**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені Ігоря Сікорського»**

**Лабораторна робота № 4**

**з курсу «Паралельні обчислення»**

**на тему «Елементи міжпроцесної взаємодії»**

**Виконав студент ІПСА групи ДА-22**

**Котляр Єгор Ярославович**

Київ – 2025

**Мета роботи**

Детально розглянути поняття процесу, ресурсів процесу, а також підходи до міжпроцесної взаємодії, як в межах одного ПК, так і на різних ПК. Отримати практичні навички організації міжпроцесної взаємодії на прикладі написання клієнт-серверного застосунку.

**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:

* надсилання даних та конфігурації обчислень (наприклад, вказати кількість потоків для виконання обчислень)
* надсилання команди (та отримання відповіді на команду) для початку обчислень
* 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.

**Варіант 14**: заповнити квадратну матрицю випадковими числами. На головній діагоналі розмістити максимальний елемент стовпчика.

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

**Хід виконання роботи**

За варіантом з першої лабораторної роботи виконаємо завдання, де створимо й використаємо клієнт-серверний застосунок:

**Код програми:**

**server.cpp:**

#include <winsock2.h>  
#include <windows.h>  
#include <iostream>  
#include <vector>  
#include <thread>  
#include <stdexcept>  
#include <chrono>  
#include <mutex>  
  
using namespace std;  
  
constexpr uint16\_t CMD\_INIT = 0x01;  
constexpr uint16\_t CMD\_START = 0x02;  
constexpr uint16\_t CMD\_STATUS = 0x03;  
constexpr uint16\_t RSP\_INIT = 0x11;  
constexpr uint16\_t RSP\_START = 0x12;  
constexpr uint16\_t RSP\_STATUS = 0x13;  
  
mutex cout\_mutex;  
  
void printError(const char\* msg) {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cerr << msg << " не вдався через помилку: " << WSAGetLastError() << endl;  
}  
  
void recvAll(SOCKET sock, char\* buf, int len) {  
 int received = 0;  
 while (received < len) {  
 int r = recv(sock, buf + received, len - received, 0);  
 if (r < 0) throw runtime\_error("recv не вдався");  
 if (r == 0) throw runtime\_error("З'єднання роз'єднано клієнтом");  
 received += r;  
 }  
}  
  
void sendAll(SOCKET sock, const char\* buf, int len) {  
 int sent = 0;  
 while (sent < len) {  
 int s = send(sock, buf + sent, len - sent, 0);  
 if (s == SOCKET\_ERROR) throw runtime\_error("відправка не вдалася");  
 sent += s;  
 }  
}  
  
void parallelColumnMax(const vector<vector<uint32\_t>>& mat, uint32\_t N, vector<uint32\_t>& result, uint32\_t start, uint32\_t end) {  
 for (uint32\_t j = start; j < end; ++j) {  
 uint32\_t mx = mat[0][j];  
 for (uint32\_t i = 1; i < N; ++i) {  
 if (mat[i][j] > mx) mx = mat[i][j];  
 }  
 result[j] = mx;  
 }  
}  
  
void computeColumnMaxParallel(const vector<vector<uint32\_t>>& mat, uint32\_t N, vector<uint32\_t>& result, uint32\_t T) {  
 result.assign(N, 0);  
 vector<thread> threads;  
 threads.reserve(T);  
  
 uint32\_t base = N / T;  
 uint32\_t rem = N % T;  
 uint32\_t start = 0;  
 for (uint32\_t t = 0; t < T; ++t) {  
 uint32\_t extra = (t < rem) ? 1 : 0;  
 uint32\_t end = start + base + extra;  
 threads.emplace\_back(parallelColumnMax, cref(mat), N, ref(result), start, end);  
 start = end;  
 }  
 for (auto &th : threads) th.join();  
}  
  
void handleClient(SOCKET client) {  
 {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cout << "[Сервер] Клієнт під'єднався: " << client << endl;  
 }  
 try {  
 uint16\_t cmd;  
 uint32\_t N = 0;  
 vector<vector<uint32\_t>> mat;  
 vector<uint32\_t> result;  
 bool dataReady = false;  
  
 while (true) {  
 recvAll(client, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 cmd = ntohs(cmd);  
 if (cmd == CMD\_INIT) {  
 recvAll(client, reinterpret\_cast<char\*>(&N), sizeof(N));  
 N = ntohl(N);  
 mat.assign(N, vector<uint32\_t>(N));  
 for (uint32\_t i = 0; i < N; ++i) {  
 for (uint32\_t j = 0; j < N; ++j) {  
 uint32\_t v;  
 recvAll(client, reinterpret\_cast<char\*>(&v), sizeof(v));  
 mat[i][j] = ntohl(v);  
 }  
 }  
 dataReady = true;  
 {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cout << "[Сервер] INIT отримано: N = " << N << endl;  
 }  
 uint16\_t rsp = htons(RSP\_INIT);  
 sendAll(client, reinterpret\_cast<char\*>(&rsp), sizeof(rsp));  
  
 } else if (cmd == CMD\_START) {  
 if (!dataReady) throw runtime\_error("Дані не ініціалізовано");  
 uint32\_t netT;  
 recvAll(client, reinterpret\_cast<char\*>(&netT), sizeof(netT));  
 uint32\_t T = ntohl(netT);  
 {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cout << "[Сервер] START отрмано: потоки = " << T << endl;  
 }  
  
 auto t1 = chrono::high\_resolution\_clock::*now*();  
 computeColumnMaxParallel(mat, N, result, T);  
 auto t2 = chrono::high\_resolution\_clock::*now*();  
 double dur = chrono::duration<double>(t2 - t1).count();  
  
 uint16\_t rsp = htons(RSP\_START);  
 sendAll(client, reinterpret\_cast<char\*>(&rsp), sizeof(rsp));  
 sendAll(client, reinterpret\_cast<char\*>(&dur), sizeof(dur));  
  
 } else if (cmd == CMD\_STATUS) {  
 if (!dataReady) throw runtime\_error("Дані не ініціалізовано");  
 {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cout << "[Сервер] запит STATUS, відправка результатів..." << endl;  
 }  
 uint16\_t rsp = htons(RSP\_STATUS);  
 sendAll(client, reinterpret\_cast<char\*>(&rsp), sizeof(rsp));  
 uint32\_t netN = htonl(N);  
 sendAll(client, reinterpret\_cast<char\*>(&netN), sizeof(netN));  
 for (uint32\_t v : result) {  
 uint32\_t nv = htonl(v);  
 sendAll(client, reinterpret\_cast<char\*>(&nv), sizeof(nv));  
 }  
 break;  
  
 } else {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cerr << "[Сервер] Незрозуміла команда: " << cmd << endl;  
 break;  
 }  
 }  
 } catch (const exception& e) {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cerr << "[Сервер] Помилка клієнта " << client << ": " << e.what() << endl;  
 }  
 closesocket(client);  
 {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cout << "[Сервер] Клієнт " << client << " від'єднався" << endl;  
 }  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
 WSADATA wsa;  
 if (WSAStartup(MAKEWORD(2,2), &wsa) != 0) {  
 printError("WSAStartup");  
 return 1;  
 }  
  
 SOCKET listenSock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  
 if (listenSock == INVALID\_SOCKET) {  
 printError("socket");  
 WSACleanup();  
 return 1;  
 }  
  
 sockaddr\_in addr{};  
 addr.sin\_family = AF\_INET;  
 addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 addr.sin\_port = htons(1234);  
 if (bind(listenSock, reinterpret\_cast<sockaddr\*>(&addr), sizeof(addr)) == SOCKET\_ERROR ||  
 listen(listenSock, SOMAXCONN) == SOCKET\_ERROR) {  
 printError("bind/listen");  
 closesocket(listenSock);  
 WSACleanup();  
 return 1;  
 }  
  
 {  
 lock\_guard<mutex> lock(cout\_mutex);  
 cout << "[Сервер] Працює на порті 1234..." << endl;  
 }  
  
 while (true) {  
 SOCKET client = accept(listenSock, nullptr, nullptr);  
 if (client == INVALID\_SOCKET) {  
 printError("accept");  
 continue;  
 }  
 thread(handleClient, client).detach();  
 }  
  
 closesocket(listenSock);  
 WSACleanup();  
 return 0;  
}

**client.cpp:**

#include <winsock2.h>  
#include <windows.h>  
#include <iostream>  
#include <vector>  
#include <thread>  
#include <random>  
  
using namespace std;  
  
constexpr uint16\_t CMD\_INIT = 0x01;  
constexpr uint16\_t CMD\_START = 0x02;  
constexpr uint16\_t CMD\_STATUS = 0x03;  
constexpr uint16\_t RSP\_INIT = 0x11;  
constexpr uint16\_t RSP\_START = 0x12;  
constexpr uint16\_t RSP\_STATUS = 0x13;  
  
int sendAll(SOCKET sock, const char\* buf, int len) {  
 int total = 0;  
 while (total < len) {  
 int sent = send(sock, buf + total, len - total, 0);  
 if (sent == SOCKET\_ERROR) return SOCKET\_ERROR;  
 total += sent;  
 }  
 return total;  
}  
  
int recvAll(SOCKET sock, char\* buf, int len) {  
 int total = 0;  
 while (total < len) {  
 int rec = recv(sock, buf + total, len - total, 0);  
 if (rec <= 0) return rec;  
 total += rec;  
 }  
 return total;  
}  
  
vector<vector<int>> createRandomMatrix(int n) {  
 vector<vector<int>> mat(n, vector<int>(n));  
 for (int i = 0; i < n; ++i) {  
 for (int j = 0; j < n; ++j) {  
 mat[i][j] = rand() % 1001;  
 }  
 }  
 return mat;  
}  
  
void run\_client(int client\_id, const string& server\_ip, int port) {  
 SOCKET sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  
 if (sock == INVALID\_SOCKET) {  
 cerr << "[Клієнт " << client\_id << "] socket не вдався" << endl;  
 return;  
 }  
  
 sockaddr\_in serverAddr{};  
 serverAddr.sin\_family = AF\_INET;  
 serverAddr.sin\_addr.s\_addr = inet\_addr(server\_ip.c\_str());  
 serverAddr.sin\_port = htons(port);  
 if (connect(sock, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) {  
 cerr << "[Клієнт " << client\_id << "] не вдалося під'єднатися" << endl;  
 closesocket(sock);  
 return;  
 }  
  
 try {  
 const int N = 10000;  
 auto matrix = createRandomMatrix(N);  
  
 // INIT  
 uint16\_t cmd = htons(CMD\_INIT);  
 sendAll(sock, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 uint32\_t netN = htonl(N);  
 sendAll(sock, reinterpret\_cast<char\*>(&netN), sizeof(netN));  
 for (int i = 0; i < N; ++i) {  
 for (int j = 0; j < N; ++j) {  
 uint32\_t v = htonl(static\_cast<uint32\_t>(matrix[i][j]));  
 sendAll(sock, reinterpret\_cast<char\*>(&v), sizeof(v));  
 }  
 }  
 recvAll(sock, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 if (ntohs(cmd) != RSP\_INIT) throw runtime\_error("INIT не вдався");  
 cout << "[Клієнт " << client\_id << "] INIT ACK" << endl;  
  
 // START  
 for (uint32\_t T : {1u, 16u}) {  
 cmd = htons(CMD\_START);  
 sendAll(sock, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 uint32\_t netT = htonl(T);  
 sendAll(sock, reinterpret\_cast<char\*>(&netT), sizeof(netT));  
  
 recvAll(sock, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 if (ntohs(cmd) != RSP\_START) throw runtime\_error("START не вдвався");  
 double dur;  
 recvAll(sock, reinterpret\_cast<char\*>(&dur), sizeof(dur));  
 cout << "[Клієнт " << client\_id << "] потоки = " << T  
 << ", час виконання: " << dur << " сек" << endl;  
 }  
  
 // STATUS  
 cmd = htons(CMD\_STATUS);  
 sendAll(sock, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 recvAll(sock, reinterpret\_cast<char\*>(&cmd), sizeof(cmd));  
 if (ntohs(cmd) != RSP\_STATUS) throw runtime\_error("STATUS не вдався");  
 uint32\_t netResN;  
 recvAll(sock, reinterpret\_cast<char\*>(&netResN), sizeof(netResN));  
 int resN = ntohl(netResN);  
 vector<uint32\_t> result(resN);  
 for (int i = 0; i < resN; ++i) {  
 uint32\_t v;  
 recvAll(sock, reinterpret\_cast<char\*>(&v), sizeof(v));  
 result[i] = ntohl(v);  
 }  
 } catch (const exception& e) {  
 cerr << "[Клієнт " << client\_id << "] Помилка: " << e.what() << endl;  
 }  
  
 closesocket(sock);  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
 WSADATA wsa;  
 if (WSAStartup(MAKEWORD(2,2), &wsa) != 0) {  
 cerr << "WSAStartup не вдався" << endl;  
 return 1;  
 }  
  
 string server\_ip = "127.0.0.1";  
 int port = 1234;  
  
 thread c1(run\_client, 1, server\_ip, port);  
 thread c2(run\_client, 2, server\_ip, port);  
  
 c1.join();  
 c2.join();  
  
 WSACleanup();  
 return 0;  
}

Вхідні дані – це фіксований розмір квадратної матриці (10000х10000) та кількість потоків для розв’язання задачі (для порівняння взято 1 потік, тобто послідовне вирішення, та 16 потоків – найоптимальніша кількість, що співпадає з кількістю моїх логічних ядер):

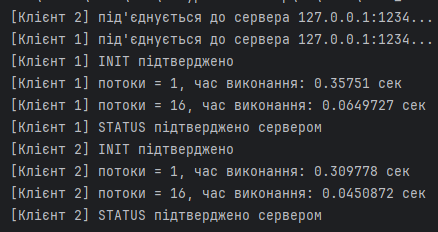


Рисунок 1 – Логи виконання для 2 клієнтів.

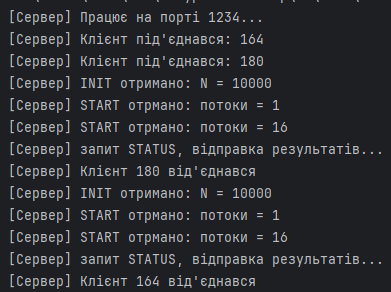


Рисунок 2 – Логи виконання сервера.

У розробленому застосунку для обміну даними між клієнтом і сервером обрано протокол TCP, оскільки він забезпечує гарантовану доставку байтів у правильному порядку, інтегровану перевірку цілісності та автоматичне відновлення втрачених сегментів. Це особливо важливо при передачі великих обсягів даних, як-от матриці розміром 10000×10000, адже будь-які втрати або перестановки призвели б до некоректного обчислення максимумів по стовпцях.

Архітектура клієнта:

* Головна функція створює два незалежних потоки, кожен з яких встановлює TCP-з’єднання із сервером.
* Ініціалізується генератор випадкових чисел, формується матриця, надсилається до сервера запит INIT з розміром і вмістом матриці, очікуємо підтвердження.
* Послідовно надсилаємо команди START з різною кількістю потоків для обчислення (T=1 і T=16).
* Отримуємо та виводимо час виконання кожного обчислення.
* Наприкінці запитує статус (STATUS) і приймає вектор результатів.

У протоколі прикладного рівня передбачено три основні команди:

* Команда INIT (0x01) – приймає на вхід 32-бітне значення N і далі послідовність N×N 32-бітних чисел, тобто елементів матриці. У відповідь сервер надсилає код RSP\_INIT (0x11), який означає готовність обробляти матрицю.
* Команда START (0x02) – приймає 32-бітне число T (кількість робочих потоків) для обчислення максимуму по стовпцях. Після отримання команди сервер відповідає кодом RSP\_START (0x12), а потім відправляє 64-бітне число з виміряним часом виконання обчислення.
* Команда STATUS (0x03) – не потребує аргументів: сервер у відповідь надсилає код RSP\_STATUS (0x13), потім 32-бітне значення N і підряд N 32-бітних чисел, які є результатами обчислення максимальних елементів кожного стовпця.

|  |  |  |  |
| --- | --- | --- | --- |
| Команда | Аргументи | Опис | Відповідь |
| INIT (0x01) | NxN (uint32\_t) – розмір матриці | Передає серверу квадратну матрицю для подальшої обробки. | RSP\_INIT (0x11) – підтвердження прийому. |
| START (0x02) | T (uint32\_t) – кількість потоків для обчислення. | Запускає на сервері паралельне обчислення максимуму кожного стовпця матриці з T потоками. | RSP\_START (0x12) та dur (8 байт, час виконання в секундах) – виміряний час обчислення. |
| STATUS (0x03) | – | Запитує сервер статус обчислення (чи завершено). | RSP\_STATUS (0x13) – сповіщення про закінчення. |

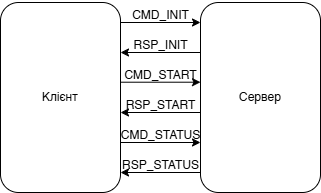


Рисунок 3 – UML діаграма.

**Висновок**

У ході розробки було реалізовано клієнт-серверний застосунок, що взаємодіє за допомогою протоколу TCP. Було створено власний протокол прикладного рівня, який включає команди для ініціалізації даних, запуску обчислень та запиту результатів. Клієнт формує велику квадратну матрицю розміром 10000х10000 елементів, передає її на сервер і ініціює обчислення максимальних елементів у кожному стовпці як в одному потоці, так і в шістнадцяти потоках для порівняння ефективності.