Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра "Вычислительной техники"

**ОТЧЁТ**

По лабораторной №3

по дисциплине "Распределенные вычисления"

на тему " Разработка вычислительной системы облачного типа"

Вариант № 7

Выполнили студенты группы 20ВВП1:

Тумасов В.В

Ермолаев А.Д

Приняли:

Федюнин Р. Н.

Елфимов А.В

Пенза 2024

Разработка вычислительной системы облачного типа

**Цель работы:** изучение подхода к реализации облачных вычислений и

разработка с его использованием вычислительной системы облачного типа.

Лабораторное задание:

Разработать консольные клиентское и серверное приложения с

использованием языка С/С++ и библиотеки WinSock, взаимодействующие по

протоколу TCP/IP. Серверное приложение должно обеспечивать одновременное

подключение нескольких клиентов, прием входящих сообщений от клиентов,

осуществлять периодическую выдачу задания каждому клиенту и прием

результатов вычислений, а также выдачу сообщения клиентам о завершении

расчетов, формирование конечного результата вычислений и отображение его в

консоли. Клиентское приложение в свою очередь должно обеспечивать

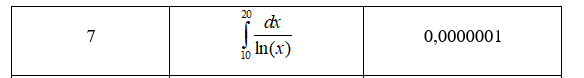
подключение к серверу (порт и IP вводятся в консоли), периодический прием от

сервера заданий, их расчет и возврат результатов вычислений серверу до тех

пор, пока сервер не сообщит о завершении вычислений. Параллелизм

реализуется как на уровне клиентских процессов, так и на уровне потоков в

клиентских приложениях, при этом число потоков должно быть равно числу

процессоров в системе.

Ход выполнения лабораторной работы:

1)Определение функции:

double Function(double x) {

return 1.0 / log(x);

}

Здесь определена подынтегральная функция ​, которая будет использоваться в вычислениях.

2)Определение функции интегрирования:

double CalculateIntegral(double start, double end, double step) {

double sum = 0.0;

for (double x = start; x < end; x += step) {

sum += Function(x) \* step;

}

return sum;

}

Функция CalculateIntegral вычисляет интеграл методом прямоугольников. Она принимает в качестве аргументов начальный и конечный пределы интегрирования (start и end) и шаг интегрирования (step). В цикле она проходит по диапазону от start до end с шагом step, вычисляя значение функции в каждой точке и умножая на шаг. Сумма этих значений дает приближенное значение определенного интеграла.

3)Ожидание активности на сокетах:

int activity = select(0, &readSet, nullptr, nullptr, nullptr);

if (activity == SOCKET\_ERROR) {

std::cerr << "Select failed\n";

break;

}

4)Ожидание подключения двух клиентов и отправка им интервалов интегрирования:

if (clients.size() == 2) {

double range = 10.0;

double step1 = ((20.0 - 10.0) / clients.size()) / 2;

double start = 7.5;

double end;

int num\_intervals = clients.size() \* 2;

int b1 = 0;

int y = 0;

for (size\_t i = 0; i < num\_intervals; ++i) { if (b1==2)

{

b1 = 100;

y++;

}

start += step1;

end = start + step1;

integrationParams.clear();

IntegrationParams params(start, end, step);

integrationParams.push\_back(params);

std::ostringstream paramsStream;

paramsStream << params.start << " " << params.end << " " << params.step;

send(clients[y], paramsStream.str().c\_str(), paramsStream.str().size() + 1, 0);

b1++;

startTime = std::chrono::system\_clock::now();

}

}

}

5)Обработка полученных интервалов интегрирования и отправка результата серверу:

std::cout << "Received first integral parameters from server: start=" << start1 << ", end=" << end1 << ", step=" << step1 << "\n";

recv(clientSocket, paramsBuffer1, sizeof(paramsBuffer1), 0);

std::istringstream paramsStream1(paramsBuffer1);

double start2, end2, step2;

paramsStream1 >> start2 >> end2 >> step2;

std::cout << "Received second integral parameters from server: start=" << start2 << ", end=" << end2 << ", step=" << step2 << "\n";

std::vector<std::thread> threads;

unsigned int numThreads = std::thread::hardware\_concurrency();

threads.reserve(numThreads);

threads.emplace\_back(CalculateAndSendResult, clientSocket, start1, end1, step1);

threads.emplace\_back(CalculateAndSendResult, clientSocket, start2, end2, step2);

6)Обработка результата и его вывод:

if (strncmp(buffer, "Result-", 7) == 0) {

double result = std::stod(buffer + 7);

std::cout << "Received integral result from client " << std::distance(clients.begin(), it) + 1 << ": " << result << "\n";

accumulatedResult += result;

endTime = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed\_seconds = endTime - startTime;

std::cout << "Integration process took : " << elapsed\_seconds.count() << " seconds\n";

std::cout << "Accumulated result: " <<

accumulatedResult << "\n";

}

}

Листинг программы(Сервер):

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

#include <vector>

#include <algorithm>

#include <cmath>

#include <sstream>

#include <thread>

#include <chrono>

#include <ctime>

#pragma comment(lib, "ws2\_32.lib")

double Function(double x) {

return 1.0 / log(x);

}

struct IntegrationParams {

double start;

double end;

double step;

IntegrationParams(double start, double end, double step)

: start(start), end(end), step(step) {}

};

int main() {

std::chrono::time\_point<std::chrono::system\_clock> startTime, endTime;

double accumulatedResult = 0.0;

WSADATA wsaData;

setlocale(LC\_ALL, "en\_us");

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "WSAStartup failed\n";

return 1;

}

int port;

std::cout << "Enter server port: ";

std::cin >> port;

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed\n";

WSACleanup();

return 1;

}

sockaddr\_in serverAddr{};

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(port);

if (bind(serverSocket, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Bind failed\n";

closesocket(serverSocket);

WSACleanup();

return 1;

}

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

std::cerr << "Listen failed\n";

closesocket(serverSocket);

WSACleanup();

return 1;

}

std::cout << "Server listening on port " << port << "\n";

std::vector<SOCKET> clients;

std::vector<IntegrationParams> integrationParams;

fd\_set readSet;

char buffer[1024];

int bytesRead;

while (true) {

FD\_ZERO(&readSet);

FD\_SET(serverSocket, &readSet);

for (const auto& client : clients) {

FD\_SET(client, &readSet);

}

int activity = select(0, &readSet, nullptr, nullptr, nullptr);

if (activity == SOCKET\_ERROR) {

std::cerr << "Select failed\n";

break;

}

if (FD\_ISSET(serverSocket, &readSet)) {

SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Accept failed\n";

break;

}

clients.push\_back(clientSocket);

char clientIP[INET\_ADDRSTRLEN];

sockaddr\_in clientAddr;

int clientAddrSize = sizeof(clientAddr);

getpeername(clientSocket, reinterpret\_cast<sockaddr\*>(&clientAddr), &clientAddrSize);

inet\_ntop(AF\_INET, &(clientAddr.sin\_addr), clientIP, INET\_ADDRSTRLEN);

std::cout << "Client " << clients.size() << " connected: " << clientIP << ":" << ntohs(clientAddr.sin\_port) << "\n";

double step = 0.0000001;

if (clients.size() == 2) {

double range = 10.0;

double step1 = ((20.0 - 10.0) / clients.size()) / 2;

double start = 7.5;

double end;

int num\_intervals = clients.size() \* 2;

int b1 = 0;

int y = 0;

for (size\_t i = 0; i < num\_intervals; ++i) {

if (b1==2)

{

b1 = 100;

y++;

}

start += step1;

end = start + step1;

integrationParams.clear();

IntegrationParams params(start, end, step);

integrationParams.push\_back(params);

std::ostringstream paramsStream;

paramsStream << params.start << " " << params.end << " " << params.step;

send(clients[y], paramsStream.str().c\_str(), paramsStream.str().size() + 1, 0);

b1++;

startTime = std::chrono::system\_clock::now();

}

}

}

for (auto it = clients.begin(); it != clients.end(); ) {

SOCKET clientSocket = \*it;

if (FD\_ISSET(clientSocket, &readSet)) {

bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);

if (bytesRead > 0) {

buffer[bytesRead] = '\0';

std::cout << "Received from client " << std::distance(clients.begin(), it) + 1 << ": " << buffer << "\n";

if (strncmp(buffer, "Result-", 7) == 0) {

double result = std::stod(buffer + 7);

std::cout << "Received integral result from client " << std::distance(clients.begin(), it) + 1 << ": " << result << "\n";

accumulatedResult += result;

endTime = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed\_seconds = endTime - startTime;

std::cout << "Integration process took : " << elapsed\_seconds.count() << " seconds\n";

std::cout << "Accumulated result: " <<

accumulatedResult << "\n";

}

}

else {

std::cout << "Client " << std::distance(clients.begin(), it) + 1 << " disconnected\n";

closesocket(clientSocket);

it = clients.erase(it);

continue;

}

}

++it;

}

}

for (const auto& client : clients) {

closesocket(client);

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

Листинг программы(Клиент):

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

#include <sstream>

#include <cmath>

#include <thread>

#include <vector>

#pragma comment(lib, "ws2\_32.lib")

double Function(double x) {

return 1.0 / log(x);

}

double CalculateIntegral(double start, double end, double step) {

double sum = 0.0;

for (double x = start; x < end; x += step) {

sum += Function(x) \* step;

}

return sum;

}

void CalculateAndSendResult(SOCKET clientSocket, double start, double end, double step) {

double result = CalculateIntegral(start, end, step);

std::ostringstream resultStr;

resultStr << "Result-" << result;

send(clientSocket, resultStr.str().c\_str(), resultStr.str().size() + 1, 0);

}

int main() {

WSADATA wsaData;

setlocale(LC\_ALL, "en\_us");

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "WSAStartup failed\n";

return 1;

}

std::string serverIP;

std::cout << "Enter server IP: ";

std::cin >> serverIP;

int port;

std::cout << "Enter server port: ";

std::cin >> port;

SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed\n";

WSACleanup();

return 1;

}

sockaddr\_in serverAddr{};

serverAddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, serverIP.c\_str(), &(serverAddr.sin\_addr));

serverAddr.sin\_port = htons(port);

if (connect(clientSocket, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Connection failed\n";

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::cout << "Connected to server\n";

char paramsBuffer[1024];

char paramsBuffer1[1024];

recv(clientSocket, paramsBuffer, sizeof(paramsBuffer), 0);

double start1, end1, step1;

std::istringstream paramsStream(paramsBuffer);

paramsStream >> start1 >> end1 >> step1;

std::cout << "Received first integral parameters from server: start=" << start1 << ", end=" << end1 << ", step=" << step1 << "\n";

recv(clientSocket, paramsBuffer1, sizeof(paramsBuffer1), 0);

std::istringstream paramsStream1(paramsBuffer1);

double start2, end2, step2;

paramsStream1 >> start2 >> end2 >> step2;

std::cout << "Received second integral parameters from server: start=" << start2 << ", end=" << end2 << ", step=" << step2 << "\n";

std::vector<std::thread> threads;

unsigned int numThreads = std::thread::hardware\_concurrency();

threads.reserve(numThreads);

threads.emplace\_back(CalculateAndSendResult, clientSocket, start1, end1, step1);

threads.emplace\_back(CalculateAndSendResult, clientSocket, start2, end2, step2);

for (auto& thread : threads) {

thread.join();

}

closesocket(clientSocket);

WSACleanup();

return 0;

}

Результат выполнения программы:

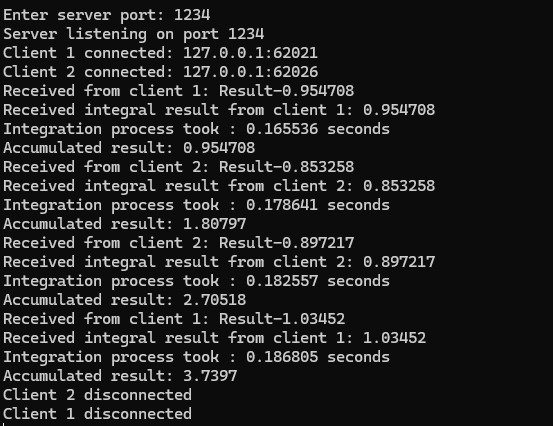


Рисунок 1-Результат работы программы

Вывод:в процессе выполнения лабораторной работы была реализована серверная составляющая,которая порционно выдает клиентам интервалы для вычисления и обрабатывает результаты вычислений.