**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

по дисциплине «Программирование сетевых приложений»

на тему: «**Организация распределённых вычислений с**

**использованием сокет *TCP/IP* средствами *WinAPI*.**»

Выполнил: студент гр. ИТИ-31

Гончар А.С.

Принял: ассистент

Карабчикова Е.А.

Гомель 2020

**Цель работы**: изучить подходы к использованию протокола *TCP/IP* средствами *WinSocks*.

Задание для 5 варианта: Распределенная сортировка, для сбора от клиентов использовать mergesort, на клиентах использовать сортировку вставкой.

**Результат выполнения:**

В ходе выполнения лабораторной работы были разработаны клиент и сервер:

– сервер для считывания данных с файла, разделения на порции, передачи на остальные клиенты и сборки результирующих частей;

– клиент, представляющий собой юнит для вычисления порции данных: сортировки последовательности, переданной с основного клиента.

Для анализа эффективности расчетов на нескольких узлах, были произведены вычисления и созданы графики, описывающие зависимость скорости вычисления от количества узлов, а также от количества исходных данных.

На графике 1 представлены графики зависимости времени выполнения сортировки от количества данных, при расчетах на разном количестве узлов.

График 1 – График зависимости количества данных от количества узлов

**Вывод**: в ходе выполнения лабораторной работы была изучена технология программирования сетевых сокетов на платформе *Windows*: *WinSocks*. Также, в ходе выполнения был произведен сравнительный анализ вычисления заданного алгоритма на одном клиенте и на нескольких, в результате чего, были доказаны преимущества распределенного подхода над вычислением на одном узле.

**ПРИЛОЖЕНИЕ А**

Листинг программы

Файл ***ServerApp.cpp***:

// Пример простого TCP-эхо-сервера

#include <stdio.h>

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

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h> // Wincosk2.h должен быть раньше windows!

#include <windows.h>

#include <locale.h>

#include <iostream>

#include <vector>

#include <ctime>

using namespace std;

#define MY\_PORT 666 // Порт, который слушает сервер 666

vector<int> merge(vector<int> a1, vector<int> a2) {

const int a1l = a1.size();

const int a2l = a2.size();

const int a3l = a1l + a2l;

vector<int> a3(a3l);

int i = 0, j = 0;

for (int k = 0; k < a3l; k++) {

if (i > a1l - 1) {

int a = a2[j];

a3[k] = a;

j++;

}

else if (j > a2l - 1) {

int a = a1[i];

a3[k] = a;

i++;

}

else if (a1[i] < a2[j]) {

int a = a1[i];

a3[k] = a;

i++;

}

else {

int b = a2[j];

a3[k] = b;

j++;

}

}

return a3;

}

// глобальная переменная - количество активных пользователей

int nclients = 0;

int arrres[1000];

vector<vector<int>> arr;

int CountOfClients;

int ArraySize;

int PartSize;

long start\_time, end\_time;

void final() {

vector<int> result(0);

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

result = merge(result, arr[i]);

}

end\_time = clock() - start\_time;

cout << end\_time;

cout << " ms\n";

for (vector<int>::iterator it = result.begin(); it != result.end(); ++it) {

cout << \*it;

cout << "\n";

}

}

int main()

{

cout << "Enter the number of clients:\n";

cin >> CountOfClients;

do {

cout << "Enter the number of items:\n";

cin >> ArraySize;

} while ((ArraySize % CountOfClients) != 0);

PartSize = ArraySize / CountOfClients;

char buff[1024];

setlocale(0, "");

arr = vector<vector<int>>(CountOfClients, vector<int>(PartSize));

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

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

arr[i][j] = rand() % 1000 + 1;

}

}

printf("TCP SERVER \n");

// Шаг 1 - Инициализация Библиотеки Сокетов

// т.к. возвращенная функцией информация не используется

// ей передается указатель на рабочий буфер, преобразуемый к указателю

// на структуру WSADATA.

// Такой прием позволяет сэкономить одну переменную, однако, буфер

// должен быть не менее полкилобайта размером (структура WSADATA

// занимает 400 байт)

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

{

// Ошибка!

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

return -1;

}

// Шаг 2 - создание сокета

SOCKET mysocket;

// AF\_INET - сокет Интернета

// SOCK\_STREAM - потоковый сокет (с установкой соединения)

// 0 - по умолчанию выбирается TCP протокол

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

{

// Ошибка!

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

WSACleanup(); // Деиницилизация библиотеки Winsock

return -1;

}

// Шаг 3 - связывание сокета с локальным адресом

sockaddr\_in local\_addr;

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

local\_addr.sin\_port = htons(MY\_PORT); // не забываем о сетевом порядке!!!

local\_addr.sin\_addr.s\_addr = 0; // сервер принимает подключения

// на все свои IP-адреса

// вызываем bind для связывания

if (bind(mysocket, (sockaddr\*)&local\_addr, sizeof(local\_addr)))

{

// Ошибка

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

closesocket(mysocket); // закрываем сокет!

WSACleanup();

return -1;

}

// Шаг 4 - ожидание подключений

// размер очереди - 0x100

if (listen(mysocket, 0x100))

{

// Ошибка

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

closesocket(mysocket);

WSACleanup();

return -1;

}

printf("Ожидание подключений...\n");

// Шаг 5 - извлекаем сообщение из очереди

SOCKET client\_socket; // сокет для клиента

sockaddr\_in client\_addr; // адрес клиента (заполняется системой)

// функции accept необходимо передать размер структуры

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

vector<SOCKET> client\_sockets(CountOfClients);

// цикл извлечения запросов на подключение из очереди

while ((client\_socket = accept(mysocket, (sockaddr\*)&client\_addr, \

& client\_addr\_size)))

{

client\_sockets[nclients] = client\_socket;

nclients++; // увеличиваем счетчик подключившихся клиентов

// пытаемся получить имя хоста

HOSTENT\* hst;

hst = gethostbyaddr((char\*)&client\_addr.sin\_addr.s\_addr, 4, AF\_INET);

// вывод сведений о клиенте

printf("+%s [%s] new connect!\n",

(hst) ? hst->h\_name : "", inet\_ntoa(client\_addr.sin\_addr));

// Вызов нового потока для обслужвания клиента

// Да, для этого рекомендуется использовать \_beginthreadex

// но, поскольку никаких вызовов функций стандартной Си библиотеки

// поток не делает, можно обойтись и CreateThread

DWORD thID;

start\_time = clock();

if (nclients == CountOfClients) {

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

send(client\_sockets[i], (char\*)&PartSize, sizeof(int), 0);

send(client\_sockets[i], (char\*)&arr[i][0], sizeof(int) \* PartSize, 0);

int bytes\_recv;

bytes\_recv = recv(client\_sockets[i], (char\*)&arr[i][0], sizeof(int) \* PartSize, 0);

nclients--; // уменьшаем счетчик активных клиентов

printf("-disconnect\n");

{

// закрываем сокет

closesocket(client\_sockets[i]);

}

}

final();

}

}

return 0;

}

Файл ***ClientApp.cpp***:

// Пример простого TCP-клиента

#include <stdio.h>

#include <string.h>

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

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

#include <winsock2.h>

#include <windows.h>

#include <iostream>

#include <vector>

using namespace std;

#define PORT 666

char SERVERADDR[11];

int main(int argc, char\* argv[])

{

char buff[1024];

printf("TCP CLIENT\n");

int choose;

cout << "Choose:\n1.Local IP\n2.Enter IP\n";

cin >> choose;

switch (choose)

{

case 1:

strcpy\_s(SERVERADDR, "127.0.0.1");

break;

case 2:

cout << "Enter server's ip address:";

cin >> SERVERADDR;

break;

}

// Шаг 1 - инициализация библиотеки Winsock

if (WSAStartup(0x202, (WSADATA\*)&buff[0]))

{

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

return -1;

}

// Шаг 2 - создание сокета

SOCKET my\_sock;

my\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (my\_sock < 0)

{

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

return -1;

}

// Шаг 3 - установка соединения

// заполнение структуры sockaddr\_in - указание адреса и порта сервера

sockaddr\_in dest\_addr;

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

dest\_addr.sin\_port = htons(PORT);

HOSTENT\* hst;

// преобразование IP адреса из символьного в сетевой формат

if (inet\_addr(SERVERADDR) != INADDR\_NONE)

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

else

{

// попытка получить IP адрес по доменному имени сервера

if (hst = gethostbyname(SERVERADDR))

// hst->h\_addr\_list содержит не массив адресов,

// а массив указателей на адреса

((unsigned long\*)&dest\_addr.sin\_addr)[0] =

((unsigned long\*\*)hst->h\_addr\_list)[0][0];

else

{

printf("Invalid address %s\n", SERVERADDR);

closesocket(my\_sock);

WSACleanup();

return -1;

}

}

// адрес сервера получен - пытаемся установить соединение

if (connect(my\_sock, (sockaddr\*)&dest\_addr, sizeof(dest\_addr)))

{

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

return -1;

}

printf("Connection to %s successfully\n \

Type quit for quit\n\n", SERVERADDR);

// Шаг 4 - чтение и передача сообщений

int arraysize = 0;

int nsize;

nsize = recv(my\_sock, (char\*)&arraysize, sizeof(int), NULL) != SOCKET\_ERROR;

vector<int> res(arraysize);

nsize = recv(my\_sock, (char\*)&res[0], arraysize \* sizeof(int), NULL) != SOCKET\_ERROR;

nsize = nsize / sizeof(int);

// ставим завершающий ноль в конце строки

buff[nsize] = 0;

int tmp;

for (int i = 1; i < arraysize; i++)

for (int j = i; j > 0 && res[j - 1] > res[j]; j--)

{

tmp = res[j - 1];

res[j - 1] = res[j];

res[j] = tmp;

}

// передаем строку клиента серверу

send(my\_sock, (char\*)&res[0], sizeof(int) \* arraysize, 0);

for (const auto& e : res) {

cout << e << endl;

}

closesocket(my\_sock);

WSACleanup();

int i;

cin >> i;

return 0;

}