

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«Київський політехнічний інститут ім. І. Сікорського»**

**Інститут Прикладного Системного Аналізу**

**Кафедра Системного Проектування**

Лабораторна робота №5

З дисципліни «Паралельні обчислення»

Виконав:

Ст. гр. ДА-12

Кракович Павло

**Лабораторна робота №5**

**Розробка HTTP-сервера для хостингу статичного веб-сайту.**

**Мета роботи:** Вивчити принцип роботи http веб-серверів. Створити простий веб-сервер длябагатопотокової обробки GET-запитів. Засвоїти інструментарій для навантажувальноготестування.

**Завдання:**

1. Спроектувати веб сервер для хостингу статичного контенту. Можна обиратидовільну технологією, вимогою є програмування на транспортному (четвертому)рівні стеку OSI (з допомогою сокетів).
2. Веб-сервер повинен опрацьовувати http запити з багатьох клієнтів. Реалізуємо GET відповідно до протоколу HTTP 1.1. RFC9112 RFC9110Root-запит веде на index.html (приклади запитів “GET http://localhost:8080”, “GET http://localhost:8080/second\_page.html”).
3. Захостити щонайменше 2 веб-сторінки (текст файлів наведено в Додатку 1, абоможете додати власні).При цьому повинні оброблятися неіснуючі сторінки. (404 відповідь)
4. Провести навантажувальне тестування з допомогою сервісів jMeter або Locust. Визначити точку відмови в кількості паралельних клієнтів.
5. Зробити висновки, припустити яким чином можна зробити сервер більшефективним і підвищити його здатність тримати навантаження.
6. (\*) Додаткове завдання на плюс 1 бал – оптимізувати веб-сервер, зробивши йогонеблокуючим.

**Опис розробленого застосунку**

Код лабораторної роботи представляє простий веб-сервер, який використовує многопоточність для обробки одночасних запитів від клієнтів. Основні етапи роботи коду можна описати так:

1. Налаштування сервера: Створюється сокет для комунікації з клієнтами на певному хості та порту. Налаштовується параметр SO\_REUSEADDR, щоб уникнути помилок під час повторного використання адреси. Сокет прослуховується на зазначеному порті.
2. Обробка запитів клієнтів: У цьому кроці відбувається прийом запитів від клієнтів. Після отримання запиту, відділяється шлях запиту (URI), з якого визначається ім'я файлу. Зчитується вміст файлу (HTML), який повертається клієнту. Якщо файл відсутній або неправильний, сервер повертає сторінку з помилкою 404.
3. Відправлення відповіді клієнту: Формується HTTP-відповідь, яка містить статус відповіді, заголовки та вміст HTML-сторінки. Відповідь відправляється клієнту через сокет.
4. Многопоточна обробка запитів: Використовуючи ThreadPoolExecutor, запити обробляються паралельно у фонових потоках. Це дозволяє серверу ефективно обробляти багато запитів одночасно та збільшує його продуктивність.

Код використовує старий сайт, який був розроблений для шкільних занять з інформатики. Оскільки він містить стилі, то вони додаються в код сайту, який повертається користувачу.

Лістинг коду:

import socket  
import os  
from concurrent.futures import ThreadPoolExecutor  
  
  
def handle\_request(client\_socket):  
 with client\_socket:  
 request = client\_socket.recv(1024).decode('utf-8')  
 if not request:  
 return  
  
 path = request.split()[1]  
 file\_path = path[1:]  
  
 if file\_path == '':  
 filename = 'index.html'  
 status = "200 OK"  
 elif os.path.exists(file\_path) and os.path.isfile(file\_path):  
 filename = file\_path  
 status = "200 OK"  
 else:  
 filename = '404.html'  
 status = "404 Not Found"  
  
 with open(filename, 'rb') as file:  
 html\_content = file.read()  
  
 css\_filename = 'css/style.css'  
 try:  
 with open(css\_filename, 'rb') as css\_file:  
 css\_content = css\_file.read()  
 except FileNotFoundError:  
 css\_content = b''  
  
 html\_response = html\_content.replace(  
 b'</head>', f'<style>{css\_content.decode("utf-8")}</style></head>'.encode("utf-8")  
 )  
  
 response\_headers = f"HTTP/1.1 {status}\r\nContent-Type: text/html\r\n"  
 response\_headers += f"Content-Length: {len(html\_response)}\r\n"  
 response\_headers += "\r\n"  
  
 client\_socket.sendall(response\_headers.encode('utf-8'))  
 client\_socket.sendall(html\_response)  
  
  
def main():  
 host = 'localhost'  
 port = 8080  
  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as server\_socket:  
 server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 server\_socket.bind((host, port))  
 server\_socket.listen(5)  
 print(f"Server listening on {host}:{port}")  
  
 with ThreadPoolExecutor(max\_workers=50) as executor:  
 while True:  
 client\_socket, client\_address = server\_socket.accept()  
  
 executor.submit(handle\_request, client\_socket)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Навантажувальне тестування за допомогою сервісу Locust**

****

Під час проведення стрес-тестів не вийшло визначити точку критичного навантаження на сервер, оскільки для його написання був використаний тред-пул, який обмежує максимальну кількість користувачів.

Від зміни кількості користувачів збільшується лише час відповіді, оскільки іншим користувачам потрібно очікувати, щоб отримати доступ до підключення.

Код для тестування:

import random  
from locust import HttpUser, task, between  
  
class WebsiteUser(HttpUser):  
 wait\_time = between(5, 15)  
 @task  
 def index(self):  
 self.client.get("/")  
  
 @task  
 def view\_page(self):  
 pages = ['/kemalia.html', '/fraklia.html', '/kliantys.html']  
 page = self.client.get(pages[random.randint(0, len(pages)-1)])

**Висновок**

У цій лабораторній роботі ми розробили простий HTTP-сервер для хостингу статичного веб-сайту. Використовуючи знання про принципи роботи HTTP-серверів та програмування на транспортному рівні стеку OSI з використанням сокетів, ми створили сервер, який може обробляти багато GET-запитів від клієнтів одночасно.

Процес розробки сервера можна розділити на кілька етапів, включаючи налаштування сервера, обробку запитів клієнтів, відправлення відповіді клієнту та многопоточну обробку запитів. Використання ThreadPoolExecutor дозволяє серверу ефективно обробляти багато запитів одночасно та збільшує його продуктивність.

Після розробки сервера ми провели навантажувальне тестування за допомогою сервісу Locust. В результаті тестування ми встановили, що сервер не досягає критичної точки навантаження, оскільки ми обмежили максимальну кількість користувачів за допомогою тред-пулу. Зміна кількості користувачів впливає лише на час відповіді сервера через очікування інших користувачів на доступ до підключення.

У цілому, лабораторна робота дала можливість краще зрозуміти принципи роботи HTTP-серверів та їх ефективне використання для хостингу веб-сайтів.