

**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 №4  
з дисципліни «Паралельні обчислення»**

Виконав:  
студент III курсу, групи ДА-22  
Байдюк Антон Олексійович

Прийняв:

асистент Яременко В. С.

Київ – 2025

**ПОСТАНОВКА ЗАДАЧІ**

1. Розробити клієнт-серверний застосунок для вирішення завдання з

лабораторної роботи номер 1, передавши масив даних з клієнта на сервер,

а потім – отримавши результат назад на сторону клієнта. Для виконання

основного завдання дозволено використовувати лише платформні

**(WinSock, POSIX**) та вбудовані засоби роботи з сокетами.

2. Самостійно розробити протокол прикладного рівня (application protocol)

для взаємодії клієнта з сервером. Для цього врахувати декілька кроків в

процесі взаємодії:

A) надсилання даних та конфігурації обчислень (наприклад, вказати

кількість потоків для виконання обчислень)

B) надсилання команди (та отримання відповіді на команду) для початку

обчислень

1. надсилання команди для запиту статусу і результату обчислень.

3. Додати до розробленого серверу підтримку підключення декількох

клієнтів одночасно. Додатковий бал можна отримати у випадку наявності

двох клієнтів, один з яких буде розроблений на мові, відмінній від мови

серверу та першого клієнту (другий клієнт дозволено створювати з

використанням скриптових мов).

4. **З**астосунок повинен коректно оброблювати виняткові ситуації як на стороні клієнту, так і на стороні серверу і адекватно реагувати на них. Без завершення своєї роботи. Обов’язковим є коректна обробка порядку кодування байтів у повідомленні.

1. В протоколі роботи необхідно навести опис розробленого застосунку. До цього опису повинні входити обґрунтування вибору протоколу передачі даних, а також архітектурний опис клієнта.
2. Занести до протоколу роботи опис розробленого протоколу прикладного рівня у вигляді таблиці, що включає: перелік всіх команд, аргументи команд та їх опис, список можливих відповідей на команду.
3. Занести до протоколу роботи UML діаграму викликів взаємодії серверу та клієнту, починаючи від запуску клієнту, до завершення роботи.

8. Надати висновок, що повинен містити аналіз та опис проблем з котрими

зіштовхнувся студент, або з якими може зіштовхнутися розробник при

організації міжпроцесової взаємодії.

1. Теоретичні відомості до роботи відсутні, частина матеріалу покривається лекційним в курсах Паралельних обчислень та Комп’ютерних мереж. Нижче представлено умовний приклад UML діаграми, необхідної для даної роботи.

**Обґрунтування вибору протоколу передачі даних, а також архітектурний опис клієнта**

У даній клієнт-серверній програмі використовується **протокол TCP** , який є частиною стеку TCP/IP. Вибір TCP зумовлений наступними критеріями:

**Надійність передачі даних**: TCP гарантує доставку всіх байтів у правильному порядку, що критично важливо при передачі масиву чисел та математичних обчислень

**Контроль цілісності**: TCP має вбудовані механізми перевірки контрольних сум, що забезпечує точність отриманих даних

**Підтримка з'єднання**: TCP створює стабільне з'єднання між клієнтом і сервером, що дає змогу легко реалізувати діалоговий обмін командами, параметрами, масивами та результатами обчислень

**Простота реалізації протоколу взаємодії**: Завдяки потоковій природі TCP, структура протоколу обміну між клієнтом і сервером може бути простою — дані передаються послідовно блоками, що дозволяє уникнути складного буферного контролю

**Підтримка великих обсягів даних**: У програмі передаються вектори довільної довжини, які можуть бути великими — TCP ідеально підходить для таких випадків, бо автоматично розбиває великі повідомлення на пакети і пересилає їх

Головним недоліком TСP є швидкість роботи із значним об’ємом даних, проте у даній роботі це не буде значною проблемою

Клієнтська частина реалізована у вигляді консольної програми, яка дозволяє користувачеві:

**Обрати режим введення: ручний, чи автоматичний**

Обрати розмір вектора даних

Обрати кількість потоків для обробки

Отримати результат роботи : суму квадратів елементів вектора та його норму

**Опис розробленого протоколу прикладного рівня у вигляді таблиці, що включає: перелік всіх команд, аргументи команд та їх опис, список можливих відповідей на команду:**

| **Команда / Дані** | **Аргументи** | **Призначення** | **Відповідь сервера / Дія** |
| --- | --- | --- | --- |
| int mode | 0 – автоматичне заповнення  1 – ручне введення | Вказує режим формування масиву на клієнті | Mode received (рядок підтвердження) |
| int size | Ціле число (n > 0) | Розмір масиву, що буде передано | Немає прямої відповіді. Сервер формує/чекає на масив |
| int[] array | Масив цілих чисел довжини size | Вводиться клієнтом (якщо mode == 1) | Немає прямої відповіді. Сервер чекає на наступну команду |
| int threadCount | 0 – без багатопоточності  > 0 – кількість потоків | Визначає режим обробки (послідовно або паралельно) | Немає прямої відповіді |
| struct Result | float sum, float norm, long long duration\_ns | Сервер повертає клієнту результат обчислень (сума квадратів, норма, час) | Клієнт виводить результат |

**Занести до протоколу роботи UML діаграму викликів взаємодії серверу та клієнту, починаючи від запуску клієнту, до завершення роботи.**

Лістинг коду для plantUML:

@startuml

autonumber

actor User

participant Client

participant Server

== Запуск клієнта ==

User -> Client : Запускає програму

== Вибір режиму ==

Client -> User : Виводить запит: оберіть режим (0 - ручний, 1 - автоматичний)

User -> Client : Вводить режим

Client -> Server : Надсилає mode

Server -> Client : Підтвердження отримання режиму

== Введення або генерація масиву ==

alt Режим = Ручний (0)

Client -> User : Введіть розмір масиву

User -> Client : Вводить size

Client -> User : Введіть елементи масиву

User -> Client : Вводить array[]

Client -> Server : Надсилає size і array[]

else Режим = Автоматичний (1)

Client -> User : Введіть розмір масиву

User -> Client : Вводить size

Client -> Client : Генерує масив

Client -> Server : Надсилає size і array[]

end

== Вибір потоків ==

Client -> User : Введіть кількість потоків (0 = без багатопоточності)

User -> Client : Вводить threadCount

Client -> Server : Надсилає threadCount

== Обробка ==

Server -> Server : Обчислення норми масиву

Server -> Server : Вимір часу обробки

== Результат ==

Server -> Client : Повертає норму, суму, час

Client -> User : Виводить результати

== Завершення ==

User -> Client : Завершення програми

Client -> Server : Закриває з’єднання

@enduml

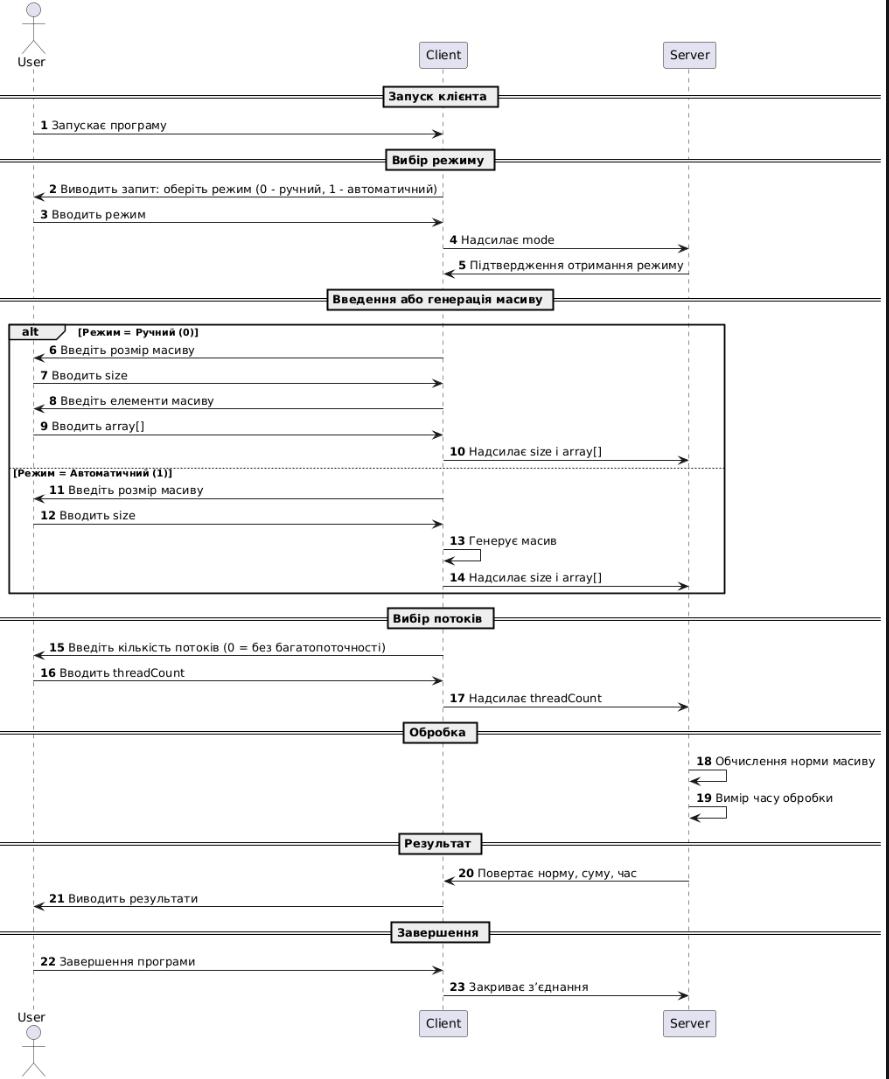


Рисунок 1 - Згенерована UML діаграма на plantUML

**Результат роботи програми:**

\*Лістинг коду буде у додатку після висновку

У процесі буде запущено сервер у visual studio 2022 та 3 клієнти у code::blocks (там легко відкрити кілька вкладок):

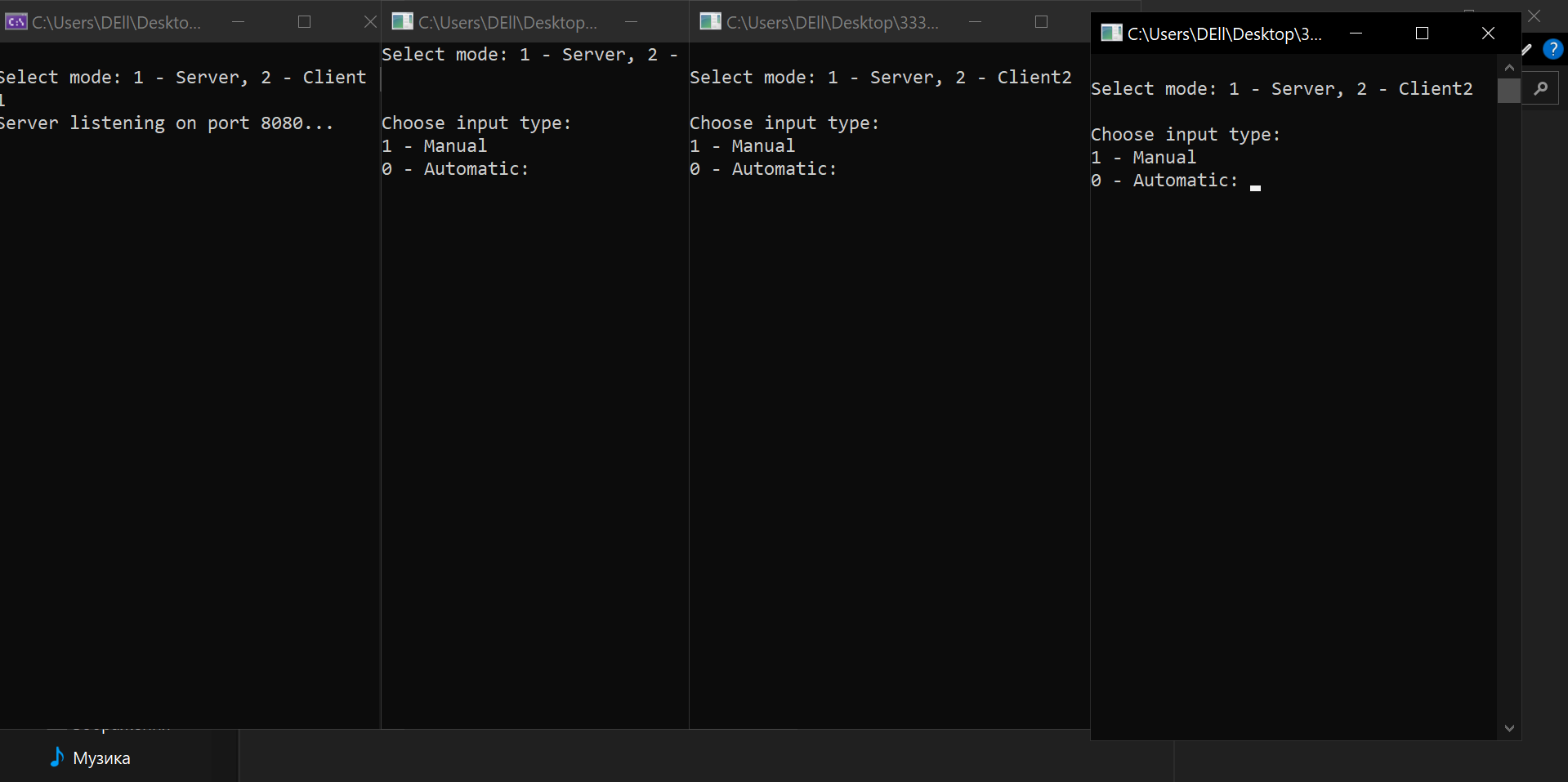


Рисунок 2 - Запуск сервера та клієнтів

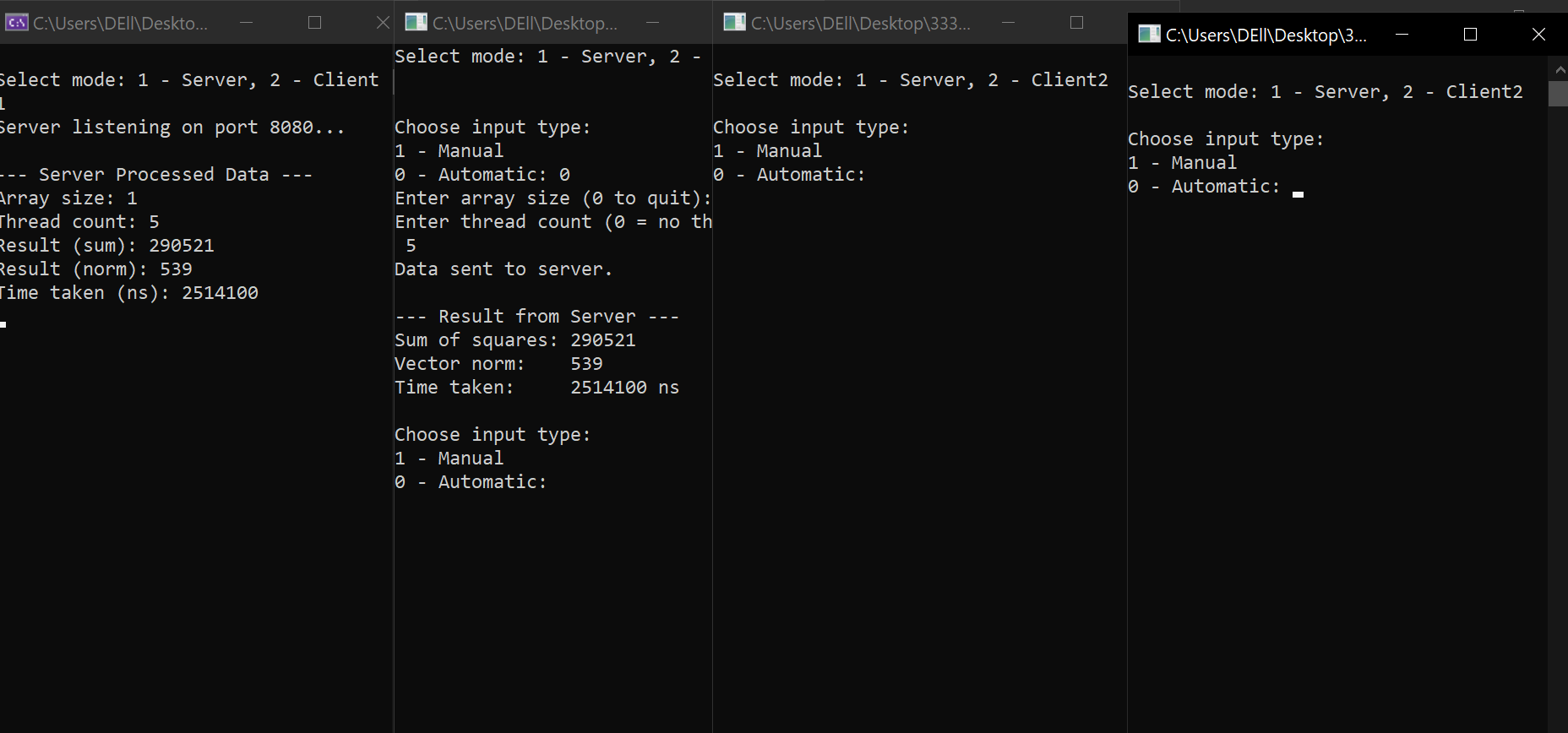


Рисунок 3 - Вибір автоматичного способу генерації масиву на 1 елемент та обрахунок 5-ма потоками на одному із клієнтів

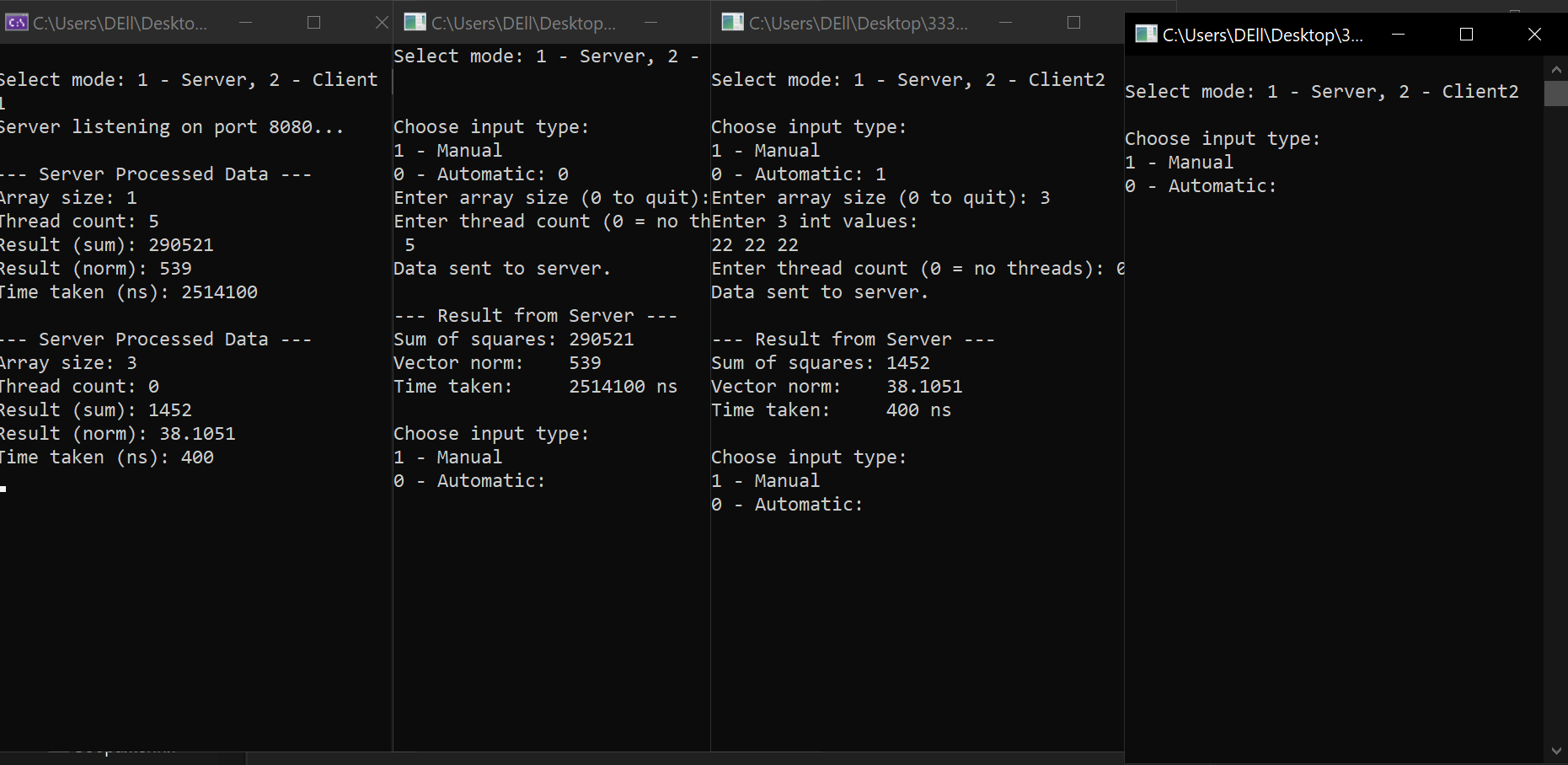


Рисунок 4 - Вибір ручного способу генерації масиву на 3 елементи та обрахунок 1-им потоком на другому із клієнтів

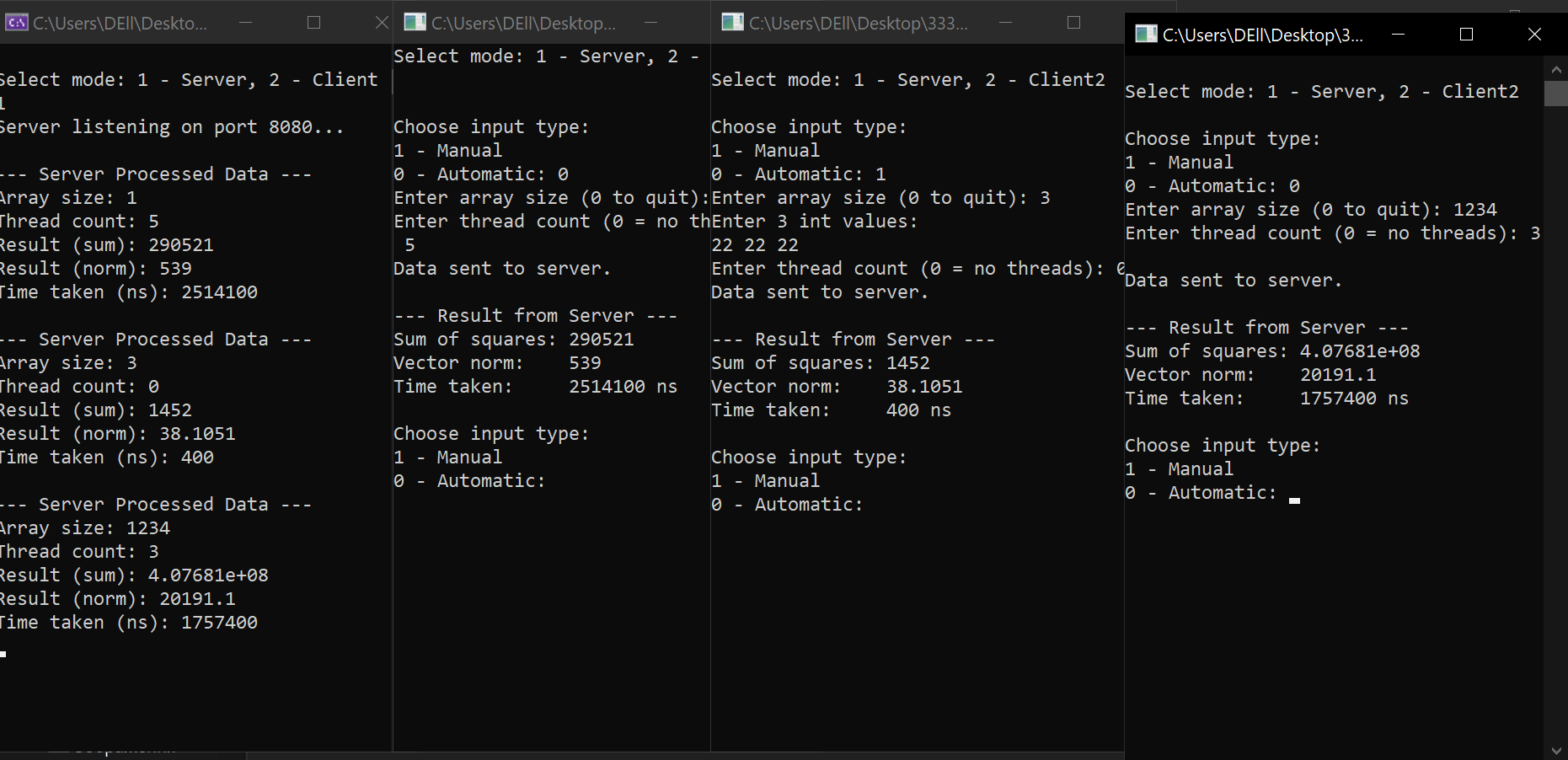


Рисунок 5 - Запуск третього клієнту

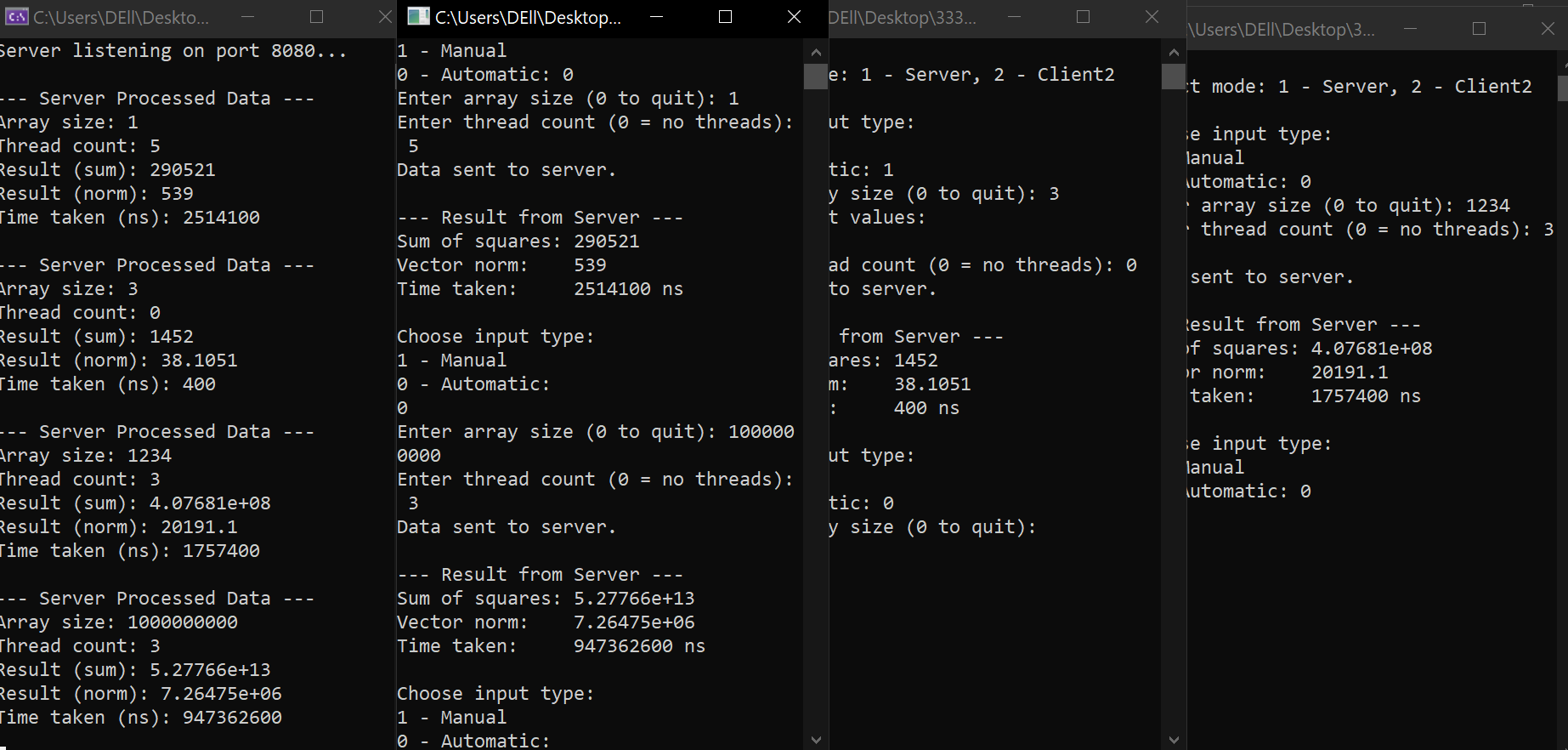


Рисунок 6 - Обробка значного масиву даних на першому клієнті

**Висновок**

У ході лабораторної роботи було реалізовано клієнт-серверну систему для обрахунку норми вектора. Програма містить серверну та клієнтську частину одночасно, функціонал залежить від вибору користувача для зручності. Програма підтримує кілька клієнтів та успішно із ними працює. Існує недолік із лімітом кількості потоків для обробки, тобто якщо вибрати кількадесят потоків на клієнтах - ноутбук не витримує таке навантаження.Також для зручності з одного клієнта можна надсилати кілька запитів на обчислення, щоб вийти з циклу треба ввести 0 для розміру вектору даних.

Посилання на репозиторій із матеріалами - [GitHub](https://github.com/anton265463/Labs_Parellel_Computing)

**Додаток із лістингом:**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <winsock2.h>

#include <vector>

#include <thread>

#include <chrono>

#include <numeric>

#include <random>

#include <iomanip>

#include <cstdlib>

#include <ctime>

#include <cmath>

#include <mutex>

#include <algorithm>

using namespace std;

using namespace std::chrono;

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

#define SERVER\_PORT 8080

#define BUFFER\_SIZE 1024

struct Result {

float sum;

float norm;

long long duration\_ns;

};

void handleError(const char\* message) {

cerr << "Error: " << message << " (" << WSAGetLastError() << ")" << endl;

exit(EXIT\_FAILURE);

}

float norm\_slow(const vector<int>& arr, float& sum, long long& duration) {

auto start = high\_resolution\_clock::now();

sum = 0;

for (int val : arr) {

sum += val \* val;

}

float norm = static\_cast<float>(sqrt(sum));

auto end = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(end - start).count();

return norm;

}

void worker(const vector<int>& arr, int start, int step, float& local\_sum) {

local\_sum = 0;

for (int i = start; i < (int)arr.size(); i += step) {

local\_sum += arr[i] \* arr[i];

}

}

float norm\_fast(const vector<int>& arr, int num\_threads, float& sum, long long& duration) {

vector<float> local\_sum(num\_threads, 0);

vector<thread> workers(num\_threads);

auto start = high\_resolution\_clock::now();

sum = 0;

for (int i = 0; i < num\_threads; ++i) {

workers[i] = thread(worker, cref(arr), i, num\_threads, ref(local\_sum[i]));

}

for (int i = 0; i < num\_threads; ++i) {

workers[i].join();

}

sum = accumulate(local\_sum.begin(), local\_sum.end(), 0.0f);

float norm = static\_cast<float>(sqrt(sum));

auto end = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(end - start).count();

return norm;

}

void processClient(SOCKET clientSocket) {

int mode;

int res = recv(clientSocket, (char\*)&mode, sizeof(mode), 0);

if (res <= 0) {

closesocket(clientSocket);

return;

}

send(clientSocket, "Mode received", 13, 0);

vector<int> array;

int size = 0;

if (mode == 1) {

res = recv(clientSocket, (char\*)&size, sizeof(size), 0);

if (res <= 0) {

closesocket(clientSocket);

return;

}

array.resize(size);

res = recv(clientSocket, (char\*)array.data(), size \* sizeof(int), 0);

if (res <= 0) {

closesocket(clientSocket);

return;

}

}

else {

res = recv(clientSocket, (char\*)&size, sizeof(size), 0);

if (res <= 0) {

closesocket(clientSocket);

return;

}

array.resize(size);

default\_random\_engine eng((unsigned)time(0));

uniform\_int\_distribution<int> dist(0, 1000);

for (int i = 0; i < size; ++i) array[i] = dist(eng);

}

int threadCount;

res = recv(clientSocket, (char\*)&threadCount, sizeof(threadCount), 0);

if (res <= 0) {

closesocket(clientSocket);

return;

}

float sum = 0;

float norm = 0;

long long duration\_ns = 0;

if (threadCount <= 0) {

norm = norm\_slow(array, sum, duration\_ns);

}

else {

norm = norm\_fast(array, threadCount, sum, duration\_ns);

}

Result result = { sum, norm, duration\_ns };

send(clientSocket, (char\*)&result, sizeof(result), 0);

cout << "\n--- Server Processed Data ---\n";

cout << "Array size: " << size << "\n";

cout << "Thread count: " << threadCount << "\n";

cout << "Result (sum): " << sum << "\n";

cout << "Result (norm): " << norm << "\n";

cout << "Time taken (ns): " << duration\_ns << "\n";

closesocket(clientSocket);

}

void startServer() {

WSADATA wsaData;

SOCKET serverSocket, clientSocket;

sockaddr\_in serverAddr, clientAddr;

int clientAddrSize = sizeof(clientAddr);

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) handleError("WSAStartup failed");

serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (serverSocket == INVALID\_SOCKET) handleError("Socket creation failed");

serverAddr.sin\_family = AF\_INET;

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

serverAddr.sin\_port = htons(SERVER\_PORT);

if (bind(serverSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)

handleError("Bind failed");

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) handleError("Listen failed");

cout << "Server listening on port " << SERVER\_PORT << "...\n";

vector<thread> clientThreads;

while (true) {

clientSocket = accept(serverSocket, (sockaddr\*)&clientAddr, &clientAddrSize);

if (clientSocket == INVALID\_SOCKET) {

cerr << "Accept failed: " << WSAGetLastError() << endl;

continue;

}

clientThreads.emplace\_back(thread([clientSocket]() {

processClient(clientSocket);

}));

clientThreads.erase(

remove\_if(clientThreads.begin(), clientThreads.end(),

[](thread& t) {

if (t.joinable() && (t.get\_id() != this\_thread::get\_id())) {

t.join();

return true;

}

return false;

}),

clientThreads.end());

}

for (auto& t : clientThreads) {

if (t.joinable()) t.join();

}

closesocket(serverSocket);

WSACleanup();

}

void startClient() {

WSADATA wsaData;

SOCKET clientSocket;

sockaddr\_in serverAddr;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) handleError("WSAStartup failed");

while (true) {

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket == INVALID\_SOCKET) handleError("Socket creation failed");

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

serverAddr.sin\_port = htons(SERVER\_PORT);

if (connect(clientSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

cerr << "Connection failed: " << WSAGetLastError() << endl;

closesocket(clientSocket);

continue;

}

int mode;

cout << "\nChoose input type:\n1 - Manual\n0 - Automatic: ";

cin >> mode;

send(clientSocket, (char\*)&mode, sizeof(mode), 0);

char buffer[BUFFER\_SIZE];

recv(clientSocket, buffer, BUFFER\_SIZE, 0);

int size;

cout << "Enter array size (0 to quit): ";

cin >> size;

if (size == 0) {

closesocket(clientSocket);

break;

}

send(clientSocket, (char\*)&size, sizeof(size), 0);

if (mode == 1) {

vector<int> array(size);

cout << "Enter " << size << " int values:\n";

for (int i = 0; i < size; ++i) {

cin >> array[i];

}

send(clientSocket, (char\*)array.data(), size \* sizeof(int), 0);

}

int threadCount;

cout << "Enter thread count (0 = no threads): ";

cin >> threadCount;

send(clientSocket, (char\*)&threadCount, sizeof(threadCount), 0);

cout << "Data sent to server.\n";

Result res;

recv(clientSocket, (char\*)&res, sizeof(res), 0);

cout << "\n--- Result from Server ---\n";

cout << "Sum of squares: " << res.sum << "\n";

cout << "Vector norm: " << res.norm << "\n";

cout << "Time taken: " << res.duration\_ns << " ns\n";

closesocket(clientSocket);

}

WSACleanup();

}

int main() {

int choice;

cout << "\nSelect mode: 1 - Server, 2 - Client\n";

cin >> choice;

if (choice == 1) {

startServer();

}

else if (choice == 2) {

startClient();

}

else {

cout << "Wrong choice" << endl;

}

return 0;

}