НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет инфокоммуникационных технологий

Дисциплина “Web программирование”

Отчет к практической работе № 1
“Сокеты”

Выполнил:
студент группы К33421 Безгин Алексей Геннадьевич

Принял:
преподаватель Говоров Антон Игоревич

Санкт-Петербург
2023 год

Ход работы

Задача №1

Реализовать клиентскую и серверную часть приложения. Клиент отсылает серверу сообщение “Hello, server”. Сообщение должно отразиться на сервере.
Сервер в ответ отсылает клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.
Обязательно использовать библиотеку socket.
Реализовать с использованием протокола UDP.

Скриншоты решения с подробным описанием в комментариях приведены ниже.

Сервер:

```python
import socket

if __name__ == "__main__":
    # Задаем IP-адрес и порт сервера, к которому мы хотим подключиться.
    server_ip = '127.0.0.1'
    server_port = 7000

    # Создаем UDP сокет (SOCK_DGRAM) и устанавливаем соединение с сервером.
    conn = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    conn.connect((server_ip, server_port))

    # Отправляем сообщение "Hello, server" серверу.
    conn.send(b"Hello, server")

    # Получаем данные от сервера и адрес, с которого пришли данные.
    data, addr = conn.recvfrom(1024)

    # Выводим полученные данные (раскодированные из байтовой строки) на экран.
    print(data.decode("utf-8"))
```

Клиент:

```python
import socket

if __name__ == "__main__":
    # Задаем IP-адрес и порт сервера, к которому мы хотим подключиться.
    server_ip = '127.0.0.1'
    server_port = 7000

    # Создаем UDP сокет (SOCK_DGRAM) и устанавливаем соединение с сервером.
    conn = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    conn.connect((server_ip, server_port))

    # Отправляем сообщение "Hello, server" серверу.
    conn.send(b"Hello, server")

    # Получаем данные от сервера и адрес, с которого пришли данные.
    data, addr = conn.recvfrom(1024)

    # Выводим полученные данные (раскодированные из байтовой строки) на экран.
    print(data.decode("utf-8"))
```

Запуск и демонстрация процесса работы на скриншотах терминала:

Задача №2

Реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры которой вводятся с клавиатуры.
Сервер обрабатывает полученные данные и возвращает результат клиенту.

Мой вариант, согласно списку группы, – теорема Пифагора.
Скриншоты решения с подробным описанием в комментариях приведены ниже.

Сервер:

```python
import socket
from math import sqrt


2 usages
def pyth_theorem(cath1, cath2=None, hyp=None):
    if not (cath1 or hyp) or (hyp and cath1 >= hyp):
        return "Invalid input"
    if not cath2:
        return sqrt(hyp ** 2 - cath1 ** 2)
    if not hyp:
        return sqrt(cath1 ** 2 + cath2 ** 2)
    return "Something went wrong"


```

```python

if __name__ == "__main__":
    server_ip = '127.0.0.1'
    server_port = 6998

    conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    conn.bind((server_ip, server_port))
    conn.listen()

    conn, addr = conn.accept()
    while True:
        data = conn.recv(1024).decode("utf-8")

        if not data:
            continue

        operation, a, b = data.split(',')
        a, b = float(a), float(b)

        if operation == "hyp":
            result = pyth_theorem(cath1=a, cath2=b)
        elif operation == "cath":
            result = pyth_theorem(cath1=a, hyp=b)
        else:
            result = "Invalid operation"

        conn.send(bytes(str(result), encoding="utf-8"))
```

Клиент:

```python
import socket

if __name__ == "__main__":
    # Задаем IP-адрес и порт сервера, к которому будем подключаться.
    server_ip = '127.0.0.1'
    server_port = 6998

    # Создаем TCP сокет (SOCK_STREAM) и устанавливаем соединение с сервером.
    conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    conn.connect((server_ip, server_port))

    # Запрашиваем у пользователя выбор операции (cath/hyp или c/h).
    operation = input("Выберите опцию (cath/hyp или c/h): ")

    # Проверяем, что пользователь ввел корректную операцию.
    while operation not in ["cath", "c", "hyp", "h"]:
        operation = input("Неверная операция. Попробуйте снова: ")

    # Преобразуем сокращенные операции в полные.
    if operation in ("c", "h"):
        operation = "cath" if operation == "c" else "hyp"

    # Вводим значения сторон треугольника с обработкой ошибок.
    while True:
        if operation == "cath":
            a = input("Введите катет: ")
            b = input("Введите гипотенузу: ")
        else:
            a = input("Введите катет: ")
            b = input("Введите катет: ")
        try:
            a = float(a)
            b = float(b)
            break
        except ValueError:
            print("Неверный ввод. Попробуйте снова.")

    # Отправляем операцию и значения сторон на сервер.
    conn.send(bytes(f"{operation}, {a}, {b}", encoding="utf-8"))

    # Получаем и выводим результат от сервера.
    result = conn.recv(1024).decode("utf-8")
    print(f"Результат: {result}")
```

Запуск и демонстрация процесса работы (см. на следующей странице):

```
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task2$ python3 client.py
Choose option (cath/hyp or c/h): c
Enter cathetus: 3
Enter hypotenuse: 5
Result: 4.0
```

```
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task2$ python3 client.py
Choose option (cath/hyp or c/h): h
Enter cathetus: 3
Enter cathetus: 4
Result: 5.0
```

```
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task2$ python3 client.py
Choose option (cath/hyp or c/h): c
Enter cathetus: -43
Enter hypotenuse: -234
Result: Invalid input
```

Задача №3

Реализовать серверную часть приложения. Клиент подключается к серверу. В ответ клиент получает http-сообщение, содержащее html-страницу, которую сервер подгружает из файла index.html.

Сервер:

```python
import socket

if __name__ == "__main__":
    # Загружаем содержимое файла "index.html".
    content = open("index.html").read()

    # Задаем IP-адрес и порт сервера, на котором будем слушать входящие соединения.
    server_ip = "127.0.0.1"
    server_port = 7002

    # Создаем TCP сокет (SOCK_STREAM), связываем его с указанным IP-адресом и портом, и начинаем слушать входящие соединения.
    conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    conn.bind((server_ip, server_port))
    conn.listen()

    while True:
        # Принимаем входящее соединение и получаем сокет клиента.
        client_socket = conn.accept()[0]

        # Получаем данные от клиента.
        data = client_socket.recv(1024)

        # Создаем HTTP-ответ с кодом 200 OK и вставляем содержимое "index.html" в ответ.
        response = "HTTP/1.1 200 OK\nContent-Type: text/html; charset=utf-8\n\n" + content

        # Отправляем ответ клиенту.
        client_socket.send(response.encode())

        # Закрываем соединение с клиентом.
        client_socket.close()
```

Результат запуска сервера при переходе по нужному сокету:

```
Terminal:  Local ×  Local (2) ×  Local (3) ×  +
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task2$ cd ../task3
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task3$ python3 server.py
```

Hello, Dear client!

PS. я решил выводить “Dear, client!”

Задача №4

Реализовать двухпользовательский или многопользовательский чат. Реализация многопользовательского чата позволяет получить максимальное количество баллов.

Сервер:

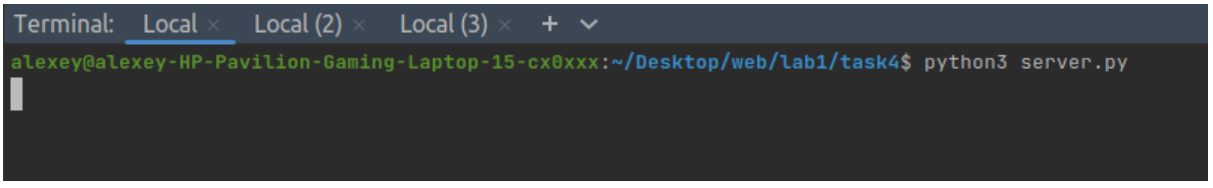
```python
import socket
import threading

if __name__ == "__main__":
    # Создаем список для хранения клиентских соединений и мьютекс для синхронизации доступа к списку.
    clients_list = []
    lock = threading.Lock()

    # Задаем IP-адрес и порт сервера, на котором будем слушать входящие соединения.
    server_ip = "127.0.0.1"
    server_port = 7003

    # Функция для обработки отправки сообщений клиентам.
    1 usage
    def send_message_handler(client, address, clients_list):
        while True:
            # Получаем данные от клиента.
            data = client.recv(16384).decode("utf-8")
            if not data:
                break
            # Отправляем сообщение от клиента всем остальным клиентам.
            with lock:
                for c in clients_list:
                    if c != client:
                        c.send(f"{address[1]}: {data}".encode("utf-8"))
```

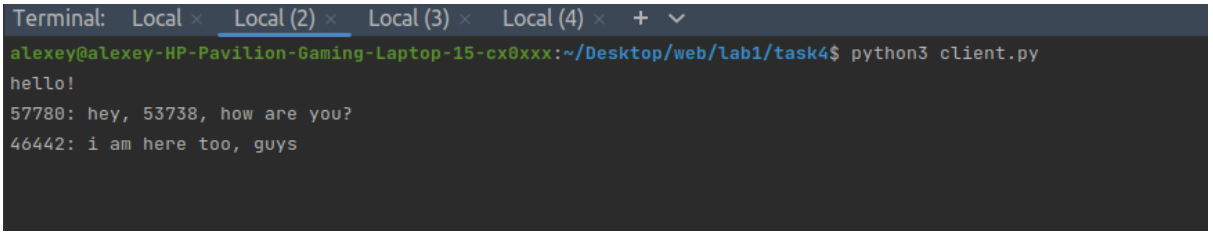
```python

    # Создаем сокет и настраиваем опции для переиспользования адреса.
    conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,  value: 1)
    conn.bind((server_ip, server_port))
    conn.listen(10)

    while True:
        try:
            # Принимаем входящее соединение от клиента.
            client, address = conn.accept()
            # Добавляем клиентский сокет в список активных клиентов.
            with lock:
                clients_list.append(client)
            # Запускаем отдельный поток для обработки отправки сообщений клиентам.
            threading.Thread(target=send_message_handler, args=(client, address, clients_list)).start()
        except KeyboardInterrupt:
            # Закрываем серверное соединение при завершении программы.
            conn.close()
            break
```

Клиент:

```python
import socket
import threading

# Функция обработки входящих сообщений от сервера
1 usage
def receive_message_handler(client):
    while True:
        # Получаем сообщение от сервера и декодируем его из байтовой строки
        message_obj = client.recv(16384).decode("utf-8")
        # Выводим полученное сообщение на экран
        print(message_obj)

if __name__ == "__main__":
    # Задаем IP-адрес и порт сервера, к которому будем подключаться.
    server_ip = "127.0.0.1"
    server_port = 7003

    # Создаем TCP сокет (SOCK_STREAM) и устанавливаем соединение с сервером.
    conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    conn.connect((server_ip, server_port))

    # Создаем отдельный поток для приема сообщений от сервера
    receive_thread = threading.Thread(target=receive_message_handler, args=(conn,))
    receive_thread.start()

    while True:
        try:
            # Читаем сообщение с клавиатуры
            message = input()
            # Отправляем сообщение серверу, предварительно кодируя его в байты
            conn.send(message.encode("utf-8"))
        except KeyboardInterrupt:
            # Закрываем соединение при нажатии Ctrl+C и завершаем программу
            conn.close()
            break

```

Демонстрация работы с 3 пользователями:





```
Terminal:  Local  Local (2)  Local (3)  Local (4)  +
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task4$ python3 client.py
53738: hello!
hey, 53738, how are you?
46442: i am here too, guys
```

```
Terminal:  Local  Local (2)  Local (3)  Local (4)  +
alexey@alexey-HP-Pavilion-Gaming-Laptop-15-cx0xxx:~/Desktop/web/lab1/task4$ python3 client.py
53738: hello!
57780: hey, 53738, how are you?
i am here too, guys
```

Для проверки работоспособности можно запустить процесс server.py и имитировать переписку пользователей через новые процессы с client.py.

Задача №5

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

Код реализации:

```python
import socket

# Класс MyHTTPServer представляет HTTP-сервер.
1 usage
class MyHTTPServer:
    def __init__(self, server_ip, server_port, num_listen=1):
        # Создаем сокет и настраиваем его параметры.
        self.conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, value: 1)
        self.conn.bind((host, port))  # Привязываем сокет к указанному IP и порту.
        self.conn.listen(num_listen)  # Начинаем слушать входящие соединения.
        self.grades = {}  # Словарь для хранения оценок.

    # Основной цикл сервера для обслуживания клиентов.
    1 usage
    def serve_forever(self):
        while True:
            client, addr = self.conn.accept()  # Принимаем входящее соединение от клиента.
            self.serve_client(client)

    # Обслуживание клиента.
    1 usage
    def serve_client(self, client):
        data = client.recv(2 * 16384).decode(encoding="utf-8", errors="ignore")  # Получаем данные от клиента.
        self.parse_request(client, data)  # Анализируем запрос.
```

```python

    # Анализ HTTP-запроса.
    1 usage
    def parse_request(self, client, data):
        lines = data.split("\n")
        method, url, version = lines[0].split()

        # Парсинг параметров запроса в зависимости от метода (GET или POST).
        if method == "GET":
            params = (
                {p.split("=")[0]: p.split("=")[1] for p in url.split("?")[1].split("&")}
                if "?" in url
                else None
            )
        elif method == "POST":
            # В POST-запросе параметры находятся в теле запроса.
            body = data.split("\n")[-1]
            params = {p.split("=")[0]: p.split("=")[1] for p in body.split("&")}
        else:
            params = None

        # Обработка запроса в зависимости от метода.
        self.handle_request(client, method, params)

```

```python
    # Обработка HTTP-запроса и отправка ответа.
    1 usage
    def handle_request(self, client, method, params):
        if method == "GET":
            self.send_response(client, code: 200, reason: "OK", self.grades_to_html())  # Отправляем HTML-страницу с оценками.
        elif method == "POST":
            discipline = params.get("discipline")
            grade = params.get("grade")
            self.grades[discipline] = grade  # Сохраняем новую оценку.
            self.send_response(client, code: 200, reason: "OK", body: "Содержимое сохранено!")  # Отправляем сообщение об успешном сохранении.
        else:
            self.send_response(client, code: 404, reason: "Not Found", body: "Некорректный метод, попробуйте снова.")  # Отправляем ошибку.

    # Отправка HTTP-ответа клиенту.
    3 usages
    def send_response(self, client, code, reason, body):
        response = f"HTTP/1.1 {code} {reason}\nContent-Type: text/html\n\n{body}"
        client.send(response.encode("utf-8"))
        client.close()

    # Генерация HTML-страницы на основе данных о оценках.
    1 usage
    def grades_to_html(self):
        page = (
            f"<html><body><ul>"
            f"{''.join([f'<li>{discipline}: {grade}' for discipline, grade in self.grades.items()])}"
            f"</ul></body></html>"
        )
        return page

```

```python

if __name__ == "__main__":
    host = "127.0.0.1"
    port = 8879
    server = MyHTTPServer(host, port)
    try:
        server.serve_forever()  # Запуск сервера для бесконечного обслуживания клиентов.
    except KeyboardInterrupt:
        server.conn.close()  # Закрытие серверного соединения при завершении программы.

```

Для демонстрации работы поднимем сервер и будем отравлять на него POST и GET запросы с помощью утилиты curl.

Демонстрация работы с пояснением:

1. Поднимаем сервер в консоли:

2. Обращаемся к серверу по нужному сокету с помощью curl:

В начале отправим GET-запрос и получим пустой HTML. Все логично

Далее сделаем POST-запрос и добавим сведения о первом предмете, проверим, что они уже на сервере с помощью нового GET-запроса.

Повторим процедуру, чтобы убедиться, что сервер может хранить информацию о нескольких предметах единовременно.