

Санкт-Петербургский Национальный Исследовательский

Университет ИТМО

Факультет «Информационных технологий и программирования»

Направление подготовки «Инфокоммуникационные технологии и системы  
связи»

**Практическая работа №6**

**Работа с сокетами**

Выполнила:

Бакланова А.Г.

Группа: К3322

Проверил:

Марченко Е.В.

Санкт-Петербург,

2024

## Содержание

Цель работы: .....	3
Ход работы.....	4
1. Практическое задание 1 .....	4
2. Практическое задание 2 .....	7
3. Практическое задание 3 .....	10
4. Практическое задание 4 .....	14
Заключение .....	21

## Цель работы:

Реализовать клиентскую и серверную часть приложения под различные задачи:

1. Клиент отправляет серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ отправляет клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.
2. Клиент запрашивает у сервера выполнение математической операции, связанной с поиском площади параллелограмма, параметры, которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту.
3. Клиент подключается к серверу. В ответ клиент получает http-сообщение, содержащее html-страницу, которую сервер подгружает из файла index.html.
4. Создание многопользовательского чата.

## **Ход работы**

### **1. Практическое задание 1**

В данном задании необходимо было реализовать клиентскую и серверную часть приложения. Клиент отправляет серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ отправляет клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.

Создано 2 файла: `server.py` и `client.py`. Файл `server.py` содержит сервер, в котором создается TCP/IP сокет, привязывается к заданному порту (а именно порт 8000) и сервер начинает ожидать подключения по аргументу 1 (размер очереди ожидающих соединений). Далее `.accept()` позволяет блокировать выполнение, пока не появится новое подключение, а при подключении клиента возвращается новый сокет (`client_socket`) и адрес клиента. В конце сервер получает и обрабатывает данные от клиента и отправляет ответ. (Рисунок 1)

The image shows a code editor with two tabs: 'server.py' and 'client.py'. The 'server.py' tab is active, displaying the following Python code:

```
1 import socket
2
3 def run_server():
4     # Создаем TCP/IP сокет
5     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     # Привязываем сокет к порту
7     server_address = ('localhost', 8000)
8     print(f"Starting server on {server_address[0]}:{server_address[1]}")
9     server_socket.bind(server_address)
10    # Слушаем входящие соединения
11    server_socket.listen(1)
12
13    while True:
14        print("Waiting for a connection...")
15        # Принимаем соединение
16        client_socket, client_address = server_socket.accept()
17        try:
18            print(f"Connection from {client_address}")
19            # Получаем данные от клиента
20            data = client_socket.recv(1024)
21            print(f"Received: {data.decode('utf-8')}")
22            # Отправляем ответ
23            response = "Hello, client"
24            client_socket.sendall(response.encode('utf-8'))
25            print(f"Sent: {response}")
26        finally:
27            client_socket.close()
28
29 if __name__ == "__main__":
30     run_server()
```

Рисунок 1 – Практическая 1. Файл server.py

Файл client.py, созданный для реализации данного задания содержит код, в котором создается TCP-сокет, устанавливается соединение с сервером localhost:8000, производится отправка данных на сервер и получение ответа. (Рисунок 2)

```

server.py x client.py x
1  import socket
2
3  def run_client():
4      # Создаем TCP/IP сокет
5      client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6      # Подключаемся к серверу
7      server_address = ('localhost', 8000)
8      print(f"Connecting to {server_address[0]}:{server_address[1]}")
9      client_socket.connect(server_address)
10
11     try:
12         # Отправляем данные
13         message = "Hello, server"
14         client_socket.sendall(message.encode('utf-8'))
15         print(f"Sent: {message}")
16         # Получаем ответ
17         data = client_socket.recv(1024)
18         print(f"Received: {data.decode('utf-8')}")
19     finally:
20         client_socket.close()
21
22     if __name__ == "__main__":
23         run_client()

```

Рисунок 2 – Практическая 1. Файл client.py

Результат работы представлен на рисунке 3.

```

C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.19045.5371]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab2-task1>python server.py
Starting server on localhost:8000
Waiting for a connection...
Connection from ('127.0.0.1', 26099)
Received: Hello, server
Sent: Hello, client
Waiting for a connection...

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5371]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab2-task1>python client.py
Connecting to localhost:8000
Sent: Hello, server
Received: Hello, client
D:\Users\User\PycharmProjects\web-lab2-task1>_


```

Рисунок 3 – Практическая 1. Результат работы

## 2. Практическое задание 2

Во втором задании необходимо было 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. Вариант выбирался в соответствии с порядковым номером в списке потока (Вариант 4: Поиск площади параллелограмма).

Также было создано 2 файла `server.py` и `client.py`. Файл `server.py` представляет собой сервер для вычисления площади параллелограмма, который принимает параметры от клиента, вычисляет площадь и отправляет результат обратно. В функции `calculate_parallelogram_area(base, height)` принимаются два аргумента (длина основания и высоты параллелограмма) и возвращается площадь параллелограмма, вычисляемая по формуле. Функция `start_server()` создает TCP-сокеты, привязывает его к заданному порту (`localhost:8001`), слушает входящее соединение, обрабатывает данные от клиента и отправляет ответ. (Рисунок 4)



```
1 import socket
2
3 def calculate_parallelogram_area(base, height):
4     return base * height
5
6 def start_server():
7     host = 'localhost'
8     port = 8001
9
10    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
11        s.bind((host, port))
12        s.listen(1)
13        print(f"Сервер запущен на {host}:{port}")
14
15        while True:
16            conn, addr = s.accept()
17            with conn:
18                print(f"Подключен клиент: {addr}")
19                try:
20                    data = conn.recv(1024).decode()
21                    if not data:
22                        continue
23
24                    print(f"Получены данные: {data}")
25                    base, height = map(float, data.split(', '))
26                    area = calculate_parallelogram_area(base, height)
27                    conn.sendall(str(area).encode())
28                except ValueError:
29                    conn.sendall("Ошибка: неверные параметры")
30                except Exception as e:
31                    conn.sendall(f"Ошибка: {str(e)}.encode()")
32
33 if __name__ == "__main__":
34     start_server()
```

Рисунок 4 – Практическая работа 2. Файл server.py

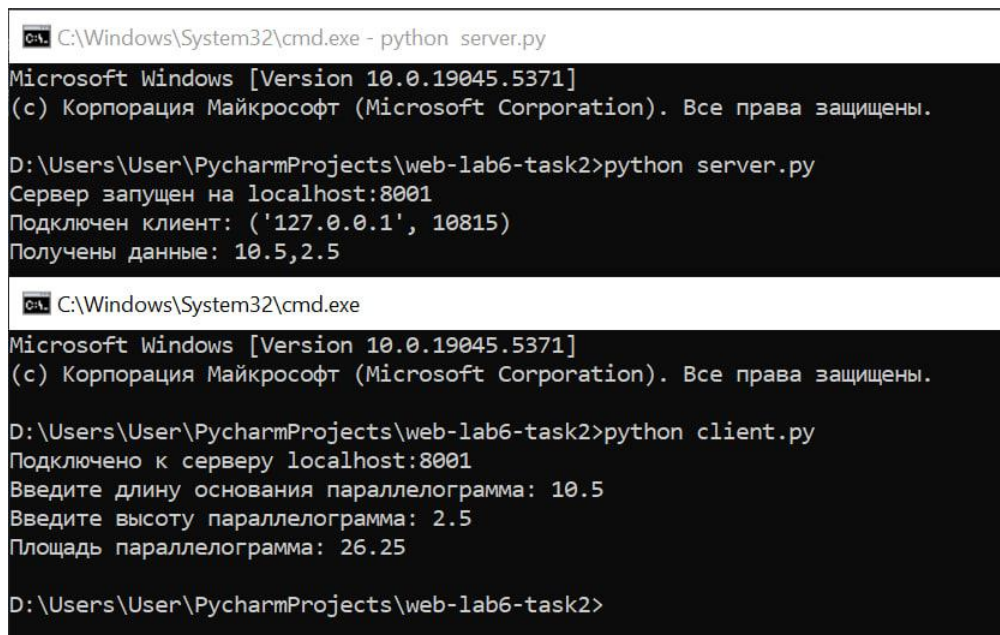
Файл client.py представляет собой клиентскую часть приложения. Функция `get_user_input()` запрашивает у пользователя длину основания и высоты параллелограмма, проверяет, что введены числа и возвращает кортеж (base, height). Далее в функции `start_server()` происходит подключение к серверу, отправка данных на этот сервер и получение ответа. (Рисунок 5)



```
server.py x client.py x
1      import socket
2
3      def get_user_input():
4          while True:
5              try:
6                  base = float(input("Введите длину основания параллелограмма: "))
7                  height = float(input("Введите высоту параллелограмма: "))
8                  return base, height
9              except ValueError:
10                 print("Ошибка: введите числовые значения")
11
12     def start_client():
13         host = 'localhost'
14         port = 8001
15
16         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
17             s.connect((host, port))
18             print(f"Подключено к серверу {host}:{port}")
19
20             base, height = get_user_input()
21             data = f"{base},{height}"
22             s.sendall(data.encode())
23
24             response = s.recv(1024).decode()
25             print(f"Площадь параллелограмма: {response}")
26
27     if __name__ == "__main__":
28         start_client()
```

Рисунок 5 – Практическая работа 2. Файл client.py

Результат работы представлен на рисунке 6.



```
C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.19045.5371]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task2>python server.py
Сервер запущен на localhost:8001
Подключен клиент: ('127.0.0.1', 10815)
Получены данные: 10.5,2.5

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5371]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task2>python client.py
Подключено к серверу localhost:8001
Введите длину основания параллелограмма: 10.5
Введите высоту параллелограмма: 2.5
Площадь параллелограмма: 26.25

D:\Users\User\PycharmProjects\web-lab6-task2>
```

Рисунок 6 – Практическая работа 2. Результат работы

### 3. Практическое задание 3

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

Созданы файлы http\_server.py, index.html и style.css. Файл http\_server.py в функции load\_html\_file(filename) загружает содержимое html-файла, в функции create\_http\_response(html\_content) формирует полный http-ответ, в функции run\_server() создает TCP-сокеты, который привязываем к заданному порту (localhost:8080) и ожидает подключения. Также используется SO\_REUSEADDR, который позволяет переиспользовать адрес (избегает ошибок при перезапуске). Далее происходит обработка подключений: принимается подключение, читается http-запрос и отправляется заранее загруженный файл в виде http-ответа. (Рисунок 7-8)

```
http_server.py x
1  import socket
2
3  def load_html_file(filename):
4      """Загружает содержимое HTML-файла"""
5      try:
6          with open(filename, 'r', encoding='utf-8') as file:
7              return file.read()
8      except FileNotFoundError:
9          return "<h1>Error: File not found</h1>"
10
11
12  def create_http_response(html_content):
13      """Формирует полный HTTP-ответ"""
14      response = f"HTTP/1.1 200 OK\r\n"
15      response += f"Content-Type: text/html; charset=utf-8\r\n"
16      response += f"Content-Length: {len(html_content)}\r\n"
17      response += "\r\n" # Пустая строка - разделитель заголовков и тела
18      response += html_content
19      return response.encode('utf-8')
20
21
22  def run_server(host='localhost', port=8080):
23      """Запускает HTTP-сервер"""
24      html_content = load_html_file('index.html')
25
26      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
27          server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
28          server_socket.bind((host, port))
29          server_socket.listen(1)
30          print(f"Server running on http://{host}:{port}")
31
32          while True:
33              client_conn, client_addr = server_socket.accept()
34              print(f"Connection from {client_addr}")
35
36              try:
```

Рисунок 7 – Практическая работа 3. Файл http\_server.py

```

32         while True:
33             client_conn, client_addr = server_socket.accept()
34             print(f"Connection from {client_addr}")
35
36             try:
37                 request = client_conn.recv(1024).decode('utf-8')
38                 print(f"Received request:\n{request}")
39
40                 # Отправляем HTTP-ответ с HTML-страницей
41                 http_response = create_http_response(html_content)
42                 client_conn.sendall(http_response)
43             finally:
44                 client_conn.close()
45
46
47 if __name__ == "__main__":
48     run_server()

```

Рисунок 8 – Практическая работа 3. Продолжение файла http\_server.py

В файле index.html создана простая html страница, которая выводит приветствие и с помощью скрипта текущую дату и время (Рисунок 9).

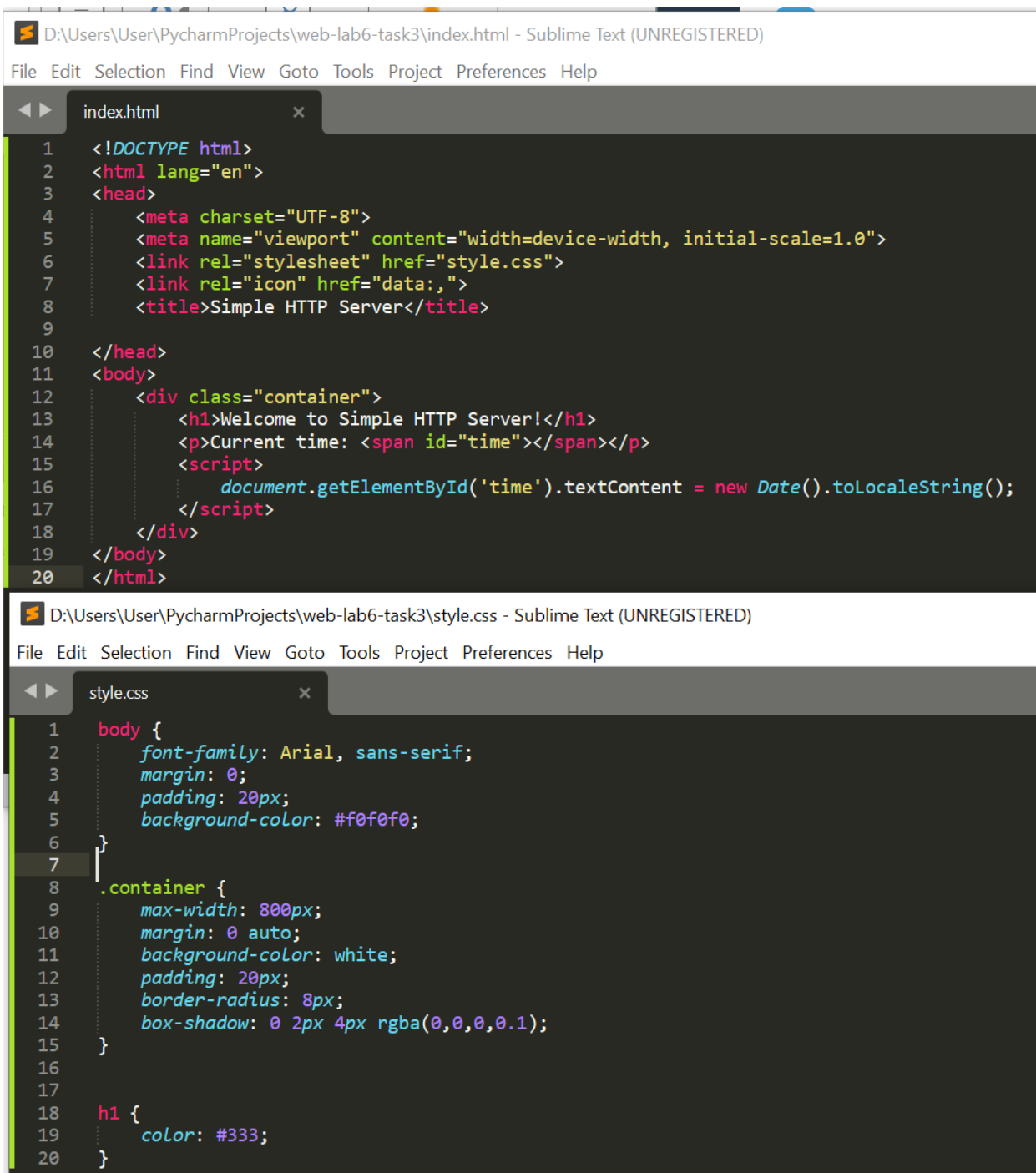


Рисунок 9 – Практическая работа 3. Файлы index.html и style.css

Результат работы представлен на рисунке 10-11.

```
C:\Windows\System32\cmd.exe - python http_server.py
Microsoft Windows [Version 10.0.19045.5371]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task3>python http_server.py
Server running on http://localhost:8080
Connection from ('127.0.0.1', 14725)
Received request:
GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132", "YaBrowser";v="25.2", "Yowser";v="2.5"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 YaBrowser/25.2.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ru,en;q=0.9

Connection from ('127.0.0.1', 14726)
Received request:
GET /style.css HTTP/1.1
Host: localhost:8080
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 YaBrowser/25.2.0.0 Safari/537.36
sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132", "YaBrowser";v="25.2", "Yowser";v="2.5"
sec-ch-ua-mobile: ?0
Accept: text/css,*/*;q=0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ru,en;q=0.9
```

Рисунок 10 – Практическая работа 3. Результат работы

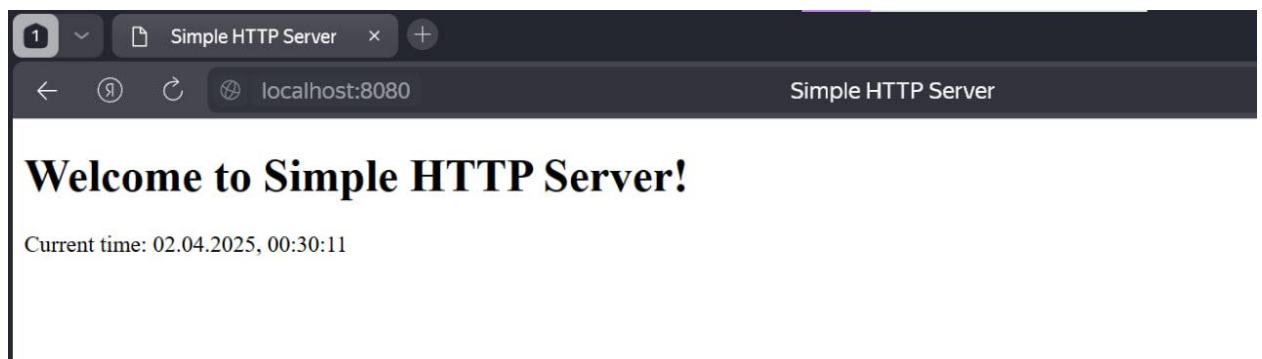


Рисунок 11 - Практическая работа 3. Результат работы (html страница)

## 4. Практическое задание 4

В данном задании создан многопользовательский чат. Также, как и в предыдущих заданиях созданы два файла `server.py` и `client.py`. Файл `server.py` содержит создание TCP-сокета, привязывается к порту и сервер начинает ожидать подключения (Рисунок 12).



```
server.py x client.py x
1 import socket
2 import threading
3
4 HOST = '127.0.0.1' # Локальный адрес
5 PORT = 55555      # Порт для подключения клиентов
6
7 # Список подключенных клиентов и их ников
8 clients = []
9 nicknames = []
10
11 # Создаем сокет сервера
12 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 server.bind((HOST, PORT))
14 server.listen()
15
16 print(f"Сервер запущен на {HOST}:{PORT}")
17
18 # Функция для трансляции сообщений всем клиентам
19 def broadcast(message, sender=None):
20     for client in clients:
21         if client != sender: # Не отправляем сообщение отправителю
22             try:
23                 client.send(message)
24             except:
25                 # Если отправка не удалась, удаляем клиента
26                 index = clients.index(client)
27                 clients.remove(client)
28                 client.close()
29                 nickname = nicknames[index]
30                 broadcast(f'{nickname} покинул чат!'.encode('utf-8'))
31                 nicknames.remove(nickname)
32
```

Рисунок 12 - Практическая работа 4. Файл server.py

В функции `handle_client(client)` происходит обработка подключения клиентов, а функция `receive()` отвечает за основные функции для принятия подключения (Рисунок 13).

```
server.py x client.py x
33 # Обработка подключений клиентов
34 def handle_client(client):
35     while True:
36         try:
37             # Получаем сообщение от клиента
38             message = client.recv(1024)
39             if message:
40                 broadcast(message, sender=client)
41             else:
42                 # Если сообщение пустое, клиент отключился
43                 index = clients.index(client)
44                 clients.remove(client)
45                 client.close()
46                 nickname = nicknames[index]
47                 broadcast(f'{nickname} покинул чат!'.encode('utf-8'))
48                 nicknames.remove(nickname)
49                 break
50         except:
51             index = clients.index(client)
52             clients.remove(client)
53             client.close()
54             nickname = nicknames[index]
55             broadcast(f'{nickname} покинул чат!'.encode('utf-8'))
56             nicknames.remove(nickname)
57             break
58
59 # Основная функция для принятия подключений
60 def receive():
61     while True:
62         # Принимаем новое подключение
63         client, address = server.accept()
64         print(f"Подключен клиент с адресом {str(address)}")
65
66         # Запрашиваем и сохраняем ник клиента
67         client.send('NICK'.encode('utf-8'))
68         nickname = client.recv(1024).decode('utf-8')
```

Рисунок 13 – Практическая работа 14. Функции handle\_client и receive

На рисунке 14 отображено продолжение кода для функции receive.



```

59     # Основная функция для принятия подключений
60     def receive():
61         while True:
62             # Принимаем новое подключение
63             client, address = server.accept()
64             print(f"Подключен клиент с адресом {str(address)}")
65
66             # Запрашиваем и сохраняем ник клиента
67             client.send('NICK'.encode('utf-8'))
68             nickname = client.recv(1024).decode('utf-8')
69             nicknames.append(nickname)
70             clients.append(client)
71
72             # Уведомляем чат о новом участнике
73             print(f"Ник клиента: {nickname}")
74             broadcast(f"{nickname} присоединился к чату!".encode('utf-8'))
75             client.send('Подключение к серверу успешно!'.encode('utf-8'))
76
77             # Запускаем поток для обработки сообщений клиента
78             thread = threading.Thread(target=handle_client, args=(client,))
79             thread.start()
80
81
82     if __name__ == '__main__':
83         receive()

```

Рисунок 14 – Практическая работа 14. Функцию receive

Файл client.py содержит создание сокета клиента, далее происходит подключение к серверу, в функции get\_nickname() происходит получение никнейма, функция receive() для приема сообщений от сервера (Рисунок 15).

```
server.py x client.py x
1  import socket
2  import threading
3
4  HOST = '127.0.0.1' # Адрес сервера
5  PORT = 55555      # Порт сервера
6
7  # Создаем сокет клиента
8  client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9
10 # Подключаемся к серверу
11 client.connect((HOST, PORT))
12
13 # Функция для получения никнейма
14 def get_nickname():
15     nickname = input("Введите ваш никнейм: ")
16     client.send(nickname.encode('utf-8'))
17     return nickname
18
19 # Функция для приема сообщений от сервера
20 def receive():
21     while True:
22         try:
23             # Получаем сообщение от сервера
24             message = client.recv(1024).decode('utf-8')
25             if message == 'NICK':
26                 client.send(nickname.encode('utf-8'))
27             else:
28                 print(message)
29         except:
30             print("Произошла ошибка!")
31             client.close()
32             break
33
```

Рисунок 15 – Практическая работа 4. Файл client.py

На рисунке 16 показана функция для отправки сообщений от сервера, получение никнейма и запуск потока для приема отправки сообщений

```
34     # Функция для отправки сообщений на сервер
35     def write():
36         while True:
37             message = f'{nickname}: {input("")}'
38             client.send(message.encode('utf-8'))
39
40     # Получаем никнейм
41     nickname = get_nickname()
42
43     # Запускаем потоки для приема и отправки сообщений
44     receive_thread = threading.Thread(target=receive)
45     receive_thread.start()
46
47     write_thread = threading.Thread(target=write)
48     write_thread.start()
```

Рисунок 16 – Практическая работа 4. Функция write

На рисунке 17 показан результат работы программы.

The image displays three separate Windows command prompt windows, each running a Python script. The first window, titled 'C:\Windows\System32\cmd.exe - python server.py', shows the execution of 'python server.py' which starts a server on 127.0.0.1:55555 and logs three client connections with nicknames 1, 2, and 3. The second window, titled 'C:\Windows\System32\cmd.exe - python client.py', shows the execution of 'python client.py' where a user enters 'NICK1' and the program outputs connection status and messages from other clients. The third window, also titled 'C:\Windows\System32\cmd.exe - python client.py', shows a similar process for a second client entering 'NICK2'.

```
C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.19045.5371]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task4>python server.py
Сервер запущен на 127.0.0.1:55555
Подключен клиент с адресом ('127.0.0.1', 48887)
Ник клиента: 1
Подключен клиент с адресом ('127.0.0.1', 48904)
Ник клиента: 2
Подключен клиент с адресом ('127.0.0.1', 48914)
Ник клиента: 3

C:\Windows\System32\cmd.exe - python client.py
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task4>python client.py
Введите ваш никнейм: 1
NICK1 присоединился к чату!Подключение к серверу успешно!
2 присоединился к чату!
3 присоединился к чату!
Привет
2: Привет 1!
3: Привет 1 и 2!

C:\Windows\System32\cmd.exe - python client.py
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task4>python client.py
Введите ваш никнейм: 2
NICK2 присоединился к чату!Подключение к серверу успешно!
3 присоединился к чату!
1: Привет
Привет 1!
3: Привет 1 и 2!

C:\Windows\System32\cmd.exe - python client.py
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\Users\User\PycharmProjects\web-lab6-task4>python client.py
Введите ваш никнейм: 3
NICK3 присоединился к чату!Подключение к серверу успешно!
1: Привет
2: Привет 1!
Привет 1 и 2!
```

Рисунок 17 – Практическая работа 4. Результат работы

## **Заключение**

В шестой практической работе реализована клиентская и серверная часть приложения под различные задачи:

1. Клиент отправляет серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ отправляет клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.
2. Клиент запрашивает у сервера выполнение математической операции, связанной с поиском площади параллелограмма, параметры, которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту.
3. Клиент подключается к серверу. В ответ клиент получает http-сообщение, содержащее html-страницу, которую сервер подгружает из файла index.html.
4. Создание многопользовательского чата.