**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные системы»

**Лабораторная работа № 6**

Управлении серверами сообщений

Студент: Ваньков Д. А.

Группа: 80-207Б

Преподаватель: Соколов А. А.

Дата: 18.12.2018

Оценка:

Москва, 2018

**1. Постановка задачи**

Вариант 17:

### **Конфигурации для клиентов и серверов. Внутреннее хранилище сервера:**

2.Вектор

### **Тип ключа клиента**

### 1.Целое число 32 бита

### **Дополнительные возможности сервера**

1.Сохранение данных о счетах клиентов при завершении работы сервера и возобновлении.

Операционная система: Unix.

**Цель работы:**

Целью является приобретение практических навыков в:

* + Управлении серверами сообщений (№6)
  + Применение отложенных вычислений (№7)
  + Интеграция программных систем друг с другом (№8)

**Задание**

Реализовать клиент-серверную систему по асинхронной обработке запросов. Необходимо составить программы сервера и клиента. При запуске сервер и клиент должны быть настраиваемы, то есть должна быть возможность поднятия на одной ЭВМ нескольких серверов по обработке данных и нескольких клиентов, которые к ним относятся. Все общение между процессами сервера и клиентов должно осуществляться через сервер сообщений.

Серверное приложение – банк. Клиентское приложение клиент банка. Клиент может отправить какую-то денежную сумму в банк на хранение. Клиент также может запросить из банка произвольную сумму. Клиенты могут посылать суммы на счета других клиентов. Запросить собственный счет. При снятии должна производиться проверка на то, что у клиента достаточно денег для снятия денежных средств. Идентификатор клиента задается во время запуска клиентского приложения, как и адрес банка. Считать, что идентификаторы при запуске клиентов будут уникальными.

**2. Решение задачи**

Используемые системные вызовы:

* **void exit(int status);** – функция выходит из процесса с заданным статусом.
* **int zmq connect(void \*socket, const char \*endpoint);** – подключает socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.
* **int zmq bind(void \*socket, const char \*endpoint);** – присоединяет socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.
* **void \*zmq socket(void \*context, int type);** – создает сокет типа type из контекста context.
* **int zmq msg send(zmq msg t \*msg, void \*socket, int flags);** – отправляет сообщение msg в socket с параметрами flags, возвращает количество отправленных байт, в случае ошибки возвращает -1.
* **int zmq msg init(zmq msg t \*msg)** – инициализирует сообщение msg как пустой объект.
* **int zmq msg recv(zmq msg t \*msg, void \*socket, int flags);** – получает сообщение из socket в msg с параметрами flags, возвращает количество полученных байт, в случае ошибки возвращает -1.
* **int zmq msg close(zmq msg t \*msg)** – очищает содержимое msg, аналог free для сообщений zmq, возвращает 0 в случае успеха и -1 в случае неудачи.
* **int zmq close(void \*socket);** – закрывает сокет socket, возвращает 0 в случае успеха и -1 в случае неудачи.
* **int zmq ctx destroy(void \*context);** – разрушает контекст context, блокирует доступ всем операциям кроме zmq close, все сообщения в сокетах либо физически отправлены, либо "висят".

**3. Руководство по использованию программы**

Компиляция и запуск программного кода в *Ubuntu* :

*make*

*./server*

*./client*

***Тестирование***

hvidsmen@hvidsmen-X505BA:~/Рабочий стол/lr6$ ./server

This is bank`s server.

Try to load DataBase from DataBase

Start scaning data...

The id: 10 The balance: 250

The id: 11 The balance: 200

The id: 12 The balance: 100

The id: 13 The balance: 800

Data succesfully scaned.

DataBase loaded sucessfully.

Enter the adress of bank: 4040

Check account ballance ID: 13

Client ballance is: 800

Client ID: 10 Client ballance: 250

Client ID: 11 Client ballance: 200

Client ID: 12 Client ballance: 100

Client ID: 13 Client ballance: 800

hvidsmen@hvidsmen-X505BA:~/Рабочий стол/lr6$ ./client

Registration...

Enter client's login: 13

Enter bank's adress: 4040

tcp://localhost:4040

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

>> Press 1) Check balance

>> Press 2) Put money on your account balance

>> Press 3) Get money from your account balance

>> Press 4) Send money to another account

>> Press 5) Get help

>> Press 6) Exit the bank

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

1

Your account ballance is: 800

2

Enter a number of money to deposit: 3

Operation was completed successfully.

3

Enter a number of money to withdrawals: 100

Operation was completed successfully.

1

Your account ballance is: 703

6

hvidsmen@hvidsmen-X505BA:~/Рабочий стол/lr6$ cat DataBase

10 250

11 200

12 100

13 703

hvidsmen@hvidsmen-X505BA:~/Рабочий стол/lr6$

chappybunny@chappybunny:~/Загрузки/lr6$ ./server

This is bank`s server.

Try to load DataBase from DataBase

Start scaning data...

The id: 10 The balance: 350

The id: 11 The balance: 200

The id: 12 The balance: 100

The id: 13 The balance: 697

Data succesfully scaned.

DataBase loaded sucessfully.

Enter the adress of bank: 4040

Send money from account ID: 13 to account ID: 10

Successfully.

Client ID: 10 Client ballance: 550

Client ID: 11 Client ballance: 200

Client ID: 12 Client ballance: 100

Client ID: 13 Client ballance: 497

chappybunny@chappybunny:~/Загрузки/lr6$

chappybunny@chappybunny:~/Загрузки/lr6$ ./client

Registration...

Enter client's login: 13

Enter bank's adress: 4040

tcp://localhost:4040

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

>> Press 1) Check balance

>> Press 2) Put money on your account balance

>> Press 3) Get money from your account balance

>> Press 4) Send money to another account

>> Press 5) Get help

>> Press 6) Exit the bank

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

4

Enter receiver`s client ID: 10

Enter a number of money to transfer: 200

Operation was completed successfully.

1

Your account ballance is: 497

6

*chappybunny@chappybunny:~/Загрузки/lr6$*

**4. Листинг программы**

**Makefile**

all: client server

client:

gcc -std=c99 -pthread -Wall -D\_POSIX\_SOURCE=600 -D\_XOPEN\_SOURCE=600 client.c -o client -L/usr/local/lib -lzmq -lm

server:

gcc -std=c99 -pthread -Wall -D\_POSIX\_SOURCE=600 -D\_XOPEN\_SOURCE=600 bank.c server.c -o server -L/usr/local/lib -lzmq -lm

clean:

rm -f \*.o client server

**bank.h**

#ifndef \_BANK\_H\_

#define \_BANK\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <inttypes.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <stropts.h>

#include <signal.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <fcntl.h>

#define STR\_SIZE 256

#define SUCCESS 1

#define NOT\_MEMORY -1

#define NOT\_ENOUGH\_MONEY -2

#define NOT\_CLIENT -3

#define RECEIVER\_NOT\_CLIENT -4

typedef int32\_t ID;

typedef struct \_msg {

ID client;

int sum;

int action;

ID receiverClient;

void \*requester;

char message[STR\_SIZE];

char fileName[STR\_SIZE];

} MsgData;

typedef struct \_client {

ID client;

int sum;

} \*Client;

typedef struct \_clientDB {

Client clients;

uint32\_t size;

uint32\_t freespace;

} \*ClientDB;

ClientDB ClientDBCreate(void);

void ClientAdd(ClientDB cDB, ID client);

void ClientDBPrint(ClientDB cDB);

Client ClientFind(ClientDB cDB, int clientint);

void ClientDBDestroy(ClientDB \*cDB);

void ClientDBSave(ClientDB cDB, char \*fileName);

void ClientDBLoad(ClientDB cDB, char \*fileName);

void ClientAccIncrease(Client cl, int sum);

void ClientAccDecrease(Client cl, int sum);

int ClientHasEnoughMoney(Client cl, int sum);

void MoneyPut(int Clientint, int sum, ClientDB cDB);

int MoneyGet(int Clientint, int sum, ClientDB cDB);

int MoneySend(ID clientSender, ID clientReceiver, int sum, ClientDB cDB);

int CheckAccount(ID client, ClientDB cDB);

#endif

**bank.c**

#include "bank.h"

ClientDB ClientDBCreate(void)

{

ClientDB cDB = (ClientDB) malloc(sizeof(\*cDB));

if (!cDB) {

fprintf(stderr, "ERROR: no memory\n");

exit(NOT\_MEMORY);

}

cDB->clients = (Client) malloc(sizeof(\*(cDB->clients)));

cDB->size = 0;

cDB->freespace = 1;

return cDB;

}

void DBResize(ClientDB cDB)

{

cDB->clients = realloc(cDB->clients, 2 \* cDB->size \* sizeof(\*(cDB->clients)));

if (!cDB->clients) {

fprintf(stderr, "ERROR: no memory\n");

exit(NOT\_MEMORY);

}

cDB->freespace = cDB->size;

}

void ClientAdd(ClientDB cDB, ID client)

{

if (!cDB->freespace) {

DBResize(cDB);

}

cDB->clients[cDB->size].client = client;

cDB->clients[cDB->size].sum = 0;

cDB->size++;

cDB->freespace--;

}

void ClientDBPrint(ClientDB cDB)

{

if (cDB) {

for (uint32\_t i = 0; i < cDB->size; ++i) {

printf("Client ID: %d\t", cDB->clients[i].client);

printf("Client ballance: %d\n", cDB->clients[i].sum);

}

}

}

Client ClientFind(ClientDB cDB, int clientint)

{

if (cDB) {

for (uint32\_t i = 0; i < cDB->size; ++i) {

if (cDB->clients[i].client == clientint) {

return &(cDB->clients[i]);

}

}

}

return NULL;

}

void ClientDBDestroy(ClientDB \*cDB)

{

free((\*cDB)->clients);

(\*cDB)->clients = NULL;

free(\*cDB);

\*cDB = NULL;

}

void ClientDBSave(ClientDB cDB, char \*fileName)

{

int file = open(fileName, O\_RDWR|O\_TRUNC|O\_CREAT, \_\_S\_IREAD|\_\_S\_IWRITE);

if (file < 0) {

printf("Cannot open file\n");

} else {

dup2(file, 1);

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

printf("%d ",cDB->clients[i].client);

printf("%d\n",cDB->clients[i].sum);

}

//dup2(1,1);

close(file);

}

}

/\*void ClientDBLoad(ClientDB cDB, char \*fileName)

{

FILE \* file = fopen(fileName, "rb");

if (!file) {

printf("Cannot open file\n");

} else {

int id = 0;

int balance = 0;

printf("Start scaning data...\n");

int i = 0;

while(fscanf(file, "%d%d", &id, &balance) != EOF) {

cDB->clients[i].client = id;

cDB->clients[i].sum = balance;

printf("The id: %d ", cDB->clients[i].client);

printf("The balance: %d\n", cDB->clients[i].sum);

printf("Scanned %d\n", i);

MoneyPut(cDB->clients[i].client, cDB->clients[i].sum, cDB);

++i;

}

printf("Data succesfully scaned.");

}

fclose(file);

}

\*/

void ClientDBLoad(ClientDB cDB, char \*fileName)

{

int file = open(fileName, O\_RDONLY|O\_CREAT, \_\_S\_IREAD|\_\_S\_IWRITE);

if (file < 0) {

printf("Cannot open file\n");

} else {

int id = 0;

int balance = 0;

printf("Start scaning data...\n");

int i = 0;

char symb;

while(read(file, &symb, sizeof(char))) {

while(symb != ' ') {

id = id \* 10 + (symb - '0');

read(file, &symb, sizeof(char));

}

printf("The id: %d ", id);

cDB->clients[i].client = id;

read(file, &symb, sizeof(char));

while(symb != '\n') {

balance = balance \* 10 + (symb - '0');

read(file, &symb, sizeof(char));

}

printf("The balance: %d\n", balance);

cDB->clients[i].sum = balance;

MoneyPut(cDB->clients[i].client, cDB->clients[i].sum, cDB);

id = 0;

balance = 0;

++i;

}

printf("Data succesfully scaned.\n");

close(file);

}

}

void ClientAccIncrease(Client cl, int sum)

{

cl->sum += sum;

}

void ClientAccDecrease(Client cl, int sum)

{

cl->sum -= sum;

}

int ClientHasEnoughMoney(Client cl, int sum)

{

return cl->sum >= sum;

}

void MoneyPut(int clientint, int sum, ClientDB cDB)

{

Client cl = ClientFind(cDB, clientint);

if (cl) {

ClientAccIncrease(cl, sum);

} else {

ClientAdd(cDB, clientint);

cl = ClientFind(cDB, clientint);

ClientAccIncrease(cl, sum);

}

}

int MoneyGet(int clientint, int sum, ClientDB cDB)

{

Client cl = ClientFind(cDB, clientint);

if (!cl) {

return NOT\_CLIENT;

}

if (ClientHasEnoughMoney(cl, sum)) {

ClientAccDecrease(cl, sum);

return SUCCESS;

} else {

return NOT\_ENOUGH\_MONEY;

}

}

int MoneySend(ID clientSender, ID clientReceiver, int sum, ClientDB cDB)

{

Client clSender = ClientFind(cDB, clientSender);

if (!clSender) {

return NOT\_CLIENT;

}

Client clReceiver = ClientFind(cDB, clientReceiver);

if (!clReceiver) {

return RECEIVER\_NOT\_CLIENT;

}

if (ClientHasEnoughMoney(clSender, sum)) {

ClientAccDecrease(clSender, sum);

ClientAccIncrease(clReceiver, sum);

return SUCCESS;

} else {

return NOT\_ENOUGH\_MONEY;

}

}

int CheckAccount(ID client, ClientDB cDB)

{

Client cl = ClientFind(cDB, client);

if (!cl) {

return NOT\_CLIENT;

}

return cl->sum;

}

**client.c**

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

#include <string.h>

#include <pthread.h>

#include "zmq.h"

#include "bank.h"

void menuAdmin()

{

printf(">> Press 1) Stop server \n");

printf(">> Press 2) Run server \n");

printf(">> Press 3) Exit \n");

}

void menuUser()

{

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf(">> Press 1) Check balance\n");

printf(">> Press 2) Put money on your account balance\n");

printf(">> Press 3) Get money from your account balance\n");

printf(">> Press 4) Send money to another account\n");

printf(">> Press 5) Get help\n");

printf(">> Press 6) Exit the bank\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

void \*SendRecv(void \*arg)

{

MsgData \*md = (MsgData \*) arg;

zmq\_msg\_t message;

zmq\_msg\_init\_size(&message, sizeof(MsgData));

memcpy(zmq\_msg\_data(&message), md, sizeof(MsgData));

zmq\_msg\_send(&message, md->requester, 0);

zmq\_msg\_close(&message);

zmq\_msg\_init(&message);

zmq\_msg\_recv(&message, md->requester, 0);

md = (MsgData \*) zmq\_msg\_data(&message);

printf("%s\n", md->message);

zmq\_msg\_close(&message);

pthread\_exit(NULL);

return 0;

}

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

{

void \*context = zmq\_ctx\_new();

int admin = 0;

ID client, bank;

if (argc == 2 && !strcmp(argv[1], "admin")) {

admin = 1;

} else {

printf("Registration...\n");

printf("Enter client's login: ");

scanf("%d", &client);

}

char adress[25];

printf("Enter bank's adress: ");

scanf("%d", &bank);

sprintf(adress, "%s%d", "tcp://localhost:", bank);

printf("tcp://localhost:%d \n", bank);

void \*sendSocket = zmq\_socket(context, ZMQ\_REQ);

zmq\_connect(sendSocket, adress);

if (admin) {

int act = 0;

menuAdmin();

do {

scanf("%d", &act);

MsgData md;

md.action = act + 10;

switch (act) {

case 1: {

pthread\_t th;

md.requester = sendSocket;

pthread\_create(&th, NULL, SendRecv, &md);

pthread\_detach(th);

break;

}

case 2: {

pthread\_t th;

md.requester = sendSocket;

pthread\_create(&th, NULL, SendRecv, &md);

pthread\_detach(th);

break;

}

case 3:

break;

default: {

printf("Inccorect command\n");

break;

}

}

} while (act != 3);

} else {

int act = 0, sum = 0;

menuUser();

do {

scanf("%d", &act);

MsgData md;

md.action = act;

md.client = client;

switch (act) {

case 1: {

pthread\_t th;

md.requester = sendSocket;

pthread\_create(&th, NULL, SendRecv, &md);

pthread\_detach(th);

break;

}

case 2: {

printf("Enter a number of money to deposit: ");

scanf("%d", &sum);

while (sum < 0) {

printf("Inccorect number of money. Plese try again: ");

scanf("%d", &sum);

}

md.sum = sum;

pthread\_t th;

md.requester = sendSocket;

pthread\_create(&th, NULL, SendRecv, &md);

pthread\_detach(th);

break;

}

case 3: {

printf("Enter a number of money to withdrawals: ");

scanf("%d", &sum);

while (sum < 0) {

printf("Inccorect number of money. Plese try again: ");

scanf("%d", &sum);

}

md.sum = sum;

pthread\_t th;

md.requester = sendSocket;

pthread\_create(&th, NULL, SendRecv, &md);

pthread\_detach(th);

break;

}

case 4: {

int receiverClient;

printf("Enter receiver`s client ID: ");

scanf("%d", &receiverClient);

printf("Enter a number of money to transfer: ");

scanf("%d", &sum);

while (sum < 0) {

printf("Inccorect number of money. Plese try again: ");

scanf("%d", &sum);

}

md.sum = sum;

md.receiverClient = receiverClient;

pthread\_t th;

md.requester = sendSocket;

pthread\_create(&th, NULL, SendRecv, &md);

pthread\_detach(th);

break;

}

case 5: {

menuUser();

break;

}

case 6:

break;

default: {

printf("Command not found.\n");

break;

}

}

} while (act != 6);

}

zmq\_close(sendSocket);

zmq\_ctx\_destroy(context);

return 0;

}

**server.c**

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <stropts.h>

#include <signal.h>

#include "zmq.h"

#include "bank.h"

volatile sig\_atomic\_t flag = 0;

void block\_func(int sig)

{

if (!flag) {

flag = 1;

} else {

exit(0);

}

}

void unblock\_func(int sig)

{

flag = 0;

}

int main(void)

{

int code;

ClientDB clientBase = ClientDBCreate();

void \*context = zmq\_ctx\_new();

void \*responsSocket = zmq\_socket(context, ZMQ\_REP);

char adress[25];

printf("This is bank`s server.\n");

char FileName[STR\_SIZE];

strcpy(FileName, "DataBase");

printf("Try to load DataBase from %s \n", FileName);

FILE \*file = fopen(FileName, "rb");

if (file == NULL) {

printf("Failed to load DataBase.\n");

printf("Try to create new DataBase.\n");

printf("DataBase create sucessfully.\n");

} else {

ClientDBLoad(clientBase, FileName);

printf("DataBase loaded sucessfully.\n");

fclose(file);

}

printf("Enter the adress of bank: ");

ID bank;

scanf("%d", &bank);

sprintf(adress, "%s%d", "tcp://\*:", bank);

zmq\_bind(responsSocket, adress);

while (1) {

signal(SIGINT, block\_func);

signal(SIGTSTP, unblock\_func);

if (!flag) {

zmq\_msg\_t message;

zmq\_msg\_init(&message);

zmq\_msg\_recv(&message, responsSocket, 0);

MsgData \*md = (MsgData \*) zmq\_msg\_data(&message);

zmq\_msg\_close(&message);

char info[STR\_SIZE];

switch (md->action) {

case 1: {

printf("Check account ballance ID: %d\n", md->client);

code = CheckAccount(md->client, clientBase);

if (code == NOT\_CLIENT) {

printf("Not bank client.\n");

strcpy(info, "You aren't client of bank.\0");

} else {

printf("Client ballance is: %d\n", code);

ClientDBPrint(clientBase);

sprintf(info, "%s%d%c", "Your account ballance is: ", code, '\0');

}

memcpy(md->message, info, strlen(info) + 1);

zmq\_msg\_init\_size(&message, sizeof(MsgData));

memcpy(zmq\_msg\_data(&message), md, sizeof(MsgData));

zmq\_msg\_send(&message, responsSocket, 0);

zmq\_msg\_close(&message);

break;

}

case 2: {

printf("Put money on the account ballance ID: %d\n", md->client);

MoneyPut(md->client, md->sum, clientBase);

ClientDBPrint(clientBase);

strcpy(info, "Operation was completed successfully.\0");

memcpy(md->message, info, strlen(info) + 1);

zmq\_msg\_init\_size(&message, sizeof(MsgData));

memcpy(zmq\_msg\_data(&message), md, sizeof(MsgData));

zmq\_msg\_send(&message, responsSocket, 0);

zmq\_msg\_close(&message);

break;

}

case 3: {

printf("Get money from the account ballance ID: %d\n", md->client);

code = MoneyGet(md->client, md->sum, clientBase);

if (code == SUCCESS) {

printf("Successfully.\n");

ClientDBPrint(clientBase);

strcpy(info, "Operation was completed successfully.\0");

} else if (code == NOT\_ENOUGH\_MONEY) {

printf("Not enough money.\n");

strcpy(info, "You not enough money to withdrawals.\0");

} else {

printf("Not bank client.\n");

strcpy(info, "You aren't a client of bank.\0");

}

memcpy(md->message, info, strlen(info) + 1);

zmq\_msg\_init\_size(&message, sizeof(MsgData));

memcpy(zmq\_msg\_data(&message), md, sizeof(MsgData));

zmq\_msg\_send(&message, responsSocket, 0);

zmq\_msg\_close(&message);

break;

}

case 4: {

printf("Send money from account ID: %d to account ID: %d\n", md->client, md->receiverClient);

code = MoneySend(md->client, md->receiverClient, md->sum, clientBase);

if (code == SUCCESS) {

printf("Successfully.\n");

ClientDBPrint(clientBase);

strcpy(info, "Operation was completed successfully.\0");

} else if (code == NOT\_ENOUGH\_MONEY) {

printf("Not enought money to transfer.\n");

strcpy(info, "You not enough money to transfer.");

} else if (code == RECEIVER\_NOT\_CLIENT) {

printf("Receiver not bank client\n");

strcpy(info, "Receiver account is not a client of bank.\0");

}

memcpy(md->message, info, strlen(info) + 1);

zmq\_msg\_init\_size(&message, sizeof(MsgData));

memcpy(zmq\_msg\_data(&message), md, sizeof(MsgData));

zmq\_msg\_send(&message, responsSocket, 0);

zmq\_msg\_close(&message);

break;

}

}

ClientDBSave(clientBase, FileName);

zmq\_msg\_close(&message);

}

}

zmq\_close(responsSocket);

zmq\_ctx\_destroy(context);

ClientDBDestroy(&clientBase);

}

**5.Вывод**

Для меня эта была самая интересная и самая сложная лабораторная работа за семестр. Удалось применить полученные ранее знания, а также разобраться с работой серверов сообщений. Сервер работает с помощью потоков. Клиент посылает сообщение серверу, а тот в свою очередь его обрабатывает и отправляет результат клиенту. Самым сложным оказалось получение базы данных пользователей и последующая ее загрузка, однако я справился с эти заданием и даже реализовал 2 способа считывания или загрузки базы данных.