

Сети ЭВМ и телекоммуникации

В. Е. Бушин

28 декабря 2015 г.

Глава 1

Задание

Разработать приложение–клиент и приложение–сервер платежной системы. Участники платежной системы имеют электронные кошельки. Электронный кошелек имеет уникальный номер. При регистрации пользователя в платежной системе на его счет зачисляется определенная сумма. Пользователя платежной системы могут осуществлять платежи друг другу через приложение–сервер.

1.1 Функциональные требования

Серверное приложение должно реализовывать следующие функции:

1. Прослушивание определенного порта
2. Обработка запросов на подключение по этому порту от клиентов платежной системы
3. Поддержка одновременной работы нескольких клиентов платежной системы через механизм нитей
4. Прием запросов от клиента на регистрацию пользователя, передачу списка электронных кошельков пользователей платежной системы, осуществление платежей одного пользователя другому, проверка состояния счета кошелька
5. Осуществление добавления пользователя в платежную систему, хранение и изменение состояния электронных кошельков в зависимости от платежей пользователей

6. Передача запросов на платежи от одного пользователя другому, подтверждений платежей, номера нового кошелька при регистрации пользователя, списка электронных кошельков
7. Обработка запроса на отключение клиента
8. Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1. Установление соединения с сервером
2. Передача запросов на передачу списка электронных кошельков пользователей платежной системы, платежи одного пользователя другому, проверку состояния счета кошелька
3. Получение от сервера запросов на платеж от другого пользователя, результатов платежа
4. Разрыв соединения
5. Обработка ситуации отключения клиента сервером

Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени сервера платежной системы и номера порта, используемого сервером.

1.2 Нефункциональные требования

При подключении клиента происходит процесс идентификации с помощью логина или процесс регистрации нового пользователя. Пользователь не может вводить команды (кроме команд, связанных с идентификацией), пока идентификация не будет успешной. Один и тот же пользователь одновременно может быть идентифицирован с нескольких клиентских приложений. Серверное и клиентское приложения должны работать на разных ОС.

1.3 Накладываемые ограничения

Серверное приложение не поддерживает одновременное подключение более 100 клиентов. Максимальная длина вводимых пользователем строковых данных — 255 символов.

Глава 2

Реализация для работы по протоколу TSP

2.1 Прикладной протокол

Команды, доступные клиентскому приложению приведены в таблице 2.1.

Имя команды	Аргумент 1	Аргумент 2
register	новый логин	-
sign in	логин	-
show users	-	-
check wallet	-	-
transfer	имя пользователя	кол-во денег
quit	-	-

Таблица 2.1: Команды, доступные клиентскому приложению

Описание команд:

1. register — создание нового пользователя. Возможные ответы сервера:
 - «ok» — команда выполнена успешно;
 - «not ok» — пользователь с таким именем уже существует.
2. sign in — вход уже существующего пользователя. Возможные ответы сервера:
 - «ok» — команда выполнена успешно;

- «no match» — пользователя с таким именем не найдено.
3. `show users` — запрос на передачу списка электронных кошельков пользователей платежной системы. В ответ сервер посылает список пользователей платёжной системы, в нём содержится имена пользователей и их идентификаторы.
 4. `check wallet` — запрос на проверку состояния электронного кошелька. Команда доступна только тестировщикам. В ответ сервер посылает количество денег на счету пользователя.
 5. `transfer` — запрос на платёж другому пользователю. Команда вводится в 2 этапа:
 - сначала пользователь вводит саму команду;
 - затем вводит имя пользователя кому будет проведён перевод и количество денег для перевода.

Возможные ответы сервера:

- «ok» — команда выполнена успешно;
 - «no match» — пользователя с таким именем не найдено.
6. `quit` — выход из клиентского приложения.

2.2 Архитектура приложения

При подключении нового клиента сервер создает новый поток. Сначала происходит идентификация подключившегося пользователя. После идентификации начинается процесс работы. Клиент отправляет серверу различные команды, сервер выполняет эти команды и отправляет клиенту результаты выполнения команд.

На сервере используются структура `user`, в которой содержится вся нужная информация о подключившемся клиенте. После идентификации вся информация о подключившемся клиенте заносится в эту структуру.

Данные о пользователе хранятся в 2-х файлах:

- `us.txt` — в этом файле хранятся имена пользователей и их идентификаторы, разделённые табуляцией, в одной строчке содержится информация только об одном пользователе;

- money.txt — здесь хранятся идентификаторы пользователей и количество денег на их счету, структура записей такая же как и в файле us.txt.

Пользователю на сервере доступны 2 команды:

1. show — вывод списка подключенных клиентов с номерами их сокетов;
2. disconnect номер клиента — отключение указанного клиента от сервера;

2.3 Тестирование

2.3.1 Описание тестового стенда и методики тестирования

Серверное приложение реализовано в ОС Linux. Тестирование проводилось на ОС Debian 8.1. Сервер запускался на виртуальной машине с подключением к сети через сетевой мост. Клиентское приложение реализовано в ОС Windows. Тестирование проводилось на ОС Windows 10. Для тестирования приложений запускается сервер платежной системы и несколько клиентов. В процессе тестирования проверяются основные возможности приложений по передаче и приему сообщений.

2.3.2 Тестовый план и результаты тестирования

Процесс тестирования:

1. Проверка идентификации:
 - Попытка войти с неправильным логином;
 - Попытка войти с правильным логином;
 - Попытка зарегистрироваться с уже занятым логином;
 - Попытка зарегистрироваться со свободным.

Результаты с попытками входа приведены на рисунке 1, результаты с попытками регистрации приведены на рис.2

```
C:\WINDOWS\system32\cmd.exe
Sign in or register
sign in
Enter your username:
ajgdkgdjadkgasd
No such username. Type correct username or register
Sign in or register
sign in
Enter your username:
wladex
Hello wladex, you has successfully logged in
Enter the command: Undefined command
Enter the command:
```

Рис.1

```
C:\WINDOWS\system32\cmd.exe
Sign in or register
register
Create new username:
wladex
User with this username is already existing. Create other username
Sign in or register
register
Create new username:
dima
Hello dima, you has successfully registered and logged in
Enter the command: Undefined command
Enter the command:
```

Рис.2

2. Проверка корректности выполнения всех команд клиента:

- Проверка функции передачи списка электронных кошельков пользователей платежной системы(рис. 3);

```
C:\WINDOWS\system32\cmd.exe
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5
Enter the command: _
```

Рис.3

- Проверка функции проверки состояния электронного кошелька (рис. 4);

```
Enter the command: check wallet
6000
Enter the command:
```

Рис.4

- Проверка функции платежа другому пользователю (рис. 5);

```

C:\WINDOWS\system32\cmd.exe
Sign in or register
sign in
Enter your username:
dima
Hello dima, you has successfully logged in
Enter the command: Undefined command
Enter the command: check wallet
6500
Enter the command: transfer
To who and how much do you want transfer money?
wladex 5000
Operation done
Enter the command: Undefined command
Enter the command:
Enter the command: check wallet
5500
Enter the command: check wallet
10500
Enter the command:

```

Рис.5

- Попытка выхода из клиентского приложения (рис. 6);

```

Enter the command: check wallet
10500
Enter the command: quit
Для продолжения нажмите любую клавишу . . .

```

Рис.6

- Проверка корректной работы нескольких клиентов (рис. 7);

```

C:\WINDOWS\system32\cmd.exe
Sign in or register
sign in
Enter your username:
wladex
Hello wladex, you has successfully logged in
Enter the command: Undefined command
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5
Enter the command: check wallet
10500
Enter the command: transfer
To who and how much do you want transfer money?
azat 3000
Operation done
Enter the command: Undefined command
Enter the command:

```

```

C:\WINDOWS\system32\cmd.exe
Sign in or register
sign in
Enter your username:
azat
Hello azat, you has successfully logged in
Enter the command: Undefined command
Enter the command: check wallet
7000
Enter the command: check wallet
10000
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5
Enter the command:

```

Рис.7

- Проверка отключения клиентов от сервера (рис. 8).

```

lera 3
egor 4
dima 5

show
5 : wladez
4 : lera
disconnect 4
wladez 1
azat 2
lera 3
egor 4
dima 5

show
5 : wladez
disconnect 5

```

C:\WINDOWS\system32\cmd.exe

```

Sign in or register
sign in
Enter your username:
wladez
Hello wladez, you has successfully logined
Enter the command: Undefined command
Enter the command: show users
wladez 1
azat 2
lera 3
egor 4
dima 5

Enter the command: check wallet
Disconnected from server
Для продолжения нажмите любую клавишу . . .

```

C:\WINDOWS\system32\cmd.exe

```

Sign in or register
sign in
Enter your username:
lera
Hello lera, you has successfully logined
Enter the command: Undefined command
Enter the command: show users
wladez 1
azat 2
lera 3
egor 4
dima 5

Enter the command: show users
Disconnected from server
Для продолжения нажмите любую клавишу . . .

```

Рис.8

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

Прикладной протокол UDP-приложения совпадает с протоколом TCP-приложения, описанного ранее в пункте 2.1.

3.2 Архитектура приложения

Архитектура UDP-приложения совпадает с архитектурой TCP-приложения, которая была описана в пункте 2.2. Небольшие изменения коснулись только клиентского приложения. Была добавлена структура `Uclient`, которая была нужна для того, чтобы в ней хранились сокет и структура `sockaddr` с адресом сервера.

3.3 Тестирование

3.3.1 Описание тестового стенда и методики тестирования

Тестовый стенд был описан ранее в пункте 2.3.1

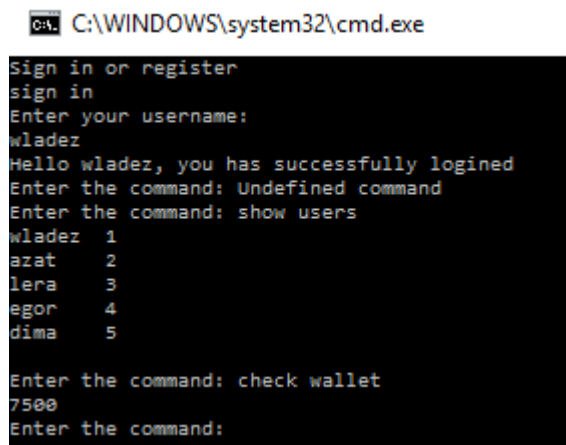
3.3.2 Тестовый план и результаты тестирования

Первый этап тестирования совпадает с тестированием TCP-приложения (пункт 2.3.2). Результаты тестирования совпали с тестированием TCP-

приложения.

Далее UDP реализация приложения была протестирована на устойчивость помехам в сети.

1. Введение дополнительной задержки в сеть 150 ± 50 мс (рис. 9)

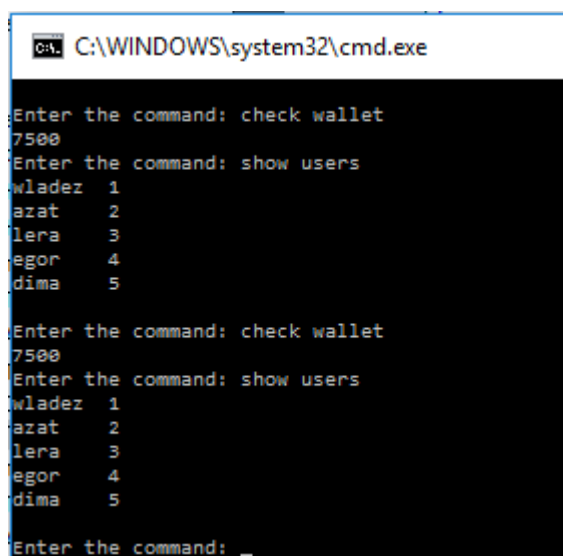


```
C:\WINDOWS\system32\cmd.exe
Sign in or register
sign in
Enter your username:
wladex
Hello wladex, you has successfully logined
Enter the command: Undefined command
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5
Enter the command: check wallet
7500
Enter the command:
```

Рис.9

При такой маленькой задержке проблем в работе приложения не возникнет.

2. Введение задержки в сеть 1500 ± 500 мс (рис. 10)

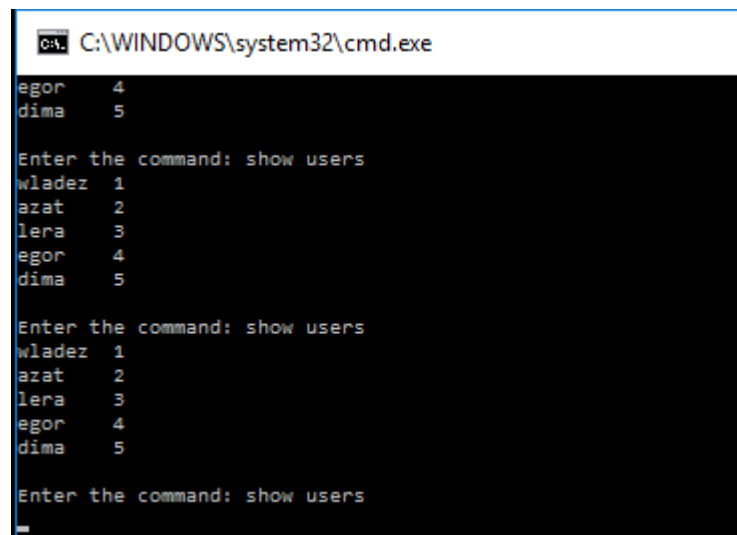


```
C:\WINDOWS\system32\cmd.exe
Enter the command: check wallet
7500
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5
Enter the command: check wallet
7500
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5
Enter the command: _
```

Рис.10

При большой задержке клиент получает некоторых сообщения от сервера, но они приходят не сразу и команды выполняются за 2 секунды.

3. Введение потери 30% пакетов (рис. 11)



```
C:\WINDOWS\system32\cmd.exe
egor 4
dima 5

Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5

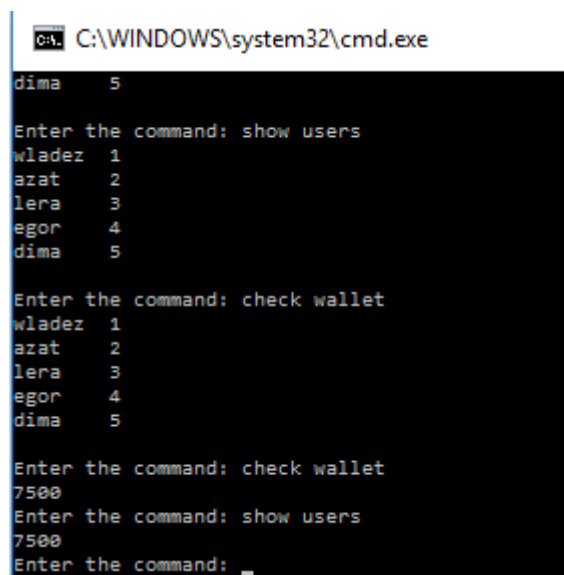
Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5

Enter the command: show users
```

Рис.11

При таком проценте потери пакетов приложение практически сразу завершает работу: ответ от сервера теряется в сети и приложение зависает.

4. Введение дубликации 20% пакетов (рис. 12)



```
C:\WINDOWS\system32\cmd.exe
dima 5

Enter the command: show users
wladex 1
azat 2
lera 3
egor 4
dima 5

Enter the command: check wallet
wladex 1
azat 2
lera 3
egor 4
dima 5

Enter the command: check wallet
7500
Enter the command: show users
7500
Enter the command:
```

Рис.12

При таком проценте дубликации пакетов приложение продолжает функционировать, однако в его работе возникают неполадки, на некоторые команды данные отображаются по предыдущей команде.

5. Введение искажения 10% пакетов (рис. 13)

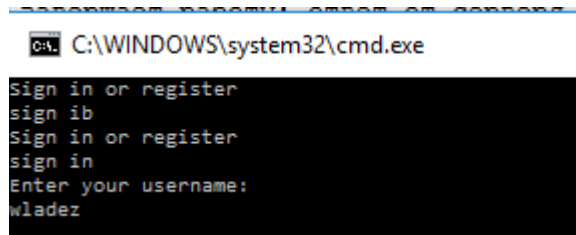


Рис.13

При таком проценте искажения пакетов приложение продолжает функционировать, однако есть большая вероятность того, что оно зависнет.

6. Введение задержки 150 ± 50 мс, искажения 5%, потери 5%, дубликации 5% пакетов (рис. 14)

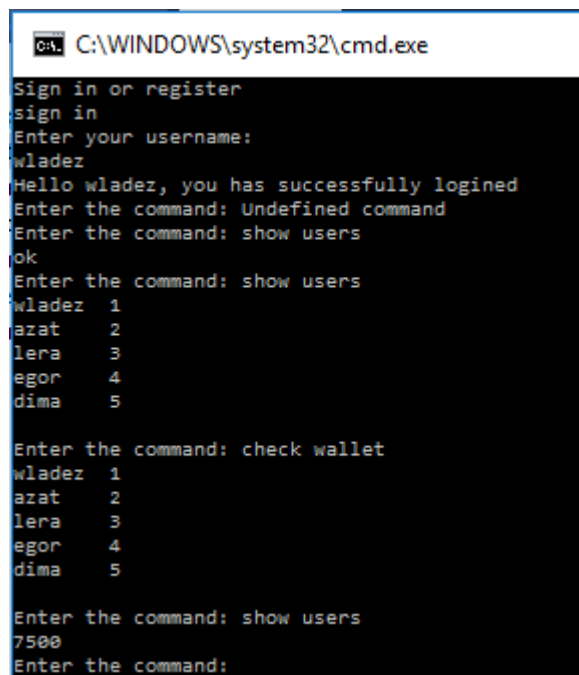


Рис.14

Приложение продолжает функционировать, однако могут возникать неполадки, так же приложение может зависнуть.

В целом можно сделать вывод, что приложение не готово к работе в реальной сети. При возникновении небольших помех в сети приложение продолжит корректно функционировать. Но при потере пакетов или резком понижении качества канала хотя бы по одному параметру приложение быстро зависнет свою работу из-за ошибки.

Глава 4

Проверка приложения с помощью Valgrind

Сначала была запущена проверка памяти с помощью команды `valgrind -leak-check=full`, проверка выдала 5 ошибок(рис. 15).

```
==1311== LEAK SUMMARY:
==1311==    definitely lost: 0 bytes in 0 blocks
==1311==    indirectly lost: 0 bytes in 0 blocks
==1311==    possibly lost: 456 bytes in 3 blocks
==1311==    still reachable: 772 bytes in 5 blocks
==1311==    suppressed: 0 bytes in 0 blocks
==1311== Reachable blocks (those to which a pointer was found) are not shown.
==1311== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==1311==
==1311== For counts of detected and suppressed errors, rerun with: -v
==1311== Use --track-origins=yes to see where uninitialised values come from
==1311== ERROR SUMMARY: 5 errors from 4 contexts (suppressed: 0 from 0)
```

Рис.15

Одна из выявленных ошибок: "Conditional jump or move depends on uninitialised value(s)" эта ошибка была исправлена добавлением инициализации одной переменной типа `int`. Остальные ошибки были связаны с созданием нового потока при вызове функции `pthread_create()`, эти ошибки были решены добавлением следующих атрибутов потока:

```
1 pthread_attr_t threadAttr;
2 pthread_attr_init(&threadAttr);
3 pthread_attr_setdetachstate(&threadAttr,
    PTHREAD_CREATE_DETACHED);
```

Затем была запущена проверка ошибок на многопоточность с помощью команды `valgrind -tool=helgrind`, проверка выдала ошибки гонки данных (рис. 16).

```
==1469== Possible data race during read of size 4 at 0x804B240 by thread #3
==1469== Locks held: none
==1469==    at 0x8049F81: disconnect(int) (server.cpp:282)
==1469==    by 0x8049868: doprocessing(void*) (server.cpp:136)
==1469==    by 0x402E6B2: mythread_wrapper (hg_intercepts.c:234)
==1469==    by 0x45A4EFA: start_thread (pthread_create.c:309)
==1469==    by 0x47F862D: clone (clone.S:129)
==1469==
==1469== This conflicts with a previous write of size 4 by thread #4
==1469== Locks held: none
==1469==    at 0x8049485: authentication(int) (server.cpp:61)
==1469==    by 0x804982C: doprocessing(void*) (server.cpp:127)
==1469==    by 0x402E6B2: mythread_wrapper (hg_intercepts.c:234)
==1469==    by 0x45A4EFA: start_thread (pthread_create.c:309)
==1469==    by 0x47F862D: clone (clone.S:129)
==1469== Address 0x804b240 is 0 bytes inside data symbol "clientsCount"
```

Рис.16

Данные ошибки были исправлены добавлением мьютекса `pthread_mutex_t` `clientsMutex` и добавлением функций `pthread_mutex_lock()` и `pthread_mutex_unlock()` для синхронизации при использовании общих объектов.

Глава 5

Выводы

В результате работы был создан прикладной протокол взаимодействия клиент-серверного приложения. В соответствии с прикладным протоколом было создано две реализации приложения для протоколов TCP и UDP. Клиентское и серверное приложения были реализованы для двух разных платформ: ОС Windows и Linux. При реализации использовались стандартные сокеты. Реализации сокетов для использованных ОС идентичны, портирование программ с одной платформы на другую выполняется достаточно просто.

В результате работы была создана клиент-серверная платёжная система. Система состоит из сервера, хранящего сведения о клиентах и состоянии их кошельков. Все поставленные требования были выполнены. Тестирование программ прошло успешно, тесты показали корректность работы приложения в условиях локальной сети.

5.1 Реализация для TCP

Протокол TCP удобен для реализации пользовательских приложений, так как обеспечивает установление соединения и надёжную доставку пакетов. Протокол обеспечивает стабильное надёжное соединение, поэтому при реализации своего протокола не требуется волноваться об этом. Однако, эти дополнительные средства синхронизации требуют больше времени на доставку, т.е. скорость передачи данных ниже чем в UDP.

С помощью утилиты `tc` при тестировании TCP реализации были про- симулированы помехи в сети. Тестирование показало, что приложение надёжно работает при низком и среднем уровне помех.

5.2 Реализация для UDP

Протокол UDP удобен для реализации приложений, не требующих точной доставки пакетов. Он позволяет передавать данные с большей скоростью, однако вероятность потери пакета при этом выше, чем в TCP. Поэтому, использовать данный протокол для реализации поставленной задачи не очень удобно. Требуется использовать дополнительные инструменты для подтверждения корректной доставки, т.е. каким-то образом «симулировать» TCP. Это неудобно и неэффективно. Для обеспечения более надежной доставки при использовании UDP было добавлено подтверждение доставки при обмене между клиентом и сервером.

С помощью утилиты `tc` при тестировании UDP реализации были про-симулированы помехи в сети. Тестирование показало, что при низком уровне помех приложение продолжает функционировать, однако могут появляться ошибки критические для приложения. Если в сети присутствуют сильные помехи, приложение начинает работать некорректно и может зависнуть с большой вероятностью.

Приложения

Описание среды разработки

Серверное приложение реализовывалось в ОС Debian версии 8.1. Среда разработки — Qt Creator версии 5.5.

Клиентское приложения реализовывалось в ОС Windows 10. Среда разработки — Microsoft Visual Studio 2010.

Листинги

ТСР сервер

Файл main.cpp

```
1 #include <QCoreApplication>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <cstdio>
5 #include <stdlib.h>
6 #include <netdb.h>
7 #include <netinet/in.h>
8 #include <string.h>
9 #include <string>
10 #include <unistd.h>
11 #include "server.h"
12
13 int main(int argc, char *argv[])
14 {
15     int sock, newsock, port_num, n;
16     struct sockaddr_in serv_addr, cli_addr;
17     socklen_t clilen;
18     pid_t pid;
19
20     sock=socket(AF_INET, SOCK_STREAM, 0);
21
22     if (sock < 0)
```

```

23         {
24             perror("ERROR opening socket");
25             exit(1);
26         }
27     bzero((char*)&serv_addr, sizeof(serv_addr));
28     port_num=12345;
29
30     serv_addr.sin_family=AF_INET;
31     serv_addr.sin_addr.s_addr=INADDR_ANY;
32     serv_addr.sin_port=htons(port_num);
33
34     if(bind(sock, (struct sockaddr*)&serv_addr, sizeof(
35         serv_addr))<0){
36         perror("ERROR on binding");
37         exit(1);
38     }
39
40     listen(sock, 10);
41     clilen=sizeof(cli_addr);
42     printf("TCP Server Waiting for client on port 12345\n");
43
44     pthread_attr_t threadAttr;
45     pthread_attr_init(&threadAttr);
46     pthread_attr_setdetachstate(&threadAttr,
47         PTHREAD_CREATE_DETACHED);
48
49     pthread_t serv_thread;
50     if (pthread_create(&serv_thread, NULL, server_handler,
51         NULL) != 0) {
52         printf("Error while creating thread for server\n");
53     }
54
55     while (1) {
56         pthread_t thread;
57         newsock = accept(sock, (struct sockaddr *) &cli_addr
58             , &clilen);
59
60         if (newsock < 0) {
61             perror("ERROR on accept");
62             close(sock);
63             exit(1);
64         }
65         pid=pthread_create(&thread, NULL, doprocessing, &
66             newsock);
67         if (pid!=0){
68             printf("Error while creating new thread\n");
69             close(newsock);
70             break;

```

```

66         }
67     } /* end of while */
68     return 0;
69 }
70 }

```

Файл server.cpp

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <string.h>
4  #include <cstdlib>
5  #include <netdb.h>
6  #include <netinet/in.h>
7  #include <unistd.h>
8  #include "server.h"
9  #include "user.h"
10
11 int clientsCount=0;
12 user connected[maxClients];
13 pthread_mutex_t clientsMutex;
14
15 int authentication(int sock){
16     char buf[bufSize+1];
17     char new_client[]="new";
18     char exist_client[]="exist";
19     char numb[bufSize];
20     int n,check;
21     int res;
22     strcpy(numb,"\t");
23     bzero(buf, bufSize+1);
24     n=recv(sock, buf, bufSize, 0);//приём сообщения от клиент
25     а, в котором указано
26     //будет ли подключен новый пользователь или уже зарегистр
27     ированный
28     if (n < 0) {
29         perror("ERROR_reading_from_socket");
30         exit(1);
31     }
32     if(strncmp(buf,exist_client,sizeof(exist_client)-1) == 0)
33     {
34         //уже существующий пользователь
35         bzero(buf,bufSize);
36         n=recv(sock, buf, bufSize, 0);
37         if (n < 0) {
38             perror("ERROR_reading_from_socket");
39             pthread_exit(0);
40         }
41         check=check_user(buf);//проверка совпадает ли имя кот
42         орое ввёл пользователь

```

```

39      //с именем в файле с пользователями
40      if (check < 0) { //нет совпадений
41          n = send(sock, "no_match", sizeof(buf), 0); //говори
42              м клиенту, что нет совпадений
43          if (n < 0)
44          {
45              perror("ERROR_writing_to_socket");
46              pthread_exit(0);
47          }
48          res=-1;
49      }
50      else { //есть совпадения
51          n = send(sock, "ok", bufSize, 0); //говорим клиенту,
52              что всё ок
53          if (n < 0)
54          {
55              perror("ERROR_writing_to_socket");
56              pthread_exit(0);
57          }
58          connected[clientsCount].sock=sock;
59          strcpy(connected[clientsCount].name, buf);
60          connected[clientsCount].uid=check;
61          connected[clientsCount].money=get_money(connected
62              [clientsCount].uid);
63          res=1;
64          add();
65          printf("Connected_client_%s\n", buf);
66      }
67      }
68      else if (strncmp(buf, new_client, sizeof(new_client)-1) ==
69          0) {
70          //создание нового пользователя
71          bzero(buf, bufSize);
72          n=recv(sock, buf, bufSize, 0);
73          if (n < 0) {
74              perror("ERROR_reading_from_socket");
75              pthread_exit(0);
76          }
77          }
78          check=check_user(buf); //проверка нет ли уже такого им
79              ени у кого-нибудь
80          if (check > 0) { //есть совпадения
81              n = send(sock, "not_ok", 6, 0); //говорим клиенту, ч
82                  то не ок
83              if (n < 0)
84              {
85                  perror("ERROR_writing_to_socket");
86                  pthread_exit(0);
87              }
88          }

```

```

82         res=-1;
83     }
84     else{//нет совпадений, клиент зарегистрирован
85         n = send(sock,"ok", 2, 0);//говорим клиенту, что
            всё ок
86         if (n < 0)
87         {
88             perror("ERROR_writing_to_socket");
89             pthread_exit(0);
90         }
91         connected[clientsCount].sock=sock;
92         strcpy(connected[clientsCount].name,buf);
93         connected[clientsCount].money=6000;
94         printf("Connected_client_%s\n",buf);
95         connected[clientsCount].uid=set_newid();//получен
            ue id пользователя
96         FILE *file;
97         char *fname = "/home/user/project_t/us.txt";
98         file = fopen(fname,"a");
99         fprintf(file,"%s\t",connected[clientsCount].name)
            ;//запись в файл нового пользователя
100        fprintf(file,"%i\n",connected[clientsCount].uid);
101        fclose(file);
102        FILE *mon;
103        char *mon_name="/home/user/project_t/money.txt";
104        mon=fopen(mon_name,"a");
105        fprintf(mon,"%i\t",connected[clientsCount].uid);
106        fprintf(mon,"%i\n",connected[
            clientsCount].money);
107        fclose(mon);
108        add();
109        res=1;
110    }
111 }
112 return res;
113 }
114
115 void* doprocessing (void* newsock) {
116     int n, socket;
117     int aut;
118     char request[bufSize];
119     char buffer[bufSize+1];
120     char command[]="show_users";
121     char quit[]="quit";
122     bzero(request,bufSize);
123     bzero(buffer,bufSize+1);
124     int *tmp=(int*)newsock;
125     socket=*tmp;
126     do {

```

```

127         aut = authentication(socket); //процесс аутентифик
           ацции клиента
128     } while (aut < 0);
129     while(recv(socket, request, bufSize, 0)>=0){
130         //n=recv(socket, request, bufSize, 0);
131         // if(n<0){
132         //     perror("ERROR reading from socket");
133         //     break;
134         // }
135         if(strncmp(request,quit,sizeof(quit)-1) == 0){
136             disconnect(socket);
137             break;
138         }
139         else if(strncmp(request,command,sizeof(command)-1) ==
            0){
140             show_users(socket);
141         }
142         else if(strcmp(request,"wallet") == 0){
143             check_wallet(socket);
144         }
145         else if(strcmp(request,"transf") == 0){
146             transfer(socket);
147         }
148         bzero(request,bufSize);
149     }
150     close(socket);
151 }
152
153 void show_users(int sock){
154     int n;
155     char tmp[bufSize];
156     char buffer[bufSize+1];
157     bzero(buffer,bufSize+1);
158     FILE *file;
159     char *fname = "/home/user/project_t/us.txt";
160     file = fopen(fname,"r");
161     if(file == NULL)
162     {
163         perror("ERROR_on_openning_file_with_users");
164         pthread_exit(0);
165     }
166     while (fgets (tmp, sizeof(tmp), file) != NULL){
167         strncat(buffer,tmp,35);
168         printf("%s", tmp);
169     }
170     printf("\n");
171     fclose(file);
172     n = send(sock,buffer,sizeof(buffer), 0);
173     if (n < 0)

```



```

174     {
175         pthread_exit(0);
176     }
177 }
178
179 void check_wallet(int sock){
180     int n,uid;
181     char buffer[bufSize+1];
182     bzero(buffer,bufSize+1);
183     int i = 0;
184     int j=0;
185     for (i = 0; i <= clientsCount; ++i) {
186         if (connected[i].sock == sock){
187             uid=connected[i].uid;
188             connected[i].money=get_money(uid);
189             j=i;
190         }
191     }
192     sprintf(buffer,"%i", connected[j].money);
193     n = send(sock,buffer,sizeof(buffer), 0);
194     if (n < 0)
195     {
196         pthread_exit(0);
197     }
198 }
199
200 void transfer(int sock){
201     int n,value,money;
202     int dest=0;
203     char buffer[bufSize+1];
204     bzero(buffer,bufSize+1);
205     n=recv(sock, buffer, bufSize+1, 0);
206     if(n<0){
207         perror("ERROR reading from socket");
208         pthread_exit(0);
209     }
210     printf("%s\n",buffer);
211     char *tmp=strstr(buffer," ");
212     value=atoi(tmp);
213     strcpy(tmp,"\0");
214     dest=check_user(buffer);
215     if (dest < 0) { //нет совпадений
216         n = send(sock,"no_match",8, 0); //говорим клиенту, что
            нет совпадений
217         if (n < 0)
218         {
219             pthread_exit(0);
220         }
221     }

```

```

222     else{
223         money=get_money(dest);
224         money+=value;
225         set_money(dest,money);
226         int i,j;
227         for (i = 0; i <= clientsCount; ++i) {
228             if (connected[i].sock == sock){
229                 dest=connected[i].uid;
230                 connected[i].money=get_money(dest);
231                 connected[i].money-=value;
232                 j=i;
233             }
234         }
235         set_money(dest,connected[j].money);
236         n = send(sock,"ok",2, 0);
237         if (n < 0)
238         {
239             pthread_exit(0);
240         }
241     }
242 }
243
244 int check_user(char buf[]){
245     char name[bufSize+1];
246     char tmp[bufSize+1];
247     char id[10];
248     int res=-1;
249     int k=-1;
250     FILE *file;
251     char *fname = "/home/user/project_t/us.txt";
252     file = fopen(fname,"r");
253     bzero(name,bufSize+1);
254     strcpy(name,buf);
255     if(file == NULL)
256     {
257         perror("ERROR on opening file with users");
258         exit(1);
259     }
260     int i = 0;
261     bzero(tmp,bufSize+1);
262     while(fscanf(file,"%s", tmp)!=EOF){
263         //fscanf(file,"%s", tmp);
264         if(k==0){
265             strcpy(id,tmp);
266             res=atoi(id);
267             printf("%d\n",res);
268             break;
269         }
270         if(!(i%2)){

```

```

271         k=strcmp(name,tmp);
272         printf("%s\n",tmp);
273     }
274     i++;
275 }
276 fclose(file);
277 return res;
278 }
279
280 void add(){
281     pthread_mutex_lock(&clientsMutex);
282     clientsCount++;
283     pthread_mutex_unlock(&clientsMutex);
284 }
285
286 void disconnect(int sock){
287     pthread_mutex_lock(&clientsMutex);
288     int i = 0;
289     for (i = 0; i < clientsCount; ++i) {
290         if (connected[i].sock == sock)
291             break;
292     }
293     if (i != clientsCount) {
294         for (++i; i < clientsCount; ++i) {
295             connected[i - 1] = connected[i];
296         }
297         --clientsCount;
298     }
299     pthread_mutex_unlock(&clientsMutex);
300 }
301
302 void* server_handler(void*){
303     while (1) {
304         char command[bufSize];
305         bzero(command, bufSize);
306         scanf("%s", command);
307         if (strcmp(command, "show") == 0) {
308             for (int i = 0; i < clientsCount; ++i) {
309                 printf("%d: %s\n", connected[i].sock,
310                     connected[i].name);
311             }
312         } else if (strcmp(command, "disconnect") == 0) {
313             int sock = 0;
314             scanf("%d", &sock);
315             disconnect(sock);
316             int n = send(sock, "exit", 4, 0);
317             if (n < 0)
318                 perror("ERROR_writing_to_socket");

```

```

319         exit(1);
320     }
321     close(sock);
322 } else {
323     printf("Undefined command\n");
324 }
325 }
326 }

```

Файл server.h

```

1 #ifndef SERVER
2 #define SERVER
3 void *doprocessing(void *sock);
4 void show_users(int sock);
5 int authentication(int sock);
6 int check_user(char buf[]);
7 void check_wallet(int sock);
8 void transfer(int sock);
9 void *server_handler(void *);
10 void disconnect(int sock);
11 void add();
12 #endif // SERVER

```

Файл user.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <cstdlib>
4 #include "user.h"
5
6 int get_money(int usid){//узнать количество денег, имеющееся
   у пользователя
7     int i=0;
8     int res,tmp=0;
9     int k=-1;
10    char buffer[bufSize+1];
11    FILE *file;
12    char *fname="/home/user/project_t/money.txt";
13    file = fopen(fname,"r");
14    if(file == NULL)
15    {
16        perror("ERROR on opening file with money");
17        exit(1);
18    }
19    bzero(buffer,bufSize+1);
20    while(!feof(file)){
21        fscanf(file,"%s", buffer);
22        tmp=atoi(buffer);
23        if(k==0){

```

```

24         res=tmp;
25         break;
26     }
27     if(!(i%2)){
28         if(usid==tmp)
29             k=0;
30     }
31     i++;
32 }
33 fclose(file);
34 return res;
35 }
36
37 int set_money(int uid, int value){
38     char buf[15];
39     int tmp=0;
40     int before=0;
41     int after=0;
42     int spaces=0;
43     fpos_t pos;
44     FILE *file;
45     char *fname="/home/user/project_t/money.txt";
46     file = fopen(fname,"r+");
47     if(file == NULL)
48     {
49         perror("ERROR on opening file with money");
50         exit(1);
51     }
52     int k=1;
53     int i = 0;
54     while(fscanf(file,"%s", buf)!=EOF){
55         if(k==0){
56             before=ftell(file);
57             fsetpos(file,&pos);
58             fprintf(file,"\t%i",value);
59             after=ftell(file);
60             if(before>after) {
61                 spaces += before - after;
62                 while(spaces!=0){
63                     fputc('_', file);
64                     spaces--;
65                 }
66             }
67             fflush(file);
68             break;
69         }
70         fgetpos(file, &pos);
71         if(!(i%2)){
72             tmp=atoi(buf);

```

```

73         if(tmp==uid){
74             k=0;
75         }
76     }
77     i++;
78 }
79
80 }
81
82 int set_newid(){//задание нового id пользователя при регистра
    ции
83     int res=0;
84     char uid[15];
85     FILE *file;
86     char *fname = "/home/user/project_t/us.txt";
87     file = fopen(fname,"r");
88     bzero(uid,sizeof(uid));
89     if(file == NULL)
90     {
91         perror("ERROR_on_openning_file_with_users");
92         exit(1);
93     }
94     int i = 0;
95     while(fscanf(file,"%s", uid)!=EOF){
96         if(i%2){
97             res=atoi(uid);
98         }
99         i++;
100     }
101     fclose(file);
102     return res+1;
103 }

```

Файл user.h

```

1  #ifndef USER
2  #define USER
3  #define bufSize 255
4  #define maxClients 100
5  typedef struct{
6      char name[bufSize];
7      int sock;
8      int uid;
9      int money;
10 }user;
11
12 //user();
13 int set_newid();
14 int get_money(int usid);
15 int set_money(int uid, int value);

```

```
16 #endif // USER
```

ТСР клиент

Файл client.cpp

```
1  //client
2  #include <winsock2.h>
3  #include <ws2tcpip.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <string.h>
7
8  #pragma comment (lib, "Ws2_32.lib")
9  #pragma comment (lib, "Mswsock.lib")
10 #pragma comment (lib, "AdvApi32.lib")
11
12 #define bufSize 255
13
14 int authentication(int sockfd);
15 void showUsers(int sockfd, char command[]);
16 void checkWallet(int sockfd);
17 void transfer(int sockfd);
18 int disconnect(int sockfd, char* buf);
19
20 int main(int argc, char *argv[]) {
21     WORD wVersionRequested = MAKEWORD(1, 1);           // Stuff
22     WSADATA wsaData;
23     int sockfd, portno, n;
24     int aut;
25     struct sockaddr_in serv_addr;
26     struct hostent *server;
27     char quit[]="quit";
28     char show[]="show_users";
29     char wallet[]="check_wallet";
30     char transf[]="transfer";
31     char buffer[bufSize+1];
32
33     WSStartup(wVersionRequested, &wsaData);
34     //portno=12345;
35     if (argc < 3) {
36         fprintf(stderr, "usage %s hostname port\n", argv[0]);
37         exit(0);
38     }
39
40     portno = atoi(argv[2]);
41 }
```

```

42  /* Create a socket point */
43  sockfd = socket(AF_INET, SOCK_STREAM, 0);
44
45  if (sockfd < 0) {
46      perror("ERROR_□opening_□socket");
47      exit(1);
48  }
49
50  server = gethostbyname(argv[1]);
51  if (server == NULL) {
52      fprintf(stderr, "ERROR, □no_□such_□host\n");
53      exit(0);
54  }
55
56  memset((char *) &serv_addr, 0, sizeof(serv_addr));
57  serv_addr.sin_family = AF_INET;
58  serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
59  //strncpy((char *)server->h_addr, (char *)&serv_addr.
   sin_addr.s_addr, server->h_length);
60  serv_addr.sin_port = htons(portno);
61
62  /* Now connect to the server */
63  if (connect(sockfd, (struct sockaddr*)&serv_addr, sizeof(
   serv_addr)) < 0) {
64      perror("ERROR_□connecting");
65      exit(1);
66  }
67
68  /* Now ask for a message from the user, this message
69  * will be read by server
70  */
71  do {
72      aut = authentication(sockfd); //процесс аутентифик
   ации клиента
73      } while (aut < 0);
74  while (1){
75      printf("Enter_□the_□command:□");
76      memset(buffer, 0, bufSize+1);
77      fgets(buffer, bufSize+1, stdin);
78
79      if(strncmp(buffer, quit, sizeof(quit)-1) == 0){
80          n = send(sockfd, buffer, strlen(buffer), 0);
81          if (n < 0) {
82              perror("ERROR_□writing_□to_□socket");
83              exit(1);
84          }
85          closesocket(sockfd);
86          break;
87      }

```



```

88     else if(strncmp(buffer,show,sizeof(show)-1) == 0){
89         showUsers(sockfd,buffer);
90     }
91     else if(strncmp(buffer, wallet,sizeof(wallet)-1) == 0){
92         checkWallet(sockfd);
93     }
94     else if(strncmp(buffer, transf,sizeof(transf)-1) == 0){
95         transfer(sockfd);
96     }
97     else{
98         printf("Undefined command\n");
99     }
100 }
101 return 0;
102 }
103
104 void showUsers(int sockfd, char command[]){
105     int n;
106     char buffer[bufSize+1];
107     memset(buffer, 0, bufSize);
108     strcpy(buffer,command);
109     n = send(sockfd, buffer, strlen(buffer),0);
110     if (n < 0) {
111         perror("ERROR writing to socket");
112         closesocket(sockfd);
113         exit(1);
114     }
115
116     /* Now read server response */
117     memset(buffer, 0, bufSize);
118     n = recv(sockfd, buffer, bufSize+1,0);
119     if (n < 0) {
120         perror("ERROR reading from socket");
121         closesocket(sockfd);
122         exit(1);
123     }
124     disconnect(sockfd, buffer);
125     printf("%s\n",buffer);
126 }
127
128 void checkWallet(int sockfd){
129     int n;
130     char buffer[bufSize+1];
131     n=send(sockfd, "wallet", 6, 0);
132     if (n < 0) {
133         perror("ERROR writing to socket");
134         closesocket(sockfd);
135         exit(1);
136     }

```

```

137     memset(buffer, 0, bufSize+1);
138     n = recv(sockfd, buffer, bufSize+1,0);
139     if (n < 0) {
140         perror("ERROR_reading_from_socket");
141         closesocket(sockfd);
142         exit(1);
143     }
144     disconnect(sockfd, buffer);
145     printf("%s\n",buffer);
146 }
147
148 void transfer(int sockfd){
149     int n;
150     char tmp[bufSize];
151     char buffer[bufSize+1];
152     memset(buffer, 0, bufSize+1);
153     n=send(sockfd, "transf", 6, 0);
154     if (n < 0) {
155         perror("ERROR_writing_to_socket");
156         closesocket(sockfd);
157         exit(1);
158     }
159     printf("To_who_and_how_much_do_you_want_transfer_money?\n");
160     scanf("%s",buffer);
161     scanf("%s",tmp);
162     strcat(buffer,"_");
163     strcat(buffer,tmp);
164     n=send(sockfd, buffer, bufSize+1, 0);
165     memset(buffer, 0, bufSize+1);
166     n=recv(sockfd, buffer, bufSize+1,0);
167     if (n < 0) {
168         perror("ERROR_reading_from_socket");
169         closesocket(sockfd);
170         exit(1);
171     }
172     disconnect(sockfd, buffer);
173     if(strcmp(buffer,"no_match")==0){
174         printf("There_is_no_user_with_such_username\n");
175     }
176     else if(strcmp(buffer,"ok")==0){
177         printf("Operation_done\n");
178     }
179     else{
180         printf("Some_error_occurs_during_the_operation\n");
181     }
182 }
183
184 int disconnect(int sockfd, char* buf){

```

```

185     if (strcmp(buf, "exit") == 0) {
186         printf("Disconnected_from_server\n");
187         closesocket(sockfd);
188         exit(1);
189     }
190     else return -1;
191 }
192
193 int authentication(int sockfd){
194     char login[bufSize +1];
195     char reg[bufSize +1];
196     char buffer[bufSize +1];
197     char ok[]="ok";
198     char sign[]="sign_in";
199     char registration[]="register";
200     int n;
201     int res=-1;
202     memset(buffer, 0, bufSize+1);
203     printf("Sign_in_or_register\n");
204     fgets(buffer, bufSize+1, stdin);
205     //scanf("%s", &buffer);
206     if(strncmp(buffer, sign, sizeof(sign)-1) == 0){//выход существ
        //ующего пользователя
207         printf("Enter_your_username:\n");
208         scanf("%s", &login);
209         n=send(sockfd, "exist", bufSize, 0); //посылка серверу со
        //общения о том, что входит существующий пользовател
        //ь
210         if (n < 0) {
211             perror("ERROR_writing_to_socket");
212             closesocket(sockfd);
213             exit(1);
214         }
215         n=send(sockfd, login, bufSize, 0); //посылка серверу имен
        //и пользователя
216         if (n < 0) {
217             perror("ERROR_writing_to_socket");
218             closesocket(sockfd);
219             exit(1);
220         }
221         memset(buffer, 0, bufSize+1);
222         n = recv(sockfd, buffer, bufSize+1, 0); //ответ от серве
        //ра, правильны ли данные или нет
223         if (n < 0) {
224             perror("ERROR_reading_from_socket");
225             closesocket(sockfd);
226             exit(1);
227         }
228         disconnect(sockfd, buffer);

```

```

229         if(strncmp(buffer,ok,sizeof(ok)-1) == 0){//данные пра
           вильны
230             printf("Hello%s, you has successfully logged\n"
               ,login);
231             res=1;
232         }
233         else{//данные не правильны
234             printf("No such username. Type correct username
               or register\n");
235             res=-1;
236         }
237     }
238     else if(strncmp(buffer,registration,sizeof(registration)
239 -1) == 0){//регистрация нового пользователя
240         printf("Create new username:\n");
241         scanf("%s", &reg);
242         n=send(sockfd,"new",bufSize,0);//посылка сообщения се
           реверу о регистрации нового пользователя
243         if (n < 0) {
244             perror("ERROR writing to socket");
245             closesocket(sockfd);
246             exit(1);
247         }
248         n=send(sockfd,reg,bufSize,0);//посылка имени нового п
           ользователя
249         if (n < 0) {
250             perror("ERROR writing to socket");
251             closesocket(sockfd);
252             exit(1);
253         }
254         memset(buffer, 0, bufSize+1);
255         n = recv(sockfd, buffer, bufSize+1,0);//ответ от серв
           ера, правильны ли данные или нет
256         if (n < 0) {
257             perror("ERROR reading from socket");
258             closesocket(sockfd);
259             exit(1);
260         }
261         disconnect(sockfd, buffer);
262         if(strncmp(buffer,ok,sizeof(ok)-1) == 0){//данные пра
           вильны
263             printf("Hello%s, you has successfully registered
               and logged\n",reg);
264             res=1;
265         }
266         else{//данные не правильны
267             printf("User with this username is already
               existing. Create other username\n");

```

```

268     }
269     return res;
270 }

```

UDP сервер

Файл main.cpp

```

1  #include <QCoreApplication>
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <cstdio>
5  #include <stdlib.h>
6  #include <netdb.h>
7  #include <netinet/in.h>
8  #include <string.h>
9  #include <string>
10 #include <unistd.h>
11 #include "server.h"
12
13
14 int main(int argc, char *argv[])
15 {
16     int sock, port_num;
17     struct sockaddr_in serv_addr;
18     socklen_t clilen;
19     pid_t pid;
20
21     sock=socket(AF_INET, SOCK_DGRAM, 0);
22
23     if (sock < 0)
24     {
25         perror("ERROR_opening_socket");
26         exit(1);
27     }
28     bzero((char*)&serv_addr, sizeof(serv_addr));
29     //port_num=12345;
30     port_num = atoi(argv[1]);
31
32     serv_addr.sin_family=AF_INET;
33     serv_addr.sin_addr.s_addr=INADDR_ANY;
34     serv_addr.sin_port=htons(port_num);
35
36     if(bind(sock,(struct sockaddr*)&serv_addr,sizeof(
37         serv_addr))<0){
38         perror("ERROR_on_binding");
39         exit(1);

```

```

40
41     printf("UDP_Server_Waiting_for_client_on_port_12345\n");
42
43     pthread_attr_t threadAttr;
44     pthread_attr_init(&threadAttr);
45     pthread_attr_setdetachstate(&threadAttr,
        PTHREAD_CREATE_DETACHED);
46
47     pthread_t serv_thread;
48     if (pthread_create(&serv_thread, NULL, server_handler,
        NULL) != 0) {
49         printf("Error_while_creating_thread_for_server\n");
50     }
51
52     while (1) {
53         pthread_t thread;
54         user client = new_connection(sock);
55         pid=pthread_create(&thread, NULL, doprocessing, (void
            *)&client);
56         if (pid!=0){
57             printf("Error_while_creating_new_thread\n");
58             //break;
59         }
60     } /* end of while */
61     return 0;
62
63 }

```

Файл server.cpp

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <string.h>
4  #include <cstdlib>
5  #include <netdb.h>
6  #include <netinet/in.h>
7  #include <unistd.h>
8  #include "server.h"
9
10 int clientsCount=0;
11 user connected[maxClients];
12 pthread_mutex_t clientsMutex;
13
14 user new_connection(int sockfd){
15     struct sockaddr_in clientaddr; /* client addr */
16     int clientlen = sizeof(clientaddr); /* byte size of
        client's address */
17     char buf[bufSize];
18     int err = 0;
19

```

```

20     bzero(buf, bufSize);
21     err = recvfrom(sockfd, buf, bufSize, 0, (struct sockaddr
        *) &clientaddr, (unsigned int*)&clientlen);
22     if (err < 0) {
23         perror("ERROR_reading_from_socket");
24         exit(1);
25     }
26
27     int newsockfd;
28     uint16_t newport;
29     new_socket(&newsockfd, &newport);
30     bzero(buf, bufSize);
31     sprintf(buf, "%d", newport);
32     err = sendto(sockfd, buf, strlen(buf), 0, (struct
        sockaddr *) &clientaddr, clientlen);
33     if (err < 0) {
34         perror("ERROR_writing_to_socket");
35         exit(1);
36     }
37
38     // create new socket for this client
39     struct sockaddr_in newclientaddr; /* client addr */
40     int newclientlen = sizeof(newclientaddr); /* byte size of
        client's address */
41     bzero(buf, bufSize);
42     err = recvfrom(newsockfd, buf, bufSize, 0, (struct
        sockaddr *) &newclientaddr, (unsigned int*)&
        newclientlen);
43     if (err < 0) {
44         perror("ERROR_reading_from_socket");
45         exit(1);
46     }
47
48     user client;
49     client.cli_addr=newclientaddr;
50     client.clilen=newclientlen;
51     client.sock=newsockfd;
52     return client;
53 }
54
55 void new_socket(int* sockfd, uint16_t* port){
56     struct sockaddr_in serveraddr;
57     int err = 0;
58     *sockfd = socket(AF_INET, SOCK_DGRAM, 0);
59     *port = 12345 + (clientsCount + 1);
60     if(*sockfd<0){
61         perror("ERROR_opening_socket");
62         exit(1);
63     }

```

```

64 //int optval = 1;
65 //setsockopt(*sockfd, SOL_SOCKET, SO_REUSEADDR, (const
    void *)&optval, sizeof(int));
66
67 bzero((char *) &serveraddr, sizeof(serveraddr));
68 serveraddr.sin_family = AF_INET;
69 serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
70 serveraddr.sin_port = htons((unsigned short)*port);
71
72 err = bind(*sockfd, (struct sockaddr *) &serveraddr,
    sizeof(serveraddr));
73 if(err<0){
74     perror("ERROR_on_binding");
75     exit(1);
76 }
77 }
78
79 int authentication(user client){
80     char buf[bufSize+1];
81     char new_client[]="new";
82     char exist_client[]="exist";
83     char numb[bufSize];
84     int n,check;
85     int res;
86     strcpy(numb,"\t");
87     bzero(buf, bufSize+1);
88     n=recvfrom(client.sock, buf, bufSize, 0, (struct sockaddr
        *)&client.cli_addr, &client.clilen);//приём сообщения
        от клиента, в котором указано
89 //будет ли подключен новый пользователь или уже зарегистр
        ированный
90     if (n < 0) {
91         perror("ERROR_reading_from_socket");
92         exit(1);
93     }
94     if(strncmp(buf,exist_client,sizeof(exist_client)-1) == 0)
        {
95         //уже существующий пользователь
96         bzero(buf,bufSize);
97         n=recvfrom(client.sock, buf, bufSize, 0, (struct
            sockaddr *) &client.cli_addr, &client.clilen);
98         if (n < 0) {
99             perror("ERROR_reading_from_socket");
100             pthread_exit(0);
101         }
102         check=check_user(buf);//проверка совпадает ли имя кот
            орое ввёл пользователь
103         //с именем в файле с пользователями
104         if (check < 0) {//нет совпадений

```



```

105         n = sendto(client.sock,"no_match",sizeof(buf), 0,
106             (struct sockaddr *) &client.cli_addr, client.
107             clilen);//говорим клиенту, что нет совпадений
108         if (n < 0)
109         {
110             perror("ERROR_writing_to_socket");
111             pthread_exit(0);
112         }
113         res=-1;
114     }
115     else{//есть совпадения
116         n = sendto(client.sock,"ok",bufSize, 0, (struct
117             sockaddr *) &client.cli_addr, client.clilen);
118         //говорим клиенту, что всё ок
119         if (n < 0)
120         {
121             perror("ERROR_writing_to_socket");
122             pthread_exit(0);
123         }
124         strcpy(client.name,buf);
125         client.uid=check;
126         client.money=get_money(client.uid);
127         res=1;
128         add(client);
129         printf("Connected_client_%s\n",buf);
130     }
131 }
132 else if(strncmp(buf,new_client,sizeof(new_client)-1) ==
133 0){
134     //создание нового пользователя
135     bzero(buf,bufSize);
136     n=recvfrom(client.sock, buf, bufSize, 0, (struct
137         sockaddr *) &client.cli_addr, &client.clilen);
138     if (n < 0) {
139         perror("ERROR_reading_from_socket");
140         pthread_exit(0);
141     }
142     check=check_user(buf);//проверка нет ли уже такого им
143     ени у кого-нибудь
144     if(check>0){//есть совпадения
145         n = sendto(client.sock,"not_ok",6, 0, (struct
146             sockaddr *) &client.cli_addr, client.clilen);
147         //говорим клиенту, что не ок
148         if (n < 0)
149         {
150             perror("ERROR_writing_to_socket");
151             pthread_exit(0);
152         }
153     }
154 }

```



```

189         } while (aut < 0);
190     while(recvfrom(client.sock, request, bufSize, 0, (struct
        sockaddr *) &client.cli_addr, &client.clilen)>=0){
191         //n=recv(socket, request, bufSize, 0);
192         //     if(n<0){
193         //         perror("ERROR reading from socket");
194         //         break;
195         //     }
196         if(strncmp(request,quit,sizeof(quit)-1) == 0){
197             disconnect(client.sock);
198             break;
199         }
200         else if(strncmp(request,command,sizeof(command)-1) ==
            0){
201             show_users(client);
202         }
203         else if(strcmp(request,"wallet") == 0){
204             check_wallet(client);
205         }
206         else if(strcmp(request,"transf") == 0){
207             transfer(client);
208         }
209         bzero(request,bufSize);
210     }
211     close(client.sock);
212 }
213
214 void show_users(user client){
215     int n;
216     char tmp[bufSize];
217     char buffer[bufSize+1];
218     bzero(buffer,bufSize+1);
219     FILE *file;
220     char *fname = "/home/user/project_t/us.txt";
221     file = fopen(fname,"r");
222     if(file == NULL)
223     {
224         perror("ERROR on opening file with users");
225         pthread_exit(0);
226     }
227     while (fgets (tmp, sizeof(tmp), file) != NULL){
228         strncat(buffer,tmp,35);
229         printf("%s", tmp);
230     }
231     printf("\n");
232     fclose(file);
233     n = sendto(client.sock,buffer,sizeof(buffer), 0, (struct
        sockaddr *) &client.cli_addr, client.clilen);
234     if (n < 0)

```

```

235     {
236         pthread_exit(0);
237     }
238 }
239
240 void check_wallet(user client){
241     int n,uid;
242     char buffer[bufSize+1];
243     bzero(buffer,bufSize+1);
244     int i = 0;
245     int j=0;
246     for (i = 0; i <= clientsCount; ++i) {
247         if (connected[i].sock == client.sock){
248             uid=connected[i].uid;
249             connected[i].money=get_money(uid);
250             j=i;
251         }
252     }
253     sprintf(buffer,"%i", connected[j].money);
254     n = sendto(client.sock,buffer,sizeof(buffer), 0, (struct
        sockaddr *) &client.cli_addr, client.clilen);
255     if (n < 0)
256     {
257         pthread_exit(0);
258     }
259 }
260
261 void transfer(user client){
262     int n,value,money;
263     int dest=0;
264     char buffer[bufSize+1];
265     bzero(buffer,bufSize+1);
266     n=recvfrom(client.sock, buffer, bufSize+1, 0, (struct
        sockaddr *) &client.cli_addr, &client.clilen);
267     if(n<0){
268         perror("ERROR_reading_from_socket");
269         pthread_exit(0);
270     }
271     printf("%s\n",buffer);
272     char *tmp=strstr(buffer,"_");
273     value=atoi(tmp);
274     strcpy(tmp,"\0");
275     dest=check_user(buffer);
276     if (dest < 0) {//нет совпадений
277         n = sendto(client.sock,"no_match",8, 0, (struct
            sockaddr *) &client.cli_addr, client.clilen);//зов
орим клиенту, что нет совпадений
278         if (n < 0)
279         {

```

```

280         pthread_exit(0);
281     }
282 }
283 else{
284     money=get_money(dest);
285     money+=value;
286     set_money(dest,money);
287     int i,j;
288     for (i = 0; i <= clientsCount; ++i) {
289         if (connected[i].sock == client.sock){
290             connected[i].money-=value;
291             dest=connected[i].uid;
292             j=i;
293         }
294     }
295     set_money(dest,connected[j].money);
296     n = sendto(client.sock,"ok",2, 0, (struct sockaddr *)
                &client.cli_addr, client.clilen);
297     if (n < 0)
298     {
299         pthread_exit(0);
300     }
301 }
302 }
303
304 int check_user(char buf[]){
305     char name[bufSize+1];
306     char tmp[bufSize+1];
307     char id[10];
308     int res=-1;
309     int k;
310     FILE *file;
311     char *fname = "/home/user/project_t/us.txt";
312     file = fopen(fname,"r");
313     bzero(name,bufSize+1);
314     strcpy(name,buf);
315     if(file == NULL)
316     {
317         perror("ERROR on opening file with users");
318         exit(1);
319     }
320     int i = 0;
321     bzero(tmp,bufSize+1);
322     while(fscanf(file,"%s", tmp)!=EOF){
323         //fscanf(file,"%s", tmp);
324         if(k==0){
325             strcpy(id,tmp);
326             res=atoi(id);
327             printf("%d\n",res);

```

```

328         break;
329     }
330     if(!(i%2)){
331         k=strcmp(name,tmp);
332         printf("%s\n",tmp);
333     }
334     i++;
335 }
336 fclose(file);
337 return res;
338 }
339
340 void add(user client){
341     pthread_mutex_lock(&clientsMutex);
342     connected[clientsCount++]=client;
343     pthread_mutex_unlock(&clientsMutex);
344 }
345
346 void disconnect(int sock){
347     pthread_mutex_lock(&clientsMutex);
348     int i = 0;
349     for (i = 0; i < clientsCount; ++i) {
350         if (connected[i].sock == sock)
351             break;
352     }
353     if (i != clientsCount) {
354         int n = sendto(connected[i].sock, "exit", 4, 0, (
            struct sockaddr *)&connected[i].cli_addr,
            connected[i].clilen);
355         if (n < 0)
356         {
357             perror("ERROR_writing_to_socket");
358             exit(1);
359         }
360         close(sock);
361         for (++i; i < clientsCount; ++i) {
362             connected[i - 1] = connected[i];
363         }
364         --clientsCount;
365     }
366     pthread_mutex_unlock(&clientsMutex);
367 }
368
369 void* server_handler(void*){
370     while (1) {
371         char command[bufSize];
372         bzero(command, bufSize);
373         scanf("%s", command);
374         if (strcmp(command, "show") == 0) {

```

```

375         for (int i = 0; i < clientsCount; ++i) {
376             printf("%d_:_%s\n", connected[i].sock,
                    connected[i].name);
377         }
378     } else if (strcmp(command, "disconnect") == 0) {
379         int sock = 0;
380         scanf("%d", &sock);
381         disconnect(sock);
382     } else {
383         printf("Undefined_ command\n");
384     }
385 }
386 }

```

Файл server.h

```

1 #ifndef SERVER
2 #define SERVER
3 #include "user.h"
4 void *doprocessing(void *c);
5 void show_users(user client);
6 int authentication(user client);
7 int check_user(char buf[]);
8 void check_wallet(user client);
9 void transfer(user client);
10 void *server_handler(void *);
11 void disconnect(int sock);
12 void add(user client);
13 user new_connection(int sockfd);
14 void new_socket(int* sockfd, uint16_t* port);
15 #endif // SERVER

```

Файл user.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <cstdlib>
4 #include "user.h"
5
6 int get_money(int usid){//узнать количество денег, имеющееся
    у пользователя
7     int i=0;
8     int res,tmp=0;
9     int k=-1;
10    char buffer[bufSize+1];
11    FILE *file;
12    char *fname="/home/user/project_t/money.txt";
13    file = fopen(fname,"r");
14    if(file == NULL)
15    {

```

```

16         perror("ERROR_on_openning_file_with_money");
17         exit(1);
18     }
19     bzero(buffer, bufSize+1);
20     while(!feof(file)){
21         fscanf(file, "%s", buffer);
22         tmp=atoi(buffer);
23         if(k==0){
24             res=tmp;
25             break;
26         }
27         if(!(i%2)){
28             if(usid==tmp)
29                 k=0;
30         }
31         i++;
32     }
33     fclose(file);
34     return res;
35 }
36
37 int set_money(int uid, int value){
38     char buf[15];
39     int tmp=0;
40     int before=0;
41     int after=0;
42     int spaces=0;
43     fpos_t pos;
44     FILE *file;
45     char *fname="/home/user/project_t/money.txt";
46     file = fopen(fname, "r+");
47     if(file == NULL)
48     {
49         perror("ERROR_on_openning_file_with_money");
50         exit(1);
51     }
52     int k=1;
53     int i = 0;
54     while(fscanf(file, "%s", buf)!=EOF){
55         if(k==0){
56             before=ftell(file);
57             fsetpos(file, &pos);
58             fprintf(file, "\t%i", value);
59             after=ftell(file);
60             if(before>after) {
61                 spaces += before - after;
62                 while(spaces!=0){
63                     fputc('_', file);
64                     spaces--;

```



```

65         }
66     }
67     fflush(file);
68     break;
69 }
70 fgetpos(file, &pos);
71 if(!(i%2)){
72     tmp=atoi(buf);
73     if(tmp==uid){
74         k=0;
75     }
76 }
77 i++;
78 }
79
80 }
81
82 int set_newid(){//задание нового id пользователя при регистра
    ции
83     int res=0;
84     char uid[15];
85     FILE *file;
86     char *fname = "/home/user/project_t/us.txt";
87     file = fopen(fname, "r");
88     bzero(uid, sizeof(uid));
89     if(file == NULL)
90     {
91         perror("ERROR_on_openning_file_with_users");
92         exit(1);
93     }
94     int i = 0;
95     while(fscanf(file, "%s", uid)!=EOF){
96         if(i%2){
97             res=atoi(uid);
98         }
99         i++;
100     }
101     fclose(file);
102     return res+1;
103 }

```

Файл user.h

```

1 #ifndef USER
2 #define USER
3 #include <netdb.h>
4 #include <netinet/in.h>
5 #include <unistd.h>
6 #define bufSize 255
7 #define maxClients 100

```

```

8 typedef struct{
9     char name[bufSize];
10    int sock;
11    int uid;
12    int money;
13    struct sockaddr_in cli_addr;
14    socklen_t clilen;
15 }user;
16
17 //user();
18 int set_newid();
19 int get_money(int usid);
20 int set_money(int uid, int value);
21 #endif // USER

```

UDP клиент

Файл client.cpp

```

1  //client
2  #include <winsock2.h>
3  #include <ws2tcpip.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <string.h>
7
8  #pragma comment (lib, "Ws2_32.lib")
9  #pragma comment (lib, "Mswsock.lib")
10 #pragma comment (lib, "AdvApi32.lib")
11
12 #define bufSize 255
13
14 typedef struct {
15     int sockfd;
16     struct sockaddr_in serv_addr;
17     int clilen;
18 } Uclient;
19
20 int authentication(Uclient client);
21 void showUsers(char command[], Uclient client);
22 void checkWallet(Uclient client);
23 void transfer(Uclient client);
24 int disconnect(int sockfd, char* buf);
25
26 int main(int argc, char *argv[]) {
27     WORD wVersionRequested = MAKEWORD(2, 2);           // Stuff
28     for WSA functions
29     WSADATA wsaData;

```

```

29     int sock, portno, n=0;
30     int aut;
31     struct sockaddr_in serv_addr;
32     struct hostent *server;
33     char quit[]="quit";
34     char show[]="show_users";
35     char wallet[]="check_wallet";
36     char transf[]="transfer";
37     char buffer[bufSize+1];
38     char buf[bufSize];
39
40     WSStartup(wVersionRequested, &wsaData);
41     //portno=12345;
42     if (argc < 3) {
43         fprintf(stderr, "usage_%s_hostname_port\n", argv[0]);
44         exit(0);
45     }
46
47     portno = atoi(argv[2]);
48
49     /* Create a socket point */
50     sock = socket(AF_INET, SOCK_DGRAM, 0);
51
52     if (sock == SOCKET_ERROR) {
53         perror("ERROR_opening_socket");
54         exit(1);
55     }
56
57     server = gethostbyname(argv[1]);
58     if (server == NULL) {
59         fprintf(stderr, "ERROR_no_such_host\n");
60         exit(0);
61     }
62
63     memset((char *) &serv_addr, 0, sizeof(serv_addr));
64     serv_addr.sin_family = AF_INET;
65     serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
66     //strncpy((char *)server->h_addr, (char *)&serv_addr.
67         sin_addr.s_addr, server->h_length);
68     serv_addr.sin_port = htons(portno);
69     int len = sizeof(serv_addr);
70
71     n = sendto(sock, "client", strlen("client"), 0, (struct
62         sockaddr *) &serv_addr, len);
72     if (n < 0) {
73         perror("ERROR_writing_to_socket");
74         closesocket(sock);
75         exit(1);

```

```

76     }
77
78     //recieve new port
79     memset(buf, 0, bufSize);
80     n = recvfrom(sock, buf, bufSize, 0, (struct sockaddr *) &
        serv_addr, &len);
81     if (n < 0) {
82         perror("ERROR reading from socket");
83         closesocket(sock);
84         exit(1);
85     }
86     disconnect(sock, buf);
87     //close old socket
88     closesocket(sock);
89     WSACleanup();
90
91     int newport = atoi(buf);
92
93     WSADATA wsa2;
94     if (WSAStartup(MAKEWORD(2, 2), &wsa2) != 0) {
95         exit(EXIT_FAILURE);
96     }
97
98     struct sockaddr_in new_addr;
99     int sockfd, new_slen = sizeof(new_addr);
100
101     //create socket
102     if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) ==
        SOCKET_ERROR) {
103         exit(EXIT_FAILURE);
104     }
105
106     //setup address structure
107     memset((char *)&new_addr, 0, new_slen);
108     new_addr.sin_family = AF_INET;
109     new_addr.sin_port = htons(newport);
110     new_addr.sin_addr.S_un.S_addr = inet_addr(argv[1]);
111
112     n = sendto(sockfd, "newclient", strlen("newclient"), 0, (
        struct sockaddr *) &new_addr, new_slen);
113     if (n < 0) {
114         perror("ERROR reading from socket");
115         closesocket(sockfd);
116         exit(1);
117     }
118
119     Uclient client;
120     client.sockfd = sockfd;
121     client.serv_addr = new_addr;

```

```

122     client.clilen = sizeof(new_addr);
123
124     do {
125         aut = authentication(client); //процесс аутентификации клиента
126     } while (aut < 0);
127     while (1){
128         printf("Enter the command: ");
129         memset(buffer, 0, bufSize+1);
130         fgets(buffer, bufSize+1, stdin);
131
132         if(strncmp(buffer, quit, sizeof(quit)-1) == 0){
133             n = sendto(client.sockfd, buffer, strlen(buffer),
134                        0, (struct sockaddr*)&client.serv_addr,
135                        client.clilen);
136             if (n < 0) {
137                 perror("ERROR writing to socket");
138                 exit(1);
139             }
140             closesocket(client.sockfd);
141             break;
142         }
143         else if(strncmp(buffer, show, sizeof(show)-1) == 0){
144             showUsers(buffer, client);
145         }
146         else if(strncmp(buffer, wallet, sizeof(wallet)-1) == 0){
147             checkWallet(client);
148         }
149         else if(strncmp(buffer, transf, sizeof(transf)-1) == 0){
150             transfer(client);
151         }
152         else{
153             printf("Undefined command\n");
154         }
155     }
156     return 0;
157 }
158
159 void showUsers(char command[], Uclient client){
160     int n;
161     char buffer[bufSize+1];
162     memset(buffer, 0, bufSize);
163     strcpy(buffer, command);
164     n = sendto(client.sockfd, buffer, strlen(buffer), 0, (
165                struct sockaddr*)&client.serv_addr, client.clilen);
166     if (n < 0) {
167         perror("ERROR writing to socket");
168         closesocket(client.sockfd);
169         exit(1);

```

```

167     }
168
169     /* Now read server response */
170     memset(buffer, 0, bufSize);
171     n = recvfrom(client.sockfd, buffer, bufSize+1, 0, (struct
        sockaddr*)&client.serv_addr, &client.clilen);
172     if (n < 0) {
173         perror("ERROR_␣reading_␣from_␣socket");
174         closesocket(client.sockfd);
175         exit(1);
176     }
177     disconnect(client.sockfd, buffer);
178     printf("%s\n",buffer);
179 }
180
181 void checkWallet(Uclient client){
182     int n;
183     char buffer[bufSize+1];
184     n=sendto(client.sockfd, "wallet", 6, 0, (struct sockaddr
        *)&client.serv_addr, client.clilen);
185     if (n < 0) {
186         perror("ERROR_␣writing_␣to_␣socket");
187         closesocket(client.sockfd);
188         exit(1);
189     }
190     memset(buffer, 0, bufSize+1);
191     n = recvfrom(client.sockfd, buffer, bufSize+1, 0, (struct
        sockaddr*)&client.serv_addr, &client.clilen);
192     if (n < 0) {
193         perror("ERROR_␣reading_␣from_␣socket");
194         closesocket(client.sockfd);
195         exit(1);
196     }
197     disconnect(client.sockfd, buffer);
198     printf("%s\n",buffer);
199 }
200
201 void transfer(Uclient client){
202     int n;
203     char tmp[bufSize];
204     char buffer[bufSize+1];
205     memset(buffer, 0, bufSize+1);
206     n=sendto(client.sockfd, "transf", 6, 0, (struct sockaddr
        *)&client.serv_addr, client.clilen);
207     if (n < 0) {
208         perror("ERROR_␣writing_␣to_␣socket");
209         closesocket(client.sockfd);
210         exit(1);
211     }

```

```

212     printf("To_who_and_how_much_do_you_want_transfer_money?\n
        ");
213     scanf("%s",buffer);
214     scanf("%s",tmp);
215     strcat(buffer,"_");
216     strcat(buffer,tmp);
217     n=sendto(client.sockfd, buffer, bufSize+1, 0, (struct
        sockaddr*)&client.serv_addr, client.clilen);
218     memset(buffer, 0, bufSize+1);
219     n=recvfrom(client.sockfd, buffer, bufSize+1, 0, (struct
        sockaddr*)&client.serv_addr, &client.clilen);
220     if (n < 0) {
221         perror("ERROR_reading_from_socket");
222         closesocket(client.sockfd);
223         exit(1);
224     }
225     disconnect(client.sockfd, buffer);
226     if(strcmp(buffer,"no_match")==0){
227         printf("There_is_no_user_with_such_username\n");
228     }
229     else if(strcmp(buffer,"ok")==0){
230         printf("Operation_done\n");
231     }
232     else{
233         printf("Some_error_occurs_during_the_operation\n");
234     }
235 }
236
237 int disconnect(int sockfd, char* buf){
238     if (strcmp(buf, "exit") == 0) {
239         printf("Disconnected_from_server\n");
240         closesocket(sockfd);
241         exit(1);
242     }
243     else return -1;
244 }
245
246 int authentication(Uclient client){
247     char login[bufSize +1];
248     char reg[bufSize +1];
249     char buffer[bufSize +1];
250     char ok[]="ok";
251     char sign[]="sign_in";
252     char registration[]="register";
253     int n;
254     int res=-1;
255     memset(buffer, 0, bufSize+1);
256     printf("Sign_in_or_register\n");
257     fgets(buffer, bufSize+1, stdin);

```

```

258 //scanf("%s", &buffer);
259 if(strncmp(buffer,sign,sizeof(sign)-1) == 0){//вход сущес
    твующего пользователя
260     printf("Enter_your_username:\n");
261     scanf("%s", &login);
262     n=sendto(client.sockfd,"exist",bufSize, 0, (struct
        sockaddr*)&client.serv_addr, client.clilen);//посы
        лка серверу сообщения о том, что входит существующ
        ий пользователь
263     if (n < 0) {
264         perror("ERROR_writing_to_socket");
265         closesocket(client.sockfd);
266         exit(1);
267     }
268     n=sendto(client.sockfd,login,bufSize, 0, (struct
        sockaddr*)&client.serv_addr, client.clilen);//посы
        лка серверу имени пользователя
269     if (n < 0) {
270         perror("ERROR_writing_to_socket");
271         closesocket(client.sockfd);
272         exit(1);
273     }
274     memset(buffer, 0, bufSize+1);
275     n = recvfrom(client.sockfd, buffer, bufSize+1, 0, (
        struct sockaddr*)&client.serv_addr, &client.clilen
        );//ответ от сервера, правильны ли данные или нет
276     if (n < 0) {
277         perror("ERROR_reading_from_socket");
278         closesocket(client.sockfd);
279         exit(1);
280     }
281     disconnect(client.sockfd, buffer);
282     if(strncmp(buffer,ok,sizeof(ok)-1) == 0){//данные пра
        вильны
283         printf("Hello%s, you has successfully logged\n"
            ,login);
284         res=1;
285     }
286     else{//данные не правильны
287         printf("No such username. Type correct username
            or register\n");
288         res=-1;
289     }
290 }
291 else if(strncmp(buffer,registration,sizeof(registration)
    -1) == 0){//регистрация нового пользователя
292     printf("Create new username:\n");
293     scanf("%s", &reg);
294     n=sendto(client.sockfd,"new",bufSize, 0, (struct

```



```

        sockaddr*)&client.serv_addr, client.clilen);//посы
        лка сообщения серверу о регистрации нового пользов
        ателя
295     if (n < 0) {
296         perror("ERROR_writing_to_socket");
297         closesocket(client.sockfd);
298         exit(1);
299     }
300     n=sendto(client.sockfd,reg,bufSize, 0, (struct
        sockaddr*)&client.serv_addr, client.clilen);//посы
        лка имени нового пользователя
301     if (n < 0) {
302         perror("ERROR_writing_to_socket");
303         closesocket(client.sockfd);
304         exit(1);
305     }
306     memset(buffer, 0, bufSize+1);
307     n = recvfrom(client.sockfd, buffer, bufSize+1, 0, (
        struct sockaddr*)&client.serv_addr, &client.clilen
        );//ответ от сервера, правильны ли данные или нет
308     if (n < 0) {
309         perror("ERROR_reading_from_socket");
310         closesocket(client.sockfd);
311         exit(1);
312     }
313     disconnect(client.sockfd, buffer);
314     if(strncmp(buffer,ok,sizeof(ok)-1) == 0){//данные пра
        вильны
315         printf("Hello%s, you has successfully registered
            and logged\n",reg);
316         res=1;
317     }
318     else{//данные не правильны
319         printf("User with this username is already
            existing. Create other username\n");
320     }
321 }
322 return res;
323 }

```