СЛАЙД 1

В качестве цели разработки программного обеспечения выбрано создание платформы для сравнения производительности и анализа преимуществ и недостатков относительно друг друга многопоточного и многопоточно-асинхронного серверов.

**Задачи:**

1. Изучение библиотеки BOOST для того, чтобы применить его технологии для логгирования и использовать его асинхронный движок.
2. Реализация изучаемых многопоточной и многопоточно-асинхронной архитектур.
3. Реализация клиентского приложения для иллюстрации работоспособности системы реальному пользователю и имитатора клиентской нагрузки для проверки производительности серверов.
4. Проведение тестирования с логгированием ключевых параметров
5. Анализ полученных данных с графическим отображением результатов.

СЛАЙД 2

Из основных технологий, примененных в проекте, стоит отметить BOOST, QT, Docker, Grafana.

Следует отдельно сказать:

Разработка велась с использованием среды разработки VS Code.

Операционная система – Ubuntu Linux.

Все необходимые библиотеки и инструменты были установлены с использованием – менеджера пакетов Conan.

СЛАЙД 3

На схеме многопоточного сервера показаны дополнительные потоки обработки, помимо основного, отвечающего исключительно за прием соединения.

1. Поток получения имен – регистрация пользователя в системе, затем пользователь передается в поток получения списка чатов.
2. Поток получения списка чатов, в нем есть возможность создать новый чат или войти в существующий, переходя к потоку чата.
3. Если пользователь заходит в пустой чат, то у него запускается отдельный собственный поток, который работает до того, как чат опустеет снова. Пользователь может отправить свое сообщение или покинуть чат, вернувшись в поток получения списка чатов.
4. Отдельный поток удаления пользователей удаляет данные пользователя из системы при его отключении. В него передаются отключенные пользователи из всех вышеописанных потоков.

СЛАЙД 4

Многопоточно-асинхронный сервер имеет более любопытную структуру

Класс сервера – основа работы системы. Он обрабатывает новое подключение, после чего создает новый объект корутины сессии. Также он хранит объект менеджера чатов, который обеспечивает необходимую синхронизацию части действий над чатами.

СЛАЙД 5

Менеджер чатов содержит данные пользователей в системе, объекты чатов.

Методы класса менеджера чатов представляют следующий функционал:

* Установка имени
* Возвращение актуального списка чатов
* Рассылка актуального списка чатов пользователям вне чатов
* Подключение пользователя к чату
* Создание чата
* Выход из чата
* Отправка сообщения
* Отключение от системы
* Определение пользователя

Набор методов класса чата состоят из:

* Добавление пользователя
* Отправка сообщения
* Проверка наличия пользователя в чате
* Удаление пользователя из чата

Методы этих классов могут показаться схожими местами, однако в менеджере чатов встроена защита от неправильного вызова методов и уведомление об этом.

Методы класса сервера включают в себя:

* Старт
* Получение умного указателя на себя
* Прием соединения
* Обработка полученного соединения
* Запуск сессии

СЛАЙД 6-8

Клиентское приложение предусматривает работу с клиентом с учетом всех возможных его состояний: установка имени / список чатов / нахождение в чате

СЛАЙД 9

Просто описать схему

СЛАЙД 10

Тестирующая система опирается на использование контейнеров:

Контейнера сервера, контейнера генератора нагрузки, Node Exporter, Prometheus, Grafana.

СЛАЙД 11

Следует отметить какие устройства были использованы для тестирования.

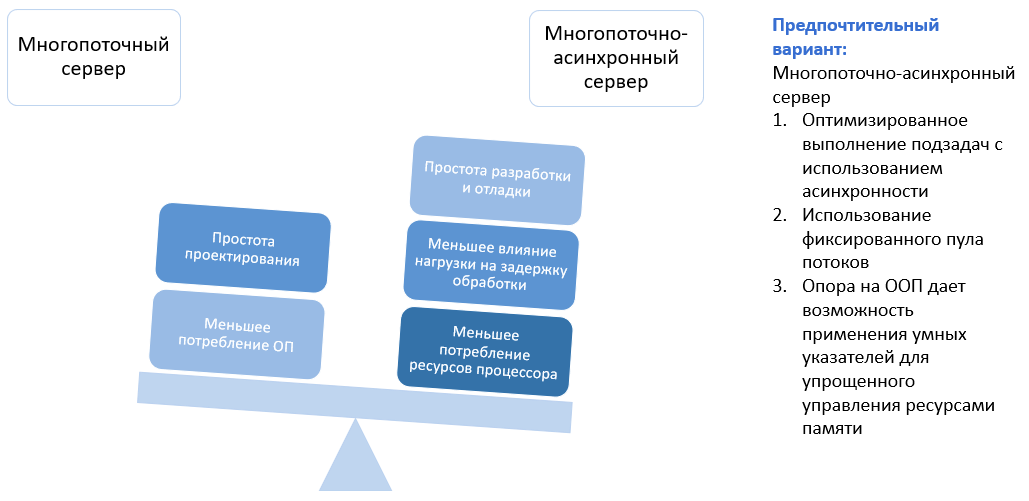
Оно проводилось в 3 этапа: 2/10/100 клиентов. Клиент имитатора куда быстрее в подаче команд относительно реального пользователя, поэтому для генерации трафика их нужно куда меньше.

Анализируются же золотые метрики сервера – **задержка**, **процент** использования ресурсов **процессора** и **процент** использования ресурсов **ОП**

СЛАЙДЫ 12-18

Пояснить логику Heatmap и указать на разницу распределения задержек.

СЛАЙД 19



СЛАЙД 20

Заключение – по слайду

Была создана платформа для сравнения производительности и анализа преимуществ и недостатков многопоточной и многопоточно-асинхронной серверных архитектур относительно друг друга.

После практического освоения технологий BOOST, были созданы реализации многопоточной и многопоточно-асинхронной реализаций серверов, клиентского приложения и имитатора клиентской нагрузки.

Проведено тестирование, в результате которого на основе полученных графически иллюстрированных данных проведен сравнительный анализ.

Получены полезные для разработчика сведения о различиях производительности двух архитектур.

Тестирующая система показала себя работоспособной и готовой к дальнейшему применению и удобной модификации под свои нужды разработчиками.