

Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №6

«Сокеты»

Выполнил:

Швалов Даниил Андреевич

Группа: К33211

Проверила:

Марченко Елена Вадимовна

Санкт-Петербург

2023

1. Введение

Цель работы: овладеть практическими навыками и умениями реализации веб-серверов и использования сокетов.

2. Ход работы

Задание №1

В данном задании необходимо 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.

Для выполнения данного задания использовался язык программирования Python. В качестве библиотеки для работы с сокетами использовалась библиотека `socket`.

Код сервера находится в приложении А. Он устроен следующим образом. Сначала создается сокет из библиотеки `socket`, после чего с помощью метода `bind` назначается адрес и порт, который будет прослушиваться сервером. В качестве порта был выбран порт 3000. С помощью метода `listen` сервер начинает прослушивать данный порт на выбранном адресе. После этого с помощью метода `accept` сервер начинает ожидать входящих соединений. При возникновении соединения с клиентом сервер считывает пришедшие данные, переводит их в строку в кодировке UTF-8 и выводит на экран. После этого сервер отправляет клиенту сообщение «Hello, client» и закрывает соединение.

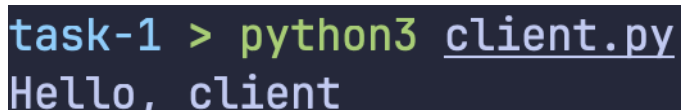
Код клиента находится в приложении Б. В нем также создается сокет из библиотеки `socket`, затем клиент подключается к серверу с помощью метода `connect`. После подключения клиент отправляет серверу сообщение «Hello, server» и считывает данные, отправляемые сервером. После считывания клиент превращает полученные данные в строку в кодировке UTF-8 и выводит на экран.

При запуске сервер начинает ожидать входящего соединения (рис. 1). После установления соединения клиент получает текст «Hello, client» (рис. 2). Сервер также получает сообщение «Hello, server» (рис. 3). После этого что сервер, что клиент, завершают свою работу.



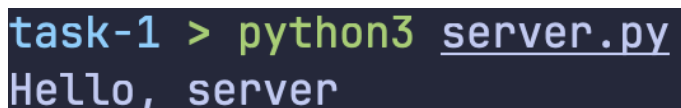
```
task-1 > python3 server.py
```

Рисунок 1 – Ожидание сервером запроса от клиента



```
task-1 > python3 client.py
Hello, client
```

Рисунок 2 – Ответ клиенту от сервера



```
task-1 > python3 server.py
Hello, server
```

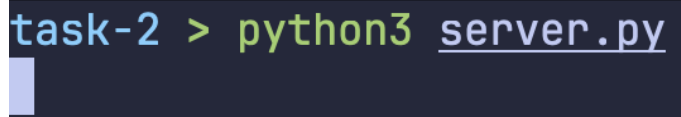
Рисунок 3 – Ответ серверу от клиента

Задание №2

В данном задании необходимо 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. В соответствии с порядковым номером в журнале, необходимо реализовать приложение, вычисляющее длину гипотенузы по теореме Пифагора.

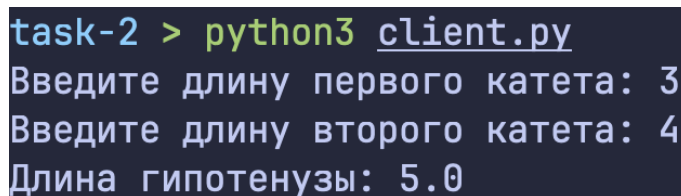
Основной код для данного задания был взят из задания №1. Исходный код сервера и клиента находится в приложениях В и Г соответственно. У клиента теперь вместо сообщения «Hello, server» отправляется сообщение, содержащее два числа разделенных пробелом. У сервера логика изменилась похожим образом: теперь сервер принимает эти два числа, вычисляет длину гипотенузы и отправляет ее клиенту.

При запуске сервер сначала ожидает запроса от клиента (рис. 4). После установления соединения, он принимает полученные числа и отправляет результат клиенту (рис. 5).



```
task-2 > python3 server.py
```

Рисунок 4 – Ожидание сервером запроса от клиента



```
task-2 > python3 client.py
Введите длину первого катета: 3
Введите длину второго катета: 4
Длина гипотенузы: 5.0
```

Рисунок 5 – Ответ клиенту от сервера

Задание №3

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

Получившийся сервер, как и в предыдущих заданиях, использует библиотеку socket. Теперь, вместо отправления простого текстового сообщения, клиент получает HTTP-ответ. В данном случае использовались следующие заголовки:

- **Content-Type: text/html** — для обозначения, что полученные данные являются HTML-страницей;
- **Content-Length: N** — длина полученных данных;
- **Connection: close** — сообщение о том, что соединение нужно закрыть.

Полезные данные, т. е. HTML-страница, подгружается из файла index.html. На рис. 6 изображен результат выполнения HTTP-запроса к серверу с помощью утилиты curl. Как видно, после выполнения запроса сервер прислал HTML-страницу, как и

ожидалось. Если открыть браузер по адресу localhost:3000, то будет загружена страница, изображенная на рис. 7. Исходный код сервера находится в приложении Д.

```
notes > curl -v localhost:3000
* Trying 127.0.0.1:3000...
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET / HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 127
< Connection: close
<
<!doctype html>
<html>
  <head>
    <title>Hello, World!</title>
  </head>
  <body>
    <p>Hello, World!</p>
  </body>
</html>
* Closing connection 0
```

Рисунок 6 – Результат выполнения HTTP-запроса к серверу

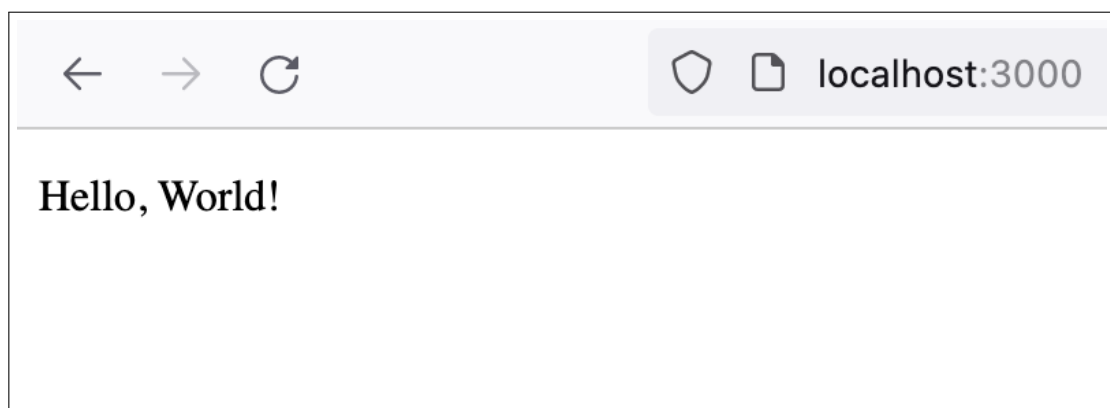


Рисунок 7 – Страница в браузере

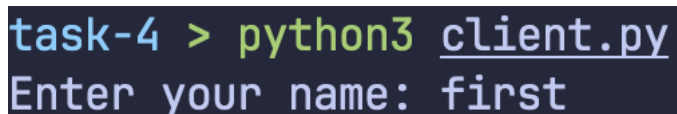
Задание №4

В данном задании необходимо реализовать многопользовательский чат. При выполнении данного задания также использовалась библиотека `socket` для реализации сетевого взаимодействия. Также использовалась библиотека `threading` для создания новых потоков. Она была нужна для того, чтобы сервер мог принимать и обрабатывать запросы сразу от нескольких клиентов. Также `threading` использовалась и для реализации клиентской части. Она использовалась для того, чтобы клиент мог получать новые сообщения сразу же, как только они появились, а не только тогда, когда клиент отправляет сообщения.

Сервер реализован следующим образом. Он, подобно предыдущим заданиям, прослушивает входящие соединения. В случае, если соединение установлено, создается новый поток с помощью библиотеки `threading`. В этом потоке происходит считывание получаемых сообщений и рассылка этих сообщений на другие клиенты. Исходный код сервера находится в приложении Е.

Клиент устроен следующим образом. Он считывает у пользователя его имя и подключается к серверу. После подключения пользователь создает дополнительный поток для получения и вывода новых сообщений, принимаемых от сервера. Далее клиент начинает считывать пользовательский ввод и отправлять его на сервер. Исходный код клиента находится в приложении Ж.

При запуске клиента пользователю сначала предлагается ввести его имя (рис. 8). После этого пользователь может начать вводить сообщения. После отправки сообщения первым пользователем (рис. 9), второй сразу же увидит его у себя на экране (рис. 10).



```
task-4 > python3 client.py
Enter your name: first
```

Рисунок 8 – Ввод имени подключившегося пользователя

```
task-4 > python3 client.py
Enter your name: first
Hello, World!
```

Рисунок 9 – Отправка сообщения первым пользователем

```
task-4 > python3 client.py
Enter your name: second
first: Hello, World!
```

Рисунок 10 – Получение сообщения вторым клиентом

3. Вывод

В ходе выполнения лабораторной работы я овладел практическими навыками и умениями реализации веб-серверов и использования сокетов.

Приложение А

Исходный код сервера задания №1

```
import socket

HOST = ""
PORT = 3000

with socket.socket() as sock:
    sock.bind((HOST, PORT))
    sock.listen()

    conn, addr = sock.accept()
    with conn:
        recieved = conn.recv(1024)
        print(recieved.decode("utf-8"))
        conn.send(b"Hello, client")
```

Приложение Б

Исходный код клиента задания №1

```
import socket

HOST = "localhost"
PORT = 3000

with socket.socket() as sock:
    sock.connect((HOST, PORT))
    sock.send(b"Hello, server")
    recieved = sock.recv(1024)
    print(recieved.decode("utf-8"))
```

Приложение В

Исходный код сервера задания №2

```
import socket
import math

HOST = ""
PORT = 3000

def find_hypotenuse(first_cathet, second_cathet):
    return math.sqrt(first_cathet**2 + second_cathet**2)

with socket.socket() as sock:
    sock.bind((HOST, PORT))
    sock.listen()

    conn, addr = sock.accept()
    with conn:
        recieved = conn.recv(1024)

        recieved = recieved.decode("utf-8")
        first_cathet, second_cathet = list(map(float, recieved.split()))

        data = find_hypotenuse(first_cathet, second_cathet)
        data = str.encode(str(data))

        conn.send(data)
```

Приложение Г

Исходный код клиента задания №2

```
import socket

HOST = "localhost"
PORT = 3000

first_cathet = input("Введите длину первого катета: ")
second_cathet = input("Введите длину второго катета: ")

data = [first_cathet, second_cathet]
data = list(map(str, data))
data = b" ".join(map(str.encode, data))

with socket.socket() as sock:
    sock.connect((HOST, PORT))

    sock.send(data)
    recieved = sock.recv(1024)
    print("Длина гипотенузы:", recieved.decode("utf-8"))
```

Приложение Д

Исходный код сервера задания №3

```
import socket
import math

HOST = ""
PORT = 3000
EMPTY_LINE = b""

def read_file(path):
    with open(path, "rb") as f:
        return f.read()

with socket.socket() as sock:
    sock.bind((HOST, PORT))
    sock.listen()

    conn, addr = sock.accept()
    with conn:
        http_code = b"HTTP/1.1 200 OK"
        content = read_file("index.html")

        headers = {
            "Content-Type": "text/html",
            "Content-Length": str(len(content)),
            "Connection": "close",
        }
        headers = [f"{key}: {value}" for key, value in headers.items()]
        headers = b"\n".join(map(str.encode, headers))

        data = [http_code, headers, EMPTY_LINE, content]
        data = b"\n".join(data)

        conn.send(data)
```

Приложение Е

Исходный код сервера задания №4

```
import socket
from threading import Thread

HOST = ""
PORT = 3000

def listen_for_client(sock):
    while True:
        try:
            data = sock.recv(1024).decode()
        except Exception as e:
            client_sockets.remove(sock)

        for client_socket in client_sockets:
            if client_socket != sock:
                client_socket.send(data.encode())

with socket.socket() as sock:
    sock.bind((HOST, PORT))
    sock.listen()
    client_sockets = set()

    while True:
        client_socket, client_address = sock.accept()
        client_sockets.add(client_socket)
        thread = Thread(target=listen_for_client, args=(client_socket,))
        thread.daemon = True
        thread.start()

    for client_sock in client_sockets:
        client_sock.close()
```

Приложение Ж

Исходный код клиента задания №4

```
import socket
from threading import Thread

HOST = "localhost"
PORT = 3000

def listen_for_messages():
    while True:
        data = sock.recv(1024).decode()
        print(data)

with socket.socket() as sock:
    sock.connect((HOST, PORT))
    username = input("Enter your name: ")

    thread = Thread(target=listen_for_messages)
    thread.daemon = True
    thread.start()

    while True:
        user_input = input()
        if user_input.lower() == "q":
            break
        user_input = f"{username}: {user_input}"
        sock.send(user_input.encode())

sock.close()
```
