МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №0

по дисциплине: **«**Программирование сетевых приложений**»**

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

средствами стека протоколов *TCP*/*IP*»

Выполнил: студент гр. ИТП-41

Дорошко Д. А.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2022

**Цель:** изучить методы решения задачи об оптимальных назначениях.

**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 распределѐнную обработку информации не менее чем на 3 компьютерах для решения конкретной прикладной задачи (Табл. 0.1).

3. Решение задачи осуществить в ОС *Windows*, ОС *Linux* и *dot.Net*. Для претендующих на оценки 9­10 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. Задание согласно варианту, представлено на рисунке 1.



Рисунок 1 – Задание согласно варианту

**Ход работы:**

Результат вычисления в ОС *Windows* на пяти серверах представлен на рисунке 1.

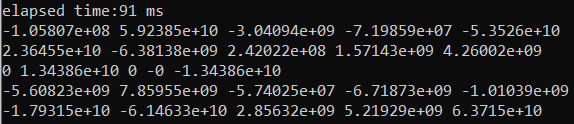


Рисунок 1 – Результат вычисления в ОС *Windows* на пяти серверах

Результат вычисления в ОС Windows без использования серверов представлен на рисунке 2.

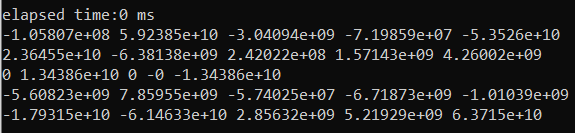


Рисунок 2 – Результат вычисления на ОС *Windows* без использования серверов

Запуск решения в ОС *Linux* представлен на рисунке 3.

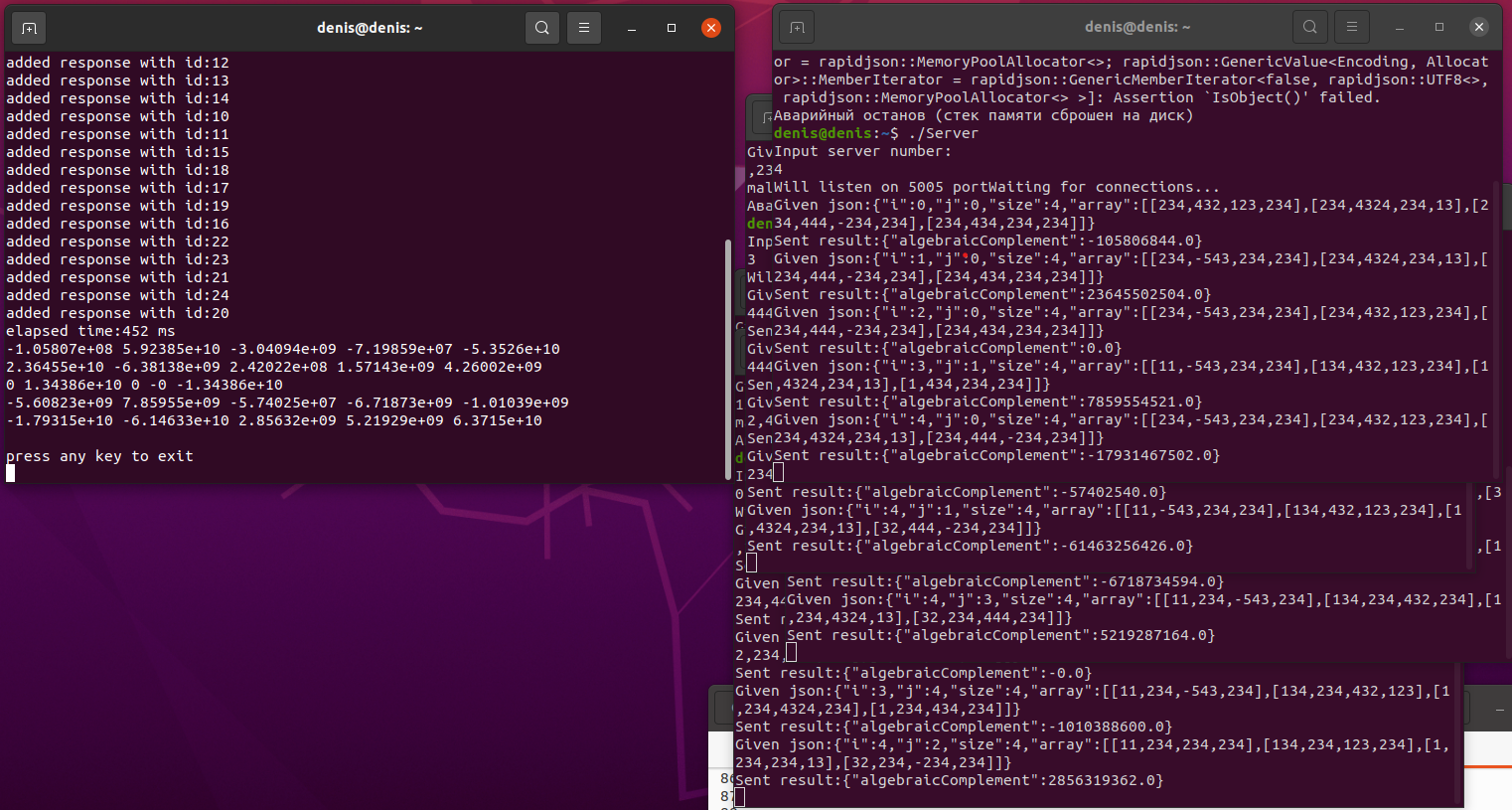


Рисунок 3 – Результат вычисления в ОС *Linux*

Результат вычисления матрицы большой размерности без использования серверов представлен на рисунке 4.

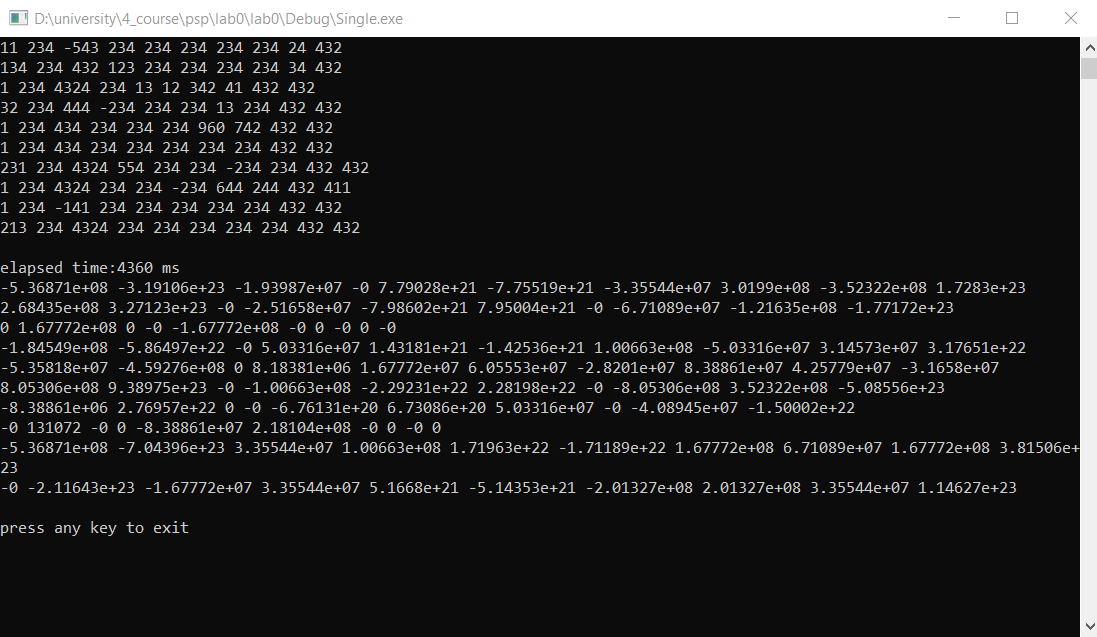


Рисунок 4 – Результат вычисления матрицы большой размерности без использования серверов

Результат вычисления матрицы большой размерности с использованием пяти серверов представлен на рисунке 5.

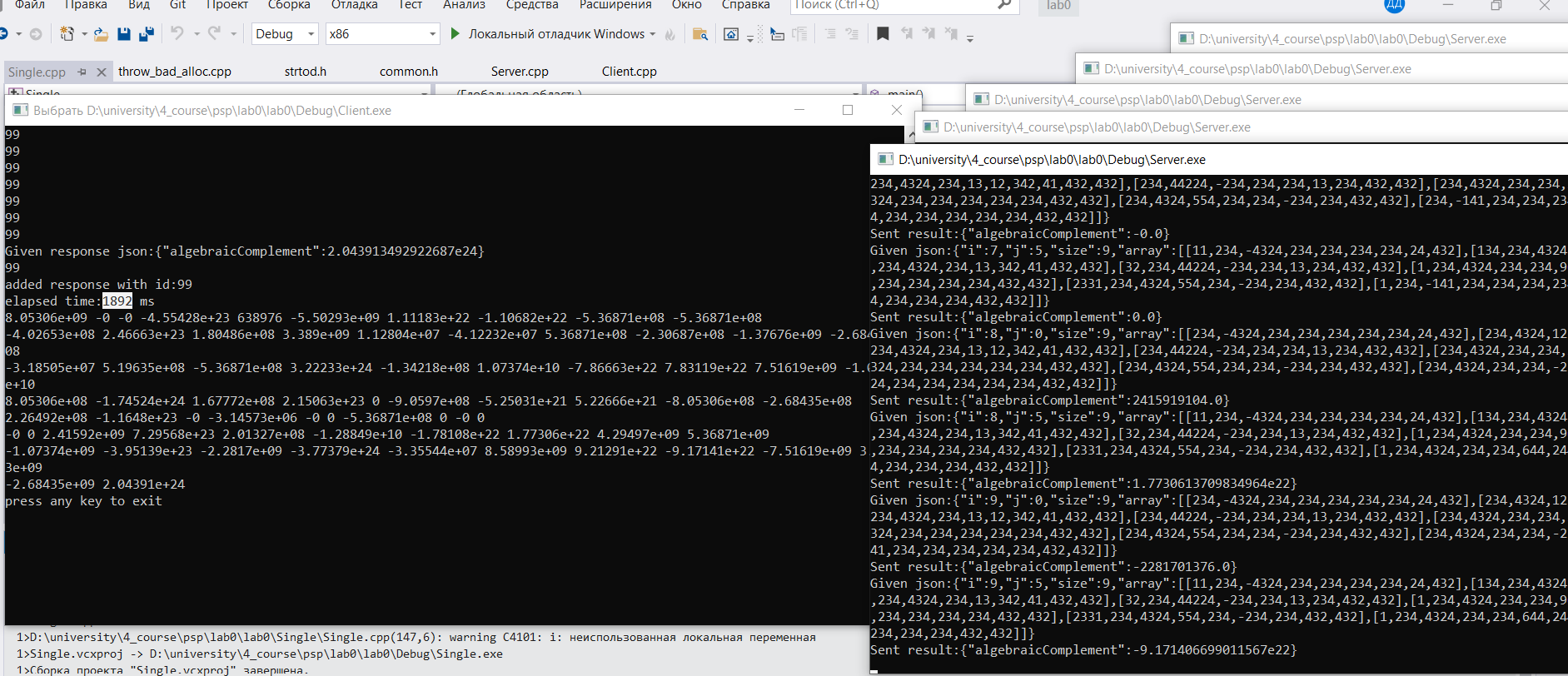


Рисунок 5 – Результат вычисления матрицы большой размерности с использованием пяти серверов

Исходя из полученных результатов можно сделать вывод что при использовании матриц небольшой размерности вычисление результата без использования серверов займет меньшее время чем при использовании, однако при повышении размерности матрицы решение с использованием серверов производится значительно быстрее.

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы получены навыки программирования сокетов на операционных системах *Windows* и *Linux*, изучены и протестированы преимущества и недостатки выполнения распределенных вычислений.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

// Single.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <iostream>

#include <cstdio>

#include <cstring>

#include <map>

#include <chrono>

using namespace std;

double algebraicComplement(int\*\* arr, int i, int j, int size);

double determinant(int\*\* arr, int size);

void inputMatrix(int\*\* arr, int size);

void outputMatrix(int\*\* arr, int size);

void outputResult(map<int, double>, int size);

int main()

{

map<int, double> results\_map;

int size;

/\*std::cout << "Input matrix size" << endl;

std::cin >> size;

int\*\* arr = new int\* [size];

int\*\* algebraicComplements = new int\* [size];

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

arr[i] = new int[size];

inputMatrix(arr, size);\*/

size = 5;

int\*\* arr = new int\* [5];

arr[0] = new int[5]{ 11, 234, -543, 234, 234 };

arr[1] = new int[5]{ 134, 234, 432, 123, 234 };

arr[2] = new int[5]{ 1, 234, 234, 234, 13 };

arr[3] = new int[5]{ 32, 234, 234, -234, 234 };

arr[4] = new int[5]{ 1, 234, 434, 234, 34 };

//size = 10;

//int\*\* arr = new int\* [10];

//arr[0] = new int[10]{ 11, 234, -543, 234, 234, 234, 234, 234, 24, 432 };

//arr[1] = new int[10]{ 134, 234, 432, 123, 234, 234, 234, 234, 34, 432 };

//arr[2] = new int[10]{ 1, 234, 4324, 234, 13, 12, 342, 41, 432, 432 };

//arr[3] = new int[10]{ 32, 234, 444, -234, 234, 234, 13, 234, 432, 432 };

//arr[4] = new int[10]{ 1, 234, 434, 234, 234, 234, 960, 742, 432, 432 };

//arr[5] = new int[10]{ 1, 234, 434, 234, 234, 234, 234, 234, 432, 432 };

//arr[6] = new int[10]{ 231, 234, 4324, 554, 234, 234, -234, 234, 432, 432 };

//arr[7] = new int[10]{ 1, 234, 4324, 234, 234, -234, 644, 244, 432, 411 };

//arr[8] = new int[10]{ 1, 234, -141, 234, 234, 234, 234, 234, 432, 432 };

//arr[9] = new int[10]{ 213, 234, 4324, 234, 234, 234, 234, 234, 432, 432 };

outputMatrix(arr, size);

auto start\_time = std::chrono::steady\_clock::now();

int id = 0;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

results\_map[id] = algebraicComplement(arr, i, j, size);

id = id + 1;

}

}

auto end\_time = std::chrono::steady\_clock::now();

auto elapsed\_ns = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time);

std::cout << "elapsed time:" << elapsed\_ns.count() << " ms" << endl;

outputResult(results\_map, size);

cout << "press any key to exit" << endl;

int nothing;

cin >> nothing;

return 0;

}

double algebraicComplement(int\*\* arr, int aci, int acj, int size)

{

int\*\* temp = new int\* [size - 1];

for (int i = 0; i < size - 1; i++)

temp[i] = new int[size - 1];

int s = 0, s1 = 0;

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

{

if (i != aci)

{

s1 = 0;

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

if (j != acj)

{

temp[s][s1] = arr[i][j];

s1++;

}

s++;

}

}

return pow(-1., aci + acj) \* determinant(temp, size - 1);

}

double determinant(int\*\* arr, int size)

{

int i, j;

double det = 0; //переменная определителя

int\*\* matr; //указатель

if (size == 1) // 1-е условие , размер 1

{

det = arr[0][0];

}

else if (size == 2) // 2-е условие , размер 2

{

det = arr[0][0] \* arr[1][1] - arr[0][1] \* arr[1][0]; //

}

else

{

matr = new int\* [size - 1]; //создание динамического массива

for (i = 0;i < size;++i)

{

for (j = 0;j < size - 1;++j)

{

if (j < i)

{

matr[j] = arr[j];

}

else

matr[j] = arr[j + 1];

}

det += pow(-1., (i + j)) \* determinant(matr, size - 1) \* arr[i][size - 1]; //подсчеты

}

delete[] matr; //удаляем массив

}

return det; //возвращаем значение определителя

}

void inputMatrix(int\*\* arr, int size)

{

int i, j;

for (i = 0; i < size; i++)

{

for (j = 0; j < size; j++)

{

std::cout << "input [" << i << "," << j << "]" << endl;

std::cin >> arr[i][j];

}

}

}

void outputMatrix(int\*\* arr, int size)

{

int i, j;

for (i = 0;i < size;i++)

{

for (j = 0;j < size;j++)

{

std::cout << arr[i][j] << " ";

}

std::cout << endl;

}

std::cout << endl;

}

void outputResult(map<int, double> results, int size)

{

int i;

int totalSize = size \* size;

int row = 0;

map <int, double> ::iterator it = results.begin();

for (int i = 0; it != results.end(); it++, i++)

{

std::cout << it->second << " ";

if (++row == size) {

std::cout << endl;

row = 0;

}

}

std::cout << endl;

}

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <cstring>

#include <winsock2.h>

#pragma comment(lib, "WS2\_32.lib")

#include "rapidjson/document.h"

#include "rapidjson/reader.h"

#include "rapidjson/writer.h"

#include "rapidjson/stringbuffer.h"

#include "rapidjson/memorybuffer.h"

#include <stack>

#include <thread>

#include "../Common/common.h"

using namespace std;

using namespace rapidjson;

constexpr auto BASE\_PORT = 5001;

constexpr auto SERVERADDR = "127.0.0.1";

constexpr auto HEADER\_SIZE = sizeof(header);

int clientsCount = 0;

int receive(SOCKET sock, char\* buffer, int chunk\_size);

DWORD WINAPI process(LPVOID client\_socket);

double algebraicComplement(int\*\* arr, int i, int j, int size);

double determinant(int\*\* arr, int size);

int main()

{

char buff[1024];

if (WSAStartup(0x0202, (WSADATA\*)&buff[0]))

{

printf("Error WSAStartup %d\n", WSAGetLastError());

return 1;

}

SOCKET serverSocket;

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("Error socket %d\n", WSAGetLastError());

WSACleanup();

return 1;

}

int serverNumber = 0;

cout << "Input server number: " << endl;

cin >> serverNumber;

int serverPort = BASE\_PORT + serverNumber;

cout << "Will listen on " << serverPort << " port";

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(serverPort);

serverAddr.sin\_addr.s\_addr = inet\_addr(SERVERADDR);

if (bind(serverSocket, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

cout << "Bind function failed with error: " << WSAGetLastError() << endl;

return -1;

}

if (listen(serverSocket, 0) == SOCKET\_ERROR) {

cout << "Listen function failed with error:" << WSAGetLastError() << endl;

return -1;

}

printf("Waiting for connections...\n");

SOCKET clientSocket;

sockaddr\_in client\_addr;

int client\_addr\_size = sizeof(client\_addr);

while ((clientSocket = accept(serverSocket, (sockaddr\*)&client\_addr, &client\_addr\_size))) {

clientsCount++;

cout << "Connected client" << endl;

DWORD tid;

HANDLE t1 = CreateThread(NULL, 0, process, &clientSocket, 0, &tid); //Создание потока для получения данных

if (t1 == NULL) {

cout << "Thread Creation Error: " << WSAGetLastError() << endl;

}

}

return 0;

}

DWORD WINAPI process(LPVOID lpParam) {

SOCKET client\_socket;

client\_socket = ((SOCKET\*)lpParam)[0];

char headerBuffer[HEADER\_SIZE] = { 0 };

//char buffer[1024] = { 0 };

header requestHeader;

int bytes\_received = 0;

while (true) {

bytes\_received = receive(client\_socket, headerBuffer, HEADER\_SIZE);

if (bytes\_received == 0) {

continue;

}

requestHeader.deserialize(headerBuffer);

char\* buffer = new char[requestHeader.size];

buffer[requestHeader.size] = '\0';

receive(client\_socket, buffer, requestHeader.size);

cout << "Given json:" << buffer << endl;

Document d;

d.Parse(buffer);

int size = d["size"].GetInt();

int\*\* arr = new int\* [size];

const Value& arrayValue = d["array"];

for (int i = 0; i < size; i++) {

arr[i] = new int[size];

for (int j = 0; j < size; j++) {

arr[i][j] = arrayValue[i][j].GetInt();

}

}

StringBuffer s;

Writer<StringBuffer> writer(s);

writer.StartObject();

writer.Key("algebraicComplement");

writer.Double(algebraicComplement(arr, d["i"].GetInt(), d["j"].GetInt(), size));

writer.EndObject();

header responseHeader = header();

responseHeader.size = s.GetSize();

char\* header\_data = new char[HEADER\_SIZE];

responseHeader.serialize(header\_data);

send(client\_socket, header\_data, HEADER\_SIZE, 0);

send(client\_socket, s.GetString(), s.GetSize(), 0);

cout << "Sent result:" << s.GetString() << endl;

}

printf("Disconnected\n");

clientsCount--;

return 0;

}

int receive(SOCKET sock, char\* buffer, int chunk\_size)

{

int offset = 0;

while (chunk\_size > 0)

{

int n = recv(sock, buffer + offset, chunk\_size, 0);

offset += n;

chunk\_size -= n;

}

return offset;

}

double algebraicComplement(int\*\* arr, int i, int j, int size)

{

return pow(-1., i + j) \* determinant(arr, size);

}

double determinant(int\*\* arr, int size)

{

int i, j;

double det = 0; //переменная определителя

int\*\* matr; //указатель

if (size == 1) // 1-е условие , размер 1

{

det = arr[0][0];

}

else if (size == 2) // 2-е условие , размер 2

{

det = arr[0][0] \* arr[1][1] - arr[0][1] \* arr[1][0]; //

}

else

{

matr = new int\* [size - 1]; //создание динамического массива

for (i = 0;i < size;++i)

{

for (j = 0;j < size - 1;++j)

{

if (j < i)

{

matr[j] = arr[j];

}

else

matr[j] = arr[j + 1];

}

det += pow(-1., (i + j)) \* determinant(matr, size - 1) \* arr[i][size - 1]; //подсчеты

}

delete[] matr; //удаляем массив

}

return det; //возвращаем значение определителя

}

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <cstdio>

#include <cstring>

#include <winsock2.h>

#pragma comment(lib, "WS2\_32.lib")

#include "rapidjson/document.h"

#include "rapidjson/reader.h"

#include "rapidjson/writer.h"

#include "rapidjson/stringbuffer.h"

#include "rapidjson/memorybuffer.h"

#include <stack>

#include <thread>

#include <boost/thread.hpp>

#include <map>

#include <chrono>

#include "../Common/common.h"

using namespace std;

using namespace rapidjson;

constexpr auto BASE\_PORT = 5001;

constexpr auto SERVERADDR = "127.0.0.1";

constexpr auto HEADER\_SIZE = sizeof(header);

constexpr auto WORKERS\_COUNT = 5;

struct poolItem {

int port;

bool isOpen;

SOCKET socket;

};

template <typename T> class ThreadSafeStack {

public:

ThreadSafeStack() {

m\_stack = stack<poolItem>();

}

void push(const T& item) {

boost::mutex::scoped\_lock lock(m\_mutex);

m\_stack.push(item);

}

T\* pop() {

boost::mutex::scoped\_lock lock(m\_mutex);

if (!m\_stack.empty()) {

T item = m\_stack.top();

m\_stack.pop();

return &item;

}

return nullptr;

}

private:

mutable boost::mutex m\_mutex;

std::stack<T> m\_stack;

};

ThreadSafeStack<poolItem> pool;

map<int, double> results\_map;

void inputMatrix(int\*\* arr, int size);

void outputMatrix(int\*\* arr, int size);

void outputResult(map<int, double>, int size);

void process(int id, poolItem worker, int\*\* arr, int i, int j, int size);

int receive(SOCKET sock, char\* buffer, int chunk\_size);

int main() {

for (int i = 0; i < WORKERS\_COUNT; i++) {

poolItem item = poolItem();

item.isOpen = false;

item.port = BASE\_PORT + i;

pool.push(item);

}

int size;

/\*std::cout << "Input matrix size" << endl;

std::cin >> size;

int\*\* arr = new int\* [size];

int\*\* algebraicComplements = new int\* [size];

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

arr[i] = new int[size];

inputMatrix(arr, size);\*/

size = 5;

int\*\* arr = new int\* [5];

arr[0] = new int[5]{ 11, 234, -543, 234, 234 };

arr[1] = new int[5]{ 134, 234, 432, 123, 234 };

arr[2] = new int[5]{ 1, 234, 234, 234, 13 };

arr[3] = new int[5]{ 32, 234, 234, -234, 234 };

arr[4] = new int[5]{ 1, 234, 434, 234, 34 };

//size = 10;

//int\*\* arr = new int\* [10];

//arr[0] = new int[10] { 11, 234, -543, 234, 234, 234, 234, 234, 24, 432};

//arr[1] = new int[10] { 134, 234, 432, 123, 234, 234, 234, 234, 34, 432};

//arr[2] = new int[10] { 1, 234, 4324, 234, 13, 12, 342, 41, 432, 432};

//arr[3] = new int[10] { 32, 234, 444, -234, 234, 234, 13, 234, 432, 432};

//arr[4] = new int[10] { 1, 234, 434, 234, 234, 234, 960, 742, 432, 432};

//arr[5] = new int[10] { 1, 234, 434, 234, 234, 234, 234, 234, 432, 432};

//arr[6] = new int[10] { 231, 234, 4324, 554, 234, 234, -234, 234, 432, 432};

//arr[7] = new int[10] { 1, 234, 4324, 234, 234, -234, 644, 244, 432, 411};

//arr[8] = new int[10] { 1, 234, -141, 234, 234, 234, 234, 234, 432, 432};

//arr[9] = new int[10] { 213, 234, 4324, 234, 234, 234, 234, 234, 432, 432};

outputMatrix(arr, size);

WSADATA WSAData;

if (WSAStartup(MAKEWORD(2, 0), &WSAData)) {

printf("WSAStart error %d\n", WSAGetLastError());

return 1;

}

auto start\_time = std::chrono::steady\_clock::now();

int id = 0;

for (int i = 0;i < size;i++) {

for (int j = 0; j < size; j++) {

bool queued = false;

while (!queued) {

poolItem\* worker = pool.pop();

if (worker != nullptr) {

poolItem w = \*worker;

std::thread tA(process, id, w, arr, i, j, size);

tA.detach();

queued = true;

id = id + 1;

}

}

}

}

int totalResults = size \* size;

while (results\_map.size() != totalResults) {

cout << results\_map.size() << endl;

//Sleep(100);

}

auto end\_time = std::chrono::steady\_clock::now();

auto elapsed\_ns = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time);

std::cout << "elapsed time:" << elapsed\_ns.count() << " ms" << endl;

outputResult(results\_map, size);

for (int i = 0; i < WORKERS\_COUNT; i++) {

poolItem\* item = pool.pop();

if (item != nullptr) {

if ((\*item).isOpen) {

closesocket((\*item).socket);

WSACleanup();

}

}

}

cout << "press any key to exit" << endl;

int nothing;

cin >> nothing;

return 0;

}

void process(int id, poolItem worker, int\*\* arr, int i, int j, int size)

{

cout << "Connecting to server with port " << worker.port << endl;

if (!worker.isOpen) {

worker.socket = socket(AF\_INET, SOCK\_STREAM, 0);

worker.isOpen = true;

if (worker.socket < 0) {

cout << "Socket error " << WSAGetLastError() << endl;

return;

}

sockaddr\_in dest\_addr;

dest\_addr.sin\_family = AF\_INET;

dest\_addr.sin\_port = htons(worker.port);

dest\_addr.sin\_addr.s\_addr = inet\_addr(SERVERADDR);

if (connect(worker.socket, (struct sockaddr\*)&dest\_addr, sizeof(dest\_addr)) < 0)

{

cout << "Connection Failed" << endl;

return;

}

}

StringBuffer s;

Writer<StringBuffer> writer(s);

writer.StartObject();

writer.Key("i");

writer.Int(i);

writer.Key("j");

writer.Int(j);

writer.Key("size");

writer.Int(size - 1);

writer.Key("array");

writer.StartArray();

for (int ki = 0; ki < size; ki++)

{

if (ki != i)

{

writer.StartArray();

for (int kj = 0;kj < size;kj++) {

if (kj != j)

{

writer.Int(arr[ki][kj]);

}

}

writer.EndArray();

}

}

writer.EndArray();

writer.EndObject();

header h = header();

h.size = s.GetSize();

cout << s.GetString() << endl;

char\* header\_data = new char[HEADER\_SIZE];

h.serialize(header\_data);

send(worker.socket, header\_data, HEADER\_SIZE, 0);

if (send(worker.socket, s.GetString(), s.GetSize(), 0) == SOCKET\_ERROR) {

cout << "send failed with error: " << WSAGetLastError() << endl;

}

char headerBuffer[HEADER\_SIZE] = { 0 };

receive(worker.socket, headerBuffer, HEADER\_SIZE);

header responseHeader;

responseHeader.deserialize(headerBuffer);

char\* buffer = new char[responseHeader.size];

buffer[responseHeader.size] = '\0';

receive(worker.socket, buffer, responseHeader.size);

cout << "Given response json:" << buffer << endl;

Document d;

d.Parse(buffer);

pool.push(worker);

results\_map[id] = d["algebraicComplement"].GetDouble();

cout << "added response with id:" << id << endl;

}

void inputMatrix(int\*\* arr, int size)

{

int i, j;

for (i = 0; i < size; i++)

{

for (j = 0; j < size; j++)

{

std::cout << "input [" << i << "," << j << "]" << endl;

std::cin >> arr[i][j];

}

}

}

void outputMatrix(int\*\* arr, int size)

{

int i, j;

for (i = 0;i < size;i++)

{

for (j = 0;j < size;j++)

{

std::cout << arr[i][j] << " ";

}

std::cout << endl;

}

std::cout << endl;

}

void outputResult(map<int, double> results, int size)

{

int i;

int totalSize = size \* size;

int row = 0;

map <int, double> ::iterator it = results.begin();

for (int i = 0; it != results.end(); it++, i++)

{

std::cout << it->second << " ";

if (++row == size) {

std::cout << endl;

row = 0;

}

}

std::cout << endl;

}

int receive(SOCKET sock, char\* buffer, int chunk\_size)

{

int offset = 0;

while (chunk\_size > 0)

{

int n = recv(sock, buffer + offset, chunk\_size, 0);

offset += n;

chunk\_size -= n;

}

return offset;

}