

Санкт-Петербургский Государственный Политехнический Университет
Кафедра Компьютерных Систем и Программных Технологий

ОТЧЕТ
по лабораторной работе
«Организация сетевого взаимодействия. Протоколы TCP и UDP»
Дисциплина: Сети ЭВМ и телекоммуникации

Работу выполнил студент гр. 43501/3:

Муравьев Ф.Э.

Преподаватель:

Вылегжанина К.Д.

Санкт-Петербург
2016

Глава 1.

1.1 Функциональные требования

Сетевая игра «Рулетка»

Задание: разработать клиент-серверную игру, имитирующую сетевую игру «Рулетка», состоящую из игрового сервера и клиентов.

Основные возможности. 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) Регистрация подключившегося клиента в качестве крупье (один) или в качестве игрока (несколько)
- 5) Выдача клиенту (крупье и игроку) списка ставок текущей игры (ставок других игроков)
- 6) Получение от клиента ставки: суммы ставки и типа ставки (чет, нечет, номер)
- 7) Получение от крупье команды на начало розыгрыша
- 8) Уведомление всех клиентов о результате розыгрыша
- 9) Обработка запроса на отключение клиента
- 10) Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

- 1) Установление соединения с сервером
- 2) Посылка регистрационных данных клиента (как крупье или как игрока)
- 3) Получение и вывод списка ставок
- 4) Для игрока: посылка своей ставки (сумма и ставки и тип ставки)
- 5) Для крупье: посылка команды начала розыгрыша
- 6) Получение и вывод результатов розыгрыша
- 7) Разрыв соединения
- 8) Обработка ситуации отключения клиента сервером

Настройки приложений. Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени удалённого игрового сервера и номера порта, используемого сервером.

1.2 Нефункциональные требования

Задание выполняется для двух протоколов TCP и UDP, а также с различным выбором операционных систем для программы-клиента и программы-сервера комбинации вариантов протоколов обмена и распределения клиент-серверных функций приведены в табл. 1.

Таблица 1. Вариант задания

Сетевой протокол	Серверная операционная система	Клиентская операционная система
TCP	Linux	Windows
UDP	Windows	Linux

1.3 Теоретические сведения

Перед организацией сетевого взаимодействия необходимо распределить функции между общающимися процессами: кто из них будет выполнять функции клиента, а кто сервера. При этом «роли» могут изменяться динамически. Соответственно, в один момент времени процесс является клиентом (шлет запрос), в другой – сервером (обслуживает запрос). При этом сервер может обслуживать одного клиента (рис. 1), либо многих (рис. 2).

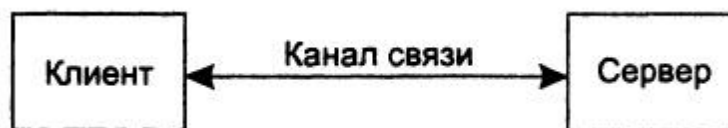


Рисунок 1. Сервер обслуживает одного клиента

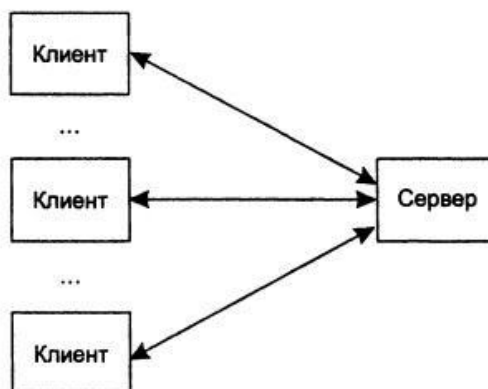


Рисунок 2. Сервер обслуживает много клиентов

После распределения функций, необходимо определить протокол обмена (формат передачи сообщений, размер, наличие повторной передачи, подтверждений и т.д.). Параметры протокола выбираются в зависимости от решаемой задачи. Например, в протоколе UDP нет гарантии доставки, но при этом данный протокол используется для передачи потокового видео, т.к. потеря пакетов не приводит к сильному искажению данных.

В данной работе сетевое взаимодействие производится на основе сокетов.

Сокет – это два значения, идентифицирующие конечную точку. В случае TCP/IP конечная точка идентифицируется IP адресом и портом.

IP адрес – 32-битное число, для удобства записываемое по байтно: «192.168.10.5», где точка – разделитель байтов.

Порт – идентификатор процесса, по которому ОС определяет, какому сокету пришли данные.

В ОС существует API сокетов, позволяющих достаточно просто организовать взаимодействие узлов в сети.

В общем виде, алгоритм работы сервера следующий (см. рис 3):

1. Создание сокета с помощью вызова `connect`. При этом локальные адрес и порт сокету назначаются из числа свободных;
2. Привязка сокета к удаленному адресу с помощью вызова `bind` (для сервера устанавливается адрес `INADDR_ANY`, который позволяет получать данные с любых адресов);
3. Перевод сокета в состояние прослушивания соединений с помощью вызова `listen`. При этом данные передавать через сокет нельзя, единственная его задача – получение запросов на соединение;

4. Прием соединений клиентов с помощью вызова `accept`. Данный вызов блокирует выполнение потока, пока не придет соединение от клиента, в результате которого создается новый сокет, связанный с адресом клиента;
5. Далее происходит прием/передача данных.

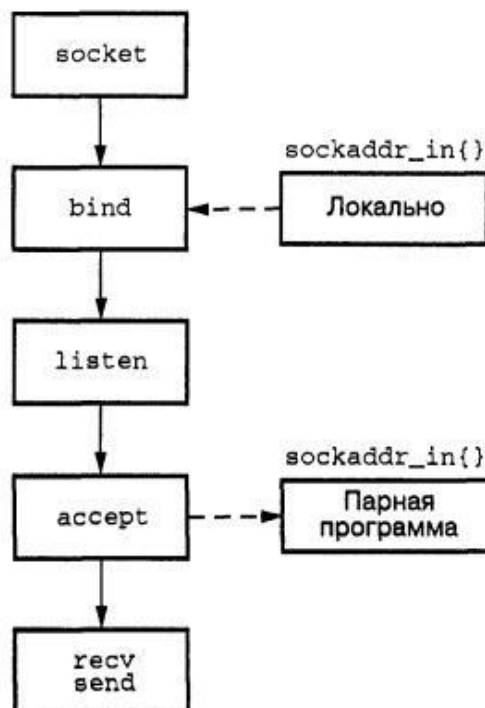


Рисунок 3. Основные вызовы API сокетов для сервера

Соответствующие вызовы для клиента, представлены на рис. 4.

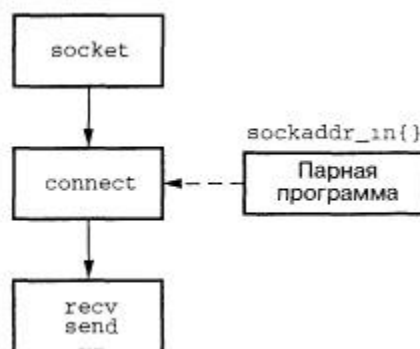


Рисунок 4. Основные вызовы API сокетов для клиента

1. Клиент создает сокет;
2. Сокет привязывается к удаленному хосту;
3. Производится обмен.

Отметим, что порядок системных вызовов выше можно использовать как для TCP, так и для UDP.

1. TCP

В данной работе первым рассматривается протокол транспортного уровня TCP, который является потоковым надежным протоколом с установлением соединения.

Протокол TCP «гарантирует доставку данных» (производит повторную передачу при необходимости, устраняет дублирование при получении двух копий пакета), производит упорядочивание данных, управление потоком. Так же, данный протокол автоматически

рассчитывает таймаут, по истечении которого будет происходить повторная передача (данный таймаут будет зависеть от размеров сети, её загруженности).

Стоит отметить, что протокол является потоковым, а значит, в нем нет понятия о сообщении (здесь уместна аналогия с последовательным портом, когда мы не можем заранее определить границу сообщений). Поэтому необходимо в приложении обеспечить механизм выявления сообщений из потока байт. Решить данную проблему позволяют два способа: передача сообщений по разделителям и передача с указанием длины сообщения.

2. UDP

Простой и ненадежный протокол передачи дейтаграмм. Поэтому приложение на прикладном уровне должно поддерживать надежность доставки, удаление неверных дейтаграмм и т.д.

Глава 2.

Реализация для работы по протоколу TCP

2.1 Прикладной протокол

Для данного индивидуального задания был разработан следующий протокол:

У клиентов есть возможность посылать серверу следующие сообщения:

Тип	Формат сообщения	Пример сообщения	Описание
При подключении клиента	'flagName.'	1Fedor. 0Fedor.	Flag – обозначает кто подключается: 1-крупье, 0-игрок.
Отключение клиента	'1'	1	Клиент посылает символ 1 если желает отключиться
Посмотреть ставки	'2'	2	Клиент посылает символ 2 для оповещения сервера о том, что он желает посмотреть весь список текущих ставок
Сделать ставку	'3типсумма.номер'	3e12000. 3o500. 3n12000.12.	Клиент отправляет следующее сообщение серверу, когда производится ставка. Тип может быть: e-even четное o-odd нечетное n-number номер Номер указывается лишь в том случае, когда тип ставки n
Начало розыгрыша	'4'	4	Клиент-крупье отправляет данное сообщение серверу, когда желает уведомить его о том пора начинать розыгрыш
Окончание розыгрыша	'5'	5	Клиент-крупье отправляет данное сообщение серверу, когда желает уведомить

			его о том пора заканчивать розыгрыш и по выпавшему номеру определять победителя
--	--	--	---

От сервера к клиентам приходят текстовые сообщения, например:

Hoax start

Hoax Finish

Roll number = 6

Sorry, you lose

You win! Your prize is ...\$

Поэтому в клиентском приложении реализована специальная функция, которая осуществляет чтение всех сообщений, приходящих на данный сокет, и печатает их на экран.

2.2 Архитектура приложения

2.2.1 Сервер

Для начала опишем хранимые на сервере глобальные структуры и переменные:

1) struct Connect_Info

```
struct Connect_Info{
    int id_con;
    char* Name;
    void* i_sock;
    int flag_dis;
};
struct Connect_Info* All_Connects;
```

Данная структура используется для хранения информации о подключенном клиенте. Соответственно на сервере динамически создается массив из этих структур, таким образом мы имеем информацию о всех подключенных на данный момент клиентах. При подключении новых или отключении старых клиентов этот массив обновляется с помощью функций `realloc` и `malloc` по хорошо проверенным алгоритмам.

Подробнее о полях структуры:

`int id_con` – ID уникальный индекс клиента

`char* Name` – имя клиента

`void* i_sock` – сокет к которому подключен данный клиент

`int flag_dis` – флаг «дисконнекта», о них будет рассказано позже

2) struct Rates_Info

```
struct Rates_Info{
    int id_pl;
    char* Name;
    void* i_sock;
    char* type;
    int summa;
    int number;
};
```

Данная структура используется для хранения информации о сделанных во время розыгрыша ставках. Соответственно на сервере динамически создается массив из этих структур, таким образом мы имеем информацию о всех ставках, сделанных в данном розыгрыше.

Подробнее о полях структуры:

`int id_pl` – ID уникальный индекс клиента

char* Name – имя клиента
void* i_sock – сокет к которому подключен данный клиент
char* type – тип ставки(см протокол)
int summa – сумма ставки
int number – номер, на который осуществлена ставка, если ее тип n

Теперь о глобальных переменных:

int Have_Croupier – флаг наличия крупье, устанавливается в 1 при подключении крупье, соответственно в 0 при его отключении. Если подключается крупье а флаг=1(т.е. крупье уже имеется), то мы отклоняем подключение клиента как крупье, и подключаем его как игрока, уведомляя его при этом.

int ID – генератор ID клиентов, его значение присваивается новому клиенту, при этом ID инкрементируется.

int Flag_start_hoax – флаг начала розыгрыша(необходим для взаимодействия потока крупье и потоков клиентов)

int Flag_finish_hoax – флаг конца розыгрыша(необходим для взаимодействия потока крупье и потоков клиентов)

int Koll_Connect – количество текущих подключений к серверу

int Koll_Rate – количество сделанных ставок в данном розыгрыше

Теперь расскажем подробнее о работе сервера:

В самом начале старта сервера отвечается поток для функции void *Server(). Она используется для реализации принудительного отключения клиента сервером. Далее в методе main создается сокет и, как было описано выше, переводится в состояние listen. Далее осуществляется прием соединений клиентов с помощью вызова accept и ответвления потока для функции void *Log_and_Work(void* socket). Таким образом для каждого клиента создается свой поток функции void *Log_and_Work(void* socket).

Функция Log_and_Work осуществляет регистрацию вновь подключенного клиента, заполняет структуру struct Connect_Info* All_Connects, увеличивает счетчик текущих подключений, определяет является клиент игроком или крупье, проверяя при этом флаг Have_Croupier. В зависимости от полученных результатов она передает управление в функцию void Croupier(char *Name, void* newsockfd, int my_id) или void Player(char *Name, void* newsockfd, int my_id).

Функция void Croupier(char *Name, void* newsockfd, int my_id) организует работу с клиентом-крупье, предоставляя ему возможности старта, завершения розыгрыша, просмотр текущих ставок. Предусмотрены обработки ошибок, например, если клиент крупье попытается завершить розыгрыш, при этом его не начав.

Функция void Player(char *Name, void* newsockfd, int my_id) организует работу с клиентом-игроком, предоставляя ему возможности просмотра текущих ставок в данном розыгрыше, осуществление собственной ставки.

Функция void disconnect(void* newsockfd, int my_id, char *Name) осуществляет отключение клиентов. В ней реализовано:

- уведомление клиента об его отключении
- уведомления сервера об отключении клиента
- коррекция структуры Connect_Info
- грамотное закрытие сокета, по которому осуществлялась связь с клиентом

Функция void Watch_Rate(void* newsockfd) осуществляет отправку клиенту данных о текущих ставках в данном розыгрыше.

Функция void Do_Rate(void* newsockfd, int my_id, char *Name) осуществляет заполнение структуры Rates_Info в соответствии с полученными от клиента данными.

Функция void disconnect_serv(int my_id) вызывается функцией Server и осуществляет установку флага flag_dis в структуре struct Connect_Info для клиента, которого

необходимо отключить. При этом каждый ”клиентский” поток проверяет в цикле свой флаг, если он станет равным 1, то поток самостоятельно вызывает функцию disconnect, что приводит к его отключению.

2.2.2 Клиент

Клиент состоит из главной функции main и функции DWORD WINAPI threadHandler(LPVOID param), для которой создается отдельный поток после организации подключения к серверу. Данная функция осуществляет прием и печать сообщений, получаемых от сервера.

2.3 Тестирование

2.3.1 Описание тестового стенда и методики тестирования

Для тестирования применялась утилита netem. Тестирование проводилось на моменте регистрации клиента и на моменте отправления клиентом сделанной ставки серверу, т.к. это самые интересные случаи.

2.3.2 Тестовый план и результаты тестирования

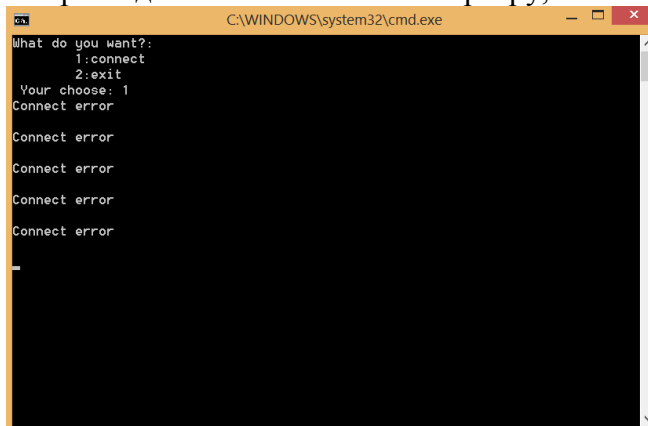
1) Потеря пакета при регистрации:

```
tc qdisc add dev eth0 root netem loss 50% 100%
```

Результаты:

Когда пакеты не терялись, клиент спокойно подключался и отключался от сервера.

При потере пакета при подключении клиента к серверу, клиент зависает:



```
What do you want?:
1:connect
2:exit
Your choose: 1
Connect error
Connect error
Connect error
Connect error
Connect error
Connect error
-
```

2) Дубликация пакетов:

```
tc qdisc change dev eth0 root netem duplicate 100%
```

При попытке клиентом сделать ставку происходит следующее:

Клиент:


```
C:\WINDOWS\system32\cmd.exe
What do you want?:
1:connect
2:exit
Your choose: 1
Do you want to be a croupier? y/n
n
Enter your name: Player
Server: Waiting the start of hoax
You can:
1: Disconnect
2: Watch rates
3: Make rate
Server: Hoax starts!
3
Choose the type of rate:
e - even
o - odd
n - number
e
Enter a cash:
12000
You can:
1: Disconnect
2: Watch rates
3: Make rate
```

Сервер:

```
qtcreator_process_stub
Connect croupier: Name: Fedor, ID: 1.
Connect player: Name: Player1, ID: 2.
Hoax starts!
New Rate from Player1: Type e Summ 12000
```

Клиент и сервер работают корректно.

3) Помехи в пакетах

tc qdisc change dev eth0 root netem corrupt 70%

Полученные результаты:

Клиенты:

```
C:\WINDOWS\system32\cmd.exe
What do you want?:
1:connect
2:exit
Your choose: 1
Connect error
Do you want to be a croupier? y/n
n
Enter your name: Player
Server: Waiting the start of hoax
Hoax starts!
You can:
1: Disconnect
2: Watch rates
3: Make rate
3
Choose the type of rate:
e - even
o - odd
n - number
n
Enter a cash:
12000000
Enter a number:
13
You can:
1: Disconnect
2: Watch rates
3: Make rate

C:\WINDOWS\system32\cmd.exe
What do you want?:
1:connect
2:exit
Your choose: 1
Do you want to be a croupier? y/n
y
Enter your name: Fedor
Server: You are a croupier.
You can:
1: Disconnect
2: Watch rates
3: Start hoaxes
4: Finish hoaxes
3
You can:
1: Disconnect
2: Watch rates
3: Start hoaxes
4: Finish hoaxes
```

Сервер:

```
qtcreator_process_stub
Connect croupier: Name: Fedor, ID: 1.
Hoax starts!
Connect player: Name: Player, ID: 2.
New Rate from Player: Type n Summ 12000000 Number 13
```

Клиент и сервер работают значительно медленнее, однако их работа корректна.

4) Изменение порядка получения пакетов

tc qdisc change dev eth0 root netem delay 10ms reorder 25% 50%

Полученные результаты:

Клиент и сервер работают корректно.

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

Прикладной протокол остается прежним, как и в реализации для работы по протоколу UDP.

3.2 Архитектура приложения

Архитектура приложения остается прежней как и для реализации для работы по протоколу TCP за исключением некоторых изменений, опишем их.

3.2.1 Сервер

Так как теперь серверная операционная система Windows, то используется библиотека winsock, а для создания потоков библиотека process.h. Также так как теперь используется протокол UDP используется структура UDP-сервера:

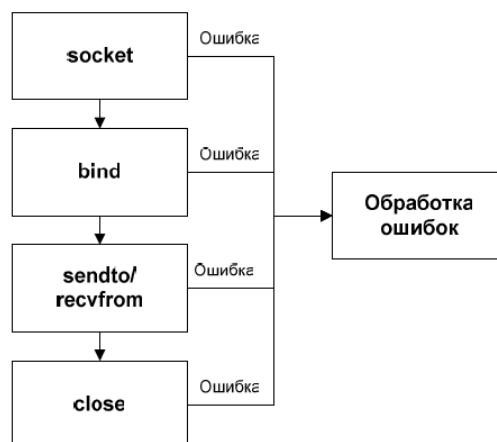


Рис 3.2.1.1 Типичная структура UDP-сервера

Также используется другой механизм подключения клиентов. Изначально на сервере имеется порт, на который приходят запросы на подключение от клиентов. Далее сервер создает новый порт, по которому он будет проводить дальнейшую работу именно с этим клиентом.

3.2.2 Клиент

Архитектура клиента остается прежней, за исключением того, что используется структура UDP-клиента:

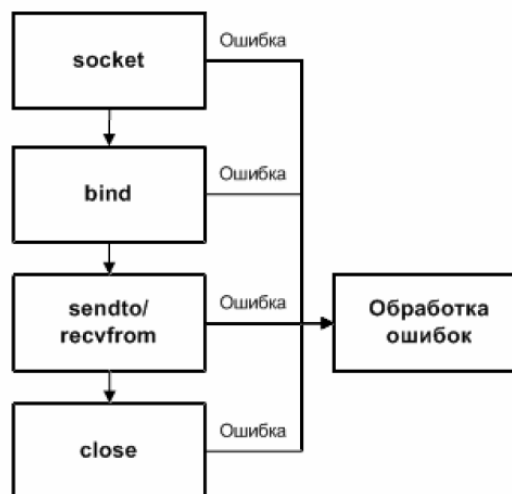


Рис 3.2.1.1 Типичная структура UDP-клиента

Также меняется механизм подключения клиента к серверу. Изначально клиент подключается к заданному порту сервера(на котором производится обработка новых подключений). Далее клиент получает от сервера номер порта, по которому производится дальнейшая работа клиента с сервером.

3.3 Тестирование

3.3.1 Описание тестового стенда и методики тестирования

Для тестирования применялась утилита netem. Тестирование проводилось на моменте регистрации клиента и на моменте отправления клиентом сделанной ставки серверу, т.к. это самые интересные случаи.

3.3.2 Тестовый план и результаты тестирования

1) Потеря пакета при регистрации:

tc qdisc add dev eth0 root netem loss 50% 100%

Результаты:

Когда пакеты не терялись, клиент спокойно подключался и отключался от сервера.

При потере пакета при подключении клиента к серверу, клиент зависает:

Клиент:

```
qtcreator_process_stub x
What do you want?:
  1:connect
  2:exit
Your choose: 1
Do you want to be a croupier? y/n
y
Enter your name: Fedor
█
```

2) Дубликация пакетов:

tc qdisc change dev eth0 root netem duplicate 100%

Теперь попробуем начать розыгрыш за крупье, и сделать ставку одним из игроков:

Клиенты:

qtcreator_process_stub x	qtcreator_process_stub x
What do you want?: 1:connect 2:exit Your choose: 1 Do you want to be a croupier? y/n y Enter your name: Fedor 8001 Server: You are a croupier. You can: 1: Disconnect 2: Watch rates 3: Start hoaxes 4: Finish hoaxes 3 Server: The hoax already starts You can: 1: Disconnect 2: Watch rates 3: Start hoaxes 4: Finish hoaxes █	Do you want to be a croupier? y/n n Enter your name: Player1 8002 Server: Waiting the start of hoax You can: 1: Disconnect 2: Watch rates 3: Make rate Server: Hoax starts! 3 Choose the type of rate: e - even o - odd n - number e Enter a cash: 10000 You can: 1: Disconnect 2: Watch rates 3: Make rate █

Сервер:

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

Received packet from 192.168.56.102:35941
Data: connect
Connect croupier: Name: Fedor. ID: 1.
Received packet from 192.168.56.102:55367
Data: connect
Connect player: Name: Player1. ID: 2.
Hoax starts!
New Rate from Player1: Type e Summ 1035228928
```

Таким образом на сервер приходит два пакета от крупье с объявлением начала розыгрыша, но приложение отлично справляется с обработкой данной ситуации. На приведенном примере видно что сервер ответил на второй пакет сообщением The hoax already starts. Однако при обработке ставки от игрока с дубликацией пакетов, серверное приложение не справилось: сумма ставки составляет 1035228928, что является ошибкой, так как игрок ставил 10000.

3) Помехи в пакетах

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

Полученные результаты:

Клиенты:

```
qtcreator_process_stub x qtcreator_process_stub x qtcreator_process_stub x

You cant:
1: Disconnect
2: Watch rates
3: Start hoaxes
4: Finish hoaxes

You cant:
1: Disconnect
2: Watch rates
3: Start hoaxes
4: Finish hoaxes

You cant:
1: Disconnect
2: Watch rates
3: Start hoaxes
4: Finish hoaxes

You cant:
1: Disconnect
2: Watch rates
3: Start hoaxes
4: Finish hoaxes

Choose the type of rate:
e - even
o - odd
n - number

Enter a cash:
12000
You cant:
1: Disconnect
2: Watch rates
3: Make rate

Choose the type of rate:
e - even
o - odd
n - number

Enter a cash:
12000
You cant:
1: Disconnect
2: Watch rates
3: Make rate

What do you want?:
1:connect
2:exit
Your choose: 1
Do you want to be a croupier? y/n
n
Enter your name: Player21221313121
```

Сервер:

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

Received packet from 192.168.56.102:35941
Data: connect
Connect croupier: Name: Fedor. ID: 1.
Received packet from 192.168.56.102:55367
Data: connect
Connect player: Name: Player1. ID: 2.
Hoax starts!
New Rate from Player1: Type e Summ 1035228928
Hoax finish! RollNumber=34
Hoax starts!
Disconnected: Name: Player1 ID:2
```

Результаты:

3-й клиент не может подключиться к серверу. Начать розыгрыш за крупье получилось только с третьего раза. При попытке сделать игроком ставку, сервер отключил его. Для выбранного протокола очень важно обеспечить передачу пакетов без помех.

Глава 4.

Выводы

В данной лабораторной работе были реализованы клиент-серверные программы на протоколах TCP и UDP.

Индивидуальным заданием было создание клиент-серверной игры - сетевой рулетки, состоящую из игрового сервера и клиентов.

На примере данной разработки были изучены основные приемы использования протокола транспортного уровня TCP – транспортного механизма, предоставляющего поток данных, с предварительной установкой соединения, за счёт этого дающего уверенность в достоверности получаемых данных, осуществляющего повторный запрос данных в случае потери данных и устраняющего дублирование при получении двух копий одного пакета. Данный механизм, в отличие от UDP, гарантирует, что приложение получит данные точно в такой же последовательности, в какой они были отправлены, и без потерь.

Приложения

Листинг 1. Серверное приложение сетевой рулетки TCP.

```
#include <stdio.h>
#include <stdlib.h>

#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <fcntl.h>

#include <string.h>
#include <pthread.h>

#include <time.h>

int Have_Croupier;
int ID;
int Flag_start_hoax;
int Flag_finish_hoax;
int Koll_Connect;
int Koll_Rate;

struct Connect_Info{
    int id_con;
    char* Name;
    void* i_sock;
    int flag_dis;
};
struct Connect_Info* All_Connects;

struct Rates_Info{
    int id_pl;
    char* Name;
    void* i_sock;
    char* type;
    int summa;
    int number;
};
struct Rates_Info* All_Rates;

void *Log_and_Work(void* socket); //обработчик клиентов
void *Server();

int main() {
    Have_Croupier=0;
    Flag_finish_hoax=0;
    ID=0;
    Koll_Connect=0;
    int sockfd, newsockfd, portno, clilen, flag_thr;
    struct sockaddr_in serv_addr, cli_addr;
    const int on = 1;
    /* First call to socket() function */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd < 0) {
        perror("ERROR opening socket");
        exit(1);
    }

    if ( setsockopt( sockfd, SOL_SOCKET, SO_REUSEADDR, &on,
        sizeof( on ) ) )
    {
        perror("ERROR call setsockopt");
        exit(1);
    }
}
```

```

}
/* Initialize socket structure */
bzero((char *) &serv_addr, sizeof(serv_addr));
//printf("Write port number: ");
//scanf("%d", &portno);
portno = 5001;

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);

/* Now bind the host address using bind() call.*/
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
    perror("ERROR on binding");
    exit(1);
}
/*Настройка атрибутов потока*/
pthread_attr_t threadAttr;
pthread_attr_init(&threadAttr);
pthread_attr_setdetachstate(&threadAttr, PTHREAD_CREATE_DETACHED);

/* Now start listening for the clients, here process will
 * go in sleep mode and will wait for the incoming connection
 */

pthread_t serv;
flag_thr = pthread_create(&serv, &threadAttr, Server, NULL);
if(flag_thr != 0)
{
    perror( "Creating thread false");
    exit(1);
}

listen(sockfd,5);
clilen = sizeof(cli_addr);
while(1)
{
    /* Accept actual connection from the client */
    newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);

    if (newsockfd < 0) {
        perror("ERROR on accept");
        exit(1);
    }
    pthread_t thread;
    flag_thr = pthread_create(&thread, &threadAttr, Log_and_Work, (void*)newsockfd);
    if(flag_thr != 0)
    {
        perror( "Creating thread false");
        exit(1);
    }
}
return 0;
}

void *Log_and_Work(void* newsockfd)
{
    char *Name;
    char buffer[255];
    int rc, flag, i, kol, my_id;
    ID=ID+1;
    my_id=ID;
    bzero(buffer,255);
    rc = read( newsockfd,buffer,255 ); //здесь крашиться
    if (rc < 0) {

```

```

        perror("ERROR reading from socket");
        exit(1);
    }
    flag=(int) buffer[0] - (int)'0';
    kol=0;
    while(buffer[kol]!='.')
    {
        kol=kol+1;
    }
    Name =(char*) malloc (sizeof(char) * kol);
    for(i=1; i<kol; i++)
    {
        Name[i-1]=buffer[i];
    }
    Name[kol-1]='\0'; //Вставляем нулевой символ окончания строки.
    if (Koll_Connect==0)
    {
        Koll_Connect=Koll_Connect+1;
        All_Connects=(struct Connect_Info*) malloc(sizeof(struct Connect_Info));
        All_Connects[0].id_con=my_id;
        All_Connects[0].i_sock=newsockfd;
        All_Connects[0].Name=Name;
        All_Connects[0].flag_dis=0;
    }
    else
    {
        Koll_Connect=Koll_Connect+1;
        All_Connects=(struct Connect_Info*) realloc(All_Connects, sizeof(struct
Connect_Info)*Koll_Connect);
        All_Connects[Koll_Connect-1].id_con=my_id;
        All_Connects[Koll_Connect-1].i_sock=newsockfd;
        All_Connects[Koll_Connect-1].Name=Name;
        All_Connects[Koll_Connect-1].flag_dis=0;
    }
    if(flag!=0)
    {
        if(Have_Croupier==1)
        {
            write(newsockfd, "We already have a croupier. You will be a player.",49);
            Player(All_Connects[Koll_Connect-1].Name, All_Connects[Koll_Connect-
1].i_sock, All_Connects[Koll_Connect-1].id_con);//Name, newsockfd, my_id
        }
        else
        {
            write(newsockfd, "You are a croupier.",20);
            Have_Croupier=1;
            Croupier(All_Connects[Koll_Connect-1].Name, All_Connects[Koll_Connect-
1].i_sock, All_Connects[Koll_Connect-1].id_con);//Name, newsockfd, my_id
        }
    }
    else
    {
        Player(All_Connects[Koll_Connect-1].Name, All_Connects[Koll_Connect-1].i_sock,
All_Connects[Koll_Connect-1].id_con);//Name, newsockfd, my_id
        free(Name);
        pthread_exit(NULL);
    }
}

void Croupier(char *Name, void* newsockfd, int my_id)
{
    char c[1],buf[60];
    int i,rc,Roll_Result;
    printf("Connect croupier: Name: %. ID: %d.\n",Name,my_id);
    fcntl(newsockfd, F_SETFL, O_NONBLOCK);
    while(1)
    {

```



```

for(i=0; i<Koll_Connect; i++)
{
    if (All_Connects[i].id_con==my_id)
    {
        if(All_Connects[i].flag_dis==1)
        {
            Have_Croupier=0;
            disconnect(newsockfd, my_id, Name);
        }
        break;
    }
}
rc=read(newsockfd, c,1);
if(rc>0)
{
    if(c[0]=='1')
    {
        Have_Croupier=0;
        disconnect(newsockfd, my_id, Name);
    }
    if(c[0]=='4')
    {
        if (Flag_start_hoax==0)
        {
            Koll_Rate=0;
            Flag_finish_hoax=0;
            Flag_start_hoax=1;
            printf("Hoax starts!\n");
        }
        else
            write(newsockfd, "The hoax already starts\n",25);
    }
    if(c[0]=='5')
    {
        if (Flag_start_hoax==0)
            write(newsockfd, "But first let the hoax starts\n",31);
        else
        {
            Flag_start_hoax=0;
            Flag_finish_hoax=1;
            srand( time(0) );
            Roll_Result=rand()%37;
            printf("Hoax finish! RollNumber=%d\n",Roll_Result);
            for(i=0; i<Koll_Rate; i++)
            {
                write(All_Rates[i].i_sock, "Hoax finish!\n\n",14);
                snprintf(buf, 60, "Winning number is %d\n\n", Roll_Result);
                write(All_Rates[i].i_sock, buf, strlen(buf));
            }
            if((All_Rates[i].type[0]=='n')&&(All_Rates[i].number==Roll_Result))
            {
                write(All_Rates[i].i_sock, "You win!\n",9);
                snprintf(buf, 60, "Your prize is %d$\n\n",
All_Rates[i].summa*35);
                write(All_Rates[i].i_sock, buf, strlen(buf));
            }
            else
            {
                if((All_Rates[i].type[0]=='e')&&(Roll_Result%2==0))
                {
                    write(All_Rates[i].i_sock, "You win!\n",9);
                    snprintf(buf, 60, "Your prize is %d$\n\n",
All_Rates[i].summa);
                    write(All_Rates[i].i_sock, buf, strlen(buf));
                }
            }
        }
    }
}
else

```

```

        if((All_Rates[i].type[0]=='o')&&(Roll_Result%2==1))
        {
            write(All_Rates[i].i_sock, "You win!\n",9);
            snprintf(buf, 60, "Your prize is %d$\n\n",
                All_Rates[i].summa);
            write(All_Rates[i].i_sock, buf, strlen(buf));
        }
        else
            write(All_Rates[i].i_sock, "Sorry, you
lose.\n\n",18);
    }
    if(Koll_Rate!=0)
        free(All_Rates);
}

}

}

}

void Player(char *Name, void* newsockfd, int my_id)
{
    int i,rc, Flag_Rate;
    char c[1],buf[255];
    fcntl(newsockfd, F_SETFL, O_NONBLOCK);
    printf("Connect player: Name: %s. ID: %d.\n",Name,my_id);
    while(1)
    {
        write(newsockfd, "Waiting the start of hoax\n",26);
        while(Flag_start_hoax==0)
        {
            for(i=0; i<Koll_Connect; i++)
            {
                if (All_Connects[i].id_con==my_id)
                {
                    if(All_Connects[i].flag_dis==1)
                        disconnect(newsockfd, my_id, Name);
                    break;
                }
            }

            rc = read( newsockfd, c, 1);
            if(rc>0)
            {
                if (c[0]=='1')
                {
                    disconnect(newsockfd, my_id, Name);
                }
                if(c[0]=='2')
                {
                    write(newsockfd, "The hoax didn't start yet\n",27);
                }
                if(c[0]=='3')
                {
                    write(newsockfd, "The hoax didn't start yet\n",27);
                    read( newsockfd, buf, 255);
                    bzero(buf,255);
                }
            }
        }
        write(newsockfd, "Hoax starts!\n",13);
        Flag_Rate=0;
        while(Flag_finish_hoax==0)
        {
            for(i=0; i<Koll_Connect; i++)

```

```

        {
            if (All_Connects[i].id_con==my_id)
            {
                if(All_Connects[i].flag_dis==1)
                    disconnect(newsockfd, my_id, Name);
                break;
            }
        }
        rc = read( newsockfd, c, 1);
        if(rc>0)
        {
            if (c[0]=='1')
            {
                disconnect(newsockfd, my_id, Name);
            }
            if(c[0]=='2')
            {
                Watch_Rate(newsockfd);
            }
            if(c[0]=='3')
            {
                if(Flag_Rate==0)
                {
                    Do_Rate(newsockfd, my_id, Name);
                    Flag_Rate=1;
                }
                else
                {
                    write(newsockfd, "You already rate! Wait results.\n",33);
                    read( newsockfd, buf, 255);
                    bzero(buf,255);
                }
            }
        }
    }
}

void disconnect(void* newsockfd, int my_id, char *Name)
{
    int flag,i,KolDis;

    for(i=0; i<Koll_Connect; i++)
    {
        if (All_Connects[i].id_con==my_id)
        {
            KolDis=i;
            break;
        }
    }
    if(Koll_Connect!=1)
    {
        for(i=KolDis; i<Koll_Connect-1; i++)
        {
            All_Connects[i].id_con=All_Connects[i+1].id_con;
            All_Connects[i].i_sock=All_Connects[i+1].i_sock;
            All_Connects[i].Name=All_Connects[i+1].Name;
            All_Connects[i].flag_dis=All_Connects[i+1].flag_dis;
        }
        Koll_Connect=Koll_Connect-1;
        All_Connects=(struct Connect_Info*) realloc(All_Connects, sizeof(struct
Connect_Info)*Koll_Connect);
    }
    else
    {
        Koll_Connect=0;
    }
}

```

```

        free(All_Connects);
        All_Connects=NULL;
    }

    printf("Disconnected: Name: %s ID:%d\n",Name,my_id);
    write(newsockfd, "You disconnected\n", 17);
    flag=shutdown(newsockfd, SHUT_RDWR);
    if (flag < 0) {
        perror("ERROR shutdown socket");
        exit(1);
    }
    flag=close(newsockfd);
    if (flag < 0) {
        perror("ERROR close socket");
        exit(1);
    }
    free(Name);
    pthread_exit(NULL);
}

void Watch_Rate(void* newsockfd)
{
    char buf[500]; //,message[105];
    int i;
    //snprintf(message, 105, "RATES\nType:\n e-even\n o-odd\n n-number(after that
write the number)\nName | ID | Type | SUMM\n\n");
    //write(newsockfd, message, strlen(message));
    for(i=0; i<Koll_Rate; i++)
    {
        //write(newsockfd, All_Rates[i].Name,strlen(All_Rates[i].Name));
        //write(newsockfd, " ",1);
        //snprintf(buf, 60, "%d", All_Rates[i].id_pl);
        //write(newsockfd, buf, strlen(buf));
        //write(newsockfd, " ",1);
        //write(newsockfd, All_Rates[i].type,1);
        //write(newsockfd, " ",1);
        if(All_Rates[i].type[0]=='n')
        {
            snprintf(buf, 500, "%s %d %c %d %d$\n", All_Rates[i].Name,
All_Rates[i].id_pl, All_Rates[i].type[0], All_Rates[i].number, All_Rates[i].summa);
            //snprintf(buf, 60, "%d", All_Rates[i].number);
            //write(newsockfd, buf, strlen(buf));
            //write(newsockfd, " ",1);
        }
        else
            snprintf(buf, 500, "%s %d %c %d$\n", All_Rates[i].Name, All_Rates[i].id_pl,
All_Rates[i].type[0], All_Rates[i].summa);
            //snprintf(buf, 60, "%d$", All_Rates[i].summa);
            write(newsockfd, buf, strlen(buf));
            //write(newsockfd, "\n",1);
        }
    }
}

void Do_Rate(void* newsockfd, int my_id, char *Name)
{
    char type[1],c[1];
    int summa,n,number;
    read( newsockfd, type, 1);
    summa=0;
    read( newsockfd, c, 1);
    do{
        summa=summa*10;
        n=(int) c[0] - (int)'0';
        summa=summa+n;
        read( newsockfd, c, 1);
    }while(c[0]!='.');
    if(type[0]=='n')

```

```

{
    number=0;
    read( newsockfd, c, 1);
    do{
        number=number*10;
        n=(int) c[0] - (int)'0';
        number=number+n;
        read( newsockfd, c, 1);
    }while(c[0]!='. ');
}
if(type[0]=='n')
    printf("New Rate from %s: Type %s Summ %d Number %d\n",Name,type,summa,number);
else
    printf("New Rate from %s: Type %s Summ %d\n",Name,type,summa);
Koll_Rate=Koll_Rate+1;
if (Koll_Rate==1)
    All_Rates=(struct Rates_Info*) malloc(sizeof(struct Rates_Info));
else
    All_Rates=(struct Rates_Info*) realloc(All_Rates, sizeof(struct
Rates_Info)*Koll_Rate);
All_Rates[Koll_Rate-1].id_pl=my_id;
All_Rates[Koll_Rate-1].Name=Name;
All_Rates[Koll_Rate-1].i_sock=newsockfd;
All_Rates[Koll_Rate-1].type=type;
All_Rates[Koll_Rate-1].summa=summa;
All_Rates[Koll_Rate-1].number=number;
}
void *Server()
{
    int i,dis_id,n,flag;
    char buf[40];
    dis_id=0;
    while(1)
    {
        scanf("%s",buf);
        flag=0;
        if((buf[0]=='d')&&(buf[9]=='t')&&(buf[10]=='_'))
        {
            dis_id=0;
            for(i=11; buf[i]!='.'; i++)
            {
                dis_id=dis_id*10;
                n=(int) buf[i] - (int)'0';
                if((n<0)|| (n>9))
                {
                    flag=1;
                    break;
                }
                dis_id=dis_id+n;
            }
            if(flag==1)
            {
                printf("Error: write the comand correctly: disconnect_id.\n");
            }
            else
            {
                disconnect_serv(dis_id);
            }
        }
    }
}

void disconnect_serv(int my_id)
{

```

```

int i,KolDis;
KolDis=-1;
for(i=0; i<Koll_Connect; i++)
{
    if (All_Connects[i].id_con==my_id)
    {
        KolDis=i;
        break;
    }
}
if(KolDis==-1)
{
    printf("Error: Cant find client with this ID\n");
    return;
}
All_Connects[KolDis].flag_dis=1;
}

```

Листинг 2. Клиентское приложение сетевой рулетки TCP.

```

#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <time.h>
#include<stdio.h>
#include<winsock2.h>
#include<WS2tcpip.h>
#include<iostream>
#include<string>
using namespace std;
#pragma comment(lib,"ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")

DWORD WINAPI threadHandler(LPVOID);
bool have_connect = false;
int main(int argc, char *argv[])
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;

    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("Failed. Error Code : %d\n", WSAGetLastError());
        return 1;
    }
    while (1)
    {
        bool start=true;
        while (start)
        {
            char choose;
            cout << "What do you want?:\n    1:connect\n    2:exit\n Your choose:
";

            cin >> choose;
            switch (choose)
            {
                case '1': start = false; break;
                case '2': return 0; break;
                default: cout << "That's not a choice.\n";
            }
        }
        if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
        {
            printf("Could not create socket : %d\n", WSAGetLastError());
        }
        start = true;
        while (start)
    }
}

```

```

{
    /*cout << "Enter server IP address: ";
    string ip_address;
    cin >> ip_address;
    cout << "Enter number of port: ";
    int port;
    cin >> port;
    server.sin_addr.s_addr = inet_addr(ip_address.c_str());
    server.sin_family = AF_INET;
    server.sin_port = htons(port);*/

    server.sin_addr.s_addr = inet_addr("192.168.56.102");
    server.sin_family = AF_INET;
    server.sin_port = htons(5001);
    //Connect to remote server
    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        puts("Connect error\n");
    }
    else
        start = false;
}
bool flag_croupier = false;
bool flag = true;
string mes_to_serv;
char user_a;
int sleep_time=100;

while (flag)
{
    cout << "Do you want to be a croupier? y/n\n";
    string answer;
    cin >> answer;
    if ((answer != "y") && (answer != "n"))
        cout << "Error. Write Y or N";
    if (answer == "y")
    {
        mes_to_serv = "1";
        flag = false;
        flag_croupier = true;
    }
    else if (answer == "n")
    {
        mes_to_serv = "0";
        flag = false;
    }
}
cout << "Enter your name: ";
string name;
cin >> name;
mes_to_serv += name;
mes_to_serv += '.';
if (send(s, mes_to_serv.c_str(), mes_to_serv.length(), 0) < 0)
{
    puts("Send failed");
    return 1;
}
char server_reply[2000];
int recv_size;
if ((recv_size = recv(s, server_reply, 2000, 0)) == SOCKET_ERROR)
{
    puts("recv failed");
}
server_reply[recv_size] = '\0';
cout << "Server: " << server_reply << "\n";

```

```

std::string buffer(server_reply);
if (buffer.find("We already have a croupier.") != string::npos)
    flag_croupier = false;

have_connect = true;
HANDLE t;
t = CreateThread(NULL, 0, threadHandler, (LPVOID)s, 0, NULL);

if (flag_croupier == true)
{
    while (have_connect)
    {
        cout << "You can:\n";
        cout << "    1: Disconnect\n";
        cout << "    2: Watch rates\n";
        cout << "    3: Start hoaxes\n";
        cout << "    4: Finish hoaxes\n";
        cin >> user_a;
        if (have_connect == false) break;
        switch (user_a)
        {
            case '1':    mes_to_serv = "1";
                        if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                        {
                            puts("Send failed\n");
                        };
                        have_connect = false;
                        Sleep(sleep_time);
                        break;
            case '2':    mes_to_serv = "2";
                        if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                        {
                            puts("Send failed\n");
                        };
                        Sleep(sleep_time);
                        break;
            case '3':    mes_to_serv = "4";
                        if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                        {
                            puts("Send failed\n");
                        };
                        Sleep(sleep_time);
                        break;
            case '4':    mes_to_serv = "5";
                        if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                        {
                            puts("Send failed\n");
                        };
                        Sleep(sleep_time);
                        break;
            default: cout << "That's not a choice.\n";
        }
    }
}
else
{
    while (have_connect)
    {
        cout << "You can:\n";
        cout << "    1: Disconnect\n";
        cout << "    2: Watch rates\n";
    }
}

```



```

        cout << "      3: Make rate\n";
        cin >> user_a;
        if (have_connect == false) break;
        int true_rate = 1;
        char rate_type;
        switch (user_a)
        {
        case '1':    mes_to_serv = "1";
                    if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                    {
                        puts("Send failed\n");
                    };
                    have_connect = false;
                    Sleep(sleep_time);
                    break;
        case '2':    mes_to_serv = "2";
                    if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                    {
                        puts("Send failed\n");
                    };
                    Sleep(sleep_time);
                    break;
        case '3':    while (true_rate)
                    {
                        cout << "Choose the type of
rate:\n";
                        cout << "      e - even\n      o -
odd\n      n - number\n";

                        cin >> rate_type;
                        switch (rate_type)
                        {
                        case 'e': mes_to_serv = "3e";
                        case 'o': mes_to_serv = "3o";
                        case 'n': mes_to_serv = "3n";
                        default: cout << "That's not a
choice.\n";
                        }
                    }
                    cout << "Enter a cash:\n";
                    int cash;
                    cin >> cash;
                    mes_to_serv += to_string(cash) + ".";
                    if (rate_type == 'n')
                    {
                        cout << "Enter a number:\n";
                        int number;
                        cin >> number;
                        mes_to_serv += to_string(number) +
".";
                    }
                    if (send(s, mes_to_serv.c_str(),
mes_to_serv.length(), 0) < 0)
                    {
                        puts("Send failed\n");
                    };
                    Sleep(sleep_time);
                    break;
        default: cout << "That's not a choice.\n";
        }
    }
}

```

```

    }
}
DWORD WINAPI threadHandler(LPVOID param){
    char server_reply[2000];
    int recv_size;
    SOCKET s = (SOCKET)param;
    while (have_connect)
    {
        if ((recv_size = recv(s, server_reply, 2000, 0)) == SOCKET_ERROR)
        {
            puts("recv failed");
            return 0;
        }
        server_reply[recv_size] = '\0';
        string mes = server_reply;
        if (mes.find("You disconnected") != string::npos)
        {
            cout << "Server: " << mes;
            have_connect = false;
            return 0;
        }
        cout << "Server: " << server_reply;
    }
    return 0;
}

```

Листинг 3. Серверное приложение сетевой рулетки UDP.

```

#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#define snprintf(buf,len, format,...) _snprintf_s(buf, len,len, format, __VA_ARGS__)
#define bzero(p, size) (void)memset((p), 0, (size))
#include <stdio.h>
#include <stdlib.h>
#include<winsock2.h>
#include<WS2tcpip.h>

#include <string.h>
#include <string>
#include <windows.h>
#include <process.h>

#include <time.h>

#pragma comment(lib,"ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")
#include <iostream>;
using namespace std;
int Have_Croupier;
int ID;
int Flag_start_hoax;
int Flag_finish_hoax;
int Koll_Connect;
int Koll_Rate;
int portno;

struct Connect_Info{
    int id_con;
    char* Name;
    struct sockaddr_in client;
    int slen;
    SOCKET i_sock;
    int flag_dis;
};

```

```

struct Connect_Info* All_Connects;

struct Rates_Info{
    int id_pl;
    char* Name;
    SOCKET i_sock;
    char type;
    int summa;
    int number;
};
struct Rates_Info* All_Rates;

unsigned __stdcall Log_and_Work(void * arg); //обработчик клиентов
unsigned __stdcall Server(void * arg);
void Watch_Rate(SOCKET newsockfd, int my_id);
void Player(char *Name, SOCKET newsockfd, int my_id, struct sockaddr_in client, int
slen);
void disconnect(SOCKET newsockfd, int my_id, char *Name);
void disconnect_serv(int my_id);
void Do_Rate(SOCKET newsockfd, int my_id, char *Name);
void Croupier(char *Name, SOCKET newsockfd, int my_i, struct sockaddr_in client, int
slen);

int main() {
    Have_Croupier = 0;
    Flag_finish_hoax = 0;
    ID = 0;
    Koll_Connect = 0;
    SOCKET s;
    struct sockaddr_in server, si_other;
    int slen, recv_len;
    char buf[255];
    //printf("Write port number: ");
    //scanf("%d", &portno);
    portno = 8000;
    WSADATA wsa;

    slen = sizeof(si_other);

    //Initialise winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("Failed. Error Code : %d", WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    //Create a socket
    if ((s = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d", WSAGetLastError());
    }
    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    //server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(portno);
    //Bind
    if (bind(s, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d", WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    HANDLE serv;
    unsigned uThreadIDs;
    serv = (HANDLE)_beginthreadex(NULL, 0,&Server, NULL, 0, &uThreadIDs);
    while (1)

```

```

{
    //clear the buffer by filling null, it might have previously received data
    memset(buf, '\0', 255);
    //try to receive some data, this is a blocking call
    if ((recv_len = recvfrom(s, buf, 255, 0, (struct sockaddr *) &si_other,
&slen)) == SOCKET_ERROR)
    {
        printf("recvfrom() failed with error code : %d", WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    //print details of the client/peer and the data received
    printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr),
ntohs(si_other.sin_port));
    printf("Data: %s\n", buf);
    ID = ID + 1;
    string new_p = to_string(portno+ID);
    if (sendto(s, new_p.c_str(), new_p.length(), 0, (struct sockaddr*)
&si_other, slen) == SOCKET_ERROR)
    {
        printf("sendto() failed with error code : %d", WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    HANDLE thread;
    thread = (HANDLE)_beginthreadex(NULL, 0, &Log_and_Work, (void*)ID, 0,
&uThreadIDs);
    CloseHandle(thread);
}
CloseHandle(serv);
return 0;
}

unsigned __stdcall Log_and_Work(void * arg)
{
    char *Name;
    char buffer[255];
    int rc, flag, i, kol, my_id;
    my_id = (int)arg;

    SOCKET newsockfd;
    struct sockaddr_in new_serv, client;
    int slen = sizeof(client);
    if ((newsockfd = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d", WSAGetLastError());
    }
    //Prepare the sockaddr_in structure
    new_serv.sin_family = AF_INET;
    new_serv.sin_addr.s_addr = INADDR_ANY;
    //new_serv.sin_addr.s_addr = inet_addr("127.0.0.1");
    new_serv.sin_port = htons(portno + my_id);
    //Bind
    if (bind(newsockfd, (struct sockaddr *)&new_serv, sizeof(new_serv)) ==
SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d", WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    bzero(buffer, 255);
    if ((rc = recvfrom(newsockfd, buffer, 255, 0, (struct sockaddr *) &client, &slen))
== SOCKET_ERROR)
    {
        printf("recvfrom() failed with error code : %d", WSAGetLastError());
        exit(EXIT_FAILURE);
    }
}

```

```

flag = (int)buffer[0] - (int)'0';
kol = 0;
while (buffer[kol] != '.')
{
    kol = kol + 1;
}
Name = (char*)malloc(sizeof(char) * kol);
for (i = 1; i < kol; i++)
{
    Name[i - 1] = buffer[i];
}
Name[kol - 1] = '\0'; //Вставляем нулевой символ окончания строки.
if (Koll_Connect == 0)
{
    Koll_Connect = Koll_Connect + 1;
    All_Connects = (struct Connect_Info*) malloc(sizeof(struct Connect_Info));
    All_Connects[0].id_con = my_id;
    All_Connects[0].i_sock = newsockfd;
    All_Connects[0].Name = Name;
    All_Connects[0].flag_dis = 0;
    All_Connects[0].client = client;
    All_Connects[0].slen = slen;
}
else
{
    Koll_Connect = Koll_Connect + 1;
    All_Connects = (struct Connect_Info*) realloc(All_Connects, sizeof(struct
Connect_Info)*Koll_Connect);
    All_Connects[Koll_Connect - 1].id_con = my_id;
    All_Connects[Koll_Connect - 1].i_sock = newsockfd;
    All_Connects[Koll_Connect - 1].Name = Name;
    All_Connects[Koll_Connect - 1].flag_dis = 0;
    All_Connects[Koll_Connect - 1].client = client;
    All_Connects[Koll_Connect - 1].slen = slen;
}
if (flag != 0)
{
    if (Have_Croupier == 1)
    {
        sendto(newsockfd, "We already have a croupier. You will be a
player.", 49, 0, (struct sockaddr*) &client, slen);
        Player(All_Connects[Koll_Connect - 1].Name, All_Connects[Koll_Connect
- 1].i_sock, All_Connects[Koll_Connect - 1].id_con, All_Connects[Koll_Connect -
1].client, All_Connects[Koll_Connect - 1].slen); //Name, newsockfd, my_id
    }
    else
    {
        sendto(newsockfd, "You are a croupier.", 20, 0, (struct sockaddr*)
&client, slen);
        Have_Croupier = 1;
        Croupier(All_Connects[Koll_Connect - 1].Name,
All_Connects[Koll_Connect - 1].i_sock, All_Connects[Koll_Connect - 1].id_con,
All_Connects[Koll_Connect - 1].client, All_Connects[Koll_Connect - 1].slen); //Name,
newsockfd, my_id
    }
}
else
    Player(All_Connects[Koll_Connect - 1].Name, All_Connects[Koll_Connect -
1].i_sock, All_Connects[Koll_Connect - 1].id_con, All_Connects[Koll_Connect - 1].client,
All_Connects[Koll_Connect - 1].slen); //Name, newsockfd, my_id
    free(Name);
}

void Croupier(char *Name, SOCKET newsockfd, int my_id, struct sockaddr_in client, int
slen)

```

```

{
    char c[1], buf[60];
    int i, rc, Roll_Result;
    printf("Connect croupier: Name: %s. ID: %d.\n", Name, my_id);
    while (1)
    {
        for (i = 0; i < Koll_Connect; i++)
        {
            if (All_Connects[i].id_con == my_id)
            {
                if (All_Connects[i].flag_dis == 1)
                {
                    Have_Croupier = 0;
                    disconnect(newsockfd, my_id, Name);
                }
                break;
            }
        }
        rc = recvfrom(newsockfd, c, 1, 0, (struct sockaddr *) &client, &slen);
        if (rc > 0)
        {
            if (c[0] == '1')
            {
                Have_Croupier = 0;
                disconnect(newsockfd, my_id, Name);
            }
            if (c[0] == '4')
            {
                if (Flag_start_hoax == 0)
                {
                    Koll_Rate = 0;
                    Flag_finish_hoax = 0;
                    Flag_start_hoax = 1;
                    printf("Hoax starts!\n");
                }
                else
                {
                    sendto(newsockfd, "The hoax already starts\n", 25, 0,
(struct sockaddr *) &client, slen);
                }
            }
            if (c[0] == '5')
            {
                if (Flag_start_hoax == 0)
                {
                    sendto(newsockfd, "But first let the hoax starts\n",
31, 0, (struct sockaddr *) &client, slen);
                }
                else
                {
                    Flag_start_hoax = 0;
                    Flag_finish_hoax = 1;
                    srand(time(0));
                    Roll_Result = rand() % 37;
                    printf("Hoax finish! RollNumber=%d\n", Roll_Result);
                    for (i = 0; i < Koll_Rate; i++)
                    {
                        int id_c;
                        for (int j = 0; j < Koll_Connect; j++)
                        {
                            if (All_Connects[j].id_con ==
All_Rates[i].id_pl)
                            {
                                id_c = j;
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        sendto(All_Rates[i].i_sock, "Hoax finish!\n\n",
14, 0, (struct sockaddr *) &All_Connects[id_c].client, All_Connects[id_c].slen);
        snprintf(buf, 60, "Winning number is %d\n\n",
Roll_Result);
        sendto(All_Rates[i].i_sock, buf, strlen(buf), 0,
(struct sockaddr *) &All_Connects[id_c].client, All_Connects[id_c].slen);
        if ((All_Rates[i].type == 'n') &&
(All_Rates[i].number == Roll_Result))
        {
            sendto(All_Rates[i].i_sock, "You win!\n",
9, 0, (struct sockaddr *) &All_Connects[id_c].client, All_Connects[id_c].slen);
            snprintf(buf, 60, "Your prize is %d$\n\n",
All_Rates[i].summa * 35);
            sendto(All_Rates[i].i_sock, buf,
strlen(buf), 0, (struct sockaddr *) &All_Connects[id_c].client, All_Connects[id_c].slen);
        }
        else
            if ((All_Rates[i].type == 'e') &&
(Roll_Result % 2 == 0))
            {
                sendto(All_Rates[i].i_sock, "You
win!\n", 9, 0, (struct sockaddr *) &All_Connects[id_c].client, All_Connects[id_c].slen);
                snprintf(buf, 60, "Your prize is
%d$\n\n", All_Rates[i].summa);
                sendto(All_Rates[i].i_sock, buf,
strlen(buf), 0, (struct sockaddr *) &All_Connects[id_c].client, All_Connects[id_c].slen);
            }
            else
                if ((All_Rates[i].type == 'o') &&
(Roll_Result % 2 == 1))
                {
                    sendto(All_Rates[i].i_sock,
"You win!\n", 9, 0, (struct sockaddr *) &All_Connects[id_c].client,
All_Connects[id_c].slen);
                    snprintf(buf, 60, "Your
prize is %d$\n\n", All_Rates[i].summa);
                    sendto(All_Rates[i].i_sock,
buf, strlen(buf), 0, (struct sockaddr *) &All_Connects[id_c].client,
All_Connects[id_c].slen);
                }
                else
                    sendto(All_Rates[i].i_sock,
"Sorry, you lose.\n\n", 18, 0, (struct sockaddr *) &All_Connects[id_c].client,
All_Connects[id_c].slen);
            }
        }
        if (Koll_Rate != 0)
            free(All_Rates);
    }
}

}

}

}

void Player(char *Name, SOCKET newsockfd, int my_id, struct sockaddr_in client, int slen)
{
    int i, rc, Flag_Rate;
    char c[1], buf[255];
    unsigned long on = 1;
    ioctlsocket(newsockfd, FIONBIO, &on);
    printf("Connect player: Name: %s. ID: %d.\n", Name, my_id);
    while (1)
    {
        sendto(newsockfd, "Waiting the start of hoax\n", 26, 0, (struct sockaddr *)
&client, slen);

```

```

while (Flag_start_hoax == 0)
{
    for (i = 0; i < Koll_Connect; i++)
    {
        if (All_Connects[i].id_con == my_id)
        {
            if (All_Connects[i].flag_dis == 1)
                disconnect(newsockfd, my_id, Name);
            break;
        }
    }

    rc = recvfrom(newsockfd, c, 1, 0, (struct sockaddr *) &client,
&slen);
    if (rc > 0)
    {
        if (c[0] == '1')
        {
            disconnect(newsockfd, my_id, Name);
        }
        if (c[0] == '2')
        {
            cout << "work";
            sendto(newsockfd, "The hoax didn't start yet\n", 27, 0,
(struct sockaddr *) &client, slen);
        }
        if (c[0] == '3')
        {
            sendto(newsockfd, "The hoax didn't start yet\n", 27, 0,
(struct sockaddr *) &client, slen);
            recvfrom(newsockfd, buf, 255, 0, (struct sockaddr *)
&client, &slen);

            bool flag = true;
            do{
                if ((buf[0] == 'n') && flag)
                {
                    do{
                        recvfrom(newsockfd, buf, 255, 0,
(struct sockaddr *) &client, &slen);

                    } while (buf[0] != '.');
                    flag = false;
                }
            } while (buf[0] != '.');
            recvfrom(newsockfd, buf, 255, 0, (struct
sockaddr *) &client, &slen);

            } while (buf[0] != '.');
            recvfrom(newsockfd, buf, 255, 0, (struct sockaddr *)
&client, &slen);

            bzero(buf, 255);
        }
    }
}

sendto(newsockfd, "Hoax starts!\n", 13, 0, (struct sockaddr *) &client,
slen);

Flag_Rate = 0;
while (Flag_finish_hoax == 0)
{
    for (i = 0; i < Koll_Connect; i++)
    {
        if (All_Connects[i].id_con == my_id)
        {
            if (All_Connects[i].flag_dis == 1)
                disconnect(newsockfd, my_id, Name);
            break;
        }
    }
}

```



```

rc = recvfrom(newsockfd, c, 1, 0, (struct sockaddr *) &client,
&slen);

if (rc>0)
{
    if (c[0] == '1')
    {
        disconnect(newsockfd, my_id, Name);
    }
    if (c[0] == '2')
    {
        Watch_Rate(newsockfd, my_id);
    }
    if (c[0] == '3')
    {
        if (Flag_Rate == 0)
        {
            Do_Rate(newsockfd, my_id, Name);
            Flag_Rate = 1;
        }
        else
        {
            sendto(newsockfd, "You already rate! Wait
results.\n", 33, 0, (struct sockaddr *) &client, slen);
            recvfrom(newsockfd, buf, 255, 0, (struct
sockaddr *) &client, &slen);

            bool flag = true;
            do{
                if ((buf[0] == 'n') && flag)
                {
                    do{
                        recvfrom(newsockfd, buf,
255, 0, (struct sockaddr *) &client, &slen);

                        } while (buf[0] != '.');
                        flag = false;
                    }
                    recvfrom(newsockfd, buf, 255, 0, (struct
sockaddr *) &client, &slen);

                    } while (buf[0] != '.');
                    recvfrom(newsockfd, buf, 255, 0, (struct
sockaddr *) &client, &slen);

                    bzero(buf, 255);
                }
            }
        }
    }
}

void disconnect(SOCKET newsockfd, int my_id, char *Name)
{
    int flag, i, KolDis;

    for (i = 0; i<Koll_Connect; i++)
    {
        if (All_Connects[i].id_con == my_id)
        {
            KolDis = i;
            break;
        }
    }
    sendto(newsockfd, "You disconnected\n", 17, 0, (struct sockaddr *)
&All_Connects[KolDis].client, All_Connects[KolDis].slen);
    if (Koll_Connect != 1)
    {
        for (i = KolDis; i<Koll_Connect - 1; i++)
        {

```

```

        All_Connects[i].id_con = All_Connects[i + 1].id_con;
        All_Connects[i].i_sock = All_Connects[i + 1].i_sock;
        All_Connects[i].Name = All_Connects[i + 1].Name;
        All_Connects[i].flag_dis = All_Connects[i + 1].flag_dis;
        All_Connects[i].client = All_Connects[i + 1].client;
        All_Connects[i].slen = All_Connects[i + 1].slen;
    }
    Koll_Connect = Koll_Connect - 1;
    All_Connects = (struct Connect_Info*) realloc(All_Connects, sizeof(struct
Connect_Info)*Koll_Connect);
}
else
{
    Koll_Connect = 0;
    free(All_Connects);
    All_Connects = NULL;
}

printf("Disconnected: Name: %s ID:%d\n", Name, my_id);
flag = shutdown(newsockfd, SD_BOTH);
if (flag < 0) {
    perror("ERROR shutdown socket");
    exit(1);
}
flag = closesocket(newsockfd);
if (flag < 0) {
    perror("ERROR close socket");
    exit(1);
}
//free(Name);
}
void Watch_Rate(SOCKET newsockfd, int my_id)
{
    int Id_c;
    for (int i = 0; i < Koll_Connect; i++)
    {
        if (All_Connects[i].id_con == my_id)
        {
            Id_c = i;
            break;
        }
    }
    char buf[500]; //, message[105];
    int i;
    //snprintf(message, 105, "RATES\nType:\n e-even\n o-odd\n n-number(after that
write the number)\nName | ID | Type | SUMM\n\n");
    //write(newsockfd, message, strlen(message));
    for (i = 0; i < Koll_Rate; i++)
    {
        //write(newsockfd, All_Rates[i].Name, strlen(All_Rates[i].Name));
        //write(newsockfd, " ", 1);
        //snprintf(buf, 60, "%d", All_Rates[i].id_pl);
        //write(newsockfd, buf, strlen(buf));
        //write(newsockfd, " ", 1);
        //write(newsockfd, All_Rates[i].type, 1);
        //write(newsockfd, " ", 1);
        //string message="";
        if (All_Rates[i].type == 'n')
            snprintf(buf, 500, "%s %d %c %d %d$\n", All_Rates[i].Name,
All_Rates[i].id_pl, All_Rates[i].type, All_Rates[i].number, All_Rates[i].summa);
        else
            snprintf(buf, 500, "%s %d %c %d$\n", All_Rates[i].Name,
All_Rates[i].id_pl, All_Rates[i].type, All_Rates[i].summa);
        sendto(newsockfd, buf, strlen(buf), 0, (struct sockaddr *)
&All_Connects[Id_c].client, All_Connects[Id_c].slen);
    }
}

```

```

    }
}
void Do_Rate(SOCKET newsockfd, int my_id, char *Name)
{
    int Id_c;
    for (int i = 0; i < Koll_Connect; i++)
    {
        if (All_Connects[i].id_con == my_id)
        {
            Id_c = i;
            break;
        }
    }
    char type[2], c[1];
    int summa, n, number;
    recvfrom(newsockfd, type, 1, 0, (struct sockaddr *) &All_Connects[Id_c].client,
    &All_Connects[Id_c].slen);
    //cout << type << endl<<"*****"<<endl;
    Sleep(100);
    summa = 0;
    recvfrom(newsockfd, c, 1, 0, (struct sockaddr *) &All_Connects[Id_c].client,
    &All_Connects[Id_c].slen);
    //cout << c[0] << endl;
    Sleep(100);
    do{
        if ((c[0] == 'n') || (c[0] == 'e') || (c[0] == 'o'))
        {
            type[0] = c[0];
            bzero(c, 1);
            recvfrom(newsockfd, c, 1, 0, (struct sockaddr *)
    &All_Connects[Id_c].client, &All_Connects[Id_c].slen);
        }
        else
        {
            summa = summa * 10;
            n = (int)c[0] - (int)'0';
            summa = summa + n;
            recvfrom(newsockfd, c, 1, 0, (struct sockaddr *)
    &All_Connects[Id_c].client, &All_Connects[Id_c].slen);
            //cout << c << endl;
            Sleep(100);
        }
    } while (c[0] != '.');
    if (type[0] == 'n')
    {
        number = 0;
        recvfrom(newsockfd, c, 1, 0, (struct sockaddr *)
    &All_Connects[Id_c].client, &All_Connects[Id_c].slen);
        do{
            number = number * 10;
            n = (int)c[0] - (int)'0';
            number = number + n;
            recvfrom(newsockfd, c, 1, 0, (struct sockaddr *)
    &All_Connects[Id_c].client, &All_Connects[Id_c].slen);
        } while (c[0] != '.');
    }
    type[1] = '\0';
    if (type[0] == 'n')
        printf("New Rate from %s: Type %s Summ %d Number %d\n", Name, type, summa,
    number);
    else
        printf("New Rate from %s: Type %s Summ %d\n", Name, type, summa);
    Koll_Rate = Koll_Rate + 1;
    if (Koll_Rate == 1)
        All_Rates = (struct Rates_Info*) malloc(sizeof(struct Rates_Info));
}

```

```

        else
            All_Rates = (struct Rates_Info*) realloc(All_Rates, sizeof(struct
Rates_Info)*Koll_Rate);
            All_Rates[Koll_Rate - 1].id_pl = my_id;
            All_Rates[Koll_Rate - 1].Name = Name;
            All_Rates[Koll_Rate - 1].i_sock = newsockfd;
            All_Rates[Koll_Rate - 1].type = type[0];
            All_Rates[Koll_Rate - 1].summa = summa;
            All_Rates[Koll_Rate - 1].number = number;
    }
    unsigned __stdcall Server(void * arg)
    {
        int i, dis_id, n, flag;
        char buf[40];
        dis_id = 0;
        while (1)
        {
            scanf("%s", buf);
            flag = 0;
            if ((buf[0] == 'd') && (buf[9] == 't') && (buf[10] == '_'))
            {
                dis_id = 0;
                for (i = 11; buf[i] != '.'; i++)
                {
                    dis_id = dis_id * 10;
                    n = (int)buf[i] - (int)'0';
                    if ((n<0) || (n>9))
                    {
                        flag = 1;
                        break;
                    }
                    dis_id = dis_id + n;
                }
                if (flag == 1)
                {
                    printf("Error: write the comand correctly: disconnect_id.\n");
                }
                else
                {
                    disconnect_serv(dis_id);
                }
            }
        }
    }

    void disconnect_serv(int my_id)
    {
        int i, KolDis;
        KolDis = -1;
        for (i = 0; i<Koll_Connect; i++)
        {
            if (All_Connects[i].id_con == my_id)
            {
                KolDis = i;
                break;
            }
        }
        if (KolDis == -1)
        {
            printf("Error: Cant find client with this ID\n");
            return;
        }
        All_Connects[KolDis].flag_dis = 1;
    }

```

Листинг 4. Клиентское приложение сетевой рулетки UDP.

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <strings.h>
using namespace std;
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
void * Print_Server_Mes(void * arg);

bool have_connect = false;
struct sockaddr_in si_other;
int slen = sizeof(si_other);
int main(int argc, char *argv[])
{
    //struct sockaddr_in si_other;
    int s;
    //int slen = sizeof(si_other);
    char buf[255];
    char message[255];
    while (1)
    {
        bool start = true;
        while (start)
        {
            char choose;
            cout << "What do you want?:\n          1:connect\n   2:exit\n Your choose: ";
            cin >> choose;
            switch (choose)
            {
                case '1': start = false; break;
                case '2': return 0; break;
                default: cout << "That's not a choice.\n";
            }
        }
        //create socket
        if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        {
            printf("socket() failed");
            exit(EXIT_FAILURE);
        }

        /*cout << "Enter server IP adress: ";
        string ip_adress;
        cin >> ip_adress;
        cout << "Enter number of port: ";
        int port;
        cin >> port;
        server.sin_addr.s_addr = inet_addr(ip_adress.c_str());
        server.sin_family = AF_INET;
        server.sin_port = htons(port);*/
        //setup address structure
        bzero((char *)&si_other, sizeof(si_other));
        si_other.sin_family = AF_INET;
        si_other.sin_port = htons(8000);
        si_other.sin_addr.s_addr = inet_addr("192.168.56.101");
```

```

bool flag_croupier = false;
bool flag = true;
string mes_to_serv;
char user_a;
int sleep_time = 1;

while (flag)
{
    cout << "Do you want to be a croupier? y/n\n";
    string answer;
    cin >> answer;
    if ((answer != "y") && (answer != "n"))
        cout << "Error. Write Y or N";
    if (answer == "y")
    {
        mes_to_serv = "1";
        flag = false;
        flag_croupier = true;
    }
    else if (answer == "n")
    {
        mes_to_serv = "0";
        flag = false;
    }
}
cout << "Enter your name: ";
string name;
cin >> name;
mes_to_serv += name;
mes_to_serv += '.';
//say connect to server
mes_to_serv += '.';
//send the message
if (sendto(s, "connect", 7, 0, (struct sockaddr *) &si_other, slen) < 0)
{
    printf("sendto() failed");
    exit(EXIT_FAILURE);
}
bzero(buf, 255);
//try to receive some data, this is a blocking call
if (recvfrom(s, buf, 255, 0, (struct sockaddr *) &si_other, (socklen_t*)&slen)
< 0)
{
    printf("recvfrom() failed");
    exit(EXIT_FAILURE);
}
int new_port = atoi(buf);
cout<<new_port<<endl;
si_other.sin_port = htons(new_port);
//send the message
if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct sockaddr *)
&si_other, slen) < 0)
{
    printf("sendto() failed");
    exit(EXIT_FAILURE);
}

char server_reply[2000];
int rcv_size;
if ((rcv_size = recvfrom(s, server_reply, 2000, 0, (struct sockaddr *)
&si_other, (socklen_t*) &slen)) < 0)
{
    puts("recv failed");
}
server_reply[rcv_size] = '\0';

```

```

cout << "Server: " << server_reply << "\n";
std::string buffer(server_reply);
if (buffer.find("We already have a croupier.") != string::npos)
    flag_croupier = false;

have_connect = true;

int flag_thr;
pthread_attr_t threadAttr;
pthread_attr_init(&threadAttr);
pthread_attr_setdetachstate(&threadAttr, PTHREAD_CREATE_DETACHED);
pthread_t serv;
flag_thr = pthread_create(&serv, &threadAttr, Print_Server_Mes, (void*)s);
if(flag_thr != 0)
{
    perror( "Creating thread false");
    exit(1);
}

if (flag_croupier == true)
{
    while (have_connect)
    {
        cout << "You can:\n";
        cout << "  1: Disconnect\n";
        cout << "  2: Watch rates\n";
        cout << "  3: Start hoaxes\n";
        cout << "  4: Finish hoaxes\n";
        cin >> user_a;
        if (have_connect == false) break;
        switch (user_a)
        {
            case '1': mes_to_serv = "1";
                if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
                {
                    puts("Send failed\n");
                };
                have_connect = false;
                sleep(sleep_time);
                break;
            case '2': mes_to_serv = "2";
                if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
                {
                    puts("Send failed\n");
                };
                sleep(sleep_time);
                break;
            case '3': mes_to_serv = "4";
                if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
                {
                    puts("Send failed\n");
                };
                sleep(sleep_time);
                break;
            case '4': mes_to_serv = "5";
                if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
                {
                    puts("Send failed\n");
                };
                sleep(sleep_time);
                break;
        }
    }
}

```

```

        default: cout << "That's not a choice.\n";
    }
}
else
{
    while (have_connect)
    {
        cout << "You can:\n";
        cout << "  1: Disconnect\n";
        cout << "  2: Watch rates\n";
        cout << "  3: Make rate\n";
        cin >> user_a;
        if (have_connect == false) break;
        int true_rate = 1;
        char rate_type;
        string type_r = "3";
        switch (user_a)
        {
            case '1': mes_to_serv = "1";
                if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
                {
                    puts("Send failed\n");
                };
                have_connect = false;
                sleep(sleep_time);
                break;
            case '2': mes_to_serv = "2";
                if (sendto(s, mes_to_serv.c_str(), mes_to_serv.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
                {
                    puts("Send failed\n");
                };
                sleep(sleep_time);
                break;
            case '3': while (true_rate)
                {
                    cout << "Choose the type of rate:\n";
                    cout << "      e - even\n      o - odd\n      n - number\n";
                    cin >> rate_type;
                    switch (rate_type)
                    {
                        case 'e': mes_to_serv = "e"; true_rate = 0;
break;
                        case 'o': mes_to_serv = "o"; true_rate = 0;
break;
                        case 'n': mes_to_serv = "n"; true_rate = 0;
break;
                        default: cout << "That's not a choice.\n";
                    }
                }
            cout << "Enter a cash:\n";
            int cash;
            cin >> cash;
            mes_to_serv += to_string(cash) + ".";
            if (rate_type == 'n')
            {
                cout << "Enter a number:\n";
                int number;
                cin >> number;
                mes_to_serv += to_string(number) + ".";
            }
            if (sendto(s, type_r.c_str(), type_r.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)

```



```

        {
            puts("Send failed\n");
        };
        for (int i = 0; i < mes_to_serv.length(); i++)
        {
            type_r = mes_to_serv[i];
            if (sendto(s, type_r.c_str(), type_r.length(), 0, (struct
sockaddr *) &si_other, slen) < 0)
            {
                puts("Send failed\n");
            };
        }
        sleep(sleep_time);
        break;
        default: cout << "That's not a choice.\n";
    }
}
}
}

void *Print_Server_Mes(void * arg)
{
    char server_reply[2000];
    int recv_size;
    int s = (int)arg;
    while (have_connect)
    {
        if ((recv_size = recvfrom(s, server_reply, 2000, 0, (struct sockaddr *)
&si_other, (socklen_t*) &slen)) < 0)
        {
            puts("recv failed");
            string mes = server_reply;
            cout << "Server:  " << mes;
            return 0;
        }
        server_reply[recv_size] = '\0';
        string mes = server_reply;
        if (mes.find("You disconnected") != string::npos)
        {
            cout << "Server:  " << mes;
            have_connect = false;
            return 0;
        }
        cout << "Server:  " << server_reply;
    }
    return 0;
}

```