МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**Практика №2**

**«**Разработка сетевого интерфейса

системы управления базой данных**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-221», «АВТФ»  Стеклов Данил Евгеньевич  «31» октября 2023г | Проверил:  *Ассистент ЗИ*  *Исаев Глеб Андреевич*  «\_\_\_» \_\_\_\_\_\_ 2023г |

Новосибирск 2023

**Цели и задачи работы:** разобраться в разработке сетевого интерфейса СУБД из практики 1 на С, Rust или Go

**Задание к работе:**

Требуется реализовать сетевой интерфейс для СУБД из практики 1.

При запуске приложения из первой практики СУБД должна

ожидать соединение по протоколу tcp на порту 6379. При

подключении требуется обработать запрос либо в отдельном

потоке, либо в отдельном процессе, либо асинхронной задачей

и отдать результат в ответ на запрос.

Также необходимо позаботится о блокировках на структуре БД

если используются потоки или процессы.

**Задание на языке C:**

**Проект Сервера:**

**1. Header.h**

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <Windows.h>

#include <string.h>

#include <locale.h>

#include <fcntl.h>

**typedef** **struct** Set { *//Множество*

char value[30];

} Set;

**typedef** **struct** List { *//Стек и очередь*

char value[30];

**struct** List\* next;

} List;

**typedef** **struct** HashTable { *//Хеш-таблица с методом открытых адресаций*

char key[15];

char value[30];

char is\_exist;

} HashTable;

void PushToHeadList(List\*\* node, List\*\* tail, char\* value); *//Добавить в начало списка*

void PushToTailList(List\*\* node, List\*\* tail, char\* value); *//Добавить в конец списка*

char\* PopFromHead(List\*\* node); *//Удалить из начала списка*

int WriteListToFile(char\* name, char\* value); *//Запись листа в файл*

int WriteHashToFile(char\* key, char\* value, char is\_exist);

int DeleteStrFromFile(char\* str, char\* path); *//Удаление строки из файла*

int ReplaceStrInFile(char\* str, char\* new\_str, char\* path); *//Замена строки в файле*

void СoncatenationStr(char\* name, char\* value, char\* str); *//Конкатенация строк*

void СoncatenationStrForHash(char\* key, char\* value, char is\_exist, char\* str); *//Конкатенация строк для хеш-таблицы*

void SetAdd(Set\* array, char\* name, char\* value, char\* callback);

void SetRem(Set\* array, char\* name, char\* value, char\* path, char\* callback);

void SetIsMember(Set\* array, char\* value, char\* callback);

void SetReadFromFile(Set\*\* array, char\* main\_name); *//Чтение множества из файла(заполнение структуры)*

void StackPush(List\* node, List\* tail, char\* name, char\* value);

void StackPop(List\* node, char\* name, char\* path, char\* callback);

void StackReadFromFile(List\*\* node, List\*\* tail, char\* name); *//Чтение стека*

void QueuePush(List\* node, List\* tail, char\* name, char\* value);

void QueuePop(List\* node, char\* name, char\* path, char\* callback);

void QueueReadFromFile(List\*\* node, List\*\* tail, char\* name); *//Чтение очереди*

void HashSet(HashTable\* row, char\* key, char\* value, char\* path);

void HashDelete(HashTable\* row, char\* key, char\* path);

void HashGet(HashTable\* row, char\* key, char\* path, char\* callback);

void HashReadFromFile(HashTable\*\* row); *//Чтение хеш-таблицы*

int FuncHash(char\* key); *//Первая хеш-функция*

int FuncHashSec(char ch); *//Вторая хеш-функция*

int Work(int argc, char\* argv[], char\* callback);

DWORD WINAPI CommunicationClient(LPVOID lpParam);

**1. DataBase.c**

#include "Header.h"

#pragma warning(disable : 4996)

**enum** Command {

SADD,

SREM,

SISMEMBER,

SPUSH,

SPOP,

QPUSH,

QPOP,

HSET,

HDEL,

HGET

};

FILE\* fp;

int Work(int argc, char\* argv[], char\* callback) {

Set\* array = NULL;

List\* node = NULL;

List\* tail = NULL;

HashTable\* row = NULL;

char order[12];

char name[20];

char key[15];

char value[30];

**if** (argc < 6) { *//Проверка на кол-во аргументов*

puts("Ошибка аргументов");

**return** EXIT\_FAILURE;

}

**if** (strcmp(argv[1], "--file") != 0) {

perror("Ошибка флагов");

**return** EXIT\_FAILURE;

}

**if** ((fopen\_s(&fp, argv[2], "a+")) != NULL) {

perror("Ошибка открытия файла");

**return** EXIT\_FAILURE;

}

**if** (strcmp(argv[3], "--query") != 0) {

perror("Ошибка флагов 2");

**return** EXIT\_FAILURE;

}

**for** (int i = 0; i < strlen(argv[4]); ++i) {

order[i] = \*(argv[4] + i + 1);

}

**if** (argc == 6) { *//Парс входных данных*

**for** (int i = 0; i <= strlen(argv[5]) - 1; ++i) {

name[i] = \*(argv[5] + i);

}

name[strlen(argv[5]) - 2] = '\0';

}

**else** **if** (argc == 7) {

**for** (int i = 0; i <= strlen(argv[5]); ++i) {

name[i] = \*(argv[5] + i);

}

**for** (int i = 0; i < strlen(argv[6]) - 1; ++i) {

value[i] = \*(argv[6] + i);

}

value[strlen(argv[6]) - 2] = '\0';

}

**else** **if** (argc == 8) {

**for** (int i = 0; i <= strlen(argv[5]); ++i) {

name[i] = \*(argv[5] + i);

}

**for** (int i = 0; i <= strlen(argv[6]); ++i) {

key[i] = \*(argv[6] + i);

}

**for** (int i = 0; i < strlen(argv[7]) - 1; ++i) {

value[i] = \*(argv[7] + i);

}

value[strlen(argv[7]) - 2] = '\0';

}

**enum** Command command;

**if** (strcmp(order, "SADD") == 0) command = SADD;

**else** **if** (strcmp(order, "SREM") == 0) command = SREM;

**else** **if** (strcmp(order, "SISMEMBER") == 0) command = SISMEMBER;

**else** **if** (strcmp(order, "QPUSH") == 0) command = QPUSH;

**else** **if** (strcmp(order, "QPOP") == 0) command = QPOP;

**else** **if** (strcmp(order, "SPUSH") == 0) command = SPUSH;

**else** **if** (strcmp(order, "SPOP") == 0) command = SPOP;

**else** **if** (strcmp(order, "HSET") == 0) command = HSET;

**else** **if** (strcmp(order, "HDEL") == 0) command = HDEL;

**else** **if** (strcmp(order, "HGET") == 0) command = HGET;

**switch** (command)

{

**case** SADD:

SetReadFromFile(&array, name);

SetAdd(array, name, value, callback);

**break**;

**case** SREM:

SetReadFromFile(&array, name);

SetRem(array, name, value, argv[2], callback);

**break**;

**case** SISMEMBER:

SetReadFromFile(&array, name);

SetIsMember(array, value, callback);

**break**;

**case** SPUSH:

StackReadFromFile(&node, &tail, name);

StackPush(node, tail, name, value);

**break**;

**case** SPOP:

StackReadFromFile(&node, &tail, name);

StackPop(node, name, argv[2], callback);

**break**;

**case** QPUSH:

QueueReadFromFile(&node, &tail, name);

QueuePush(node, tail, name, value);

**break**;

**case** QPOP:

QueueReadFromFile(&node, &tail, name);

QueuePop(node, name, argv[2], callback);

**break**;

**case** HSET:

HashReadFromFile(&row);

HashSet(row, key, value, argv[2]);

**break**;

**case** HDEL:

HashReadFromFile(&row);

HashDelete(row, value, argv[2]);

**break**;

**case** HGET:

HashReadFromFile(&row);

HashGet(row, value, argv[2], callback);

**break**;

default:

**break**;

}

fclose(fp);

**return** EXIT\_SUCCESS;

}

void PushToHeadList(List\*\* node, List\*\* tail, char\* value) {

List\* tmp = (List\*)malloc(**sizeof**(List));

strcpy\_s(tmp->value, 30 ,value);

**if** (\*node == NULL) {

\*tail = tmp;

\*node = tmp;

tmp->next = NULL;

}

**else** {

tmp->next = \*node;

\*node = tmp;

}

}

void PushToTailList(List\*\* node, List\*\* tail, char\* value) {

List\* tmp = (List\*)malloc(**sizeof**(List));

strcpy\_s(tmp->value, 30 ,value);

tmp->next = NULL;

**if** (\*node == NULL) {

\*tail = tmp;

\*node = tmp;

}

**else** {

(\*tail)->next = tmp;

(\*tail) = tmp;

}

}

char\* PopFromHead(List\*\* node) {

**if** (\*node == NULL) {

char value[30] = "Список пуст";

**return** value;

}

List\* tmp = \*node;

char value[30];

strcpy\_s(value, 30, tmp->value);

\*node = tmp->next;

free(tmp);

**return** value;

}

int WriteListToFile(char\* name, char\* value) {

char str[50];

strcat\_s(value, 30, "**\n**");

СoncatenationStr(name, value, str);

**if** (fwrite(str, 1, strlen(str), fp) != strlen(str)) {

perror("Ошибка записи в файл");

**return** EXIT\_FAILURE;

}

**return** EXIT\_SUCCESS;

}

int WriteHashToFile(char\* key, char\* value, char is\_exist) {

char str[50];

СoncatenationStr(key, value, str);

str[strlen(str) + 3] = '\0';

str[strlen(str) + 2] = '\n';

str[strlen(str) + 1] = is\_exist;

str[strlen(str)] = ' ';

**if** (fwrite(str, 1, strlen(str), fp) != strlen(str)) {

perror("Ошибка записи в файл");

**return** EXIT\_FAILURE;

}

**return** EXIT\_SUCCESS;

}

int DeleteStrFromFile(char\* str, char\* path) {

char buff[50];

char change\_buff[50];

char is\_del = '0';

fseek(fp, 0, SEEK\_END);

int size = ftell(fp);

char\* data = (char\*)calloc(size, **sizeof**(char));

fseek(fp, 0, SEEK\_SET);

**while** (fgets(buff, 50, fp)) {

strcpy\_s(change\_buff, 50, buff);

change\_buff[strlen(change\_buff) - 1] = '\0';

**if** ((strcmp(change\_buff, str) != 0) || (is\_del == '1')) {

strcat\_s(data, size, buff);

}

**else** is\_del = '1';

}

fclose(fp);

**if** ((fopen\_s(&fp, path, "w")) != NULL) {

perror("Ошибка открытия файла");

**return** EXIT\_FAILURE;

}

fputs (data, fp);

free(data);

**return** EXIT\_SUCCESS;

}

int ReplaceStrInFile(char\* str, char\* new\_str, char\* path) {

char buff[50];

fseek(fp, 0, SEEK\_END);

int size = ftell(fp) + 50;

char\* data = (char\*)calloc(size, **sizeof**(char));

fseek(fp, 0, SEEK\_SET);

**while** (fgets(buff, 50, fp)) {

**if** (strcmp(buff, str) != 0) {

strcat\_s(data, size, buff);

}

**else** {

strcat\_s(data, size, new\_str);

}

}

fclose(fp);

**if** ((fopen\_s(&fp, path, "w")) != NULL) {

perror("Ошибка открытия файла");

**return** EXIT\_FAILURE;

}

fputs (data, fp);

free(data);

**return** EXIT\_SUCCESS;

}

void СoncatenationStr(char\* name, char\* value, char\* str) {

strcpy\_s(str, 50, name);

str[strlen(str) + 1] = '\0';

str[strlen(str)] = ' ';

strcat\_s(str, 50, value);

}

void СoncatenationStrForHash(char\* key, char\* value, char is\_exist, char\* str) {

СoncatenationStr(key, value, str);

str[strlen(str) + 3] = '\0';

str[strlen(str) + 2] = '\n';

str[strlen(str) + 1] = is\_exist;

str[strlen(str)] = ' ';

}

void SetAdd(Set\* array, char\* name, char\* value, char\* callback) {

int index = FuncHash(value);

char is\_coincidence = '0';

**while** (array[index].value[0] != '\0') {

**if** (strcmp(array[index].value, value) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(value[0])) % 100;

}

**if** (is\_coincidence == '0') WriteListToFile(name, value);

**else** strcpy(callback, "Данный элемент уже есть в множестве");

}

void SetRem(Set\* array, char\* name, char\* value, char\* path, char\* callback) {

char str[50];

int index = FuncHash(value);

char is\_coincidence = '0';

**while** (array[index].value[0] != '\0') {

**if** (strcmp(array[index].value, value) == 0) {

is\_coincidence = '1';

СoncatenationStr(name, value, str);

**break**;

}

index = (index + FuncHashSec(value[0])) % 100;

}

**if** (is\_coincidence == '1') DeleteStrFromFile(str, path);

**else** strcpy(callback, "Такого элемента нет в множестве");

}

void SetIsMember(Set\* array, char\* value, char\* callback) {

int index = FuncHash(value);

char is\_coincidence = '0';

**while** (array[index].value[0] != '\0') {

**if** (strcmp(array[index].value, value) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(value[0])) % 100;

}

**if** (is\_coincidence == '1') strcpy(callback, "Такой элемент есть в множестве");

**else** strcpy(callback, "Такого элемента нет в множестве");

}

void SetReadFromFile(Set\*\* array, char\* main\_name) {

\*array = (Set\*)calloc(100, **sizeof**(Set));

char name[20];

char value[30];

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s**\n**", name, (unsigned)**sizeof**(name), value, (unsigned)**sizeof**(value));

**if** (strcmp(main\_name, name) == 0) {

int index = FuncHash(value);

**while** ((\*array)[index].value[0] != '\0') {

index = (index + FuncHashSec(value[0])) % 100;

}

strcpy\_s((\*array)[index].value, 30, value);

}

}

}

void StackPush(List\* node, List\* tail, char\* name, char\* value) {

PushToHeadList(&node, &tail, value); *//Добавляю в голову - становится первым элементом*

WriteListToFile(name, value);

}

void StackPop(List\* node, char\* name, char\* path, char\* callback) {

char value[30];

char str[50];

strcpy\_s(value, 30, PopFromHead(&node)); *//Удаляю голову - первый элемент*

strcpy(callback, value);

СoncatenationStr(name, value, str);

DeleteStrFromFile(str, path);

}

void StackReadFromFile(List\*\* node, List\*\* tail, char\* name) {

char text\_name[20];

char text\_value[30];

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s**\n**", text\_name, (unsigned)**sizeof**(text\_name), text\_value, (unsigned)**sizeof**(text\_value));

**if** (strcmp(text\_name, name) == 0) {

PushToHeadList(node, tail, text\_value); *//Первый элмент в файле - хвост*

}

}

}

void QueuePush(List\* node, List\* tail, char\* name, char\* value) {

PushToTailList(&node, &tail, value); *//Ложу в хвост - становится последним элементом*

WriteListToFile(name, value);

}

void QueuePop(List\* node, char\* name, char\* path, char\* callback) {

char value[30];

char str[50];

strcpy\_s(value, 30, PopFromHead(&node)); *//Удаляю голову - первый элемент*

strcpy(callback, value);

СoncatenationStr(name, value, str);

DeleteStrFromFile(str, path);

}

void QueueReadFromFile(List\*\* node, List\*\* tail, char\* name) {

char text\_name[20];

char text\_value[30];

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s**\n**", text\_name, (unsigned)**sizeof**(text\_name), text\_value, (unsigned)**sizeof**(text\_value));

**if** (strcmp(text\_name, name) == 0) {

PushToTailList(node, tail, text\_value); *//Первый элемент в файле - голова*

}

}

}

void HashSet(HashTable\* row, char\* key, char\* value, char\* path) {

int index = FuncHash(key);

char is\_coincidence = '0';

char is\_del = '0';

int index\_del;

char str[50];

char new\_str[50];

**while** (row[index].value[0] != '\0') {

**if** (row[index].is\_exist == '0') {

**if** (is\_del == '0') {

index\_del = index;

is\_del = '1';

}

**else** **break**;

}

**if** (strcmp(row[index].key, key) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(key[0])) % 100;

}

**if** (is\_coincidence == '1') {

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, str);

strcpy\_s(row[index].key, 15, key);

strcpy\_s(row[index].value, 30, value);

row[index].is\_exist = '1';

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, new\_str);

ReplaceStrInFile(str, new\_str, path);

}

**else** **if** (is\_del == '1') {

СoncatenationStrForHash(row[index\_del].key, row[index\_del].value, row[index\_del].is\_exist, str);

strcpy\_s(row[index\_del].key, 15, key);

strcpy\_s(row[index\_del].value, 30, value);

row[index\_del].is\_exist = '1';

СoncatenationStrForHash(row[index\_del].key, row[index\_del].value, row[index\_del].is\_exist, new\_str);

ReplaceStrInFile(str, new\_str, path);

}

**else** {

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, str);

strcpy\_s(row[index].key, 15, key);

strcpy\_s(row[index].value, 30, value);

row[index].is\_exist = '1';

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, new\_str);

WriteHashToFile(row[index].key, row[index].value, row[index].is\_exist);

}

}

void HashDelete(HashTable\* row, char\* key, char\* path) {

int index = FuncHash(key);

char str[50];

char new\_str[50];

char is\_coincidence = '0';

**while** (row[index].value[0] != '\0') {

**if** (strcmp(row[index].key, key) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(key[0])) % 100;

}

**if** (is\_coincidence == '1') {

СoncatenationStrForHash(row[index].key, row[index].value, '1', str);

СoncatenationStrForHash(row[index].key, row[index].value, '0', new\_str);

ReplaceStrInFile(str, new\_str, path);

}

}

void HashGet(HashTable\* row, char\* key, char\* path, char\* callback) {

int index = FuncHash(key);

char is\_coincidence = '0';

**while** (row[index].value[0] != '\0') {

**if** ((strcmp(row[index].key, key) == 0) && (row[index].is\_exist == '1')) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(key[0])) % 100;

}

**if** (is\_coincidence == '1') strcpy(callback, row[index].value);

**else** strcpy(callback, "Значение по такому ключу не найдено");

}

void HashReadFromFile(HashTable\*\* row) {

\*row = (HashTable\*)calloc(100, **sizeof**(HashTable));

char key[15];

char value[30];

char is\_exist = '\0';

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s %c**\n**", key, (unsigned)**sizeof**(key), value, (unsigned)**sizeof**(value), &is\_exist, (unsigned)**sizeof**(is\_exist));

int index = FuncHash(key);

**while** ((\*row)[index].value[0] != '\0') {

index = (index + FuncHashSec(key[0])) % 100;

}

strcpy\_s((\*row)[index].key, 15, key);

strcpy\_s((\*row)[index].value, 30, value);

(\*row)[index].is\_exist = is\_exist;

}

}

int FuncHash(char\* key) {

int sum = 0;

**for** (int i = 0; key[i]; ++i)

sum += key[i];

**return** sum % 100;

}

int FuncHashSec(char ch) {

int sum = ch \* 37;

**return** sum % 100;

}

**3. Main.c**

#include "Header.h"

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

#pragma warning(disable : 4996)

**typedef** **struct** ThreadArgs {

SOCKET socket;

}ThreadArgs;

int main() {

setlocale(0, "Ru");

WSADATA ws\_data; *//Необходимо для установки версии WinSock и последующего использования скоетов*

**if** (WSAStartup(MAKEWORD(2, 2), &ws\_data) != 0) {

printf("Ошибка инициализации WinSock. Код ошибки: %d**\n**", WSAGetLastError());

**return** EXIT\_FAILURE;

}

**else**

printf("WinSock инициализирован верно**\n**");

SOCKET serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); *//Создаем сокет IPv4, на основе протокола TCP,*

**if** (serv\_sock == INVALID\_SOCKET) {

printf("Ошибка создания сокета. Код ошибки: %d**\n**", WSAGetLastError());

closesocket(serv\_sock);

WSACleanup();

**return** EXIT\_FAILURE;

}

**else**

printf("Сокет инициализирован верно**\n**");

in\_addr ip\_to\_num; *//Создаем структуру адреса*

**if** (inet\_pton(AF\_INET, "127.0.0.1", &ip\_to\_num) <= 0) {

perror("Ошибка при преобразовании IP-адреса");

**return** EXIT\_FAILURE;

}

sockaddr\_in serv\_addr; *//Создаем структуру адреса сокета*

memset(&serv\_addr, 0, **sizeof**(serv\_addr)); *//Обнуляем все байты структуры*

serv\_addr.sin\_family = AF\_INET; *//IPv4*

serv\_addr.sin\_port = htons(6379); *//Порт 6379*

**if** (bind(serv\_sock, (sockaddr\*)&serv\_addr, **sizeof**(serv\_addr)) == -1) { *//Биндим сокет по указанной структуре*

printf("Ошибка привязки. Код ошибки: %d", WSAGetLastError());

closesocket(serv\_sock);

WSACleanup();

**return** EXIT\_FAILURE;

}

**else**

printf("Сокет успешно привязан**\n**");

**if** (listen(serv\_sock, 5) != 0) { *//Прослушиваем данный порт*

printf("Ошибка прослушивания. Код ошибки: %d**\n**", WSAGetLastError());

closesocket(serv\_sock);

WSACleanup();

**return** EXIT\_FAILURE;

}

**else** {

printf("Сервер ожидает подключений...**\n**");

}

**while** (1) {

sockaddr\_in client\_addr;

int client\_addr\_len = **sizeof**(client\_addr);

SOCKET client\_sock = accept(serv\_sock, (sockaddr\*)&client\_addr, &client\_addr\_len);

**if** (client\_sock == INVALID\_SOCKET) {

printf("Ошибка при принятии подключения. Код ошибки: %d**\n**", WSAGetLastError());

closesocket(serv\_sock);

WSACleanup();

**return** EXIT\_FAILURE;

}

**else** {

printf("Подключение приянто**\n**");

DWORD dw\_thread\_id;

HANDLE h\_thread;

ThreadArgs thread\_args;

thread\_args.socket = client\_sock;

h\_thread = CreateThread(NULL, 0, CommunicationClient, (LPVOID)&thread\_args, 0, &dw\_thread\_id); *// создание потока*

**if** (h\_thread == NULL) {

printf("Не удалось создать поток**\n**");

**return** EXIT\_FAILURE;

}

CloseHandle(h\_thread);

}

}

}

DWORD WINAPI CommunicationClient(LPVOID lpParam) {

char buffer[200];

char\* point\_buffer[8];

ThreadArgs\* args = (ThreadArgs\*)lpParam;

SOCKET client\_sock = args->socket;

**while** (recv(client\_sock, buffer, **sizeof**(buffer), 0) > 0) {

char\* str = strtok (buffer," ");

int i = 1;

**while** (str != NULL) {

point\_buffer[i] = (char\*)malloc(**sizeof**(str));

strcpy(point\_buffer[i], str);

str = strtok (NULL, " ");

i++;

}

char ans\_to\_client[50];

memset(ans\_to\_client, 0, 50);

int isErc = Work(i, point\_buffer, ans\_to\_client);

**if** ((isErc == EXIT\_SUCCESS) && (ans\_to\_client[0] == 0)) {

char callback[] = "Успешно!";

send(client\_sock, callback, **sizeof**(callback), 0);

}

**else** **if** ((isErc == EXIT\_SUCCESS) && (ans\_to\_client[0] != 0)){

send(client\_sock, ans\_to\_client, **sizeof**(ans\_to\_client), 0);

}

**else** {

char callback[] = "Произошла ошибка";

send(client\_sock, callback, **sizeof**(callback), 0);

}

}

**return** EXIT\_SUCCESS;

}

**Проект Клиента:**

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <string.h>

#include <locale.h>

#include <Windows.h>

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

int main() {

setlocale(0, "Ru");

WSADATA ws\_data; *//Необходимо для установки версии WinSock и последующего использования скоетов*

**if** (WSAStartup(MAKEWORD(2, 2), &ws\_data) != 0) {

printf("Ошибка инициализации WinSock. Код ошибки: %d**\n**", WSAGetLastError());

**return** EXIT\_FAILURE;

}

**else**

printf("WinSock инициализирован верно**\n**");

SOCKET serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); *//Создаем сокет IPv4, на основе протокола TCP,*

**if** (serv\_sock == INVALID\_SOCKET) {

printf("Ошибка создания сокета. Код ошибки: %d**\n**", WSAGetLastError());

closesocket(serv\_sock);

WSACleanup();

**return** EXIT\_FAILURE;

}

**else**

printf("Сокет инициализирован верно**\n**");

in\_addr ip\_to\_num; *//Создаем структуру адреса*

**if** (inet\_pton(AF\_INET, "127.0.0.1", &ip\_to\_num) <= 0) {

perror("Ошибка при преобразовании IP-адреса");

**return** EXIT\_FAILURE;

}

sockaddr\_in serv\_addr; *//Создаем структуру адреса сокета*

memset(&serv\_addr, 0, **sizeof**(serv\_addr)); *//Обнуляем все байты структуры*

serv\_addr.sin\_family = AF\_INET; *//IPv4*

serv\_addr.sin\_port = htons(6379); *//Порт 6379*

serv\_addr.sin\_addr = ip\_to\_num; *//127.0.0.1*

**if** (connect(serv\_sock, (sockaddr\*)&serv\_addr, **sizeof**(serv\_addr)) == SOCKET\_ERROR) {

printf("Ошибка подключения к сокету. Код ошибки: %d**\n**", WSAGetLastError());

closesocket(serv\_sock);

WSACleanup();

**return** EXIT\_FAILURE;

}

char buffer[200];

**do** {

printf("Введите команду: ");

fgets(buffer, **sizeof**(buffer), stdin);

send(serv\_sock, buffer, **sizeof**(buffer), 0);

recv(serv\_sock, buffer, **sizeof**(buffer), 0);

printf("%s**\n**", buffer);

} **while** (GetKeyState(VK\_ESCAPE) >= 0);

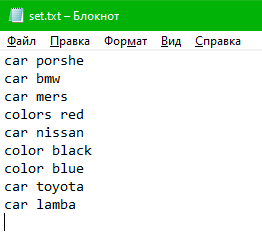
closesocket(serv\_sock);

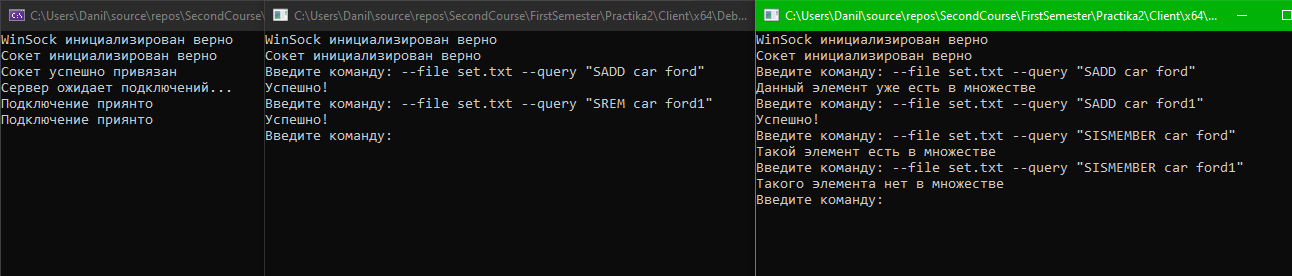
WSACleanup();

**return** EXIT\_SUCCESS;

}

**1. Пример работы на структуре множества:**

Файл set.txt



Запускаем сервер и двух клиентов, после выполняем ряд действий:

1. Первый клиент добавляет форд в множество

2. Второй клиент пытается добавить форд в множество

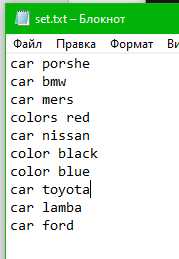
3. Второй клиент добавляет форд1 в множество

4. Второй клиент проверяет есть ли форд в множестве

5. Первый клиент удаляет форд1 из множества

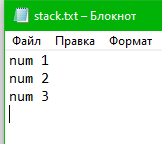
6. Второй проверяет, есть ли форд1 в множестве

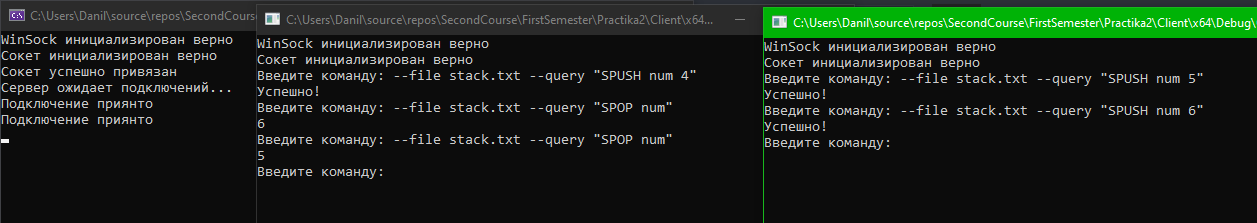
Файл set.txt после работы с множеством



**2. Пример работы на структуре стека:**

Файл stack.txt





Запускаем сервер и двух клиентов, после выполняем ряд действий:

1. Первый клиент добавляет 4 в стек

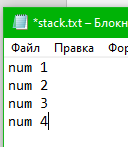
2. Второй клиент добавляет 5 в стек

3. Второй клиент добавляет 6 в стек

4. Первый клиент удаляет из стека

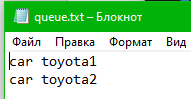
5. Первый клиент удаляет из стека

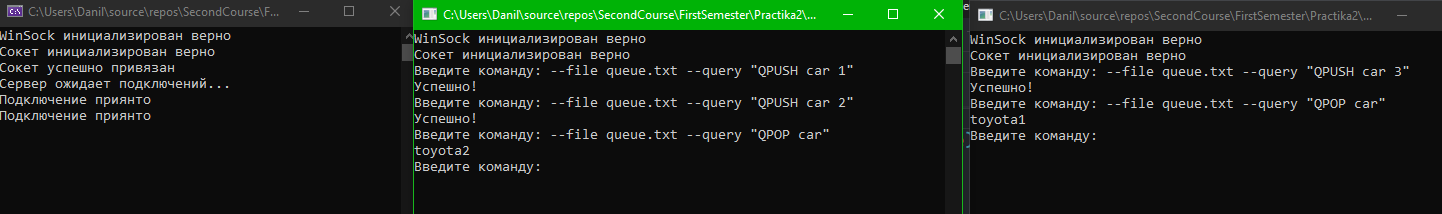
Файл stack.txt после работы



**3. Пример работы на структуре очереди:**

Файл queue.txt до работы





Запускаем сервер и двух клиентов, после выполняем ряд действий:

1. Первый клиент добавляет 1 в очередь

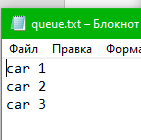
2. Первый клиент добавляет 2 в очередь

3. Второй клиент добавляет 3 в очередь

4. Второй клиент удаляет из очереди

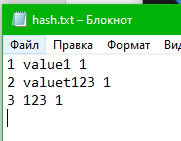
5. Первый клиент удаляет из очереди

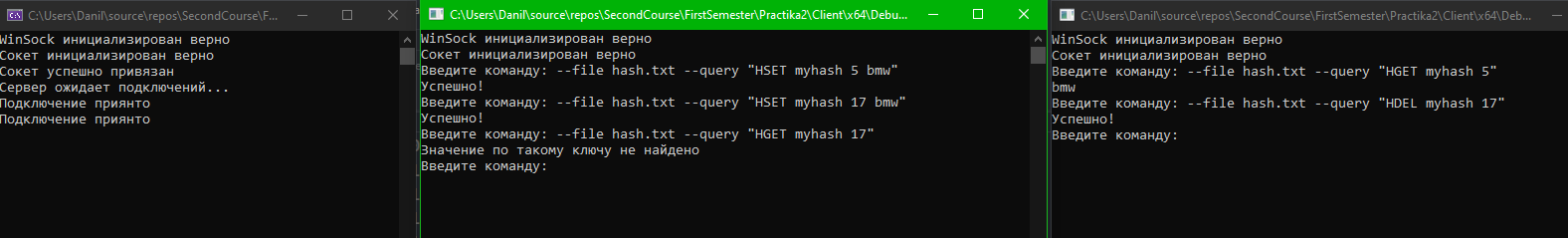
Файл queue.txt после работы



**3. Пример работы на структуре хеш-таблицы:**

Файл hash.txt до работы





Запускаем сервер и двух клиентов, после выполняем ряд действий:

1. Первый клиент добавляет bmw по ключу 5 в таблицу

2. Первый клиент добавляет bmw по ключу 17 в таблицу

3. Второй клиент возвращает элемент по ключу 5

4. Второй клиент удаляет элемент по ключу 17

5. Первый клиент возвращает элемент по ключу 17

Файл hash.txt после работы

