

**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-22  
Шаповалов Михайло

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**

Лістинг програм

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <thread>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 8080

#define ROOT "webroot"

using namespace std;

string readAll(const string& path)

{

ifstream in(path, ios::binary);

if (!in) return {};

ostringstream ss;

ss << in.rdbuf();

return ss.str();

}

void reply(SOCKET s, int code, const string& body)

{

string status = (code == 200 ? "200 OK" :

code == 404 ? "404 Not Found" :

code == 405 ? "405 Method Not Allowed" :

"500 Internal Server Error");

ostringstream hdr;

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

<< "Content-Type: text/html\r\n"

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

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

string out = hdr.str() + body;

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

}

void session(SOCKET client)

{

char buf[4096];

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

if (n <= 0)

{

closesocket(client); return;

}

buf[n] = '\0';

string req(buf);

if (req.rfind("GET ", 0) != 0)

{

reply(client, 405, "<h1>405 Method Not Allowed</h1>");

closesocket(client);

return;

}

size\_t pos = req.find(' ', 4);

string uri = req.substr(4, pos - 4);

if (uri == "/") uri = "/index.html";

string path = string(ROOT) + uri;

string body = readAll(path);

if (body.empty()) {

reply(client, 404, "<h1>404 Not Found</h1>");

}

else {

reply(client, 200, body);

}

closesocket(client);

}

int main()

{

WSADATA wsa;

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {

cerr << "WSAStartup failed\n"; return 1;

}

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

if (srv == INVALID\_SOCKET) { WSACleanup(); return 1; }

sockaddr\_in addr{};

addr.sin\_family = AF\_INET;

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

addr.sin\_port = htons(PORT);

if (bind(srv, (sockaddr\*)&addr, sizeof(addr)) == SOCKET\_ERROR || listen(srv, SOMAXCONN) == SOCKET\_ERROR)

{

cerr << "bind/listen failed\n";

closesocket(srv);

WSACleanup();

return 1;

}

cout << "Listening on http://localhost:" << PORT << "\n";

while (true)

{

SOCKET cli = accept(srv, nullptr, nullptr);

if (cli == INVALID\_SOCKET) continue;

thread(session, cli).detach();

}

closesocket(srv);

WSACleanup();

return 0;

}

index.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Page 1</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background: #f4f4f4;

            margin: 0;

            padding: 0;

            text-align: center;

        }

        header {

            background: #2196f3;

            color: white;

            padding: 20px;

        }

        main {

            padding: 40px;

        }

        a {

            color: #2196f3;

            text-decoration: none;

            font-weight: bold;

        }

        a:hover {

            text-decoration: underline;

        }

    </style>

</head>

<body>

    <header>

        <h1>Welcome to Page 1</h1>

    </header>

    <main>

        <p>This is the homepage of our simple static site.</p>

        <p>Click <a href="page2.html">here</a> to go to Page 2.</p>

    </main>

</body>

</html>

page2.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Page 2</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background: #fff3e0;

            margin: 0;

            padding: 0;

            text-align: center;

        }

        header {

            background: #ff9800;

            color: white;

            padding: 20px;

        }

        main {

            padding: 40px;

        }

        a {

            color: #ff9800;

            text-decoration: none;

            font-weight: bold;

        }

        a:hover {

            text-decoration: underline;

        }

    </style>

</head>

<body>

    <header>

        <h1>Welcome to Page 2</h1>

    </header>

    <main>

        <p>This is the second page. Great to have you here!</p>

        <p>Click <a href="index.html">here</a> to go back to Page 1.</p>

    </main>

</body>

</html>

Lab5.py:

from locust import HttpUser, task, between  
  
class WebsiteUser(HttpUser):  
 wait\_time = between(1, 3)  
  
 @task(2)  
 def index\_page(self):  
 self.client.get("/")  
  
 @task(1)  
 def second\_page(self):  
 self.client.get("/page2.html")

Для тестування буде використовуватись Locust, запускаємо сервер

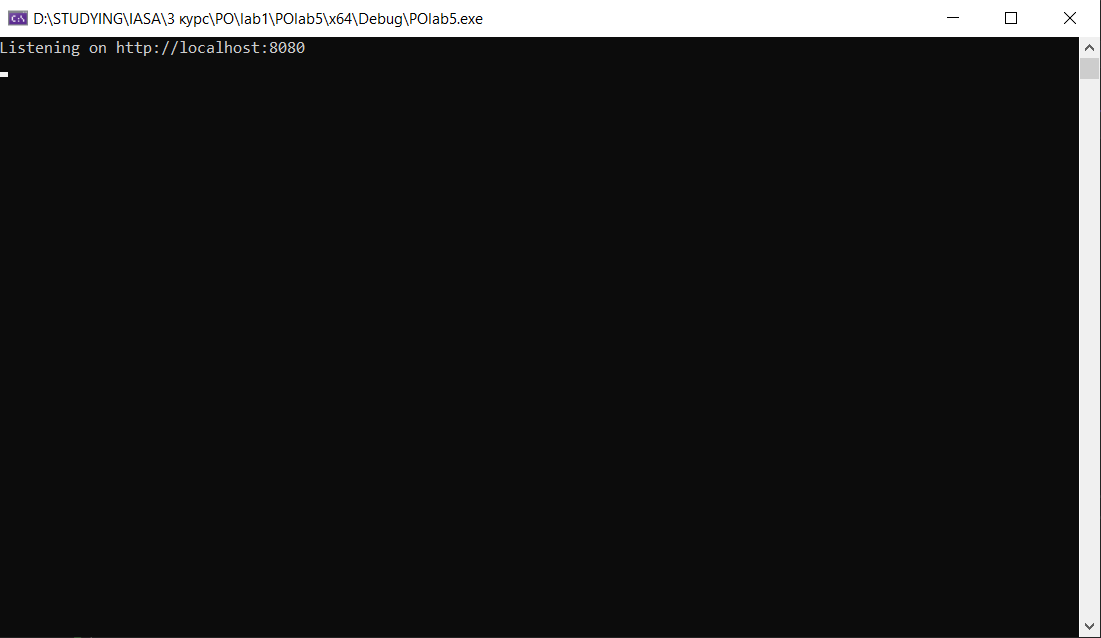
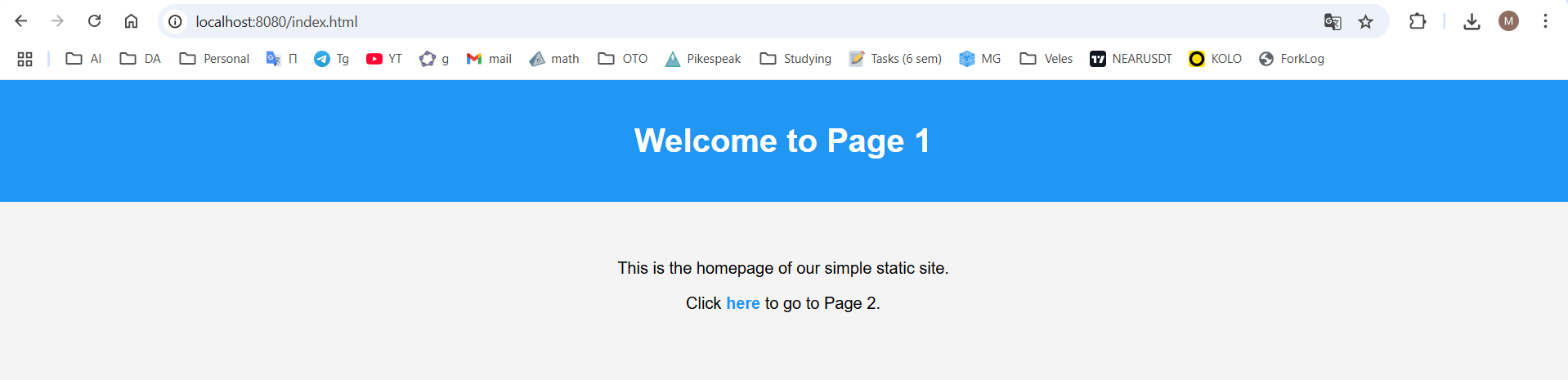
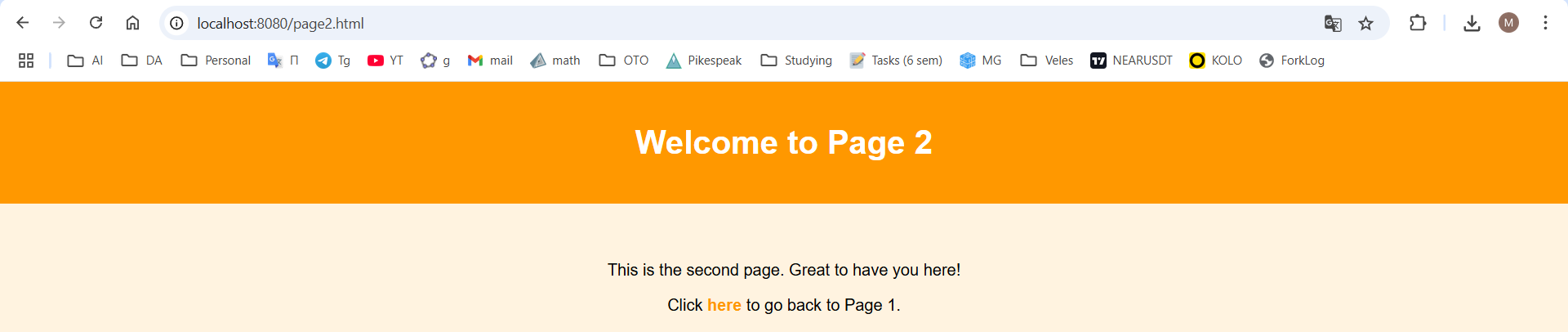


Рисунок 1 — Запуск сервера.





Так виглядають наші сторінки

Запускаємо Locust та переходимо за посиланням <http://localhost:8089/>

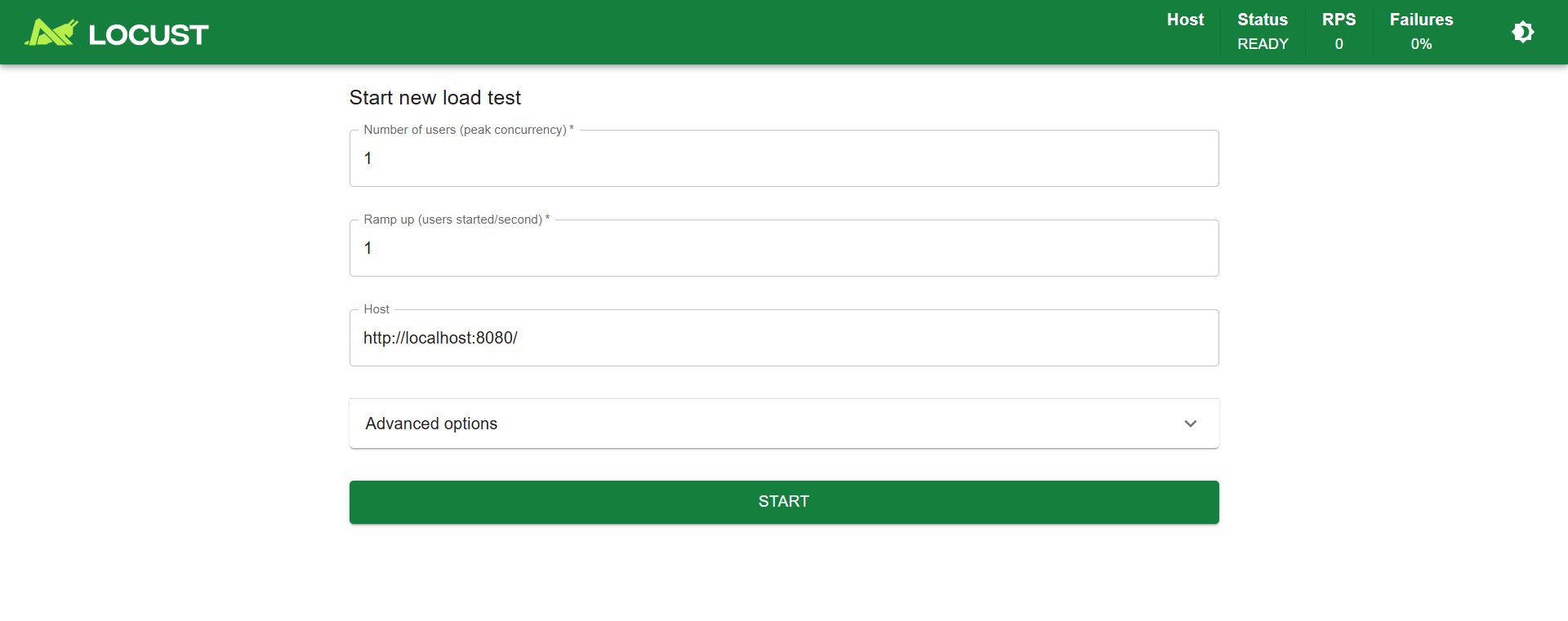


Рисунок 2 — Інтерфейс сайту для тестування.

Введемо адресу нашого сайту та перевіримо, чи працює взагалі тестування.

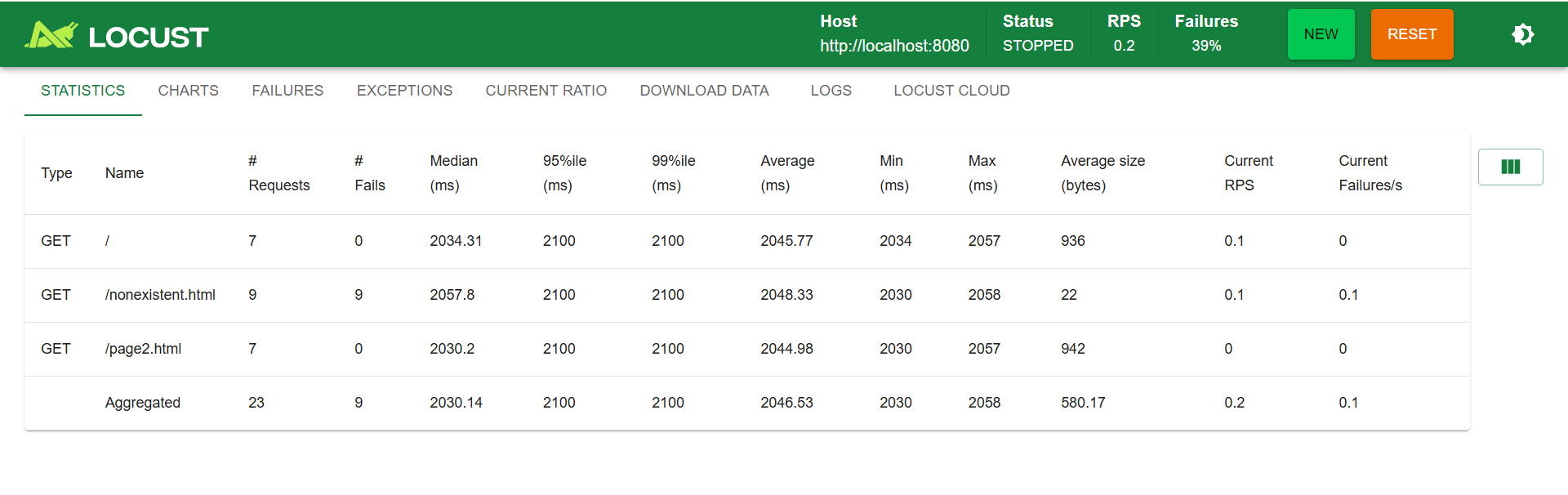
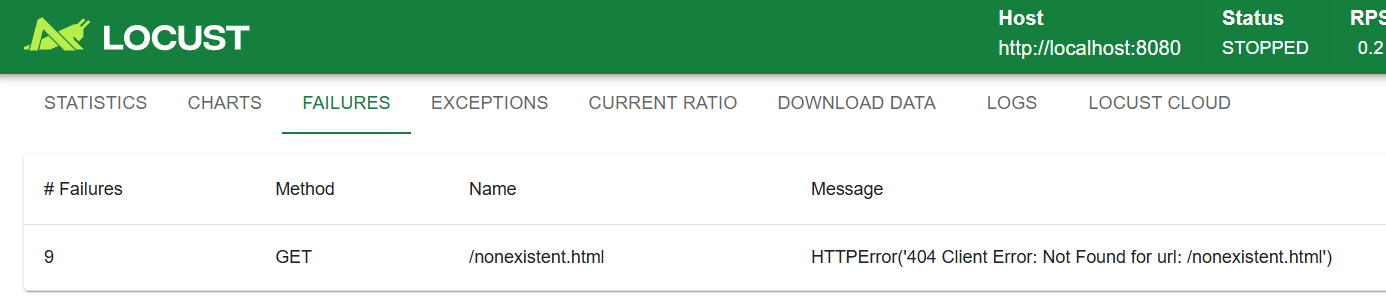


Рисунок 3 — Отриманий результат.

Наші основні дві сторінки працюють правильно, а неіснуюча повертає помилку завжди.



Причому як і вказано у завданні, неіснуючі сторінки повертають помилку 404

Тепер спробуємо запустити тестування на 500 юзерів.

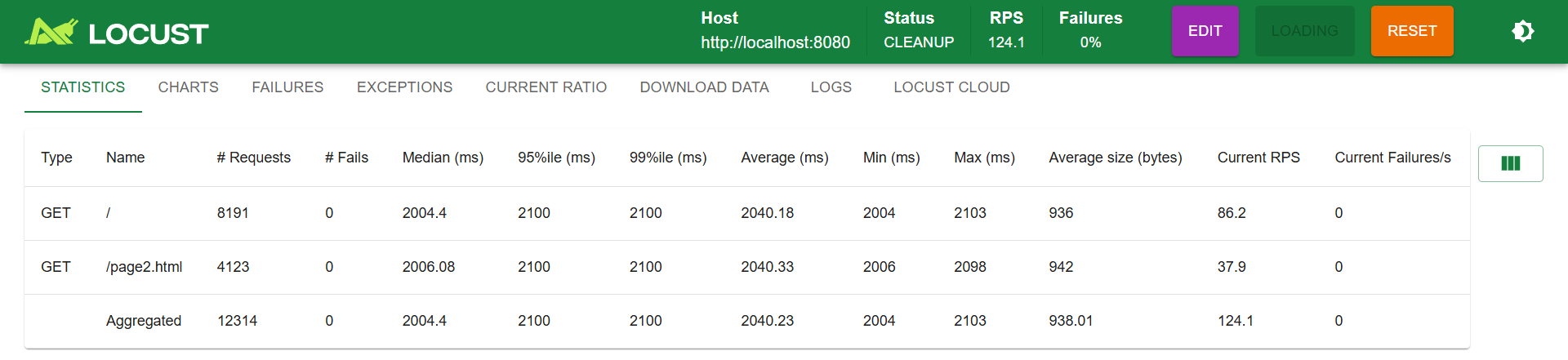


Рисунок 4 — Результати для 500 користувачів.

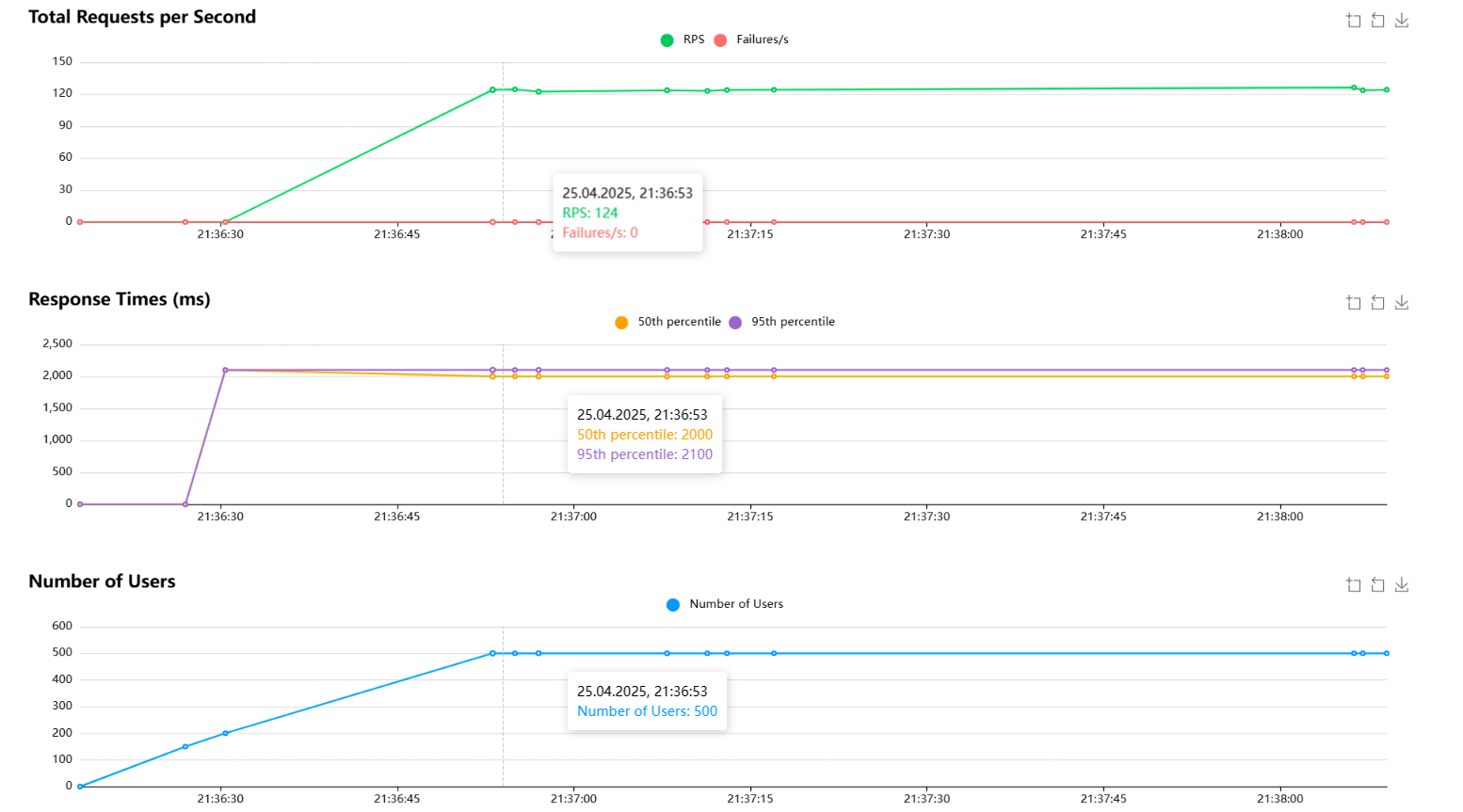


Рисунок 5 — Відповідні графіки.

Всі запити виявились успішними, тобто сервер витримав навантаження, збільшимо кількість до 50000 юзерів.



Рисунок 6 — Результати для 50000 користувачів.

При тесті на 50000 юзерів отримали такі результати, видно, що помилки не з’явились, але час очікування зріз і став більше за хвилину, бо сервер має RPS близько 800, тобто більшість просто стоїть в черзі та чекає (дуже довго).

Модифікуємо трохи код, щоб обмежити максимальну кількість користувачів в черзі, а всі хто не влізає будуть отримувати помилки

Код з обмеженням максимальної кількості:

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <thread>

#include <atomic>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 8080

#define ROOT "webroot"

#define MAX\_CLIENTS 1000

using namespace std;

atomic<int> active\_clients(0);

string readAll(const string& path) {

ifstream in(path, ios::binary);

if (!in) return {};

ostringstream ss;

ss << in.rdbuf();

return ss.str();

}

void reply(SOCKET s, int code, const string& body) {

string status = (code == 200 ? "200 OK" :

code == 404 ? "404 Not Found" :

code == 405 ? "405 Method Not Allowed" :

"503 Service Unavailable");

ostringstream hdr;

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

<< "Content-Type: text/html\r\n"

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

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

string out = hdr.str() + body;

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

}

void session(SOCKET client) {

active\_clients++;

char buf[4096];

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

if (n <= 0) {

active\_clients--;

closesocket(client);

return;

}

buf[n] = '\0';

string req(buf);

if (req.rfind("GET ", 0) != 0) {

reply(client, 405, "<h1>405 Method Not Allowed</h1>");

active\_clients--;

closesocket(client);

return;

}

size\_t pos = req.find(' ', 4);

string uri = req.substr(4, pos - 4);

if (uri == "/") uri = "/index.html";

string path = string(ROOT) + uri;

string body = readAll(path);

if (body.empty()) {

reply(client, 404, "<h1>404 Not Found</h1>");

}

else {

reply(client, 200, body);

}

active\_clients--;

closesocket(client);

}

int main() {

WSADATA wsa;

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {

cerr << "WSAStartup failed\n";

return 1;

}

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

if (srv == INVALID\_SOCKET) {

WSACleanup();

return 1;

}

sockaddr\_in addr{};

addr.sin\_family = AF\_INET;

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

addr.sin\_port = htons(PORT);

if (bind(srv, (sockaddr\*)&addr, sizeof(addr)) == SOCKET\_ERROR || listen(srv, SOMAXCONN) == SOCKET\_ERROR) {

cerr << "bind/listen failed\n";

closesocket(srv);

WSACleanup();

return 1;

}

cout << "Listening on http://localhost:" << PORT << "\n";

while (true) {

SOCKET cli = accept(srv, nullptr, nullptr);

if (cli == INVALID\_SOCKET) continue;

if (active\_clients >= MAX\_CLIENTS) {

string busy\_response = "<h1>503 Service Unavailable</h1>";

reply(cli, 503, busy\_response);

closesocket(cli);

continue;

}

thread(session, cli).detach();

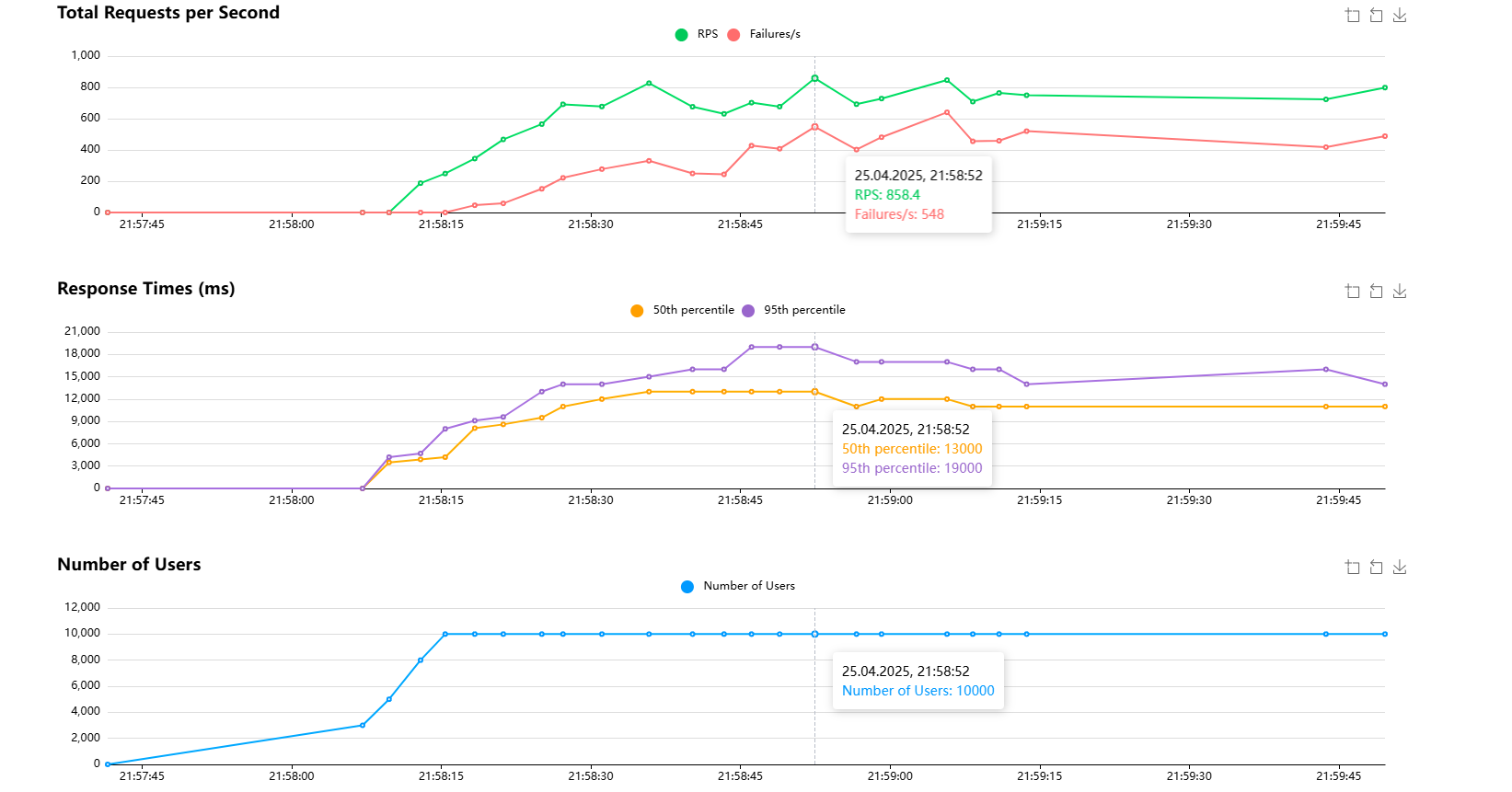
}

closesocket(srv);

WSACleanup();

return 0;

}

Рисунок 7 — Результати для 10000 користувачів.

Тепер спробувавши запустити тестування з навантаженням до 10000 юзерів, які надходять 1500/сек отримали такі результати, час відповіді зменшився приблизно в п’ять разів, але майже половина користувачів почала отримувати повідомлення про недоступність сайту.

Провівши ще декілька тестів, встановлено, що точка відмови, після якої система перестає стабільно працювати, це надходження більше ніж 800 користувачів у секунду

**ВИСНОВКИ**

У ході виконання лабораторної роботи було спроєктовано та реалізовано простий веб-сервер для хостингу статичного контенту на транспортному рівні мережевого стеку OSI за допомогою сокетів (Winsock API). Сервер обробляє HTTP GET-запити відповідно до протоколу HTTP/1.1, коректно повертає відповіді з кодами 200 (OK), 404 (Not Found), 405 (Method Not Allowed) та 503 (Service Unavailable) при перевищенні максимальної кількості одночасних клієнтів.

Для навантажувального тестування був використаний інструмент Locust, що дозволило проаналізувати продуктивність сервера під великим навантаженням. Результати тестування показали, що без обмежень сервер здатний обробляти до 50 000 запитів, однак час відповіді при цьому критично збільшується до понад 60 секунд. Введення обмеження на кількість активних клієнтів до 1000 дозволило визначити точку відмови сервера на рівні близько 800 паралельних користувачів при стабільній роботі.

Для підвищення ефективності роботи сервера у майбутньому доцільно реалізувати неблокуючу модель введення-виведення, використовувати пул потоків замість створення нового потоку для кожного з'єднання, встановити таймаути на прийом та передачу даних, а також оптимізувати обробку відповідей шляхом кешування або стиснення контенту.