ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет «Инфокоммуникационных технологий»
Направление подготовки «09.03.03 Прикладная информатика»
Бакалаврская программа «Мобильные и сетевые технологии»

Лабораторная работа №1

по дисциплине «Веб программирование»

«Работа с сокетами»

Выполнил

_____ / Комаров Г. Ю., К33402
(подпись) (Фамилия И.О., группа)

Проверил

_____ / Говоров А. И.

Дата _____

Санкт-Петербург
2021

Цель работы

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

Выполнение работы

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.

Файл task1_server.py

```
import argparse
import socket

parser = argparse.ArgumentParser(description='Task 1 Server')
parser.add_argument('--host', help='Hostname/IP to bind', default='127.0.0.1')
parser.add_argument('--port', help='Port to bind', default=1337)
parser.add_argument('-b', '--buffer', help='Connection buffer size', type=int, default=1024)
args = parser.parse_args()

    print(f'Server is listening for TCP connections at {args.host}:{args.port}')
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.bind((args.host, args.port))
        sock.listen(1)

        conn, address = sock.accept()

        data = conn.recv(args.buffer).decode()
        print(data, end='')

        if data.strip() == 'Hello, server!':
            conn.sendall('Hello, client!\n'.encode())
        else:
            conn.sendall('Unknown message!\n'.encode())
```

Файл task1_client.py

```
import argparse
import socket

parser = argparse.ArgumentParser(description='Task 2 Client')
parser.add_argument('--host', help='Hostname/IP to connect', default='127.0.0.1')
parser.add_argument('--port', help='Port to connect', default=1337)
parser.add_argument('-b', '--buffer', help='Connection buffer size', type=int, default=1024)
args = parser.parse_args()

inp = input('Enter quadratic equation coefficients (a, b, c) divided by space. Such as:\n1 2 1\n>> ')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as conn:
    conn.connect((args.host, args.port))
    conn.sendall(inp.encode())

    received = conn.recv(args.buffer).decode()
    print(received, end='')
```

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.

Вариант b. Решение квадратного уравнения.

Файл task2_server.py

```
1. import argparse
2. import socket
3. from math import sqrt
4.
5. parser = argparse.ArgumentParser(description='Task 2 Server')
6. parser.add_argument('--host', help='Hostname/IP to bind',
   default='127.0.0.1')
7. parser.add_argument('--port', help='Port to bind', default=1337)
8. parser.add_argument('-b', '--buffer', help='Connection buffer
   size', type=int, default=1024)
9. args = parser.parse_args()
10.
11.     print(f'Server is listening for TCP connections at
   {args.host}:{args.port}')
12.     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
   sock:
13.         sock.bind((args.host, args.port))
14.         sock.listen(1)
15.
16.         conn, address = sock.accept()
17.
18.         data = conn.recv(args.buffer).decode()
19.
20.         try:
21.             a, b, c = map(int, data.strip().split())
22.         except:
23.             conn.sendall('Wrong input format!\n'.encode())
24.         else:
25.             D = b**2 - 4 * a * c
26.             if D > 0:
27.                 x1, x2 = (-b + sqrt(D)) / (2 * a), (-b -
   sqrt(D)) / (2 * a)
28.                 conn.sendall(f'Roots are: {x1},
   {x2}\n'.encode())
29.             elif D == 0:
30.                 x = (-b) / (2 * a)
31.                 conn.sendall(f'Root is: {x}\n'.encode())
32.             else:
33.                 conn.sendall(f'No real roots found.'.encode())
```

Файл task2_client.py

```
import argparse
import socket

parser = argparse.ArgumentParser(description='Task 2 Client')
parser.add_argument('--host', help='Hostname/IP to connect',
default='127.0.0.1')
parser.add_argument('--port', help='Port to connect',
default=1337)
parser.add_argument('-b', '--buffer', help='Connection buffer
size', type=int, default=1024)
args = parser.parse_args()

    inp = input('Enter quadratic equation coefficients (a, b, c)
divided by space. Such as:\n1 2 1\n>> ')

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
conn:
        conn.connect((args.host, args.port))
        conn.sendall(inp.encode())

        received = conn.recv(args.buffer).decode()
        print(received, end='')
```

Задание 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.

Файл task3_server.py

```
import argparse
import json
import socket
from email.parser import Parser
from functools import lru_cache
from urllib.parse import parse_qs, urlparse


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

```
            self.body = body


    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) -> dict:
            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


    class MyHTTPServer:
        MAX_HEADERS = 100
        MAX_LINE = 64 * 1024

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

            self._host = host
            self._port = port
            self._server_name = server_name

        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_line(self, rfile):
            raw = rfile.readline(self.MAX_LINE + 1)
            if len(raw) > self.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 words

        def parse_request(self, conn):
            rfile = conn.makefile('rb')
            method, target, ver = self.parse_request_line(rfile)
            headers = self.parse_headers(rfile)
            host = headers.get('Host')
            if not host:
```

```
114.                 raise Exception('Bad request')
115.             if host not in (self._server_name,
116.
   f'{self._server_name}:{self._port}'):
117.                 raise Exception('Not found')
118.             return Request(method, target, ver, headers, rfile)
119.
120.         def parse_headers(self, rfile):
121.             headers = []
122.             while True:
123.                 line = rfile.readline(self.MAX_LINE + 1)
124.                 if len(line) > self.MAX_LINE:
125.                     raise Exception('Header line is too long')
126.
127.                 if line in (b'\r\n', b'\n', b''):
128.                     break
129.
130.                 headers.append(line)
131.                 if len(headers) > self.MAX_HEADERS:
132.                     raise Exception('Too many headers')
133.
134.             sheaders = b''.join(headers).decode('iso-8859-1')
135.             return Parser().parsestr(sheaders)
136.
137.         def handle_request(self, req: Request) -> Response:
138.             if req.path == '/marks' and req.method == 'POST':
139.                 return self.handle_post_marks(req)
140.
141.             if req.path == '/marks' and req.method == 'GET':
142.                 return self.handle_get_marks(req)
143.
144.             raise Exception('Not found')
145.
146.         def handle_get_marks(self, req: Request) -> Response:
147.             accept = req.headers.get('Accept')
148.             if 'text/html' in accept:
149.                 contentType = 'text/html; charset=utf-8'
150.                 body = '<html><head></head><body>'
151.                 body += f'<div>Оценки группы по дисциплине (в
   базе студентов: {len(self._marks)})</div>'
152.                 body += '<ul>'
153.                 for student in self._marks:
154.                     body += '<li>'
155.                     body += student
156.                     body += '<ul>'
157.                     for task in self._marks[student]:
158.                         body += f'<li>{task}:
   {self._marks[student][task]}</li>'
159.                     body += '</ul>'
160.                     body += '</li>'
161.                 body += '</ul>'
```

```
                body += '</body></html>'

            elif 'application/json' in accept:
                contentType = 'application/json; charset=utf-8'
                body = json.dumps(self._marks)

            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 handle_post_marks(self, req: Request) -> Response:
            student = req.query['student'][0]
            task = req.query['task'][0]
            mark = req.query['mark'][0]

            if student not in self._marks:
                self._marks[student] = {}
            if task not in self._marks[student]:
                self._marks[student][task] = {}
            self._marks[student][task] = mark

            return Response(204, 'Created')

        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: HTTPError):
            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)


    if __name__ == '__main__':
        parser = argparse.ArgumentParser(description='Task 3
  Server')
        parser.add_argument('--name', help='Host of the server',
  default='127.0.0.1')
        parser.add_argument('--host', help='Hostname/IP to
  bind', default='127.0.0.1')
        parser.add_argument('--port', help='Port to bind',
  default=1337)
        parser.add_argument('-b', '--buffer', help='Connection
  buffer size', type=int, default=1024)
        args = parser.parse_args()

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

Результат выполнения GET запроса:

Оценки группы по дисциплине (в базе студентов: 2)

- Комаров Георгий
  - ЛР1: 4
  - ЛР2: 5
- Фамилия Имя
  - ЛР2: 5

Задание 4.

Реализовать двухпользовательский или многопользовательский чат.

Файл task4_server.py

```
import argparse
import socket
from threading import *


class ChatThread(Thread):
    def __init__(self, conn, sender):
        self.conn = conn
        self.sender = sender

            super().__init__()

         def run(self):
            while True:
                try:
                    message = self.conn.recv(args.buffer)
                    message = self.sender + ': ' +
message.decode()
                    broadcast(message, self.sender)
                except:
                    self.conn.close()


    def broadcast(message, sender):
        for client in clients:
            if client.sender != sender:
                try:
                    client.conn.send(message.encode())
                except:
                    client.conn.close()
                    clients.remove(client)


    parser = argparse.ArgumentParser(description='Task 4
Server')
    parser.add_argument('--host', help='Hostname/IP to bind',
default='127.0.0.1')
    parser.add_argument('--port', help='Port to bind',
default=1337)
    parser.add_argument('-b', '--buffer', help='Connection
buffer size', type=int, default=1024)
    args = parser.parse_args()

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
sock:
```

```
        sock.bind((args.host, args.port))
        sock.listen()

        clients = []
        while True:
            try:
                conn, addr = sock.accept()
                client_host, client_port = addr

                new_thread = ChatThread(conn, str(client_port))
                clients.append(new_thread)
                new_thread.start()

            except KeyboardInterrupt:
                sock.close()
                for client in clients:
                    client.conn.close()

```

Вывод

В результате выполненной работы были изучены основы клиент-серверного взаимодействия, работа с сокетами, протокол HTTP, потоки в Python.