МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет
о лабораторной работе №1
по дисциплине
«Web программирование»

Выполнила
Голуб А.Л.
группа К33421

Проверил
Говоров А. И.

Санкт-Петербург
2021

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.

Код

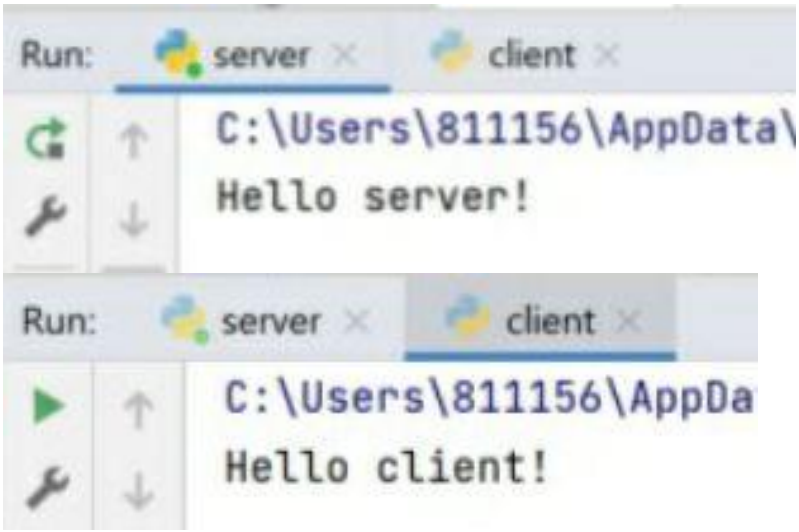
server.py

```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind(('127.0.0.1', 9000))
conn.listen(10)

while True:
    try:
        # receiving client's message
        client_socket, address = conn.accept()
        data = client_socket.recv(16384)
        data = data.decode('utf-8')
        print(data)

        # sending a response
        client_socket.send(b'Hello client! \n')

    except KeyboardInterrupt:
        conn.close()
        break
```

client.py

```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(("127.0.0.1", 9000))

# sending message to server
conn.send(b'Hello server! \n')

# receiving server's response
data = conn.recv(16384)
data = data.decode('utf-8')
print(data)

conn.close()
```

Скриншоты выполнения программы



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. Математическая операция – поиск площади трапеции.

Код

server.py

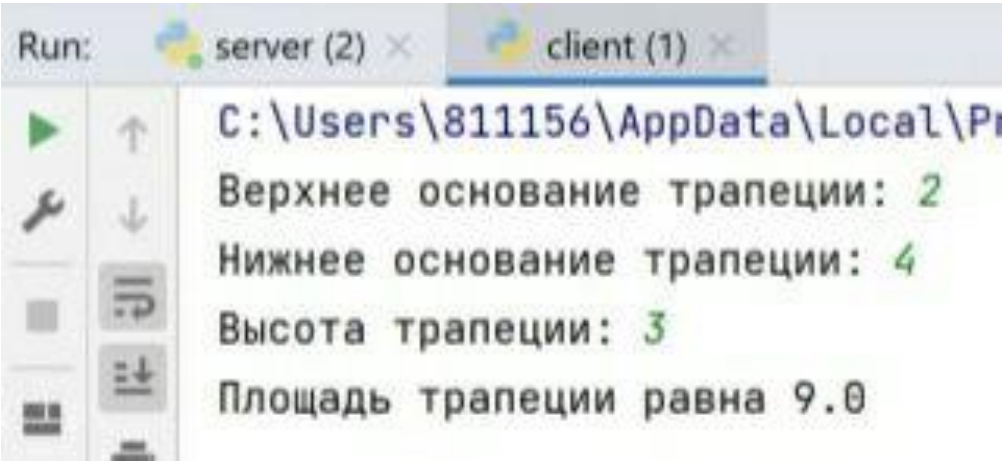
```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind(('127.0.0.1', 9000))
conn.listen(10)

while True:
    try:
        # receiving client's message
        client_socket, address = conn.accept()
        data = client_socket.recv(16384)
        data = data.decode('utf-8')
        a, b, h = map(int, data.lstrip().rstrip().split())
        print('a =', a)
        print('b =', b)
        print('h =', h)

        # calculating and sending response
        s = 0.5 * (a + b) * h
```

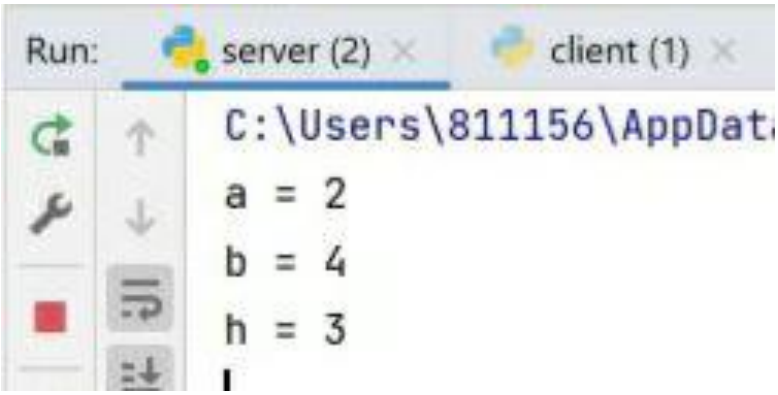
```
        s = str(s).encode()
        client_socket.send(s)

    except KeyboardInterrupt:
        conn.close()
        break
```

client.py

```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(("127.0.0.1", 9000))

# sending message to server
a = input('Верхнее основание трапеции: ')
b = input('Нижнее основание трапеции: ')
h = input('Высота трапеции: ')
message = ' '.join([str(a), str(b), str(h)]).encode()
conn.send(message)

# receiving server's response
data = conn.recv(16384)
data = data.decode('utf-8')
print('Площадь трапеции равна', data)

conn.close()
```

Скриншоты выполнения программы





Задание 3

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

- принять и записать информацию о дисциплине и оценке по дисциплине
- отдать информацию обо всех оценках по дисциплине в виде html-страницы

Код

http_server.py

```
import socket
import sys
from functools import lru_cache
from urllib.parse import parse_qs, urlparse

MAX_LINE = 64 * 1024
MAX_HEADERS = 100


class MyHTTPServer:
    def __init__(self, host, port, server_name):
        self._host = host
        self._port = port
        self._server_name = server_name
        self._marks = dict()

    def serve_forever(self):
        serv_sock = socket.socket(
            socket.AF_INET,
            socket.SOCK_STREAM,
            proto=0)

        try:
            serv_sock.bind((self._host, self._port))
            serv_sock.listen()

            while True:
                conn, _ = serv_sock.accept()
                try:
                    self.serve_client(conn)
                except Exception as e:
                    print('Client serving failed:', e)
        finally:
            serv_sock.close()

    def serve_client(self, conn):
        try:
            req = self.parse_request(conn)
            resp = self.handle_request(req)
            self.send_response(conn, resp)
        except ConnectionResetError:
            conn = None
```

```
        except Exception as e:
            self.send_error(conn, e)
        if conn:
            conn.close()

    def parse_request(self, conn):
        rfile = conn.makefile('rb')
        method, target, ver = self.parse_request_line(rfile)
        headers = self.parse_headers(rfile)
        return Request(method, target, ver, headers, rfile)

    def parse_headers(self, rfile):
        headers = []
        while True:
            line = rfile.readline(MAX_LINE + 1)
            if len(line) > MAX_LINE:
                raise Exception('Header line is too long')
            if line in (b'\r\n', b'\n', b''):
                break
            headers.append(line)
            if len(headers) > MAX_HEADERS:
                raise Exception('Too many headers')

        headers_dict = dict()
        for h in headers:
            h = h.decode('iso-8859-1')
            k, v = h.split(':', 1)
            headers_dict[k] = v
        return headers_dict

    def parse_request_line(self, rfile):
        raw = rfile.readline(MAX_LINE + 1)
        if len(raw) > MAX_LINE:
            raise Exception('Request line is too long')

        req_line = str(raw, 'iso-8859-1')
        req_line = req_line.rstrip('\r\n')
        words = req_line.split()
        if len(words) != 3:
            raise Exception('Malformed request line')

        method, target, ver = words
        if ver != 'HTTP/1.1':
            raise Exception('Unexpected HTTP version')
        return method, target, ver

    def handle_request(self, req):
        # print('handle_request')
        if req.path == '/marks' and req.method == 'POST':
            subject = req.query['subject'][0]
            mark = req.query['mark'][0]
            if subject in self._marks:
                self._marks[subject].append(mark)
            else:
                self._marks[subject] = [mark]
            return Response(204, 'Created')

        if req.path == '/marks' and req.method == 'GET':
            subject = req.query['subject'][0]
            contentType = 'text/html; charset=utf-8'
            body = '<html><head></head><body>'
            body += f'Subject: <b>{subject}</b><br>'
            if subject in self._marks:
                body += f'Marks: {", ".join(self._marks[subject])}'
```

```
            else:
                body += f'Marks: no data'
            body += '</body></html>'
            body = body.encode('utf-8')
            headers = [('Content-Type', contentType), ('Content-Length',
len(body))]
            return Response(200, 'OK', headers, body)

        raise HTTPError(404, 'Not found')

    def send_response(self, conn, resp):
        wfile = conn.makefile('wb')
        status_line = f'HTTP/1.1 {resp.status} {resp.reason}\r\n'
        wfile.write(status_line.encode('iso-8859-1'))

        if resp.headers:
            for (key, value) in resp.headers:
                header_line = f'{key}: {value}\r\n'
                wfile.write(header_line.encode('iso-8859-1'))
        wfile.write(b'\r\n')

        if resp.body:
            wfile.write(resp.body)
        wfile.flush()
        wfile.close()

    def send_error(self, conn, err):
        try:
            status = err.status
            reason = err.reason
            body = (err.body or err.reason).encode('utf-8')
        except:
            status = 500
            reason = b'Internal Server Error'
            body = b'Internal Server Error'
        resp = Response(status, reason,
                        [('Content-Length', len(body))],
                        body)
        self.send_response(conn, resp)


class Request:
    def __init__(self, method, target, version, headers, rfile):
        self.method = method
        self.target = target
        self.version = version
        self.headers = headers
        self.rfile = rfile

    @property
    def path(self):
        return self.url.path

    @property
    @lru_cache(maxsize=None)
    def query(self):
        return parse_qs(self.url.query)

    @property
    @lru_cache(maxsize=None)
    def url(self):
        return urlparse(self.target)
```

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


class HTTPError(Exception):
    def __init__(self, status, reason, body=None):
        super()
        self.status = status
        self.reason = reason
        self.body = body


if __name__ == '__main__':
    host = sys.argv[1]
    port = int(sys.argv[2])
    name = sys.argv[3]

    serv = MyHTTPServer(host, port, name)
    try:
        serv.serve_forever()
    except KeyboardInterrupt:
        pass
```

Скриншоты выполнения программы

POST

```
Командная строка - nc  localhost 53210
C:\Users\811156\Downloads\nc111nt>nc localhost 53210
POST /marks?subject=math&mark=3 HTTP/1.1
Host:example.local

HTTP/1.1 204 Created

^C
C:\Users\811156\Downloads\nc111nt>nc localhost 53210
POST /marks?subject=math&mark=5 HTTP/1.1
Host:example.local

HTTP/1.1 204 Created
```

GET

```
Командная строка - nc  localhost 53210

C:\Users\811156\Downloads\nc111nt>nc localhost 53210
GET /marks?subject=math HTTP/1.1
Host:example.local

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74

<html><head></head><body>Subject: <b>math</b><br>Marks: 3, 5</body></html>
```

```
C:\Users\811156\Downloads\nc111nt>nc localhost 53210
GET /marks?subject=english HTTP/1.1
Host:example.local

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 80

<html><head></head><body>Subject: <b>english</b><br>Marks: no data</body></html>
```

Задание 4

Реализовать двухпользовательский или многопользовательский чат.

Код

server.py

```
import socket
from threading import *


# chat thread class
class ChatThread(Thread):
    def __init__(self, conn, client_name):
        Thread.__init__(self)
        self.conn = conn
        self.client_name = client_name

    def run(self):
        while True:
            try:
                # receive and broadcast message
                message = self.conn.recv(1024)
                message = self.client_name + ': ' + message.decode()
                print(message)
                broadcast(message, self.client_name)
            except:
                self.conn.close()
```

```
# broadcast message to all other chat members
def broadcast(message, author):
    for client in client_list:
        if client.client_name != author:
            try:
                client.conn.send(message.encode())
            except:
                client.conn.close()
                client_list.remove(client)


# main part
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('127.0.0.1', 9000))
server.listen(10)

client_list = []
while True:
    try:
        # connect to a new client
        conn, addr = server.accept()
        print(addr, 'connected')

        # create a new thread
        new_thread = ChatThread(conn, str(addr[1]))
        client_list.append(new_thread)
        new_thread.start()

    except KeyboardInterrupt:
        server.close()
        for client in client_list:
            client.conn.close()
```

client.py

```
import socket
import sys

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 9000))

sockets_list = [sys.stdin, client]

while True:
    try:
        # read text from input and send it
        message = input('You: ')
        client.send(message.encode())

        # receive message from server
        message = client.recv(1024)
        message = message.decode()
        print(message)

    except KeyboardInterrupt:
        client.close()

    except:
        continue
```

Скриншоты выполнения программы

```
Run:  server  client_1  client_2
C:\Users\811156\AppData\Local\Pr
You: hey
58396: hello
You: how r u
58396: hi
You: hi
58396: good and you
You: good
58396: hi
You:
```

```
Run:  server  client_1  client_2
C:\Users\811156\AppData\Local\Pro
You: hello
58395: hey
You: hi
58395: how r u
You: good and you
58395: hi
You: hi
58395: good
You:
```

```
Run:  server  client_1  client_2
C:\Users\811156\AppData\Local\Pr
('127.0.0.1', 58395) connected
('127.0.0.1', 58396) connected
58395: hey
58396: hello
58395: how r u
58396: hi
58396: good and you
58395: hi
58395: good
58396: hi
```