САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Web программирование

Отчет

Лабораторная работа №1

Выполнил:
Золотов Павел

Группа К33401

Проверил:
Говоров А. И.

Санкт-Петербург

2021 г.

Задача

Научиться работать с сокетами на Python. Написать простой web-сервер для обработки GET и POST http запросов средствами Python и библиотеки socket.

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.

task1 > server.py

```python
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind(('127.0.0.1', 65432))
    s.listen()
    connection, address = s.accept()
    data = connection.recv(1024).decode('utf-8')
    print(data)
    connection.send(b'Hello, client')

```

task1 > client.py

```python
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect(('127.0.0.1', 65432))
    s.sendall(b'Hello, server')
    data = s.recv(1024).decode('utf-8')
print(data)

```

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.

task2 > server.py

```python
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind(('127.0.0.1', 65432))
    s.listen(3)
    connection, address = s.accept()
    side = int(connection.recv(1024).decode('utf-8'))
    height = int(connection.recv(1024).decode('utf-8'))
    connection.send(str(side*height).encode())
```

task2 > client.py

```python
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect(('127.0.0.1', 65432))
    side = input("Сторона параллелограмма: ")
    height = input("Высота, проведенная к данной стороне: ")
    s.send(side.encode())
    s.send(height.encode())
    data = s.recv(1024).decode('utf-8')
print('Площадь параллелограмма: ', data)
```

3) 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 оценах по дсициплине в виде html-страницы.

task3 > server.py

```python
import socket
import sys

from functools import lru_cache
from urllib.parse import parse_qs, urlparse

MAX_LINE = 64*1024

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)

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

```python
class MyHTTPServer:

    def __init__(self, host, port, server_name):
        self._host = host
        self._port = port
        self._server_name = server_name
        self._marks = {}


    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, address = 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
        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_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 parse_headers(self, rfile):
    headers = []
    for line in rfile:
      if line in (b'\r\n', b'\n', b''):
        break
      headers.append(line)
    return headers

  def handle_request(self, req):
    if req.method == 'POST':
      mark_id = len(self._marks) + 1
      self._marks[mark_id] = {'id': mark_id, 'subject': req.query['subject'][0], 'mark': req.query[
      return Response(204, 'Created')
    if req.method == 'GET':
      contentType = 'text/html; charset=utf-8'
      body = '<html><head><title>Marks</title></head><body><ul>'
      for u in self._marks.values():
        body += f'<li>#{u["id"]} {u["subject"]}, {u["mark"]}</li>'
      body += '</ul>'
      body += '</body></html>'
      body = body.encode('utf-8')
      headers = [('Content-Type', contentType), ('Content-Length', len(body))]
      return Response(200, 'OK', headers, body)
    raise Exception('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()

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
```

4) Реализовать многопользовательский чат.

task4 > server.py

```python
import socket
from threading import Thread

def send(message, sender):
    for client in clients:
        if sender != client:
            client.send(message)


def listen(client):
    while True:
        message = client.recv(1024)
        send(message, client)

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind(('127.0.0.1', 65432))
conn.listen()
clients = []
while True:
    new_client, address = conn.accept()
    if new_client not in clients:
        clients.append(new_client)
    Thread(target=listen, args=[new_client]).start()
```

task4 > client.py

```python
import socket
from threading import Thread

def send():
    while True:
        message = input()
        conn.send(message.encode('utf-8'))


def get():
    while True:
        data = conn.recv(1024).decode('utf-8')
        print(data)

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(('127.0.0.1', 65432))
Thread(target=send).start()
Thread(target=get).start()
```

Вывод

В результате выполнения работы я познакомился с работой сокетов в Python, в том числе реализовал связь со множеством клиентов. Также в ходе выполнения работы был создан свой веб сервер, поддерживающий обработку GET и POST запросов.