Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра "Вычислительной техники"

**ОТЧЁТ**

По лабораторной №2

по дисциплине "Распределенные вычисления"

на тему " Разработка простейшей распределенной вычислительной системы"

Вариант № 7

Выполнили студенты группы 20ВВП1:

Тумасов В.В

Ермолаев А.Д

Приняли:

Федюнин Р. Н.

Елфимов А.В

Пенза 2024

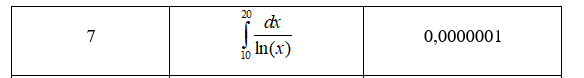
Разработка простейшей распределенной вычислительной системы

**Цель работы:** изучение подхода к реализации распределенных

вычислений и разработка с его использованием простейшей распределенной

вычислительной системы.

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 выдачу задания каждому клиенту и прием результатов  
вычислений, а также 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), прием от  
сервера задания, расчет задания и возврат результата вычислений серверу.  
Параллелизм реализуется на уровне клиентских процессов без применения  
многопоточной архитектуры внутри каждого из клиентов. Объем задания для  
каждого клиента должен быть одинаков и определен в самом начале  
вычислений, после подключения всех клиентов.

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)Обработка сообщения "Task" клиентом:

if (message == "Task") {

double start, end, step;

char paramsBuffer[1024];

// Receive the parameters from the server

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

// Parse the received string back to double values

std::istringstream paramsStream(paramsBuffer);

paramsStream >> start >> end >> step;

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

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

// Convert the result to a string before sending

std::ostringstream resultStr;

resultStr << "Result-" << result;

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

}

}

5)Обработка сообщения "Task" сервером:

if (std::string(buffer) == "Task") {

double start = 10.0;

double end = 20.0;

double step = 0.0000001;

if (std::distance(clients.begin(), it) + 1 == 1)

{

start = 10;

end = 15;

}

else {

start = 15;

end = 20;

}

// Convert double values to strings

std::ostringstream startStr, endStr, stepStr;

startStr << start;

endStr << end;

stepStr << step;

std::string params = startStr.str() + " " + endStr.str() + " " + stepStr.str();

std::cout << "Sending integral parameters to client " << std::distance(clients.begin(), it) + 1 << ": " << params << "\n";

// Send the parameters to the client

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

6)Обработка результата и его вывод:

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

// Extract the result from the message (skip "Result-" prefix)

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

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

// Accumulate the result received from the client

accumulatedResult += result;

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>

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

int main() {

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;

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)) {

// New connection

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";

}

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";

// Handle your logic here based on received data

// For example, if received "Task"

if (std::string(buffer) == "Task") {

double start = 10.0;

double end = 20.0;

double step = 0.0000001;

if (std::distance(clients.begin(), it) + 1 == 1)

{

start = 10;

end = 15;

}

else {

start = 15;

end = 20;

}

// Convert double values to strings

std::ostringstream startStr, endStr, stepStr;

startStr << start;

endStr << end;

stepStr << step;

std::string params = startStr.str() + " " + endStr.str() + " " + stepStr.str();

std::cout << "Sending integral parameters to client " << std::distance(clients.begin(), it) + 1 << ": " << params << "\n";

// Send the parameters to the client

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

}

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

// Extract the result from the message (skip "Result-" prefix)

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

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

// Accumulate the result received from the client

accumulatedResult += result;

std::cout << "Accumulated result: " << accumulatedResult << "\n";

}

}

else {

// Connection closed by client

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

closesocket(clientSocket);

it = clients.erase(it);

continue;

}

}

++it;

}

}

// Cleanup

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>

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

double Function(double x) {

return 1.0 / log(x);

}

// Функция для расчета интеграла

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;

}

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";

while (true) {

std::string message;

std::cout << "Enter message (or 'exit' to quit): ";

std::getline(std::cin, message);

if (message == "exit") {

break;

}

send(clientSocket, message.c\_str(), message.size(), 0);

char buffer[1024];

if (message == "Task") {

double start, end, step;

char paramsBuffer[1024];

// Receive the parameters from the server

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

// Parse the received string back to double values

std::istringstream paramsStream(paramsBuffer);

paramsStream >> start >> end >> step;

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

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

// Convert the result to a string before sending

std::ostringstream resultStr;

resultStr << "Result-" << result;

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

}

}

closesocket(clientSocket);

WSACleanup();

return 0;

}

Результат выполнения программы:



Рисунок 1-Результат работы программы

Вывод:в процессе выполнения лабораторной работы был изучен метод создания и использования потоков для параллельного выполнения задач,на примере вычисления определенного интеграла.