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

Отчёт по лабораторной работе
Дисциплина: Сети и телекоммуникационные технологии
Тема: Организация сетевого взаимодействия. Протокол TCP

Выполнил студент группы 43501/1 _____ У.А. Пеньевская
(подпись)

Преподаватель _____ А.О.Алексюк
(подпись)

22 декабря 2017 г.

Санкт-Петербург
2017

Содержание

1 Цель работы

Изучение принципов программирования сокетов с использованием протокола TCP.

2 Индивидуальное задание

Разработать распределенную систему, состоящую из приложений клиента и сервера, для распределенного выставления/просмотра курсов валют. Информационная система должна обеспечивать параллельную работу нескольких клиентов.

Основные возможности: Серверное приложение должно реализовывать следующие функции:

1. Прослушивание определенного порта
2. Обработка запросов на подключение по этому порту клиентов
3. Поддержка одновременной работы нескольких клиентов с использованием механизма нитей и средств синхронизации доступа к разделяемым между нитями ресурсам.
4. Принудительное отключение конкретного клиента
5. Добавление новой валюты (кода валюты)
6. Удаление валюты
7. Добавление курса конкретной валюты
8. Выдача пользователю списка имеющихся валют с текущими курсами и абсолютными/относительными приращениями к предыдущим значениям
9. Выдача пользователю истории курса конкретной валюты
10. *Сохранение состояния при выключении сервера

Клиентское приложение должно реализовывать следующие функции:

1. Возможность параллельной работы нескольких клиентов с одного или нескольких IP-адресов
2. Установление соединения с сервером (возможно, с регистрацией на сервере)
3. Разрыв соединения
4. Обработка ситуации отключения сервером
5. Получение и вывод списка валют с котировками/изменениями
6. Передача команды на добавление валюты
7. Передача команды на удаление валюты
8. Передача команды на добавление курса валюты
9. Получение и вывод истории котировок валюты

Настройка приложений:

Разработанное клиентское приложение должно предоставлять пользователю возможность задания IP-адреса или доменного имени сервера, а также номера порта сервера.

Тестирование: Для тестирования запускается сервер системы котировок валют и несколько клиентов. В процессе тестирования проверяются основные функциональные возможности разработанной системы.

3 Разработанный прикладной протокол

Протокол TCP имеет следующий шаблон сообщения:

<команда> <аттрибут> <аттрибут>

В начале сообщения, всегда присутствует тип команды, далее взависимости от команды могут идти(взависимости от типа команды) аттрибуты, которые отделены друг от друга пробелом.

	Код команды	Аттрибуты	Действия	Ответ сервера
1	A	<ID> <NAME> <SHORT NAME> <COURCE>	Добавление новой валюты Add New Finance Position	Added Finance Data: ID: Name: Short name: Course: absCource: conCource
2	D	<ID>	Удаление валюты Delete Finance Position	Finance Data: ID:0 Name: Short name: Course: 0.00 absCource: 0.00 conCource: 0.00
3	R	Отсутствуют	Выдача пользователю списка имеющихся валют с текущими курсами и абсолютными/относительными приращениями к предыдущим значениям Read All Finance Courcess	Finance Data: ID:0 Name: Short name: Course: 0.00 absCource: 0.00 conCource: 0.00
4	C	<ID> <COURCE>	Добавление курса конкретной валюты Add Finance Cource	Finance Data: ID:0 Name: Short name: Course: 0.00 absCource: 0.00 conCource: 0.00
5	O	Отсутствуют	Разрыв соединения Exit Programm	-disconnect
6	M	Отсутствуют	Вывод меню Show Menu	-
7	H	<ID>	Получение и вывод истории котировок валюты Show History	-

Таблица 1: Команды клиента

Список команд, которыми оперирует клиент:

```
| Add New Finance Position:   | A <ID> <NAME> <SHORT NAME> <COURCE>
| Delete Finance Position:    | D <ID>
| Read All Finance Courcess:  | R
| Add Finance Cource:         | C <ID> <COURCE>
| Exit Programm:              | O
| Show Menu:                  | M
| Show History                | H <ID>
```

Посылка от сервера – клиенту. Посылка состоит из символа типа char – команды, а также, при необходимости, - данных.

```
C - Client List
k <Socket number> - Kill klient
```

3.1 Описание структуры приложения

Сервер:
Функция main:

- Инициализация всех используемых переменных;
- Запуск событий, если таковые уже имеются;
- Создание сокета;
- Создание потока для прослушивания сокета;
- Цикл чтения команд.

Поток для прослушивания сокета:

- Прослушивание сокета;
- Добавление подключенного клиента в список;
- Создание сокета для работы с клиентом;
- Создание потока для работы с клиентом.

Цикл чтения команд:

- Чтение команды из стандартного ввода;
- Реакция на введенную команду (при корректном вводе команды) или игнорирование команды.

Поток для работы с клиентом:

- Получение сообщения от клиента;
- Анализ сообщения – является ли сообщение командой;
- Ответ на команду или вывод сообщения с пометкой отправителя.

Клиент:
Функция main:

- Чтение ip адреса сервера;
- Подключение к серверу;
- Создание потока для получения данных от сервера;
- Цикл чтения данных и отправка серверу.

Поток для получения данных от сервера:

- Побайтовое получение данных от сервера;
- Реакция на полученные данные .

4 Реализация программы

4.1 Структура проекта

При разработке приложения для операционной системы семейства Windows использовалась среда разработки Microsoft Visual Studio.

Язык программирования — C++.

4.2 Сетевая часть TCP

Клиентское приложение в TCP только отсылает команды на сервер, поэтому оно ничем не отличается от telnet клиента. Сервер обрабатывает команды, работает с коллекциями, сохраняет и загружает свое состояние, присылает уведомления и др. Делаем вывод, что клиентская программа потребляет ничтожно малый процессорный ресурс, в то время как сервер - наоборот.

На сервере, в первую очередь, происходит инициализация WinSock (на Windows), создание сокета (функция socket), привязка сокета к конкретному адресу (функция bind). Реализация инициализации сервера представлена в **Приложении 1.**

После этого ожидаем подключения клиентов в бесконечном цикле, с помощью функции accept. Если функция возвращает положительное значение, которое является клиентским сокетом, то создаем новый поток, в котором обрабатываем клиентские сообщения. Реализация подключения клиентов представлена в **Приложении 2.**

Клиентский поток вызывает функцию считывания символов в бесконечном цикле, как только при считывается знак перевода строки, функция возращает прочитанные символы. Если функция не вернула исключение, то посылаем команду на обработку, в противном случае это обозначает отключение клиента. Также отключение клиента может быть произведено извне обработчика клиентского потока, посредством закрытия клиентского сокета(функция считывания в этом случае сразу же вернет исключение) командой **k <Socket number>** (Kill klient). Реализация клиентского потока представлена в **Приложении 3**.

5 Тестирование

5.1 Тестирование серверного приложения

Запускаем клиент-серверное приложение

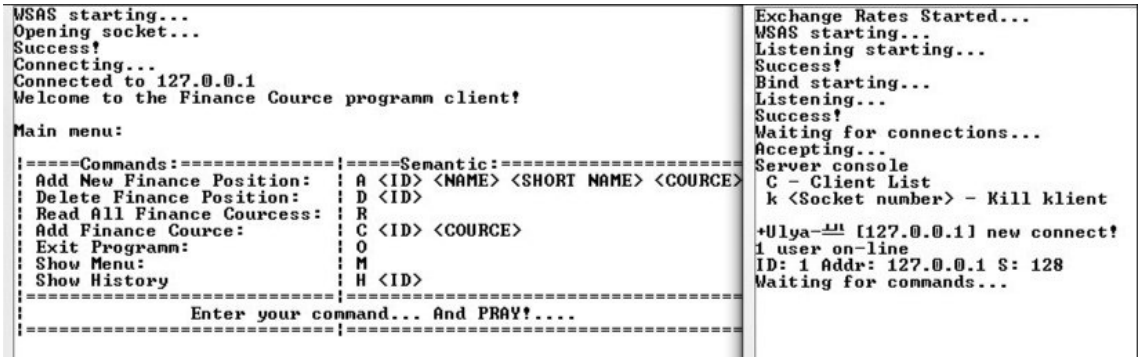

Рис. 1: Клиент-серверное приложение "Курс валют"

Запускаем еще одного клиента

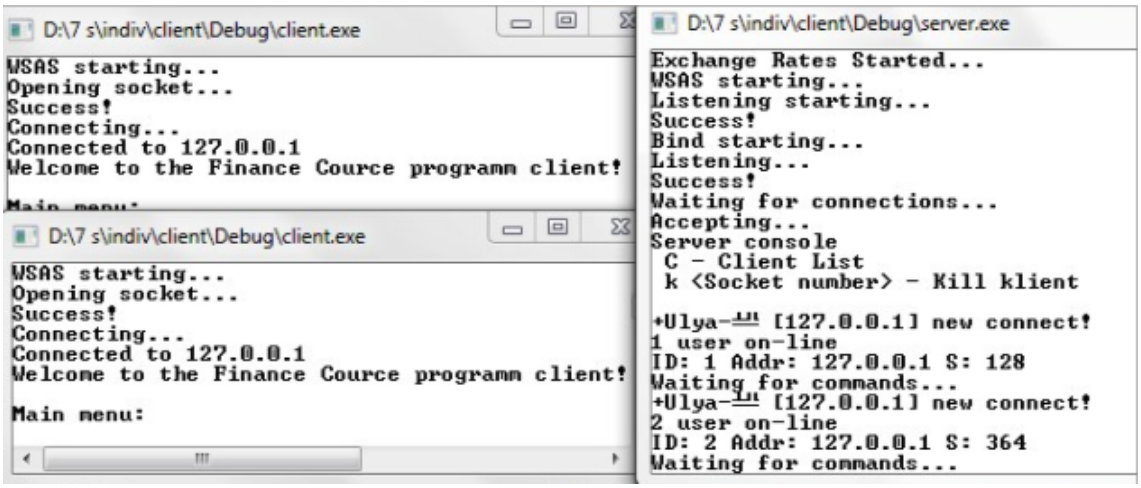

Рис. 2: Подключение второго клиента

Список подключенных клиентов можно посмотреть с помощью команды - C

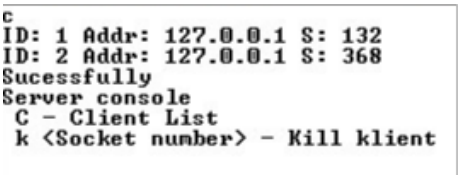

Рис. 3: Список подключенных клиентов

Отключить клиента - k <Socket number>

```
c
ID: 1 Addr: 127.0.0.1 S: 132
ID: 2 Addr: 127.0.0.1 S: 368
Sucessfully
Server console
 C - Client List
 k <Socket number> - Kill klient

k 132
Server console
 C - Client List
 k <Socket number> - Kill klient

recv failed with error: 10004
-disconnect
1 user on-line
Complited!
c
ID: 2 Addr: 127.0.0.1 S: 368
Sucessfully
Server console
 C - Client List
 k <Socket number> - Kill klient
```

Рис. 4: Отключение одного из клиентов

5.2 Тестирование клиентского приложения

Проверка работоспособности команд

Разорвем соединение с помощью команды O (Exit programm)

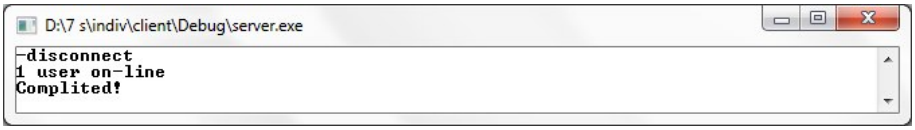

Рис. 5: Разрыв соединения

Добавим новую валюту вводом команды: A 1 RUB RU 70.23

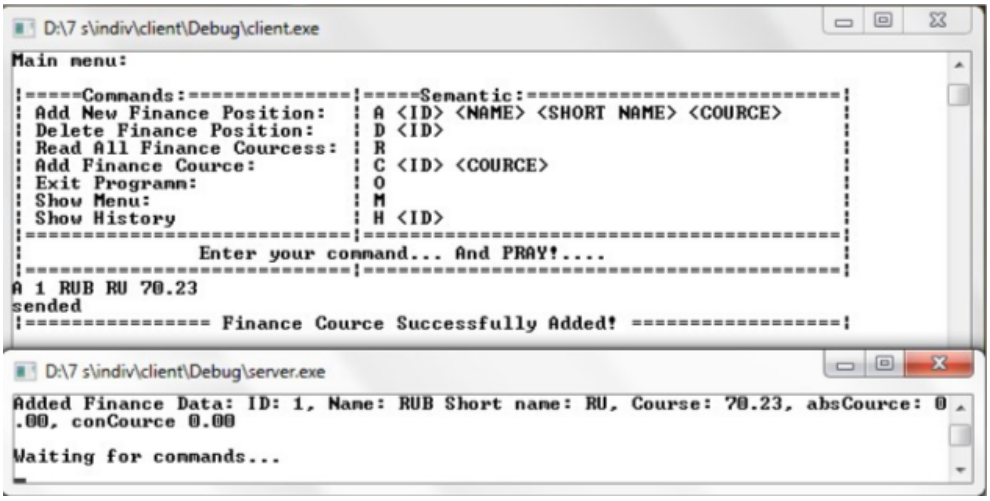

Рис. 6: Добавление новой валюты

Сервер выполнил команду добавление валюты (Added Finance Data)

Изменим курс валюты с помощью команды: A 1 30
На сервере получили сообщение о том, что курс валюты изменился.
absCourse = 70.23-30=40.23
Добавим еще одну валюту (A 2 USA US 1.23), удалим её (D 2) и прочитаем список всех валют (R)

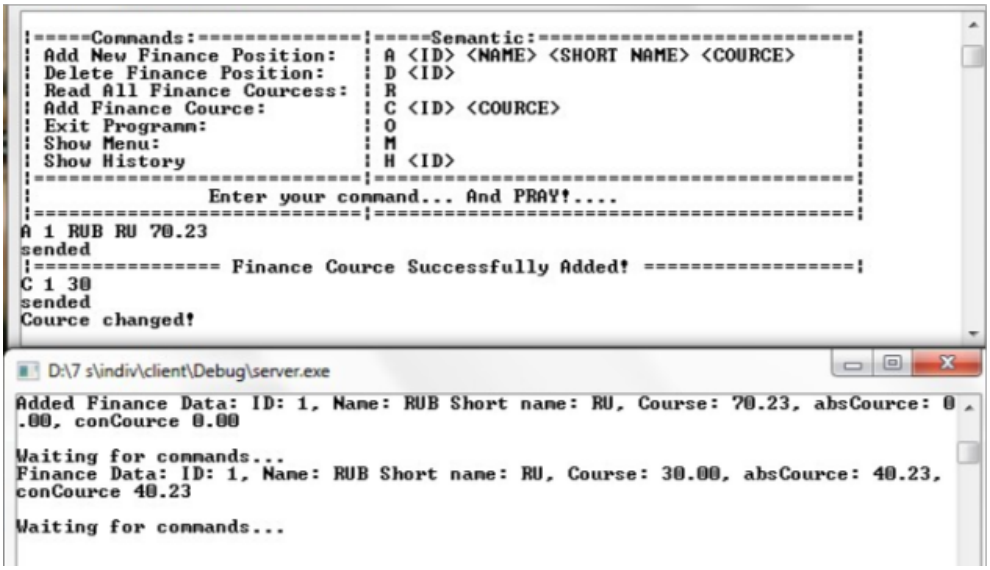

Рис. 7: Изменение курса валюты

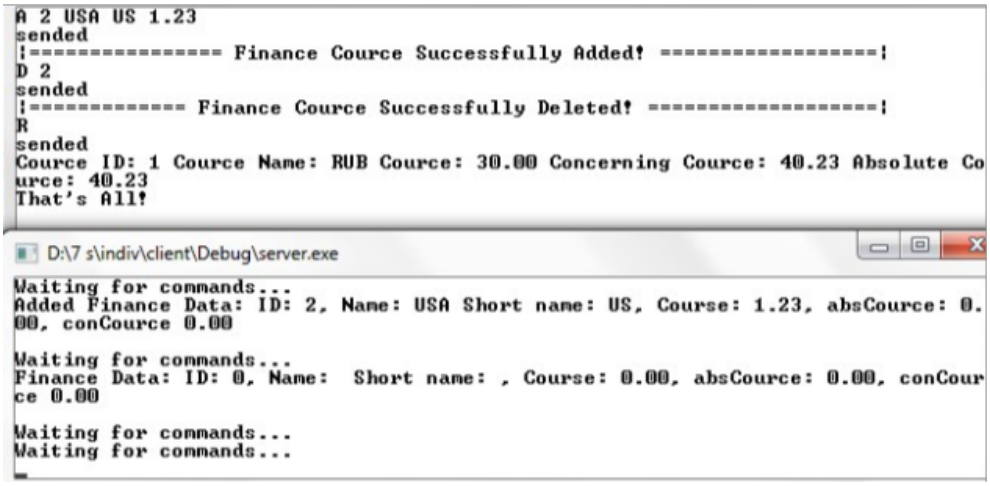

Рис. 8: Удаление валюты

Вывести меню еще раз командой M

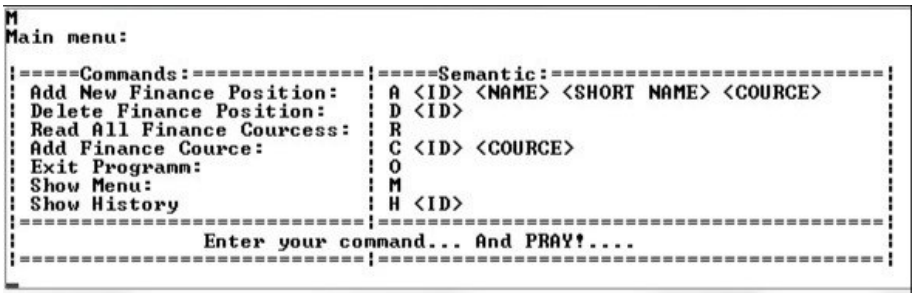

Рис. 9: Вывод меню

Вывести историю изменения валюты командой H <ID>

Рис. 10: Вывод истории

5.3 Обработка ошибок и предупреждений клиентского приложения

5.3.1 Неполный список аргументов

Исходные данные: A 1

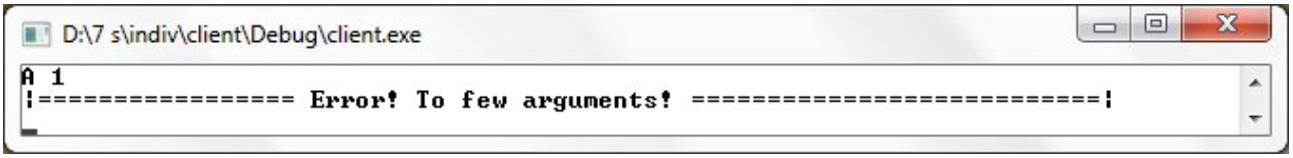

Рис. 11: Обработка неполного количества аргументов

5.3.2 Избыток цифр в ID

Исходные данные: A 231241214 RUB RU 12

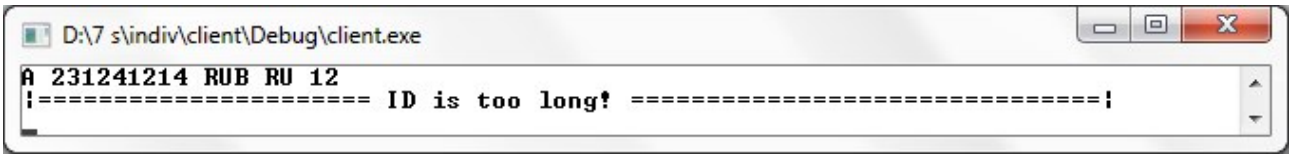

Рис. 12: Обработка избытка цифр в ID

5.3.3 Некорректный ID

Исходные данные: A 34 RUB RU 12

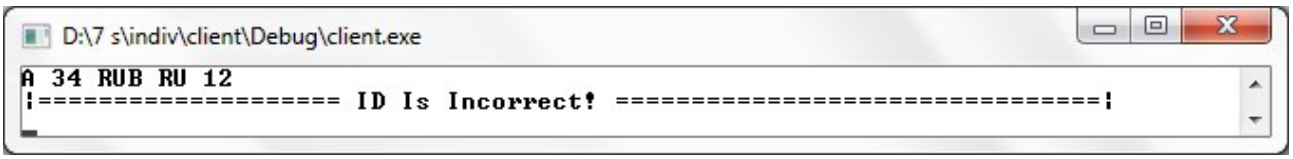

Рис. 13: Обработка некорректного ID

5.3.4 Слишком длинное название <NAME>

Исходные данные: A 2 ruuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuub ru 12

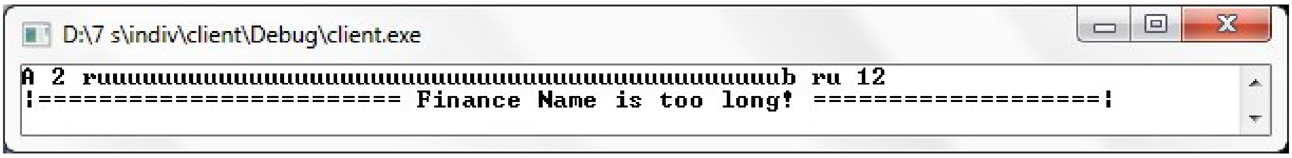

Рис. 14: Обработка <Name>

5.3.5 Слишком длинное <SHORT NAME>

Исходные данные: A 2 RUB ruuuuuuuuuuuuuuuuuuu 12

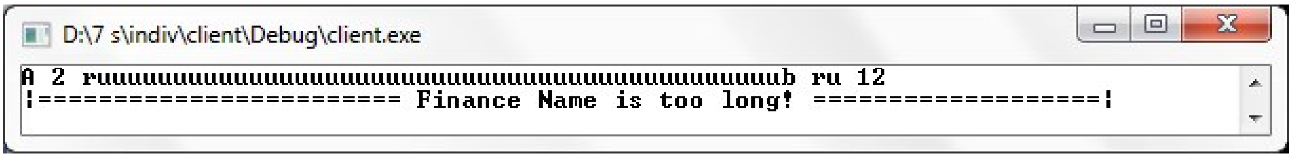

Рис. 15: Обработка <Short name>

5.3.6 Курс с таким ID не найден

Исходные данные: D 4

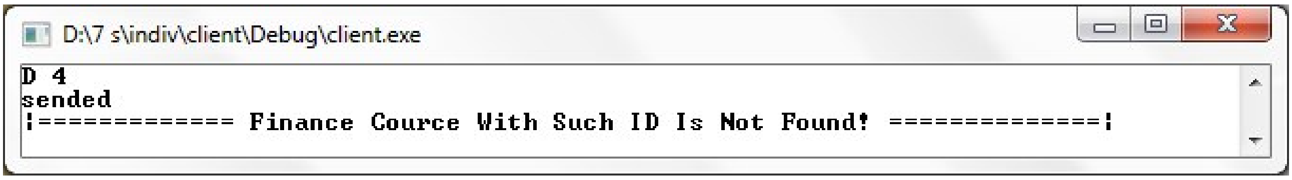

Рис. 16: Обработка <Short name>

6 Вывод

В ходе работы были изучены принципы программирования сокетов с использованием протокола TCP.
Во время выполнения индивидуального задания была реализована клиент-серверная программа выставления/просмотра курса валют, с написанием собственного протокола на основе TCP. Приложение позволяет клиенту добавлять, удалять, просматривать курс валют; разрывать соединение с сервером. Протокол был реализован на языке C++ для операционной системы Windows.
Были получены навыки организации многопоточного сервера, изучены принципы синхронизации доступа к глобальным переменным. В разработанном в ходе работы сервере для каждого клиента создается отдельный поток. Такой подход оправдан, т.к. клиенты могут исполнять долгие операции и операции различной трудоемкости. В этом случае использование отдельного потока для каждого клиента обеспечивает минимизацию взаимного влияния клиентов друг на друга.

Приложение 1

```
DWORD thID;

WSADATA wsaData;
 int iResult;
 int length=0;

 struct addrinfo *result = NULL;
 struct addrinfo hints;
```

```
9
10 printf("Exchange Rates Started...\n");
11
12 iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
13 printf("WSAS starting... \n");
14 if (iResult != 0)
15 {
16     printf("WSAStartup failed with error: %d\n", iResult);
17     return 1;
18 }
19
20 ZeroMemory(&hints, sizeof(hints));
21
22 sockaddr_in Server;
23 Server.sin_family = AF_INET;
24 Server.sin_addr.s_addr = inet_addr("127.0.0.1"); //local_addr.sin_addr.s_addr=0;
25 Server.sin_port = htons(DEFAULT_PORT);
26
27 SOCKET MySocket;
28
29 MySocket = socket(AF_UNSPEC, SOCK_STREAM, IPPROTO_TCP);
30 printf("Listening starting... \n");
31 if (MySocket == INVALID_SOCKET)
32 {
33     printf("socket failed with error: %ld\n", WSAGetLastError());
34     freeaddrinfo(result);
35     WSACleanup();
36     return 1;
37 }
38 printf("Success! \n");
39
40 iResult = bind( MySocket, (SOCKADDR *) & Server, sizeof (Server));
41 printf("Bind starting... \n");
42 if (iResult == SOCKET_ERROR) {
43     printf("bind failed with error: %d\n", WSAGetLastError());
44     freeaddrinfo(result);
45     closesocket(MySocket);
46     WSACleanup();
47     return 1;
48 }
```

Приложение 2

```
1  DWORD WINAPI ServToClient(LPVOID client_socket)
2  {
3      SOCKET my_sock;
4      my_sock=((SOCKET *) client_socket)[0];
5      int MyID = ID;
6      char ID[8];
7      char SockBuf[8];
8      _itoa_s(my_sock, SockBuf, 10);
9      _itoa_s(MyID, ID, 10);
10     strcpy_s(UserInfo[MyID], "ID: ");
11     strcat_s(UserInfo[MyID], ID);
12     strcat_s(UserInfo[MyID], " Addr: ");
13     strcat_s(UserInfo[MyID], inet_ntoa(client_addr.sin_addr));
14     strcat_s(UserInfo[MyID], " S: ");
15     strcat_s(UserInfo[MyID], SockBuf);
16     strcat_s(UserInfo[MyID], "\n");
17     printf("%s", UserInfo[MyID]);
18     fflush(stdout);
19
20     char recvbuf[DEFAULT_BUFLEN];
21     int iResult;
22     char user[30];
23
24     memset(recvbuf, 0, DEFAULT_BUFLEN);
25     int end = 0;
26     memset(user, 0, 30);
27
28     bool r = false;
29
30     CCount = CCount -1; // Umenshaem schetchik klientov
31     printf("-disconnect\n"); PRINTNUSERS
32
```

```
    // cleanup

    // WSACleanup();

    printf("Complited! \n");

    return 0;
}
```

Приложение 3

```
  iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
printf("WSAS starting... \n");
if (iResult != 0) {
    printf("WSAStartup failed with error: %d\n", iResult);
    return 1;
}

SOCKET my_sock;
my_sock = socket(AF_UNSPEC, SOCK_STREAM, IPPROTO_TCP);
printf("Opening socket... \n");
if (my_sock == INVALID_SOCKET)
{
    printf("socket failed with error: %ld\n", WSAGetLastError());
    WSACleanup();
    return 1;
}
printf("Success! \n");

if (inet_addr(SERVERADDR)!=INADDR_NONE)
client.sin_addr.s_addr=inet_addr(SERVERADDR);
else

if (hst=gethostbyname(SERVERADDR))
{
    ((unsigned long *)&client.sin_addr)[0]=((unsigned long **)hst->h_addr_list)[0][0];
}
else
{
    printf("Invalid address %s\n",SERVERADDR);
    closesocket(my_sock);
    WSACleanup();
    return -1;
}

iResult = connect( my_sock, (SOCKADDR *) & client, sizeof(client));
printf("Connecting... \n");
if (iResult == SOCKET_ERROR)
{
    closesocket(my_sock);
    my_sock = INVALID_SOCKET;
}
if (my_sock == INVALID_SOCKET)
{
    printf("Unable to connect to server!\n");
    WSACleanup();
    return 1;
}
printf("Connected to %s \n",SERVERADDR);
```