

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«Київський політехнічний інститут ім. І. Сікорського»**

**Інститут Прикладного Системного Аналізу**

**Кафедра Системного Проектування**

Лабораторна робота №4

З дисципліни «Паралельні обчислення»

Виконав:

Ст. гр. ДА-12

Кракович Павло

**Лабораторна робота №4**

**Дослідження міжпроцесної взаємодії**

**Мета роботи:** Детально розглянути поняття процесу, ресурсів процесу, атакож підходи до міжпроцесової взаємодії, як в межах одного ПК, так і нарізних ПК. Отримати практичні навички організації міжпроцесової взаємодіїна прикладі написання клієнт-серверного застосунку.

**Завдання:**

1. Розробити клієнт-серверний застосунок для вирішення завдання з лабораторний робот 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. Розробити протокол прикладного рівня для взаємодії клієнта з сервером. Для цього врахувати декілька кроків в процесі взаємодії. Надсилання даних та конфігурації обчислень, надсилання команди (та отримання відповіді на команду) для початку обчислень та команди для запиту статусу і результату обчислень.

3. Додати до розробленого сервер 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.

1. **Опис розробленого застосунку**

Код лабораторної роботи реалізує простий клієнт-серверний додаток для виконання матричних обчислень на стороні сервера. Клієнт надсилає дві матриці на сервер, запитує обчислення, а потім отримує отриману матрицю.

Програма використовує протокол TCP (Transmission Control Protocol) для передачі даних. TCP - це надійний, орієнтований на з'єднання протокол, який забезпечує впорядковану доставку пакетів даних і перевірку помилок. Це має вирішальне значення для передачі матриць, оскільки пошкодження даних може призвести до неправильних результатів обчислень.

Архітектурний опис клієнта:

Клієнт у власному потоці надсилає на сервер дві матриці за допомогою TCP-з'єднання. Після надсилання матриць клієнт відправляє запит серверу на обробку даних. Після отримання результату обчислень він виводить його на екран і завершує роботу (підключення до серверу).

Архітектурний опис серверу:

Сервер чекає на вхідні підключення клієнтів і обробляє їх у окремих потоках. Після отримання матриць від клієнта та команди про запуск обробки, сервер запускає обчислення, розділяючи обчислення між декількома потоками. Після завершення обчислень сервер надсилає результат клієнту та зві.

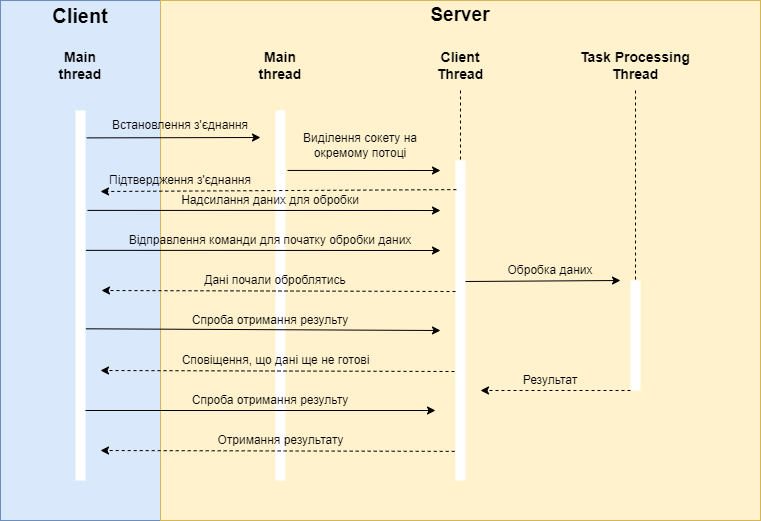
Клієнт та серверу взаємодіють за допомогою прикладного протоколу на основні TCP. Протокол можна вважати прикладним тому, що він визначає спосіб обміну даними між клієнтом і сервером на вищому рівні абстракції, що є характерною особливістю прикладних протоколів.

Протокол визначає конкретні команди (Request та Response), які клієнт та сервер можуть взаємодіяти. Це дозволяє сторонам взаємодіяти на рівні вищого рівня абстракції, не звертаючи увагу на деталі роботи мережевого рівня. Також він визначає абстрактні операції, такі як відправка даних, запуск обчислення та отримання результатів. Це дозволяє працювати з протоколом на вищому рівні, не звертаючи увагу на деталі роботи TCP.

1. **Опис розробленого протоколу прикладного рівня:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Команда** | **Аргументи** | **Опис** | **Відповідь** |
| SendData | Дві матриці | Надсилає дві матриці на сервер для подальшого обчислення | – |
| StartComputing | – | Сигналізує сервер про те, що всі дані передано та він має розпочати обчислення | StartedProcessing (Обробка даних розпочата) |
| GetData | – | Запитує сервер на отримання результату обчислень | Processing (Обробка даних ще триває)  Processed (Обробка даних ще триває) |
| EndConnection | – | Сигнал про завершення з'єднання клієнта з сервером | – |

1. **UML-діаграма викликів взаємодії серверу та клієнту, починаючи від запуску клієнту, до завершення роботи:**



**Висновок**

В ході реалізації завдання Розроблений код демонструє функціональний клієнт-серверний додаток для виконання матричних обчислень на сервері. Він ефективно використовує TCP для надійної передачі даних та паралельні обчислення на сервері за допомогою пералелелізму даних з використанням різних потоків для обчислень окремих сегментів даних.

TCP гарантує впорядковану доставку даних та перевірку помилок, що критично важливо для роботи з матрицями. Сервер може обслуговувати декілька клієнтів одночасно, створюючи нові потоки для кожного з них.

Потенційними проблеми коду є обмеженість даних, які має передавати користувач на сервер, їх обов’язкову чітку розмірність та необхідність синхронізації розмірності із зазначеною на сервері.

Виконуючи лабораторну роботу виникали проблеми з корректною передачою даних через сокет.

**Додатки**

Код програми:

* client.cpp

#include <iostream>  
#include <winsock2.h>  
#include "utils.h"  
#include "protocol.h"  
  
#pragma comment(lib, "ws2\_32.lib")  
  
const int PORT = 8000;  
const char\* SERVER\_IP = "127.0.0.1";  
const int MATRIX\_SIZE = 3;  
  
void sendData(SOCKET socket\_fd) {  
 std::vector<int> matrix1(MATRIX\_SIZE \* MATRIX\_SIZE);  
 std::vector<int> matrix2(MATRIX\_SIZE \* MATRIX\_SIZE);  
 Utility::fillRandomMatrix(matrix1, MATRIX\_SIZE, 10);  
 Utility::fillRandomMatrix(matrix2, MATRIX\_SIZE, 10);  
  
 Request request = Request::SendData;  
 send(socket\_fd, reinterpret\_cast<const char\*>(&request), sizeof(Request), 0);  
  
 std::cout << "Sending matrix 1 to the server..." << std::endl;  
 send(socket\_fd, reinterpret\_cast<const char\*>(matrix1.data()), sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, 0);  
 std::cout << "Matrix 1 sent:" << std::endl;  
 Utility::printMatrix(matrix1, MATRIX\_SIZE);  
  
 std::cout << "Sending matrix 2 to the server..." << std::endl;  
 send(socket\_fd, reinterpret\_cast<const char\*>(matrix2.data()), sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, 0);  
 std::cout << "Matrix 2 sent:" << std::endl;  
 Utility::printMatrix(matrix2, MATRIX\_SIZE);  
}  
  
void startCloudComputing(SOCKET socket\_fd) {  
 Request request = Request::StartComputing;  
 send(socket\_fd, reinterpret\_cast<const char\*>(&request), sizeof(Request), 0);  
  
 Response response;  
 recv(socket\_fd, reinterpret\_cast<char\*>(&response), sizeof(Response), 0);  
  
 if (response == Response::Processing) {  
 std::cout << "Server has started computing data." << std::endl;  
 }  
}  
  
void getData(SOCKET socket\_fd) {  
 Request request = Request::GetData;  
 send(socket\_fd, reinterpret\_cast<const char\*>(&request), sizeof(Request), 0);  
 Response response;  
  
 while (true) {  
 recv(socket\_fd, reinterpret\_cast<char\*>(&response), sizeof(Response), 0);  
  
 if (response == Response::Processing) {  
 std::cout << "Server has started computation." << std::endl;  
 }  
 else if (response == Response::DataProcessed) {  
 std::vector<int> result(MATRIX\_SIZE \* MATRIX\_SIZE);  
 std::cout << "Receiving result from the server..." << std::endl;  
 recv(socket\_fd, reinterpret\_cast<char \*>(result.data()), sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, 0);  
 std::cout << "Result received:" << std::endl;  
 Utility::printMatrix(result, MATRIX\_SIZE);  
 break;  
 } else {  
 std::cout << "Data not yet processed. Waiting for server..." << std::endl;  
 Sleep(1000);  
 }  
 }  
}  
  
void endConnection(SOCKET socket\_fd) {  
 Request request = Request::EndConnection;  
 send(socket\_fd, reinterpret\_cast<const char\*>(&request), sizeof(Request), 0);  
 std::cout << "Connection ended by client." << std::endl;  
 closesocket(socket\_fd);  
}  
  
int main() {  
 WSADATA wsaData;  
 SOCKET socket\_fd;  
 struct sockaddr\_in server\_address{};  
  
 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {  
 std::cout << "Failed to initialize socket" << std::endl;  
 return 1;  
 }  
  
 socket\_fd = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  
 if (socket\_fd == INVALID\_SOCKET) {  
 std::cout << "Failed to create socket" << std::endl;  
 WSACleanup();  
 return 1;  
 }  
  
 server\_address.sin\_family = AF\_INET;  
 server\_address.sin\_port = htons(PORT);  
 server\_address.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);  
 if (server\_address.sin\_addr.s\_addr == INADDR\_NONE) {  
 std::cout << "Invalid IP address" << std::endl;  
 closesocket(socket\_fd);  
 WSACleanup();  
 return 1;  
 }  
  
 if (connect(socket\_fd, (struct sockaddr\*)&server\_address, sizeof(server\_address)) == SOCKET\_ERROR) {  
 std::cout << "Failed to connect to server" << std::endl;  
 closesocket(socket\_fd);  
 WSACleanup();  
 return 1;  
 }  
  
 std::cout << "Connected to the server!" << std::endl;  
  
 sendData(socket\_fd);  
 startCloudComputing(socket\_fd);  
 getData(socket\_fd);  
 endConnection(socket\_fd);  
  
 closesocket(socket\_fd);  
 WSACleanup();  
  
 return 0;  
}

* server.cpp

#include <iostream>  
#include <winsock2.h>  
#include <thread>  
#include "utils.h"  
#include "protocol.h"  
  
#pragma comment(lib, "ws2\_32.lib")  
  
const int PORT = 8000;  
const int MATRIX\_SIZE = 3;  
const int NUM\_THREADS = 3;  
  
void closeConnection (SOCKET client\_socket) {  
 std::cout << "Connection ended by server." << std::endl << std::endl;  
 closesocket(client\_socket);  
}  
  
void checkReceivedBytes(SOCKET client\_socket, int bytesReceived) {  
 if (bytesReceived != sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE) {  
 closeConnection(client\_socket);  
 return;  
 }  
}  
  
void receiveData(SOCKET client\_socket, std::vector<int>& matrix1, std::vector<int>& matrix2) {  
 std::cout << "Receiving matrix 1 from client..." << std::endl;  
 checkReceivedBytes(client\_socket, recv(client\_socket, reinterpret\_cast<char\*>(matrix1.data()), sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, 0));  
 std::cout << "Matrix 1 received." << std::endl;  
 Utility::printMatrix(matrix1, MATRIX\_SIZE);  
  
 std::cout << "Receiving matrix 2 from client..." << std::endl;  
 checkReceivedBytes(client\_socket, recv(client\_socket, reinterpret\_cast<char\*>(matrix2.data()), sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, 0));  
 std::cout << "Matrix 2 received." << std::endl;  
 Utility::printMatrix(matrix2, MATRIX\_SIZE);  
}  
  
void computeData(SOCKET client\_socket, std::vector<int>& result,  
 const std::vector<int>& matrix1, const std::vector<int>& matrix2) {  
 std::vector<std::thread> threads;  
  
 Response response = Response::Processing;  
 send(client\_socket, reinterpret\_cast<const char\*>(&response), sizeof(Response), 0);  
  
 for (int i = 0; i < NUM\_THREADS; ++i) {  
 threads.emplace\_back(Utility::compute, i \* MATRIX\_SIZE / NUM\_THREADS, (i + 1) \* MATRIX\_SIZE / NUM\_THREADS,  
 MATRIX\_SIZE, std::ref(result), std::cref(matrix1), std::cref(matrix2));  
 }  
 for (auto& thread : threads) {  
 thread.join();  
 }  
  
 std::cout << "Data processed. Result:" << std::endl;  
 Utility::printMatrix(result, MATRIX\_SIZE);  
  
 response = Response::DataProcessed;  
 send(client\_socket, reinterpret\_cast<const char\*>(&response), sizeof(Response), 0);  
}  
  
void returnData (SOCKET client\_socket, std::vector<int>& result) {  
 std::cout << "Sending result to client..." << std::endl;  
 send(client\_socket, reinterpret\_cast<const char\*>(result.data()), sizeof(int) \* MATRIX\_SIZE \* MATRIX\_SIZE, 0);  
 std::cout << "Result sent." << std::endl;  
}  
  
void handleClient(SOCKET client\_socket) {  
 std::cout << "Connection with client established." << std::endl;  
 std::vector<int> matrix1(MATRIX\_SIZE \* MATRIX\_SIZE);  
 std::vector<int> matrix2(MATRIX\_SIZE \* MATRIX\_SIZE);  
 std::vector<int> result(MATRIX\_SIZE \* MATRIX\_SIZE);  
 while (true) {  
 Request request;  
 int bytesReceived = recv(client\_socket, reinterpret\_cast<char\*>(&request), sizeof(Request), 0);  
  
 if (bytesReceived == SOCKET\_ERROR || bytesReceived == 0) {  
 if (bytesReceived == SOCKET\_ERROR) {  
 std::cout << "Error receiving data from client. Error code: " << WSAGetLastError() << std::endl;  
 } else {  
 std::cout << "Connection with client terminated." << std::endl;  
 }  
 closeConnection(client\_socket);  
 return;  
 }  
  
 switch (request) {  
 case Request::SendData: {  
 receiveData(client\_socket, matrix1, matrix2);  
 break;  
 }  
 case Request::StartComputing:  
 computeData(client\_socket, result, matrix1, matrix2);  
 break;  
  
 case Request::GetData:  
 returnData(client\_socket, result);  
 break;  
  
 case Request::EndConnection:  
 closeConnection(client\_socket);  
 return;  
 }  
 }  
}  
  
int main() {  
 WSADATA wsaData;  
 SOCKET server\_socket, client\_socket;  
 struct sockaddr\_in server\_address{}, client\_address{};  
 int client\_address\_len = sizeof(client\_address);  
  
 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {  
 std::cout << "Failed to initialize socket" << std::endl;  
 return 1;  
 }  
  
 server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
 if (server\_socket == INVALID\_SOCKET) {  
 std::cout << "Failed to create socket" << std::endl;  
 WSACleanup();  
 return 1;  
 }  
  
 server\_address.sin\_family = AF\_INET;  
 server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 server\_address.sin\_port = htons(PORT);  
  
 if (bind(server\_socket, (struct sockaddr\*)&server\_address, sizeof(server\_address)) == SOCKET\_ERROR) {  
 std::cout << "Failed to bind socket" << std::endl;  
 closeConnection(server\_socket);  
 WSACleanup();  
 return 1;  
 }  
  
 if (listen(server\_socket, 5) == SOCKET\_ERROR) {  
 std::cout << "Failed to listen on socket" << std::endl;  
 closeConnection(server\_socket);  
 WSACleanup();  
 return 1;  
 }  
  
 try {  
 std::cout << "Listening for client to connect..." << std::endl << std::endl;  
 while (true) {  
 client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_address, &client\_address\_len);  
 if (client\_socket == INVALID\_SOCKET) {  
 std::cout << "Failed to accept client connection" << std::endl;  
 continue;  
 }  
  
 std::thread client\_thread(handleClient, client\_socket);  
 client\_thread.detach();  
 }  
 }  
 catch (...) {  
 std::cout << "Exception caught" << std::endl;  
 }  
  
 closeConnection(server\_socket);  
 WSACleanup();  
  
 return 0;  
}

* utils.h:

#ifndef LAB4\_UTILS\_H  
#define LAB4\_UTILS\_H  
  
#include <random>  
#include <iostream>  
#include <vector>  
  
class Utility {  
public:  
 static void printMatrix(const std::vector<int>& matrix, int matrix\_size)  
 {  
 for (int i = 0; i < matrix\_size; ++i) {  
 for (int j = 0; j < matrix\_size; ++j) {  
 std::cout << matrix[i \* matrix\_size + j] << " ";  
 }  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
 }  
  
 static void fillRandomMatrix(std::vector<int>& matrix, int matrix\_size, int max\_value)  
 {  
 std::random\_device rd;  
 std::mt19937 gen(rd());  
 std::uniform\_int\_distribution<int> distribution(0, max\_value);  
  
 for (int i = 0; i < matrix\_size \* matrix\_size; ++i) {  
 matrix[i] = distribution(gen);  
 }  
 }  
  
 static void compute(const std::size\_t start\_row, const std::size\_t end\_row, const int matrix\_size,  
 std::vector<int>& result,  
 const std::vector<int>& matrix1, const std::vector<int>& matrix2)  
 {  
 for (std::size\_t i = start\_row; i < end\_row; ++i) {  
 for (std::size\_t j = 0; j < matrix\_size; ++j) {  
 result[i \* matrix\_size + j] = matrix1[i \* matrix\_size + j] - matrix2[i \* matrix\_size + j];  
 }  
 }  
 }  
};  
  
#endif //LAB4\_UTILS\_H

* protocol.h:

#ifndef PROTOCOL\_H  
#define PROTOCOL\_H  
  
#include <vector>  
  
enum class Request {  
 SendData,  
 StartComputing,  
 GetData,  
 EndConnection  
};  
  
enum class Response {  
 Processing,  
 DataProcessed  
};  
  
#endif // PROTOCOL\_H