**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  
Моренець Сергій Андрійович

Прийняв:

асистент Яременко В. С.

Київ – 2025

***Завдання:***

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:
   1. надсилання даних та конфігурації обчислень (наприклад, вказати кількість потоків для виконання обчислень)
   2. надсилання команди (та отримання відповіді на команду) для початку обчислень
   3. 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. 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.
5. В протоколі роботи необхідно навести опис розробленого застосунку. До цього опису повинні входити обґрунтування вибору протоколу передачі даних, а також архітектурний опис клієнта.
6. 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.
7. 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.

**Application protocol:**

C -> S: HELLO (Tag 0x00, Length 0x0005, Value "HELLO")

S -> C: WELCOME (Tag 0x01, Length 0x0007, Value "WELCOME")

C -> S: SET\_CONFIG (Tag 0x02, Length 0x0004, Value [uint32: 8])

C -> S: SET\_SIZE (Tag 0x03, Length 0x0004, Value [uint32: 100])

C -> S: SEND\_DATA

(Tag 0x04, Length 0xN^2, Value [N×N int32 matrix])

S -> C: EXEC\_STARTED (Tag 0x05, Length 0x0001, Value 0x00)

[loop while processing:]

C -> S: STATUS\_REQUEST (Tag 0x06, Length 0x0000, Value -)

S -> C: IN\_PROGRESS

(Tag 0x07, Length 0x000B, Value "IN\_PROGRESS")

[when done:]

C -> S: STATUS\_REQUEST (Tag 0x06, Length 0x0000, Value -)

S -> C: EXEC\_RESULT

(Tag 0x08, Length 0x0004, Value [uint32: execution time])

S -> C: MATRIX\_RESULT

(Tag 0x09, Length 0xN^2, Value [N×N int32 matrix])

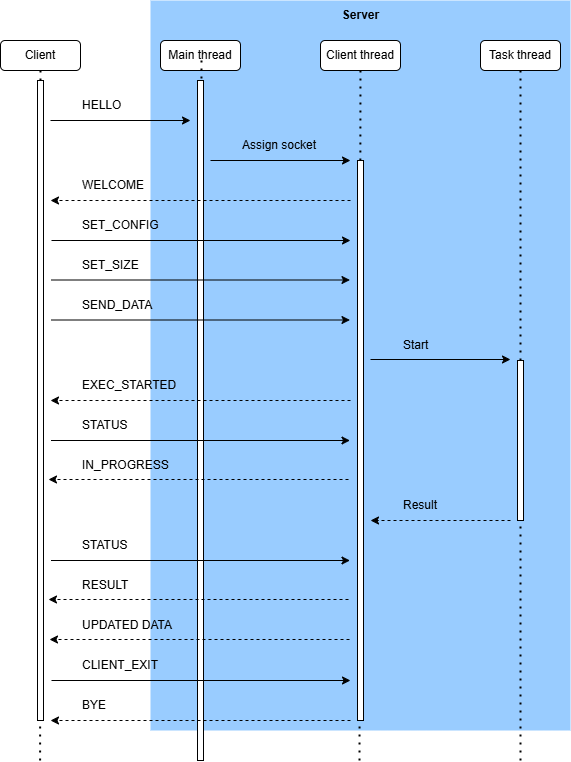
C -> S: CLIENT\_EXIT (Tag 0x0A, Length 0x0000, Value -)

S -> C: BYE (Tag 0x0B, Length 0x0000, Value -)

**Опис прикладного протоколу взаємодії між клієнтом і сервером:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва команди** | **Тип (код)** | **Аргументи / Дані** | **Опис команди** | **Відповідь сервера** |
| HELLO | 0x00 | Текст: HELLO | Клієнт ініціалізує з'єднання | 0x01 – WELCOME |
| WELCOME | 0x01 | Текст: WELCOME | Відповідь сервера на HELLO | - |
| SET\_CONFIG | 0x02 | 4 байти: кількість потоків (uint32) | Клієнт задає кількість потоків для обчислень | - |
| SET\_SIZE | 0x03 | 4 байти: розмір матриці N (uint32) | Клієнт задає розмір квадратної матриці N x N | - |
| SEND\_DATA | 0x04 | N×N×4 байти: матриця int32 | Відправка сировинної матриці серверу | 0x05 – EXEC\_STARTED |
| EXEC\_STARTED | 0x05 | 1 байт: статус (наприклад, 0x00) | Сервер підтверджує початок обробки | - |
| STATUS\_REQUEST | 0x06 | - | Клієнт перевіряє стан обробки матриці | 0x07 або 0x08 + 0x09 |
| IN\_PROGRESS | 0x07 | Текст: IN\_PROGRESS | Сервер ще обробляє матрицю | - |
| EXEC\_RESULT | 0x08 | 4 байти: час виконання (uint32 мс) | Сервер надсилає час обробки після завершення | - |
| MATRIX\_RESULT | 0x09 | N×N×4 байти: оброблена матриця | Сервер повертає оновлену (перетворену) матрицю | - |
| CLIENT\_EXIT | 0x0A | - | Клієнт завершує сеанс | 0x0B – BYE |
| BYE | 0x0B | - | Сервер підтверджує завершення з'єднання | - |

**Модель взаємодії клієнта та сервера:**



***Висновок:***

У ході реалізації даного застосунку було спроєктовано TLV-протокол, який забезпечує обмін даними між клієнтом і сервером у форматі "Type-Length-Value". Як протокол передачі даних обрано TCP, що зумовлено необхідністю гарантованої доставки, збереження порядку повідомлень та цілісності при передачі великих обсягів даних.

Архітектура клієнта є подієвою: після встановлення з'єднання клієнт ініціалізує сеанс, передає конфігураційні параметри (кількість потоків, розмір матриці) та саму матрицю. Після цього клієнт циклічно опитує сервер про статус виконання і отримує результат обчислень та оновлену матрицю. Усі повідомлення відправляються у TLV-структурі з фіксованим заголовком (1 байт тегу + 2 байти довжини) та змінною частиною значення.

У процесі реалізації було враховано типові проблеми міжпроцесової взаємодії:

* Складність синхронізації потоків на сервері (обчислення виконуються асинхронно, з контролем через atomic<bool>).
* Перевірка цілісності даних: важливо перевіряти тип та довжину TLV перед обробкою.
* Обробка помилок мережевого рівня: можливість закриття з’єднання або втрати зв’язку вимагає обробки винятків та завершення сесії.

Також можливо зіткнутися з проблемами масштабування — при великій кількості клієнтів можуть виникнути проблеми з необмеженими потоками. Ще однією складністю є забезпечення універсальності формату TLV – він не є досконалим, та підтримує не досить великі обсяги даних.