

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе №1
по дисциплине «Сети ЭВМ и телекоммуникации»
Система обмена сообщениями

Работу выполнил:
студент гр. 43501/3
Я. В. Хуторной

Санкт - Петербург
2015

Глава 1

Задание

Разработать приложение-клиент и приложение сервер, обеспечивающие функции мгновенного обмена сообщений между пользователями.

1.1 Функциональные требования

Серверное приложение должно реализовывать следующие функции:

- 1) Прослушивание определенного порта
- 2) Обработка запросов на подключение по этому порту от клиентов
- 3) Поддержка одновременной работы нескольких клиентов через механизм нитей
- 4) Передача текстового сообщения одному клиенту
- 5) Передача текстового сообщения всем клиентам
- 6) Прием и ретрансляция входящих сообщений от клиентов
- 7) Обработка запроса на отключение клиента
- 8) Принудительное отключение указанного клиента

Клиентское приложение должно реализовывать следующие функции:

- 1) Установление соединения с сервером
- 2) Передача сообщения всем клиентам
- 3) Передача сообщения указанному клиенту
- 4) Прием сообщения от сервера с последующей индикацией
- 5) Разрыв соединения
- 6) Обработка ситуации отключения клиента сервером

Настройки приложений. Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени сервера сообщений и номера порта сервера.

Методика тестирования. Для тестирования приложений запускается один экземпляр серверного приложения и несколько экземпляров клиентских. В процессе тестирования проверяются основные возможности приложений по передаче-приёму сообщений, ситуации самостоятельного и принудительного отключения клиента.

1.2 Нефункциональные требования

При подключении нового клиента должен производиться запрос имени пользователя.

Приложение может быть выполнено с использованием одного из двух протоколов: TCP или UDP, а также с различным выбором операционных систем для программы - клиента и программы-сервера. Все комбинации вариантов протоколов обмена и распределения клиент-серверных функций приведены в табл. 1.

№	Сетевой протокол	Серверная операционная система	Клиентская операционная система
1	TCP	Linux	Windows
2	TCP	Windows	Linux
3	UDP	Linux	Windows
4	UDP	Windows	Linux

Табл. 1. Варианты протоколов обмена и распределения клиент-серверных функций

1.3 Ограничения

Серверное приложение не поддерживает одновременное подключение более 32 клиентов.

Имя пользователя не должно быть длиннее 30 символов.

Глава 2

Реализация для работы по протоколу TCP

2.1 Прикладной протокол

Формат передаваемого сообщения представлен на рис. 1. Оно содержит 6 полей.

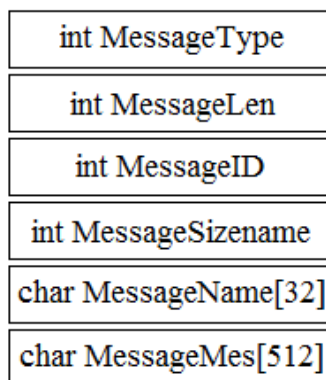


Рис. 1. Формат передаваемого сообщения

Поле MessageType предназначено для передачи типа сообщения и может принимать следующие значения:

- 1) MessageType = 1. Задание/изменение имени пользователя.
- 2) MessageType = 2. Запрос списка подключенных к серверу клиентов.
- 3) MessageType = 0. Обычная пересылка сообщения.

Поле MessageID предназначено для передачи ID-номера того клиента, которому требуется передать сообщение.

Поле MessageName предназначено для передачи имени отправителя.

Поле MessageSizename предназначено для передачи длины имени клиента.

Поле MessageMes предназначено для передачи сообщения.

Поле MessageLen предназначено для передачи длины передаваемого сообщения.

2.2 Тестирование и описание работы приложений

2.2.1 Описание тестового стенда и методики тестирования

Серверное приложение запускается на ОС Windows 7. Клиентские приложения запускаются на ОС Windows 7 и ОС Linux на разных компьютерах.

2.2.2 Тестовый план, описание работы приложений и результаты тестирования

Процесс тестирования:

- 1) Проверка на корректность ввода IP – адреса. Ввод некорректного IP – адреса (сервер).

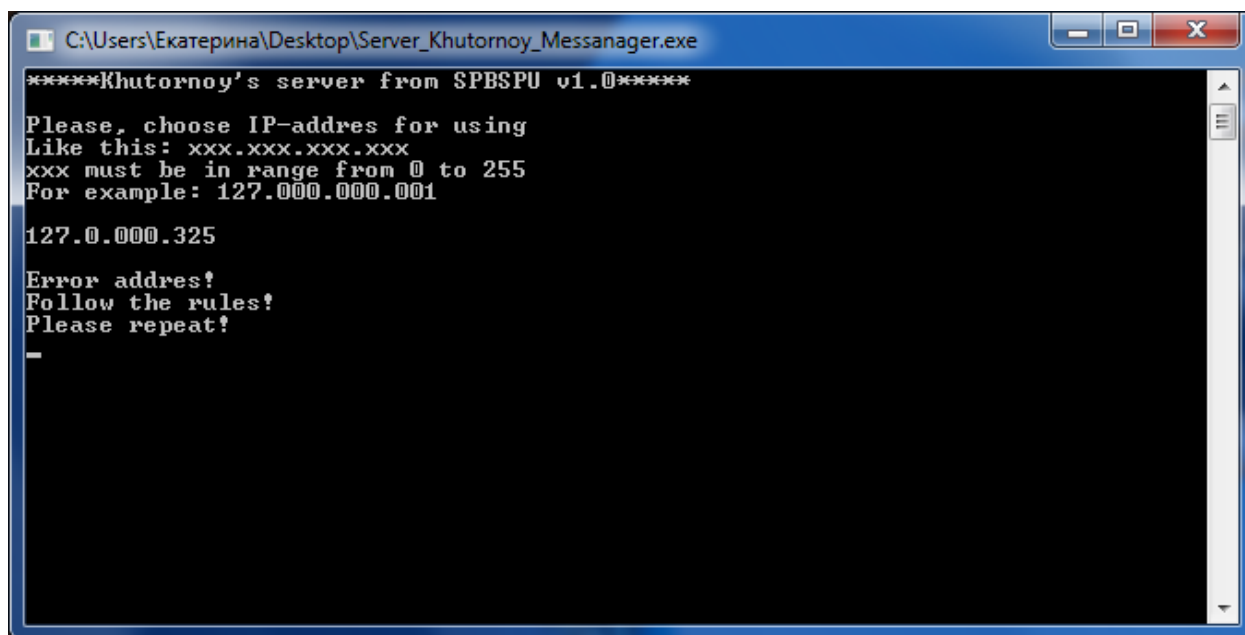


Рис. 2. Проверка корректности ввода IP-адреса.

После запуска приложение просит пользователя ввести IP – адрес. Приложение показывает требуемый формат вводимого IP – адреса. При несоблюдении правил ввода приложение выводит сообщение об ошибке и просит ввести адрес заново.

- 2) Ввод корректного IP – адреса (сервер).

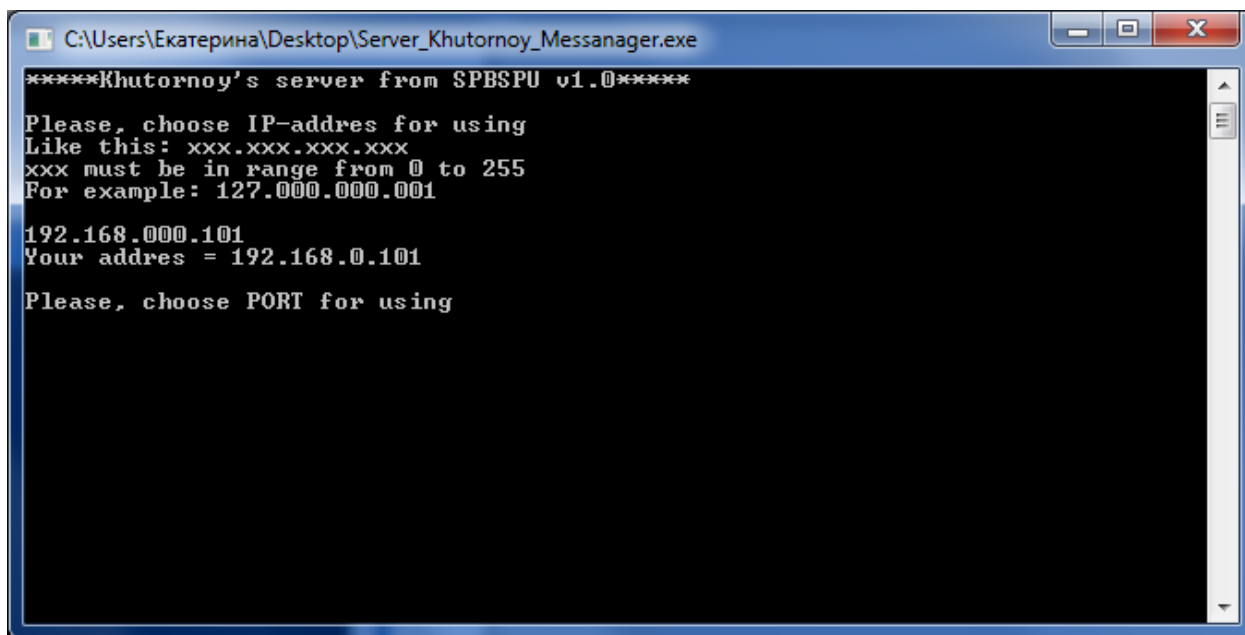


Рис. 3. Ввод корректного IP – адреса

После корректного ввода IP – адреса приложение предлагает ввести номер порта.

- 3) Ввод номера порта (сервер).

После успешного ввода IP – адреса приложение просит ввести номера порта. Если введенный номер порта некорректный, например не в диапазоне от 0 до 65535, то приложение выбирает номер порта по умолчанию – 8888. Результат работы сервера после ввода IP – адреса и номера порта представлен на рис. 4.

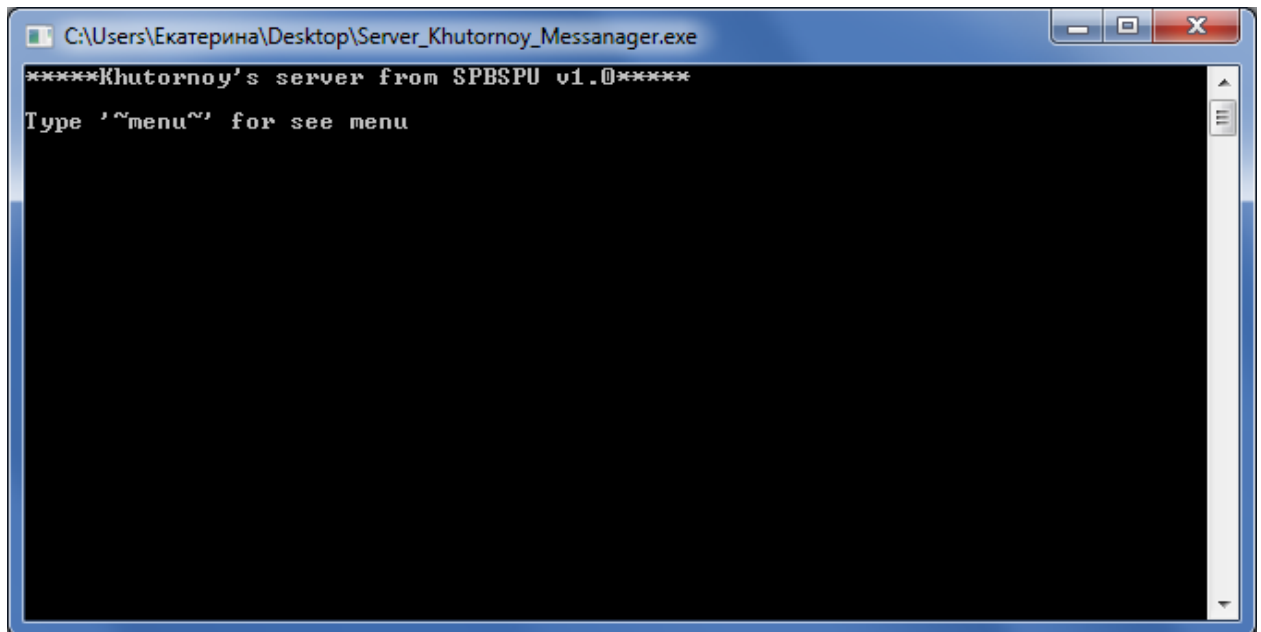


Рис. 4. Результат работы сервера после ввода порта

- 4) Запуск приложения - клиента. Ввод IP – адреса и номера порта сервера.

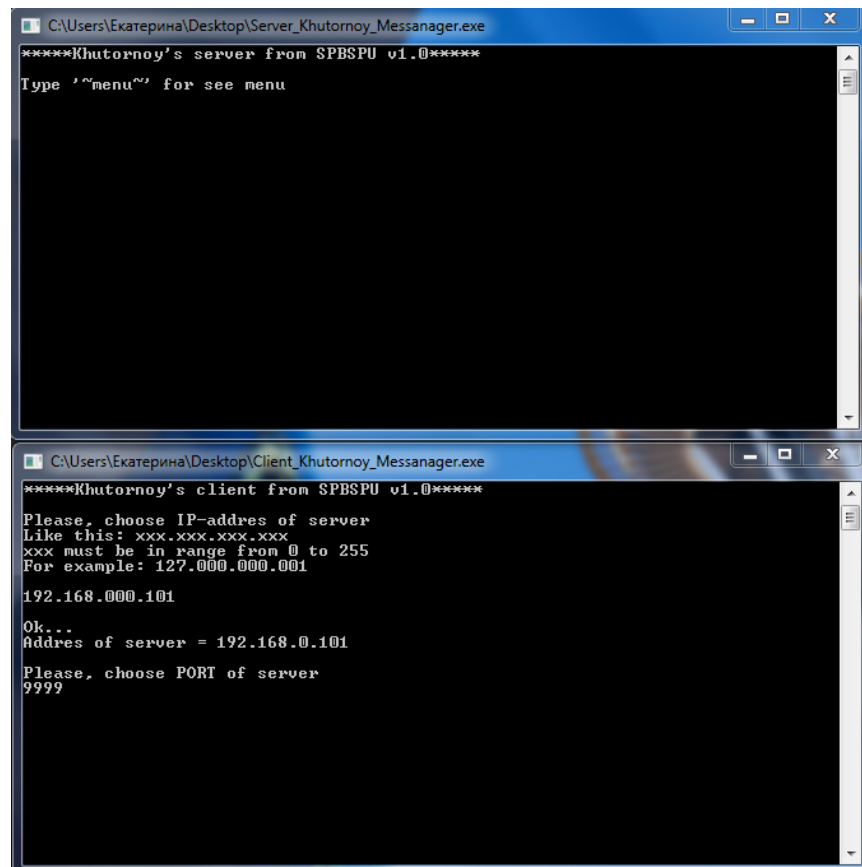


Рис. 5. Результат работы приложения-сервера и приложения-клиента

После запуска приложения-клиента требуется ввести IP – адрес и номер порта сервера. Приложение-клиент проверяет введенные значения на корректность точно так же, как было рассмотрено выше для приложения – сервера.

- 5) Подключение клиента к серверу. Запрос приложения на ввод имени пользователя.

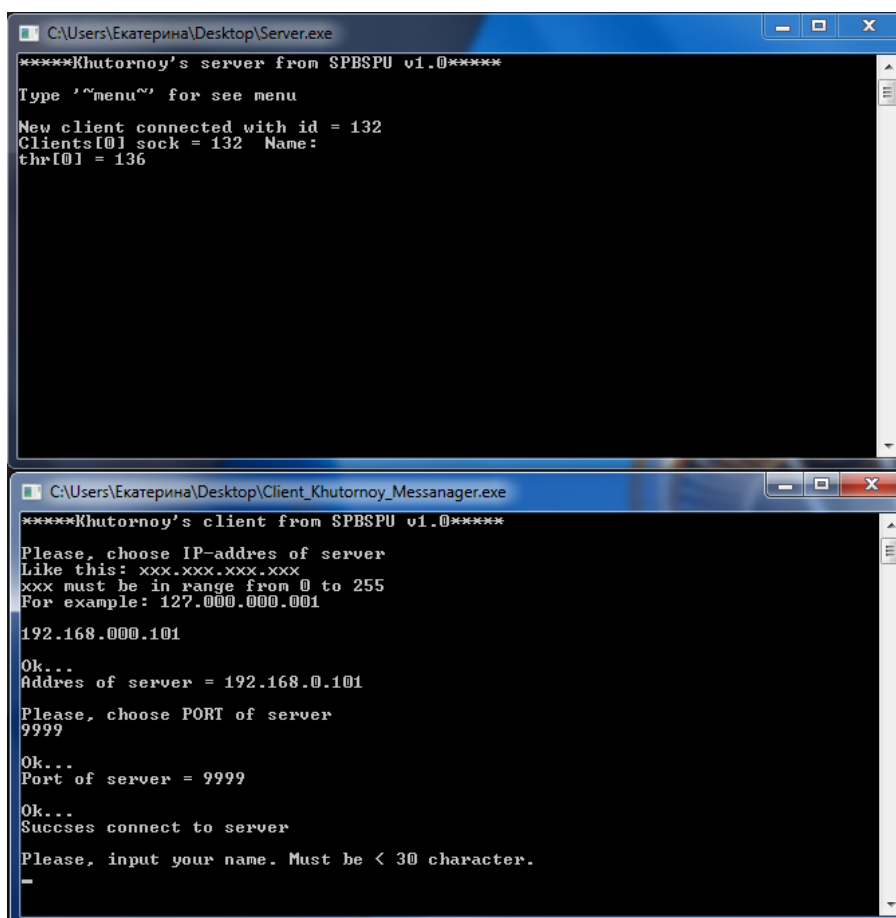


Рис. 6. Результат работы приложения-сервера и приложения-клиента

После успешного подключения клиента к серверу на стороне приложения – сервера отображается сообщение о подключении нового клиента, его уникальный номер (номер сокета), номер потока, обслуживающего данного клиента и имя клиента. На стороне приложения – клиента программа запрашивает имя пользователя.

- 6) Ввод имени пользователя на стороне приложения – клиента.

Приложение – клиент после ввода имени пользователя очищает экран и выводит сообщение о том, что при вводе текста *~menu~* в стандартный поток ввода, пользователь зайдет в меню приложения. На стороне сервера приложение отображает поля принятого сообщения. Как видно на рис. 7. поле MessageType принятого сообщения равно 1. Это означает, что клиент запросил изменение своего имени.

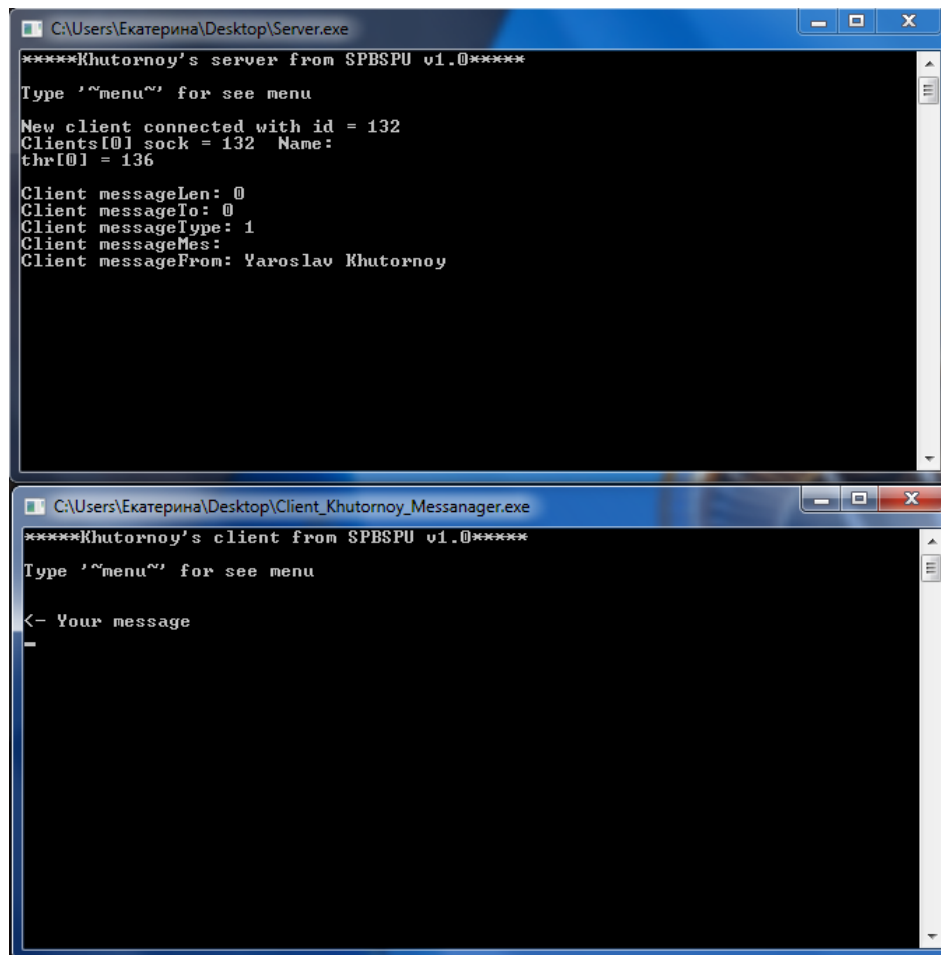


Рис. 7. Результат работы приложения-сервера и приложения-клиента

7) Вход в меню приложения на стороне приложения – клиента.

После ввода текста ~меню~ в стандартный поток ввода приложения, приложение отображает меню. В меню допустимы три вида действий:

1. Выбрать ID – номер клиента для переписки.
2. Отобразить список подключенных к серверу клиентов.
3. Выйти из меню.

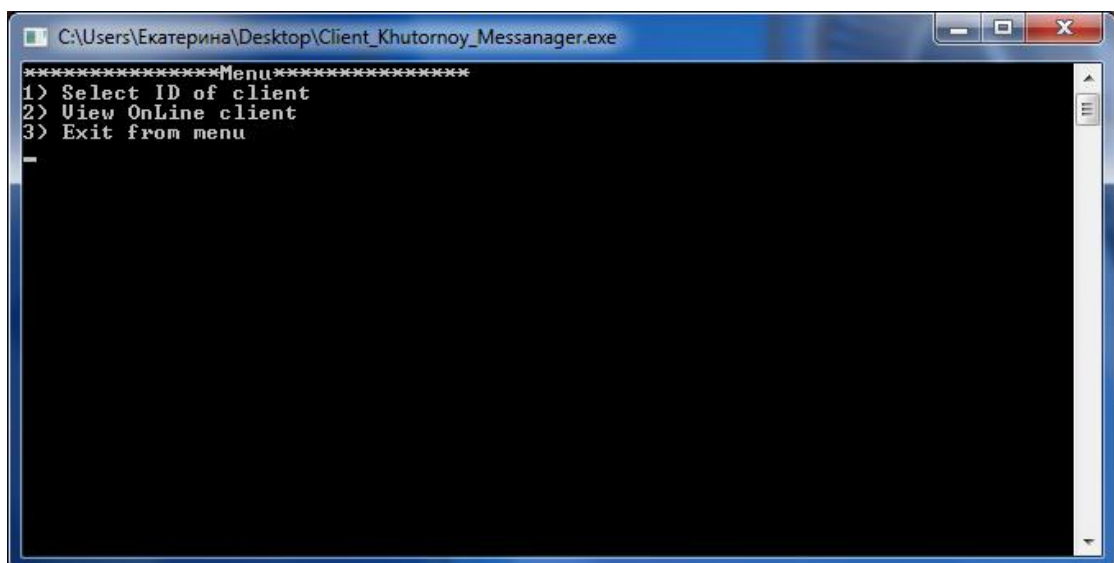


Рис. 8. Меню приложения – клиента

8) Отображение подключенных к серверу клиентов на стороне приложения – клиента.

Для проведения данного теста к серверу были подключены 5 клиентов. На стороне одного из приложений – клиентов была выбрана функция отображения списка онлайн клиентов.

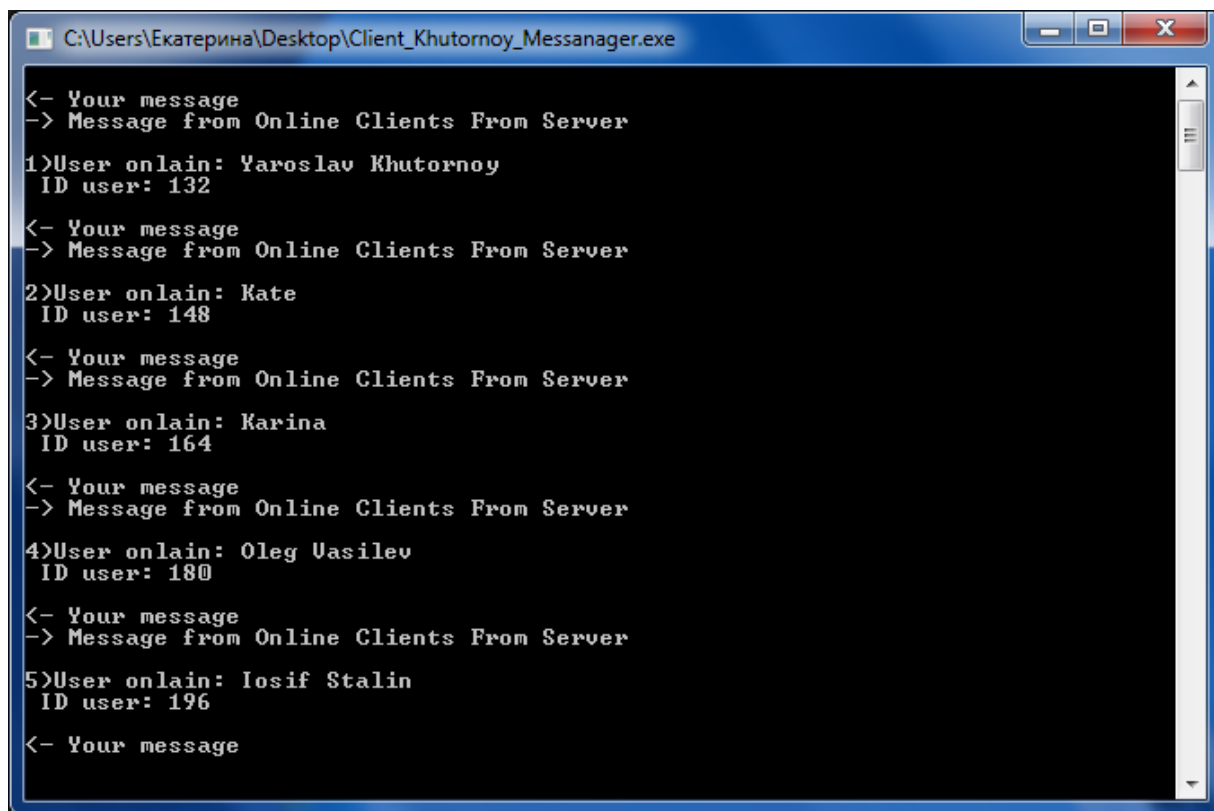


Рис. 9. Отображения списка подключенных к серверу клиентов на стороне приложения – клиента.

9) Отображение подключенных к серверу клиентов на стороне приложения – сервера.

Приложение – сервер позволяет отображать список подключенных клиентов. Для этого требуется зайти в меню и выбрать функцию отображения подключенных клиентов.

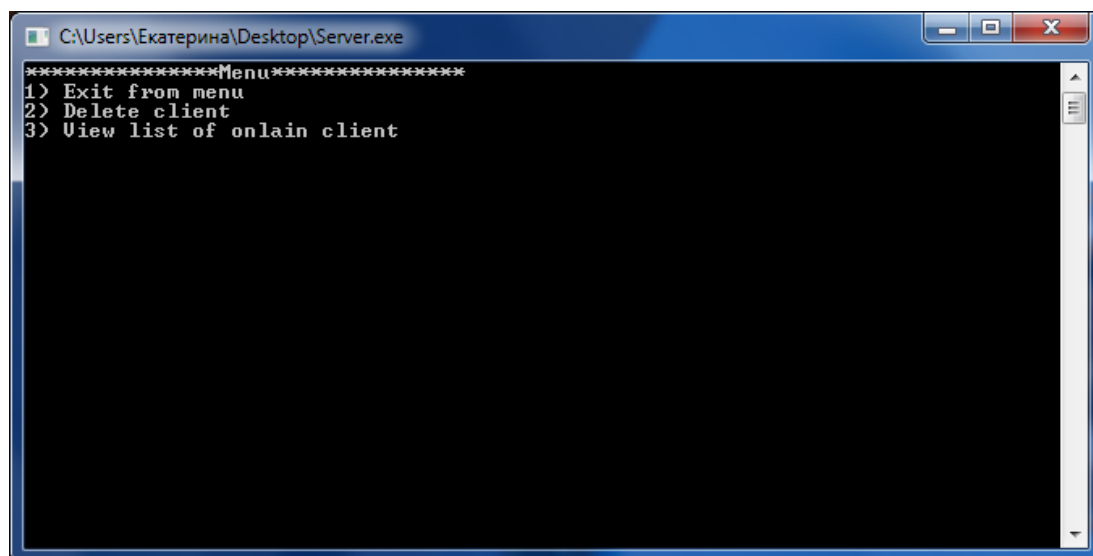


Рис. 10. Отображение меню на стороне сервера

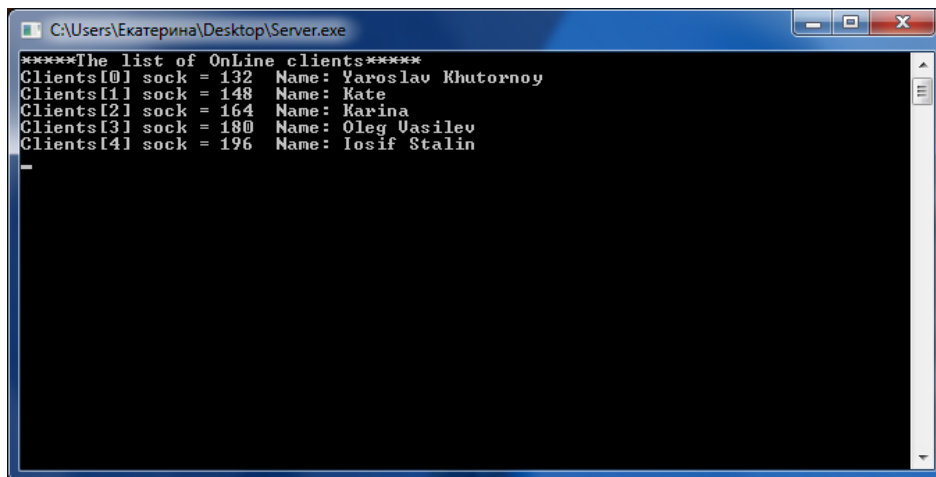


Рис. 11. Отображение списка подключенных к серверу клиентов.

10) Принудительное отключение клиентов со стороны сервера.

Приложение – сервер позволяет принудительно отключать клиентов. Для этого нужно зайти в меню и выбрать функцию номер 2 – Delete client.

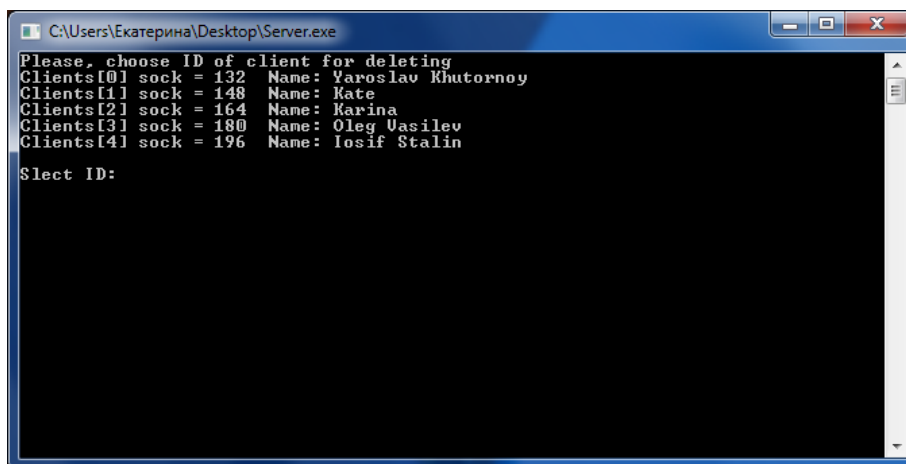


Рис. 12. Отображение меню на стороне сервера

После выбора этой функции приложение предлагает выбрать ID – номер клиента, которого требуется отключить. Выберем номер 4 и удалим клиента Iosif Stalin.

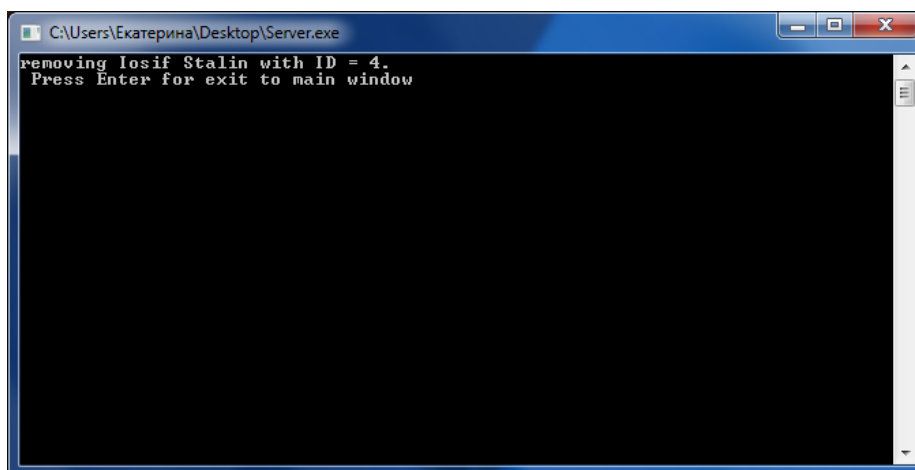


Рис. 13. Результат работы сервера при удалении клиента

Проверим, что клиент действительно удалился, зайдя в меню и выбрав функцию отображения списка подключенных клиентов на стороне одного из приложений-клиентов.

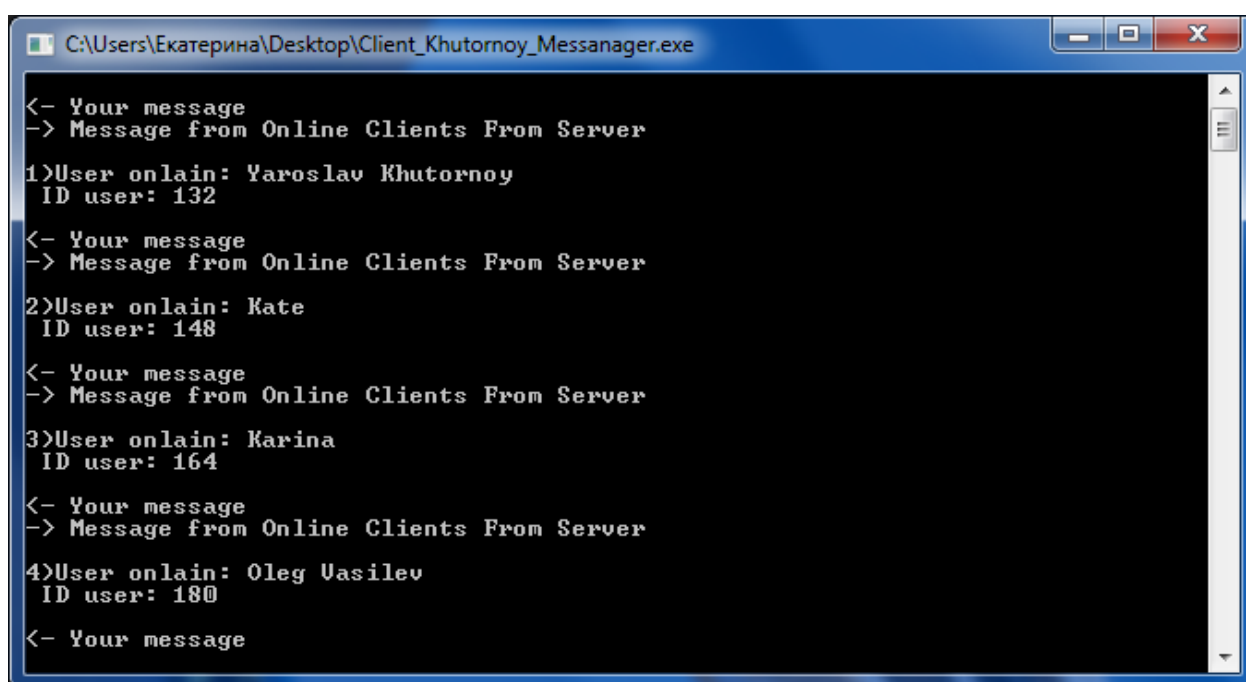


Рис. 14. Результат работы приложения – клиента

Как видим, клиент с именем Iosif Stalin больше не отображается в списке подключенных к серверу клиентов.

На стороне приложения – клиента, которое было отключено сервером выводится следующее сообщение.

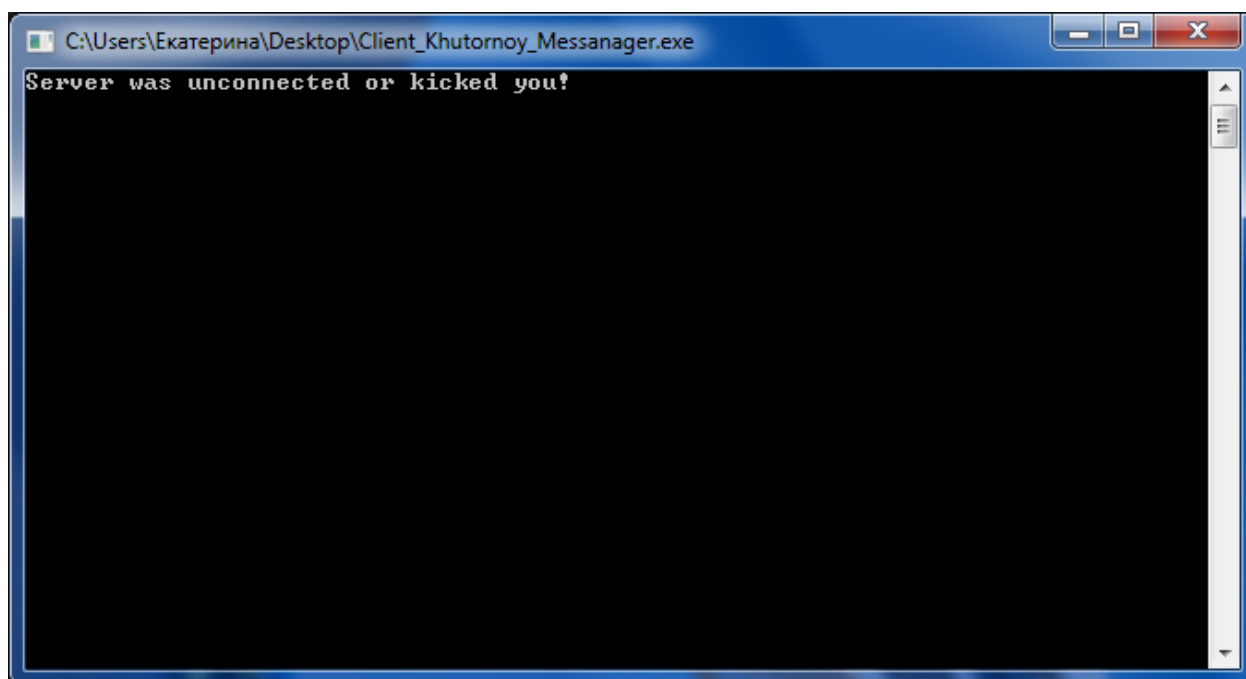
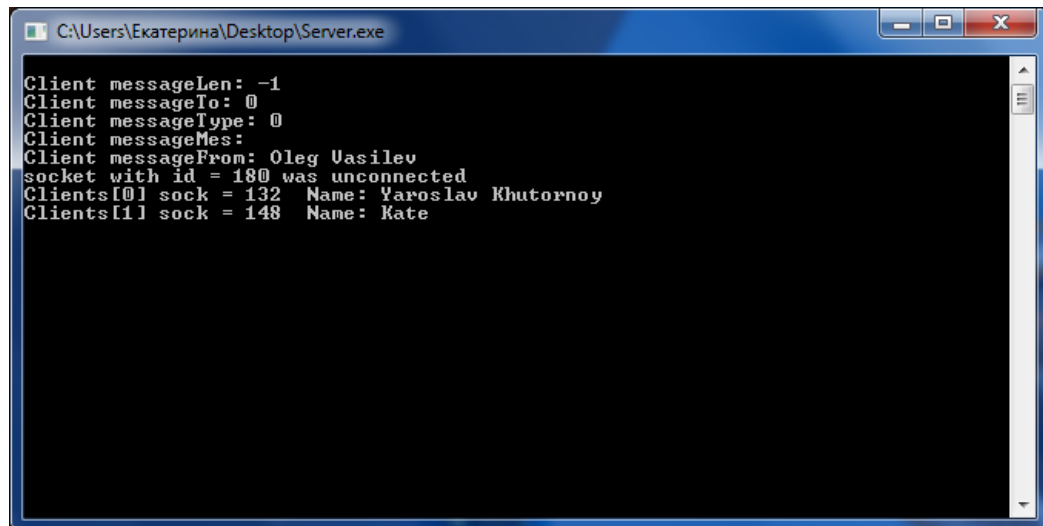


Рис. 15. Результат работы приложения – клиента при принудительном отключении его от сервера.

11) Отключение клиента на стороне приложения – клиента.

При завершении приложения – клиента программа – сервер выводит сообщение о том, что данный клиент был отключен и выводит обновленный список подключенных клиентов.

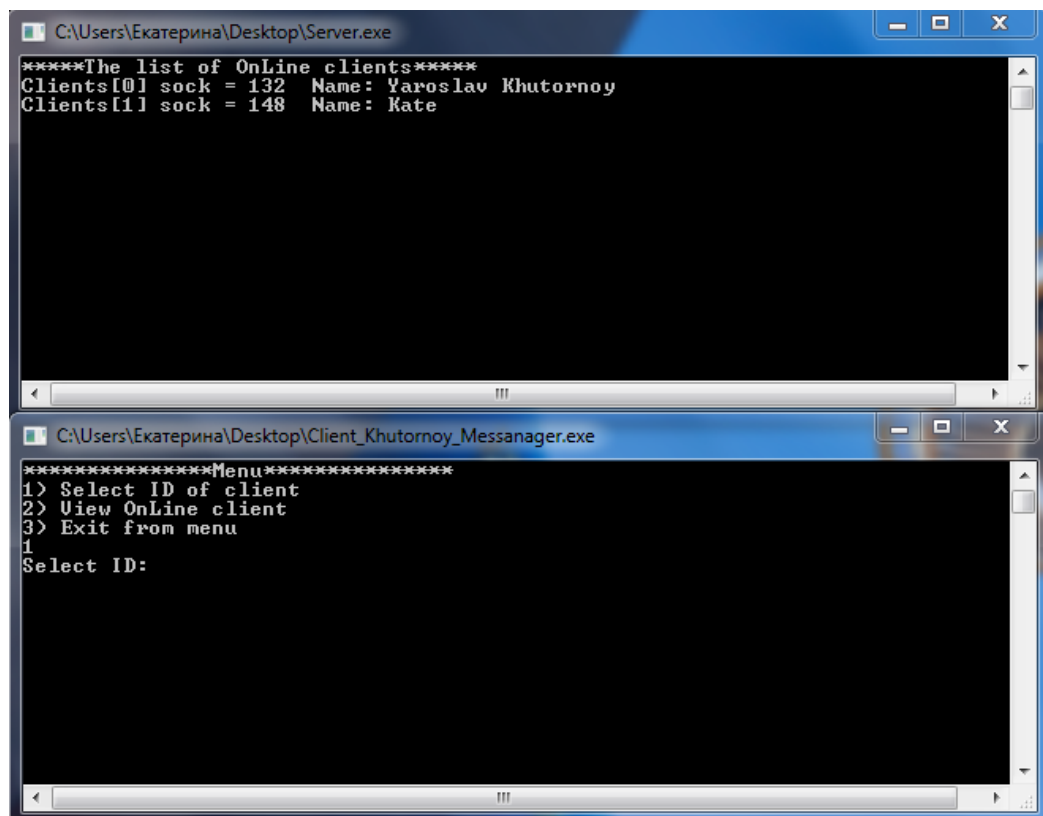


```
C:\Users\Екатерина\Desktop\Server.exe
Client messageLen: -1
Client messageId: 0
Client messageType: 0
Client messageMes:
Client messageFrom: Oleg Uasilev
socket with id = 180 was unconnected
Clients[0] sock = 132 Name: Yaroslav Khutornoy
Clients[1] sock = 148 Name: Kate
```

Рис. 16. Результат работы сервера при самостоятельном отключении клиента.

12) Выбор адресата на стороне приложения – клиента и отправка сообщения.

Выбор адресата производится путем запуска меню программы и выбора функции номер 1 - Select ID of client.



```
C:\Users\Екатерина\Desktop\Server.exe
*****The list of OnLine clients*****
Clients[0] sock = 132 Name: Yaroslav Khutornoy
Clients[1] sock = 148 Name: Kate

C:\Users\Екатерина\Desktop\Client_Khutornoy_Messenger.exe
*****Menu*****
1> Select ID of client
2> View OnLine client
3> Exit from menu
1
Select ID:
```

Рис. 17. Результат работы приложения-сервера и приложения-клиента

На стороне клиента Yaroslav Khutornoy выберем ID – номер клиента Kate, после чего попробуем отправить выбранному адресату сообщение.

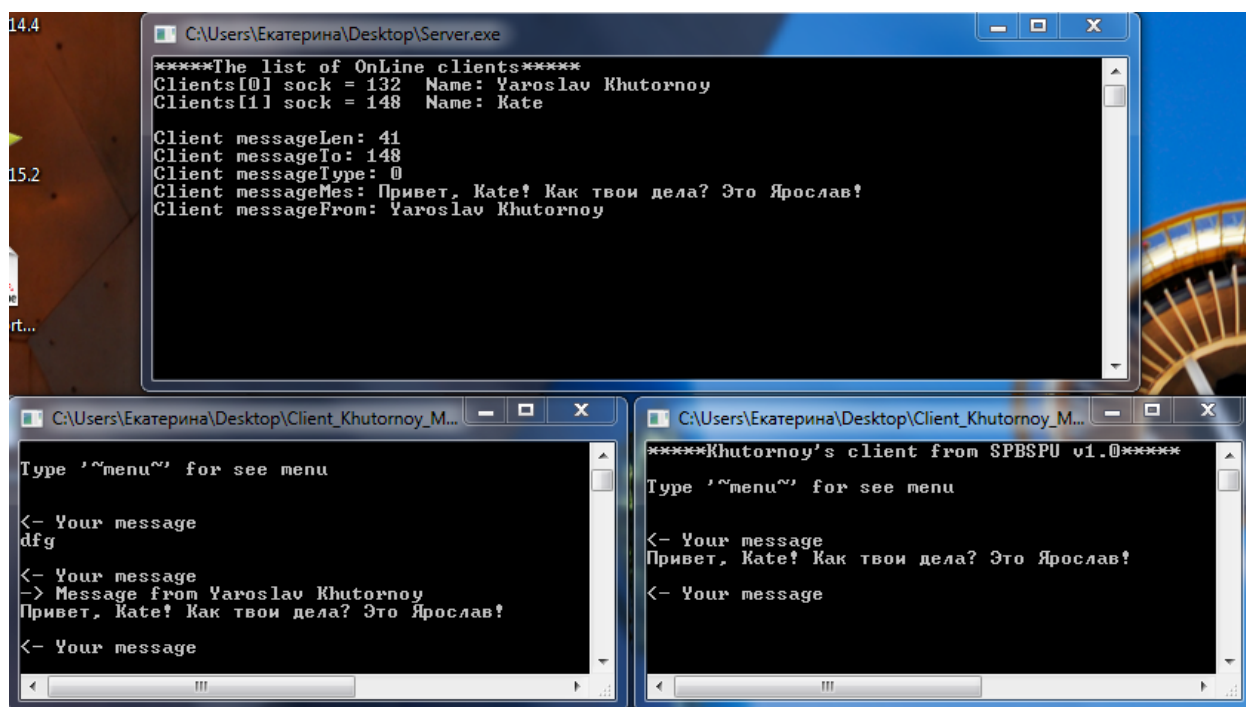


Рис. 18. Результат работы приложения-сервера и приложений-клиентов.

Мы отправили со стороны клиента Yaroslav Khutornoy сообщение клиенту Kate. На стороне клиента Kate сообщение успешно принято. Отобразился текст сообщения и имя отправителя. На стороне сервера видим отправленное и ретранслированное сообщение от клиента Yaroslav Khutornoy.

Тестирование для приложений на ОС Linux аналогично тестированию приложений на ОС Windows.

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

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

3.2 Тестирование и описание работы приложений

3.2.1 Описание тестового стенда и методики тестирования

Серверное приложение запускается на ОС Linux. Клиентские приложения запускаются на ОС Linux на разных компьютерах.

3.2.2 Тестовый план, описание работы приложений 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 с помощью утилиты netem.

- 1) Потеря 50% пакетов при подключении клиента к серверу

```
tc qdisc add dev eth0 root netem loss 50%
```

Результаты:

Когда пакеты не терялись, клиент спокойно подключался к серверу. При потерях пакетов при подключении клиента к серверу, приложение-клиент показывает, что подключение к серверу состоялось, однако информация на сервере это не подтверждает. Приложение-сервер не обнаружил новых подключений. В данном эксперименте к серверу подключались два клиента. В результате один подключился успешно, второй подключиться не смог.

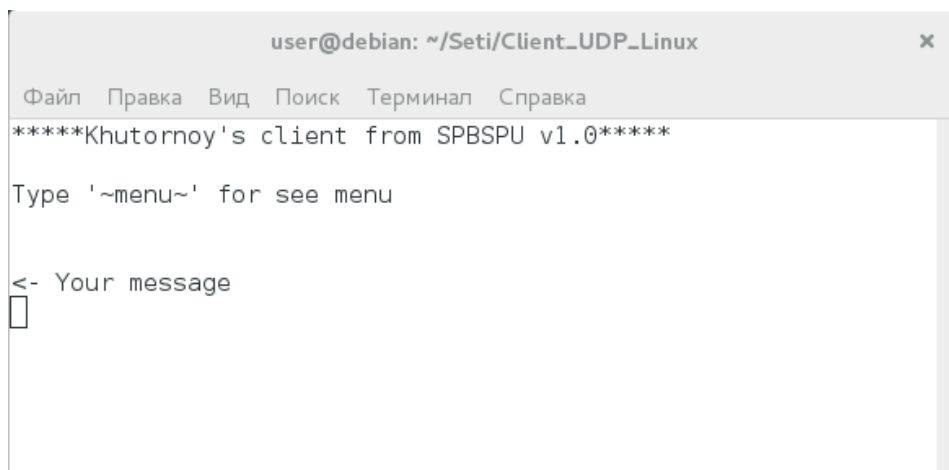


Рис. 19. Результат работы программы – клиента при подключении к серверу при потере пакетов

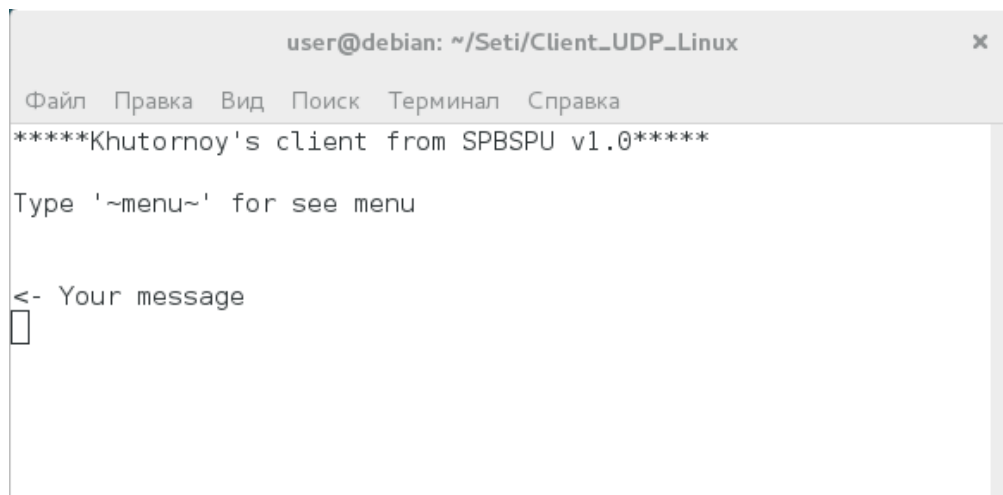


Рис. 20. Результат работы программы – клиента при подключении к серверу при потере пакетов

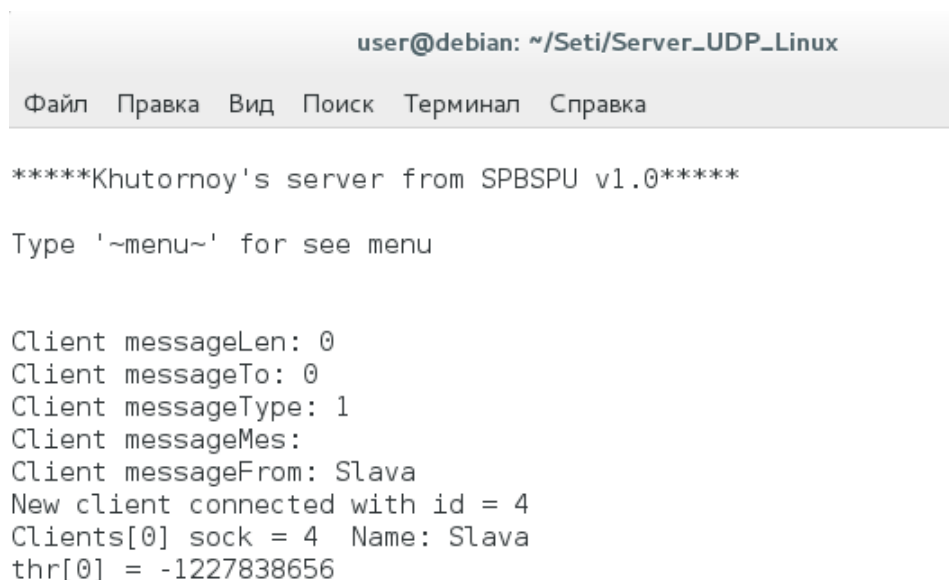


Рис. 21. Результат работы программы – сервера при подключении к нему двух клиентов при потере пакетов

2) Пересылка сообщений при 50% потере пакетов в сети

```
tc qdisc change dev eth0 root netem loss 50%
```

Результаты:

Без потерь пакетов в сети все сообщения, отправляемые клиентом серверу, были успешно доставлены. При добавлении в сеть потерь, сообщения терялись. В этом эксперименте приложение – клиент будет отправлять 8 сообщений серверу. В результате эксперимента из 8 отправленных сообщений со стороны клиента на стороне сервера было принято только 6 сообщений.

```
user@debian: ~/Seti/Client_UDP_Linux x
Файл Правка Вид Поиск Терминал Справка
*****Khutornoy's client from SPBSPU v1.0
*****

Type '~menu~' for see menu

<- Your message
1

<- Your message
2

<- Your message
3

<- Your message
4

<- Your message
5

<- Your message
6

<- Your message
7

<- Your message
8

<- Your message

```

Рис. 22. Результат работы программы – клиента. Отправка 8 сообщений серверу.

```
user@debian: ~/Seti/Server_UDP_Linux x
Файл Правка Вид Поиск Терминал Справка
New client connected with id = 4
Clients[0] sock = 4 Name: Slava
thr[0] = -1226892480

Client messageLen: 1
Client messageTo: 0
Client messageType: 0
Client messageMes: 2
Client messageFrom: Slava

Client messageLen: 1
Client messageTo: 0
Client messageType: 0
Client messageMes: 3
Client messageFrom: Slava

Client messageLen: 1
Client messageTo: 0
Client messageType: 0
Client messageMes: 5
Client messageFrom: Slava

Client messageLen: 1
Client messageTo: 0
Client messageType: 0
Client messageMes: 6
Client messageFrom: Slava

Client messageLen: 1
Client messageTo: 0
Client messageType: 0
Client messageMes: 7
Client messageFrom: Slava

Client messageLen: 1
Client messageTo: 0
Client messageType: 0
Client messageMes: 8
Client messageFrom: Slava

```

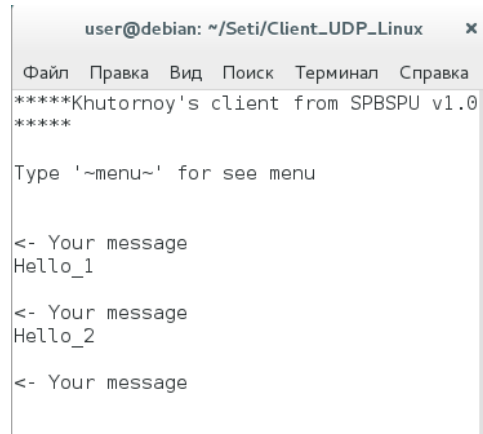
Рис. 23. Результат работы программы – сервера. Прием сообщений от клиента.

3) Пересылка сообщений при дубликации пакетов в сети

```
tc qdisc change dev eth0 root netem duplicate 50%
```

Результаты:

В этом эксперименте со стороны клиента отправлялись два разных сообщения. В результат эксперимента на стороне сервера были приняты продублированные сообщения.



```
user@debian: ~/Seti/Client_UDP_Linux x
Файл  Правка  Вид  Поиск  Терминал  Справка
*****Khutornoy's client from SPBSPU v1.0
*****
Type '~menu~' for see menu

<- Your message
Hello_1

<- Your message
Hello_2

<- Your message
```

Рис. 24. Результат работы программы – клиента при отправке сообщений



```
user@debian: ~/Seti/Server_UDP_Linux x
Файл  Правка  Вид  Поиск  Терминал  Справка
*****Khutornoy's server from SPBSPU v1.0
*****
Type '~menu~' for see menu

Client messageLen: 0
Client messageTo: 0
Client messageType: 1
Client messageMes:
Client messageFrom: Slava
New client connected with id = 4
Clients[0] sock = 4 Name: Slava
thr[0] = -1227654336

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_1
Client messageFrom: Slava

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_1
Client messageFrom: Slava

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_2
Client messageFrom: Slava

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_2
Client messageFrom: Slava
]
```

Рис. 25. Результат работы программы – сервера при приеме сообщений

4) Помехи в пакетах

```
tc qdisc change dev eth0 root netem corrupt 70%
```

Результаты:

В этом эксперименте со стороны клиента отправлялось 8 сообщений. В результате действия помех (70%) серверу удалось принять только 3 сообщения из 8. При помехах в 50% все сообщения были успешно доставлены.



```
user@debian: ~/Seti/Client_UDP_Linux x
Файл Правка Вид Поиск Терминал Справка
*****Khutornoy's client from SPBSPU v1.0
*****

Type '~menu~' for see menu

<- Your message
Hello_1

<- Your message
Hello_2

<- Your message
Hello_3

<- Your message
Hello_4

<- Your message
Hello_5

<- Your message
Hello_6

<- Your message
Hello_7

<- Your message
Hello_8

<- Your message
|
```

Рис. 26. Результат работы программы – клиента при отправке сообщений



```
user@debian: ~/Seti/Server_UDP_Linux x
Файл Правка Вид Поиск Терминал Справка

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_3
Client messageFrom: Slava

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_4
Client messageFrom: Slava

Client messageLen: 7
Client messageTo: 0
Client messageType: 0
Client messageMes: Hello_7
Client messageFrom: Slava
```

Рис. 27. Результат работы программы – сервера при приеме сообщений

Глава 4

Выводы

В результате работы была создана клиент-серверная система обмена сообщениями. Система состоит из сервера, который служит ретранслятором клиентских сообщений. Также он может предоставлять клиентам список подключенных к серверу клиентов. Сервер позволяет принудительно отключить клиента. Клиенты могут выбрать адресата и запросить у сервера список онлайн – клиентов. Все поставленные требования были выполнены. Тестирование программ прошло успешно, тесты показали корректность работы приложения. Однако данное приложение нельзя назвать законченным. Оно лишь демонстрирует сетевую передачу данных через сокеты. Например, при некорректном завершении работы программы – клиента, сервер будет воспринимать этого клиента как подключенного, из-за этого могут возникнуть проблемы с обменом сообщениями с данным клиентом. Задачу проверки наличия соединения можно легко решить путем добавления механизмов пульсации.

Протокол TCP удобен для реализации пользовательских приложений, так как обеспечивает установление соединения и надежную доставку пакетов. Протокол обеспечивает стабильное надежное соединение. 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 просимулированы помехи в сети. Тестирование показало, что приложение надежно работает при низком и среднем уровне помех.

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. С помощью утилиты `tc` при тестировании UDP реализации были просимулированы помехи в сети. Тестирование показало, что при низком уровне помех приложение продолжает функционировать, однако могут появляться небольшие ошибки и потери. Если в сети присутствуют сильные помехи, приложение начинает работать некорректно и сообщения могут не достигнуть адресата.

Приложения

Листинг 1. Server_TCP_Windows.cpp

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>
#include <io.h>
#include <devioctl.h>
#include <stdlib.h>
#include <winsock2.h>
#include <windows.h>
#include <clocale>
#include <cstring>

#pragma comment(lib, "ws2_32.lib")

#define DEF_PORT 8888
#define DEF_IP "127.0.0.1"

DWORD WINAPI threadHandler(LPVOID);
DWORD WINAPI clientHandler(LPVOID);

char names[32][32];
char wrbuf[512];
int fd;
int clients[128];
int sizenames[128];
int c = 0;
HANDLE threads[128];

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

int cliview()
{
    int i = 0;
    int y = 0;

    if(c == 0)
    {
        printf("No clients\n");
    }

    for(i=0;i<c;i++)
    {
        printf("Clients[%i] sock = %i  ", i, clients[i]);
        printf("Name: ",i);
    }
}
```

```

        for(y=0;y<sizeofnames[i];y++)
        {
            printf("%c", names[i][y]);
        }
        printf("\n");
    }

return 0;
}

int thrview()
{
    int i =0;

    if(c == 0)
    {
        printf("No thr\n");
    }

    for(i=0;i<c;i++)
    {
        printf("thr[%i] = %i\n", i, threads[i]);
    }
return 0;
}

int clirm(int sock)
{
    int i = 0;
    int i1;
    int i2;
    while(i < c)
    {
        if(clients[i] == sock)
        {
            c=c-1;
            for(i1=i;i1<c;i1++)
            {
                clients[i1] = clients[i1+1];
                sizeofnames[i1] = sizeofnames[i1+1] ;

                for(i2=0;i2<32;i2++)
                {
                    names[i1][i2] = names[i1+1][i2];
                }
            }
            else i++;
        }
    }

return 0;
}

int menu(char wrbuf[512])
{
    char wrbuf2[8];
    char wrbuf3[8];

```

```

int numcli;
int nummenu;

if (strcmp(wrbuf, "~menu~\n", 7) == 0)
{
    system("cls");
    printf("*****Menu*****\n");
    printf("1) Exit from menu\n");
    printf("2) Delete client\n");
    printf("3) View list of onlain client\n");

    fgets(&wrbuf2[0], sizeof(wrbuf2)-1, stdin);
    nummenu = atoi(&wrbuf2[0]);

    if (nummenu == 1)
    {
        system("cls");
        return 0;
    }

    if (nummenu == 2)
    {
        system("cls");
        printf("Please, choose ID of client for deleting\n");
        cliview();
        printf("\nSlect ID: ");

        fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
        numcli = atoi(&wrbuf3[0]);

        TerminateThread(threads[numcli], NULL);
        closesocket(clients[numcli]);
        CloseHandle(threads[numcli]);
        clirm(clients[numcli]);

        system("cls");
        printf("removing %s with ID = %i.\n Press Enter for exit to\n", &names[numcli], numcli);

        fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
        if (strcmp(wrbuf3, "\n", 1) == 0)
        {
            system("cls");
            return 0;
        };

        return 0;
    }

    if (nummenu == 3)
    {
        system("cls");
        printf("*****The list of OnLine clients*****\n");
        cliview();
        return 0;
    }

}
return 0;

```

```

}

int main()
{
char ipbuf[18];
char portbuf[10];
char buff[1024];
int port = 0;

for(;;)
{

system("cls");
printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");

int ip1 = 0;
int ip2 = 0;
int ip3 = 0;
int ip4 = 0;

char cip1[10];
char cip2[10];
char cip3[10];
char cip4[10];

printf("Please, choose IP-addres for using\n");
printf("Like this: xxx.xxx.xxx.xxx\n");
printf("xxx must be in range from 0 to 255\n");
printf("For example: 127.000.000.001\n\n");

fgets(&ipbuf[0],sizeof(ipbuf)-1,stdin);

ip1= atoi(&ipbuf[0]);
ip2= atoi(&ipbuf[4]);
ip3= atoi(&ipbuf[8]);
ip4= atoi(&ipbuf[12]);

sprintf_s(cip1,"%03i",ip1);
sprintf_s(cip2,"%03i",ip2);
sprintf_s(cip3,"%03i",ip3);
sprintf_s(cip4,"%03i",ip4);

if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2] !=
ipbuf[2])) ||
    ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
    ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
    ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2] !=
ipbuf[14])))

    || (((ip1<0) || (ip1>255)) ||
        ((ip2<0) || (ip2>255)) ||
        ((ip3<0) || (ip3>255)) ||
        ((ip4<0) || (ip4>255))))

{
printf("\nError addres!\n");

```

```

printf("Follow the rules!\n");
printf("Please repeat!\n");
Sleep(3000);
}
else

{
printf("Your addres = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);
break;
}
}

printf("Please, choose PORT for using\n");
fgets(&portbuf[0],sizeof(portbuf)-1,stdin);
port = atoi(&portbuf[0]);
printf("Your port = %i\n", port);

if ((port < 0) || (port > 65535))
{
printf("Uncorrect port = %i\n", port);
printf("Using default port %d\n",DEF_PORT);
port = DEF_PORT;
}

if (WSAStartup(0x202,(WSADATA *)&buff[0]))
{
printf("WSAstart error %d\n",WSAGetLastError());
return -1;
}

struct sockaddr_in listenerInfo;
listenerInfo.sin_family = AF_INET;
listenerInfo.sin_port = htons( port );
listenerInfo.sin_addr.s_addr = inet_addr(ipbuf);

int listener = socket(AF_INET, SOCK_STREAM, 0 );
if ( listener < 0 ) {
perror( "Can't create socket to listen: " );
exit(1);
}

int res = bind(listener, ( struct sockaddr * )&listenerInfo, sizeof(
listenerInfo ) );
if ( res < 0 ) {
perror( "Can't bind socket" );
exit( 1 );
}

res = listen(listener,5);
if (res) {
perror("Error while listening:");
exit(1);
}

HANDLE t;
int number = 0;
t = CreateThread(NULL, 0, threadHandler, (LPVOID)number, 0, NULL);

system("cls");
printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");

```



```

printf("Type '~menu~' for see menu\n\n");

for(;;)
{
    int client = accept(listener, NULL, NULL );
    printf("New client connected with id = %i\n", client);
    clients[c] = client;

    HANDLE t2;
    t2 = CreateThread(NULL, 0, clientHandler, (LPVOID)client,
0, NULL);

    if (t2)
    {
        //printf("Error while creating new thread\n");
    }

    threads[c] = t2;
    c++;
    cliview();
    thrview();
}

return 0;
}

DWORD WINAPI threadHandler(LPVOID param)
{
    for(;;)
    {
        _read(0, wrbuf, sizeof(wrbuf)-1);
        menu(wrbuf);
    }
    return 0;
}

DWORD WINAPI clientHandler(LPVOID args)
{
    int sock = (SOCKET)args;
    int res = 0;
    struct Message msg;

    for(;;)
    {
        res = recv(sock, (char*)&msg, sizeof(msg) ,0);
        if (res < 0)
        {
            printf("socket with id = %i was unconnected\n", sock);
            clirm(sock);
            cliview();
            closesocket(sock);
            CloseHandle(0);
            return 0;
        }

        printf("\nClient messageLen: %i\n", msg.messageLen);
        printf("Client messageTo: %i\n", msg.messageID);
    }
}

```

```

printf("Client messageType: %i\n", msg.messageType);
printf("Client messageMes: ");

int i;
for (i=0;i<msg.messageLen;i++)
{
printf("%c",msg.messageMes[i]);
}

printf("\n");
printf("Client messageFrom: %s\n", msg.messageName);

int type;
type = msg.messageType;

if(type == 0)
{
    send(msg.messageID, (char*)&msg, sizeof(msg), 0);
}

if(type == 1)
{
    int k = 0;
    int u = 0;

    for(k=0;k<128;k++)
    {
        if(clients[k]==sock)
        {
            sizenames[k] = msg.messageSizename;

            for(u=0;u<32;u++)
            {
                names[k][u] = msg.messageName[u];
            }
        }
    }
}

if(type == 2)
{
    int x = 0 ;
    char str[512];
    char sername[] = "Online Clients From Server \n";
    for (x=0;x<c;x++)
    {
        memcpy(msg.messageName, sername, sizeof(sername));
        msg.messageSizename = strlen(msg.messageName);

        sprintf_s(str,"%i)User onlain: %s\n ID user: %i",x+1,
&names[x],clients[x]);
        msg.messageLen = strlen(str);
        memcpy(msg.messageMes, str, sizeof(str)-1);
        send(sock, (char*)&msg, sizeof(msg)-1, 0);
        Sleep(100);
        memset(&str, NULL, sizeof(str));
    }
}

```

```

    }
    return 0;
}

```

Листинг 2. Client_TCP_Windows.cpp

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>
#include <io.h>
#include <devioctl.h>
#include <stdlib.h>
#include <winsock2.h>
#include <windows.h>
#include <clocale>
#include <cstring>

#define MAXBUF 1024
#define DEF_PORT 8888

#pragma comment(lib, "ws2_32.lib")

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

char wrbuf[512];
struct Message msg;
struct Message msg2;
int client_sockfd;

fd_set readfds, testfds;

DWORD WINAPI threadHandler(LPVOID);

int menu(char wrbuf[512],int rc)
{
    int nummenu;
    int numid;
    char wrbuf2[8];
    char wrbuf3[8];
    int len = rc;

    if (strncmp(wrbuf,"~menu~\n",7)==0)
    {
        system("cls");
        printf("*****Menu*****\n");
        printf("1) Select ID of client\n");
        printf("2) View OnLine client\n");
        printf("3) Exit from menu\n");
    }
}

```

```

    fgets(&wrbuf2[0], sizeof(wrbuf2)-1, stdin);
    nummenu = atoi(&wrbuf2[0]);

    if (nummenu == 1)
    {
        printf("Select ID: ");
        fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
        numid = atoi(&wrbuf3[0]);
        msg2.messageID = numid;

        system("cls");
        printf("*****Khutornoy's client from SPBSPU
v1.0*****\n\n");
        printf("Type '~menu~' for see menu\n\n");

        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

    if (nummenu == 3)
    {
        system("cls");
        printf("*****Khutornoy's client from SPBSPU
v1.0*****\n\n");
        printf("Type '~menu~' for see menu\n\n");
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

    if (nummenu == 2)
    {
        system("cls");
        msg2.messageType = 2;
        msg2.messageLen = 0;
        memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
        send(client_sockfd, (char*)&msg2, sizeof(msg2), 0);

        msg2.messageType = 0;
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

}

return 0;
}

int main()
{
    char ipbuf[18];
    char portbuf[10];
    char buff[1024];
    int port = 0;

    for(;;)
    {

```

```

system("cls");
printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");

int ip1 = 0;
int ip2 = 0;
int ip3 = 0;
int ip4 = 0;

char cip1[10];
char cip2[10];
char cip3[10];
char cip4[10];

printf("Please, choose IP-addres of server\n");
printf("Like this: xxx.xxx.xxx.xxx\n");
printf("xxx must be in range from 0 to 255\n");
printf("For example: 127.000.000.001\n\n");

fgets(&ipbuf[0], sizeof(ipbuf)-1, stdin);

ip1= atoi(&ipbuf[0]);
ip2= atoi(&ipbuf[4]);
ip3= atoi(&ipbuf[8]);
ip4= atoi(&ipbuf[12]);

sprintf_s(cip1, "%03i", ip1);
sprintf_s(cip2, "%03i", ip2);
sprintf_s(cip3, "%03i", ip3);
sprintf_s(cip4, "%03i", ip4);

if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2] !=
ipbuf[2])) ||
    ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
    ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
    ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2] !=
ipbuf[14])))

    || (((ip1<0) || (ip1>255)) ||
        ((ip2<0) || (ip2>255)) ||
        ((ip3<0) || (ip3>255)) ||
        ((ip4<0) || (ip4>255))))

{
printf("\nError addres!\n");
printf("Follow the rules!\n");
printf("Please repeat!\n");
Sleep(3);
}
else

{
printf("\nOk...\n");
printf("Addres of server = %i.%i.%i.%i\n\n", ip1, ip2, ip3, ip4);
break;
}
}

printf("Please, choose PORT of server\n");

```

```

    fgets(&portbuf[0], sizeof(portbuf)-1, stdin);
    port = atoi(&portbuf[0]);
    printf("\nOk...\n");
    printf("Port of server = %i\n", port);

    if ((port < 0) || (port > 65535))
    {
        printf("Uncorrect port = %i\n", port);
        printf("Using default port %d\n", DEF_PORT);
        port = DEF_PORT;
    }

    if (WSAStartup(0x202, (WSADATA *)&buff[0]))
    {
        printf("WSAStart error %d\n", WSAGetLastError());
        return -1;
    }

    char namebuf[32];
    struct sockaddr_in server;
    int rc;

    server.sin_addr.s_addr = inet_addr(ipbuf);
    server.sin_family = AF_INET;
    server.sin_port = htons(port);

    client_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (client_sockfd == -1)
    {
        printf("Could not create socket");
    }

    if (connect(client_sockfd, (struct sockaddr *)&server, sizeof(server)) <
    0)
    {
        perror("connect failed. Error");
        return 1;
    }
    printf("\nOk...\n");
    printf("Succses connect to server\n");

    printf("\nPlease, input your name. Must be < 30 character.\n");
    gets_s(&namebuf[0], sizeof(namebuf)-1);
    namebuf[31] = '\0';
    namebuf[30] = '\n';
    printf("\nOk...\n");

    memcpy(msg2.messageName, namebuf, sizeof(namebuf));
    msg2.messageType = 1;
    msg2.messageSizeName = strlen(namebuf);
    send(client_sockfd, (char*)&msg2, sizeof(msg2), 0);
    msg2.messageType = 0;

    HANDLE t;
    int number = 0;
    t = CreateThread(NULL, 0, threadHandler, (LPVOID)number, 0, NULL);

```

```

system("cls");
printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");
printf("Type '~menu~' for see menu\n\n");

while(1)
{
    printf("\n<- Your message \n");
    rc = _read(0, wrbuf, sizeof(wrbuf)-1);

    if (strncmp(wrbuf, "~menu~\n", 7) == 0)
    {
        menu(wrbuf, rc);
    }

    else
    {
        msg2.messageLen = (rc-1);
        memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
        send(client_sockfd, (char*)&msg2, sizeof(msg2), 0);
        memset(&wrbuf, NULL, sizeof(wrbuf));
    }
}

DWORD WINAPI threadHandler(LPVOID param)
{
    int rc;
    for(;;)
    {
        rc = recv(client_sockfd, (char*)&msg, sizeof(msg)
, 0);

        if (rc < 0)
        {
            system("cls");
            printf("Server was unconnected or kicked you!");
            closesocket(client_sockfd);
            return 0;
        }

        if (rc < 0) perror("error vizova recv");

        Beep(1150, 100);
        //printf("Server messageLen: %i\n", msg.messageLen);
        //printf("Server messageID: %i\n", msg.messageID);
        printf("-> Message from %s\n", msg.messageName);

        int i;
        for (i=0; i<msg.messageLen; i++)
        {
            printf("%c", msg.messageMes[i]);
        }

        printf("\n\n<- Your message \n");
    }

    return 0;
}

```

Листинг 3. Server_UDP_Windows.cpp

```

#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>
#include <io.h>
#include <devioctl.h>
#include <stdlib.h>
#include <winsock2.h>
#include <windows.h>
#include <locale>
#include <cstring>

#pragma comment(lib, "ws2_32.lib")

#define DEF_PORT 8888
#define DEF_IP "127.0.0.1"

DWORD WINAPI threadHandler(LPVOID);
DWORD WINAPI clientHandler(LPVOID);

char names[32][32];
char wrbuf[512];
int fd;
int clients[128];
int sizenames[128];
int c = 0;
HANDLE threads[128];
sockaddr_in soki[512];
struct sockaddr_in newlistenerInfo;

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

int cliview()
{
    int i = 0;
    int y = 0;

    if(c == 0)
    {
        printf("No clients\n");
    }

    for(i=0;i<c;i++)
    {
        printf("Clients[%i] sock = %i  ", i, clients[i]);
        printf("Name: ",i);

        for(y=0;y<sizenames[i];y++)

```



```

        {
            printf("%c", names[i][y]);
        }
        printf("\n");
    }

return 0;
}

int thrview()
{
    int i =0;

    if(c == 0)
    {
        printf("No thr\n");
    }

    for(i=0;i<c;i++)
    {
        printf("thr[%i] = %i\n", i, threads[i]);
    }
return 0;
}

int clirm(int sock)
{
    int i = 0;
    int i1;
    int i2;
    while(i < c)
    {
        if(clients[i] == sock)
        {
            c=c-1;

            for(i1=i;i1<c;i1++)
            {
                clients[i1] = clients[i1+1];
                sizenames[i1] = sizenames[i1+1] ;
                soki[i1] = soki[i1+1];

                for(i2=0;i2<32;i2++)
                {
                    names[i1][i2] = names[i1+1][i2];
                }
            }
            else i++;
        }
    }

return 0;
}

int menu(char wrbuf[512])
{
    char wrbuf2[8];

```

```

char wrbuf3[8];
int numcli;
int nummenu;

if (strcmp(wrbuf, "~menu~\n", 7) == 0)
{
    system("cls");
    printf("*****Menu*****\n");
    printf("1) Exit from menu\n");
    printf("2) Delete client\n");
    printf("3) View list of onlain client\n");

    fgets(&wrbuf2[0], sizeof(wrbuf2)-1, stdin);
    nummenu = atoi(&wrbuf2[0]);

    if (nummenu == 1)
    {
        system("cls");
        return 0;
    }

    if (nummenu == 2)
    {
        system("cls");
        printf("Please, choose ID of client for deleting\n");
        cliview();
        printf("\nSlect ID: ");

        fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
        numcli = atoi(&wrbuf3[0]);

        TerminateThread(threads[numcli], NULL);
        closesocket(clients[numcli]);
        ZeroMemory(&soki[numcli], sizeof(sockaddr_in));
        ZeroMemory(&clients[numcli], sizeof(int));
        ZeroMemory(&sizenames[numcli], sizeof(int));
        CloseHandle(threads[numcli]);
        clirm(clients[numcli]);

        system("cls");
        printf("removing %s with ID = %i.\n Press Enter for exit to\n", &names[numcli], numcli);

        fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
        if (strcmp(wrbuf3, "\n", 1) == 0)
        {
            system("cls");
            return 0;
        }

        return 0;
    }

    if (nummenu == 3)
    {
        system("cls");
        printf("*****The list of OnLine clients*****\n");
        cliview();
        return 0;
    }
}

```

```

    }

    }
    return 0;
}

int main()
{
    char ipbuf[18];
    char portbuf[10];
    char buff[1024];
    int port = 0;

    for(;;)
    {

        system("cls");
        printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");

        int ip1 = 0;
        int ip2 = 0;
        int ip3 = 0;
        int ip4 = 0;

        char cip1[10];
        char cip2[10];
        char cip3[10];
        char cip4[10];

        printf("Please, choose IP-addres for using\n");
        printf("Like this: xxx.xxx.xxx.xxx\n");
        printf("xxx must be in range from 0 to 255\n");
        printf("For example: 127.000.000.001\n\n");

        fgets(&ipbuf[0], sizeof(ipbuf)-1, stdin);

        ip1= atoi(&ipbuf[0]);
        ip2= atoi(&ipbuf[4]);
        ip3= atoi(&ipbuf[8]);
        ip4= atoi(&ipbuf[12]);

        sprintf_s(cip1,"%03i",ip1);
        sprintf_s(cip2,"%03i",ip2);
        sprintf_s(cip3,"%03i",ip3);
        sprintf_s(cip4,"%03i",ip4);

        if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2] !=
ipbuf[2])) ||
            ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
            ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
            ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2] !=
ipbuf[14])))

            || (((ip1<0) || (ip1>255)) ||
                ((ip2<0) || (ip2>255)) ||
                ((ip3<0) || (ip3>255)) ||

```

```

        ((ip4<0) || (ip4>255))))

{
printf("\nError addres!\n");
printf("Follow the rules!\n");
printf("Please repeat!\n");
Sleep(3000);
}
else

{
printf("Your addres = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);
break;
}
}

printf("Please, choose PORT for using\n");
fgets(&portbuf[0],sizeof(portbuf)-1,stdin);
port = atoi(&portbuf[0]);
printf("Your port = %i\n", port);

if ((port < 0) || (port > 65535))
{
printf("Uncorrect port = %i\n", port);
printf("Using default port %d\n",DEF_PORT);
port = DEF_PORT;
}

if (WSAStartup(0x202, (WSADATA *)&buff[0]))
{
printf("WSAStart error %d\n",WSAGetLastError());
return -1;
}

struct sockaddr_in listenerInfo;
listenerInfo.sin_family = AF_INET;
listenerInfo.sin_port = htons( port );
listenerInfo.sin_addr.s_addr = inet_addr(ipbuf);

int listener = socket(AF_INET, SOCK_DGRAM, 0 );
if ( listener < 0 ) {
perror( "Can't create socket to listen: " );
exit(1);
}

int res = bind(listener, ( struct sockaddr * )&listenerInfo, sizeof(
listenerInfo ) );
if ( res < 0 ) {
perror( "Can't bind socket" );
exit( 1 );
}

HANDLE t;
int number = 0;
t = CreateThread(NULL, 0, threadHandler, (LPVOID)number, 0, NULL);

system("cls");
printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");
printf("Type '~menu~' for see menu\n\n");

```

```

struct Message msg;

for(;;)
{
    sockaddr_in client_addr;
    int client_addr_size = sizeof(client_addr);
    int bsize = recvfrom(listener, (char*)&msg, sizeof(msg), 0,
(sockaddr *) &client_addr, &client_addr_size);

    soki[c] = client_addr;

    if (bsize==SOCKET_ERROR) printf("recvfrom() error: %d\n",
WSAGetLastError());

    printf("\nClient messageLen: %i\n", msg.messageLen);
    printf("Client messageTo: %i\n", msg.messageID);
    printf("Client messageType: %i\n", msg.messageType);
    printf("Client messageMes: ");

    int i;
    for (i=0;i<msg.messageLen;i++)
    {
        printf("%c",msg.messageMes[i]);
    }

    printf("\n");
    printf("Client messageFrom: %s\n", msg.messageName);

    int type;
    type = msg.messageType;

    if(type==1)
    {

        sizenames[c] = msg.messageSizename;

        int u = 0;
        for(u=0;u<32;u++)
        {
            names[c][u] = msg.messageName[u];
        }

        msg.messageSizename = 0;
        msg.messageLen = 0;
        msg.messageID = 10000+c;
        msg.messageType = 1;
        int yer = msg.messageID;
        sendto(listener, (char*)&msg, sizeof(msg), 0, (sockaddr
*)&client_addr, sizeof(client_addr));

        newlistenerInfo.sin_family = AF_INET;
        newlistenerInfo.sin_port = htons( msg.messageID );
        newlistenerInfo.sin_addr = client_addr.sin_addr;

        int newlistener = socket(AF_INET, SOCK_DGRAM, 0 );
        if ( newlistener < 0 ) {
            perror( "Can't create socket to listen: " );
            exit(1);
        }
    }
}

```

```

        int res = bind(newlistener, ( struct sockaddr *
)&newlistenerInfo, sizeof(newlistenerInfo) );
        if ( res < 0 ) {
            perror( "Can't bind socket" );
            exit( 1 );
        }

        soki[c].sin_port = newlistenerInfo.sin_port;

        HANDLE t2;
        t2 = CreateThread(NULL, 0, clientHandler, (LPVOID)newlistener,
0, NULL);

        if (t2 == NULL)
        {
            printf("Error while creating new thread\n");
        }

        printf("New client connected with id = %i\n", newlistener);

        clients[c] = newlistener;
        threads[c] = t2;
        c++;

        cliview();
        thrview();
    }
}
return 0;
}

DWORD WINAPI threadHandler(LPVOID param)
{
    for(;;)
    {
        _read(0, wrbuf, sizeof(wrbuf)-1);
        menu(wrbuf);
    }
    return 0;
}

DWORD WINAPI clientHandler(LPVOID args)
{
    int sock = (SOCKET)args;
    int res = 0;
    struct Message msg;
    sockaddr_in client_addr;
    int client_addr_size = sizeof(client_addr);

    res = recvfrom(sock, (char*)&msg, sizeof(msg), 0, (sockaddr *)
&client_addr, &client_addr_size);
    soki[c-1].sin_port = client_addr.sin_port;

    if (res == 0)
    {
        printf("socket with id = %i was unconnected\n", sock,
WSAGetLastError());
        clirm(sock);
        cliview();
    }
}

```

```

closesocket(sock);
CloseHandle(0);

int h = 0;
for (h=0;h<c;h++)
{
    if(clients[h] == sock)
    {
        ZeroMemory(&soki[h], sizeof(sockaddr_in));
        ZeroMemory(&clients[h], sizeof(int));
        ZeroMemory(&sizenames[h], sizeof(int));
    }
}

return 0;
}

printf("\nClient messageLen: %i\n", msg.messageLen);
printf("Client messageTo: %i\n", msg.messageID);
printf("Client messageType: %i\n", msg.messageType);
printf("Client messageMes: ");

int i;
for (i=0;i<msg.messageLen;i++)
{
    printf("%c",msg.messageMes[i]);
}

printf("\n");
printf("Client messageFrom: %s\n", msg.messageName);

int type;
type = msg.messageType;

if(type == 0)
{
    int r = 0;
    for (r=0;r<c;r++)
    {
        if (msg.messageID == clients[r])
        {
            break;
        }
    }

    sendto(msg.messageID, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*)&soki[r], sizeof(soki[r]));
}

if(type == 2)
{
    int x = 0 ;
    char str[512];
    char sername[] = "Online Clients From Server \n";
    for (x=0;x<c;x++)
    {
        memcpy(msg.messageName, sername, sizeof(sername));
        msg.messageSizenam = strlen(msg.messageName);
        sprintf_s(str,"%i)User onlain: %s\n ID user: %i",x+1,
&names[x],clients[x]);
    }
}

```

```

        msg.messageLen = strlen(str);
        memcpy(msg.messageMes, str, sizeof(str)-1);
        sendto(sock, (char*)&msg, sizeof(msg), 0, (sockaddr *)
*&client_addr, sizeof(client_addr));
        Sleep(100);
        memset(&str, NULL, sizeof(str));
    }
}

for(;;)
{
    res = 0;
    res = recvfrom(sock, (char*)&msg, sizeof(msg), 0, (sockaddr *)
&client_addr, &client_addr_size);

    if (res == 0)
    {
        printf("socket with id = %i was unconnected\n", sock,
WSAGetLastError());
        clirm(sock);
        cliview();
        closesocket(sock);
        CloseHandle(0);

        int h = 0;
        for (h=0;h<c;h++)
        {
            if(clients[h] == sock)
            {
                ZeroMemory(&soki[h], sizeof(sockaddr_in));
                ZeroMemory(&clients[h], sizeof(int));
                ZeroMemory(&sizenames[h], sizeof(int));
            }
        }
    }

    printf("\nClient messageLen: %i\n", msg.messageLen);
    printf("Client messageTo: %i\n", msg.messageID);
    printf("Client messageType: %i\n", msg.messageType);
    printf("Client messageMes: ");

    int i;
    for (i=0;i<msg.messageLen;i++)
    {
        printf("%c",msg.messageMes[i]);
    }

    printf("\n");
    printf("Client messageFrom: %s\n", msg.messageName);

    int type;
    type = msg.messageType;

    if(type == 0)
    {
        if((msg.messageMes[0] == '~') &&
            (msg.messageMes[1] == 'q') &&
            (msg.messageMes[2] == 'u') &&
            (msg.messageMes[3] == 'i') &&
            (msg.messageMes[4] == 't') &&

```



```

        (msg.messageMes[5] == '~') &&
        (msg.messageMes[6] == '\n'))
    {
        int h = 0;
        for (h=0;h<c;h++)
        {
            if(clients[h] == sock)
            {
                ZeroMemory(&soki[h], sizeof(sockaddr_in));
                ZeroMemory(&clients[h], sizeof(int));
                ZeroMemory(&sizenames[h], sizeof(int));
                clirm(clients[h]);
            }
        }
        closesocket(sock);
        return 0;
    }

    int r = 0;
    for (r=0;r<c;r++)
    {
        if (msg.messageID == clients[r])
        {
            break;
        }
    }

    sendto(msg.messageID, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*)&soki[r], sizeof(soki[r]));
}

if(type == 2)
{
    int x = 0 ;
    char str[512];
    char sername[] = "Online Clients From Server \n";
    for (x=0;x<c;x++)
    {
        memcpy(msg.messageName, sername, sizeof(sername));
        msg.messageSizenam = strlen(msg.messageName);
        sprintf_s(str,"%i>User onlain: %s\n ID user: %i",x+1,
&names[x],clients[x]);
        msg.messageLen = strlen(str);
        memcpy(msg.messageMes, str, sizeof(str)-1);
        sendto(sock, (char*)&msg, sizeof(msg), 0, (sockaddr
*)&client_addr, sizeof(client_addr));
        Sleep(100);
        memset(&str,NULL,sizeof(str));
    }
}

return 0;
}

```

Листинг 4. Client_UDP_Windows.cpp

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>

```

```

#include <io.h>
#include <devioctl.h>
#include <stdlib.h>
#include <winsock2.h>
#include <windows.h>
#include <clocale>
#include <cstring>

#define MAXBUF 1024
#define DEF_PORT 8888

#pragma comment(lib, "ws2_32.lib")

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

char wrbuf[512];
struct Message msg;
struct Message msg2;
int client_sockfd;
int client_sockfd2;
struct sockaddr_in server;

DWORD WINAPI threadHandler(LPVOID);

int menu(char wrbuf[512],int rc)
{
    int nummenu;
    int numid;
    char wrbuf2[8];
    char wrbuf3[8];
    int len = rc;

    if (strncmp(wrbuf,"~menu~\n",7)==0)
    {
        system("cls");
        printf("*****Menu*****\n");
        printf("1) Select ID of client\n");
        printf("2) View OnLine client\n");
        printf("3) Exit from menu\n");

        fgets(&wrbuf2[0],sizeof(wrbuf2)-1,stdin);
        nummenu = atoi(&wrbuf2[0]);

        if (nummenu == 1)
        {
            printf("Select ID: ");
            fgets(&wrbuf3[0],sizeof(wrbuf3)-1,stdin);
            numid = atoi(&wrbuf3[0]);
            msg2.messageID = numid;

```

```

        system("cls");
        printf("*****Khutornoy's client from SPBSPU
v1.0*****\n\n");
        printf("Type '~menu~' for see menu\n\n");

        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

    if (nummenu == 3)
    {
        system("cls");
        printf("*****Khutornoy's client from SPBSPU
v1.0*****\n\n");
        printf("Type '~menu~' for see menu\n\n");
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

    if (nummenu == 2)
    {
        system("cls");
        msg2.messageType = 2;
        msg2.messageLen = 0;
        memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
        sendto(client_sockfd2, (char*)&msg2, sizeof(msg2), 0,
(sockaddr *)&server, sizeof(server));

        msg2.messageType = 0;
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }
}
return 0;
}

int main()
{
    char ipbuf[18];
    char portbuf[10];
    char buff[1024];
    int port = 0;

    for(;;)
    {
        system("cls");
        printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");

        int ip1 = 0;
        int ip2 = 0;
        int ip3 = 0;
        int ip4 = 0;

        char cip1[10];
        char cip2[10];
        char cip3[10];
        char cip4[10];

        printf("Please, choose IP-addres of server\n");
        printf("Like this: xxx.xxx.xxx.xxx\n");

```

```

printf("xxx must be in range from 0 to 255\n");
printf("For example: 127.000.000.001\n\n");

fgets(&ipbuf[0],sizeof(ipbuf)-1,stdin);

ip1= atoi(&ipbuf[0]);
ip2= atoi(&ipbuf[4]);
ip3= atoi(&ipbuf[8]);
ip4= atoi(&ipbuf[12]);

sprintf_s(cip1,"%03i",ip1);
sprintf_s(cip2,"%03i",ip2);
sprintf_s(cip3,"%03i",ip3);
sprintf_s(cip4,"%03i",ip4);

if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2] !=
ipbuf[2])) ||
    ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
    ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
    ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2] !=
ipbuf[14])))

    || (((ip1<0) || (ip1>255)) ||
        ((ip2<0) || (ip2>255)) ||
        ((ip3<0) || (ip3>255)) ||
        ((ip4<0) || (ip4>255))))

{
printf("\nError adres!\n");
printf("Follow the rules!\n");
printf("Please repeat!\n");
Sleep(3);
}
else

{
printf("\nOk...\n");
printf("Adres of server = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);
break;
}
}

printf("Please, choose PORT of server\n");
fgets(&portbuf[0],sizeof(portbuf)-1,stdin);
port = atoi(&portbuf[0]);
printf("\nOk...\n");
printf("Port of server = %i\n", port);

if ((port < 0) || (port > 65535))
{
printf("Uncorrect port = %i\n", port);
printf("Using default port %d\n",DEF_PORT);
port = DEF_PORT;
}

if (WSAStartup(0x202,(WSADATA *)&buff[0]))
{
printf("WSAStart error %d\n",WSAGetLastError());
}

```

```

        return -1;
    }

    char namebuf[32];
    int rc;

    server.sin_addr.s_addr = inet_addr(ipbuf);
    server.sin_family = AF_INET;
    server.sin_port = htons(port);

    client_sockfd = socket(AF_INET , SOCK_DGRAM , 0);
    if (client_sockfd == -1)
    {
        printf("Could not create socket");
    }

    if (connect(client_sockfd , (struct sockaddr *)&server , sizeof(server)) <
    0)
    {
        perror("connect failed. Error");
        return 1;
    }
    printf("\nOk...\n");
    printf("Succses connect to server\n");

    printf("\nPlease, input your name. Must be < 30 character.\n");
    gets_s(&namebuf[0],sizeof(namebuf)-1);
    namebuf[31] = '\0';
    namebuf[30] = '\n';
    printf("\nOk...\n");

    memcpy(msg2.messageName, namebuf, sizeof(namebuf));
    msg2.messageType = 1;
    msg2.messageSizename = strlen(namebuf);
    sendto(client_sockfd, (char*)&msg2, sizeof(msg2), 0, (sockaddr *)&server,
    sizeof(server));
    msg2.messageType = 0;

    HANDLE t;
    int number = 0;
    t = CreateThread(NULL, 0, threadHandler, (LPVOID)number, 0, NULL);

    system("cls");
    printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");
    printf("Type '~menu~' for see menu\n\n");

    while(1)
    {
        printf("\n<- Your message \n");
        rc = _read(0,wrbuf,sizeof(wrbuf)-1);

        if (strncmp(wrbuf,"~menu~\n",7)==0)
        {
            menu(wrbuf,rc);
        }

        else
        {
            msg2.messageType = 0;
            msg2.messageLen = (rc-1);

```

```

        memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
        sendto(client_sockfd2, (char*)&msg2,
sizeof(msg2), 0, (sockaddr *)&server, sizeof(server));
        memset(&wrbuf, NULL, sizeof(wrbuf));
    }

}

DWORD WINAPI threadHandler(LPVOID param)
{
    sockaddr_in server_addr;
    int server_addr_size=sizeof(server_addr);

    int bsize=recvfrom(client_sockfd, (char*)&msg,
sizeof(msg), 0, (sockaddr *)&server_addr, &server_addr_size);

    if (bsize==SOCKET_ERROR)
    {
        printf("You was unconnected or kicked\n",
WSAGetLastError());
        closesocket(client_sockfd);
        return 0;
    }

    if(msg.messageType == 1)
    {

        int newport;
        newport = msg.messageID;
        server.sin_port = htons(msg.messageID);
        client_sockfd2 = socket(AF_INET , SOCK_DGRAM , 0);

        printf("NEW PORT IS: %i", server.sin_port);

        if (client_sockfd2 == -1)
        {
            printf("Could not create socket");
        }

        if (connect(client_sockfd2 , (struct sockaddr
*)&server , sizeof(server)) < 0)
        {
            perror("connect failed. Error");
            return 1;
        }
        printf("\nOk...\n");
        printf("Succses connect to server\n");
        closesocket(client_sockfd);
    }

    for(;;)
    {
        sockaddr_in server_addr2;
        int server_addr_size2=sizeof(server_addr2);

        int bsize=recvfrom(client_sockfd2, (char*)&msg,
sizeof(msg), 0, (sockaddr *)&server_addr2, &server_addr_size2);

        if (bsize==SOCKET_ERROR)
        {

```

```

        printf("You was unconnected or kicked\n",
WSAGetLastError());
        closesocket(client_sockfd2);
        return 0;
    }

    if((msg.messageType == 0) || (msg.messageType == 2))
    {
        Beep(1000,50);
        printf("-> Message from %s\n",msg.messageName);

        int i;
        for (i=0;i<msg.messageLen;i++)
        {
            printf("%c",msg.messageMes[i]);
        }

        printf("\n\n<- Your message \n");
    }
}
return 0;
}

```

Листинг 5. Server_TCP_Linux.c

```

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <signal.h>

#define DEF_PORT 8888
#define DEF_IP "127.0.0.1"

fd_set readfds, testfds;
pthread_t threads[128];

char names[32][32];
char wrbuf[512];
int fd;
int clients[128];
int sizenames[128];
int c = 0;

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};

```

```

#pragma pack(pop)

int cliview()
{
    int i = 0;
    int y = 0;

    if(c == 0)
    {
        printf("No clients\n");
    }

    for(i=0;i<c;i++)
    {
        printf("Clients[%i] sock = %i  ", i, clients[i]);

        for(y=0;y<sizeofnames[i];y++)
        {
            printf("%c", names[i][y]);
        }
        printf("\n");
    }

    return 0;
}

int thrview()
{
    int i =0;

    if(c == 0)
    {
        printf("No thr\n");
    }

    for(i=0;i<c;i++)
    {
        printf("thr[%i] = %i\n", i, (int)threads[i]);
    }

    return 0;
}

int clirm(int sock)
{
    int i = 0;
    int i1;
    int i2;
    while(i < c)
    {
        if(clients[i] == sock)
        {
            FD_CLR(clients[i], &readfds);

            c=c-1;
        }
    }
}

```



```

        for(i1=i;i1<c;i1++)
        {
            clients[i1] = clients[i1+1];
            sizenames[i1] = sizenames[i1+1] ;

            for(i2=0;i2<32;i2++)
            {
                names[i1][i2] = names[i1+1][i2];
            }
        }
        else i++;
    }

return 0;
}

void obr()
{
pthread_exit(0);
signal(SIGUSR1,obr);
}

void* clientHandler(void* args)
{
    signal(SIGUSR1,obr);
    int sock = (int)args;
    int res = 0;
    struct Message msg;

    for(;;)
    {

        res = recv(sock, (char*)&msg, sizeof(msg) ,0);
        if (res == 0)
        {
            printf("socket with id = %i was unconnected\n", sock);
            clirm(sock);
            cliview();
            close(sock);
            pthread_exit(NULL);
        }

        printf("\nClient messageLen: %i\n", msg.messageLen);
        printf("Client messageTo: %i\n", msg.messageID);
        printf("Client messageType: %i\n", msg.messageType);
        printf("Client messageMes: ");

        int i;
        for (i=0;i<msg.messageLen;i++)
        {
            printf("%c",msg.messageMes[i]);
        }

        printf("\n");
        printf("Client messageFrom: %s\n", msg.messageName);
    }
}

```

```

int type;
type = msg.messageType;

if(type == 0)
{
    //PüPüCßPüPsC,PiCßP°PIPeP° CíPeP°P·P°PSPSPsPjCí
    PeP»PëPüPSC,Cí//
    send(msg.messageID, (char*)&msg, sizeof(msg), 0);
}

if(type == 1)
{
    int k = 0;
    int u = 0;

    for(k=0;k<128;k++)
    {
        if(clients[k]==sock)
        {
            sizenames[k] = msg.messageSizename;

            for(u=0;u<32;u++)
            {
                names[k][u] = msg.messageName[u];
            }
        }
    }
}

if(type == 2)
{
    int x = 0 ;
    char str[512];
    char sername[] = "Online Clients From Server \n";
    for (x=0;x<c;x++)
    {
        msg.messageLen = sizeof(str)-1;

        memcpy(msg.messageName, sername, sizeof(sername));

        msg.messageSizename = strlen(msg.messageName);

        sprintf(str,"%i)User onlain: %s\n ID user: %i",x+1,
&names[x],clients[x]);
        memcpy(msg.messageMes, str, sizeof(str)-1);
        send(sock, (char*)&msg, sizeof(msg)-1, 0);
        sleep(1);
        memset(&str,NULL,sizeof(str));
    }
}
}
}

```

```

int menu(char wrbuf[512])
{
    int wrbuf2[8];
    char wrbuf3[8];
    int numcli;
    int nummenu;

    if (!strcmp(&wrbuf[0], "~menu~\n"))
    {
        system("clear");
        printf("*****Menu*****\n");
        printf("1) Exit from menu\n");
        printf("2) Delete client\n");
        printf("3) View list of onlain client\n");

        fgets(&wrbuf2[0], sizeof(wrbuf2)-1, stdin);
        nummenu = atoi(&wrbuf2[0]);

        if (nummenu == 1)
        {
            system("clear");
            return 0;
        }

        if (nummenu == 2)
        {
            system("clear");
            printf("Please, choose ID of client for deleting\n");
            cliview();
            printf("\nSlect ID: ");

            fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
            numcli = atoi(&wrbuf3[0]);

            system("clear");
            pthread_kill(threads[numcli], SIGUSR1);
            close(clients[numcli]);
            clirm(clients[numcli]);

            printf("removing %s with ID = %i.\n Press Enter for exit to
main window\n", &names[numcli], numcli);

            fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
            if (!strcmp(&wrbuf3[0], "\n"))
            {
                system("clear");
                return 0;
            }
            return 0;
        }

        if (nummenu == 3)
        {
            system("clear");

```

```

        printf("*****The list of OnLine clients*****\n");
        cliview();
        return 0;
    }

    }
    return 0;
}

int main()
{
    signal(SIGUSR1,SIG_IGN);
    char ipbuf[18];
    char portbuf[10];

    int port = 0;

    for(;;)
    {

        system("clear");
        printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");

        int ip1 = 0;
        int ip2 = 0;
        int ip3 = 0;
        int ip4 = 0;

        char cip1[10];
        char cip2[10];
        char cip3[10];
        char cip4[10];

        printf("Please, choose IP-addres for using\n");
        printf("Like this: xxx.xxx.xxx.xxx\n");
        printf("xxx must be in range from 0 to 255\n");
        printf("For example: 127.000.000.001\n\n");

        fgets(&ipbuf[0],sizeof(ipbuf)-1,stdin);

        ip1= atoi(&ipbuf[0]);
        ip2= atoi(&ipbuf[4]);
        ip3= atoi(&ipbuf[8]);
        ip4= atoi(&ipbuf[12]);

        sprintf(cip1,"%03i",ip1);
        sprintf(cip2,"%03i",ip2);
        sprintf(cip3,"%03i",ip3);
        sprintf(cip4,"%03i",ip4);

        if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2] !=
        ipbuf[2])) ||
            ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
        ipbuf[6])) ||

```

```

        ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
        ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2] !=
ipbuf[14]))))

    || (((ip1<0) || (ip1>255)) ||
        ((ip2<0) || (ip2>255)) ||
        ((ip3<0) || (ip3>255)) ||
        ((ip4<0) || (ip4>255))))

{
printf("\nError address!\n");
printf("Follow the rules!\n");
printf("Please repeat!\n");
sleep(3);
}
else

{
printf("Your address = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);
break;
}
}

printf("Please, choose PORT for using\n");
fgets(&portbuf[0],sizeof(portbuf)-1,stdin);
port = atoi(&portbuf[0]);
printf("Your port = %i\n", port);

if ((port < 0) || (port > 65535))
{
printf("Uncorrect port = %i\n", port);
printf("Using default port %d\n",DEF_PORT);
port = DEF_PORT;
}

struct sockaddr_in listenerInfo;
listenerInfo.sin_family = AF_INET;
listenerInfo.sin_port = htons( port );
listenerInfo.sin_addr.s_addr = inet_addr(ipbuf);

int listener = socket(AF_INET, SOCK_STREAM, 0 );
if ( listener < 0 ) {
perror( "Can't create socket to listen: " );
exit(1);
}

int res = bind(listener, ( struct sockaddr * )&listenerInfo, sizeof(
listenerInfo ) );
if ( res < 0 ) {
perror( "Can't bind socket" );
exit( 1 );
}

res = listen(listener,5);

```

```

if (res) {
perror("Error while listening:");
exit(1);
}

FD_ZERO(&readfds);
FD_SET(listener, &readfds);
FD_SET(0, &readfds);

int ressel;

for(;;)
{
    testfds = readfds;

    ressel = select(FD_SETSIZE, &testfds, (fd_set *)0, (fd_set *)0,
(struct timeval *)0);
    if (ressel < 0)
    {
        perror("error select");
        exit(1);
    }

    for (fd = 0; fd < FD_SETSIZE; fd++)
    {

        if (FD_ISSET(fd, &testfds))

        {

            if (fd == listener)
            {
                int client = accept(listener, NULL, NULL );
                printf("New client connected with id = %i\n", client);
                clients[c] = client;

                pthread_t thrd;
                res = pthread_create(&thrd, NULL, clientHandler, (void
*) (client));
                if (res)
                {
                    printf("Error while creating new thread\n");
                }
                threads[c] = thrd;
                c++;
                cliview();
                thrview();

            }

            if (fd == 0)
            {
                read(0, wrbuf, sizeof(wrbuf)-1);
                menu(wrbuf);
            }

        }

    }
}

```

```

    }
}

return 0;
}

```

Листинг 6. Client_TCP_Linux.c

```

#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<math.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <stdlib.h>

#define MAXBUF 1024
#define DEF_PORT 8888

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizenam;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

char wrbuf[512];
struct Message msg;
struct Message msg2;
fd_set readfds, testfds;

int menu(char wrbuf[512])
{
    int nummenu;
    int numid;
    char wrbuf2[8];
    char wrbuf3[8];

    if (!strcmp(&wrbuf[0], "~menu~\n"))
    {
        system("clear");
        printf("*****Menu*****\n");
        printf("1) Select ID of client\n");
        printf("2) View OnLine client\n");
        printf("3) Exit from menu\n");

        fgets(&wrbuf2[0], sizeof(wrbuf2)-1, stdin);
    }
}

```

```

        nummenu = atoi(&wrbuf2[0]);

    if (nummenu == 1)
    {

        printf("Select ID: ");

        fgets(&wrbuf3[0], sizeof(wrbuf3)-1, stdin);
        numid = atoi(&wrbuf3[0]);

        msg2.messageID = numid;
        system("clear");
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

    if (nummenu == 3)
    {
        system("clear");
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

    if (nummenu == 2)
    {
        system("clear");

        msg2.messageType = 2;
        msg2.messageLen = 0;
        memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
        send(3, (char*)&msg2, sizeof(msg2), 0);

        msg2.messageType = 0;
        memset(&wrbuf, NULL, sizeof(wrbuf));
        return 0;
    }

}

return 0;
}

int main()
{

    char ipbuf[18];
    char portbuf[10];

    int port = 0;

    for(;;)
    {

```



```

system("clear");
printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");

int ip1 = 0;
int ip2 = 0;
int ip3 = 0;
int ip4 = 0;

char cip1[10];
char cip2[10];
char cip3[10];
char cip4[10];

printf("Please, choose IP-addres of server\n");
printf("Like this: xxx.xxx.xxx.xxx\n");
printf("xxx must be in range from 0 to 255\n");
printf("For example: 127.000.000.001\n\n");

fgets(&ipbuf[0], sizeof(ipbuf)-1, stdin);

ip1= atoi(&ipbuf[0]);
ip2= atoi(&ipbuf[4]);
ip3= atoi(&ipbuf[8]);
ip4= atoi(&ipbuf[12]);

sprintf(cip1,"%03i",ip1);
sprintf(cip2,"%03i",ip2);
sprintf(cip3,"%03i",ip3);
sprintf(cip4,"%03i",ip4);

if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2]
!= ipbuf[2])) ||
    ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
    ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
    ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2]
!= ipbuf[14])))

    || (((ip1<0) || (ip1>255)) ||
        ((ip2<0) || (ip2>255)) ||
        ((ip3<0) || (ip3>255)) ||
        ((ip4<0) || (ip4>255))))

{
printf("\nError addres!\n");
printf("Follow the rules!\n");
printf("Please repeat!\n");
sleep(3);
}
else

{
printf("\nOk...\n");
printf("Addres of server = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);

```

```

        break;
    }
}

printf("Please, choose PORT of server\n");
fgets(&portbuf[0], sizeof(portbuf)-1, stdin);
port = atoi(&portbuf[0]);
printf("\nOk...\n");
printf("Port of server = %i\n", port);

if ((port < 0) || (port > 65535))
{
    printf("Uncorrect port = %i\n", port);
    printf("Using default port %d\n", DEF_PORT);
    port = DEF_PORT;
}

char namebuf[32];
int client_sockfd;
struct sockaddr_in server;
int result;
int rc;

server.sin_addr.s_addr = inet_addr(ipbuf);
server.sin_family = AF_INET;
server.sin_port = htons(port);

client_sockfd = socket(AF_INET , SOCK_STREAM , 0);
if (client_sockfd == -1)
{
    printf("Could not create socket");
}

if (connect(client_sockfd , (struct sockaddr *)&server ,
sizeof(server)) < 0)
{
    perror("connect failed. Error");
    return 1;
}
printf("\nOk...\n");
printf("Succses connect to server\n");

FD_ZERO(&readfds);
FD_SET(client_sockfd, &readfds);
FD_SET(0, &readfds);

printf("\nPlease, input your name. Must be < 30 character.\n");
gets(&namebuf[0]);
namebuf[31] = '\0';
namebuf[30] = '\n';
printf("\nOk...\n");

memcpy(msg2.messageName, namebuf, sizeof(namebuf));
msg2.messageType = 1;
msg2.messageSizename = strlen(namebuf);
send(3, (char*)&msg2, sizeof(msg2), 0);

```

```

msg2.messageType = 0;

system("clear");
printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");
printf("Type '~menu~' for see menu\n\n");

while(1)
{
    int fd;
    int nread;

    testfds = readfds;

    result = select(FD_SETSIZE, &testfds, (fd_set *)0, (fd_set *)0,
        (struct timeval *)0);

    if (result < 0)
    {
        perror("error select");
        exit(1);
    }

    for (fd = 0; fd < FD_SETSIZE; fd++)
    {
        if (FD_ISSET(fd, &testfds))
        {
            //PsP±CᔡP°P±PsC,PeP° PIPiPsPrP° Cᔡ PeP»P°PIPᔡP°C,CᔡCᔡCᔡCᔡ
            if (fd == 0)
            {
                printf("\n<- Your message \n");
                rc = read(0,wrbuf,sizeof(wrbuf)-1);

                if (!strcmp(&wrbuf[0],"~menu~\n"))
                {
                    menu(wrbuf);
                }
                else
                {
                    msg2.messageLen = (rc-1); // htonl
                    memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
                    send(3,(char*)&msg2, sizeof(msg2),0);

                    //printf("Your message \n");
                    memset(&wrbuf,NULL,sizeof(wrbuf));}
            }

            //PᔡCᔡPᔡPᔡPᔡ CᔡPᔡPsPᔡCᔡ%PᔡPᔡSPᔡPᔡPᔡ Pᔡ PsPᔡCᔡP°P±PsC,PeP°
            PsC,PeP»CᔡCᔡPᔡPᔡSPᔡCᔡ PsC, CᔡPᔡCᔡPIPᔡCᔡP°
            else
            {
                ioctl(fd, FIONREAD, &nread);
            }
        }
    }
}

```

```

        if ((nread == 0))
        {
            close(fd);
            FD_CLR(fd, &readfds);
            printf("Server was unconnected or kicked you!");
        }

        else
        {
            rc = recv(fd, (char*)&msg , sizeof(msg) ,0);
            if (rc == 0)
                perror("uncorrect socket from select");
            if (rc < 0)
                perror("error vizova recv");

            // printf("Server messageLen: %i\n", msg.messageLen);
            // printf("Server messageID: %i\n", msg.messageID);
            printf("-> Message from %s\n",msg.messageName);

            int i;
            for (i=0;i<msg.messageLen;i++)
            {
                printf("%c",msg.messageMes[i]);
            }

            printf("\n\n");
            printf("<- Your message \n");
        }
    }
}
}
}

```

Листинг 7. Server_UDP_Linux.c

```

#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <signal.h>

```

```

#define DEF_PORT 8888
#define DEF_IP "127.0.0.1"

//DWORD WINAPI threadHandler(LPVOID);
//DWORD WINAPI clientHandler(LPVOID);

char names[32][32];
char wrbuf[512];
int fd;
int clients[128];
int sizenames[128];
int c = 0;
//HANDLE threads[128];
pthread_t threads[128];
struct sockaddr_in soki[512];
struct sockaddr_in newlistenerInfo;

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

int cliview()
{
    int i = 0;
    int y = 0;

    if(c == 0)
    {
        printf("No clients\n");
    }

    for(i=0;i<c;i++)
    {
        printf("Clients[%i] sock = %i  ", i, clients[i]);
        printf("Name: ",i);

        for(y=0;y<sizenames[i];y++)
        {
            printf("%c", names[i][y]);
        }
        printf("\n");
    }

    return 0;
}

```

```

int thrview()
{
    int i =0;

    if(c == 0)
    {
        printf("No thr\n");
    }

    for(i=0;i<c;i++)
    {
        printf("thr[%i] = %i\n", i, threads[i]);
    }
return 0;
}

int clirm(int sock)
{
    int i = 0;
    int i1;
    int i2;
    while(i < c)
    {
        if(clients[i] == sock)
        {
            c=c-1;

            for(i1=i;i1<c;i1++)
            {
                clients[i1] = clients[i1+1];
                sizenames[i1] = sizenames[i1+1] ;
                soki[i1] = soki[i1+1];

                for(i2=0;i2<32;i2++)
                {
                    names[i1][i2] = names[i1+1][i2];
                }
            }
            else i++;
        }
    }

return 0;
}

int menu(char wrbuf[512])
{
    char wrbuf2[8];
    char wrbuf3[8];
    int numcli;
    int nummenu;

    if (strncmp(wrbuf, "~menu~\n", 7)==0)
    {
        system("clear");
    }
}

```

```

printf("*****Menu*****\n");
printf("1) Exit from menu\n");
printf("2) Delete client\n");
printf("3) View list of onlain client\n");

fgets(&wrbuf2[0],sizeof(wrbuf2)-1,stdin);
nummenu = atoi(&wrbuf2[0]);

if (nummenu == 1)
{
    system("clear");
    return 0;
}

if (nummenu == 2)
{
    system("clear");
    printf("Please, choose ID of client for deleting\n");
    cliview();
    printf("\nSlect ID: ");

    fgets(&wrbuf3[0],sizeof(wrbuf3)-1,stdin);
    numcli = atoi(&wrbuf3[0]);

    //TerminateThread(threads[numcli],NULL);
    pthread_kill(threads[numcli],SIGUSR1);
    close(clients[numcli]);
//    ZeroMemory(&soki[numcli], sizeof(struct sockaddr_in));
//    ZeroMemory(&clients[numcli], sizeof(int));
//    ZeroMemory(&sizenames[numcli], sizeof(int));
    memset(&soki[numcli],NULL,sizeof(struct sockaddr_in));
    memset(&clients[numcli],NULL,sizeof(int));
    memset(&sizenames[numcli],NULL,sizeof(int));
    //CloseHandle(threads[numcli]);
    clirm(clients[numcli]);

    system("clear");
    printf("removing %s with ID = %i.\n Press Enter for exit to
main window\n",&names[numcli], numcli);

    fgets(&wrbuf3[0],sizeof(wrbuf3)-1,stdin);
    if (strncmp(wrbuf3,"\n",1)==0)
    {
        system("clear");
        return 0;
    };

    return 0;
}

if (nummenu == 3)
{
    system("clear");
    printf("*****The list of OnLine clients*****\n");
    cliview();
    return 0;
}

```

```

    }

    }
    return 0;
}

void obr()
{
pthread_exit(0);
signal(SIGUSR1, obr);
}

void* threadHandler(void* args)
{
    for(;;)
    {
        read(0, wrbuf, sizeof(wrbuf)-1);
        menu(wrbuf);
    }
    return 0;
}

void* clientHandler(void* args)
{
    signal(SIGUSR1, obr);
    int sock = args;
    int res = 0;
    struct Message msg;
    struct sockaddr_in client_addr;
    int client_addr_size = sizeof(client_addr);

    res = recvfrom(sock, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*) &client_addr, &client_addr_size);
    soki[c-1].sin_port = client_addr.sin_port;

    if (res == 0)
    {
        printf("socket with id = %i was unconnected\n", sock);
        clirm(sock);
        cliview();
        close(sock);

        int h = 0;
        for (h=0;h<c;h++)
        {
            if(clients[h] == sock)
            {
                memset(&soki[h], NULL, sizeof(struct sockaddr_in));
                memset(&clients[h], NULL, sizeof(int));
                memset(&sizenames[h], NULL, sizeof(int));
            }
        }

        return 0;
    }
}

```



```

printf("\nClient messageLen: %i\n", msg.messageLen);
printf("Client messageTo: %i\n", msg.messageID);
printf("Client messageType: %i\n", msg.messageType);
printf("Client messageMes: ");

int i;
for (i=0;i<msg.messageLen;i++)
{
printf("%c",msg.messageMes[i]);
}

printf("\n");
printf("Client messageFrom: %s\n", msg.messageName);

int type;
type = msg.messageType;

if(type == 0)
{
    int r = 0;
    for (r=0;r<c;r++)
    {
        if (msg.messageID == clients[r])
        {
            break;
        }
    }

    sendto(msg.messageID, (char*)&msg, sizeof(msg), 0, (struct
sockaddr *)&soki[r], sizeof(soki[r]));
}

if(type == 2)
{
    int x = 0 ;
    char str[512];
    char sername[] = "Online Clients From Server \n";
    for (x=0;x<c;x++)
    {
        memcpy(msg.messageName, sername, sizeof(sername));
        msg.messageSizenam = strlen(msg.messageName);
        sprintf(str,"%i)User onlain: %s\n ID user: %i",x+1,
&names[x],clients[x]);
        msg.messageLen = strlen(str);
        memcpy(msg.messageMes, str, sizeof(str)-1);
        sendto(sock, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*)&client_addr, sizeof(client_addr));
        sleep(1);
        memset(&str,NULL,sizeof(str));
    }
}

for(;;)
{
    res = 0;
    res = recvfrom(sock, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*)&client_addr, &client_addr_size);
}

```

```

if (res == 0)
{
printf("socket with id = %i was unconnected\n", sock);
clirm(sock);
cliview();
close(sock);

int h = 0;
for (h=0;h<c;h++)
{
    if(clients[h] == sock)
    {
        memset(&soki[h],NULL,sizeof(struct sockaddr_in));
        memset(&clients[h],NULL,sizeof(int));
        memset(&sizenames[h],NULL,sizeof(int));
    }
}

printf("\nClient messageLen: %i\n", msg.messageLen);
printf("Client messageTo: %i\n", msg.messageID);
printf("Client messageType: %i\n", msg.messageType);
printf("Client messageMes: ");

int i;
for (i=0;i<msg.messageLen;i++)
{
printf("%c",msg.messageMes[i]);
}

printf("\n");
printf("Client messageFrom: %s\n", msg.messageName);

int type;
type = msg.messageType;

if(type == 0)
{
    if((msg.messageMes[0] == '~') &&
        (msg.messageMes[1] == 'q') &&
        (msg.messageMes[2] == 'u') &&
        (msg.messageMes[3] == 'i') &&
        (msg.messageMes[4] == 't') &&
        (msg.messageMes[5] == '~') &&
        (msg.messageMes[6] == '\n'))
    {
        int h = 0;
        for (h=0;h<c;h++)
        {
            if(clients[h] == sock)
            {
                memset(&soki[h],NULL,sizeof(struct sockaddr_in));
                memset(&clients[h],NULL,sizeof(int));
                memset(&sizenames[h],NULL,sizeof(int));
                clirm(clients[h]);
            }
        }
    }
}

```

```

    }
    close(sock);
    return 0;
}

int r = 0;
for (r=0;r<c;r++)
{
    if (msg.messageID == clients[r])
    {
        break;
    }
}

sendto(msg.messageID, (char*)&msg, sizeof(msg), 0, (struct
sockaddr *)&soki[r], sizeof(soki[r]));
}

if(type == 2)
{
    int x = 0 ;
    char str[512];
    char sername[] = "Online Clients From Server \n";
    for (x=0;x<c;x++)
    {
        memcpy(msg.messageName, sername, sizeof(sername));
        msg.messageSizenam = strlen(msg.messageName);
        sprintf(str,"%i)User onlain: %s\n ID user: %i",x+1,
&names[x],clients[x]);
        msg.messageLen = strlen(str);
        memcpy(msg.messageMes, str, sizeof(str)-1);
        sendto(sock, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*)&client_addr, sizeof(client_addr));
        sleep(1);
        memset(&str,NULL,sizeof(str));
    }
}

return 0;
}

int main()
{
    signal(SIGUSR1,SIG_IGN);
    char ipbuf[18];
    char portbuf[10];
    int port = 0;

    for(;;)
    {

        system("clear");
        printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");

        int ip1 = 0;
        int ip2 = 0;

```

```

int ip3 = 0;
int ip4 = 0;

char cip1[10];
char cip2[10];
char cip3[10];
char cip4[10];

printf("Please, choose IP-address for using\n");
printf("Like this: xxx.xxx.xxx.xxx\n");
printf("xxx must be in range from 0 to 255\n");
printf("For example: 127.000.000.001\n\n");

fgets(&ipbuf[0], sizeof(ipbuf)-1, stdin);

ip1= atoi(&ipbuf[0]);
ip2= atoi(&ipbuf[4]);
ip3= atoi(&ipbuf[8]);
ip4= atoi(&ipbuf[12]);

sprintf(cip1,"%03i",ip1);
sprintf(cip2,"%03i",ip2);
sprintf(cip3,"%03i",ip3);
sprintf(cip4,"%03i",ip4);

if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2] !=
ipbuf[2])) ||
    ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
    ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
    ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2] !=
ipbuf[14])))

    || (((ip1<0) || (ip1>255)) ||
        ((ip2<0) || (ip2>255)) ||
        ((ip3<0) || (ip3>255)) ||
        ((ip4<0) || (ip4>255))))

{
printf("\nError adres!\n");
printf("Follow the rules!\n");
printf("Please repeat!\n");
sleep(3);
}
else

{
printf("Your adres = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);
break;
}
}

printf("Please, choose PORT for using\n");
fgets(&portbuf[0], sizeof(portbuf)-1, stdin);

```

```

port = atoi(&portbuf[0]);
printf("Your port = %i\n", port);

if ((port < 0) || (port > 65535))
{
printf("Uncorrect port = %i\n", port);
printf("Using default port %d\n", DEF_PORT);
port = DEF_PORT;
}

struct sockaddr_in listenerInfo;
listenerInfo.sin_family = AF_INET;
listenerInfo.sin_port = htons( port );
listenerInfo.sin_addr.s_addr = inet_addr(ipbuf);

int listener = socket(AF_INET, SOCK_DGRAM, 0 );
if ( listener < 0 ) {
perror( "Can't create socket to listen: " );
exit(1);
}

int res = bind(listener, ( struct sockaddr * )&listenerInfo, sizeof(
listenerInfo ) );
if ( res < 0 ) {
perror( "Can't bind socket" );
exit( 1 );
}

int number = 0;
pthread_t t;
int res24 = pthread_create(&t, NULL, threadHandler, (void *) (number));
if (res24)
{
printf("Error while creating new thread\n");
}

system("clear");
printf("*****Khutornoy's server from SPBSPU v1.0*****\n\n");
printf("Type '~menu~' for see menu\n\n");

struct Message msg;

for(;;)
{
    struct sockaddr_in client_addr;
    int client_addr_size = sizeof(client_addr);
    int bsize = recvfrom(listener, (char*)&msg, sizeof(msg), 0,
(struct sockaddr *) &client_addr, &client_addr_size);

    soki[c] = client_addr;

    if (bsize==0) printf("recvfrom() error: %d\n");

    printf("\nClient messageLen: %i\n", msg.messageLen);
    printf("Client messageTo: %i\n", msg.messageID);
    printf("Client messageType: %i\n", msg.messageType);
}

```

```

printf("Client messageMes: ");

int i;
for (i=0;i<msg.messageLen;i++)
{
printf("%c",msg.messageMes[i]);
}

printf("\n");
printf("Client messageFrom: %s\n", msg.messageName);

int type;
type = msg.messageType;

if(type==1)
{

    sizenames[c] = msg.messageSizename;

    int u = 0;
    for(u=0;u<32;u++)
    {
names[c][u] = msg.messageName[u];
    }

    msg.messageSizename = 0;
    msg.messageLen = 0;
    msg.messageID = 10000+c;
    msg.messageType = 1;
    sendto(listener, (char*)&msg, sizeof(msg), 0, (struct sockaddr
*)&client_addr, sizeof(client_addr));

    newlistenerInfo.sin_family = AF_INET;
    newlistenerInfo.sin_port = htons( msg.messageID );
    newlistenerInfo.sin_addr = client_addr.sin_addr;

    int newlistener = socket(AF_INET, SOCK_DGRAM, 0 );
    if ( newlistener < 0 ) {
perror( "Can't create socket to listen: " );
exit(1);
    }

    int res = bind(newlistener, ( struct sockaddr *
*)&newlistenerInfo, sizeof(newlistenerInfo) );
    if ( res < 0 ) {
perror( "Can't bind socket" );
exit( 1 );
    }

    soki[c].sin_port = newlistenerInfo.sin_port;

    pthread_t t2;
    int res123 = pthread_create(&t2, NULL, clientHandler, (void
*)(newlistener));
    if (res123)
    {
printf("Error while creating new thread\n");
    }
}

```

```

    }

    printf("New client connected with id = %i\n", newlistener);

    clients[c] = newlistener;
    threads[c] = t2;
    c++;

    cliview();
    thrview();

}
}
return 0;
}

```

Листинг 8. Client_UDP_Linux.c

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define MAXBUF 1024
#define DEF_PORT 8888

#pragma pack(push,1)
struct Message {
int messageType;
int messageLen;
int messageID;
int messageSizename;
char messageName[32];
char messageMes[512];
};
#pragma pack(pop)

char wrbuf[512];
struct Message msg;
struct Message msg2;
int client_sockfd;
int client_sockfd2;
struct sockaddr_in server;

void* threadHandler(void* args)
{
    struct sockaddr_in server_addr;
    int server_addr_size=sizeof(server_addr);

```

```

        int bsize=recvfrom(client_sockfd, (char*)&msg,
sizeof(msg), 0, (struct sockaddr *)&server_addr, &server_addr_size);

        if (bsize==0)
        {
            printf("You was unconnected or kicked\n");
            close(client_sockfd);
            return 0;
        }

        if(msg.messageType == 1)
        {

            int newport;
            newport = msg.messageID;
            server.sin_port = htons(msg.messageID);
            client_sockfd2 = socket(AF_INET , SOCK_DGRAM , 0);

            printf("NEW PORT IS: %i", server.sin_port);

            if (client_sockfd2 == -1)
            {
                printf("Could not create socket");
            }

            if (connect(client_sockfd2 , (struct sockaddr
*&server , sizeof(server)) < 0)
            {
                perror("connect failed. Error");
                return 1;
            }
            printf("\nOk...\n");
            printf("Succses connect to server\n");
            close(client_sockfd);
        }

        for(;;)
        {
            struct sockaddr_in server_addr2;
            int server_addr_size2=sizeof(server_addr2);

            int bsize=recvfrom(client_sockfd2, (char*)&msg,
sizeof(msg), 0, (struct sockaddr *)&server_addr2, &server_addr_size2);

            if (bsize==0)
            {
                printf("You was unconnected or kicked\n");
                close(client_sockfd2);
                return 0;
            }

            if((msg.messageType == 0)|| (msg.messageType == 2))
            {
                printf("-> Message from %s\n",msg.messageName);

                int i;
                for (i=0;i<msg.messageLen;i++)

```



```

        {
            printf("%c",msg.messageMes[i]);
        }
        printf("\n\n<- Your message \n");
    }
}
return 0;
}

int menu(char wrbuf[512],int rc)
{
    int nummenu;
    int numid;
    char wrbuf2[8];
    char wrbuf3[8];
    int len = rc;

    if (strncmp(wrbuf,"~menu~\n",7)==0)
    {
        system("clear");
        printf("*****Menu*****\n");
        printf("1) Select ID of client\n");
        printf("2) View OnLine client\n");
        printf("3) Exit from menu\n");

        fgets(&wrbuf2[0],sizeof(wrbuf2)-1,stdin);
        nummenu = atoi(&wrbuf2[0]);

        if (nummenu == 1)
        {
            printf("Select ID: ");
            fgets(&wrbuf3[0],sizeof(wrbuf3)-1,stdin);
            numid = atoi(&wrbuf3[0]);
            msg2.messageID = numid;

            system("clear");
            printf("*****Khutornoy's client from SPBSPU
v1.0*****\n\n");
            printf("Type '~menu~' for see menu\n\n");

            memset(&wrbuf,NULL,sizeof(wrbuf));
            return 0;
        }

        if (nummenu == 3)
        {
            system("clear");
            printf("*****Khutornoy's client from SPBSPU
v1.0*****\n\n");
            printf("Type '~menu~' for see menu\n\n");
            memset(&wrbuf,NULL,sizeof(wrbuf));
            return 0;
        }

        if (nummenu == 2)

```

```

        {
            system("clear");
            msg2.messageType = 2;
            msg2.messageLen = 0;
            memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
            sendto(client_sockfd2, (char*)&msg2, sizeof(msg2), 0,
(struct sockaddr *)&server, sizeof(server));

            msg2.messageType = 0;
            memset(&wrbuf, NULL, sizeof(wrbuf));
            return 0;
        }
    }
return 0;
}

```

```

int main()
{
    char ipbuf[18];
    char portbuf[10];
    char buff[1024];
    int port = 0;

    for(;;)
    {
        system("clear");
        printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");

        int ip1 = 0;
        int ip2 = 0;
        int ip3 = 0;
        int ip4 = 0;

        char cip1[10];
        char cip2[10];
        char cip3[10];
        char cip4[10];

        printf("Please, choose IP-addres of server\n");
        printf("Like this: xxx.xxx.xxx.xxx\n");
        printf("xxx must be in range from 0 to 255\n");
        printf("For example: 127.000.000.001\n\n");

        fgets(&ipbuf[0], sizeof(ipbuf)-1, stdin);

        ip1= atoi(&ipbuf[0]);
        ip2= atoi(&ipbuf[4]);
        ip3= atoi(&ipbuf[8]);
        ip4= atoi(&ipbuf[12]);

        sprintf(cip1,"%03i",ip1);
        sprintf(cip2,"%03i",ip2);
        sprintf(cip3,"%03i",ip3);
        sprintf(cip4,"%03i",ip4);

        if (((cip1[0] != ipbuf[0]) || (cip1[1] != ipbuf[1]) || (cip1[2]
!= ipbuf[2])) ||

```

```

        ((cip2[0] != ipbuf[4]) || (cip2[1] != ipbuf[5]) || (cip2[2] !=
ipbuf[6])) ||
        ((cip3[0] != ipbuf[8]) || (cip3[1] != ipbuf[9]) || (cip3[2] !=
ipbuf[10])) ||
        ((cip4[0] != ipbuf[12]) || (cip4[1] != ipbuf[13]) || (cip4[2]
!= ipbuf[14]))))

        || (((ip1<0) || (ip1>255)) ||
            ((ip2<0) || (ip2>255)) ||
            ((ip3<0) || (ip3>255)) ||
            ((ip4<0) || (ip4>255))))

    {
        printf("\nError adres!\n");
        printf("Follow the rules!\n");
        printf("Please repeat!\n");
        sleep(3);
    }
    else

    {
        printf("\nOk...\n");
        printf("Adres of server = %i.%i.%i.%i\n\n", ip1,ip2,ip3,ip4);
        break;
    }
}

printf("Please, choose PORT of server\n");
fgets(&portbuf[0],sizeof(portbuf)-1,stdin);
port = atoi(&portbuf[0]);
printf("\nOk...\n");
printf("Port of server = %i\n", port);

if ((port < 0) || (port > 65535))
{
    printf("Uncorrect port = %i\n", port);
    printf("Using default port %d\n",DEF_PORT);
    port = DEF_PORT;
}

char namebuf[32];
int rc;

server.sin_addr.s_addr = inet_addr(ipbuf);
server.sin_family = AF_INET;
server.sin_port = htons(port);

client_sockfd = socket(AF_INET , SOCK_DGRAM , 0);
if (client_sockfd == -1)
{
    printf("Could not create socket");
}

if (connect(client_sockfd , (struct sockaddr *)&server ,
sizeof(server)) < 0)
{
    perror("connect failed. Error");
}

```

```

return 1;
}
printf("\nOk...\n");
printf("Succses connect to server\n");

printf("\nPlease, input your name. Must be < 30 character.\n");
gets(&namebuf[0]);
namebuf[31] = '\0';
namebuf[30] = '\n';
printf("\nOk...\n");

memcpy(msg2.messageName, namebuf, sizeof(namebuf));
msg2.messageType = 1;
msg2.messageSizename = strlen(namebuf);
sendto(client_sockfd, (char*)&msg2, sizeof(msg2), 0, (struct sockaddr
*)&server, sizeof(server));
msg2.messageType = 0;

int number = 0;
pthread_t t;
int res24 = pthread_create(&t, NULL, threadHandler, (void *) (number));
if (res24)
{
printf("Error while creating new thread\n");
}

system("clear");
printf("*****Khutornoy's client from SPBSPU v1.0*****\n\n");
printf("Type '~menu~' for see menu\n\n");

while(1)
{
printf("\n<- Your message \n");
rc = read(0, wrbuf, sizeof(wrbuf)-1);

if (strncmp(wrbuf, "~menu~\n", 7) == 0)
{
menu(wrbuf, rc);
}

else
{
msg2.messageType = 0;
msg2.messageLen = (rc-1);
memcpy(msg2.messageMes, wrbuf, sizeof(wrbuf));
sendto(client_sockfd2, (char*)&msg2, sizeof(msg2),
0, (struct sockaddr *)&server, sizeof(server));
memset(&wrbuf, NULL, sizeof(wrbuf));
}
}
}

```