**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине: «Программирование сетевых приложений»

на тему: **«**Стек протоколов *TCP/IP*. Передача данных по сети

средствами стека протоколов *TCP/IP*»

Выполнил: студент гр. ИТИ-41

Ковшаров Г. Ю.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2025

**Цель работы:** освоение методологии создания приложений с использованием протоколов *TCP/IP*.

**Задание:**

1. Разработать программное обеспечение, реализующее передачу данных между компьютерами на уровне стека протоколов TCP/IP средствами ОС Windows, OC Linux, .Net.

2. Используя возможности стека протоколов TCP/IP организовать распределённую обработку информации не менее чем на 3 компьютерах для решения конкретной прикладной задачи (Табл. 1)

3. Решение задачи осуществить в ОС Windows, ОС Linux и dot.Net. Для претендующих на оценки 9, 10 обеспечить кроссплатформенное взаимодействие.

4. Сравнить время нахождения решения на нескольких компьютерах с временем решения задачи на одном компьютере.

5. Сравнить время нахождения решений в разных ОС и платформах.

Таблица 1 – Вариант задания

|  |  |
| --- | --- |
| Вариант | Задание |
| 13 | Найти результат возведения квадратной матрицы размерности N в степень M |

**Ход работы**

Разработаем и запустим сервера для выполнения задания по варианту. Каждый сервер отвечает за свою часть работы, таким образом задача распределяется между серверами. Запущенные сервера представлены на рисунке 1.

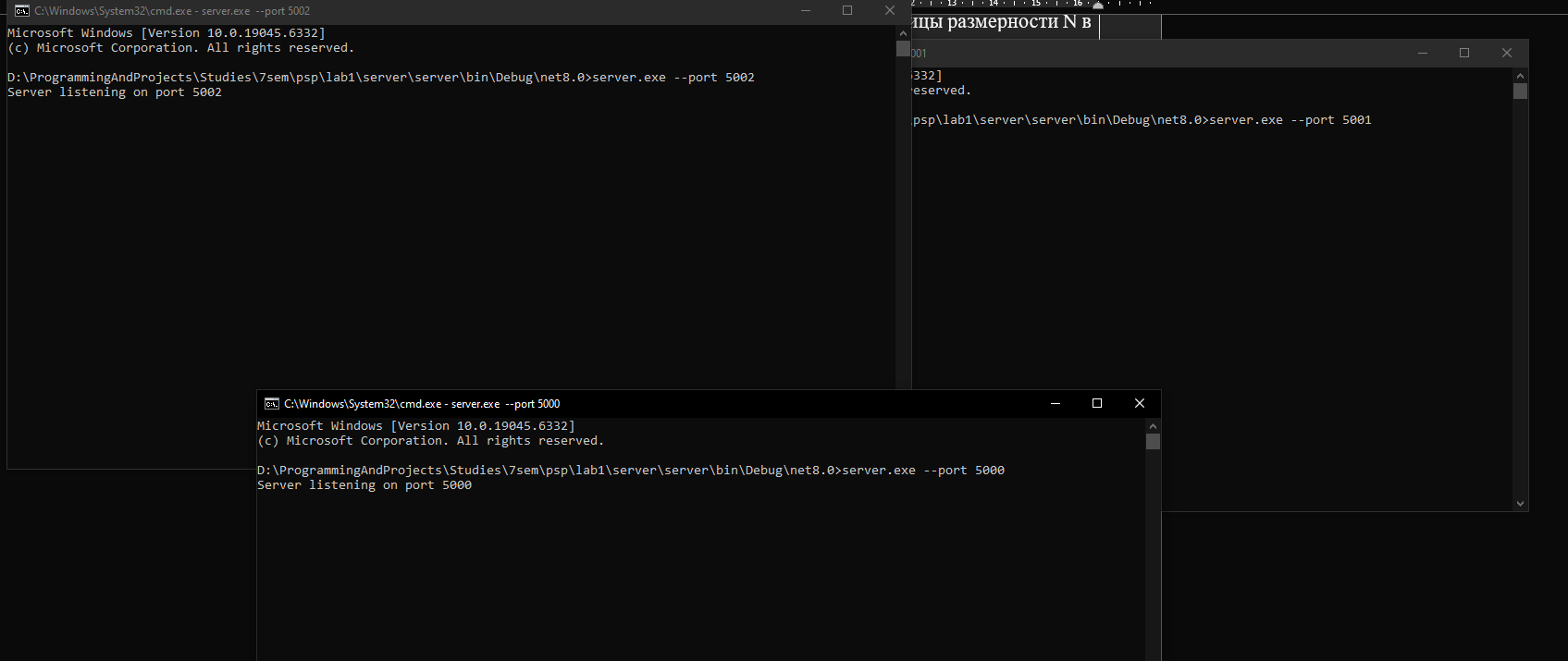


Рисунок 1 – Запущенные сервера

Разработаем и запустим клиент. Он подключается ко всем серверам и отправляет каждому часть матрицы для перемножения с оригинальной матрицей. На рисунке 2 представлен клиент, отправленные им данные, а также ответ полученный в итоге.

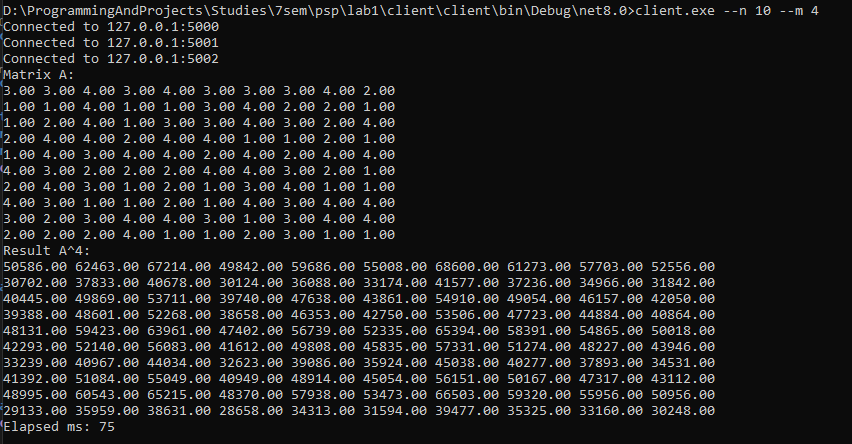


Рисунок 2 – Отправленные клиентом данные

На рисунке 3 представлена блок-схема приложения.

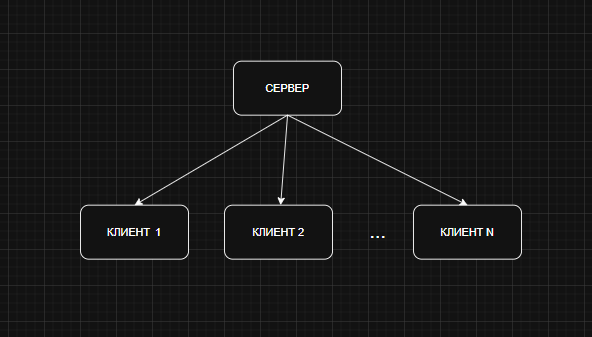


Рисунок 3 – Блок-схема приложения

Код приложения представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы освоили методологии создания приложений с использованием протоколов *TCP/IP*.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг приложения**

**Client**

**Program.cs**

using client.dto;

using System.Diagnostics;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Text.Json;

namespace Client;

internal class Program

{

static async Task Main(string[] args)

{

string serversArg = "127.0.0.1:5000,127.0.0.1:5001,127.0.0.1:5002";

int n = 4;

int power = 3;

for (int i = 0; i < args.Length; i++)

{

if (args[i] == "--servers" && i + 1 < args.Length) serversArg = args[++i];

if (args[i] == "--n" && i + 1 < args.Length) int.TryParse(args[++i], out n);

if (args[i] == "--m" && i + 1 < args.Length) int.TryParse(args[++i], out power);

}

var serverEndpoints = serversArg.Split(',', StringSplitOptions.RemoveEmptyEntries)

.Select(s =>

{

var parts = s.Split(':');

return new IPEndPoint(IPAddress.Parse(parts[0]), int.Parse(parts[1]));

}).ToArray();

var sockets = new List<Socket>();

foreach (var ep in serverEndpoints)

{

var sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

await sock.ConnectAsync(ep);

sockets.Add(sock);

Console.WriteLine($"Connected to {ep}");

}

var random = new Random();

float[][] A = new float[n][];

for (int i = 0; i < n; i++)

{

A[i] = new float[n];

for (int j = 0; j < n; j++) A[i][j] = (float)random.Next(1, 5);

}

Console.WriteLine("Matrix A:");

PrintMatrix(A);

Stopwatch sw = Stopwatch.StartNew();

float[][] result = await DistributedPower(A, power, sockets);

sw.Stop();

Console.WriteLine($"Result A^{power}:");

PrintMatrix(result);

Console.WriteLine($"Elapsed ms: {sw.ElapsedMilliseconds}");

foreach (var s in sockets) s.Close();

}

static async Task<float[][]> DistributedPower(float[][] A, int power, List<Socket> sockets)

{

int n = A.Length;

float[][] cur = CloneMatrix(A);

for (int t = 1; t <= power - 1; t++)

{

cur = await DistributedMultiply(cur, A, sockets);

}

return cur;

}

static async Task<float[][]> DistributedMultiply(float[][] curr, float[][] origMatrix, List<Socket> sockets)

{

int n = curr.Length;

int k = sockets.Count;

int baseRows = n / k;

int remain = n % k;

var tasks = new List<Task<ServerResponse>>();

int rowCursor = 0;

for (int i = 0; i < k; i++)

{

int rowsForThis = baseRows + (i < remain ? 1 : 0);

if (rowsForThis == 0) { tasks.Add(Task.FromResult(new ServerResponse { RowStart = rowCursor, Rows = new float[0][] })); continue; }

var aChunk = new float[rowsForThis][];

for (int r = 0; r < rowsForThis; r++) aChunk[r] = curr[rowCursor + r];

var req = new ClientRequest { Chunk = aChunk, OrigMatrix = origMatrix, RowStart = rowCursor };

tasks.Add(SendRequestAsync(sockets[i], req));

rowCursor += rowsForThis;

}

var responses = await Task.WhenAll(tasks);

var result = new float[n][];

foreach (var resp in responses)

{

if (resp.Rows == null) continue;

int rs = resp.RowStart;

for (int i = 0; i < resp.Rows.Length; i++)

{

result[rs + i] = resp.Rows[i];

}

}

return result;

}

static async Task<ServerResponse> SendRequestAsync(Socket socket, ClientRequest req)

{

var json = JsonSerializer.Serialize(req);

var bytes = Encoding.UTF8.GetBytes(json);

await socket.SendAsync(bytes, SocketFlags.None);

var buffer = new byte[10 \* 1024 \* 1024];

int rec = await socket.ReceiveAsync(buffer, SocketFlags.None);

var respJson = Encoding.UTF8.GetString(buffer, 0, rec);

var resp = JsonSerializer.Deserialize<ServerResponse>(respJson);

return resp;

}

static float[][] CloneMatrix(float[][] m) => m.Select(r => r.ToArray()).ToArray();

static void PrintMatrix(float[][] m)

{

for (int i = 0; i < m.Length; i++)

{

Console.WriteLine(string.Join(' ', m[i].Select(x => x.ToString("F2"))));

}

}

}

**Server**

**Program.cs**

using System.Net.Sockets;

using System.Net;

using System.Text;

using System.Text.Json;

using server.dto;

namespace Server

{

internal class Program

{

static async Task Main(string[] args)

{

int port = 5000;

if (args.Length >= 2 && args[0] == "--port") int.TryParse(args[1], out port);

var ipPoint = new IPEndPoint(IPAddress.Any, port);

using var listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

listener.Bind(ipPoint);

listener.Listen(100);

Console.WriteLine($"Server listening on port {port}");

while (true)

{

var client = await listener.AcceptAsync();

\_ = Task.Run(async () =>

{

Console.WriteLine($"Client connected: {client.RemoteEndPoint}");

var buffer = new byte[10 \* 1024 \* 1024];

try

{

while (true)

{

int received = await client.ReceiveAsync(buffer, SocketFlags.None);

if (received == 0) break;

var json = Encoding.UTF8.GetString(buffer, 0, received);

var req = JsonSerializer.Deserialize<MultiplyRequest>(json);

var resultRows = MultiplyRows(req);

var respBytes = SerializeRows(resultRows);

await client.SendAsync(respBytes, SocketFlags.None);

}

}

catch (Exception ex)

{

Console.WriteLine("Exception: " + ex);

}

finally

{

client.Close();

}

});

}

}

static MultiplyResponse MultiplyRows(MultiplyRequest req)

{

int rows = req.Chunk.Length;

int n = req.OrigMatrix.Length;

float[][] outRows = new float[rows][];

for (int i = 0; i < rows; i++)

{

outRows[i] = new float[n];

for (int j = 0; j < n; j++)

{

double s = 0;

for (int k = 0; k < n; k++)

{

s += (double)req.Chunk[i][k] \* req.OrigMatrix[k][j];

}

outRows[i][j] = (float)s;

}

}

return new MultiplyResponse { RowStart = req.RowStart, Rows = outRows };

}

static byte[] SerializeRows(MultiplyResponse resp)

{

return Encoding.UTF8.GetBytes(JsonSerializer.Serialize(resp));

}

}

}