входящие соединения и обрабатывает запросы от клиентов.

1. Ожидание подключений клиентов

* Асинхронная функция handleClient вызывается для каждого нового клиента.
* Программа получает информацию о клиенте, сохраняет его соединение и начинает слушать запросы.

1. Обработка запросов от клиента При получении запроса:

* Сервер декодирует сообщение клиента с помощью библиотеки pickle.
* Обрабатываются следующие типы запросов:
  + Создание комнаты: Если клиент запросил создание комнаты, сервер добавляет новую комнату в список доступных.
  + Получение списка комнат: Сервер отправляет клиенту текущий список комнат.
  + Отправка сообщения: Если клиент отправил сообщение, оно сохраняется в соответствующей комнате и может быть доступно другим пользователям.
  + Получение сообщений: Сервер возвращает все сообщения, отправленные в указанной комнате.

1. Обработка сообщений в комнатах

* Для каждой комнаты сервер хранит список сообщений, состоящий из имен пользователей и текста их сообщений.
* Сообщения сохраняются для дальнейшего доступа клиентами, когда они запрашивают обновления.

1. Завершение сеанса

* При отключении клиента сервер закрывает соединение с ним.
* Сообщение об отключении выводится на сервере.

1. Ожидание новых подключений

* Сервер остаётся активным и ждёт новых подключений, продолжая обслуживать подключенных клиентов.

Client.py

1. Запуск программы

* Программа запускается с графическим интерфейсом, где пользователю предлагается ввести своё имя.
* После ввода имени программа устанавливает асинхронное соединение с сервером.

1. Подключение к серверу

* Клиент инициирует асинхронное соединение с сервером по адресу "localhost" и порту 5555.
* После успешного подключения активируются элементы интерфейса для создания или присоединения к комнатам.

1. Получение списка комнат

* Программа запрашивает список всех доступных чат-комнат с сервера и обновляет отображение комнат в интерфейсе.

1. Создание новой комнаты

* Пользователь может создать новую комнату, вводя её имя через диалоговое окно.
* Клиент отправляет запрос на сервер для создания комнаты.
* После успешного создания программа обновляет список комнат.

1. Присоединение к комнате

* Пользователь выбирает комнату из списка и присоединяется к ней.
* Клиент очищает окно сообщений и начинает асинхронно обновлять список сообщений, полученных в этой комнате.

1. Получение сообщений

* Клиент отправляет запрос на сервер для получения всех сообщений из текущей комнаты.
* При получении ответа сообщения выводятся в текстовом поле. Прокрутка окна сообщений всегда опускается до последнего сообщения, чтобы пользователь видел новые сообщения.

1. Отправка сообщений

* Пользователь вводит сообщение в текстовое поле и нажимает "Enter".
* Программа отправляет сообщение на сервер, которое затем транслируется другим клиентам, находящимся в той же комнате.

1. Обновление сообщений

* Клиент регулярно запрашивает новые сообщения в комнате, в которую он присоединился.
* Обновления происходят автоматически, обеспечивая отображение новых сообщений в реальном времени.

1. Завершение работы программы

* При закрытии программы соединение с сервером закрывается, и все асинхронные задачи останавливаются.

# Описание программы

Программная реализация асинхронного чат-сервера и клиента выполнена на языке Python 3.12.6 с использованием библиотек: asyncio, tkinter, и pickle. Программа состоит из двух модулей: server.py и client.py, которые взаимодействуют через асинхронные сокеты и поддерживают обмен сообщениями в режиме реального времени. Основной задачей программы является создание системы для обмена текстовыми сообщениями между пользователями в различных чат-комнатах с поддержкой многозадачности.

Таблица 1. Server.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| handleClient | Обрабатывает подключение клиента, принимает запросы, отвечает на них. Поддерживает создание комнат и рассылку сообщений. | None |
| main | Запускает сервер, прослушивающий определённый порт для подключения клиентов. | None |

Таблица 1. Client.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| connectToServer | |  | | --- | |  |   Устанавливает соединение с сервером, активирует элементы интерфейса для взаимодействия с пользователем. | None |
| fetchRooms | Запрашивает список доступных чат-комнат с сервера и обновляет отображение комнат в интерфейсе клиента. | None |
| sendChatMessage | Отправляет текстовое сообщение текущей комнате и обрабатывает ответ сервера. | None |
| updateMessages | Периодически запрашивает новые сообщения с сервера для отображения в интерфейсе в реальном времени. | None |
| createRoom | Создаёт новую комнату для чата, отправляет запрос на сервер и обновляет список комнат. | None |
| joinRoom | Позволяет пользователю присоединиться к выбранной комнате и начать получать сообщения. | None |

# Рекомендации пользователя

Программа представляет собой асинхронный чат-сервер и клиент, который позволяет пользователям обмениваться сообщениями в режиме реального времени в различных чат-комнатах.

1. Запуск сервера:

Перед началом работы запустите скрипт server.py. Сервер начнет прослушивать подключения клиентов на порту 5555. Убедитесь, что этот порт доступен на вашем компьютере.

1. Запуск клиента:

Для подключения к серверу запустите скрипт client.py. При запуске программы откроется графический интерфейс, где вам будет предложено ввести имя пользователя.

1. Создание и присоединение к чат-комнатам:

После подключения к серверу вы сможете создавать новые чат-комнаты или присоединяться к существующим. Для этого:

* Нажмите кнопку "Создать комнату", чтобы создать новую комнату. Введите название комнаты в появившемся диалоговом окне.
* Или выберите комнату из списка слева и нажмите "Присоединиться", чтобы войти в существующую чат-комнату.

1. Отправка сообщений:

После присоединения к комнате вы можете отправлять сообщения, вводя текст в поле ввода внизу окна и нажимая клавишу Enter. Ваше сообщение будет доставлено всем участникам комнаты.

1. Обновление сообщений:

Все новые сообщения будут автоматически появляться в текстовом поле чата. Программа будет обновлять список сообщений в реальном времени.

1. Завершение работы:

Для завершения работы клиента просто закройте окно программы. Сервер продолжит работу до тех пор, пока не будет закрыт вручную.

# Рекомендации программиста

Для успешного запуска программы убедитесь, что у вас установлен Python версии не ниже 3.12.0 и необходимые библиотеки, такие как asyncio, tkinter, и pickle.

Поддерживайте актуальные версии используемых библиотек и Python для обеспечения работоспособности программы на современных системах. Следуйте передовым практикам разработки, уделяя внимание четкому именованию переменных и функций, чтобы улучшить читаемость и понимание кода.

Регулярно проводите тестирование программы на различных входных данных, чтобы убедиться в ее надежности и корректности работы. Рассмотрите возможность добавления логирования для отслеживания процесса выполнения и отладки. Используйте комментарии для пояснения сложных участков кода и логики программы.

# Исходный код программы

<https://github.com/FasterXaos/Functional_programming>

# Контрольный пример

1. Запуск программы

Для начала запустите Server.py, появится консоль (Рис. 1), означающая, что сервер успешно запущен. После запустите один или несколько клиентов, которые попросят ввести имя пользователя для авторизации (Рис. 2).

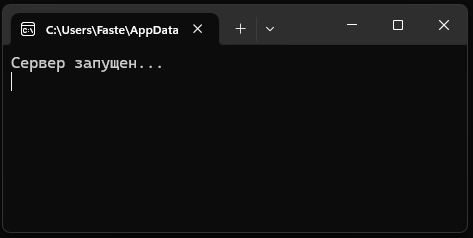


Рис 1. Пример вывода Server.py при запуске

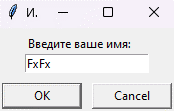


Рис 2. Пример ввода имени в Client.py при запуске

2. Создание и вход в комнату

Для создания новой комнаты нажимаем на “Создать комнату” и вводим ее название (Рис. 3). Для того, чтобы войти в комнату нужно выбрать ее в столбце слева и нажать на “Присоединиться”, после чего статус “Текущая комната” справа внизу изменится (Рис. 4).

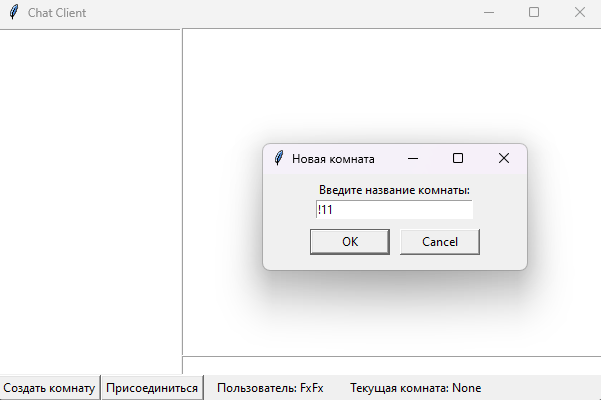


Рис 3 Пример создания комнаты

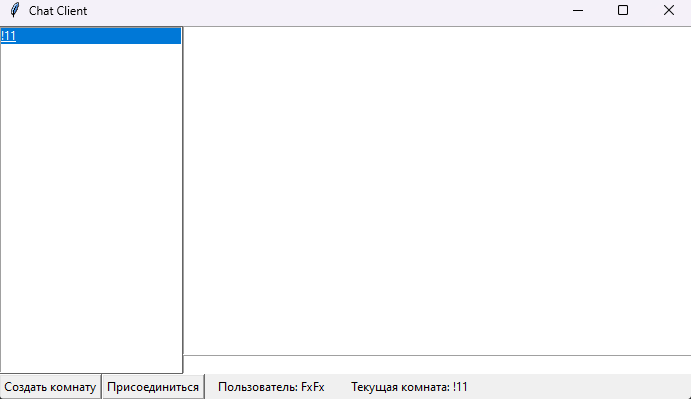


Рис 4. Пример входа в комнату

3. Отправка сообщений

Для создания отправки сообщения введите его в поле справа внизу и нажмите Enter (Рис. 5).

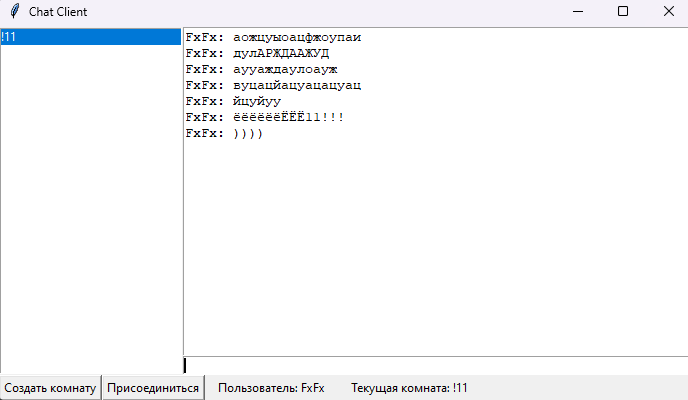


Рис 5. Пример отправки сообщений

# Вывод

В ходе данной работы был разработан асинхронный чат-сервер, способный обслуживать множество клиентов одновременно и обеспечивать обмен сообщениями в реальном времени. Основной целью работы было создание высокопроизводительной системы для организации многопользовательских чатов с поддержкой нескольких комнат.

В ходе работы была реализована возможность параллельной обработки подключений клиентов с использованием библиотеки asyncio, что позволило эффективно управлять асинхронными событиями, не блокируя основную работу сервера. Были реализованы такие функции, как создание чат-комнат, рассылка сообщений между участниками комнаты и автоматическое обновление списка сообщений.

Программа была написана на языке Python с использованием таких библиотек, как asyncio для асинхронного программирования и tkinter для создания графического интерфейса клиента. Эта архитектура позволила обеспечить масштабируемость приложения и удобное взаимодействие с пользователем.

# Литература

1. asyncio — Asynchronous I/O // docs.python.org URL: https://docs.python.org/3/library/asyncio.html (дата обращения: 20.10.2024).
2. tkinter — Python interface to Tcl/Tk // docs.python.org URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 20.10.2024).
3. pickle — Python object serialization // docs.python.org URL: https://docs.python.org/3/library/pickle.html (дата обращения: 20.10.2024).