

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

---

**Кафедра компьютерных систем и программных технологий**

**Отчет**

**Дисциплина: сетевые технологии**

**Тема: изучение протоколов TCP/UDP на основе разработки  
системы мгновенного обмена сообщений между пользователями**

**Студентка гр.43501/3: \_\_\_\_\_ Замотаева Ю.И**

**Преподаватель: \_\_\_\_\_ Вылегжанина К. Д**

**Санкт-Петербург  
2014**

## **Глава 1**

Система мгновенного обмена сообщений между пользователями

### **1.1 Функциональные требования**

#### **1.1.1 Задание**

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

#### **1.1.2 Основные возможности**

Серверное приложение должно реализовывать следующие функции:

- Прослушивание определенного порта
- Обработка запросов на подключение по этому порту от клиентов
- Поддержка одновременной работы нескольких клиентов через механизм нитей
- Передача текстового сообщения одному клиенту
- Передача текстового сообщения всем клиентам (реклама)
- Прием и ретрансляция входящих сообщений от клиентов
- Обработка запроса на отключение клиента
- Принудительное отключение указанного клиента

Клиентское приложение должно реализовывать следующие функции:

- Установление соединения с сервером
- Передача сообщения указанному клиенту
- Прием сообщения от сервера (сообщений от других пользователей и получение рекламы)
- Разрыв соединения
- Обработка ситуации отключения клиента сервером

### **1.2 Нефункциональные требования**

#### **1.2.1 Требования к реализации**

Разработанные приложения должны иметь понятный пользователю интерфейс для удобной работы с ними. Приложения должны поддерживать кроссплатформенность - работать в Windows и Linux. Также передача и прием данных должны обеспечиваться по протоколам TCP и UDP.

#### **1.2.2 Требования к надежности**

Длина всего отправляемого пакета от клиента серверу (и наоборот) должна проверяться на максимальное значение, так мы защищаем сервера и клиента от «падения» при отправке слишком длинного пакета. Контролируем длину всего передаваемого пакета (поле длина пакета), чтобы корректно извлекать нужное количество данных при получении пакета.

При отправке от клиента серверу запросов на наличие сервера в сети, получении логина, списка пользователей формируем пакеты длиной=6 (3 байта на длину пакета, 2 байта на код команды и 1 байт данных), так как больше нам и не требуется.

### 1.3 Накладываемые ограничения

- Ограничение на длину пакетов: максимальный размер пакета 512 символов;
- Не поддерживаются русские символы.

## Глава 2

### Реализация для работы по протоколу TCP

#### 2.1 Прикладной протокол

Рассмотрим структуру данных (пакет), с помощью которой сервер и клиент обмениваются сообщениями. Формат пакета представлен в следующей таблице:

Длина всего пакета	Тип команды	Данные
3 байта	2 байта	Длина данных ограничена только максимальным размером пакета

Рассмотрим команды, которые используются в данном протоколе:

Команда	ID	Пример (поле данных)	Расшифровка
Echo Request - клиент проверяет наличие сервера в сети	0	1	1 - флаг окончания данных
Echo Answer - сервер отвечает клиенту о своем присутствии в сети	1	1	1 - флаг окончания данных
Login Request - клиент посылает свой логин серверу	2	05julia	05 – длина логина julia - логин
Login Answer - сервер отвечает клиенту о возможности регистрации в сети	3	1 0	1 – логин принят 0 – логин занят или некорректен
Users Request - клиент запрашивает список онлайн-пользователей	4	1	1 - флаг окончания данных
User Answer - сервер выдает клиенту список онлайн-пользователей	5	00205julia04user 000	002 – число пользователей 05 – длина логина julia - логин
Unauthorized action – сервер говорит пользователю, что он не авторизован в сети	7	08	08 – id запрещенной для анонимного пользователя команды
Message – клиент посылает сообщение другому клиенту через сервер	8	05julia06gekkon 25Mon_Nov_10_01:22:24 2014_005hello	05 - длина имени получателя Julia - получатель 06 - длина имени отправителя Gekkon - отправитель 25 – длина времени отправления 01:22:24 – время отправления 005 – длина сообщения

			Hello - сообщение
Message: Incorrect name - сервер уведомляет клиента, что пользователь с запрашиваемым именем не найден	9	05julia	05 – длина имени пользователя  Julia – некорректное имя / отсутствующий пользователь
Message: User offline - сервер уведомляет клиента, что пользователь с запрашиваемым именем оффлайн	11	05julia	05 – длина имени пользователя  Julia – пользователь offline
Message: Success sended - сервер уведомляет клиента об успешной доставке сообщения другому клиенту	12	1	1 - флаг окончания данных
Advert – рассылка сервером рекламы	13	022 <a href="https://www.google.ru/">https://www.google.ru/</a>	022 – длина рекламы Google - реклама
Quit Request - клиент уведомляет сервера о своем отключении	16	1	1 - флаг окончания данных

Команды с ID 6,10,14,15 не имеют формата поля данных и не используются.

Рассмотрим Sequence Diagram протокола (схему обмена данными), которая представлена на рис.1 и проанализируем последовательность действий. Первым запускается сервер, начинает прослушивать определенный порт и ждет подключение от клиента. Клиент хочет удостовериться о наличии сервера в сети, для этого он отправляет серверу соответствующий запрос (ECHO REQUEST). После принятия данного запроса, сервер отвечает клиенту о своем присутствии в сети (ECHO ANSWER). После того, как соединение установлено, клиент может вводить свой логин - на стороне клиента появляется сообщение: input your name. Клиент вводит свой логин, серверу отправляется LOGIN REQUEST. После принятия такого запроса сервер анализирует данный логин – нет ли еще пользователя с таким же логином, проверяет длину логина (не больше 32 символов) и наличие в логине запрещенных символов, далее сервер отправляет клиенту ответ ( LOGIN\_ ANSWER) принят ли данный логин или нет. Если логин не принят, то клиенту предлагает ввести другой логин, если принят, то клиент отправляет серверу запрос на получение списка клиентов онлайн (USERS\_REQUEST). Сервер в ответ отправляет клиенту список клиентов онлайн (USERS\_ANSWER). Далее клиент может выбрать нужного пользователя и отправить ему сообщение через сервер, так и происходит общение клиентов друг с другом. Клиент может самостоятельно разорвать соединение, посылая при этом серверу соответствующее уведомление QUIT\_REQUEST. Сервер контролирует наличие пользователя в сети, периодически посылая ему соответствующий запрос ALIVE\_REQUEST . Пользователь должен ответить на данный запрос ALIVE\_ANSWER, чтобы оставаться в сети и сервер не отключил его.

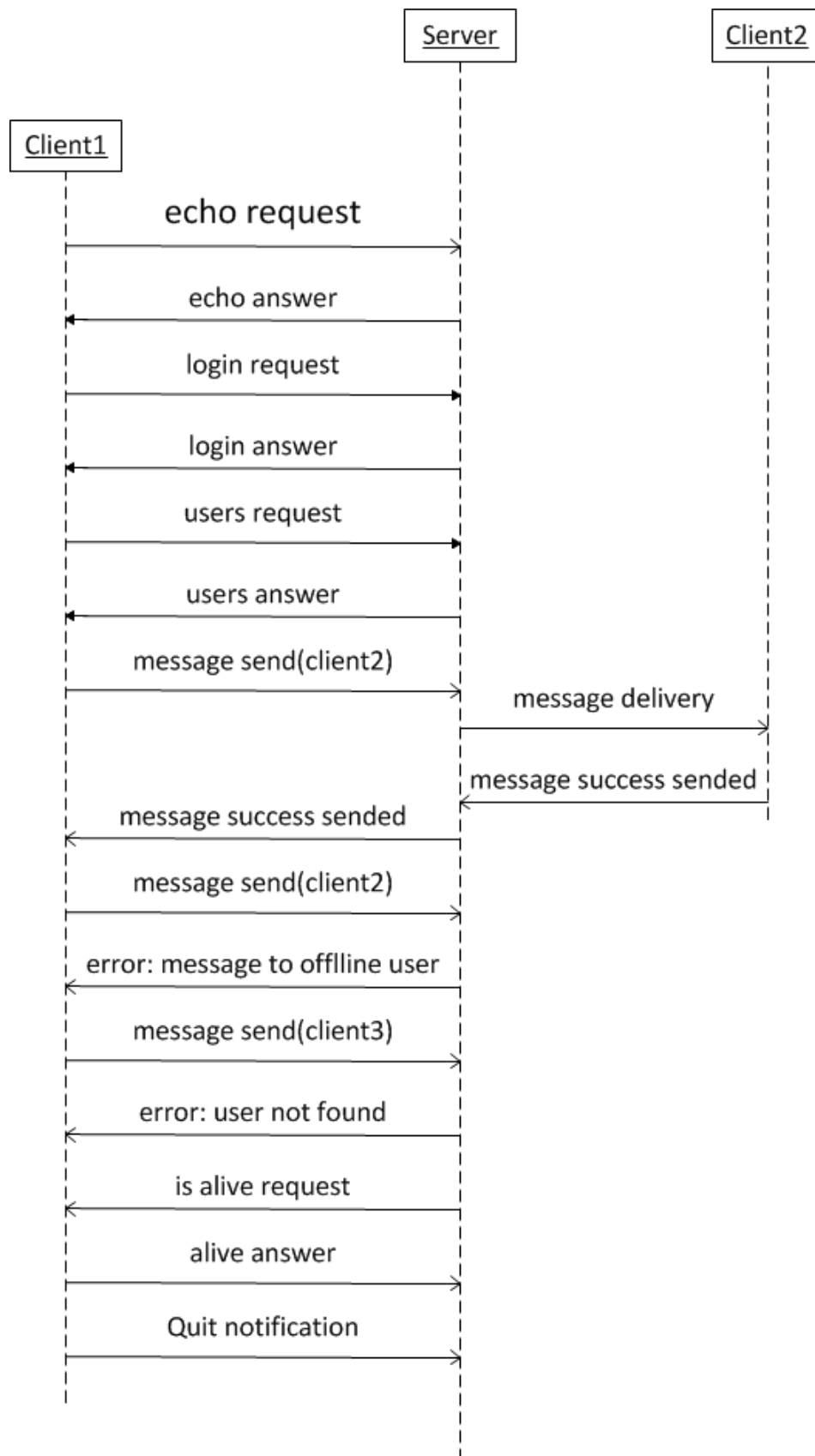
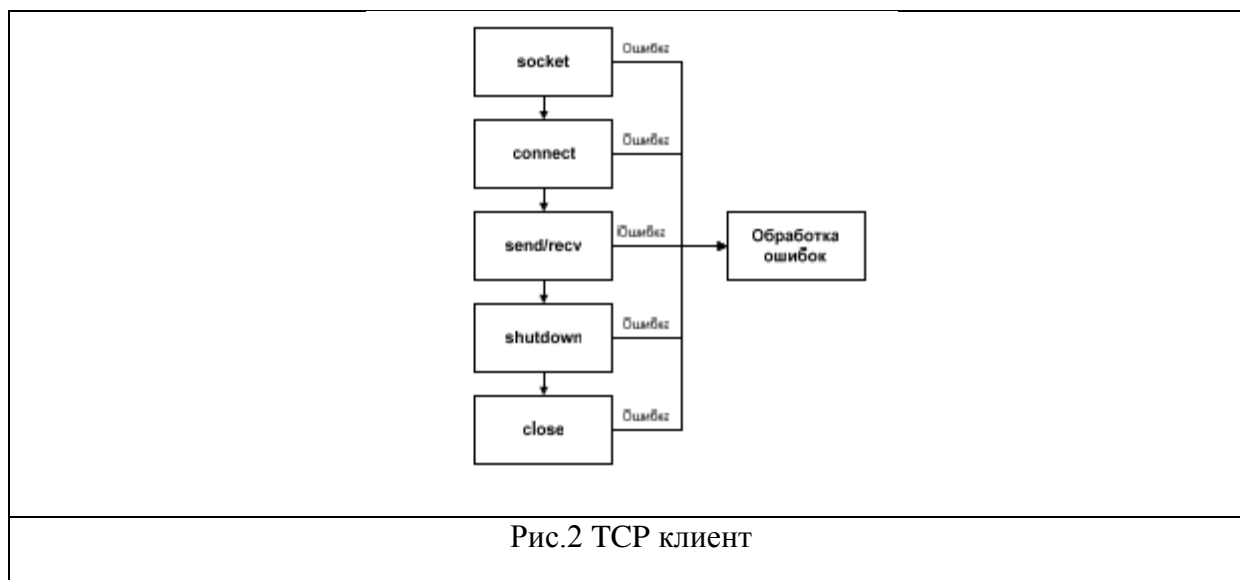


Рис.1 Sequence Diagram

## 2.2 Архитектура приложения

### 2.2.1 Описание клиента

После запуска приложения происходит создание соединения к серверу (настройка библиотеки для работы в Windows или Linux, создание сокета, задания адреса сервера в сети). Далее клиент осуществляет обмен данными, в соответствии с ранее изложенным протоколом прикладного уровня. Структура TCP-клиента представлена на рис.2.



Для проверки наличия сервера в сети отправляется echo request. После получения ответа (echo answer), отправляется запрос на введенный пользователем логин (login request). Если логин отвечает условиям уникальности и длины, сервер отправляет положительный ответ (login answer с «1» в поле данных) – клиент может отправлять и получать данные. Иначе приложение попросит пользователя повторно ввести логин.

Дальше приложение - клиент работает в двух потоках – один обеспечивает ввод / вывод данных (IO thread) – которые пользователь видит в консоли, а второй получает и отправляет данные по сети (socket thread), что можно увидеть на рис.3. IO thread также может отправлять три типа пакетов – Users Request, Message Send, Quit Request (в зависимости от нажатия пользователем определенного номера клавиши), но получения ответа на них происходит в socket Thread. Для взаимодействия между потоками используется 4 указателя – логин, дескриптор сокета, список пользователей, список сообщений.

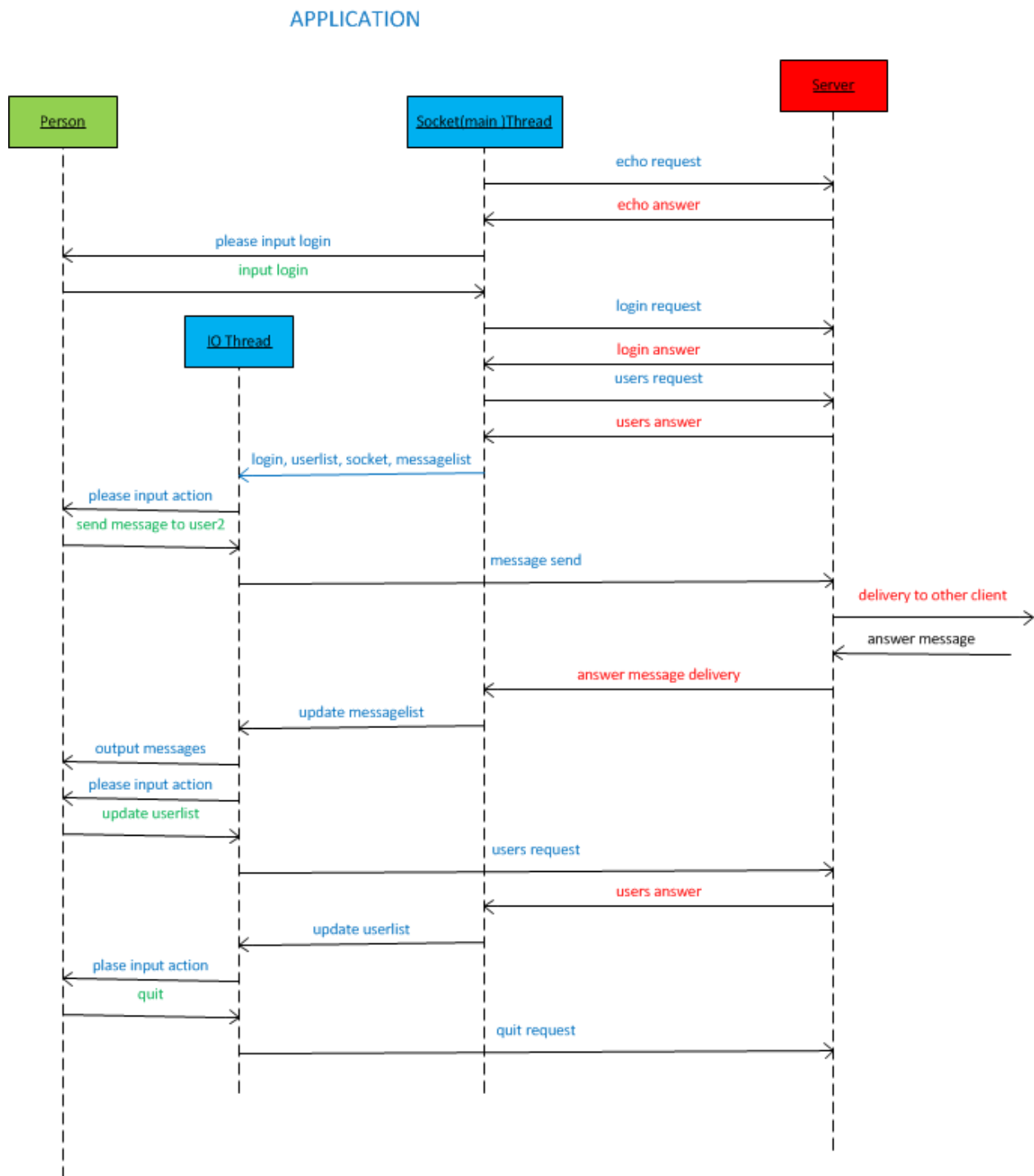


Рис.3 Работа клиента и сервера

Для удобства пользователя ему предоставляется список возможных действий:

1. Отправить сообщение
2. Посмотреть сообщения
3. Посмотреть пользователей онлайн
4. Выйти

Над меню (рис.4) располагается имя самого пользователя, счетчик сообщений и пользователей онлайн во время последнего действия. Таким образом, пользователь может просматривать сообщения (от других пользователей и приходящую от сервера рекламу), отправлять сообщения другим пользователям, просматривать пользователей онлайн и выйти.

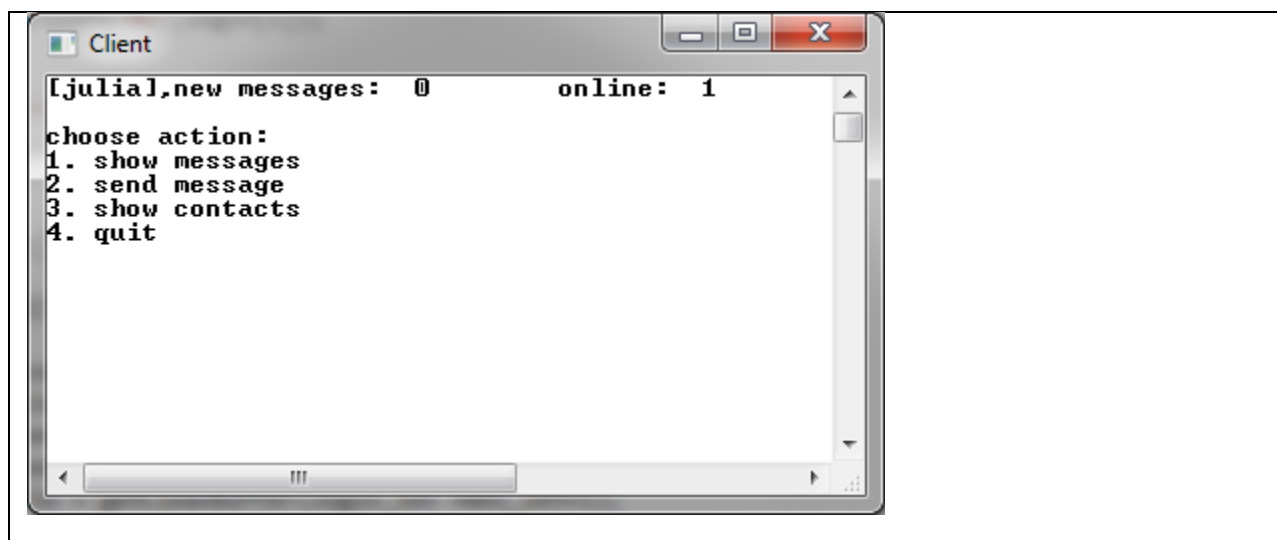


Рис.4 Меню пользователя

## 2.2.2 Описание сервера

Организация TCP-сервера отличается от TCP-клиента в первую очередь созданием слушающего сокета. Такой сокет находится в состоянии listen и предназначен только для приёма входящих соединений. В случае прихода запроса на соединение создаётся дополнительный сокет, который и занимается обменом данными с клиентом. Структура TCP-сервера и взаимосвязь сокетов изображена на рис.5.

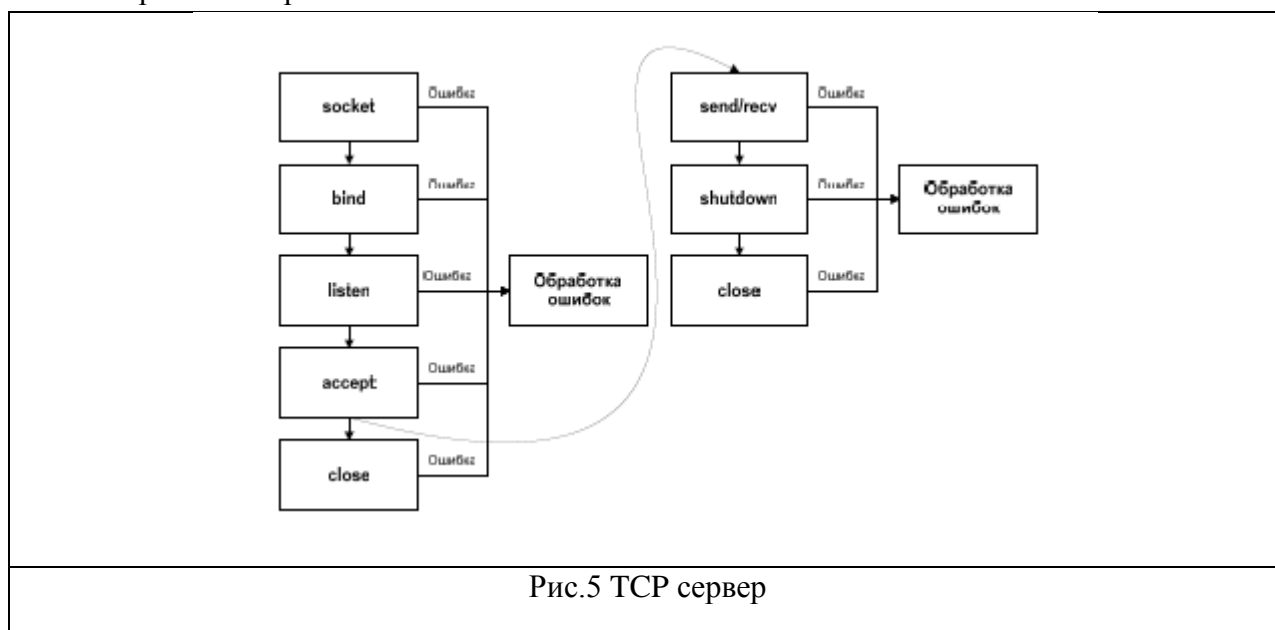


Рис.5 TCP сервер



Итак, после настройки сокета (настройка библиотеки для windows или линукс, bind и переключения в режим слушания сокета), сервер ждет подключения от клиентов.

Для хранения списка пользователей на сервере есть список, который хранит следующие сведения о подключившемся клиенте – идентификатор сокета, имя пользователя и время до отключения.

При новом подключении создается новая запись, в которую записывается идентификатор сокета, получаемый из функции accept, имя пользователя из Login Request (при правильном имени), а время жизни выставляется в максимально возможное для сервера значение (1 минута и 20 секунд).

Для каждого подключения создается новый поток, в котором и происходит отправка и получение пакетов (для TCP).

Существует также отдельный поток для отключения пользователей по таймауту, и поток для отправления рекламы.

Каждый входящий пакет восстанавливает время жизни клиента до максимального значения, но если от пользователя не приходят данные, то время жизни (время до отключения) уменьшается.

Каждый три секунды происходит уменьшение времени до отключения. Если время до отключения становится меньше определенного значения (20 секунд), то отправляется echo запрос. Если ответ приходит – то клиент не отключается. Если не приходит echo answer, то через 20 секунд пользователь отключается, сокет закрывается, а запись из списка пользователей удаляется.

Также клиент сам может послать запрос на отключение.

Работу сервера можно увидеть на рис.6-10. Вначале запускаем сервер, далее запускаем клиентов. Видно, что соединения приняты, клиенты зарегистрированы. Наблюдаем на сервере все запросы, приходящие к серверу и отсылаемые клиентам. Видим, что клиентам каждую минуту приходит реклама. Клиенты посылают сообщения друг другу, все работает корректно. Сервер проверяет наличие клиентов в сети – отправка предупреждения отключения каждую минуту (когда до отключения остается 20 сек). Пока пользователи находятся онлайн, они посылают серверу Echo answer, и тогда сервер их не отключает.

Один из клиентов (Masha) вышел – серверу приходит Quit request, тогда сервер удаляет данного пользователя из списка пользователей. Клиент Julia просто закрыла приложение, не уведомив сервер о своем выходе (не посылая Quit request). Тогда сервер, не получив ответа от клиента на посылаемый timeout warning (предупреждение об отключении), самостоятельно отключает данного пользователя и удаляет его из списка пользователей.

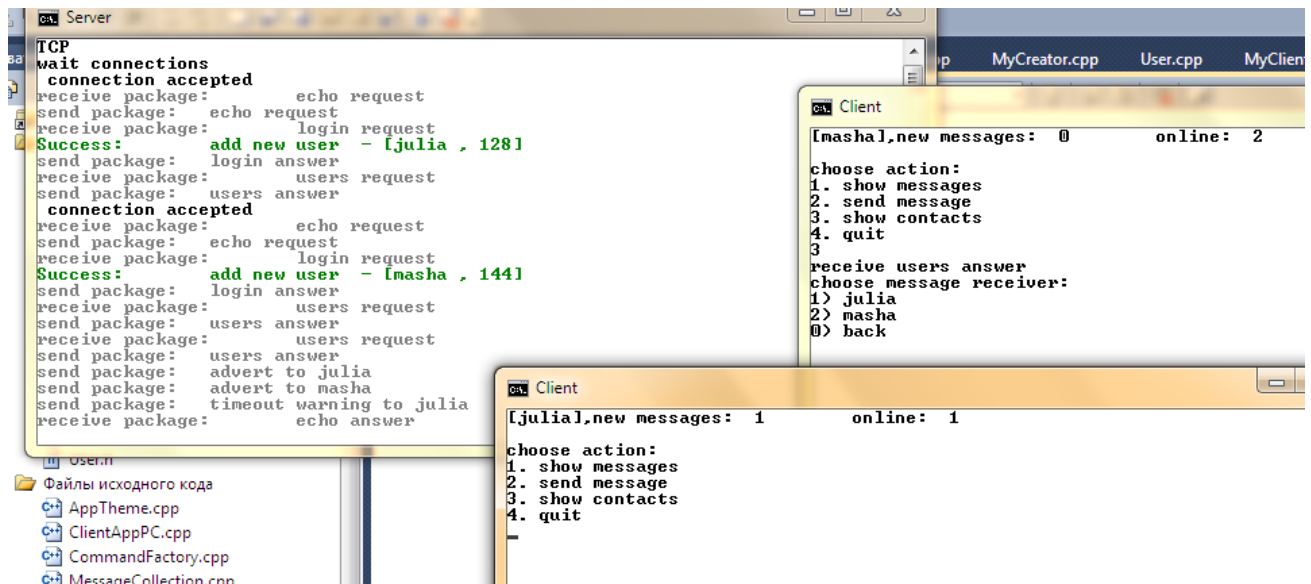


Рис.6

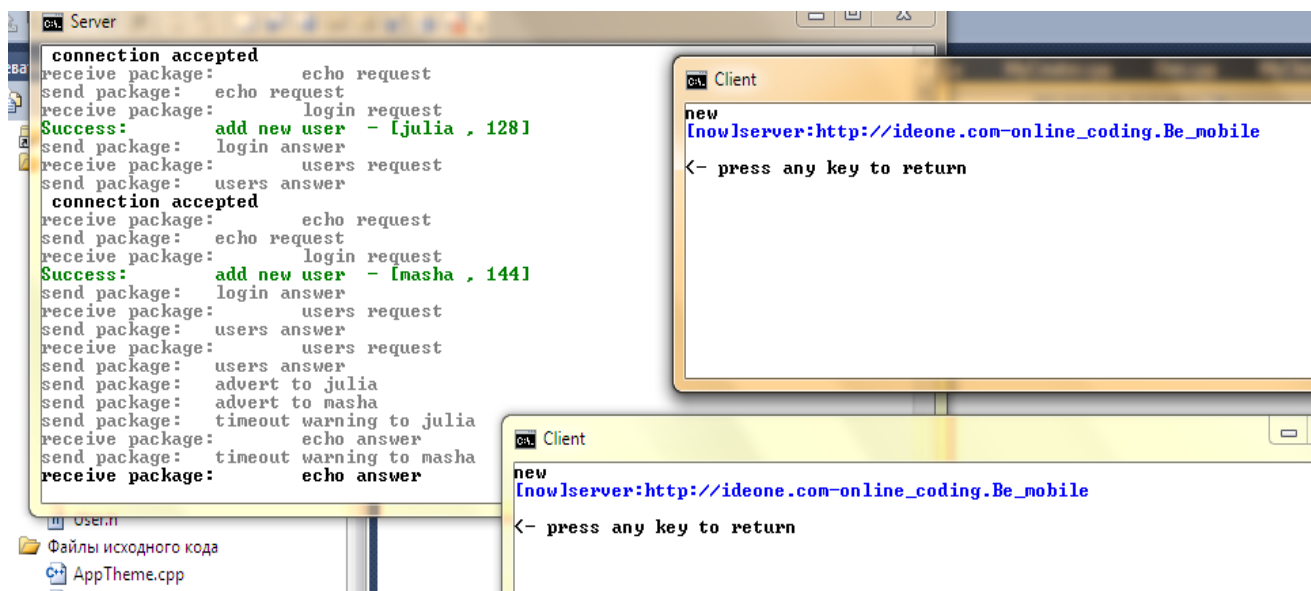


Рис.7

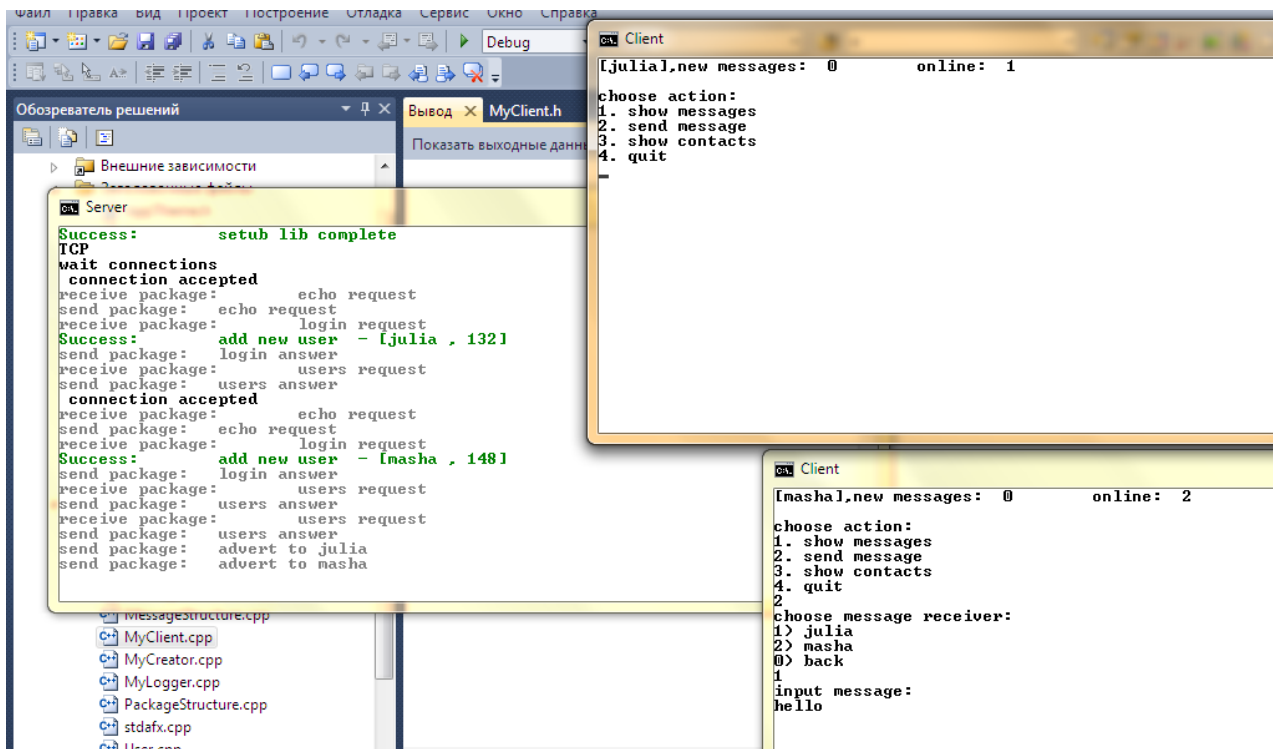


Рис.8

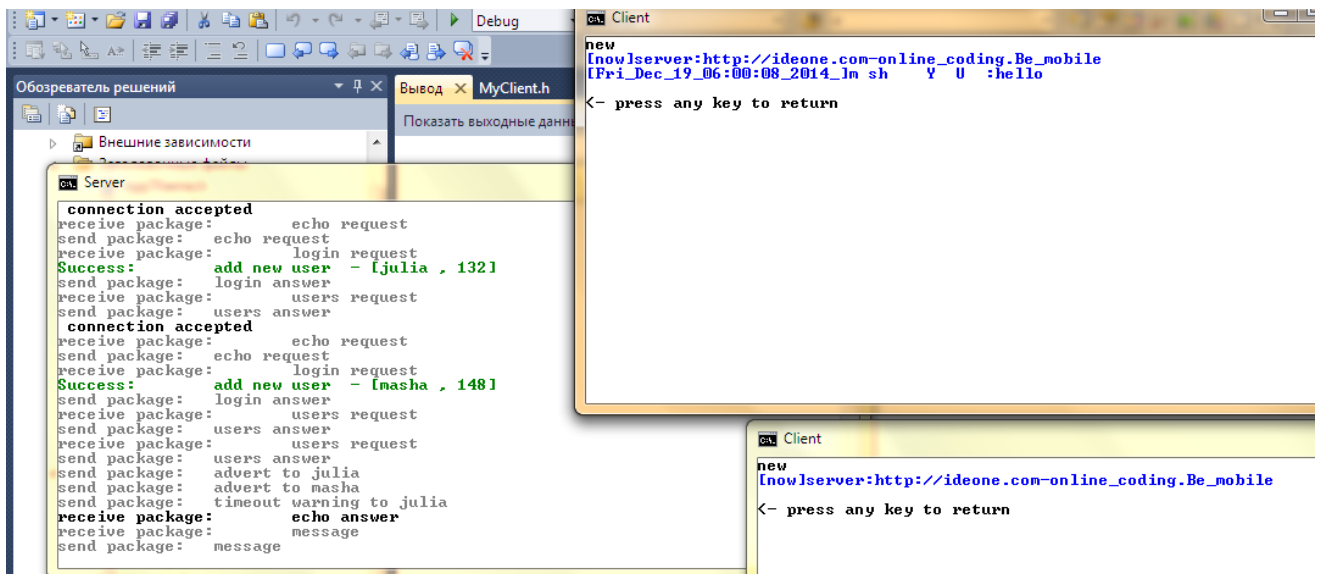


Рис.9



```
Server
user deleted[quit]: masha
send package: timeout warning to julia
receive package: echo answer
send package: advert to julia
send package: timeout warning to julia
receive package: echo answer
send package: advert to julia
send package: timeout warning to julia
receive package: echo answer
send package: advert to julia
send package: timeout warning to julia
receive package: echo answer
send package: advert to julia
send package: timeout warning to julia
receive package: echo answer
send package: advert to julia
send package: timeout warning to julia
receive package: echo answer
fail to send advert to julia
fail to send timeout warning to julia
user deleted[timeout]: julia
```

Рис.10

### 2.2.3 Обмен сообщениями

Каждый пользователь может отправлять сообщения только серверу, так как знает только его адрес в сети. Для отправления сообщения другому пользователю нужно знать имя этого пользователя. Его можно получить из списка пользователей (отправить Users Request, и расшифровать Users Answer).

После выбора получателя и ввода сообщения создается пакет для сервера, с полем данных:

05julia06gekkon25Mon\_Nov\_10\_01:22:24 2014\_005hello

Полная расшифровка описана в таблице описания команд, но для обмена сообщениями интересны только первые две записи: длина получателя и его имя: 05julia.

Сервер ищет пользователя с таким именем в списке, и если находит, то пересылает сообщение без изменений с помощью сокета, ассоциированного с именем, а получателю посылается Message Delivery.

Если же клиент отключен, но еще не удален из списка пользователей, то отправителю посылается Message User Offline сообщение. Если же клиента нет в списке или введено неправильное имя (пустое, слишком длинное или с запрещенными спецсимволами), то отправитель получает ответ Message Incorrect Name.

### 2.2.4 Кроссплатформенность

Для реализации совместимости кода на Linux и Windows необходимо учесть их различия в сокетах и потоках и написать код, который в зависимости от платформы компилировался бы правильно. Для этого будем использовать условную компиляцию с помощью команд препроцессора `#if-  
def/#elif/#endif`.

Вначале необходимо учесть различия в подключаемых заголовочных файлах

```
#ifdef _WIN32
#include <WinSock2.h>
```

```

#include <windows.h>
#elif __linux
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pthread.h>
#endif

```

В Windows для работы с сокетами необходимо подключить библиотеку (вариант для Visual Studio)  
`#pragma comment(lib, "WS2_32.lib")`

И настроить для работы(версия для winsock2):

```

WSADATA wsaData;
int errorcode = WSAStartup(MAKEWORD(2,2), &wsaData);

```

и после работы сбросить настройки

```

WSACleanup();

```

Другим отличием является различие в дескрипторе сокета, поэтому создадим универсальный тип для сокета

```

#ifdef _WIN32
typedef SOCKET mysocket;
#elif __linux
typedef int mysocket;
#endif

```

Для работы потоков также необходимо учесть различия.

У разных платформ функция точки входа в поток имеет разный тип возвращаемого значения:

```

typedef DWORD returnType; //win thread
typedef void * returnType; //linux thread

```

И разное описание этих функция (наличие модификатора WINAPI в windows)

```

#ifdef _WIN32
static returnType WINAPI Communicate(void * socket_param);
static returnType WINAPI TimerUserTimeOut(void * user_list_arg);
static returnType WINAPI AdvertMessage(void * user_list_arg);
#elif __linux
static returnType Communicate(void * socket_param);
static returnType TimerUserTimeOut(void * user_list_arg);
static returnType AdvertMessage(void * user_list_arg);
#endif

```

## 2.2.5 Описание основных классов сервера

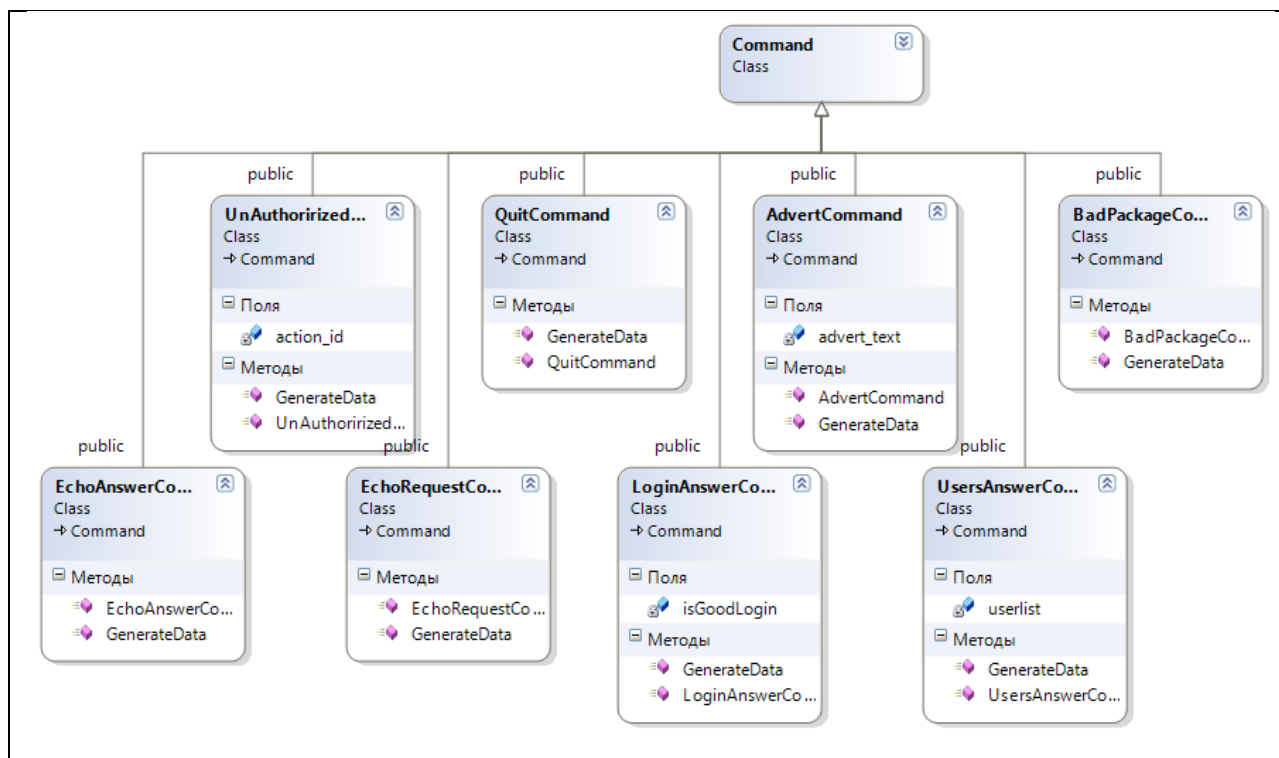


Рис.11 Список команд

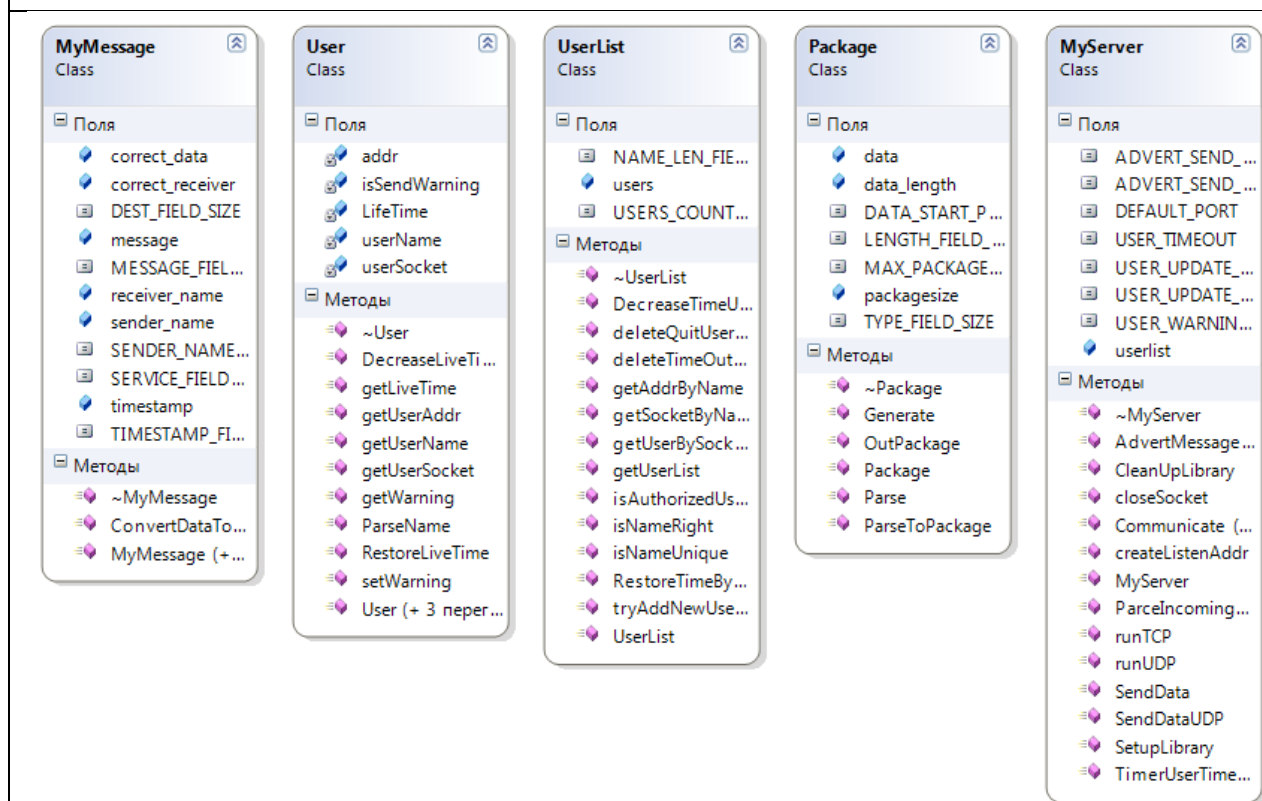
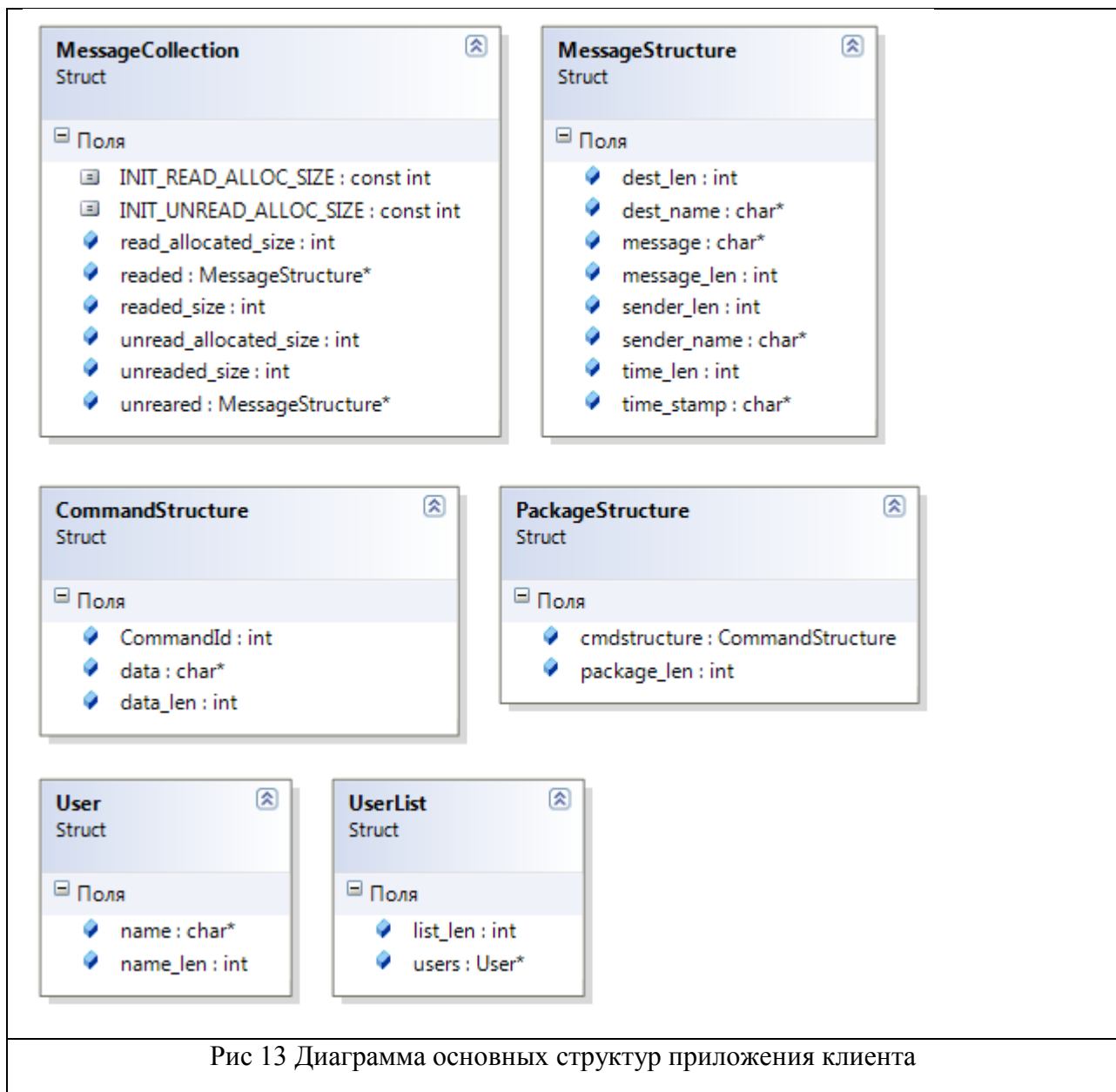


Рис.12 Основные классы

Класс	Описание
ServerApp AppColorTheme ApplicationOptions	Классы, описывающие приложение.  Задают параметры приложения (заголовок, цвет).
MyServer	Отвечает за работу сервера как процесса – подключения, отключения клиентов, прием-передачу данных, работа со списком пользователей
UserList User	Список пользователей.  Обеспечивает добавление и удаления из списка. Поиск по имени, сокету(TCP) или адресу/sockaddr (UDP).
MyLogger	Вывод сообщений на экран в заданной форме и цвете.
MyMessage	Работа с сообщениями.  Позволяет расшифровывать сообщения из входящих данных.
Command CommandFactory	Создание и расшифровка команд.
AdvertMessageList	Список рекламных сообщений
Package	Подготовка данных перед отправлением клиентам

## 2.2.6. Клиент - структуры





Классы (структуры)	Описание
MyClient	Отвечает за работу клиента как процесса – подключения, отключения от сервера, прием-передачу данных, работа со списком пользователей
ClientAppPC AppTheme	Классы, описывающие приложение. Задают параметры приложения (заголовок, цвет).
MyLogger	Вывод сообщений на экран в заданной форме и цвете.
MessageCollection MessageStructure	Структуры для хранения отдельного сообщения и списка сообщений.
CommandStructure	Обеспечивает генерацию команд и расшифровку полученной команды из сокета
PackageStructure	Подготавливает данные для отправки серверу в виде пакета
User UserList	Список пользователей. Обновления списка пользователей, расшифровка входящего пакета UsersAnswer

В терминах языка “с” нет понятия класс. В данном случае под классом подразумевается структура и/или набор методов, определяемых в одноименных файлах.

## Глава 3

### Реализация для работы по протоколу UDP

#### 3.1 Прикладной протокол

Аналогичен предыдущему протоколу.

#### 3.2 Архитектура приложения

Ввиду того, что в протоколе UDP не устанавливается логический канал связи между клиентом и сервером, то для обмена данными между несколькими клиентами и сервером нет необходимости использовать со стороны сервера несколько сокетов. Для определения источника полученной дейтаграммы серверный сокет может использовать поля структуры from вызова recvfrom.

Способ организации UDP-сервера приведён на рис. 14.

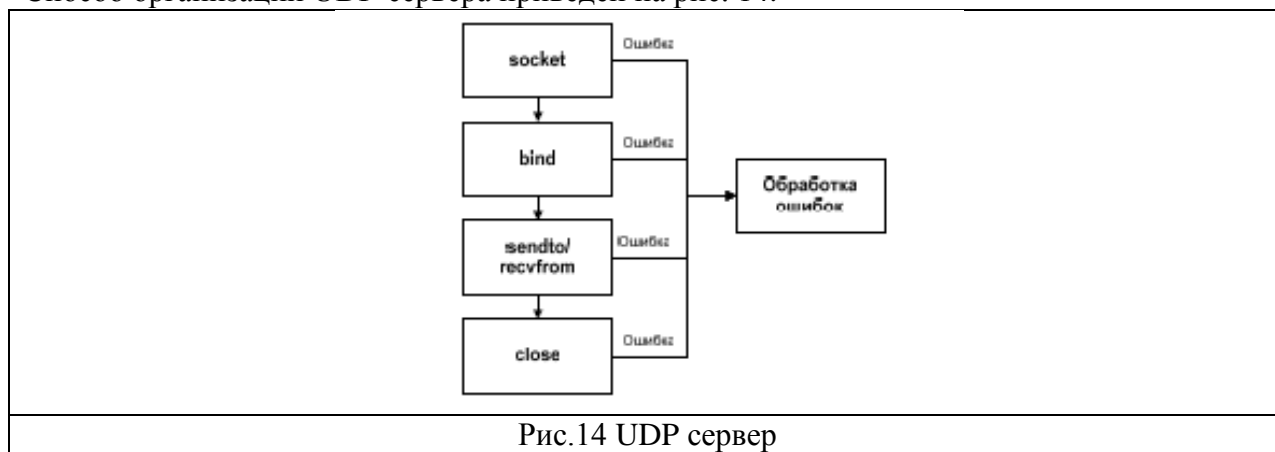


Рис.14 UDP сервер

Структура UDP-клиента ещё более простая, чем у TCP-клиента, так как нет необходимости создавать и разрывать соединение (рис.15).

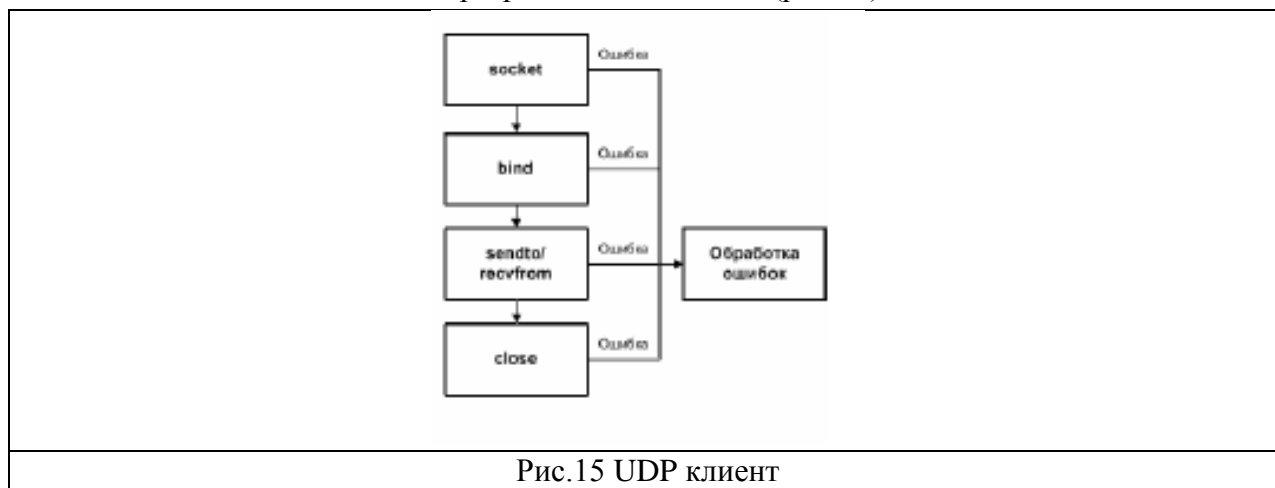


Рис.15 UDP клиент

UDP сервер не создает сокеты для каждого соединения, поэтому, в отличие от TCP сервера, не нужно создавать отдельные потоки для каждого подключившегося клиента.

Для UDP не нужно использовать отключение пользователя по таймауту, так как подключения нет.

Поэтому в сервере нет и потока, отвечающего за таймер отключения. Список пользователей на TCP хранит данные об имени и сокете пользователя, а UDP – об имени и адресе.

## Глава 4

### Выводы

Анализ выполненных заданий, сравнение удобства/эффективности/количества проблем при программировании TCP/UDP

#### 4.1 Реализация для TCP

##### Теоритические сведения

TCP — ориентированный на соединение протокол, что означает необходимость .рукопожатия. для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

##### Особенности протокола TCP

- Надёжность — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных тайм-аутов) разорванных соединений.
- Упорядоченность — если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.
- Тяжеловесность — TCP необходимо три пакета для установки сокет -соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.
- Поточность — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

##### Анализ

TCP протокол является надежным протоколом с установлением соединения, в связи с чем для TCP клиента помимо создания сокета, необходимо организовать соединение с помощью функции connect. TCP сервер должен содержать, как минимум, 2 сокета. Один сокет необходим для фиксации прихода запроса на соединение. После чего для каждого подключившегося клиента создается отдельный сокет. Это послужило затруднением при реализации многопоточной работы приложения, так как первоначально необходимо создать главный поток, слушающий определенный порт. Как только к серверу подключается клиент, создается новый сокет для этого клиента, после чего он передается в новую нить, содержащую необходимые действия клиента. Главная нить имеет возможность закрыть сокет любого подключенного клиента.

#### 4.2 Реализация для UDP

##### Теоритические сведения

UDP — более простой, основанный на сообщениях протокол без установления соединения. Протоколы такого типа не устанавливают выделенного соединения между двумя хостами. Связь достигается путем передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя.

##### Особенности протокола UDP

- Ненадёжный — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут.
- Неупорядоченность — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
- Легковесность — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.
- Датаграммы — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано.

## Анализ

Заметим, что структура протокол UDP более простая, чем TCP протокола. Во-первых, нет необходимости использования функции connect, то есть установления адреса и порта по умолчанию для протокола UDP. Однако тогда параметры удалённой стороны будем указывать или получать при каждом вызове операций записи или чтения с помощью функций sendto и recvfrom. Однако это создаёт проблемы при попытке реализации многопоточной работы сервера. Так как на каждого клиента не создается отдельного сокета и все клиенты используют 1 сокет, не получилось организовать параллельной работы клиентов.