Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
Факультет инфокоммуникационных технологий

Лабораторная работа №1

«Работа с сокетами»

по дисциплине

«Web-программирование»

Выполнил:

студент III курса ФИКТ

группы К33402

Ф.И.О. Кондрашов Егор Юрьевич

Проверил:

Говоров А. И.

Санкт-Петербург

2021

Цель работы:

Реализовать клиентскую и серверную часть четырёх программ на Python, использующих сокеты.

Выполнение работы:

Задание 1:

клиент:

```
import socket


HOST, PORT = "localhost", 9999

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((HOST, PORT))
    sock.sendall(bytes("Hello, server" + "\n", "utf-8"))

    received = str(sock.recv(1024), "utf-8")
    print(f"Received data: {received}")
```

сервер:

```
import socketserver


class MyTCPHandler(socketserver.BaseRequestHandler):

    def handle(self):
        self.data = self.request.recv(1024).strip()
        print(f"Received data: {self.data.decode()}")
        if self.data.decode() == "Hello, server":
            self.request.sendall(b"Hello, client")
        else:
            self.request.sendall(b"Try again")


if __name__ == "__main__":
    HOST, PORT = "localhost", 9999

      with socketserver.TCPServer((HOST, PORT), MyTCPHandler) as
server:
        server.serve_forever()
```

Задание 2:

сервер:

```
import socketserver


class MyTCPHandler(socketserver.BaseRequestHandler):

    def calculate_area(self, base: float, altitude: float) ->
float:
        """Метод для расчёта площадь параллелограмма
        по заданной стороне и высоте, проведённой к ней"""
        area: float = base * altitude
        return area

    def handle(self):
        self.data = self.request.recv(1024).strip()
        print(f"Received data: {self.data.decode()}")
        try:
            base, altitude = self.data.decode().split()
            base = float(base)
            altitude = float(altitude)
            resp = bytes(str(self.calculate_area(
                base, altitude)) + "\n", "utf-8")
        except ValueError:
            resp = bytes(
                "Необходимо ввести сторону и высоту параллелограмма
через пробел",
                "utf-8")
        self.request.sendall(resp)


if __name__ == "__main__":
    HOST, PORT = "localhost", 9999

    with socketserver.TCPServer((HOST, PORT), MyTCPHandler) as
server:
        server.serve_forever()
```

Клиент:

```
import socket
```

```
HOST, PORT = "localhost", 9999

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((HOST, PORT))
    params = input("Введите сторону и высоту параллелограмма: ")
    sock.sendall(bytes(params, "utf-8"))

    received = str(sock.recv(1024), "utf-8")
    print(f"Received data: {received}")
```

Задание 3.

Сервер:

```
import socketserver


class MyTCPHandler(socketserver.BaseRequestHandler):

    def handle(self):
        self.data = self.request.recv(1024).strip().decode()
        split_data = self.data.split()
        method = split_data[0]
        path = split_data[1]
        if method == "GET" and path == "/index.html":
            with open("index.html", "rb") as f:
                resp = f.read()
        else:
            resp = b"Unsupported request method or path\n"
        self.request.sendall(resp)


if __name__ == "__main__":
    HOST, PORT = "localhost", 9999

    with socketserver.TCPServer((HOST, PORT), MyTCPHandler) as
server:
        server.serve_forever()
```

Клиент:

```
import socket
```

```
HOST, PORT = "localhost", 9999

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((HOST, PORT))
    sock.sendall((
        b"GET /index.html HTTP/1.1\r\n" +
          bytes(f"Host: {HOST}\r\nAccept: text/html\r\nConnection:
close\r\n\r\n",
                "utf-8")))
    received = str(sock.recv(1024), "utf-8")
    print(received)
```

Файл index.html:

```
<!DOCTYPE html>
<html>

<head>
    <title>Index page</title>
</head>

<body>

    <h1>Hello World</h1>

</body>

</html>
```

Задание 4.

Сервер:

```
import socketserver


class ThreadedTCPServer(socketserver.ThreadingTCPServer):

    def __init__(self, server_address, request_handler_class):
            super().__init__(server_address, request_handler_class,
True)
        print("Server started")
```

```
            self.receivers = set()

        def add_receiver(self, receiver):
            print("Client connected")
            self.receivers.add(receiver)

        def send_message(self, source, data):
            for receiver in self.receivers:
                if receiver.token != source.token:
                    receiver.request.sendall(data)

        def remove_receiver(self, receiver):
            self.receivers.remove(receiver)


    class ThreadedTCPRequestHandler(socketserver.BaseRequestHandler):

        RECEIVER = 0
        SENDER = 1
        kind = None
        token: str = ""

        def handle(self):
            while True:
                self.data = self.request.recv(1024).strip()
                if self.data:
                    print(f"Received data: {self.data.decode()}")

                    if b"Kind" in self.data:

                        if b"Kind: receiver" in self.data:
                            self.server.add_receiver(self)
                            self.kind = self.RECEIVER
                        elif b"Kind: sender" in self.data:
                            self.kind = self.SENDER

                        token = self.data.decode(
                        )[self.data.decode().find("Token")+6:]
                        self.token = token

                    else:
                        self.server.send_message(self, self.data)
```

```
        def finish(self):
            if self.kind == self.RECEIVER:
                self.server.remove_receiver(self)
            super().finish()


    if __name__ == "__main__":
        HOST, PORT = "localhost", 9999

                    server     =     ThreadedTCPServer((HOST,     PORT),
ThreadedTCPRequestHandler)
        server.serve_forever()
```

Клиент:

```
    import socket
    import threading
    from string import ascii_letters, digits

    from secrets import choice


    HOST, PORT = "localhost", 9999


    def receive_messages(token: str) -> None:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.connect((HOST, PORT))
            connect_msg = "Kind: receiver\nToken: " + token + "\n"
            sock.sendall(bytes(connect_msg, "utf-8"))
            while True:
                received = sock.recv(1024)
                if received:
                            print("Received message: " + str(received,
"utf-8"))


    def send_messages(token: str) -> None:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.connect((HOST, PORT))
            connect_msg = "Kind: sender\nToken: " + token + "\n"
            sock.sendall(bytes(connect_msg, "utf-8"))
            while True:
```

```
                inp = input()
                sock.sendall(bytes(inp, "utf-8"))


    if __name__ == "__main__":
        print("To send a message, type it into the terminal and press
enter")
        token = ''.join(choice(
                ascii_letters + digits
        ) for _ in range(16))
        recv = threading.Thread(target=receive_messages, args=(token,))
        recv.start()
        send = threading.Thread(target=send_messages, args=(token,))
        send.start()
```

Задание 3 (обновлённое):

```
    import socket

    from email.parser import Parser
    from functools import lru_cache
    from urllib.parse import parse_qs, urlparse


    MAX_LINE = 64*1024
    MAX_HEADERS = 100


    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)

        def body(self):
            size = self.headers.get('Content-Length')
            if not size:
                return None
            return self.rfile.read(size)


    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


    class MyHTTPServer:

        def __init__(self, host, port, server_name):
            self._host = host
            self._port = port
            self._server_name = server_name
            self.data: "dict[str, list[str]]" = {}

        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:
                print("\nServing client")
                print("Parsing request")
                req = self.parse_request(conn)
                print("Handling request")
                resp = self.handle_request(req)
                print("Sending response")
                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)
            host = headers.get('Host')
            if not host or host not in (self._server_name,

f'{self._server_name}:{self._port}'):
                raise HTTPError(400, 'Bad request')
            return Request(method, target, ver, headers, rfile)
```

```
    def parse_headers(self, rfile):
        print("Parsing headers")
        headers = []
        while True:
            line = rfile.readline(MAX_LINE + 1)
            if len(line) > MAX_LINE:
                raise HTTPError(400, 'Header line is too long')

            if line in (b'\r\n', b'\n', b''):
                # завершаем чтение заголовков
                break

            headers.append(line)
            if len(headers) > MAX_HEADERS:
                raise HTTPError(400, 'Too many headers')
        sheaders = b''.join(headers).decode('iso-8859-1')
        return Parser().parsestr(sheaders)

    def parse_request_line(self, rfile):
        print("Parsing request line")
        raw = rfile.readline(MAX_LINE + 1)
        if len(raw) > MAX_LINE:
            raise HTTPError(400, '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 HTTPError(400, 'Malformed request line')

        method, target, ver = words
        print(f"Target: {target}")
        if ver != 'HTTP/1.1':
            raise HTTPError(400, 'Unexpected HTTP version')
        return method, target, ver

    def handle_request(self, req):
        if req.path == '/subjects' and req.method == 'POST':
            return self.handle_post_subject(req)

        if req.path == '/subjects' and req.method == 'GET':
            return self.handle_get_subjects(req)
```

```
            if req.path.startswith('/subjects/'):
                subject_name = req.path[len('/subjects/'):]
                if subject_name in self.data:
                    return self.handle_get_subject(req, subject_name)
                else:
                    raise HTTPError("404", "Not found")

            raise HTTPError(404, 'Not found')

        def handle_post_subject(self, request: Request) -> Response:
            """Сохраняет оценку по предмету"""
            print("Handling create subject")
            try:
                subject_name: str = request.query["subject"][0]
                grade: str = request.query["grade"][0]
            except KeyError:
                raise HTTPError("400", "Bad request")
            if subject_name in self.data:
                self.data[subject_name].append(grade)
            else:
                subject_lst = []
                subject_lst.append(grade)
                self.data[subject_name] = subject_lst
            return Response(201, "Created")

        def handle_get_subjects(self, request: Request) -> Response:
            """Возвращает список оценок по всем предметам"""
            print("Handling list request")
            content_type = 'text/html; charset=utf-8'
                           body   =   '<html><head>Список   оценок   по
предметам</head><body>'

            for subject in self.data:
                body += f"<h2>{subject}</h2>"
                for grade in self.data[subject]:
                    body += f"<p>{grade}</p>"
            body += '</body></html>'

            body = body.encode('utf-8')

            headers = [('Content-Type', content_type),
                       ('Content-Length', len(body))]
```

```
            return Response(200, 'OK', headers, body)

        def handle_get_subject(self, request: Request,
                               subject_name: str) -> Response:
            print("Handling retrieve request")
            """Возвращает список оценок по определённому предмету"""
            content_type = 'text/html; charset=utf-8'
                        body   =   '<html><head>Список   оценок   по
предметам</head><body>'

            body += f"<h2>{subject_name}</h2>"
            for grade in self.data[subject_name]:
                body += f"<p>{grade}</p>"
            body += '</body></html>'

            body = body.encode('utf-8')

            headers = [('Content-Type', content_type),
                       ('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()

        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)


    if __name__ == '__main__':
        host = "localhost"
        port = 9999
        name = "localhost"
        serv = MyHTTPServer(host, port, name)
        try:
            serv.serve_forever()
        except KeyboardInterrupt:
            pass
```

Вывод:

В ходе работы были написаны 4 программы на Python, использующие сокеты для коммуникации между клиентом и сервером.