**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені Ігоря Сікорського»**

**Лабораторна робота № 5**

**з курсу «Паралельні обчислення»**

**на тему «Розробка HTTP-сервера для хостингу статичного веб-сайту»**

**Виконав студент ІПСА групи ДА-22**

**Котляр Єгор Ярославович**

Київ – 2025

**Мета роботи**

Вивчити принцип роботи http веб-серверів. Створити простий веб-сервер для багатопотокової обробки GET-запитів. Засвоїти інструментарій для навантажувального тестування.

**Завдання**

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.

Посилання на [репозиторій](https://github.com/Kott1/PC).

**Хід виконання роботи**

Для даної лабораторної напишемо сервер у файлі mian.cpp, а для тестування навантаження використаємо locust, для якого завантажимо python:

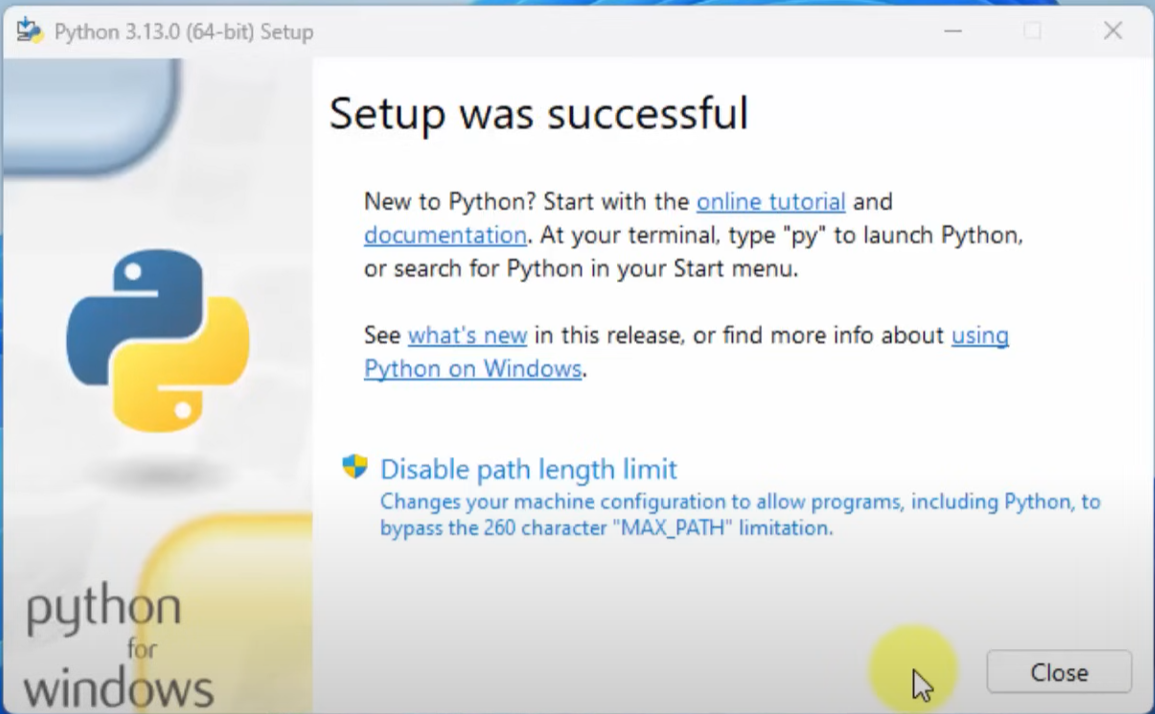


Рисунок 1 – Лаунчер завантаження python.

Перевіримо в консолі:

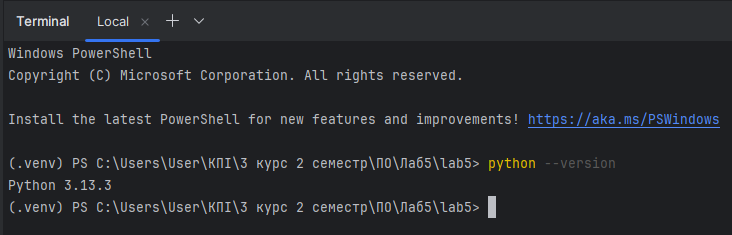


Рисунок 2 – Перевірка завантаження python.

Установимо locust за допомогою pip install locust і перевіримо його наявність:

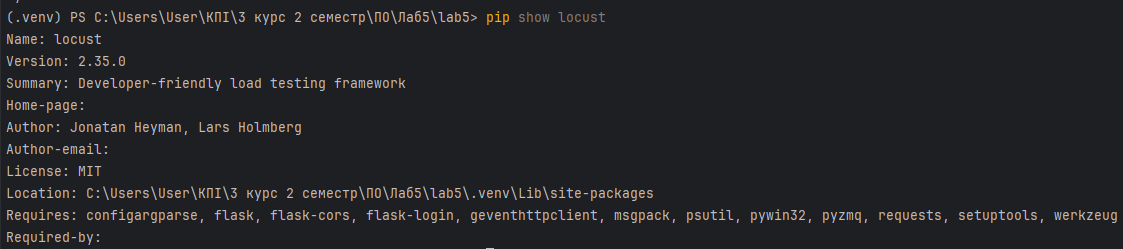


Рисунок 3 – Завантаження locust.

**Код програми:**

#include <winsock2.h>  
#include <ws2tcpip.h>  
#include <windows.h>  
#include <fstream>  
#include <iostream>  
#include <sstream>  
#include <string>  
#include <thread>  
  
using namespace std;  
  
#define PORT 8080  
const string PAGE\_DIR = "pages";  
  
void sendResponse(SOCKET clientSock, const string& status, const string& contentType, const string& body) {  
 ostringstream oss;  
 oss << "HTTP/1.1 " << status << "\r\n"  
 << "Content-Length: " << body.size() << "\r\n"  
 << "Content-Type: " << contentType << "\r\n"  
 << "Connection: close\r\n\r\n"  
 << body;  
 auto resp = oss.str();  
 send(clientSock, resp.c\_str(), (int)resp.size(), 0);  
}  
  
void handleClient(SOCKET clientSock) {  
 char buffer[4096];  
 int bytes = recv(clientSock, buffer, sizeof(buffer) - 1, 0);  
 if (bytes <= 0) {  
 closesocket(clientSock);  
 return;  
 }  
 buffer[bytes] = '\0';  
  
 istringstream req(buffer);  
 string method, path, version;  
 req >> method >> path >> version;  
  
 if (method != "GET") {  
 sendResponse(clientSock, "405 Method Not Allowed",  
 "text/plain", "Method Not Allowed");  
 closesocket(clientSock);  
 return;  
 }  
  
 if (path == "/") path = "/home.html";  
 string filePath = PAGE\_DIR + path;  
  
 ifstream ifs(filePath, ios::*binary*);  
 if (!ifs) {  
 string body =  
 "<!DOCTYPE html>"  
 "<html><head><meta charset=\"utf-8\">"  
 "<title>404 Not Found</title>"  
 "<style>"  
 "body { font-family: Arial, sans-serif; text-align: center; padding-top: 50px; }"  
 "h1 { font-size: 48px; color: #cc0000; }"  
 "p { font-size: 24px; color: #555; }"  
 "</style>"  
 "</head><body>"  
 "<h1>404 Not Found</h1>"  
 "</body></html>";  
 sendResponse(clientSock, "404 Not Found", "text/html", body);  
 } else {  
 string body((istreambuf\_iterator<char>(ifs)), {});  
 string ct = (path.find(".html") != string::*npos*)  
 ? "text/html"  
 : "application/octet-stream";  
 sendResponse(clientSock, "200 OK", ct, body);  
 }  
  
 closesocket(clientSock);  
}  
  
int main() {  
 WSADATA wsaData;  
 if (WSAStartup(MAKEWORD(2,2), &wsaData) != 0) {  
 cerr << "WSAStartup failed\n";  
 return 1;  
 }  
  
 SOCKET listenSock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  
 if (listenSock == INVALID\_SOCKET) {  
 cerr << "socket() failed\n";  
 WSACleanup();  
 return 1;  
 }  
  
 sockaddr\_in serverAddr{};  
 serverAddr.sin\_family = AF\_INET;  
 serverAddr.sin\_port = htons(PORT);  
 serverAddr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;  
  
 int opt = 1;  
 setsockopt(listenSock, SOL\_SOCKET, SO\_REUSEADDR, (const char\*)&opt, sizeof(opt));  
  
 if (bind(listenSock, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {  
 cerr << "bind() failed\n";  
 closesocket(listenSock);  
 WSACleanup();  
 return 1;  
 }  
  
 if (listen(listenSock, SOMAXCONN) == SOCKET\_ERROR) {  
 cerr << "listen() failed\n";  
 closesocket(listenSock);  
 WSACleanup();  
 return 1;  
 }  
  
 cout << "Listening on port " << PORT << "...\n";  
 while (true) {  
 sockaddr\_in clientAddr;  
 int addrLen = sizeof(clientAddr);  
  
 SOCKET clientSock = accept(listenSock, (sockaddr\*)&clientAddr, &addrLen);  
 if (clientSock == INVALID\_SOCKET) {  
 cerr << "accept() failed\n";  
 continue;  
 }  
  
 thread(handleClient, clientSock).detach();  
 }  
  
 closesocket(listenSock);  
 WSACleanup();  
 return 0;  
}

**Код сторінки home:**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Page 1</title>  
</head>  
<body>  
<h1>Голоовна сторінка</h1>  
<p>натисни <a href="page.html">тут</a> щоб перейти на другу сторінку.</p>  
</body>  
</html>

**Код сторінки page:**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Page 2</title>  
</head>  
<body>  
<h1>Друга сторінка</h1>  
<p>Натисни <a href="home.html">тут</a> Щоб повернутися на головну.</p>  
</body>  
</html>

**Код файлу locust:**

from locust import HttpUser, task  
  
class WindowsStaticUser(HttpUser):  
  
 @task  
 def home(self):  
 self.client.get("/home.html")  
  
 @task  
 def page(self):  
 self.client.get("/page.html")  
  
 @task  
 def page404(self):  
 self.client.get("/page404.html")

Далі, запускаємо програму й окремо в консолі запускаємо locust командою:

locust -f locust\_lab5.py --host=http://localhost:8080

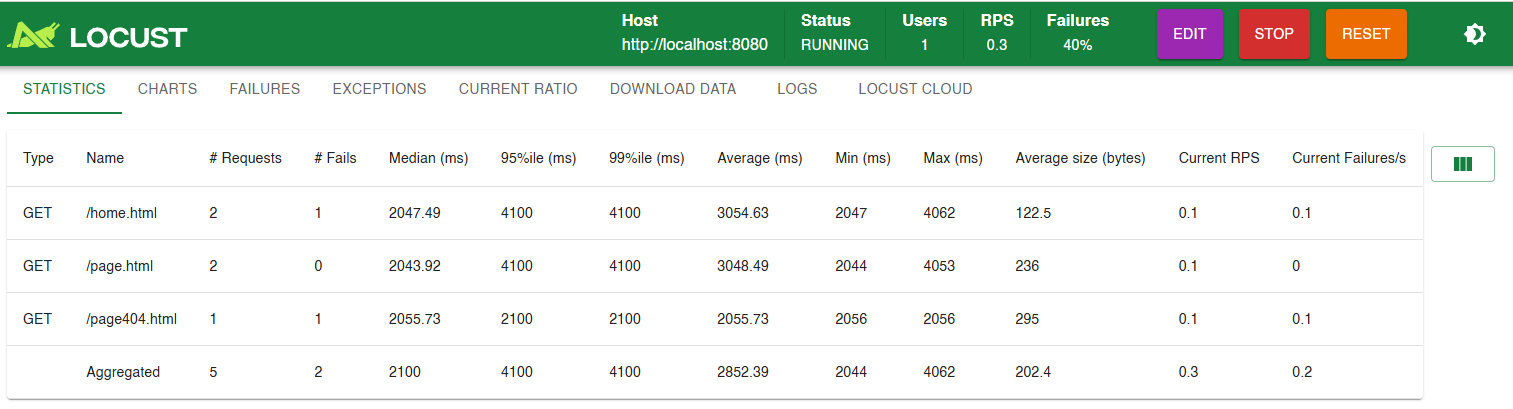


Рисунок 4 – Locust у браузері.

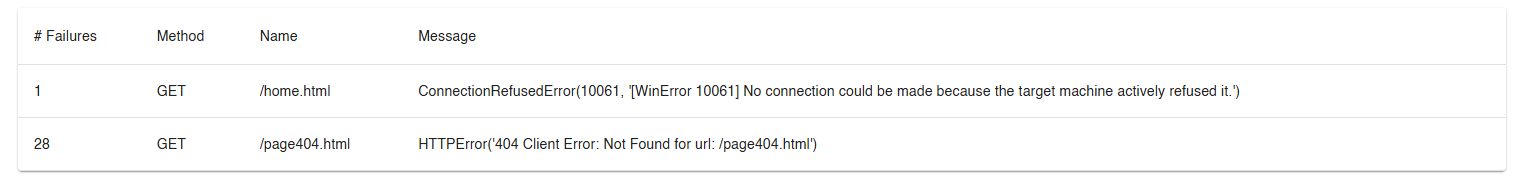
****

Рисунок 5 – Помилка для неіснуючої сторінки.

Як бачимо, сторінки home, page та page404, що ми вказали, з’явилися. Тепер можемо робити тести з навантаженнями серверу:

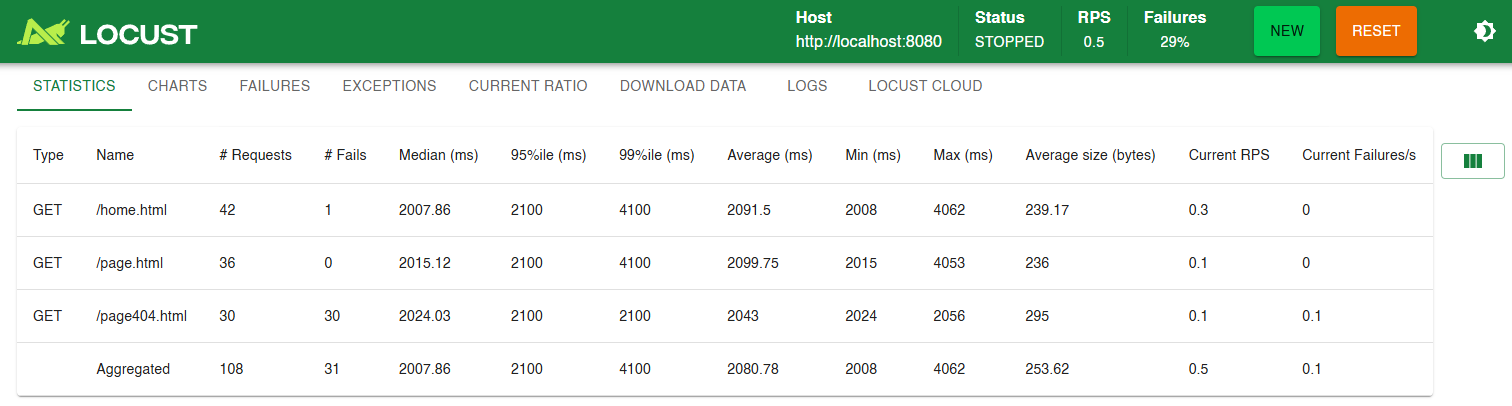
****

Рисунок 6 – Дані при 1 користувачі.

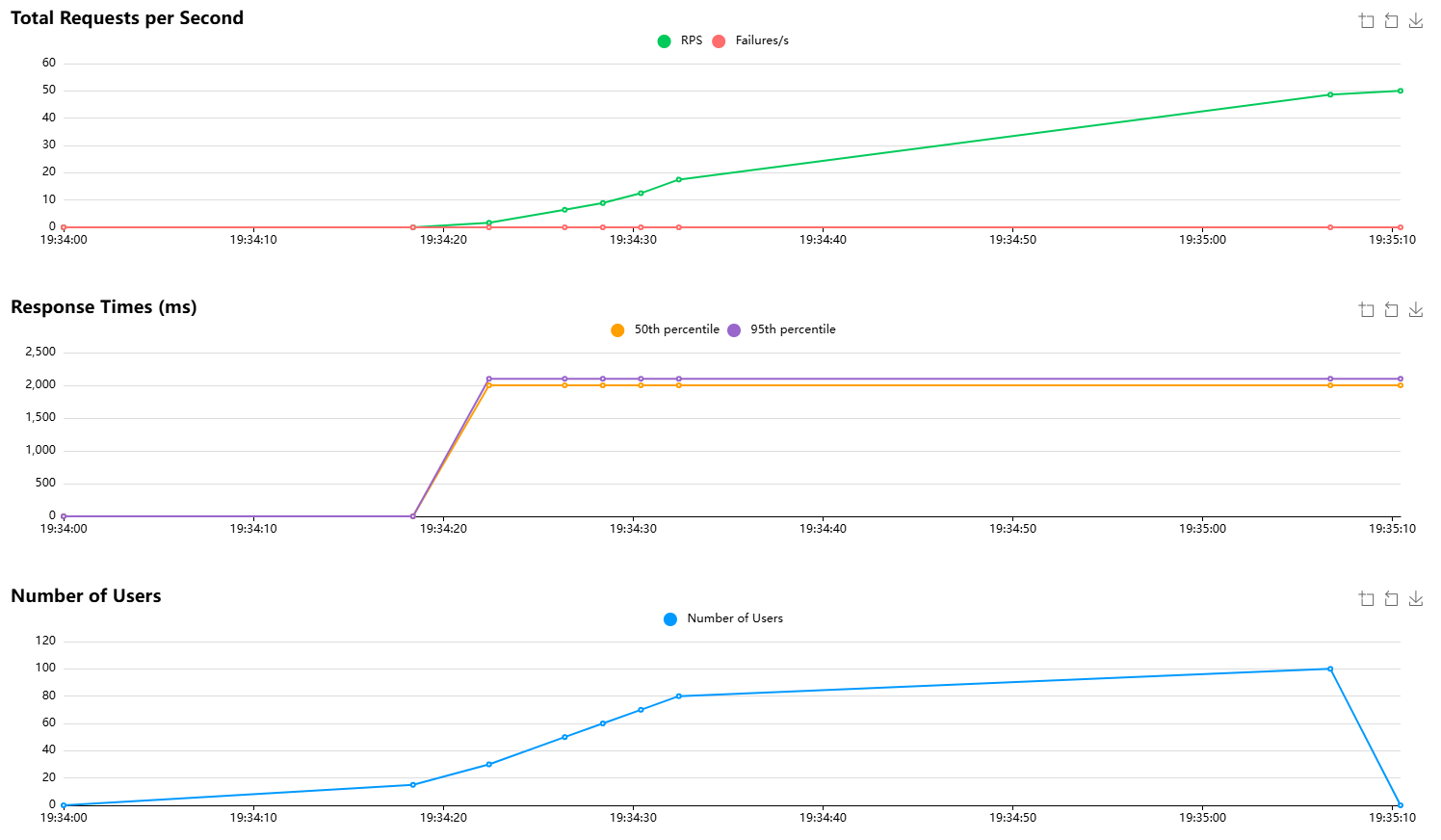
****

Рисунок 6 – Графіки при 1 користувачі.

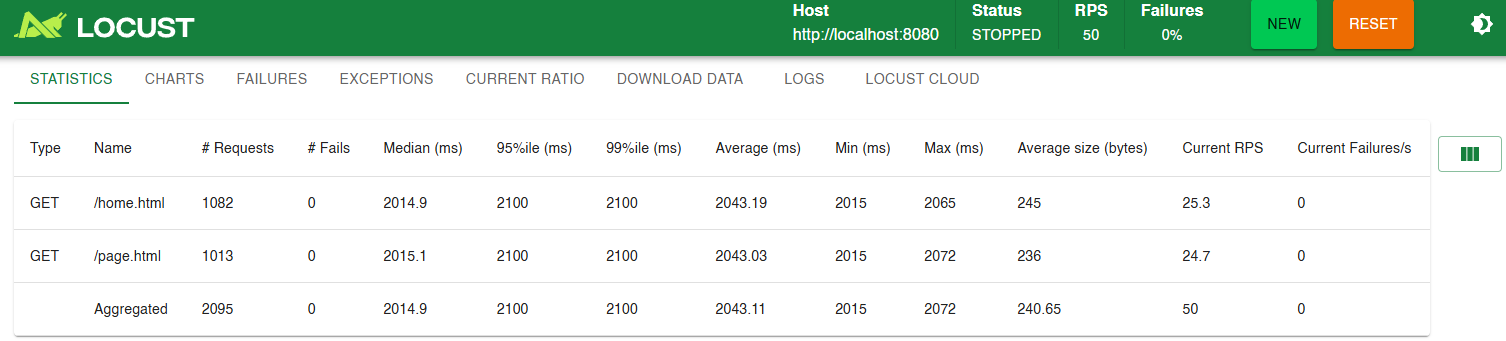
****

Рисунок 7 – Дані при 100 користувачах.

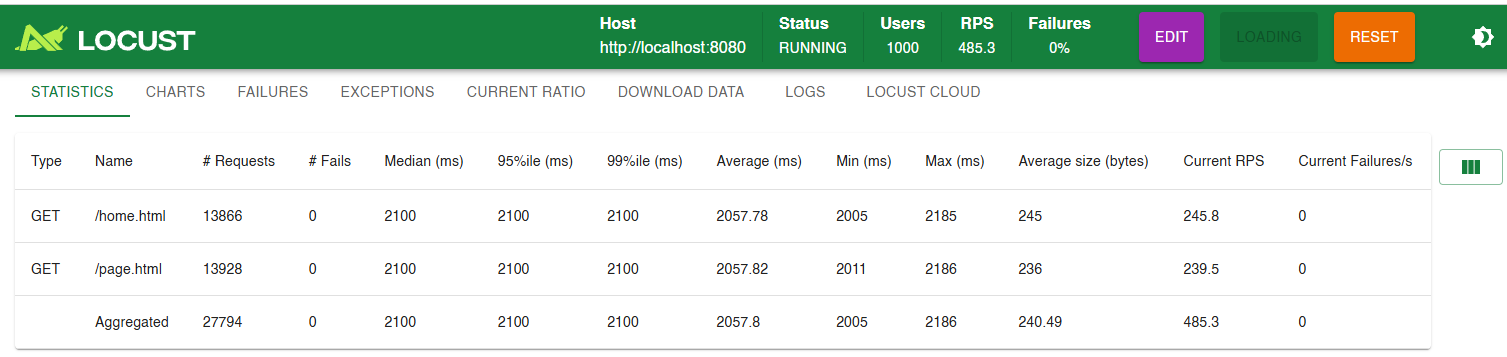
****

Рисунок 8 – дані при 1000 користувачах.

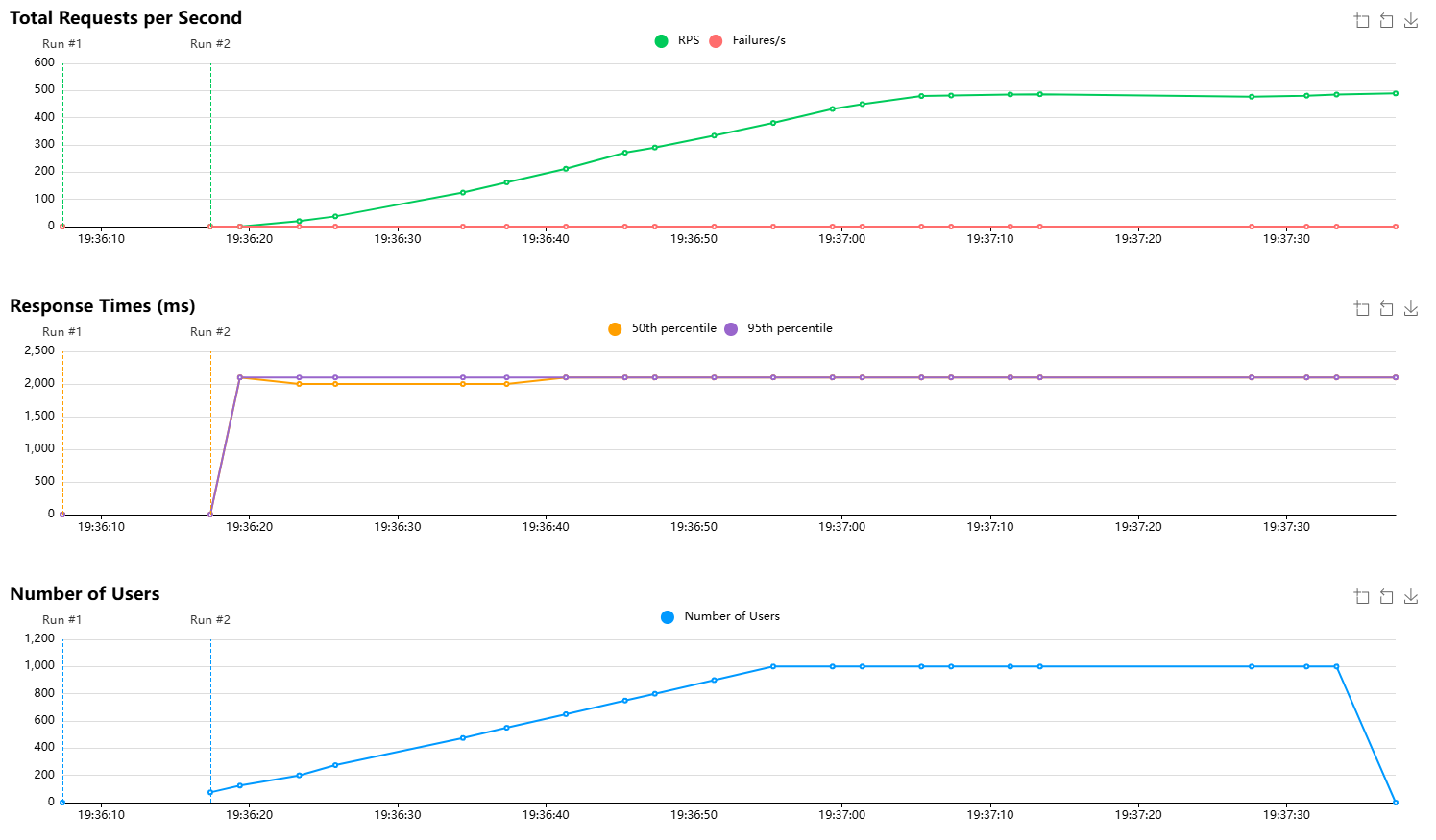
****

Рисунок 9 – Графіки при 1000 користувачах.

**Висновок**

У ході виконання лабораторної роботи було досліджено основні принципи роботи HTTP‑веб‑серверів на транспортному рівні OSI та реалізовано власний простий багатопотоковий сервер на основі сокетів Winsock2 у C++. Сервер обробляє GET‑запити відповідно до специфікації HTTP/1.1, спрямовуючи кореневий запит на home.html, правильно визначає MIME‑тип відповіді і повертає вбудовану стилізовану 404‑сторінку у разі відсутності запитаного ресурсу.

Було створено дві HTML‑сторінки (home.html та page.html), а також протестовано обробку неіснуючої сторінки через запит до /page404.html. Для навантажувального тестування використовувався Locust: із зростанням кількості одночасних користувачів (від 1 до 1000) спостерігалося поступове збільшення середнього часу відповіді. Точка критичного зниження продуктивності серверу виявилася поблизу 1000 паралельних клієнтів, коли зростання кількості потоків без додаткових оптимізацій призвело до перевитрат ресурсів ОС і деградації швидкодії.