

**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  
Гладишев Андрій 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:
3. надсилання даних та конфігурації обчислень (наприклад, вказати кількість потоків для виконання обчислень)
4. надсилання команди (та отримання відповіді на команду) для початку обчислень
5. надсилання команди для запиту статусу і результату обчислень.
6. 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).
7. Застосунок 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.
8. В протоколі роботи необхідно навести опис розробленого застосунку. До цього опису повинні входити обґрунтування вибору протоколу передачі даних, а також архітектурний опис клієнта.
9. 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.
10. Занести до протоколу роботи UML діаграму викликів взаємодії серверу та клієнту, починаючи від запуску клієнту, до завершення роботи.
11. Надати висновок, що повинен містити аналіз та опис проблем з котрими зіштовхнувся студент, або з якими може зіштовхнутися розробник при організації міжпроцесової взаємодії.

**Варіант завдання – 5:**

Створити квадратну матрицю n×n та вектор розмірності n, n>=100; написати паралельну програму множення матриці на вектор.

**ХІД РОБОТИ**

Визначимо вимоги для клієнт-серверного застосунку:

Сервер повинен мати наступний функціонал:

* прийняття та асинхронна обробка клієнтів
* отримання від клієнтів даних про розмірність N матриці та вектору, кількість потоків для виконання завдання, безпосередньо матрицю та вектор тощо
* виконання задач
* отримання статусу виконання задачі
* відправка результатів назад до клієнта
* розрив з’єднання із клієнтом

Для реалізації передачі даних між клієнтом та сервером було обрано протокол TCP транспортного рівня, оскільки потрібно забезпечити надійність передачі даних.

При надходження запиту на підключення від клієнта, на стороні сервера створюється новий потік для обробки запита від цього клієнта для забезпечення асинхронності між багатьма клієнтами (що не є оптимальним у загальній практиці, коли багато клієнтів намагаються надіслати запити на один сервер, проте є достатнім у контексті завдання даної лабораторної роботи). Також окремий потік створюється для виконання задачі, дані якої надіслані клієнтом, тому є можливість отримання клієнтом статусу виконання у будь-який момент.

Клієнт у свою чергу має забезпечувати:

* підключення до серверу
* можливість обрати конкретну опцію з консолі:
  + відправляти дані до сервера
  + відправити серверу повідомлення про початок виконання задачі
  + отримання статусу виконання задачі (якщо вона виконалась – отримання результуючого вектора)
  + отримати результат
  + відправити серверу повідомлення про завершення з’єднання

|  |  |  |  |
| --- | --- | --- | --- |
| Дія | Складові | Формула / Аргументи | Опис логіки |
| Підключення до сервера | Встановлення з'єднання | WSADATA wsaData; int wsInit = WSAStartup(MAKEWORD(2, 2), &wsaData); SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); connect(clientSocket, (sockaddr\*)&addr, sizeof(addr)); | Якщо з'єднання успішне — клієнт підключається до сервера та отримує підтвердження. |
| Надсилання підтвердження про старт | Команда 'M': матриця, вектор, потоки | message.setCommand('M'); message.setUserMatrix(...); message.setUserVector(...); message.setUserNumThreads(...); send(clientSocket, message.encode(), message.getTotalsize(), 0); | Сервер приймає початкові дані для обробки й повертає 'C'. |
| Запуск обчислення | Команда 'P' | message.setCommand('P'); send(clientSocket, message.encode(), message.getTotalsize(), 0); | Сервер викликає `Matrix::multiplyMatrixVectorParallel()` та відповідає 'S'. |
| Перевірка статусу виконання | Команда 'L' | message.setCommand('L'); send(...); recv(...); | Сервер надсилає статус: 'W' (очікує), 'I' (виконується), 'F' (завершено), 'D' (є дані, очікує запуску). |
| Отримання результату | Команда 'R' | message.setCommand('R'); send(...); recv(...); message.decode(...); | Сервер відповідає командою 'O' та надсилає результат множення, тривалість і кількість потоків. |
| Завершення сесії | Команда 'Q' | message.setCommand('Q'); send(...); closesocket(clientSocket); | Сервер надсилає 'B' і закриває з'єднання. |
| Обробка помилкової команди | Будь-яка невідома команда | message.setCommand('X'); // Наприклад send(...); recv(...); | Сервер відповідає 'u', що означає — команда не розпізнана. |

Зобразимо UML-діаграму нашого застосунку:

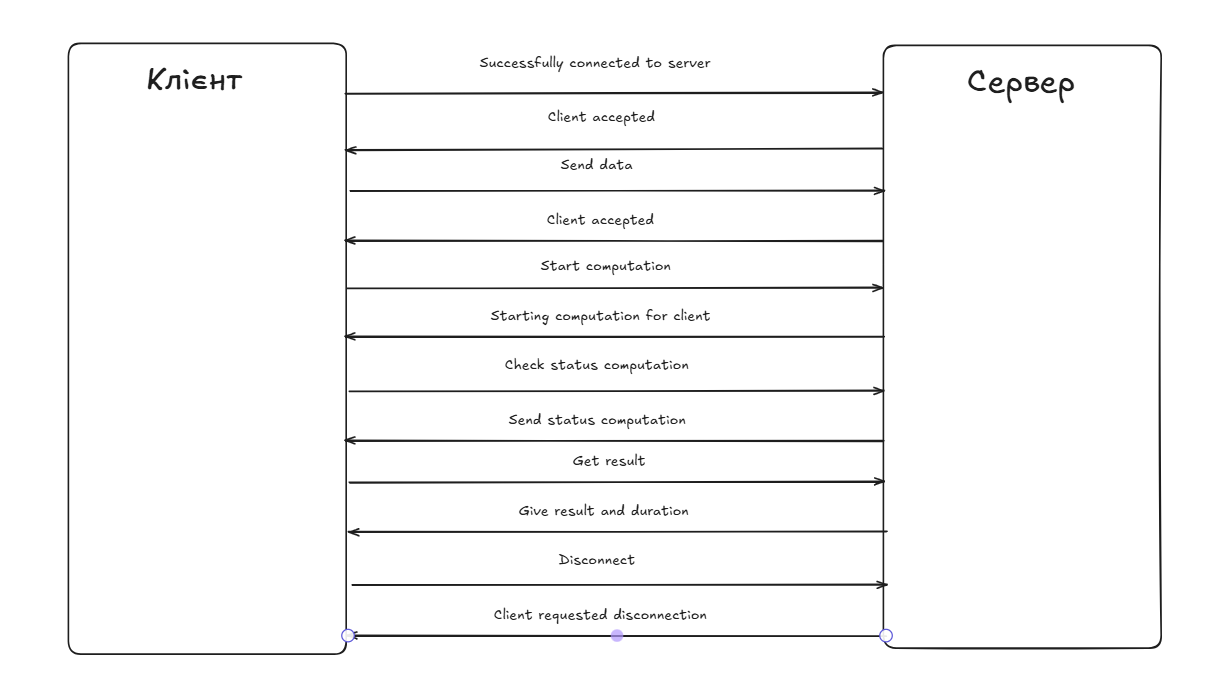


Рисунок 1 — UML-діаграма застосунку.

Далі протестуємо роботу нашого сервера, запустивши майже одночасно двох клієнтів, підключених до сервера.

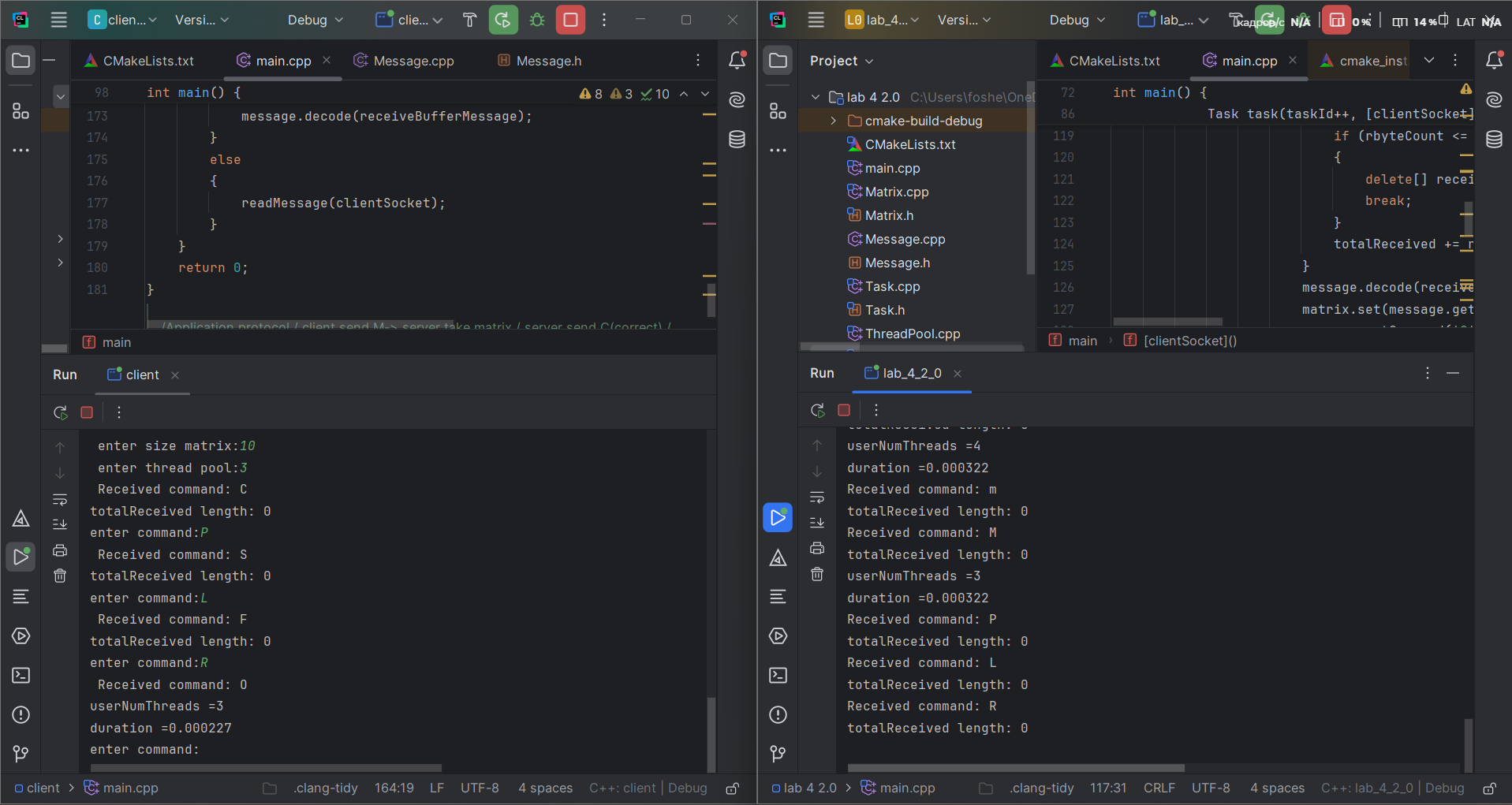


Рисунок 2 – Дії першого клієнта з сервером

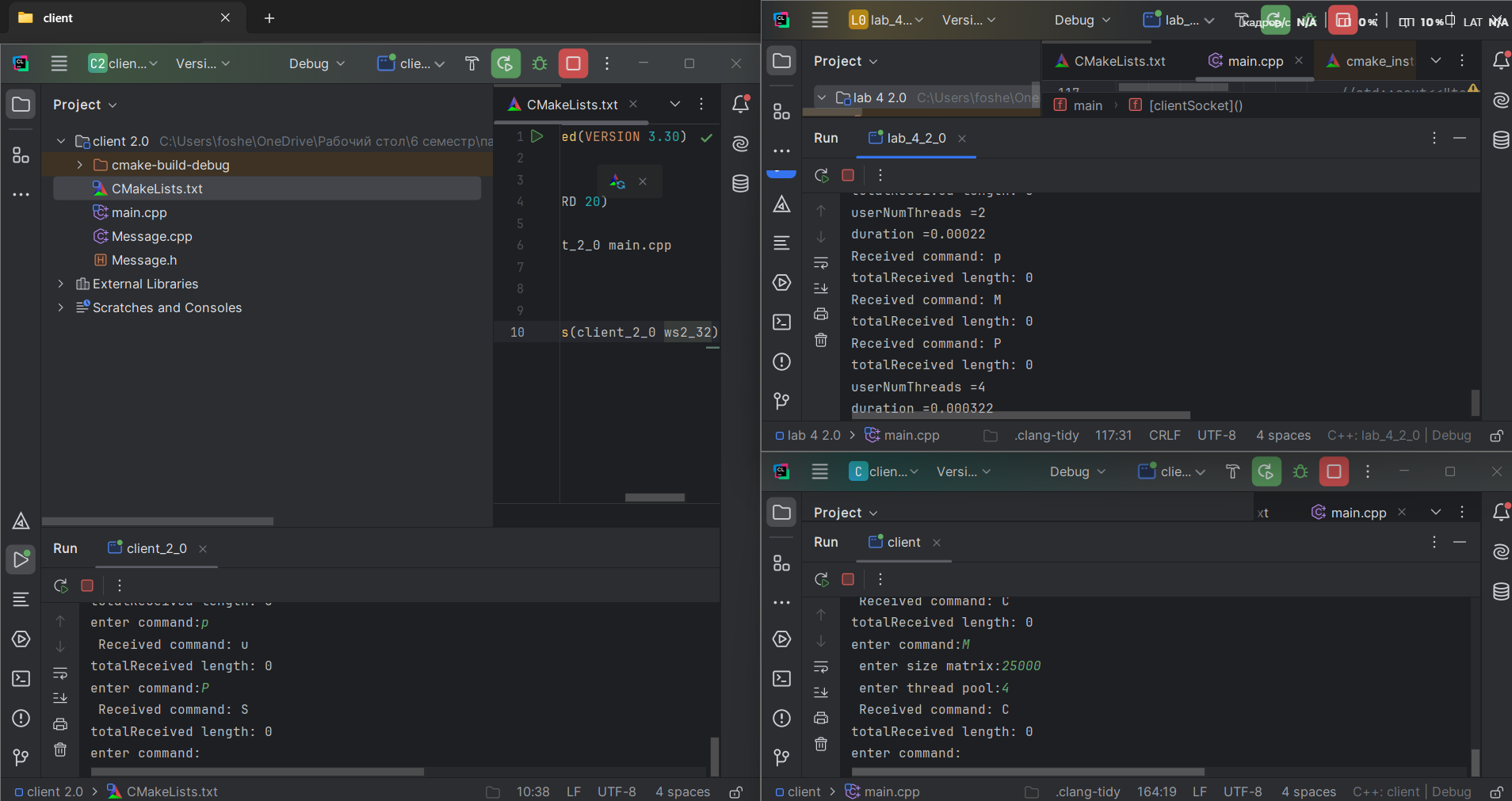


Рисунок 3 — Одночасна робота з двома клієнтами.

З логів серверу можна побачити, що поки перший клієнт ще надсилає дані, другий надсилає власні запити та отримує результати виконання від серверу.

**ВИСНОВКИ**

У ході виконання лабораторної роботи було розроблено систему взаємодії між клієнтом і сервером з використанням мови програмування C++ та бібліотеки Winsock2. Обмін даними здійснюється за допомогою протоколу TCP, що гарантує надійність і цілісність передавання.

Архітектура системи побудована таким чином, що сервер може одночасно обслуговувати кілька клієнтів, виділяючи для кожного окремий потік обробки. Клієнт надсилає серверу вхідні дані для обчислень — зокрема, матрицю, вектор і кількість потоків — після чого ініціює початок обробки. Сервер виконує паралельне множення матриці на вектор, розподіляючи завдання між потоками, відстежує статус виконання й реагує на відповідні запити клієнта. По завершенню обчислень клієнт отримує готовий результат.

Такий підхід ілюструє ефективність багатопотокової обробки, проте водночас постає проблема масштабованості: зі зростанням кількості підключень навантаження на систему збільшується, що може негативно впливати на стабільність роботи сервера.

Для полегшення розуміння логіки взаємодії було створено таблицю команд протоколу та UML-діаграму послідовності, які відображають основні етапи обміну повідомленнями між клієнтом і сервером.