

**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-21  
Терещенко Олексій Ігорович

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.

**ХІД РОБОТИ**

Програма реалізує клієнт-серверну архітектуру для обробки матриць у багатьох потоках. Клієнт передає матрицю та конфігурацію потоків на сервер, який виконує обчислення. Після завершення обчислень сервер повертає результати клієнту.

Клієнт:

* Ініціює введення параметрів обчислень: визначає розмір матриці та кількість робочих потоків (опція 1).
* Генерує матрицю заданого розміру та передає її на сервер (опція 2).
* Ініціює початок обчислень на сервері (опція 3).
* Періодично запитує статус обчислень (опція 4) та, після їх завершення, отримує результуючу матрицю (опція 4).

Сервер:

* Приймає від клієнта параметри конфігурації та дані матриці.
* Виконує багатопотокове обчислення мінімальних елементів у стовпцях матриці відповідно до отриманої конфігурації.
* Надсилає клієнту проміжні коди стану та фінальний результат обробки.

Протокол прикладного рівня (Application Protocol) визначає команди, які використовуються для взаємодії між клієнтом і сервером. У таблиці нижче наведено опис команд, аргументів та можливих відповідей.

|  |  |  |  |
| --- | --- | --- | --- |
| Команда | Тег | Формат аргументів | Можливі відповіді |
| CONFIG | 0x01 | • uint32\_t n – розмір матриці (nxn) • uint32\_t threads – кількість потоків | STATUS\_RESP (0x06): • 0x00 – STATUS\_NOT\_STARTED |
| MATRIX | 0x02 | • n×n значень int32\_t (мережевий порядок байтів) | — (немає негайної відповіді) |
| START\_PROCESS | 0x03 | — | STATUS\_RESP (0x06): • 0x01 – STATUS\_IN\_PROGRESS |
| STATUS\_REQUEST | 0x04 | — | • Якщо обчислення не завершене: STATUS\_RESP (0x06) з кодом 0x00 або 0x01 • Якщо завершене: RESULT (0x05) з серіалізованою матрицею |
| RESULT | 0x05 | • n×n значень int32\_t – результат обробки | — (сервер надсилає цей пакет як відповідь на STATUS\_REQUEST) |
| STATUS\_RESP | 0x06 | • uint8\_t statusCode   0x00 – NOT\_STARTED   0x01 – IN\_PROGRESS   0x02 – FINISHED | — |

Також зобразимо UML-діаграму нашого застосунку:

A diagram of a diagram

AI-generated content may be incorrect.

Рисунок 1 — UML-діаграма застосунку.

Далі протестуємо роботу нашого сервера, запустивши одночасно двох клієнтів, підключених до сервера (один на С++, інший на TypeScript).

A screenshot of a computer program

AI-generated content may be incorrect.

Рисунок 2 — Запуск серверу.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 3 — Одночасна робота з двома клієнтами.

**ВИСНОВКИ**

У процесі розробки клієнт‑серверного застосунку для лабораторної роботи № 1 найбільшою складністю виявилося створення надійного та одночасно гнучкого прикладного протоколу обміну даними. Використання TLV‑структури дозволило однозначно розділяти заголовки й корисне навантаження, однак вимагало ретельної обробки порядку байтів (htonl/ntohl) і коректного читання/запису частин повідомлення (recvAll/sendAll).

Під час реалізації багатопотокового серверу довелося приділяти особливу увагу синхронізації доступу до спільного стану сеансу (SessionState): без правильного використання mutex і lock\_guard легко допустити гонки даних або блокування.

Підключення кількох клієнтів одночасно вимагає масштабованої обробки accept() та детачення (або управління) потоків‑обробників. Тут критичними виявилися питання обробки помилок на рівні сокета: необхідно було забезпечити коректне закриття з’єднання при виникненні виключень і відмов з боку клієнта, зберігаючи працездатність інших сесій.

При розробці другого клієнта на TypeScript виникли додаткові виклики: розбивка буфера на частини в JavaScript, асинхронне очікування повідомлень і гарантування цілісності TLV‑пакетів. Взаємодія між компонентами на різних мовах програмування підкреслює важливість документування протоколу — детальна таблиця команд і UML‑діаграма послідовностей дозволяють швидко адаптувати нові реалізації клієнтів чи серверів.