**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<br>julia - логин |
| Login Answer - сервер отвечает клиенту о возможности регистрации в сети | 3 | 1<br><br>0 | 1 – логин принят<br>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<br><br>000 | 002 – число пользователей<br>05 – длина логина<br>julia - логин |
| Unauthorized action – сервер говорит пользователю,что он не авторизирован в сети | 7 | 08 | 08 – id запрещенной для анонимного пользователя команды |
| Message – клиент посылает сообщение другому клиенту через сервер | 8 | 05julia06gekkon<br>25Mon_Nov_10_01:22:24<br>2014_005hello | 05  - длина имени получателя<br>Julia -  получатель<br>06  - длина имени отправителя<br>Gekkon  - отправитель<br>25 – длина времени отправления<br>01:22:24 – время отправления<br>005 – длина сообщения |

| | | | |
|---|---|---|---|
| | | | Hello - сообщение |
| Message:<br>Incorrect name - сервер уведомляет клиента, что пользователь с запрашиваемым именем не найден | 9 | 05julia | 05 – длина имени пользователя<br><br>Julia – некорректное имя / отсутствующий пользователь |
| Message:<br>User offline - сервер уведомляет клиента, что пользователь с запрашиваемым именем оффлайн | 11 | 05julia | 05 – длина имени пользователя<br><br>Julia – пользователь offline |
| Message:<br>Success sended - сервер уведомляет клиента об успешной доставке сообщения другому клиенту | 12 | 1 | 1 - флаг окончания данных |
| Advert – рассылка сервером рекламы | 13 | 022https://www.google.ru/ | 022 – длина рекламы<br>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 REQUST. 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 соответвующее увеломление 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.

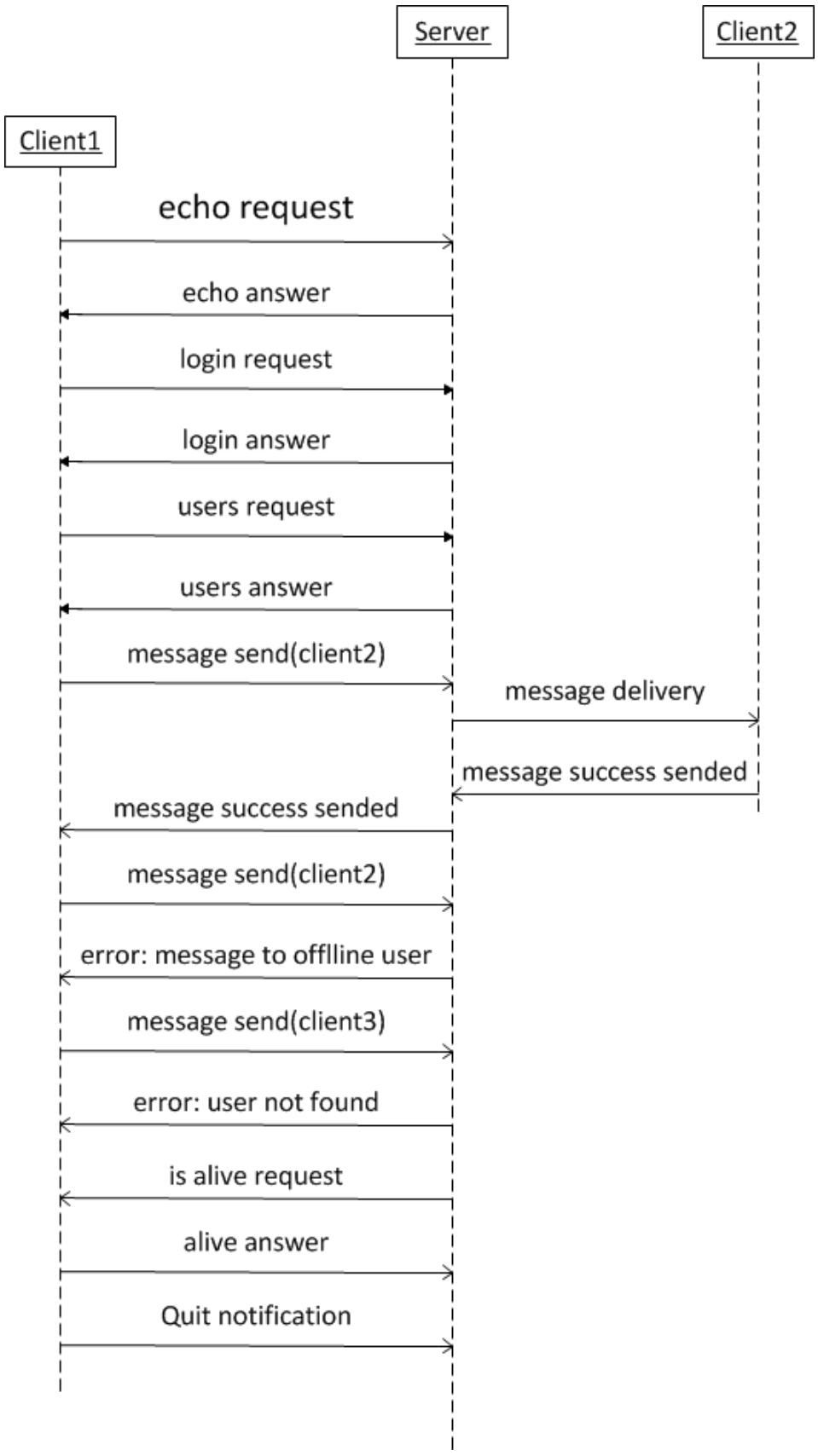


Рис.1 Sequence Diagram

## 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.

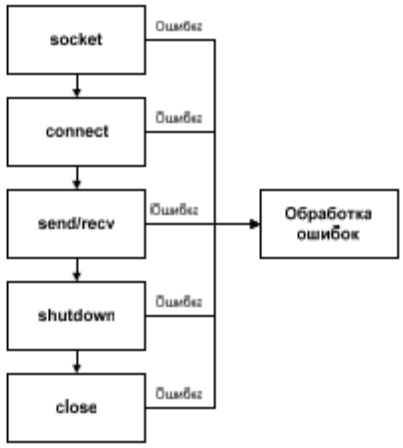


Рис.2 TCP клиент

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.

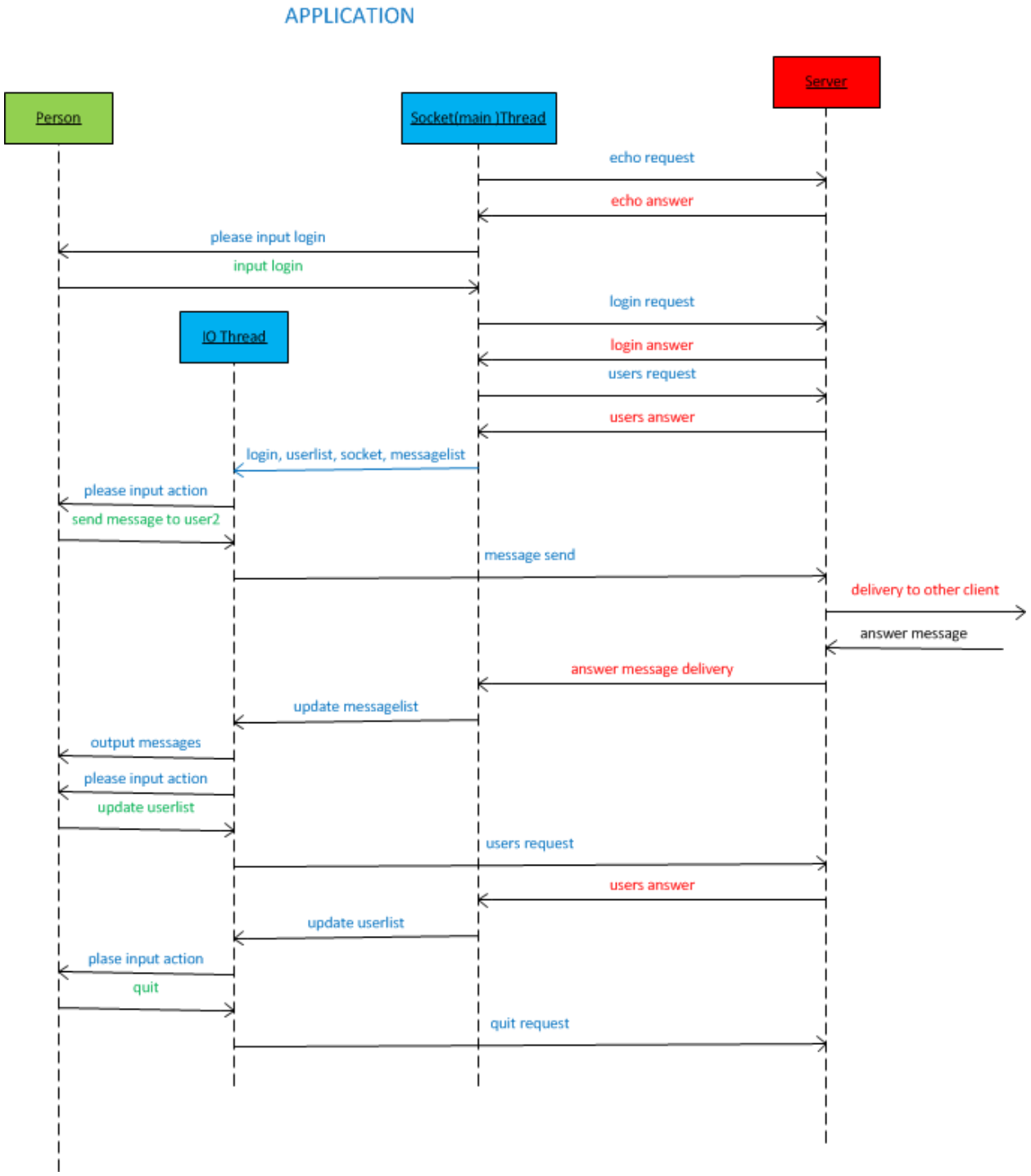


Рис.3 Работа клиента и сервера

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 рис5.

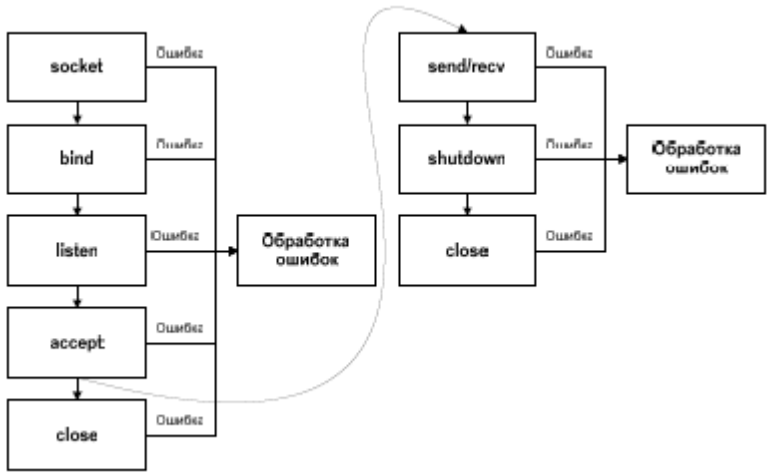


Рис.5 TCP сервер

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.

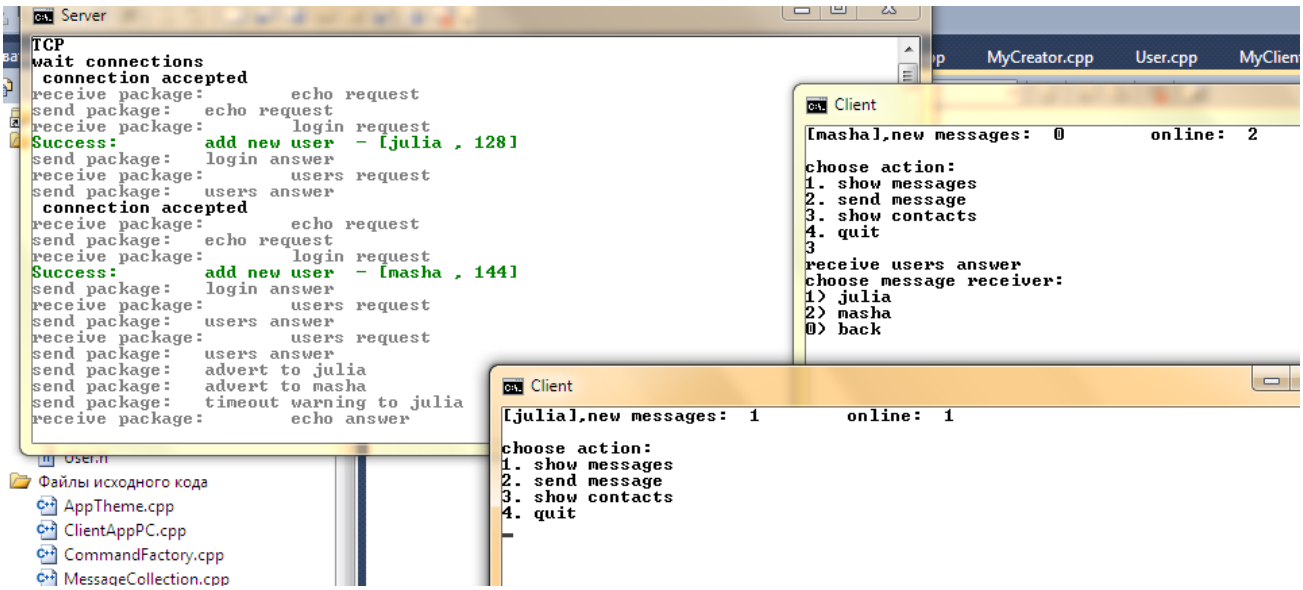
Server
TCP
wait connections
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 128]
send package: login answer
receive package: users request
send package: users answer
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [masha , 144]
send package: login answer
receive package: users request
send package: users answer
receive package: users request
send package: users answer
send package: advert to julia
send package: advert to masha
send package: timeout warning to julia
receive package: echo answer
Client
[masha],new messages: 0 online: 2
choose action:
1. show messages
2. send message
3. show contacts
4. quit
3
receive users answer
choose message receiver:
1) julia
2) masha
0) back
Client
[julia],new messages: 1 online: 1
choose action:
1. show messages
2. send message
3. show contacts
4. quit
MyCreator.cpp
User.cpp
Файлы исходного кода
AppTheme.cpp
ClientAppPC.cpp
CommandFactory.cpp
MessageCollection.cpp

Рис.6

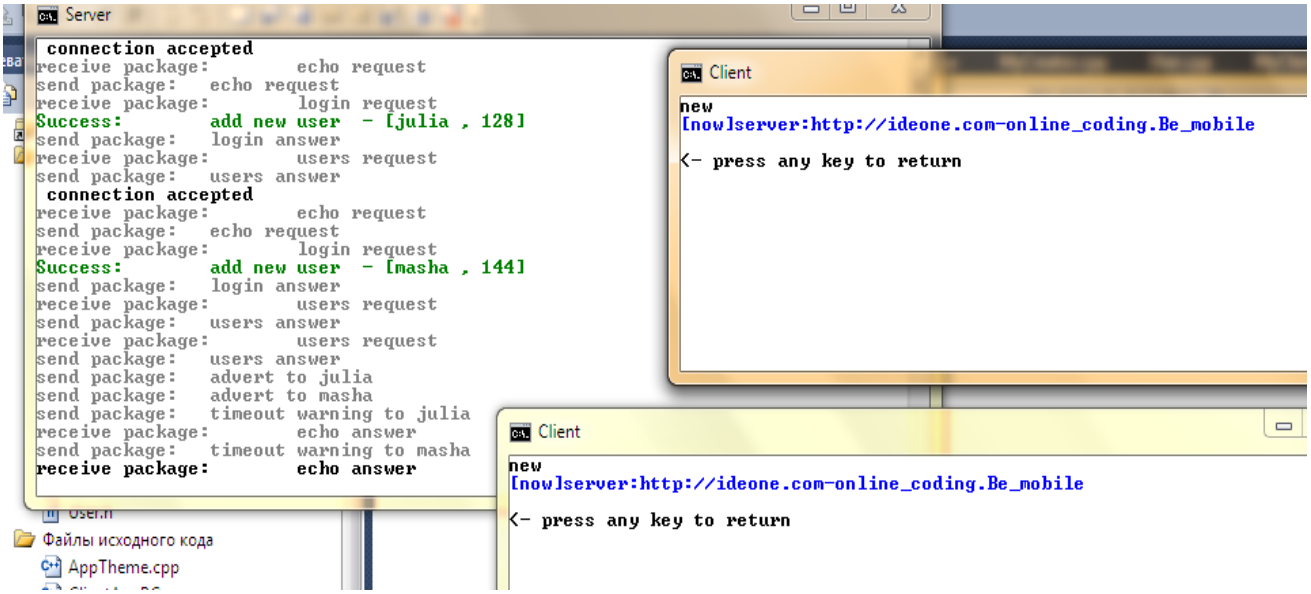
Server
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 128]
send package: login answer
receive package: users request
send package: users answer
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [masha , 144]
send package: login answer
receive package: users request
send package: users answer
receive package: users request
send package: users answer
send package: advert to julia
send package: advert to masha
send package: timeout warning to julia
receive package: echo answer
send package: timeout warning to masha
receive package: echo answer
Client
new
[nowlserver:http://ideone.com-online_coding.Be_mobile
<- press any key to return
Client
new
[nowlserver:http://ideone.com-online_coding.Be_mobile
<- press any key to return
Файлы исходного кода
AppTheme.cpp

Рис.7

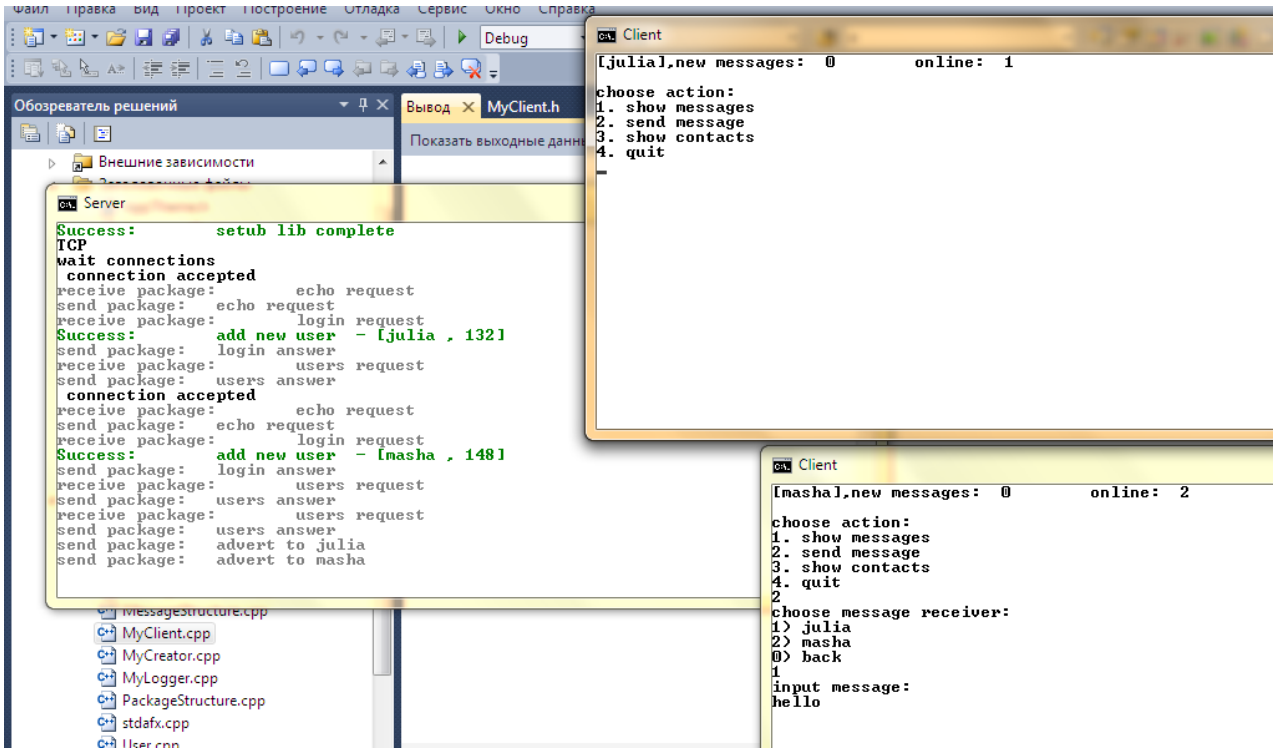
Client
[julia],new messages: 0 online: 1
choose action:
1. show messages
2. send message
3. show contacts
4. quit
Server
Success: setub lib complete
TCP
wait connections
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 132]
send package: login answer
receive package: users request
send package: users answer
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [masha , 148]
send package: login answer
receive package: users request
send package: users answer
receive package: users request
send package: users answer
send package: advert to julia
send package: advert to masha
Client
[masha],new messages: 0 online: 2
choose action:
1. show messages
2. send message
3. show contacts
4. quit
2
choose message receiver:
1) julia
2) masha
0) back
1
input message:
hello
Обозреватель решений
Внешние зависимости
MyClient.cpp
MyCreator.cpp
MyLogger.cpp
PackageStructure.cpp
stdafx.cpp
Вывод
MyClient.h
Debug

Рис.8

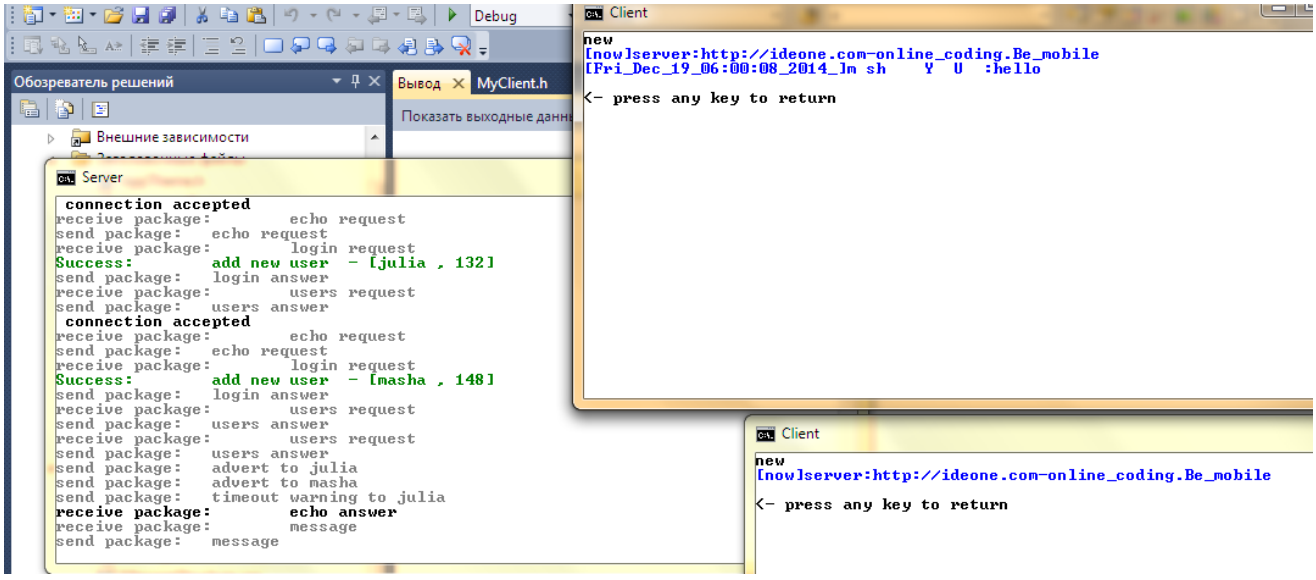
Client
new
[nowlserver:http://ideone.com-online_coding.Be_mobile
[Fri_Dec_19_06:00:08_2014_lm sh Y U :hello
<- press any key to return
Server
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 132]
send package: login answer
receive package: users request
send package: users answer
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [masha , 148]
send package: login answer
receive package: users request
send package: users answer
receive package: users request
send package: users answer
send package: advert to julia
send package: advert to masha
send package: timeout warning to julia
receive package: echo answer
receive package: message
send package: message
Client
new
[nowlserver:http://ideone.com-online_coding.Be_mobile
<- press any key to return
Обозреватель решений
Внешние зависимости
Вывод
MyClient.h
Debug

Рис.9

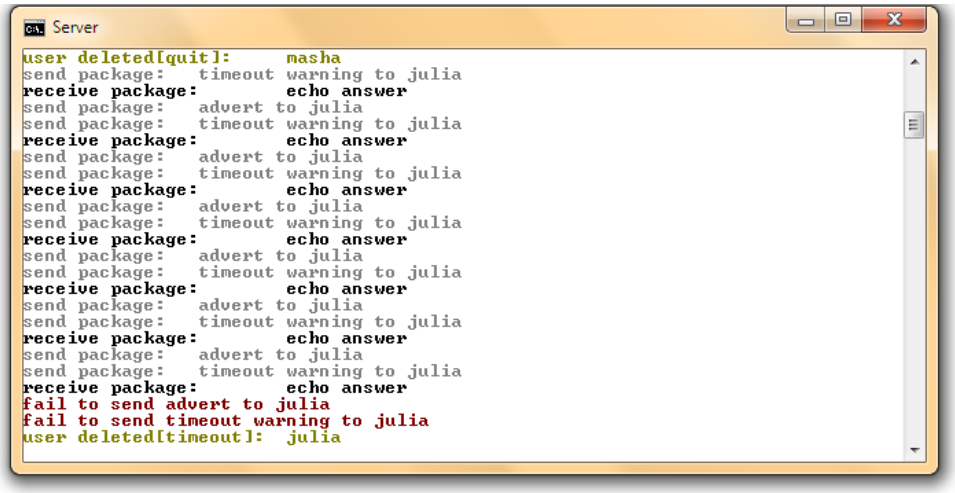


Рис.10

### 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

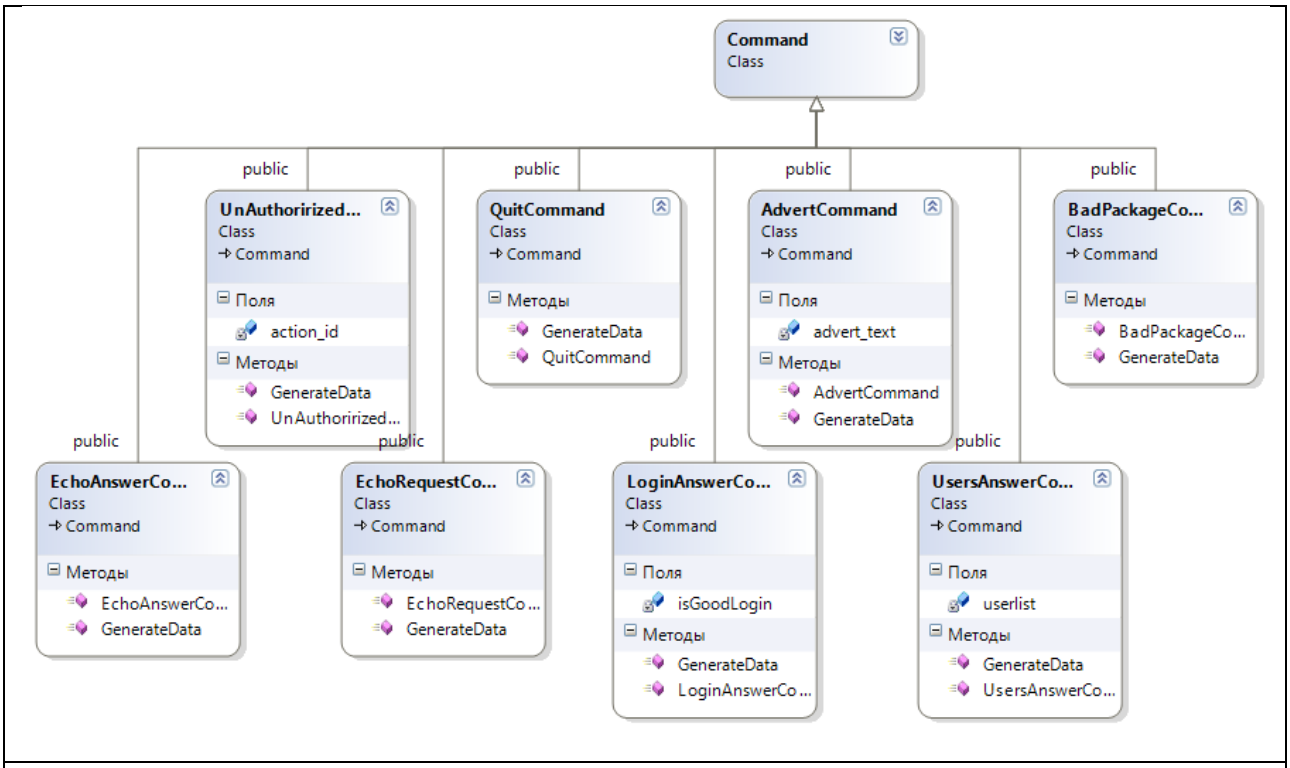


Рис.11 Список команд

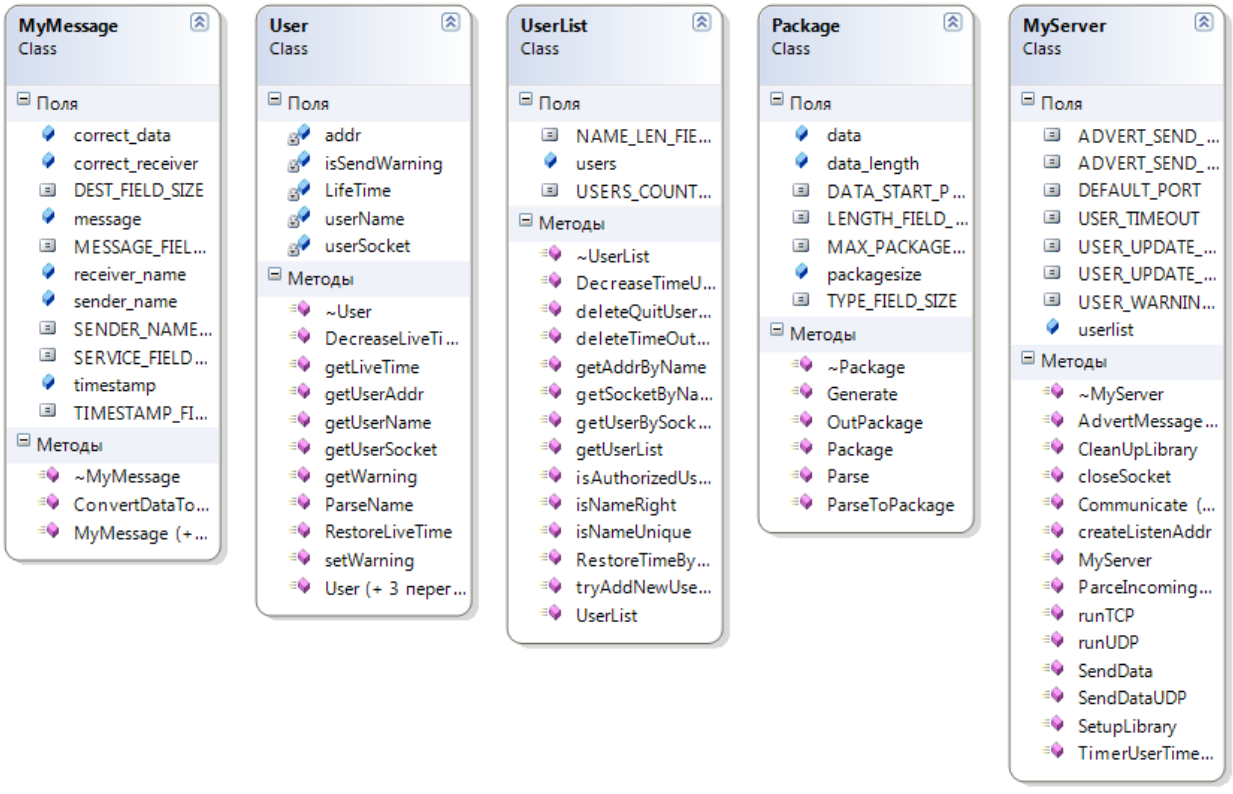


Рис.12 Основные классы

| Класс | Описание |
|---|---|
| ServerApp<br><br>AppColorTheme<br><br>ApplicationOptions | Классы, описывающие приложение.<br><br>Задают параметры приложения (заголовок, цвет). |
| MyServer | Отвечает за работу сервера как процесса – подключения, отключения клиентов, прием-передачу данных, работа со списком пользователей |
| UserList<br><br>User | Список пользователей.<br><br>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.<br><br>Позволяет расшифровывать сообщения из входящих данных. |
| Command<br><br>CommandFactory | Создание и расшифровка команд. |
| AdvertMessageList | Список рекламных сообщений |
| Package | Подготовка данных перед отправлением клиентам |

2.2.6. **Клиент - структуры**

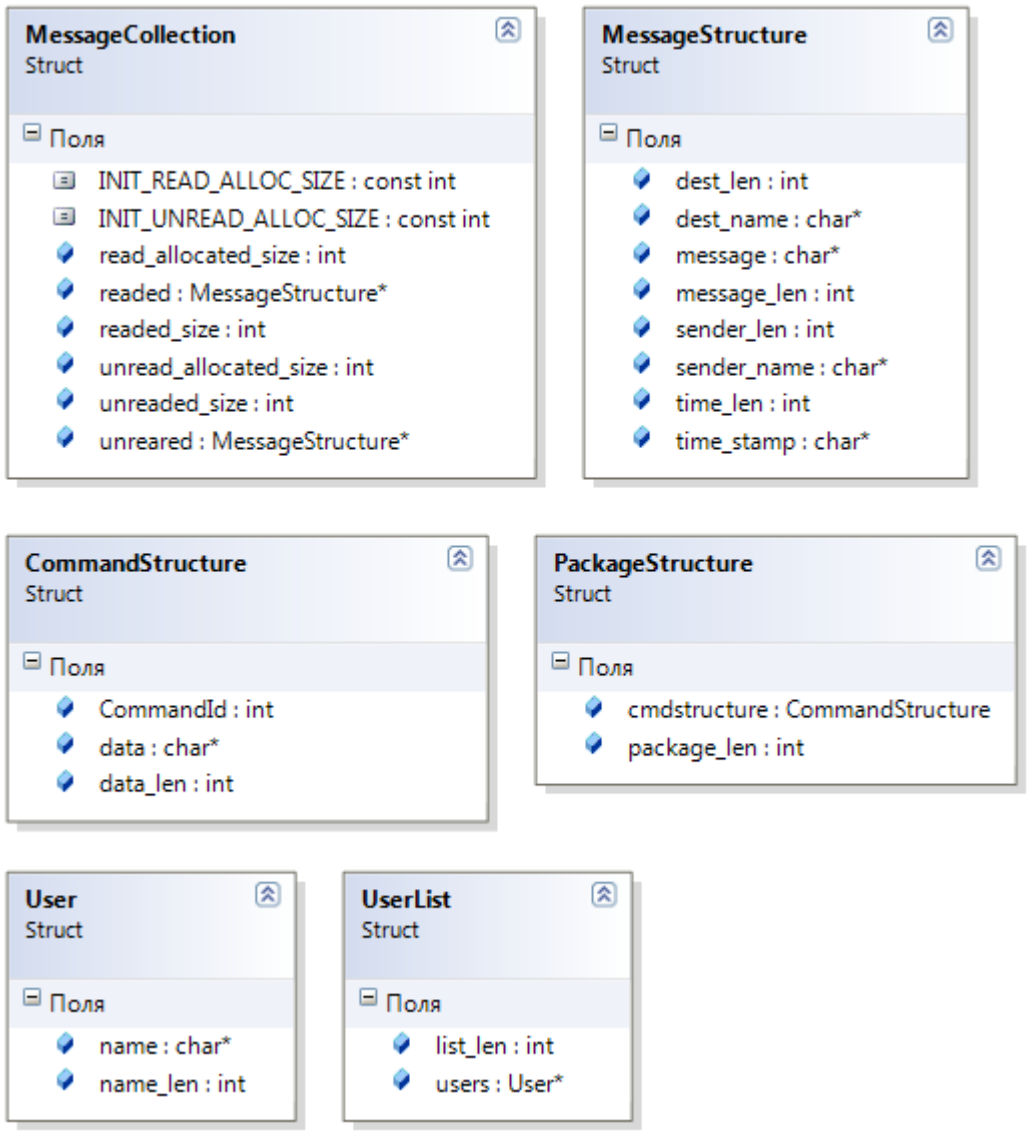


Рис 13 Диаграмма основных структур приложения клиента

| Классы (структуры) | Описание |
|---|---|
| MyClient | Отвечает за работу клиента как процесса – подключения, отключения от сервера, прием-передачу данных, работа со списком пользователей |
| ClientAppPC<br><br>AppTheme | Классы, описывающие приложение.<br><br>Задают параметры приложения (заголовок, цвет). |
| MyLogger | Вывод сообщений на экран в заданной форме и цвете. |
| MessageCollection<br><br>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<br><br>UserList | Список пользователей.<br><br>Обновления списка пользователей, расшифровка входящего пакета UsersAnswer |

В терминах языка “c” 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

## 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.

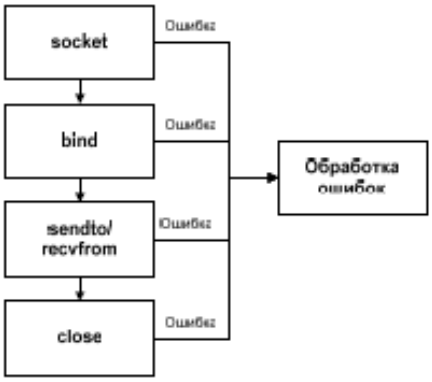


Рис.14 UDP сервер

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

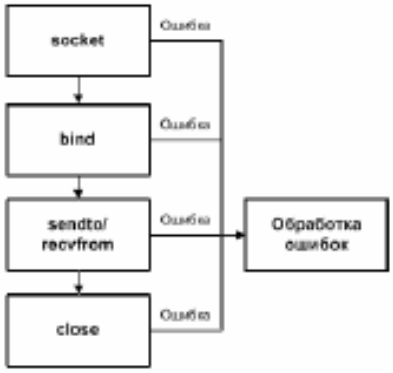


Рис.15 UDP клиент

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.