Кафедра защиты информации



**Практическая работа №2  
по дисциплине: «Программирование»**Разработка сетевого интерфейса системы управления базой данных

Выполнил: Проверил:  
Студент гр. АИ-22, АВТФ Преподаватель кафедры ЗИ  
Тимиров Т.Н. Архипова А. Б.

**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.

***Текст программ***

**Клиентская часть:**

**Source.cpp**

#include <iostream>

#include <locale.h>

#include <String>

#include "Client.h"

using namespace std;

string processFilePath(int argc, char\* argv[]) {

if (argc >= 4 && string(argv[1]) == "--file") return argv[2];

throw "Отсутсвует или неверно указан путь.";

}

string processQuery(int argc, char\* argv[]) {

if (argc >= 4 && string(argv[3]) == "--query") {

string query;

for (int i = 4; i < argc; i++) {

query = query + argv[i] + " ";

}

query = query.substr(1);

query.pop\_back();

query.pop\_back();

return query;

}

throw "Отсутсвует или неверно указан запрос.";

}

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

setlocale(LC\_ALL, "Russian");

Flags dataToSend;

try {

dataToSend.file\_path = processFilePath(argc, argv);

dataToSend.query = processQuery(argc, argv);

ClientConnection(dataToSend);

}

catch(const char\* error\_message){

cout << "->" << error\_message << endl;

}

return 0;

}

**Client.h**

#pragma once

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

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <thread>

#include <iostream>

#include <winsock2.h>

using namespace std;

SOCKET Connection;

struct Flags {

string file\_path;

string query;

};

enum Packet {

P\_ChatMessage,

P\_Test

};

bool ProcessPacket(Packet packettype) {

switch (packettype) {

case P\_ChatMessage:

{

int msg\_size;

recv(Connection, (char\*)&msg\_size, sizeof(int), NULL);

char\* msg = new char[msg\_size + 1];

msg[msg\_size] = '\0';

recv(Connection, msg, msg\_size, NULL);

cout << msg << endl;

delete[] msg;

break;

}

case P\_Test:

cout << "Test packet.\n";

break;

default:

cout << "Unrecognized packet: " << packettype << endl;

break;

}

return true;

}

void ClientHandler() {

int queryLength;

recv(Connection, reinterpret\_cast<char\*>(&queryLength), sizeof(queryLength), 0);

char\* queryBuffer = new char[queryLength + 1];

recv(Connection, queryBuffer, queryLength, 0);

queryBuffer[queryLength] = '\0'; // Добавление завершающего нулевого символа

cout << "->" << queryBuffer << endl;

delete[] queryBuffer;

}

void sendFlags(int sock, const Flags& data) {

// Отправка длины пути к файлу

int pathLength = data.file\_path.length();

if (send(sock, reinterpret\_cast<char\*>(&pathLength), sizeof(pathLength), 0) == SOCKET\_ERROR) {

cout << "Ошибка отправки длины пути к файлу.\n";

return;

}

// Отправка пути к файлу

if (send(sock, data.file\_path.c\_str(), pathLength, 0) == SOCKET\_ERROR) {

cout << "Ошибка отправки пути к файлу.\n";

return;

}

// Отправка длины запроса

int queryLength = data.query.length();

if (send(sock, reinterpret\_cast<char\*>(&queryLength), sizeof(queryLength), 0) == SOCKET\_ERROR) {

cout << "Ошибка отправки длины запроса.\n";

return;

}

// Отправка запроса

if (send(sock, data.query.c\_str(), queryLength, 0) == SOCKET\_ERROR) {

cout << "Ошибка отправки запроса.\n";

return;

}

}

int ClientConnection(const Flags& data) {

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if (WSAStartup(DLLVersion, &wsaData) != 0) throw "Ошибка:\n";

SOCKADDR\_IN addr;

int sizeofaddr = sizeof(addr);

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons(6379);

addr.sin\_family = AF\_INET;

Connection = socket(AF\_INET, SOCK\_STREAM, NULL);

if (connect(Connection, (SOCKADDR\*)&addr, sizeof(addr)) != 0) throw "Ошибка: невозможно подключится к серверу.\n";

cout << "Подключен к серверу!\n";

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)ClientHandler, NULL, NULL, NULL);

sendFlags(Connection, data);

Sleep(50);

// Закрытие соединения и освобождение ресурсов

closesocket(Connection);

WSACleanup();

return 0;

}

**Серверная часть:**

**Source.cpp**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <winsock2.h>

#include <thread>

#include "Locale.h"

#include "FileProcess.h"

#include <mutex>

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

using namespace std;

SOCKET Connections[100];

mutex counterMutex;

int Counter = 0;

struct Flags {

string file\_path;

string query;

};

enum Packet {

P\_ChatMessage,

P\_Test

};

bool ProcessPacket(int index, Packet packettype) {

switch (packettype) {

case P\_ChatMessage:

{

int msg\_size;

recv(Connections[index], (char\*)&msg\_size, sizeof(int), NULL);

char\* msg = new char[msg\_size + 1];

msg[msg\_size] = '\0';

recv(Connections[index], msg, msg\_size, NULL);

cout << msg << endl;

delete[] msg;

break;

}

default:

cout << "Unrecognized packet: " << packettype << std::endl;

break;

}

return true;

}

Flags receiveFlags(int clientSocket) {

Flags receivedData;

// Получение длины пути к файлу

int pathLength;

if (recv(clientSocket, reinterpret\_cast<char\*>(&pathLength), sizeof(pathLength), 0) == SOCKET\_ERROR) throw "Ошибка: ошибка чтения длины пути к файлу.\n";

// Получение пути к файлу

char\* pathBuffer = new char[pathLength + 1];

if (recv(clientSocket, pathBuffer, pathLength, 0) == SOCKET\_ERROR) throw "Ошибка: ошибка чтения пути к файлу.\n";

pathBuffer[pathLength] = '\0'; // Добавляем завершающий символ строки

receivedData.file\_path = std::string(pathBuffer);

delete[] pathBuffer;

// Получение длины запроса

int queryLength;

if (recv(clientSocket, reinterpret\_cast<char\*>(&queryLength), sizeof(queryLength), 0) == SOCKET\_ERROR) throw "Ошибка: ошибка чтения длины запроса.\n";

// Получение запроса

char\* queryBuffer = new char[queryLength + 1];

if (recv(clientSocket, queryBuffer, queryLength, 0) == SOCKET\_ERROR) throw "Ошибка: ошибка чтения запроса.\n";

queryBuffer[queryLength] = '\0'; // Добавляем завершающий символ строки

receivedData.query = std::string(queryBuffer);

delete[] queryBuffer;

return receivedData;

}

void ClientHandler(int index) {

Flags receivedData = receiveFlags(Connections[index]);;

cout << "\t->file\_path: " << receivedData.file\_path << "\t query: " << receivedData.query << endl;

{

lock\_guard<mutex> lock(counterMutex);

processCommand(receivedData.file\_path, receivedData.query, Connections[index]);

closesocket(Connections[index]);

Counter++;

}

}

int server() {

// Инициализация Winsock

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if (WSAStartup(DLLVersion, &wsaData) != 0) throw "Ошибка: проблема инициализации Winsock.\n";

SOCKADDR\_IN addr;

int sizeofaddr = sizeof(addr);

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons(6379);

addr.sin\_family = AF\_INET;

SOCKET sListen = socket(AF\_INET, SOCK\_STREAM, NULL);

bind(sListen, (SOCKADDR\*)&addr, sizeof(addr));

listen(sListen, SOMAXCONN);

cout << "->Сервер создан." << endl;

SOCKET newConnection;

for (int i = 0; i < 100; i++) {

newConnection = accept(sListen, (SOCKADDR\*)&addr, &sizeofaddr);

if (newConnection == 0) throw "Ошибка: проблема создания сокета.\n";

else {

cout << "Новое подключение\n";

{

lock\_guard<mutex> lock(counterMutex);

Connections[i] = newConnection;

}

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)ClientHandler, (LPVOID)(i), NULL, NULL);

}

}

return 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

server();

return 0;

}

**FileProcess.h**

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include "Queue.h"

#include "Stack.h"

#include "Set.h"

#include "Locale.h"

#include "HashTable.h"

using namespace std;

void removeLineFromFile(string filePath, string lineToRemove) {

ifstream inputFile(filePath);

if (!inputFile) {

ofstream inputFile(filePath);

if (!inputFile) throw "Не удалось открыть файл.\n";

}

ofstream outputFile("temp.txt");

if (!outputFile) throw "Не удалось создать временный файл.\n";

string line;

while (getline(inputFile, line)) {

if (line != lineToRemove) {

outputFile << line << endl;

}

}

inputFile.close();

outputFile.close();

if (remove(filePath.c\_str()) != 0) throw"Не удалось удалить исходный файл.\n";

if (rename("temp.txt", filePath.c\_str()) != 0) throw "Не удалось переименовать временный файл.\n";

return;

}

void createFile(string filePath) {

ofstream inputFile(filePath);

if (!inputFile) throw "Не удалось создать файл.\n";

inputFile.close();

}

Queue getQueueFromFile(string filePath, string containerName) {

string line;

Queue tempQueue;

ifstream in(filePath);

if (in.is\_open()) {

while (getline(in, line)) {

size\_t colonPos = line.find(':');

if (colonPos != string::npos) {

string queueName = line.substr(0, colonPos - 1);

if (queueName == containerName) {

string elements = line.substr(colonPos + 1);

size\_t bracePos1 = elements.find('{');

size\_t bracePos2 = elements.find('}');

string type = elements.substr(bracePos2 + 2);

if (type == "queue") {

if (bracePos1 != string::npos && bracePos2 != string::npos) {

elements = elements.substr(bracePos1 + 1, bracePos2 - bracePos1 - 1);

size\_t commaPos = elements.find(',');

while (commaPos != string::npos) {

string element = elements.substr(0, commaPos);

tempQueue.push(element);

elements = elements.substr(commaPos + 2);

commaPos = elements.find(',');

}

tempQueue.push(elements);

break;

}

}

}

}

}

}

in.close();

removeLineFromFile(filePath, line);

return tempQueue;

}

Stack getStackFromFile(string filePath, string containerName) {

string line;

Stack tempStack;

ifstream in(filePath);

if (in.is\_open()) {

while (getline(in, line)) {

size\_t colonPos = line.find(':');

if (colonPos != string::npos) {

string queueName = line.substr(0, colonPos - 1);

if (queueName == containerName) {

string elements = line.substr(colonPos + 1);

size\_t bracePos1 = elements.find('{');

size\_t bracePos2 = elements.find('}');

string type = elements.substr(bracePos2 + 2);

if (type == "stack") {

if (bracePos1 != string::npos && bracePos2 != string::npos) {

elements = elements.substr(bracePos1 + 1, bracePos2 - bracePos1 - 1);

size\_t commaPos = elements.find(',');

while (commaPos != string::npos) {

string element = elements.substr(0, commaPos);

tempStack.push(element);

elements = elements.substr(commaPos + 2);

commaPos = elements.find(',');

}

tempStack.push(elements);

break;

}

}

}

}

}

}

in.close();

removeLineFromFile(filePath, line);

return tempStack;

}

Set getSetFromFile(string filePath, string containerName, bool removeline = true) {

string line;

Set tempStack;

ifstream in(filePath);

if (in.is\_open()) {

while (getline(in, line)) {

size\_t colonPos = line.find(':');

if (colonPos != string::npos) {

string queueName = line.substr(0, colonPos - 1);

if (queueName == containerName) {

string elements = line.substr(colonPos + 1);

size\_t bracePos1 = elements.find('{');

size\_t bracePos2 = elements.find('}');

string type = elements.substr(bracePos2 + 2);

if (type == "set") {

if (bracePos1 != string::npos && bracePos2 != string::npos) {

elements = elements.substr(bracePos1 + 1, bracePos2 - bracePos1 - 1);

size\_t commaPos = elements.find(',');

while (commaPos != string::npos) {

string element = elements.substr(0, commaPos);

tempStack.insert(element);

elements = elements.substr(commaPos + 2);

commaPos = elements.find(',');

}

tempStack.insert(elements);

break;

}

}

}

}

}

}

in.close();

if (removeline) {

removeLineFromFile(filePath, line);

}

return tempStack;

}

HashTable getHTFromFile(string filePath, string containerName, bool removeline = true) {

string line;

HashTable tempHT = HashTable(10);

ifstream in(filePath);

if (in.is\_open()) {

while (getline(in, line)) {

size\_t colonPos = line.find(':');

if (colonPos != string::npos) {

string queueName = line.substr(0, colonPos - 1);

if (queueName == containerName) {

string elements = line.substr(colonPos + 1);

size\_t bracePos1 = elements.find('{');

size\_t bracePos2 = elements.find('}');

string type = elements.substr(bracePos2 + 2);

if (type == "hashtable") {

if (bracePos1 != string::npos && bracePos2 != string::npos) {

elements = elements.substr(bracePos1 + 1, bracePos2 - bracePos1 - 1);

size\_t commaPos = elements.find(',');

while (commaPos != string::npos) {

string element = elements.substr(0, commaPos);

size\_t tempcommaPos = element.find(' ');

string tempelement = element.substr(0, tempcommaPos);

element = element.substr(tempcommaPos, string::npos);

element = element.substr(3, string::npos);

tempelement = tempelement.substr(2, string::npos);

tempHT.insert(tempelement, element);

elements = elements.substr(commaPos + 2);

commaPos = elements.find(',');

}

size\_t tempcommaPos = elements.find(' ');

string tempelement = elements.substr(0, tempcommaPos);

elements = elements.substr(tempcommaPos, string::npos);

elements = elements.substr(3, string::npos);

tempelement = tempelement.substr(2, string::npos);

tempHT.insert(tempelement, elements);

break;

}

}

}

}

}

}

in.close();

if (removeline) {

removeLineFromFile(filePath, line);

}

return tempHT;

}

void sendToClient(int Socket, string data) {

int dataLen = sizeof(data);

send(Socket, reinterpret\_cast<char\*>(&dataLen), sizeof(dataLen), 0);

send(Socket, data.c\_str(), dataLen, 0);

}

void sendToClient(int Socket, bool value) {

string data = value ? "true" : "false";

int dataLen = sizeof(data);

send(Socket, reinterpret\_cast<char\*>(&dataLen), sizeof(dataLen), 0);

send(Socket, data.c\_str(), dataLen, 0);

}

void processCommand(string filePath, string query, int Socket) {

size\_t colonPos = query.find(' ');

string command = query.substr(0, colonPos);

string containerName = query.substr(colonPos+1);

containerName = containerName.substr(0, containerName.find(' '));

if (command == "QPUSH") {

Queue tempQueue = getQueueFromFile(filePath, containerName);

string data = query.substr(colonPos + 1);

auto temp = data.find(' ');

if (temp < 100) {

data = data.substr(temp+1);

tempQueue.push(data);

tempQueue.writeContainerToFile(filePath, containerName);

string s = "Удачно";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

else {

string s = "Мало аргументов";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

}

else if (command == "QPOP") {

Queue tempQueue = getQueueFromFile(filePath, containerName);

string s = tempQueue.pop();

sendToClient(Socket, s);

cout << "\t->Удачно" << endl;

tempQueue.writeContainerToFile(filePath, containerName);

}

else if (command == "SPUSH") {

Stack tempStack = getStackFromFile(filePath, containerName);

string data = query.substr(colonPos + 1);

data = data.substr(data.find(' ') + 1);

tempStack.push(data);

tempStack.writeContainerToFile(filePath, containerName);

string s = "Удачно";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

else if (command == "SPOP") {

Stack tempStack = getStackFromFile(filePath, containerName);

sendToClient(Socket, tempStack.pop());

cout << "\t->Удачно" << endl;

tempStack.writeContainerToFile(filePath, containerName);

}

else if (command == "SADD") {

string s;

Set tempSet = getSetFromFile(filePath, containerName);

string data = query.substr(colonPos + 1);

auto temp = data.find(' ');

if(temp < 100){

data = data.substr(temp + 1);

s = tempSet.insert(data);

cout << "\t->" << s << endl;

sendToClient(Socket, s);

tempSet.writeContainerToFile(filePath, containerName);

}

else {

s = "Мало аргументов";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

}

else if (command == "SREM") {

Set tempSet = getSetFromFile(filePath, containerName);

string data = query.substr(colonPos + 1);

auto temp = data.find(' ');

if (temp < 100) {

data = data.substr(temp + 1);

string s = tempSet.remove(data);

if (s == "True") {

tempSet.writeContainerToFile(filePath, containerName);

cout << "\t->Удачно" << endl;

}

else {

sendToClient(Socket, s);

}

}

else {

string s = "Мало аргументов";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

}

else if (command == "SISMEMBER") {

Set tempSet = getSetFromFile(filePath, containerName, false);

string data = query.substr(colonPos + 1);

auto temp = data.find(' ');

if (temp < 100) {

data = data.substr(temp + 1);

cout << "\t->Удачно" << endl;

sendToClient(Socket, tempSet.contains(data));

}

else {

string s = "Мало аргументов";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

}

else if (command == "HSET") {

HashTable tempHashTable = getHTFromFile(filePath, containerName);

string key = query.substr(colonPos + 1);

key = key.substr(key.find(' ') + 1);

auto temp = key.find(' ');

if (temp < 100) {

string value = key.substr(temp + 1);;

key.erase(temp, value.length() + 1);

string s = tempHashTable.insert(key, value);

sendToClient(Socket, s);

cout << "\t->" << s << endl;

tempHashTable.writeContainerToFile(filePath, containerName);

}

else {

string s = "Мало аргументов";

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

}

else if (command == "HDEL") {

HashTable tempHashTable = getHTFromFile(filePath, containerName);

string key = query.substr(colonPos + 1);

key = key.substr(key.find(' ') + 1);

string s = tempHashTable.remove(key);

cout << "\t->" << s << endl;

sendToClient(Socket, s);

tempHashTable.writeContainerToFile(filePath, containerName);

}

else if (command == "HGET") {

HashTable tempHashTable = getHTFromFile(filePath, containerName, false);

string key = query.substr(colonPos + 1);

key = key.substr(key.find(' ') + 1);

string s = tempHashTable.get(key);

cout << "\t->" << s << endl;

sendToClient(Socket, s);

}

else {

string s = "Неверная команда";

cout << s << endl;

sendToClient(Socket, s);

}

}

**Set.h**#pragma once

#include <iostream>

#include <string>

using namespace std;

struct Node {

string data;

Node\* next;

Node(const string& value) : data(value), next(nullptr) {}

};

struct Set {

Node\* head;

Set() : head(nullptr) {}

string insert(const string& value) {

if (contains(value))

return "В set уже есть данный элемент.";

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

return "Удачно";

}

string remove(const string& value) {

if (isEmpty())

return "Set пуст\0";

if (head->data == value) {

Node\* temp = head;

head = head->next;

delete temp;

return "Удачно";

}

Node\* current = head;

while (current->next != nullptr) {

if (current->next->data == value) {

Node\* temp = current->next;

current->next = current->next->next;

delete temp;

return "Удачно";

}

current = current->next;

}

}

bool contains(const string& value) const {

Node\* current = head;

while (current != nullptr) {

if (current->data == value)

return true;

current = current->next;

}

return false;

}

bool isEmpty() const {

return (head == nullptr);

}

void printSet() {

if (isEmpty()) {

cout << "Set is empty." << endl;

return;

}

Node\* current = head;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void writeContainerToFile(string filePath, string containerName) {

string temp;

ofstream out(filePath, ios::app);

if (out.is\_open()) {

temp = containerName + " : {";

Node\* current = head;

if (current == nullptr) {

out << "" << endl;

out.close();

return;

}

while (current != nullptr) {

temp += current->data + ", ";

current = current->next;

}

temp.pop\_back();

temp.pop\_back();

temp += "} set";

out << temp << endl;

}

out.close();

}

};

**HashTable.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

using namespace std;

struct NodeHT {

string key;

string value;

NodeHT(const string& k, const string& v) : key(k), value(v) {}

};

struct HashTable {

vector<NodeHT\*> table;

int size;

HashTable(int initialSize) : size(initialSize) {

table.resize(size, nullptr);

}

int hashFunction(const string& key) const {

int hash = 0;

for (char c : key) {

hash += c;

}

return hash % size;

}

int secondHashFunction(const string& key) const {

int hash = 0;

for (char c : key) {

hash += c;

}

return 1 + (hash % (size - 1));

}

string insert(const string& key, const string& value) {

int index = hashFunction(key);

int step = secondHashFunction(key);

while (table[index] != nullptr) {

if (table[index]->key == key) {

string s = "Ключ уже существует.";

return s;

}

index = (index + step) % size;

}

table[index] = new NodeHT(key, value);

return "Удачно";

}

string get(const string& key) const {

int index = hashFunction(key);

int step = secondHashFunction(key);

while (table[index] != nullptr && table[index]->key != key) {

index = (index + step) % size;

}

if (table[index] != nullptr && table[index]->key == key) {

return table[index]->value;

}

return "Ключ не найден";

}

string remove(const string& key) {

int index = hashFunction(key);

int step = secondHashFunction(key);

while (table[index] != nullptr && table[index]->key != key) {

index = (index + step) % size;

}

if (table[index] != nullptr && table[index]->key == key) {

delete table[index];

table[index] = nullptr;

return "Удачно";

}

return "Хэштаблица пуста";

}

void display() {

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

if (table[i] != NULL) {

cout << "ключ: " << table[i]->key << "\t значение: " << table[i]->value << endl; // Выводим ключ и значение записи, если запись существует и не удалена

}

}

}

void writeContainerToFile(string filePath, string containerName) {

string temp;

int error = 0;

ofstream out(filePath, ios::app);

if (out.is\_open()) {

temp = containerName + " : {";

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

if (table[i] != NULL) {

temp += "k:" + table[i]->key + " v:" + table[i]->value + ", ";

}

else {

error++;

}

}

if (error == size) {

out << "" << endl;

out.close();

return;

}

temp.pop\_back();

temp.pop\_back();

temp += "} hashtable";

out << temp << endl;

}

out.close();

}

};

**Queue.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct NodeQ {

string data;

NodeQ\* next;

NodeQ(const string& value) : data(value), next(nullptr) {}

};

struct Queue {

NodeQ\* front;

NodeQ\* rear;

Queue() : front(nullptr), rear(nullptr) {}

void push(const string& value) {

NodeQ\* newNodeQ = new NodeQ(value);

if (isEmpty()) {

front = rear = newNodeQ;

}

else {

rear->next = newNodeQ;

rear = newNodeQ;

}

}

string pop() {

if (isEmpty()) {

string s = "queue is empty";

return s;

}

NodeQ\* temp = front;

front = front->next;

string poppedValue = temp->data;

delete temp;

if (front == nullptr) {

rear = nullptr;

}

return poppedValue;

}

string getFront() const {

if (isEmpty()) {

return "Queue is empty.";

}

return front->data;

}

bool isEmpty() const {

return (front == nullptr);

}

void printQueue() {

if (isEmpty()) {

cout << "Queue is empty." << endl;

return;

}

NodeQ\* current = front;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void writeContainerToFile(string filePath, string containerName) {

string temp;

ofstream out(filePath, ios::app);

if (out.is\_open()) {

temp = containerName + " : {";

NodeQ\* current = front;

if (current == nullptr) {

out << "" << endl;

out.close();

return;

}

while (current != nullptr) {

temp += current->data + ", ";

current = current->next;

}

temp.pop\_back();

temp.pop\_back();

temp += "} queue";

out << temp << endl;

}

out.close();

}

};

**Stack.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct NodeS {

string data;

NodeS\* next;

NodeS(const string& value) : data(value), next(nullptr) {}

};

struct Stack {

NodeS\* top;

Stack() : top(nullptr) {}

void push(const string& value) {

NodeS\* newNodeS = new NodeS(value);

newNodeS->next = top;

top = newNodeS;

}

string pop() {

if (isEmpty()) {

return "Stack is empty.";

}

NodeS\* temp = top;

top = top->next;

string poppedValue = temp->data;

delete temp;

return poppedValue;

}

string getTop() const {

if (isEmpty()) {

return "Stack is empty.";

}

return top->data;

}

bool isEmpty() const {

return (top == nullptr);

}

void printStack() {

if (isEmpty()) {

cout << "Stack is empty." << endl;

return;

}

NodeS\* current = top;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void writeContainerToFile(string filePath, string containerName) {

string temp;

ofstream out(filePath, ios::app);

if (out.is\_open()) {

temp = containerName + " : {";

NodeS\* current = top;

if (current == nullptr) {

out << "" << endl;

out.close();

return;

}

while (current != nullptr) {

temp += current->data + ", ";

current = current->next;

}

temp.pop\_back();

temp.pop\_back();

temp += "} stack";

out << temp << endl;

}

out.close();

}

};

**Пример работы программы:**

Переходим в каталог к client.exe  
  
Client --file file.data --query 'QPUSH name\_queue value'

Client --file file.data --query 'QPOP name\_queue'

Client --file file.data --query 'SPUSH mvq value'

Client --file file.data --query 'SPOP mvq'

Client --file file.data --query 'SADD myset value'

Client --file file.data --query 'SREM myset value'

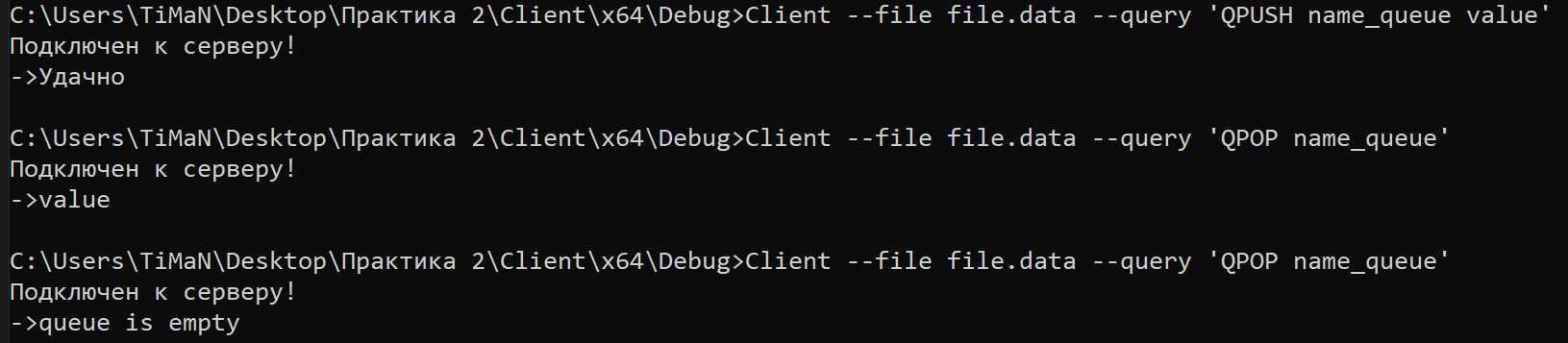
Client --file file.data --query 'SISMEMBER myset value'

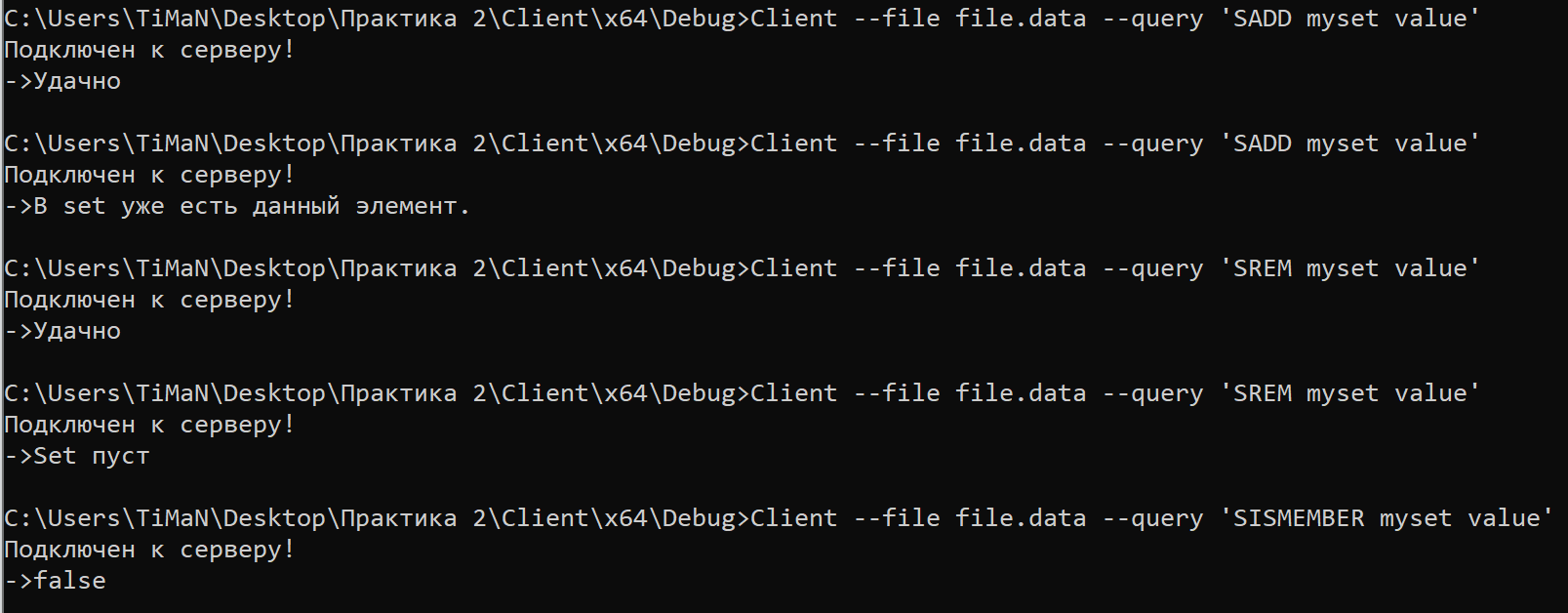
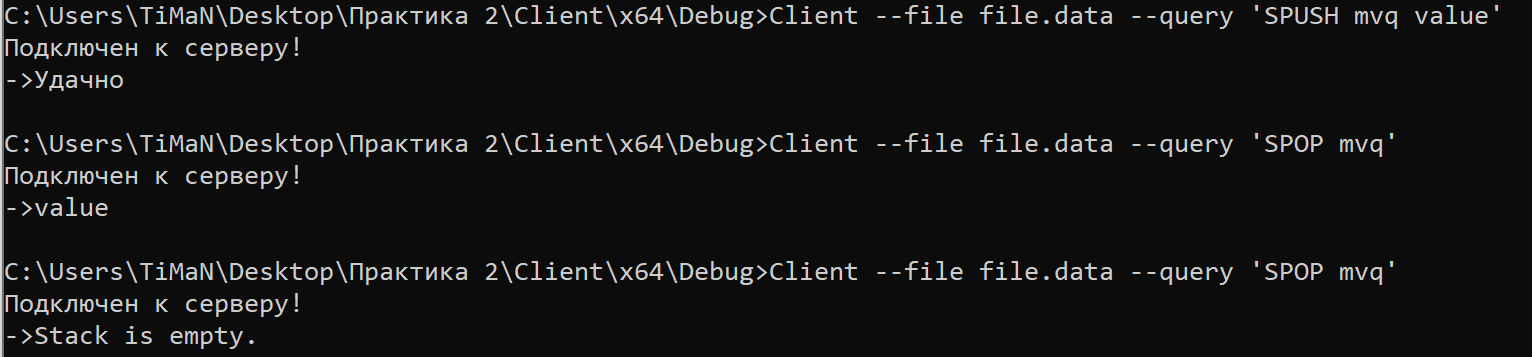
Client --file file.data --query 'HSET myht key value'

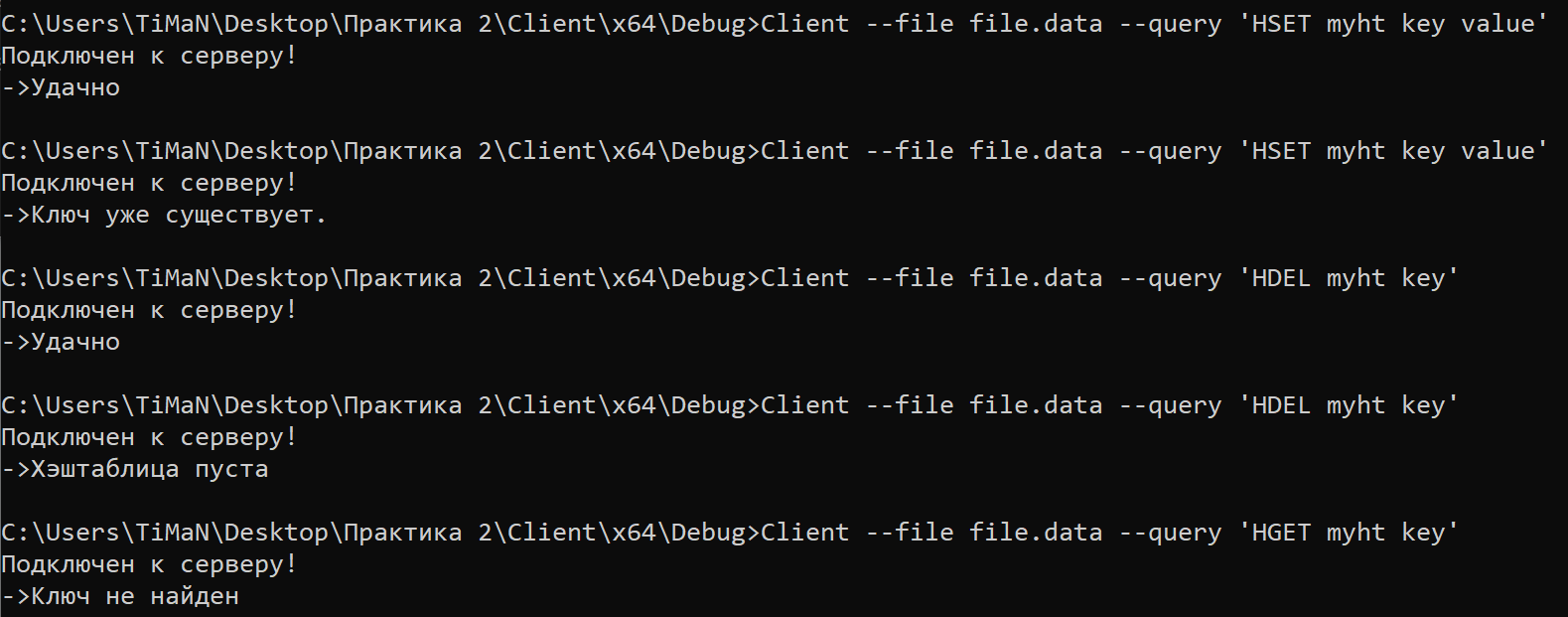
Client --file file.data --query 'HDEL myht key'

Client --file file.data --query 'HGET myht key'

Вывод на клиенте







Вывод на сервере

