САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Web-программирование

Отчет

Лабораторная работа №1

Выполнила:

Мухина Юлия

Группа К33401

Санкт-Петербург

2021 г.

Задача

овладеть практическими навыками и умениями реализации web-серверов и использования сокетов.

Ход работы

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.

Файл клиента:

Подключается к хосту, отправляет сообщение «Hello server» и принимает ответ.

```
1   import socket
2
3   sock = socket.socket()
4   sock.connect(('localhost', 9090))
5   sock.send(b'Hello, server')
6
7   data = sock.recv(1024)
8
9   udata = data.decode("utf-8")
10  print (udata)
11
12
13  sock.close()
14
```

Файл сервера:

Открывает порт, принимает данные, печатает их и сам выдает ответ «Hello, client».

```python
import socket

sock = socket.socket()

sock.bind(('localhost', 9090))
sock.listen(1)
conn, addr = sock.accept()


while True:
    data = conn.recv(1024)
    udata = data.decode("utf-8")

    if not data:
        break
    print(udata)
    conn.send(b'Hello, client')

conn.close()
```

Работа программы:

Сервер:

```
PS C:\Users\NITRO\web-programming\ITMO_ICT_WebDevelopment_2022-2023\students\k33401\laboratory_works\Mukhina_Yuliya\LR1\task_1> py server.py
Hello, server
PS C:\Users\NITRO\web-programming\ITMO_ICT_WebDevelopment_2022-2023\students\k33401\laboratory_works\Mukhina_Yuliya\LR1\task_1>
```

Клиент:

```
PS C:\Users\NITRO\web-programming\ITMO_ICT_WebDevelopment_2022-2023\students\k33401\laboratory_works\Mukhina_Yuliya\LR1\task_1> py client.py
Hello, client
PS C:\Users\NITRO\web-programming\ITMO_ICT_WebDevelopment_2022-2023\students\k33401\laboratory_works\Mukhina_Yuliya\LR1\task_1>
```

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. Вариант:

Поиск площади параллелограмма.

Файл клиента:

Подключается, отправляет данные двух сторон и угла между ними, получает ответ.

```
import socket
import numpy as np
import pickle

sock = socket.socket()
sock.connect(('localhost', 9090))

obj = {
    'a': input('Введите первую сторону: '),
    'b': input('Введите вторую сторону: '),
    'gr': input('Введите угол между сторонами: ')
}

data = pickle.dumps(obj)
if data:
    sock.send(data)

data = sock.recv(1024)
udata = data.decode("utf-8")
print (udata)

sock.close()
```

Файл сервера:

Подключается, обрабатывает данные (перемножением) и выдаёт ответ.

```python
import socket
import numpy as np
import pickle

sock = socket.socket()
sock.bind(('localhost', 9090))
sock.listen(1)
conn, addr = sock.accept()

while True:
    data = conn.recv(1024)
    #udata = data.decode("utf-8")
    data_variable = pickle.loads(data)
    if not data:
        break

    t_answer = str(float(data_variable['a'])*float(data_variable['b'])*np.sin(float(data_variable['gr'])))


    conn.send(t_answer.encode())

conn.close()
```

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.

Файл сервера:

```python
import socket


class MyHTTPServer:
    def __init__(self, host, port, name):
        self.host = host
        self.port = port
        self.name = name
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def serve_forever(self):
        try:
            self.server.bind((self.host, self.port))
            self.server.listen()
            while True:
                client, address = self.server.accept()
                self.serve_client(client)
        except KeyboardInterrupt:
            self.server.close()

    def serve_client(self, client):
        html = self.handle_request()
        self.send_response(client, html)
        client.close()

    def handle_request(self):
        with open("index.html", "r") as file:
            body = file.read()
        return body

    def send_response(self, client, html):
        client.sendall(f'HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n{html}'.encode())


if __name__ == '__main__':
    MyHTTPServer('127.0.0.1', 2000, 'example.com').serve_forever()

```

Функция serve_forever: подключается для чтения, бесконечно принимает данные осуществляет прием входящих соединений

Функция serve_client: с помощью функции hadle_request получает нужный файл, отправляет ответ с помощью send_response. Если возникает ошибка, сервер закрывается.

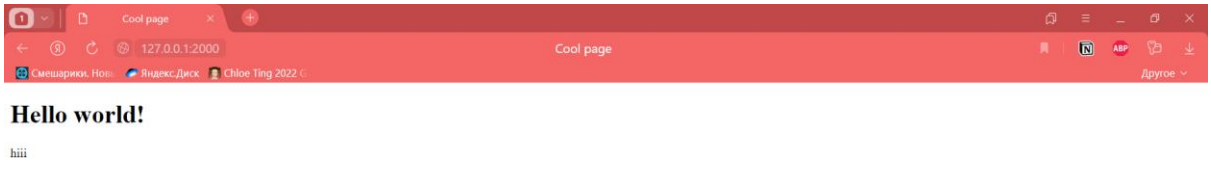
Файл index.html.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Cool page</title>
</head>
<body>
    <h1>Hello world!</h1>
    <article>
        hiii
    </article>
</body>
</html>
```

Запуск сервера:

```
LR1\task_3> py server.py
```

Ответ на нужном хосте:



4. Реализовать двухпользовательский или многопользовательский чат.

Файл сервера с комментариями:

```python
import socket, threading

HOST = "127.0.0.1"
PORT = 9091


class MyChat:
    def __init__(self, ip, host):
        # Ключ - клиент, значение - имя пользователя
        self.connections: dict[socket.socket, str] = {}
        # Иниализация самого сервера
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Сервер сокета будет привязан к указанном IP адресу и порту (в данном случае localhost и порт 9091)
        self.server.bind((ip, host))
        # Сервер начинает слушать входящие подключения
        self.server.listen()

    # Функция для отправки сообщений подключенным клиентам
    def broadcast(self, message: str, username: str):
        for client in self.connections.keys():
            client.send(f"{username}: {message}".encode())


    # Обработчики сообщений клиентов
    def handle_client(self, client: socket.socket):
        while True:
            username = str(self.connections.get(client))

            try:
                # Получаем сообщение от клиента в виде байтов и преобразуем в строку
                message = client.recv(1024).decode()
                self.broadcast(message, username)
            except:
                # В случае ошибки закрываем соединение с клиентом
                client.close()
                # Удаляем клиент из известных серверу соединений
                self.connections.pop(client)
                # Информируем других
                self.broadcast(f"{username} left...", "Server")
                break
```

```
    def receive(self):
        print("It's alive!")

        while True:
            client, address = self.server.accept()
            print(f"{str(address)} connected!")

            # При новом подключении узнаём имя пользователя и сохраняем его
            client.send(b"What is your username?")
            username = client.recv(1024).decode()
            self.connections[client] = username

            # Информируем подключенных клиентов о новом подключении
            self.broadcast(f"{username} joined!", "Server")

            # Создаём поток под новый клиент, чтобы обрабатывать их одновременно
            thread = threading.Thread(target=self.handle_client, args=(client,))
            thread.start()

    def run(self):
        self.receive()


if __name__ == "__main__":
    MyChat(HOST, PORT).run()
```

Файл клиента с комментариями:

```python
import socket
import threading


class MyClient:
    def __init__(self, ip, port):
        self.username = ""
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Подулючаем клиентский сокет к серверу
        self.sock.connect((ip, port))

    # Функция отвечающая за получение сообщений
    def receive(self):
        while True:
            try:
                message = self.sock.recv(1024).decode()
                # "Регистрация" пользователя, клиент отправит серверу имя пользователя
                if message == "What is your username?":
                    self.sock.send(self.username.encode())
                else:
                    print(message)
            except:
                print("Error!")
                self.sock.close()
                break

    # Функция отвечающая за отправку сообщение
    def send(self):
        while True:
            # Постоянно читаем и отправляем ввод клиента
            message = input()
            self.sock.send(message.encode())

    def start(self):
        self.username = input("Enter your username: ")
        # Запускаем поток, считывающий сообщения от сервера
        receive_thread = threading.Thread(target=self.receive)
        receive_thread.start()
        # Поток, считывающий ввод
        send_thread = threading.Thread(target=self.send)
        send_thread.start()


if __name__ == "__main__":
    MyClient("127.0.0.1", 9091).start()
```

Пример работы:

```
PS C:\Users\NITRO\web-program
Enter your username: igor
Server: igor joined!
ghbdtn
igor: ghbdtn
tasha: здарова
ну пока
igor: ну пока
tasha: пока
Server: tasha left...
```

5. Необходимо написать простой web-сервер для обработки GET и POST http запросов средствами Python и библиотеки socket.

Задание: сделать сервер, который может:

Принять и записать информацию о дисциплине и оценке по дисциплине.

Отдать информацию обо всех оценках по дисциплине в виде html-страницы.

Файл клиента:

```python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((socket.gethostname(), 3030))

msg1 = 'POST /grades?discipline=CV&name=Vasia&grade=4 HTTP/1.1\r\nHost: example.local\r\nAccept: text/html\r\nUser-Agent: Mozilla/5.0\r\n\r\n'
s.send(msg1.encode('iso-8859-1'))
msg4 = s.recv(1024)
print(msg4.decode())
s.close()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((socket.gethostname(), 3030))

msg2 = 'GET /grades?discipline=CV HTTP/1.1\r\nHost: example.local\r\nAccept: text/html\r\nUser-Agent: Mozilla/5.0\r\n\r\n'
s.send(msg2.encode('iso-8859-1'))
msg = s.recv(1024)
print(msg.decode())
s.close()
```

Реализованы методы: отправляющий информацию на сервер, принимающий информацию

Файл сервера:

```python
import socket
from Request import Request
from Response import Response
from urllib.parse import parse_qs

class MyHTTPServer:
    def __init__(self, host, port, name):
        self.host = host
        self.port = port
        self.name = name
        # Иниализация самого сервера
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.marks = {"Web-программирование": "1",
                      "Компьютерные сети": "5",
                      "Операционные системы": "1",
                      "Иностранный язык": "1"}

    def serve_forever(self):
        try:
            self.server.bind((self.host, self.port))
            # Сервер начинает слушать входящие подключения
            self.server.listen()
            while True:
                client, address = self.server.accept()
                # В бесконечном цикле осуществляет прием входящих соединений
                self.serve_client(client)
        except KeyboardInterrupt:
            self.server.close()
```

```
    def serve_client(self, client):
        try:
            data = client.recv(1024).decode()
            req = self.parse_request(data)
            res = self.handle_request(req)
            self.send_response(client, res)
        except Exception as e:
            print(e)
        client.close()
# Чтение и разбор HTTP-запроса
    def parse_request(self, data):
        #Читает строки
        req = data.split("\r\n")
        method, target, ver = req[0].split(" ")
        headers = self.parse_headers(req)
        # Отправляем соответствующий Request
        return Request(method=method, target=target, version=ver, headers=headers, data=data)

    def parse_headers(self, req):
        # Создаём словарик из заголовков
        headers = [h for h in req[1:req[1:].index("") + 1]]
        hdict = {}
        for h in headers:
            k, v = h.split(':', 1)
            hdict[k] = v
        return hdict
```

Декодируем сообщение, формируем соответствующие данные для ответа.

```
    def handle_request(self, req):
        try:
            if req.method == "GET" and req.path == "/":
                return self.handle_root()
            if req.method == "POST" and req.path.startswith("/api"):
                _id = int(req.query["id"][0]) - 1
                value = int(req.query["value"][0])
                if value > 5 or value < 1:
                    raise Exception("Неверное значение оценки")
                self.marks[list(self.marks.keys())[_id]] = value
                return self.handle_root()
            if req.method == "POST" and req.path.startswith("/form-request"):
                q = parse_qs(req.data[-int(req.headers["Content-Length"]):])
                _id = int(q["id"][0]) - 1
                value = int(q["value"][0])
                if value > 5 or value < 1:
                    raise Exception("Неверное значение оценки")
                self.marks[list(self.marks.keys())[_id]] = value
                return self.handle_root()
            return self.get_error(404, "Ты даже не гражданин!")
        except Exception as e:
            print(f"ERROR: {e}")
            return self.get_error(500, e)

    def send_response(self, client, res):
        client.sendall(f'HTTP/1.1 {res.status} OK\r\n{res.headers}\r\n\r\n{res.body}'.encode())
```

В зависимости от метода, которым у нас запрашивают данные, обновляем или не обновляем данные и вызываем следующую функцию.

```
    def send_response(self, client, res):
        client.sendall(f'HTTP/1.1 {res.status} OK\r\n{res.headers}\r\n\r\n{res.body}'.encode())

    def handle_root(self):
        body = """<!DOCTYPE html><html lang="en"><head>"""
        with open("res/style.css", "r") as file:
            body += "<style>"
            body += file.read()
            body += "</style>"
        body += """<meta charset="UTF-8"><title>Start page</title></head><body><table>"""
        body += f"<thead><tr><th>ID</th><th>Предмет</th><th>Оценка</th></tr></thead><tbody>"
        for i, mark in enumerate(self.marks.items()):
            body += f"<tr><td>{i + 1}</td><td>{mark[0]}</td><td>{mark[1]}</td></tr>"
        body += """</tbody></table>"""
        with open("res/form.html", "r") as file:
            body += file.read()
        body += """</body></html>"""
        return Response(200, "OK", "Content-Type: text/html; charset=utf-8", body)

    def get_error(self, code, text):
        return Response(code, "OK", "Content-Type: text/html; charset=utf-8", text)


if __name__ == '__main__':
    MyHTTPServer('127.0.0.1', 2001, 'example.com').serve_forever()
```

Формируем наш html ответ, подключая файлы извне для отображения и отправляя каждый элемент в словаре.

Файл Response.py, для удобства вынесен отдельно.

```
class Response:
    def __init__(self, status, reason, headers=None, body=None):
        self.status = status
        self.reason = reason
        self.headers = headers
        self.body = body
```

Файл Request.py, для удобства для удобства вынесен отдельно.

```python
from urllib.parse import parse_qs, urlparse


class Request:
    def __init__(self, method, target, headers, version, data):
        self.method = method
        self.target = target
        self.version = version
        self.url = urlparse(self.target)
        self.query = parse_qs(self.url.query)
        self.path = self.url.path
        self.headers = headers
        self.data = data
```

Вывод

Овладела практическими навыками и умениями реализации web-серверов и использования сокетов.