**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

# **Дисциплина:** Веб программирование

Отчет

Лабораторная работа №1

Выполнил:

Никифоров Арсен

Группа: К3343d

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

**Ход работы**

**Лаб1**

**Задание 1**

Реализовать клиентскую и серверную часть приложения. Клиент отсылает серверу

сообщение «Hello, server». Сообщение должно отразиться на стороне сервера.

Сервер в ответ отсылает клиенту сообщение «Hello, client». Сообщение должно

отобразиться у клиента.

Обязательно использовать библиотеку socket

Реализовать с помощью протокола UDP

**Решение:**

Создал 2 отдельных файла для клиента и сервера. Сервер прослушивает клиента и отвечает на приветствие.

Сервер

**import** socket

# Создаем сокет для UDP

server\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_DGRAM**)**

# Привязываем сокет к адресу и порту

server\_address **=** **(**'localhost'**,** 12345**)**

server\_socket**.**bind**(**server\_address**)**

**print(**"Сервер запущен и слушает на порту 12345..."**)**

**while** **True:**

# Принимаем данные от клиента и его адрес

#сервер получает сообщение от клиента и сохраняет это сообщение в переменной data

data**,** client\_address **=** server\_socket**.**recvfrom**(**1024**)** #это команда для сервера, а 1024 количество байт короче

# Выводим полученные данные от клиента

**print(**f"Получено от клиента ({client\_address}): {data**.**decode**()**}"**)**

# Отправляем ответ клиенту

response **=** "Hello, client"

server\_socket**.**sendto**(**response**.**encode**(),** client\_address**)**

клиент

**import** socket

# Создаем сокет для UDP

client\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_DGRAM**)**

# Адрес и порт сервера

server\_address **=** **(**'localhost'**,** 12345**)**

# Сообщение для отправки серверу

message **=** "Hello, server"

# Отправляем сообщение серверу

client\_socket**.**sendto**(**message**.**encode**(),** server\_address**)**

# Принимаем ответ от сервера

data**,** \_ **=** client\_socket**.**recvfrom**(**1024**)**

# Выводим полученный ответ от сервера

**print(**f"Получено от сервера: {data**.**decode**()**}"**)**

# Закрываем сокет клиента

client\_socket**.**close**()**

**запуск** через cmd:

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

**Задание 2**

Реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры, которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту. Варианты:

a. Теорема Пифагора

b. Решение квадратного уравнения.

c. Поиск площади трапеции.

d. Поиск площади параллелограмма.

Вариант выбирается в соответствии с порядковым номером в журнале. Пятый

студент получает вариант 1 и т.д.

Обязательно использовать библиотеку socket

Реализовать с помощью протокола TCP

**Решение:**

Первый пункт.

сервер

**import** socket

# Создаем сокет для сервера

server\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

# Указываем адрес и порт, на котором сервер будет слушать

server\_address **=** **(**'localhost'**,** 12345**)**

# Привязываем сервер к адресу и порту

server\_socket**.**bind**(**server\_address**)**

# Начинаем слушать клиентские запросы (максимум 5 одновременных подключений)

server\_socket**.**listen**(**5**)**

**print(**"Сервер запущен и ждет клиентов..."**)**

**while** **True:**

# Ожидаем подключение клиента

client\_socket**,** client\_address **=** server\_socket**.**accept**()**

**print(**f"Подключено клиентом {client\_address}"**)**

# Принимаем данные от клиента

data **=** client\_socket**.**recv**(**1024**)**# .recv() - это метод сокета, который используется для получения данных из сети

**print(**f"Получено от клиента: {data**.**decode**()**}"**)**

# Разбираем полученные данные (параметры для теоремы Пифагора)

params **=** data**.**decode**().**split**(**','**)**

a **=** **float(**params**[**0**])**

b **=** **float(**params**[**1**])**

# Вычисляем гипотенузу (результат теоремы Пифагора)

c **=** **(**a**\*\***2 **+** b**\*\***2**)\*\***0.5

# Отправляем результат обратно клиенту

response **=** f"Гипотенуза (c) равна: {c}"

client\_socket**.**send**(**response**.**encode**())**

# Закрываем соединение с клиентом

client\_socket**.**close**()**

клиент

**import** socket

# Создаем сокет для клиента

client\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

# Адрес и порт сервера

server\_address **=** **(**'localhost'**,** 12345**)**

# Подключаемся к серверу

client\_socket**.**connect**(**server\_address**)**

**print(**"Подключено к серверу..."**)**

# Вводим параметры для теоремы Пифагора (длины катетов) с клавиатуры

a **=** **float(input(**"Введите длину первого катета: "**))**

b **=** **float(input(**"Введите длину второго катета: "**))**

# Отправляем параметры серверу

message **=** f"{a},{b}"

client\_socket**.**send**(**message**.**encode**())**

# Получаем ответ от сервера

response **=** client\_socket**.**recv**(**1024**)**

**print(**f"Ответ от сервера: {response**.**decode**()**}"**)**

# Закрываем соединение с сервером

client\_socket**.**close**()**

Запуск:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт

Автоматически созданное описание

**Задание 3**

Реализовать серверную часть приложения. Клиент подключается к серверу. В ответ клиент получает http-сообщение, содержащее html-страницу, которую сервер подгружает из файла index.html.

Создал html-страницу с сообщением

<!DOCTYPE html>

<html>

<head>

<title>**MESSAGE LAB1**</title>

</head>

<body>

<h1>**NIKIFOROV ARSEN**</h1>

<p>**HTML-PAGE FROM MY SERVER**</p>

</body>

</html>

И сам код сервера

**import** socket

# Создаем сокет для сервера

server\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

# Указываем IP-адрес и порт, на котором сервер будет слушать

host **=** "127.0.0.1" # IP-адрес localhost

port **=** 8080 # Произвольный порт

# Привязываем сокет к указанному адресу и порту

server\_socket**.**bind**((**host**,** port**))**

# Начинаем прослушивать соединения

server\_socket**.**listen**(**1**)** # 1 - максимальное количество ожидающих соединений

**print(**f"Сервер слушает на http://{host}:{port}/"**)**

**while** **True:**

# Принимаем соединение

client\_socket**,** client\_address **=** server\_socket**.**accept**()**

# Читаем запрос от клиента

request **=** client\_socket**.**recv**(**1024**).**decode**()**

# Генерируем HTTP-ответ

response **=** """HTTP/1.1 200 OK

Content-Type: text/html

"""

# Открываем и читаем содержимое файла "index.html"

**with** **open(**"index.html"**,** "r"**)** **as** file**:**

response **+=** file**.**read**()**

# Отправляем ответ клиенту

client\_socket**.**sendall**(**response**.**encode**())**

# Закрываем соединение с клиентом

client\_socket**.**close**()**

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Клиентский запрос через порт 8080

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Задание 4**

Создать чат для двух или более пользователей. Использование многопоточности дает больше баллов. Необходимо использовать библиотеку threading. Полезные ссылки для ознакомления:

Использовать библиотеку threading. Для TCP необходимо создать потоки для подключения клиентов и отправки/приема сообщений между всеми пользователями на сервере. Важно сохранять информацию о пользователях для последующей отправки сообщений.

Сервер:

**import** threading

**import** socket

**class** **Server:**

**def** \_\_init\_\_**(**self**,** host **=** '127.0.0.1'**,** port **=** 55555**):**

self**.**host **=** host

self**.**port **=** port

self**.**server **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

self**.**server**.**bind**((**self**.**host**,** self**.**port**))**

self**.**server**.**listen**()**

self**.**clients **=** **[]** # Список подключенных клиентов

self**.**nicknames **=** **[]** # Список никнеймов клиентов

**def** broadcast**(**self**,** message**):**

# Рассылка сообщения всем клиентам

**for** client **in** self**.**clients**:**

client**.**send**(**message**)**

**def** handle**(**self**,** client**):**

**while** **True:**

**try:**

message **=** client**.**recv**(**1024**)** # Прием сообщения от клиента (максимум 1024 байта)

self**.**broadcast**(**message**)** # Рассылка сообщения всем клиентам

**except:**

# Обработка исключений, если клиент отключился

index **=** self**.**clients**.**index**(**client**)**

self**.**clients**.**remove**(**client**)**

client**.**close**()**

nickname **=** self**.**nicknames**[**index**]**

self**.**nicknames**.**remove**(**nickname**)**

self**.**broadcast**(**f'{nickname} left the chat!'**.**encode**(**'ascii'**))**

**break**

**def** receive**(**self**):**

**while** **True:**

client**,** address **=** self**.**server**.**accept**()** # Принятие нового клиента и его адреса

**print(**f'Connected with {**str(**address**)**}'**)**

client**.**send**(**'NICK'**.**encode**(**'ascii'**))** # Запрос никнейма у клиента

nickname **=** client**.**recv**(**1024**).**decode**(**'ascii'**)** # Получение никнейма от клиента

self**.**nicknames**.**append**(**nickname**)**

self**.**clients**.**append**(**client**)**

**print(**f'Nickname of the client is {nickname}!'**)**

self**.**broadcast**(**f'{nickname} joined the chat!'**.**encode**(**'ascii'**))** # Оповещение о входе клиента

client**.**send**(**'Connected to the server!'**.**encode**(**'ascii'**))** # Отправка клиенту сообщения о подключении

thread **=** threading**.**Thread**(**target**=**self**.**handle**,** args**=(**client**,))**

thread**.**start**()** # Запуск обработки сообщений клиента в отдельном потоке

**def** run**(**self**):**

**print(**"Server started..."**)**

self**.**receive**()** # Запуск прослушивания подключений и обработки клиентов

server **=** Server**()**

server**.**run**()**

Клиент:

**import** socket

**import** threading

**class** **Client:**

**def** \_\_init\_\_**(**self**,** host **=** '127.0.0.1'**,** port **=** 55555**):**

self**.**nickname **=** **input(**"Enter your nickname: "**)**

self**.**client **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

self**.**client**.**connect**((**host**,** port**))** # Подключение к серверу по указанному адресу и порту

**def** receive**(**self**):**

**while** **True:**

**try:**

message **=** self**.**client**.**recv**(**1024**).**decode**(**'ascii'**)** # Прием сообщения от сервера

**if** message **==** 'NICK'**:**

self**.**client**.**send**(**self**.**nickname**.**encode**(**'ascii'**))** # Отправка своего никнейма серверу

**else:**

**print(**message**)** # Вывод сообщения от сервера

**except:**

**print(**"An error occurred!"**)**

self**.**client**.**close**()**

**break**

**def** write**(**self**):**

**while** **True:**

message **=** f'{self**.**nickname}: {**input(**""**)**}' # Ввод сообщения с указанием своего никнейма

self**.**client**.**send**(**message**.**encode**(**'ascii'**))** # Отправка сообщения серверу

**def** run**(**self**):**

receive\_thread **=** threading**.**Thread**(**target**=**self**.**receive**)**

receive\_thread**.**start**()** # Запуск потока для приема сообщений от сервера

write\_thread **=** threading**.**Thread**(**target**=**self**.**write**)**

write\_thread**.**start**()** # Запуск потока для отправки сообщений на сервер

client **=** Client**()**

client**.**run**()**

Написал код для клиентов и общего сервера. Запуск произвел через командную строку. Как видно из скрина, каждый клиент независимо добавляется в чат через отдельный поток.

Сервер слушает подключения новых клиентов

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Клиенты могут общаться друг с другом через общий интерфейс

cИзображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Задание 5**

# Эта строка позволяет нам использовать функции для работы с сетевыми соединениями из библиотеки socket.

**import** http**.**server

**import** urllib**.**parse

# Запись оценок

grades **=** **{}**

**class** **MyHttpRequestHandler(**http**.**server**.**SimpleHTTPRequestHandler**):**

**def** do\_GET**(**self**):**

**if** self**.**path **==** '/'**:**

self**.**send\_response**(**200**)**

self**.**send\_header**(**"Content-type"**,** "text/html"**)**

self**.**end\_headers**()**

# HTML-форма для добавления оценки

html **=** """

<html>

<head><title>Add Grade</title></head>

<body>

<h1>Add Grade</h1>

<form method="POST" action="/add\_grade">

<label for="subject">Subject:</label><br>

<input type="text" id="subject" name="subject"><br>

<label for="grade">Grade:</label><br>

<input type="number" id="grade" name="grade" min="1" max="5"><br><br>

<input type="submit" value="Submit">

</form>

<p><a href="/grades">Grades Overview</a></p>

</body></html>

"""

self**.**wfile**.**write**(bytes(**html**,** "utf8"**))**

**return**

**elif** self**.**path **==** '/grades'**:**

self**.**send\_response**(**200**)**

self**.**send\_header**(**"Content-type"**,** "text/html"**)**

self**.**end\_headers**()**

# Страница с обзором оценок в виде таблицы

html **=** """

<html>

<head><title>Grades Overview</title></head>

<body>

<h1>Grades Overview</h1>

<table border="1">

<tr>

<th>Subject</th>

<th>Grade</th>

</tr>

"""

**for** subject**,** grade **in** grades**.**items**():**

html **+=** f"<tr><td>{subject}</td><td>{grade}</td></tr>"

html **+=** "</table>"

html **+=** "<p><a href='/'>Add Grade</a></p>"

html **+=** "</body></html>"

self**.**wfile**.**write**(bytes(**html**,** "utf8"**))**

**return**

**def** do\_POST**(**self**):**

content\_length **=** **int(**self**.**headers**[**'Content-Length'**])**

post\_data **=** self**.**rfile**.**read**(**content\_length**).**decode**(**'utf-8'**)**

params **=** urllib**.**parse**.**parse\_qs**(**post\_data**)**

subject **=** params**[**'subject'**][**0**]**

grade **=** params**[**'grade'**][**0**]**

**try:**

grade **=** **int(**grade**)**

**if** grade **<** 1 **or** grade **>** 5**:**

**raise** **ValueError(**"Grade out of range"**)**

grades**[**subject**]** **=** grade

self**.**send\_response**(**303**)**

self**.**send\_header**(**'Location'**,** '/'**)**

self**.**end\_headers**()**

**except** **ValueError:**

self**.**send\_response**(**400**)**

self**.**end\_headers**()**

self**.**wfile**.**write**(bytes(**"Bad request. Grade must be between 1 and 5."**,** "utf8"**))**

# Создание объекта обработчика запросов

handler\_object **=** MyHttpRequestHandler

# Создание HTTP-сервера

my\_server **=** http**.**server**.**HTTPServer**((**"localhost"**,** 8080**),** handler\_object**)**

**print(**"Server is running on localhost:8080..."**)**

# Запуск сервера

my\_server**.**serve\_forever**()**

**Добавление оценок**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Просмотр оценок по дисциплинам**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Вывод**

В ходе выполнения практического задания были реализованы различные аспекты работы с сетью и веб-программирования с использованием библиотеки socket в языке Python.

1. **Протокол UDP:** Разработано клиент-серверное приложение, в котором клиент отправляет сообщение "Hello, server", а сервер отвечает "Hello, client". Работа выполнена с использованием протокола UDP, что подчеркнуто важным аспектом данного этапа.
2. **Математические операции:** Создано клиент-серверное приложение для выполнения математических операций (теорема Пифагора, решение квадратного уравнения и т.д.). Работа выполнена с использованием протокола TCP, что добавляет надежности и устойчивости в обмене данными между клиентом и сервером.
3. **HTTP-сообщение с html-страницей:** Разработан сервер, который обрабатывает запросы клиента и отправляет http-сообщение, содержащее html-страницу. Это представляет собой базовый функционал web-сервера.
4. **Многопользовательский чат:** Создан многопользовательский чат с использованием TCP и многопоточности. Пользователи могут обмениваться сообщениями в реальном времени. Работа выполнена с применением библиотеки threading, что повысило эффективность обработки сообщений.
5. **Простой web-сервер:** Разработан простой web-сервер, который способен принимать и записывать информацию о дисциплине и оценке по дисциплине, а также отдавать информацию об оценках в виде html-страницы.

Выполненные задания демонстрируют понимание принципов работы с сетью, применение различных протоколов, в том числе TCP и UDP, а также навыки работы с веб-серверами и обмена данными между клиентом и сервером. Использование многопоточности позволило эффективно обрабатывать одновременные подключения пользователей к чату, что повысило функциональность и производительность приложения.