САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Web-программирование

Отчет

Лабораторная работа №1. Работа с сокетами

Выполнил:

Костылев Иван

Группа К33401

Проверил:
Говоров А. И.

Санкт-Петербург

2021 г.

Цель работы: 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.

Client

```
import socket

HOST, PORT = "127.0.0.1", 14900

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(("127.0.0.1", 14900))
hello_server = "Hello, Server!"
conn.send(hello_server.encode("utf-8"))
data = conn.recv(16384)
print(data.decode("utf-8"))
conn.close()
```

server

```
import socket

HOST, PORT = "127.0.0.1", 14900

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind((HOST, PORT))
conn.listen(10)

sock, address = conn.accept()
client_data = sock.recv(16384)
client_data = client_data.decode("utf-8")
print(client_data)
msg_for_client = "Hello, Client!"
sock.send(msg_for_client.encode("utf-8"))
conn.close()
```

2. Реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры, которые вводятся с клавиатуры.

Client

```python
import socket

HOST, PORT = "127.0.0.1", 14900

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect((HOST, PORT))
a_b = input("Write two numbers with space: ")
conn.send(a_b.encode("utf-8"))
data = conn.recv(16384)
print("Pifagor result: " + data.decode("utf-8"))
conn.close()
```

Server

```python
import socket
import math
from typing import cast

HOST, PORT = "127.0.0.1", 14900

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind((HOST, PORT))
conn.listen(5)

try:
    sock, address = conn.accept()
    client_data = sock.recv(16384)
    client_data = client_data.decode("utf-8")
    print("Calculating Pifagor for data: " + client_data)
    a, b = [float(num) for num in client_data.split(" ")]
    res = math.sqrt(a ** 2 + b ** 2)
    msg_for_client = str(res)
    sock.send(msg_for_client.encode("utf-8"))
    conn.close()
except:
    print("except")
    conn.close()
```

3. Разработка простого HTTP-сервера с использованием сокетов

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

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._data = dict()

    def serve_forever(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            sock.bind((self._host, self._port))
            sock.listen()
            while True:
                conn, _ = sock.accept()
                try:
                    self.serve_client(conn)
                except Exception as e:
                    print("Client serving failed with exception", e)
        finally:
            sock.close()


    def serve_client(self, connection):
        try:
            request = self.parse_request(connection)
            response = self.handle_request(request)
            self.send_response(connection, response)
        except ConnectionResetError as e:
            connection = None
        except Exception as e:
            connection = None


    def parse_request(self, connection):
        rfile = connection.makefile('rb')
        method, url, version = self.parse_request_line(rfile)
        headers = self.parse_headers(rfile)
        return Request(method, url, version, rfile)


    def parse_request_line(self, rfile):
        line = rfile.readline(MAX_LINE + 1)
        if len(line) > MAX_LINE:
            raise Exception('Request line is too long')
        req_line = str(line, 'iso-8859-1')
        req_line = req_line.rstrip('\r\n')
        words = req_line.split()
```

```
        if len(words) != 3:
            raise Exception('Malformed request line')
        method, url, version = words
        if version != 'HTTP/1.1':
            raise Exception('Unexpected HTTP version')
        return (method, url, version)


    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')
        return headers


    def handle_request(self, req):
        if req.method == 'POST':
            return self.handle_post(req)
        if req.method == 'GET':
            return self.handle_get(req)
        return Response(400, 'Bad request')


    def handle_post(self, req):
        self._data = req.query
        return Response(204, 'Created')

    def handle_get(self, req):
        accept = req.headers.get('Accept')
        if 'application/json' in accept:
            contentType = 'application/json; charset=utf-8'
            body = json.dumps(self._data)
        else:
            return Response(406, 'Not Acceptable')
        body = body.encode('utf-8')
        headers = [('Content-Type', contentType),
                ('Content-Length', len(body))]
        return Response(200, 'OK', headers, body)

    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()


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

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. Разработка многопользовательского чата

```
aboratory_work_1/task_4$ python3 server.py
Client #0 connected 127.0.0.1:41814
ivan > Hello, Server!
ivan > Hello, Server!
Client #1 connected 127.0.0.1:41816
vadim > Hello, Server!
ivan > Hello, Server!
vadim > Hello, Server!
ivan > Hello, Server!
vadim > Hello, Server!
ivan > Hello, Server!
vadim > Hello, Server!
ivan > Hello, Server!
vadim > Hello, Server!
ivan > Hello, Server!
vadim > Hello, Server!
ivan > Hello, Server!
ivan > Hello, Server!
```

server.py

```python
import threading
import socket
import time
import sys

def run_server(port=44346):
    serv_sock = create_serv_sock(port)
    cid = 0
    while True:
        client_sock = accept_client_conn(serv_sock, cid)
        t = threading.Thread(target=serve_client,
                             args=(client_sock, cid))
        t.start()
        cid += 1

def serve_client(client_sock, cid):
    while True:
        mes = client_sock.recv(100)
        data = mes.decode("utf-8")
        if data:
            print(data)

def create_serv_sock(serv_port):
    serv_sock = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM,
                              proto=0)
    serv_sock.bind(('', serv_port))
    serv_sock.listen()
    return serv_sock

def accept_client_conn(serv_sock, cid):
    client_sock, client_addr = serv_sock.accept()
    print(f'Client #{cid} connected '
          f'{client_addr[0]}:{client_addr[1]}')
```

```
    return client_sock

if __name__ == '__main__':
    try:
        port=int(sys.argv[1])
        run_server(port)
    except:
        run_server()
```

Вывод

Мы овладели практическими навыками и умениями реализации web-серверов и использования сокетов.