

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
 НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу  
Кафедра системного проектування**

**Звіт**

**про виконання практичної роботи №5  
з дисципліни «Паралельні обчислення»**

Виконав:  
студент III курсу, групи ДА-21  
Терещенко Олексій Ігорович

Прийняв:

асистент Яременко В. С.

Київ – 2025

**ПОСТАНОВКА ЗАДАЧІ**

1. Спроектувати веб сервер для хостингу статичного контенту. Можна обирати довільну 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 RFC9110 Root-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, або можете 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 більш ефективним і підвищити його здатність тримати навантаження.
6. (\*) Додаткове завдання на плюс 1 бал – оптимізувати веб-сервер, зробивши його неблокуючим.

**ХІД РОБОТИ**

Створимо сервер для хостингу:

#define WIN32\_LEAN\_AND\_MEAN

#include <winsock2.h>

#include <ws2tcpip.h>

#include <thread>

#include <vector>

#include <queue>

#include <functional>

#include <mutex>

#include <condition\_variable>

#include <fstream>

#include <sstream>

#include <iostream>

#pragma comment(lib, "Ws2\_32.lib")

static const int PORT = 8080;

static const int THREAD\_COUNT = 4;

static const std::string WWW = "static";

std::string get\_type(const std::string& path) {

if (path.size() >= 5 && path.rfind(".html") == path.size() - 5)

return "text/html";

if (path.size() >= 4 && path.rfind(".css") == path.size() - 4)

return "text/css";

if (path.size() >= 3 && path.rfind(".js") == path.size() - 3)

return "application/javascript";

if (path.size() >= 4 && path.rfind(".svg") == path.size() - 4)

return "image/svg+xml";

return "application/octet-stream";

}

class ThreadPool {

public:

ThreadPool(size\_t n) : stop(false) {

for (size\_t i = 0; i < n; ++i)

workers.emplace\_back([this]() {

for (;;) {

std::function<void()> task;

{

std::unique\_lock<std::mutex> lock(mtx);

cv.wait(lock, [this]() { return stop || !tasks.empty(); });

if (stop && tasks.empty()) return;

task = std::move(tasks.front());

tasks.pop();

}

task();

}

});

}

void enqueue(std::function<void()> f) {

{

std::lock\_guard<std::mutex> lock(mtx);

tasks.push(std::move(f));

}

cv.notify\_one();

}

~ThreadPool() {

{

std::lock\_guard<std::mutex> lock(mtx);

stop = true;

}

cv.notify\_all();

for (auto& t : workers) t.join();

}

private:

std::vector<std::thread> workers;

std::queue<std::function<void()>> tasks;

std::mutex mtx;

std::condition\_variable cv;

bool stop;

};

void handle\_client(SOCKET client) {

char buf[4096];

int len = recv(client, buf, sizeof(buf) - 1, 0);

if (len <= 0) { closesocket(client); return; }

buf[len] = '\0';

std::istringstream ss(buf);

std::string method, path, version;

ss >> method >> path >> version;

if (method != "GET") {

const char\* resp =

"HTTP/1.1 405 Method Not Allowed\r\n"

"Connection: close\r\n\r\n";

send(client, resp, (int)strlen(resp), 0);

closesocket(client);

return;

}

if (path == "/" || path.empty()) path = "/index.html";

std::string file\_path = WWW + path;

std::ifstream f(file\_path, std::ios::binary);

std::ostringstream body;

std::string status;

if (f) {

body << f.rdbuf();

status = "200 OK";

}

else {

body << "<h1>404 Not Found</h1>";

status = "404 Not Found";

}

std::string content = body.str();

std::ostringstream resp;

resp

<< "HTTP/1.1 " << status << "\r\n"

<< "Content-Length: " << content.size() << "\r\n"

<< "Content-Type: " << get\_type(file\_path) << "\r\n"

<< "Connection: close\r\n\r\n"

<< content;

std::string out = resp.str();

send(client, out.c\_str(), (int)out.size(), 0);

closesocket(client);

}

int main() {

WSADATA wsa;

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) return 1;

SOCKET listenSock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

sockaddr\_in addr{};

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(PORT);

addr.sin\_addr.s\_addr = INADDR\_ANY;

bind(listenSock, (sockaddr\*)&addr, sizeof(addr));

listen(listenSock, SOMAXCONN);

std::cout << "Minimal pool server listening on port " << PORT << "\n";

ThreadPool pool(THREAD\_COUNT);

while (true) {

SOCKET client = accept(listenSock, nullptr, nullptr);

if (client == INVALID\_SOCKET) continue;

pool.enqueue([client]() { handle\_client(client); });

}

closesocket(listenSock);

WSACleanup();

return 0;

}

Використаємо сторінки з курсу Веб-лизайн. Файли доволі великі, тому просто частина файлу головної сторінки:

<!DOCTYPE html>

<html lang="uk">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Фотограф Олексій Терещенко</title>

    <link rel="stylesheet" href="../styles/build/common.css" />

    <link rel="stylesheet" href="../styles/build/home.css" />

    <link

      href="https://fonts.googleapis.com/css2?family=Roboto:wght@100;300;400;500;600;700;900&family=Roboto+Condensed:wght@300;400;500;600;700&display=swap"

      rel="stylesheet"

    />

  </head>

  <body>

    <header id="header">

      <div class="logo-and-button">

        <div class="logo">

          <img src="../images/logo.png" alt="Логотип" />

        </div>

        <button class="burger-menu" aria-label="Відкрити меню">

          <span class="burger-icon"

            ><svg

              width="36"

              height="36"

              viewBox="0 0 36 36"

              fill="none"

              xmlns="http://www.w3.org/2000/svg"

            >

              <path

                d="M4.5 27V24H31.5V27H4.5ZM4.5 19.5V16.5H31.5V19.5H4.5ZM4.5 12V9H31.5V12H4.5Z"

                fill="#0D1821"

              />

            </svg>

          </span>

          <span class="close-icon" style="display: none"

            ><svg

              width="40"

              height="40"

              viewBox="0 0 40 40"

              fill="none"

              xmlns="http://www.w3.org/2000/svg"

            >

              <path

                d="M10.6667 31.6666L8.33337 29.3333L17.6667 19.9999L8.33337 10.6666L10.6667 8.33325L20 17.6666L29.3334 8.33325L31.6667 10.6666L22.3334 19.9999L31.6667 29.3333L29.3334 31.6666L20 22.3333L10.6667 31.6666Z"

                fill="#0D1821"

              />

            </svg>

          </span>

        </button>

      </div>

      <nav class="nav">

        <ul class="nav-list">

          <li>

            <a href="./index.html" class="nav-item active">

              <img

                id="home-icon"

                src="../images/material-symbols\_home.png"

                alt="Home Icon"

              />Головна

            </a>

          </li>

          <li><a href="./price.html" class="nav-item">Послуги</a></li>

          <li><a href="./contact.html" class="nav-item">Контакти</a></li>

        </ul>

        <div class="social-icons">

          <a href="#" class="social-icon"

            ><img src="../images/prime\_twitter.png" alt="Icon 1"

          /></a>

          <a href="#" class="social-icon"

            ><img src="../images/prime\_telegram.png" alt="Icon 2"

          /></a>

          <a href="#" class="social-icon"

            ><img src="../images/prime\_instagram.png" alt="Icon 3"

          /></a>

          <a href="#" class="social-icon"

            ><img src="../images/prime\_linkedin.png" alt="Icon 4"

          /></a>

        </div>

      </nav>

    </header>

    <section id="main-banner" class="banner">

      <div class="banner-content">

        <h1 class="photographer-name">Олексій Терещенко</h1>

        <p class="photographer-title">Фотограф</p>

      </div>

    </section>

…

Та архітектура сайту:

A screen shot of a computer

AI-generated content may be incorrect.

Рисунок 1 — Архітектура сайту

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 2 — Запущений сервер

Також створимо файл для тестування locustfile.py:

*from* locust *import* HttpUser, task, between

class StaticUser(HttpUser):

    wait\_time = between(1, 2)

    @task(1)

    def index(*self*):

*with* self.client.get("/", *catch\_response*=True) *as* resp:

*if* resp.elapsed.total\_seconds() > 10:

                resp.failure(f"Too slow: {resp.elapsed.total\_seconds():.3f}s")

    @task(1)

    def contact(*self*):

*with* self.client.get("/contact.html", *catch\_response*=True) *as* resp:

*if* resp.elapsed.total\_seconds() > 10:

                resp.failure(f"Too slow: {resp.elapsed.total\_seconds():.3f}s")

    @task(1)

    def price(*self*):

*with* self.client.get("/price.html", *catch\_response*=True) *as* resp:

*if* resp.elapsed.total\_seconds() > 10:

                resp.failure(f"Too slow: {resp.elapsed.total\_seconds():.3f}s")

    @task(1)

    def gtaVI(*self*):

*with* self.client.get("/gtaVI.html", *catch\_response*=True) *as* resp:

*if* resp.elapsed.total\_seconds() > 10:

                resp.failure(f"Too slow: {resp.elapsed.total\_seconds():.3f}s")

Запускаємо тестування через команду ***locust***.

Веб інтерфейс locust запущено, можна розпочати тестування:

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 3 — Інтерфейс locust.

Для початку запустимо тести з 1 юзером на секунду та подивимося на правильність роботи програми.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 4 — Результати роботи програми.

Бачимо, що gtaVI.html не відкривається, так як такої сторінки немає. Функціонал помилки 404 працює правильно.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5 — Текст помилки для не існуючого сайту.

Як бачимо, не існуючий сайт повертає помилку 404, що прописана в коді. Тепер приберемо цей сайт та протестимо три основні сторінки. Для цього змінимо locustfile.py та перезапустимо locust.

Візьмемо 3000 юзерів з 100 нових юзерів в секунду:

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 6 - 8 — 3000 одночасних користувачів.

RPS вийшов 300 запитів в секунду. Умова провалу – очікування 10 секунд для завантаження сайту. Як можна побачити, зі 116 тисяч запитів, 12 було провальними. Середній час відповіді був 5.5 секунд, 95%тиль був 8.1 секунд.

**ВИСНОВКИ**

У ході виконання лабораторної роботи було спроектовано та реалізовано мінімальний веб-сервер для хостингу статичного контенту на транспортному рівні моделі OSI із застосуванням сокетів Windows (Winsock). Сервер успішно обробляє HTTP 1.1 GET-запити для декількох веб-сторінок, повертаючи коректні заголовки Content-Type та Content-Length. За відсутності запитуваного ресурсу реалізовано механізм обробки помилок із поверненням статусу 404 Not Found, а при використанні некоректних методів – статусу 405 Method Not Allowed.

Для забезпечення одночасної обробки багатьох клієнтів було застосовано ThreadPool. Така архітектура дозволила розподілити вхідні з’єднання між потоками та запобігти блокуванням під час виконання окремих запитів. Навантажувальне тестування за допомогою Locust продемонструвало, що сервер витримує до приблизно 300 запитів за секунду, після чого починаються збої в обробці запитів із перевищенням максимального часу відповіді (10 с).

Отримані результати свідчать про доцільність подальшої оптимізації для підвищення стійкості та пропускної здатності сервера. Зокрема, можна реалізувати неблокуючий ввід/вивід із використанням I/O Completion Ports (IOCP) на платформі Windows або асинхронних API (WSAEventSelect), а також запровадити підтримку HTTP 1.1 Keep-Alive для повторного використання TCP-з’єднань. Додаткові заходи оптимізації включають кешування статичних ресурсів у пам’яті та балансування навантаження між кількома екземплярами сервера.