Отчет по лабораторной работе №1: Работа с сокетами

Цель работы:

Целью данной лабораторной работы является овладение практическими навыками и умениями реализации web-серверов и использования сокетов в Python.

Оборудование и программное обеспечение:

- **Оборудование**: компьютерный класс.
- **Программное обеспечение**: Python 2.7-3.6, библиотеки Python: `sys`, `socket`.

Теоретические сведения

Клиент-серверное взаимодействие

В основе работы web-приложений лежит модель взаимодействия клиент-сервер, которая позволяет разделять функционал и вычислительную нагрузку между клиентскими и серверными приложениями. Клиент (например, браузер) отправляет запросы к серверу (например, HTTP-серверу), который обрабатывает эти запросы и возвращает соответствующие ответы.

Протоколы

Клиент и сервер взаимодействуют посредством различных сетевых протоколов, таких как HTTP, FTP и другие. HTTP-запросы содержат методы, которые указывают серверу, как обрабатывать запрос, а HTTP-ответы содержат информацию о статусе выполнения запроса.

Модель TCP/IP

Стек TCP/IP обеспечивает объединение пакетных подсетей через шлюзы. Каждое сообщение, отправляемое по сети, делится на фрагменты, которые снабжаются адресами отправителя и получателя. Это позволяет маршрутизаторам определять путь следования пакетов.

Инкапсуляция и декапсуляция данных

Данные, передаваемые по сети, проходят процесс инкапсуляции (добавление заголовков на каждом уровне модели передачи данных) и декапсуляции (удаление заголовков на принимающей стороне).

IP-адреса и порты

Каждый компьютер в сети имеет уникальный IP-адрес и номер порта, который соответствует конкретному приложению. Например, веб-сервер использует порт 80, а почтовый сервер — порт 25.

Практическая часть

Реализация сервера

В ходе лабораторной работы был реализован простой HTTP-сервер на Python с использованием библиотеки `socket`. Сервер обрабатывает GET и POST запросы.

Код сервера:

```python
import socket
import sys
from urllib.parse import urlparse, parse_qs

class MyHTTPServer:
    def __init__(self, host='localhost', port=8080):
        self.host = host
        self.port = port
        self.grades = {}

    def serve_forever(self):
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
            server_socket.bind((self.host, self.port))
            server_socket.listen()
            print(f'Server running on http://{self.host}:{self.port}')
            while True:
                client_socket, _ = server_socket.accept()
                self.serve_client(client_socket)

    def serve_client(self, client_socket):
        with client_socket:
            request = client_socket.recv(1024).decode()
            print(f'Received request:\n{request}')
            request_line, headers = self.parse_request(request)
            self.handle_request(request_line, headers, client_socket)

    def parse_request(self, request):
        lines = request.splitlines()
        request_line = lines[0]
        headers = self.parse_headers(lines[1:])
        return request_line, headers

    def parse_headers(self, lines):
        headers = {}
        for line in lines:
            if line == '':
                break
            key, value = line.split(': ', 1)
            headers[key] = value
        return headers

    def handle_request(self, request_line, headers, client_socket):
        method, url, _ = request_line.split()
```

```
        parsed_url = urlparse(url)
        path = parsed_url.path
        query = parse_qs(parsed_url.query)

        if method == 'POST':
            self.handle_post(query)
            self.send_response(client_socket, 200, 'OK', 'Data received')
        elif method == 'GET':
            response_body = self.handle_get()
            self.send_response(client_socket, 200, 'OK', response_body)
        else:
            self.send_response(client_socket, 405, 'Method Not Allowed', '')

    def handle_post(self, query):
        discipline = query.get('discipline', [''])[0]
        grade = query.get('grade', [''])[0]
        if discipline and grade:
            self.grades[discipline] = grade

    def handle_get(self):
        html_content = '<html><body><h1>Grades</h1><ul>'
        for discipline, grade in self.grades.items():
            html_content += f'<li>{discipline}: {grade}</li>'
        html_content += '</ul></body></html>'
        return html_content

    def send_response(self, client_socket, status_code, reason, body):
        response_line = f'HTTP/1.1 {status_code} {reason}\r\n'
        headers = 'Content-Type: text/html\r\n'
        headers += f'Content-Length: {len(body)}\r\n'
        headers += '\r\n'    End of headers
        response = response_line + headers + body
        client_socket.sendall(response.encode())

if __name__ == '__main__':
    host = 'localhost'
    port = 8080
    serv = MyHTTPServer(host, port)
    try:
        serv.serve_forever()
    except KeyboardInterrupt:
        print("\nServer stopped.")
```

Реализация клиента

Для взаимодействия с сервером был также реализован клиент, который отправляет POST-запросы для записи оценок и GET-запросы для получения списка оценок.

Код клиента:

```python
import requests

def send_post_request():
    url = 'http://localhost:8080'
    data = {
        'discipline': 'math',
        'grade': '5'
    }
    response = requests.post(url, data=data)
    print(response.text)

def get_grades():
    url = 'http://localhost:8080'
    response = requests.get(url)
    print(response.text)

if __name__ == '__main__':
    send_post_request()
    get_grades()
```

Заключение

В ходе лабораторной работы была успешно реализована модель клиент-серверного взаимодействия с использованием сокетов. Полученные знания и навыки позволят в дальнейшем разрабатывать более сложные сетевые приложения и понимать принципы работы сетевых протоколов.