МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

дисциплина «Программирование сетевых приложений»

на тему «Организация распределённых вычислений с использованием сокет TCP/IP средствами WinAPI»

Выполнил: студент гр. ИТП-41

Болотникова Е.В.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2022

**Цель работы:** научиться организовывать распределенные вычисления с использованием сокетов *TCP*/*IP* средствами *WinAPI*.

**Задание:** распределенное вычисление площади поверхности фигуры одним из предложенных по варианту методов. Исходными данными для вычисления являются:

● отрезок [A; B];

● точность вычислений;

● количество клиентов, решающих задачу.

Вычислить площадь фигуры, описанную следующей функцией

методом трапеций.

Для выполнения лабораторной работы было разработано два консольных приложения: клиентское и серверное. Сервер принимает от пользователя информацию о количестве клиентов и пределы интегрирования. Серверная часть делит данные для обработки между клиентами, после получения результатов вычислений от клиентов подсчитывает финальный результат и выводит его на экран. Клиентская часть подключается к серверному приложению по установленному порту, получает от сервера данные для расчетов, производит необходимые вычисления и возвращает результат серверу.

Результат работы первого клиента приведен на рисунке 1.

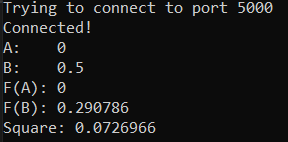


Рисунок 1 – Результат работы первого клиента

Результат работы второго клиента приведен на рисунке 2.

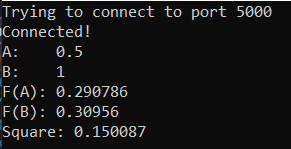


Рисунок 2 – Результат работы второго клиента

На рисунке 3 приведен результат подключения клиентов к серверу, результаты работы клиентов и площадь фигуры, описанной заданной функцией.

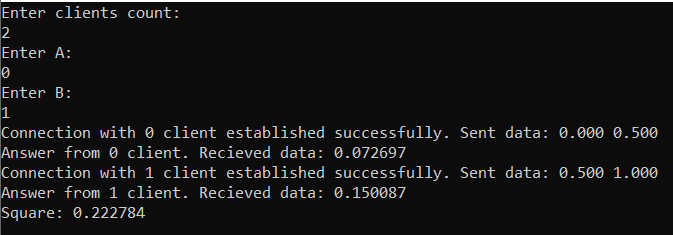


Рисунок 3 –Результат работы сервера

Для проверки точности вычисления результаты работы программы были сравнены с результатами, полученными в онлайн калькуляторе. Результат проверки на онлайн калькуляторе представлен на рисунке 4.

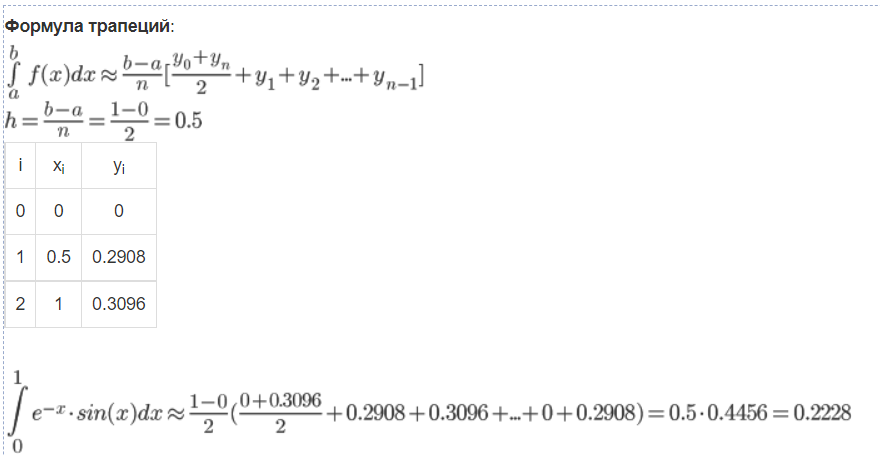


Рисунок 4 – Результаты вычислений калькулятора

Листинг приложения приведён в приложении А.

**Вывод**: в ходе выполнения лабораторной работы было разработано сетевое приложение, которое организует распределенные вычисления с использованием сокетов *TCP*/*IP* средствами *WinAPI*.

Приложение А

Server

#include "ServerTCP.h"

int main()

{

const int PORT = 5000;

int clientCount = 0;

std::string input;

std::string a;

std::string b;

do

{

std::cout << "Enter clients count:" << std::endl;

std::cin >> input;

} while (input[0] < '0' || input[0] > '9');

clientCount = stoi(input);

Server server(PORT, clientCount);

do

{

std::cout << "Enter A:" << std::endl;

std::cin >> a;

} while (a[0] == '-');

do

{

std::cout << "Enter B:" << std::endl;

std::cin >> b;

} while (b[0] == '-');

std::thread\* threads = server.listenClients(stod(a), stod(b));

while (server.clientFinished != clientCount);

std::cout << "Square: " << server.getSquare() << std::endl;

int tmp = getchar();

return 0;

}

ServerTCP

#include "ServerTCP.h"

Server::Server(int port, int clientCount)

{

square = 0.0;

clientFinished = 0;

this->clientCount = clientCount;

clients = new SOCKET[clientCount];

clientThreads = new std::thread[clientCount];

sockVer = MAKEWORD(2, 2);

retVal = WSAStartup(sockVer, &wsaData);

//создание сокета

sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (sock == INVALID\_SOCKET)

{

//ошибка создания сокета

std::cout << "Unable to create socket" << std::endl;

WSACleanup();//Деинициаоизация бибилиотеки Winsoock

}

//связывание сокета с локальным адресом

SOCKADDR\_IN sin;

sin.sin\_family = PF\_INET;

sin.sin\_port = htons(port);

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

//вызываем bind для связывания

retVal = bind(sock, (LPSOCKADDR)&sin, sizeof(sin));

if (retVal == SOCKET\_ERROR)

{

std::cout << "Unable to bind" << std::endl;

WSACleanup();//Деинициаоизация бибилиотеки Winsoock

}

}

std::thread\* Server::listenClients(double start, double end)

{

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

{

int retVal = listen(sock, 10);

if (retVal == SOCKET\_ERROR)

{

std::cout << "Unable to listen" << std::endl;

}

clients[i] = accept(sock, NULL, NULL);

if (clients[i] == INVALID\_SOCKET)

{

std::cout << "Unable to accept" << std::endl;

}

clientThreads[i] = std::thread(ProcessingClients, clients[i], clientCount, i, &clientFinished, start, end, &square);

clientThreads[i].detach();

}

return clientThreads;

}

double Server::getSquare()

{

return square;

}

void ProcessingClients(SOCKET client, int count, int index, int\* clientFinished, double start, double end, double\* square) {

const int RESVBUF = 32;

double step = (end - start) / count;

double x1 = step \* (double)index + start;

double x2 = step \* (1 + (double)index) + start;

std::stringstream stream\_x1;

stream\_x1 << std::fixed << std::setprecision(3) << x1;

std::stringstream stream\_x2;

stream\_x2 << std::fixed << std::setprecision(3) << x2;

std::string sendMessage = stream\_x1.str() + " " + stream\_x2.str();

std::cout << "Connection with " << index << " client established successfully. Sent data: " << sendMessage << std::endl;

int err = 0;

//

err = send(client, sendMessage.c\_str(), sendMessage.length(), 0);

char receiveMessageBuf[RESVBUF];

//

err = recv(client, receiveMessageBuf, RESVBUF, 0);

std::string receiveMessage = "";

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

{

receiveMessage += receiveMessageBuf[i];

}

double receive = atof(receiveMessage.c\_str());

std::cout << "Answer from " << index << " client. Recieved data: " << receive << std::endl;

\*square += receive;

\*clientFinished += 1;

}

Server::~Server()

{

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

{

closesocket(clients[i]);

}

delete[] clientThreads;

delete[] clients;

closesocket(sock);

WSACleanup();

}

#include <iostream>

#include <cmath>

#include <cstdlib>

#include "ClientTCP.h"

double function(double x) {

return sin(x) \* exp(-x);

}

double methodTrapezoid(double x1, double x2, double y1, double y2) {

return (y1 + y2) \* (x2 - x1) / 2;

}

int main()

{

const int PORT = 5000;

std::string message;

double x1, x2, y1, y2;

double result = 0;

ClientTCP client(PORT);

message = client.receiveMessage(15);

x1 = atof(message.substr(0, message.find(' ')).c\_str());

x2 = atof(message.substr(message.find(' ') + 1, message.length() - message.find(' ') - 1).c\_str());

y1 = function(x1);

y2 = function(x2);

std::cout << "A: " << x1 << std::endl;

std::cout << "B: " << x2 << std::endl;

std::cout << "F(A): " << y1 << std::endl;

std::cout << "F(B): " << y2 << std::endl;

result = methodTrapezoid(x1, x2, y1, y2);

std::cout << "Square: " << result << std::endl;

client.sendMessage(std::to\_string(result));

int tmp = getchar();

return 0;

}

#include "ClientTCP.h"

ClientTCP::ClientTCP(int port)

{

std::cout << "Trying to connect to port " << port << std::endl;

ver = MAKEWORD(2, 2);

retVal = WSAStartup(ver, (LPWSADATA)&wsaData);

hostEnt = gethostbyname("localhost");

if (!hostEnt)

{

std::cout << "Unable to collect gethostbyname" << std::endl;

WSACleanup();

}

sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (sock == SOCKET\_ERROR)

{

std::cout << "Unable to create socket" << std::endl;

WSACleanup();

}

SOCKADDR\_IN serverInfo;

serverInfo.sin\_family = PF\_INET;

if (hostEnt != nullptr)

{

serverInfo.sin\_addr = \*((LPIN\_ADDR)\*hostEnt->h\_addr\_list);

}

serverInfo.sin\_port = htons(port);

retVal = connect(sock, (LPSOCKADDR)&serverInfo, sizeof(serverInfo));

if (retVal == SOCKET\_ERROR)

{

std::cout << "Unable to connect" << std::endl;

WSACleanup();

} else {

std::cout << "Connected!" << std::endl;

}

}

ClientTCP::~ClientTCP()

{

closesocket(sock);

WSACleanup();

}

void ClientTCP::sendMessage(std::string message)

{

retVal = send(sock, message.c\_str(), message.length(), 0);

if (retVal == SOCKET\_ERROR)

{

std::cout << "Unable to send" << std::endl;

WSACleanup();

}

}

std::string ClientTCP::receiveMessage(int size)

{

char \*szResponse = new char[size];

retVal = recv(sock, szResponse, size, 0);

if (retVal == SOCKET\_ERROR)

{

std::cout << "Unable to receive" << std::endl;

WSACleanup();

}

return to\_string(szResponse, size);

}

std::string to\_string(char\* arr, int size) {

std::string s = "";

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

s = s + arr[i];

}

return s;

}