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

Требования к производительности, надежности и т.п. 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. Серверное приложение должно работать одновременно со многими клиентами. Также приложение должно выдвать клиенту список его возможных дальнейших операций(чтение сообщений, отправка сообщения). Разработанное клиентское приложение должно не выходить из строя при отправке и получении сообщений.

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, равно 6 байт (3 байта на длину пакета, 2 байта на код команды и 1 байт данных), так как больше нам и не требуется. В остальных случаях длина поля данных ограничена только максимальным размером пакета.

1.3 Накладываемые ограничения

• Ограничение на длину пакетов: максимальный размер пакета 512 символов (оптимальный размер пакетов для работы по протоколам TCP/UDP). Поясним подробнее:
MSS (Maximum segment size) является параметром протокола TCP и определяет максимальный размер полезного блока данных в байтах для TCP пакета (сегмента). Таким образом, этот параметр не учитывает длину заголовков TCP и IP. Для установления корректной TCP сессии с удалённым хостом должно соблюдаться следующее условие: $MSS + TCP + IP \leq MTU$. Таким образом, максимальный размер MSS = MTU — размер заголовка IPv4 — размер заголовка TCP. Так каждый хост на IPv4 требует доступности для MSS последних 536 октетов (= 576 — 20 — 20) а на IPv6 — 1220 октетов (= 1280 —40 — 20). Мы выбрали значение, являющееся степенью 2 - 512. Это является оптимальным размером пакетов.
UDP передает пакеты отдельными датаграммами (заголовок и данные) в байтах. Таким образом, если длина пакета с UDP будет превышать MTU (для Ethernet по умолчанию 1500 байт), то отправка такого пакета может вызвать его фрагментацию, что может привести к тому, что он вообще не сможет быть доставлен, если промежуточные маршрутизаторы или конечный хост не будут поддерживать фрагментированные IP пакеты. Также в RFC791 указывается минимальная длина IP пакета не менее 512 байт и рекомендуется отправлять IP пакеты большего размера только в том случае если вы уверены, что принимающая сторона может принять пакеты такого размера. Следовательно, чтобы избежать фрагментации UDP пакетов (и возможной их потери), размер данных в UDP не должен превышать: MTU — (Max IP Header Size) — (UDP Header Size) = 1500 — 60 — 8 = 1432 байт. Для надежности, зададим максимальную размерность отправляемых пакетов 512 байт.

- Имя пользователя не должно превышать 32 символов, сообщение не должно превышать 507 байт.

•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 авторизирован 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	022https://www.google.ru/	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 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 предлагается 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.

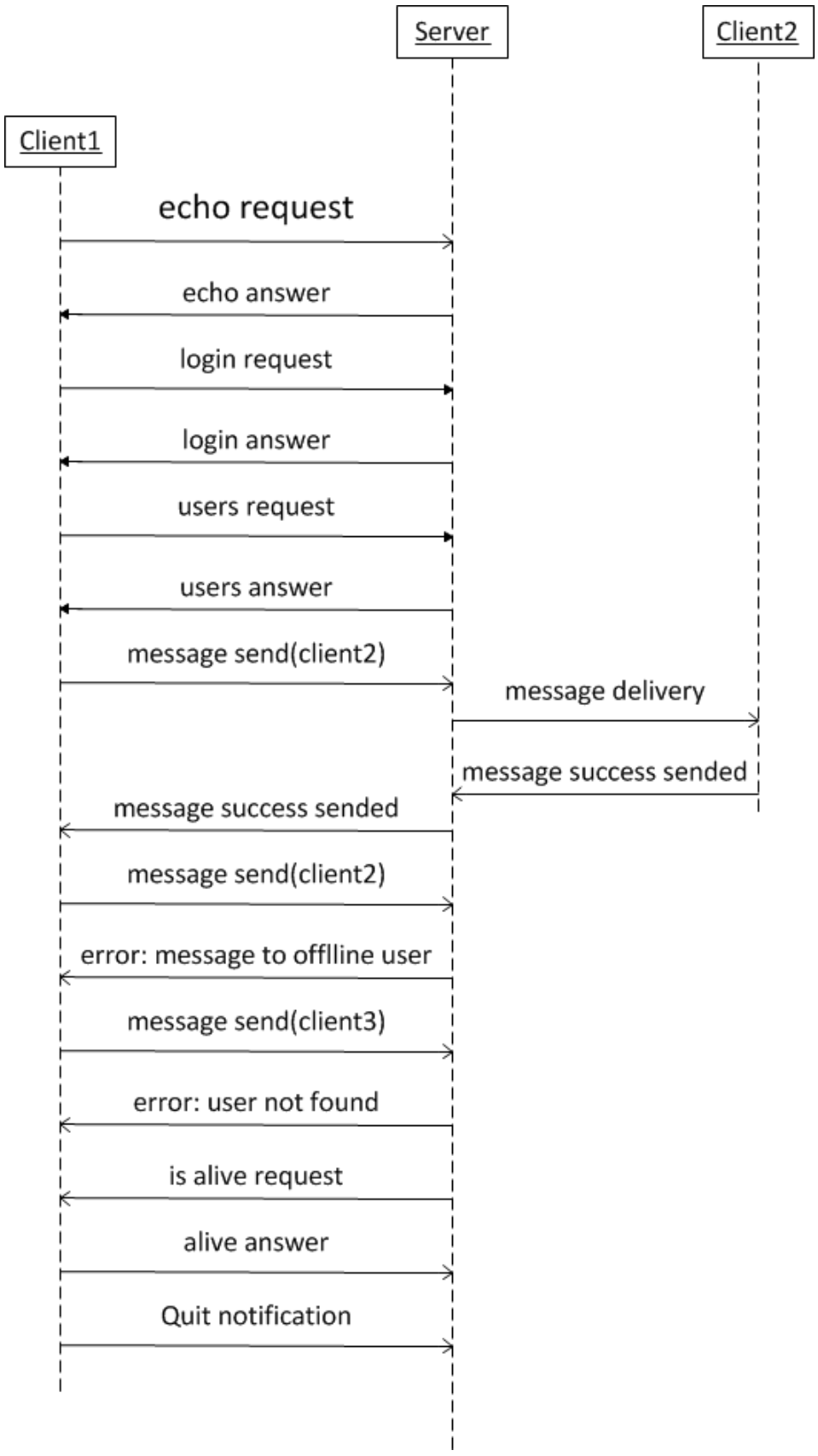

Рис.1 Sequence Diagram

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.

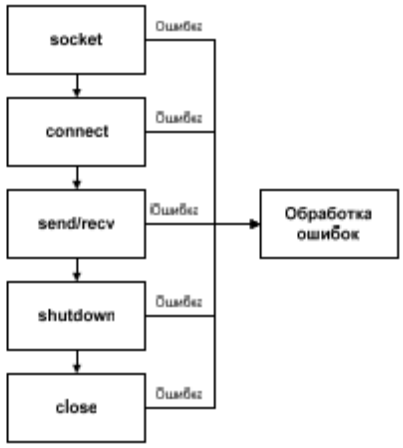

Рис.2 TCP клиент

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.

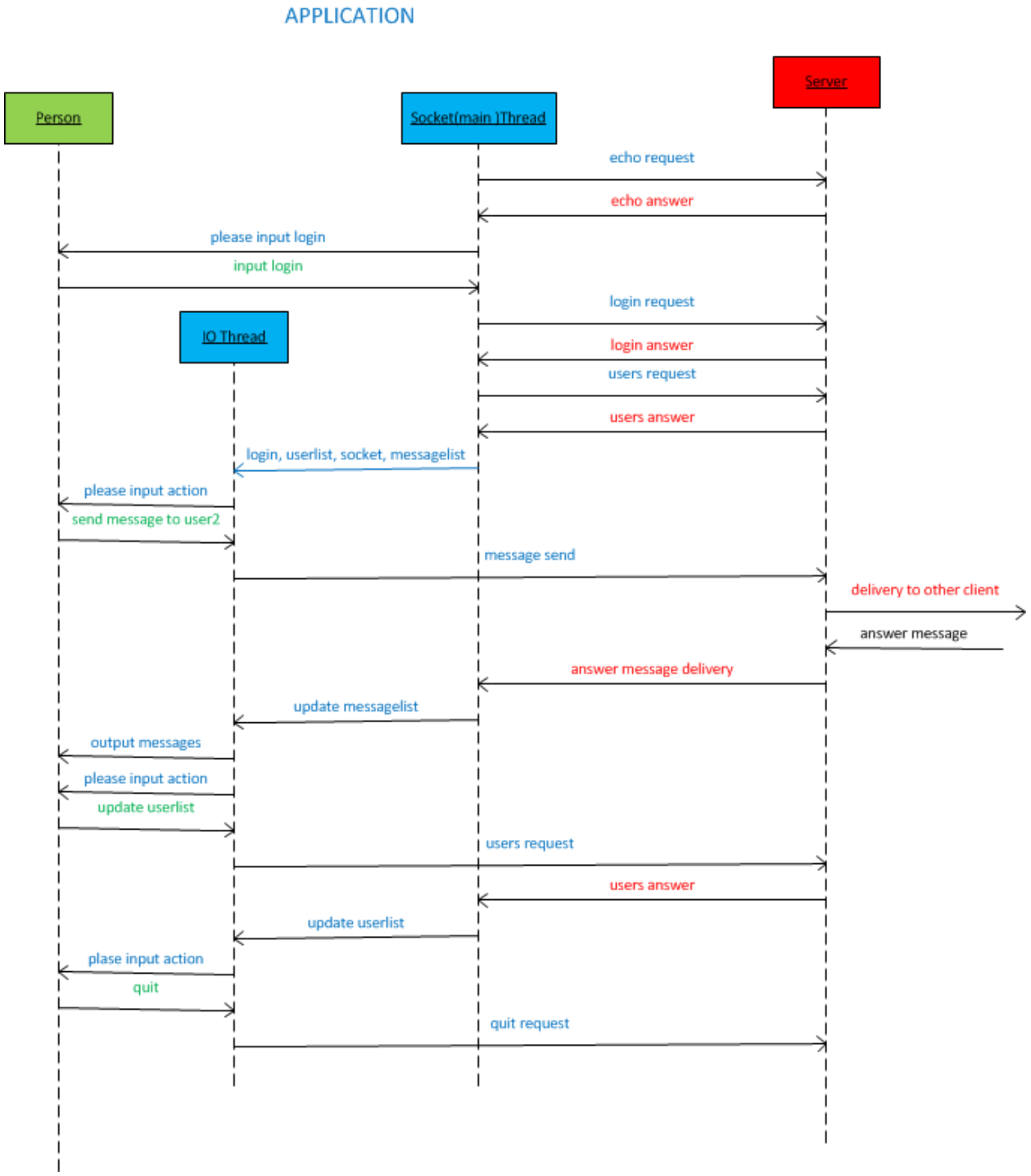

Рис.3 Работа клиента и сервера

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.

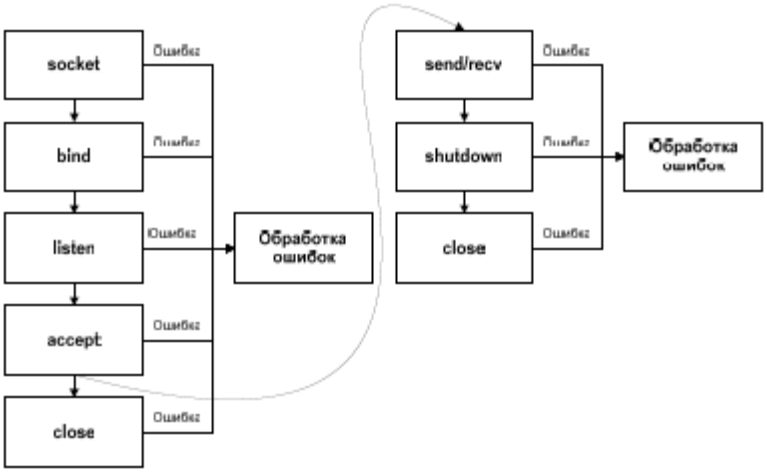

Рис.5 TCP сервер

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.

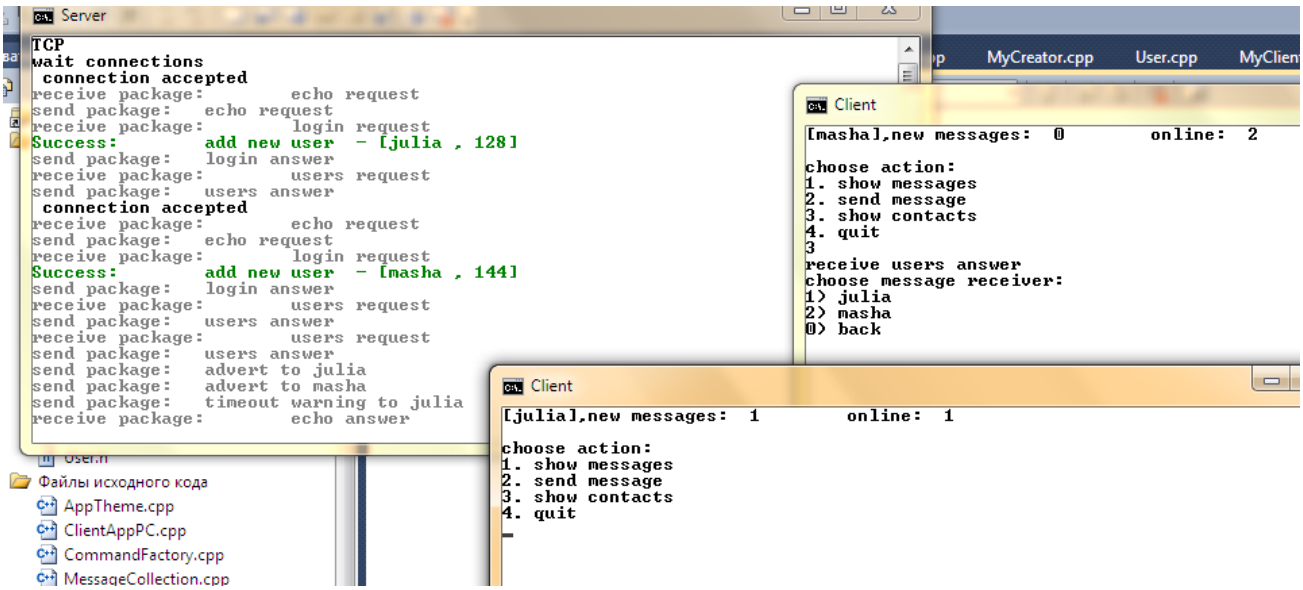
Server
TCP
wait connections
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 128]
send package: login answer
receive package: users request
send package: users answer
connection accepted
receive package: echo 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

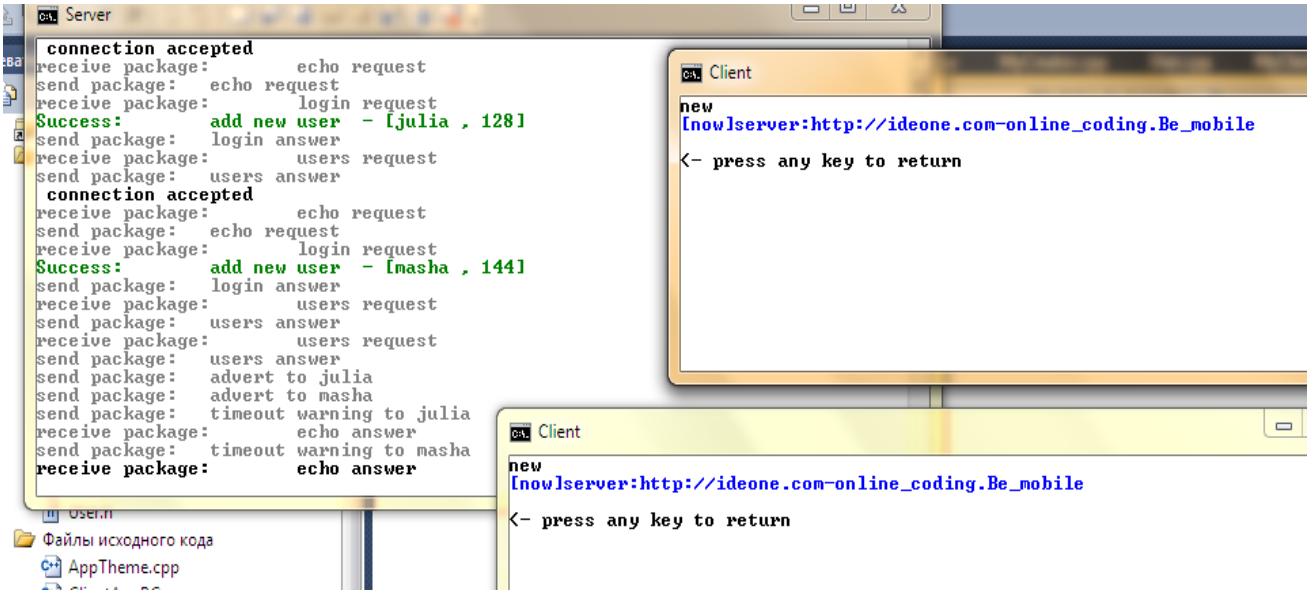
Server
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 128]
send package: login answer
receive package: users request
send package: users answer
connection accepted
receive package: echo request
send package: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

Рис.7

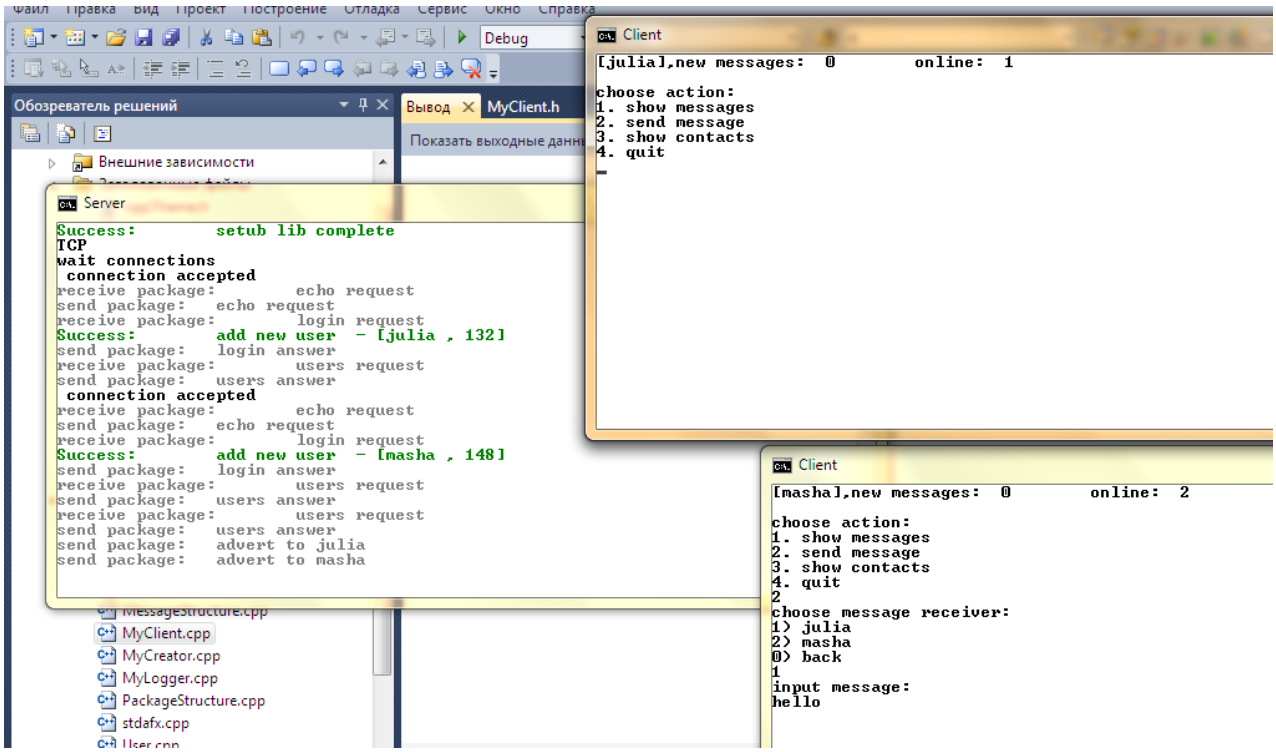
Client
[julia],new messages: 0 online: 1
choose action:
1. show messages
2. send message
3. show contacts
4. quit
Server
Success: setub lib complete
TCP
wait connections
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [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

Рис.8

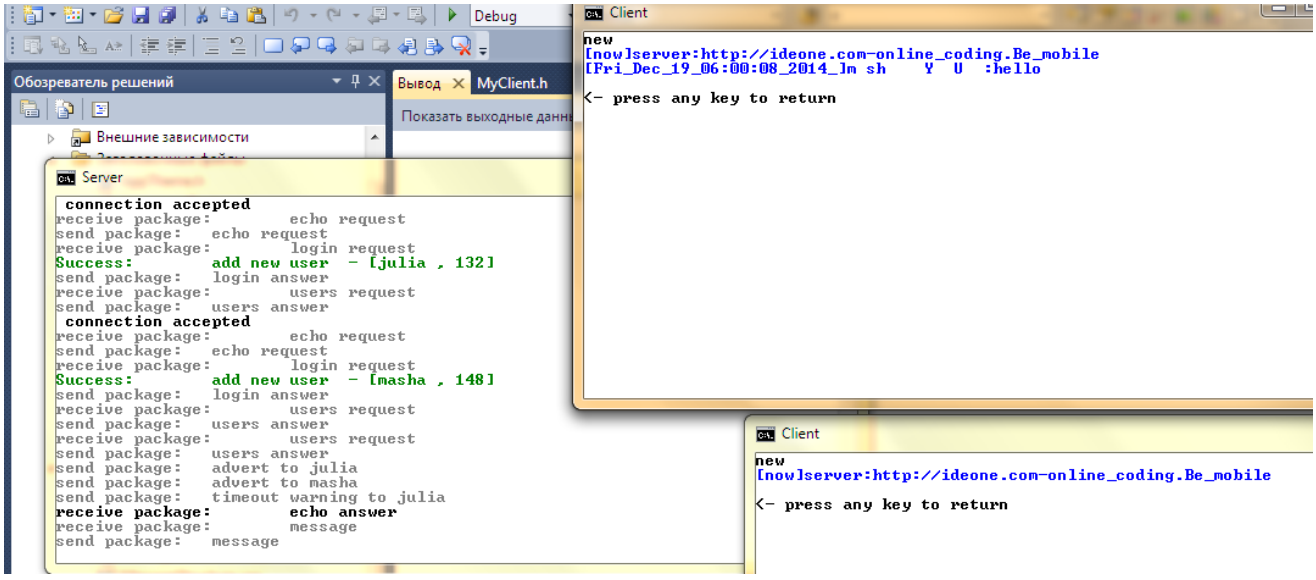
Client
new
[nowlserver:http://ideone.com-online_coding.Be_mobile
[Fri_Dec_19_06:00:08_2014_lm sh Y U :hello
<- press any key to return
Server
connection accepted
receive package: echo request
send package: echo request
receive package: login request
Success: add new user - [julia , 132]
send package: 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

Рис.9

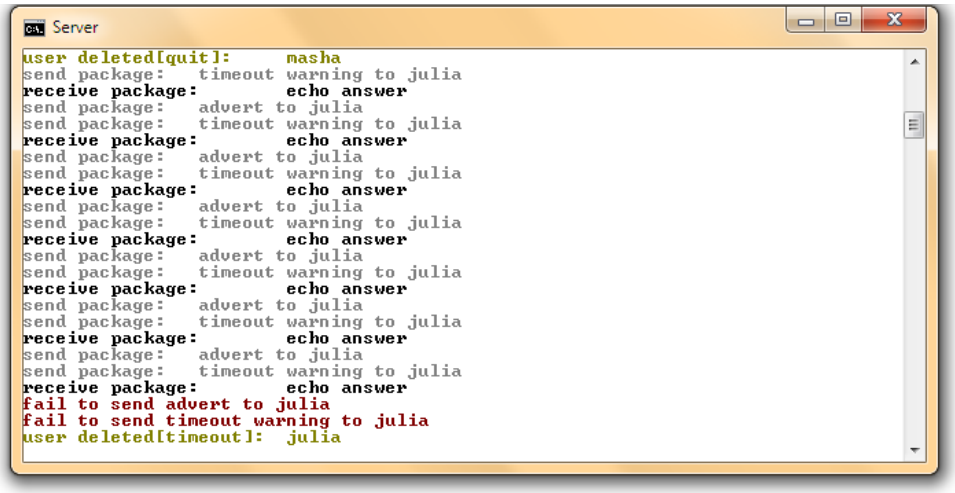

Рис.10

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

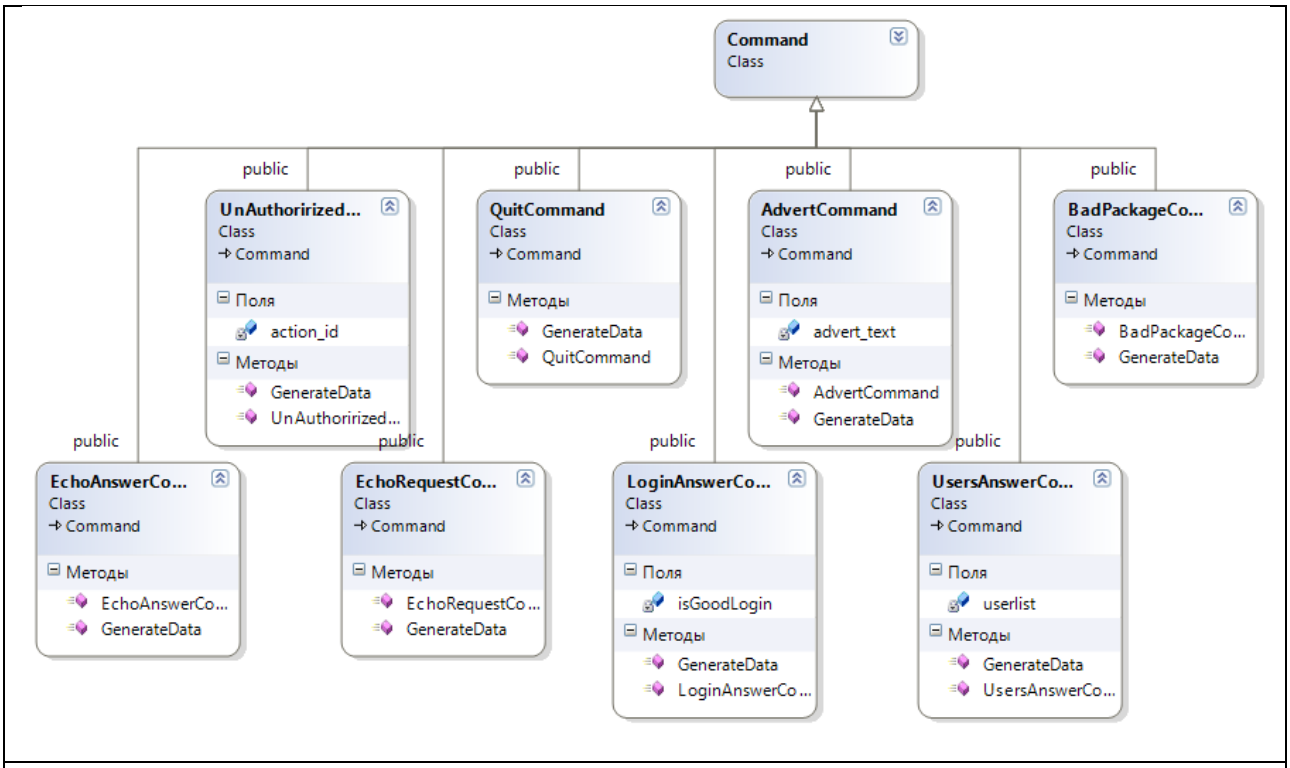

Рис.11 Список команд

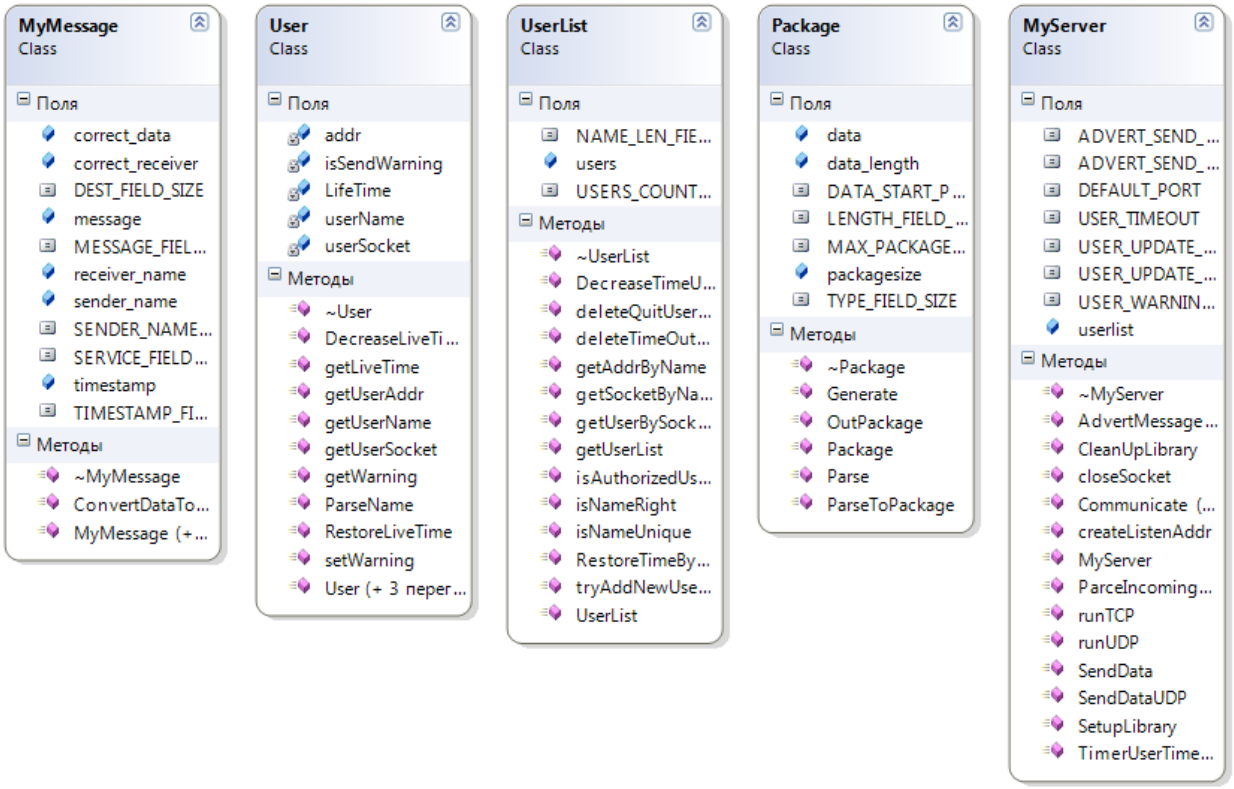

Рис.12 Основные классы

Класс	Описание
ServerApp 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**

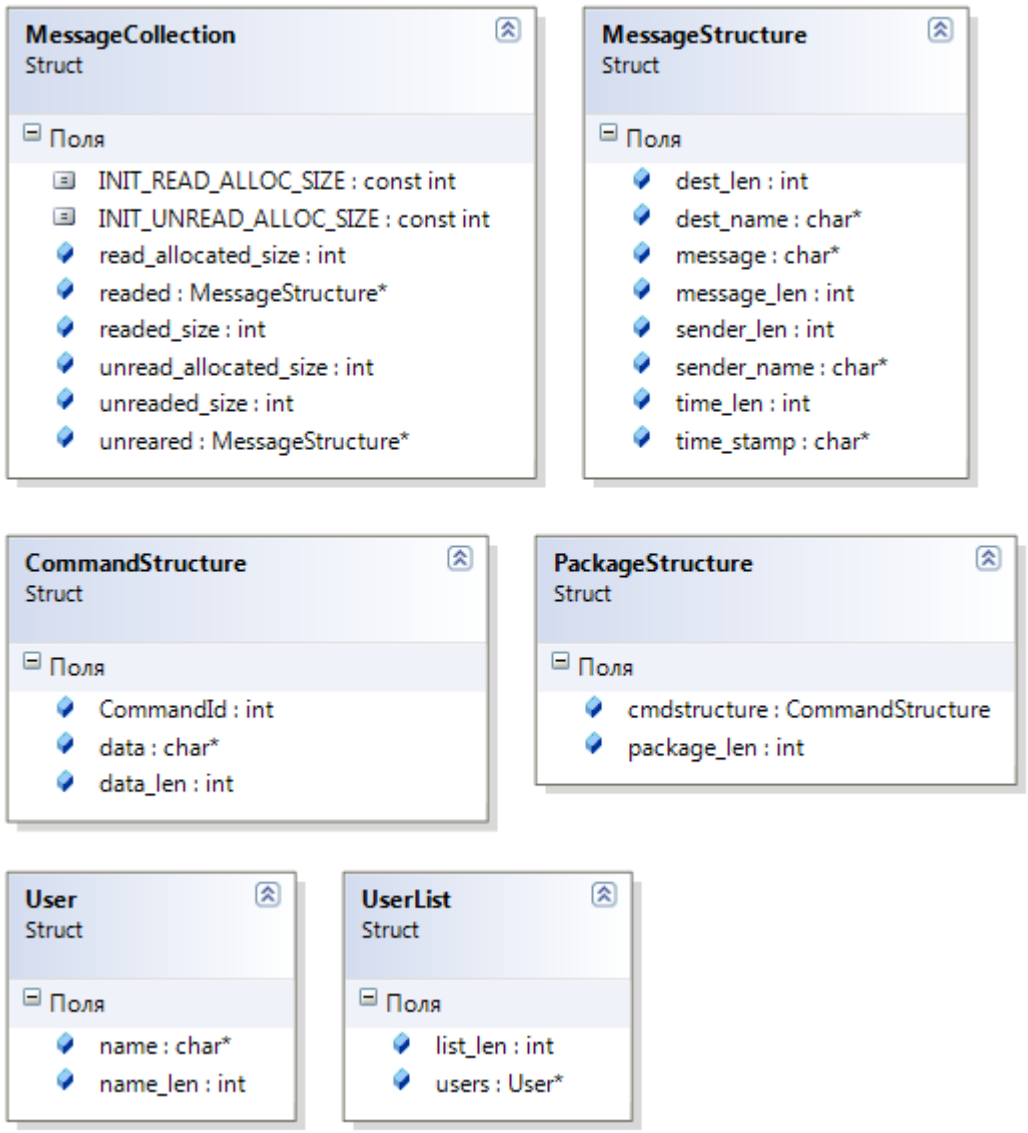

Рис 13 Диаграмма основных структур приложения клиента

Классы (структуры)	Описание
MyClient	Отвечает за работу клиента как процесса – подключения, отключения от сервера, прием-передачу данных, работа со списком пользователей
ClientAppPC 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 “c” нет понятия класс. В данном случае под классом подразумевается структура и/или набор методов , определяемых в одноименных файлах.

2.3 Тестирование

Для тестирования приложения сначала запустим сервер и несколько клиентов на одной ОС (Linux, Windows). Результаты работы приведены на рисунках 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 посы-

лаемый timeout warning (предупреждение об отключении), самостоятельно отключает данного пользователя и удаляет его из списка пользователей.
Приложения работают корректно. Далее запустим приложения в разных ОС, при этом корректность их работы сохраняется. Также были проведены тесты на введение пользователем некорректных данных (длина логина больше 32 символов или слишком длинное сообщение другому пользователю – больше 507 байт), тогда пользователь получал сообщения об ошибках.

Глава 3
Реализация для работы по протоколу UDP

3.1 Прикладной протокол
Размер пакетов 512 байт, выбор размера проанализирован выше в разделе 1.3. Заметим, что данное задание позволяет это сделать, потому что приложение по обмену мгновенными сообщениями не предполагает передачу слишком больших пакетов и размера в 512 байт будет достаточно для корректной работы протокола.
В связи с этим, изменения в реализации протокола UDP по сравнению с протоколом TCP будут незначительны, и общей архитектуры они не заденут. Подробности в разделе 2.1.

3.2 Архитектура приложения

Особенности архитектуры соответсвуют архитектуре приложения для TCP протокола, раздел 2.2.

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.

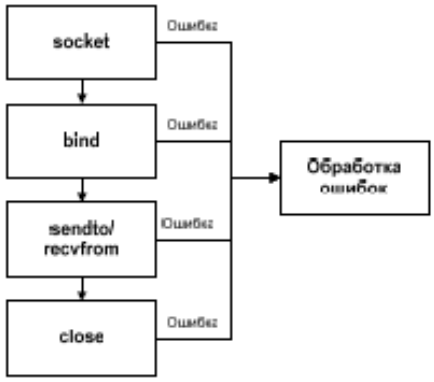

Рис.14 UDP сервер

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

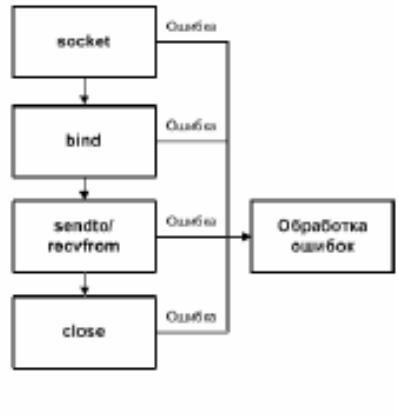

Рис.15 UDP клиент

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.

3.3 Тестирование

См. раздел 2.3

Глава 4
Выводы

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

4.1 Реализация для TCP
Теоретические 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.
· Потоковость — 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 фиксирования 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
Теоретические 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, таймаут.
· 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 TCPпротокола.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 попытки 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.

Приложения

Описание среды разработки
Использованные версии ОС:
Windows 7
Debian GNU/Linux 7.6 (wheezy)

Листинги
Общие для TCP/UDP,Windows/Linux заголовочные файлы и файлы исходных кодов

Код для сервера:

Userlist.h

```
#pragma once
#include <vector>
#include "User.h"
//#include "MyServer.h"
class UserList
{
public:
	static const int USERS_COUNT_FIELD_SIZE = 3;
	static const int NAME_LEN_FIELD_SIZE = 2;
public:
	std::vector<User> users;
public:
	UserList();
	~UserList();
    mysocket getSocketByName(string name);
	sockaddr_in getAddrByName(string name);
    User* getUserBySocketDesc(mysocket s);
    bool RestoreTimeBySocket(mysocket s);
    bool isNameUnique(string name);
    bool isNameRight(string name);
	bool tryAddNewUser(User * user);
	bool tryAddNewUser(string name,mysocket s);
    void DecreaseTimeUsers(int decValue_ms);
	bool isAuthorizedUser(mysocket s);
	bool isAuthorizedUser(sockaddr_in addr);

	bool deleteTimeOutedUsers();
	bool deleteQuitUser(mysocket s);
	bool deleteQuitUser(sockaddr_in addr);
	string getUserList();

	bool CompareAddr(sockaddr_in addr1, sockaddr_in addr2);
};
```

Userlist.cpp

```
#include "stdafx.h"
#include "UserList.h"
#include "string.h"
#include "StringsHelper.h"
#include "MyLogger.h"
#include "MyServer.h"
UserList::UserList(void)
```

```
{
}
UserList::~UserList(void)
{
}
void UserList::DecreaseTimeUsers(int decValue_ms) //decrement time for all users
{
	for(unsigned int i=0;i<this->users.size();i++)
	{
		users[i].DecreaseLiveTime(decValue_ms);
	}
}
mysocket UserList::getSocketByName(string name)
{
	for(unsigned int i=0;i<this->users.size();i++)
	{
		User user = users[i];
		if(name == user.getUserName())
		{
			return user.getUserSocket();
		}
	}
	return INVALID_SOCKET;
}
sockaddr_in  UserList::getAddrByName(string name)
{
	sockaddr_in addr;
	for(unsigned int i=0;i<this->users.size();i++)
	{
		User user = users[i];
		if(name == user.getUserName())
		{
			return user.getUserAddr();
		}
	}
	return addr;
}
bool UserList::isNameUnique(string name)
{
	for(unsigned int i=0;i<this->users.size();i++)
	{
		User user = users[i];
		if (name == user.getUserName())
		{
			return false;
		}
	}
	return true;
}
bool UserList::isNameRight(string name)
{
	for(int i = 0; i< name.size();i++)
	{
		int i_symb  = (int)(name[i]);
		if(i_symb < 0 || i_symb > 255)
		{
			return false;
		}
	}
	if(name.length() <= 0)
	{
		return false;
	}
```

```
        if(name.length() > 32)
        {
                return false;
        }
        return true;
}
User* UserList::getUserBySocketDesc(mysocket s)
{
        for(unsigned int i=0;i<this->users.size();i++)
        {
                if(users[i].getUserSocket() == s)
                {
                        return &users[i];
                }
        }
        return NULL;
}
bool UserList::RestoreTimeBySocket(mysocket s)
{
        User* user = getUserBySocketDesc(s);
        if(user != NULL)
        {
                user->RestoreLiveTime();
                return true;
        }
        return false;
}
bool UserList::tryAddNewUser(User* user)
{
        this->users.push_back(*user);
        return true;
}
bool UserList::tryAddNewUser(string name,mysocket s)
{
        User *user = new User(name,s);
        tryAddNewUser(user);
        return true;
}
string UserList::getUserList()
{
        string userlist = "";
        char * c_users_count = StringsHelper::getCleanBuffer(USERS_COUNT_FIELD_SIZE);
        int users_count = users.size();
        sprintf(c_users_count,"%3d",users_count);
        userlist +=c_users_count;
        for(int i=0;i<userlist.size();i++)
        {
                if(userlist[i] == ' ')
                {
                        userlist[i] = '0';
                }
        }
        char * c_name_len = StringsHelper::getCleanBuffer(NAME_LEN_FIELD_SIZE);
        for(int i=0;i<users.size();i++)
        {
                int name_len = users[i].getUserName().size();
                sprintf(c_name_len,"%2d",name_len);
                c_name_len = StringsHelper::replaceEmptySymbols(c_name_len,2);
                userlist+=c_name_len;
                userlist+=users[i].getUserName();
        }
        //userlist+= '\0';
        return userlist;
```

```
}
bool UserList::isAuthorizedUser(mysocket s)
{
        for(unsigned int i=0;i<this->users.size();i++)
        {
                if(users[i].getUserSocket() == s)
                {
                        return true;
                }
        }
        return false;
}
bool UserList::isAuthorizedUser(sockaddr_in addr)
{
        /*
        for(unsigned int i=0;i<this->users.size();i++)
        {
                if(users[i].getUserAddr() == addr)
                {
                        return true;
                }
        }
        */
        return true;
}
bool UserList::deleteTimeOutedUsers()
{
        bool delAtLeastOneUser = false;
        for(int i = users.size() - 1; i >=0; i--)
        {
                if(users[i].getLiveTime() <= 0)
                {
                        MyLogger::WriteObjectInfo("user de-
leted[timeout]",(char*)(users[i].getUserName().c_str()));
                        mysocket s = users[i].getUserSocket();
                        MyServer::closeSocket(s);
                        users.erase(users.begin() + i);
                        delAtLeastOneUser = true;
                }
        }
        return delAtLeastOneUser;
}
bool UserList::deleteQuitUser(mysocket s)
{
        bool delAtLeastOneUser = false;
        for(int i = users.size() - 1; i >=0; i--)
        {
                if(users[i].getUserSocket() == s)
                {
                        MyLogger::WriteObjectInfo("user de-
leted[quit]",(char*)(users[i].getUserName().c_str()));
                        MyServer::closeSocket(s);
                        users.erase(users.begin() + i);
                        delAtLeastOneUser = true;
                }
        }
        return delAtLeastOneUser;
}
bool UserList::deleteQuitUser(sockaddr_in addr)
{
        bool delAtLeastOneUser = false;
        for(int i = users.size() - 1; i >=0; i--)
        {
```

```
                bool identAddr = this->CompareAddr(addr,users[i].getUserAddr());
                if(identAddr == true)
                {
                        MyLogger::WriteObjectInfo("user de-
leted[quit]",(char*)(users[i].getUserName().c_str()));
                        users.erase(users.begin() + i);
                        delAtLeastOneUser = true;
                }
        }
        return delAtLeastOneUser;
}
bool UserList::CompareAddr(sockaddr_in addr1, sockaddr_in addr2)
{
        if (addr1.sin_family!= addr1.sin_family)
        {
                return false;
        }
        if(addr1.sin_port != addr2.sin_port)
        {
                return false;
        }
        return true;
}
```

User.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include "CrossPlatformDefines.h"


using namespace std;


class User
{
public:
       User(void);
       User(string userName,mysocket socket);
       User(string userName,sockaddr_in addr);
       User(mysocket socket);
       ~User(void);

    int getLiveTime();
    string getUserName();
       mysocket  getUserSocket();
       sockaddr_in getUserAddr();
    void DecreaseLiveTime(int value);//decrem
    void RestoreLiveTime();//restart
       void setWarning(bool isWarning);
       bool getWarning();

       static string ParseName(string data);
private:
       string userName;
       sockaddr_in addr;
    //unsigned int userID;
       mysocket  userSocket;
    int LifeTime; //user life time before disconnect
       bool isSendWarning;
};
```

User.cpp

```
#include "stdafx.h"
#include "User.h"
#include "MyServer.h"
User::User(void)
{
    RestoreLiveTime();
}
User::User(string userName,mysocket socket)
{
        this->userName = userName;
        this->userSocket = socket;
        RestoreLiveTime();
}
User::User(string userName,sockaddr_in addr)
{
        this->userName = userName;
        this->addr= addr;
        RestoreLiveTime();
}
User::User(mysocket socket)
{
        this->userName = "";
        this->userSocket = socket;
        RestoreLiveTime();
}
User::~User(void)
{
}
int User::getLiveTime()
{
    return this->LifeTime;
}
void User::DecreaseLiveTime(int value) //decrease livetime to value
{
    value = (value < 0)? 0: value;
        this->LifeTime -=value;
}
void User::RestoreLiveTime() //restart
{
    this->LifeTime = MyServer::USER_TIMEOUT; //max time
        this->isSendWarning = false;
}
string User::getUserName()
{
    return this->userName;
}
mysocket  User::getUserSocket()
{
        return this->userSocket;
}
sockaddr_in User::getUserAddr()
{
        return this->addr;
}
string User::ParseName(string data)
{
        if(data.size() <=0)
        {
                return "";
        }
        string s_name_len = data.substr(0,2);
```

```
        int name_len = stoi(s_name_len);
        if(name_len <=0)
        {
                return "";
        }
        string s_name = data.substr(2,name_len);
        //s_name +='\0';
        return s_name;
}
        void User::setWarning(bool isWarning)
        {
                this->isSendWarning = isWarning;
        }
        bool User::getWarning()
        {
                return this->isSendWarning;
      }
```

Command.h

```
#ifndef _COMMAND_H_
#define _COMMAND_H_

#include <iostream>
#pragma once
class CommandsIDs
{
public:
    enum IDs
    {
        ECHO_REQUEST,
        ECHO_ANSWER,

        LOGIN_REQUEST,
        LOGIN_ANSWER,

        USERS_REQUEST,
        USERS_ANSWER,

        BAD_PACKAGE_ANSWER,
        UNAUTHORIZED_ACTION,// user not login

        MESSAGE_SEND,
        MESSAGE_INCORRECT_NAME,
        MESSAGE_SUCCESS_SENDED,
        MESSAGE_USER_OFFLINE,
        MESSAGE_DELIVERY,

        ADVERT_MESSAGE,

        ALIVE_REQUEST, //server asks client about life
        ALIVE_ANSWER, //client wants to be alive

        QUIT_REQUEST, //client want to disconnect


    };
};

class Command
{
public:
      int commandID; //ident
      Command(int commandID);
```

```
        ~Command(void);
    //char * data
        std::string data;
        int data_len;
        void GenerateData();
};


#endif
```

Command.cpp

```
#include <stdio.h>
#include "Command.h"


Command::Command(int commandID)
{
        this->commandID = commandID;
}

Command::~Command(void)
{
}
```

Command Factory.h

```
#ifndef COMMANDFACTORY_H
#define COMMANDFACTORY_H



#include <string>
#include "Command.h"
#include "UserList.h"

class EchoAnswerCommand : public Command
{
public:
    EchoAnswerCommand(void); //create command id
    void GenerateData(); //create data field
};

//---------------------------------------------
//---------------------------------------------

class EchoRequestCommand : public Command
{
public:
        EchoRequestCommand(void);
        void GenerateData();
};
//---------------------------------------------
//---------------------------------------------
class LoginAnswerCommand : public Command
{
public:
    LoginAnswerCommand(bool isGoodLogin);
    void GenerateData();

private:
    bool isGoodLogin;
};
//---------------------------------------------
```

```
//-------------------------------------------

class UsersAnswerCommand :public Command
{
public:
    UsersAnswerCommand(UserList * userlist);
    void GenerateData();
private:
       UserList * userlist;
};

//-------------------------------------------
//-------------------------------------------

class BadPackageCommand : public Command
{
public:
    BadPackageCommand(void);
    void GenerateData();
};
//-------------------------------------------
//-------------------------------------------
class UnAuthoririzedActionCommand: public Command
{
private:
       int action_id;
public:
       UnAuthoririzedActionCommand(int id);
       void GenerateData();
};
//-------------------------------------------
//-------------------------------------------
class QuitCommand: public Command
{
public:
       QuitCommand(void);
       void GenerateData();
};
//-------------------------------------------
//-------------------------------------------
class AdvertCommand: public Command
{
private:
       string advert_text;
public:
       AdvertCommand(string advert_text);
       void GenerateData();
};
//-------------------------------------------
//-------------------------------------------

class CommandFactory
{
public:
    CommandFactory();
};



#endif // COMMANDFACTORY_H
```

Command Factory.cpp

```
#include "commandFactory.h"
#include "../ServerApp/StringsHelper.h"
CommandFactory::CommandFactory()
{
}
//-----------------------------------------------
//----------------EchoAnswerCommand------------

EchoAnswerCommand::EchoAnswerCommand(void): Command(CommandsIDs::ECHO_ANSWER) //command type
{
	GenerateData();
}

void EchoAnswerCommand::GenerateData() //data
{
	this->data = "1";
	this->data_len = data.size();
}




//-----------------------------------------------
//-----------------LoginAnswerCommand----------

LoginAnswerCommand::LoginAnswerCommand(bool isGoodLogin) :Command(CommandsIDs::LOGIN_ANSWER)
{
    this->isGoodLogin = isGoodLogin;
	GenerateData();
}

void LoginAnswerCommand::GenerateData()
{
	if(this->isGoodLogin == true)
	{
		this->data = "1";
		this->data_len = data.size();
	}
	else
	{
		this->data = "0";
		this->data_len = data.size();
	}
}


//-----------------------------------------------
//-----------------------------------------------

UsersAnswerCommand::UsersAnswerCommand(UserList * userlist) : Command(CommandsIDs::USERS_ANSWER)
{
	this->userlist = userlist;
	GenerateData();
}

void UsersAnswerCommand::GenerateData()
{
	string s_usertext = this->userlist->getUserList();
	this->data = s_usertext;
	this->data_len = s_usertext.size();
}
```

```
//-------------------------------------------
//-------------------------------------------
UnAuthoririzedActionCommand::UnAuthoririzedActionCommand(int
id):Command(CommandsIDs::UNAUTHORIZED_ACTION)
{
	this->action_id = id;
	GenerateData();
}
void UnAuthoririzedActionCommand::GenerateData()
{
	this->data = ""+action_id;
	this->data_len = data.size();
}

//-------------------------------------------
//-------------------------------------------
BadPackageCommand::BadPackageCommand(void) :Command(CommandsIDs::BAD_PACKAGE_ANSWER)
{
	this->data = "1";
	this->data_len = data.size();
}

//-------------------------------------------
//-------------------------------------------
EchoRequestCommand::EchoRequestCommand() :Command(CommandsIDs::ECHO_REQUEST)
{
}
void EchoRequestCommand::GenerateData()
{
	this->data = "1";
	this->data_len = data.size();
}


//-------------------------------------------
//-------------------------------------------
QuitCommand::QuitCommand() :Command(CommandsIDs::QUIT_REQUEST)
{

}
void QuitCommand::GenerateData()
{
	this->data = "1";
	this->data_len = data.size();
}

//-------------------------------------------
//-------------------------------------------
AdvertCommand::AdvertCommand(string advert_text): Command(CommandsIDs::ADVERT_MESSAGE)
{
	this->advert_text = advert_text;
}
void AdvertCommand::GenerateData()
{
	int advert_len = advert_text.size();
	string data = "" + advert_len+advert_text;
	this->data_len = data.size();
	this->data = data;
}
```

Package.h

```
#pragma once
#include "Command.h"
class Package
{
public:
    static const int LENGTH_FIELD_SIZE = 3;
    static const int TYPE_FIELD_SIZE = 2;
    static const int DATA_START_POSITION = LENGTH_FIELD_SIZE+TYPE_FIELD_SIZE;
    static const int MAX_PACKAGE_SIZE = 512;
public:
       Package(void);
       ~Package(void);
    static Command Parse(char* inputmessage); //convert inputmessage to command (id+data)
    Package ParseToPackage(char* inputmessage);
    static Package Generate(Command cmd);
    //char * data;
       std::string data;
       int data_length;
       int packagesize;

       void OutPackage();

};
```

Package.cpp

```
#include "Package.h"
#include <stdio.h>
#include "StringsHelper.h"
Package::Package(void)
{
}


Package::~Package(void)
{
}

Command Package::Parse(char* inputmessage)
{
       int shift = 0;
       //parse package length
       char *c_packlen = StringsHelper::getCleanBuffer(LENGTH_FIELD_SIZE);
       int i_packlen = 0;
       for(int i=0; i<Package::LENGTH_FIELD_SIZE;i++)
       {
              c_packlen[i] =inputmessage[shift];
              shift++;
       }
       sscanf(c_packlen,"%d",&i_packlen);

       inputmessage[i_packlen] = '\0';


       //parse command id
       char *c_packtype = StringsHelper::getCleanBuffer(TYPE_FIELD_SIZE);
       int i_packtype = 0;
       for(int i=0; i<Package::TYPE_FIELD_SIZE;i++)
       {
              c_packtype[i] =inputmessage[shift];
              shift++;
       }
    sscanf(c_packtype,"%d",&i_packtype); //convert type char-->int ('00' - 0)
```

```
        //get package data
        int data_size = i_packlen - shift+1;
        string templine = "";
        if(data_size > 0)
        {
                for(int i = 0;i<data_size;i++)
                {
                        templine += inputmessage[shift];
                        shift++;
                }
                templine[data_size-1] = '\0';

        }



    //generate command - type,data
    Command command(i_packtype);
    command.data = templine;
        command.data_len = templine.size();

        //destroy pointers

        //return commands
        return command;
}

Package Package::Generate(Command cmd) //generate package from command
{
    Package pack;
    pack.data_length = cmd.data_len;
        if(cmd.data_len < 0)
        {
                return pack;
        }
    pack.packagesize = Package::DATA_START_POSITION + pack.data_length;
    char * packagedata = StringsHelper::getCleanBuffer(pack.packagesize);


    for(int i=0;i< pack.packagesize;i++)
        {
                packagedata[i] = ' ';
        }

    for(int i=0;i< pack.data_length;i++)
        {
                packagedata[DATA_START_POSITION+i] = cmd.data[i];
        }
        int commandID = cmd.commandID;
        char commID_char[TYPE_FIELD_SIZE+1] = {};
        sprintf(commID_char,"%2i",commandID);

        char length_char[LENGTH_FIELD_SIZE+1] = {};
    sprintf(length_char,"%3i",pack.packagesize);


        for(int i = 0;i < Package::LENGTH_FIELD_SIZE;i++)
        {
                packagedata[i] = length_char[i];
        }

        for(int i = 0;i < Package::TYPE_FIELD_SIZE;i++)
```

```
	{
		packagedata[i+LENGTH_FIELD_SIZE] = commID_char[i];
	}

	//replace empty char to '0'
    for(int i=0;i< pack.packagesize;i++)
	{
		if(packagedata[i]==' ')
		{
			packagedata[i] = '0';
		}
	}
    pack.data = packagedata;
	return pack;
}


void Package::OutPackage()
{
	printf("[");
	int shift = 0;

	for(int i = 0; i< LENGTH_FIELD_SIZE;i++)
	{
		printf("%c",this->data[shift]);
		shift++;
	}
	printf("]");

	printf("[");
	for(int i = 0; i< TYPE_FIELD_SIZE;i++)
	{
		printf("%c",this->data[shift]);
		shift++;
	}
	printf("]");

	printf("[");
	for(int i = 0; i< this->data_length;i++)
	{
		printf("%c",this->data[shift]);
		shift++;
	}
	printf("]");

	printf("\n");
}


CrossPlatformDefines.h
#pragma once



#ifdef _WIN32
#include <WinSock2.h>
#include <windows.h>
typedef SOCKET mysocket;
typedef DWORD returnType; //win thread
typedef int sock_len;
typedef long long SockDescr;
#elif __linux
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
#include <pthread.h>
typedef int mysocket;
typedef void * returnType; //linux thread
typedef socklen_t sock_len;
typedef int SockDescr;
#endif

#ifndef INVALID_SOCKET
#define INVALID_SOCKET          (mysocket)(~0)
#endif

#ifndef SOCKET_ERROR
#define SOCKET_ERROR            (-1)
#endif

#ifndef _TCHAR
#define _TCHAR  wchar_t
#endif

MyMessage.h
#pragma once

#include <iostream>

using namespace std;
class MyMessage
{

public:
       static const int DEST_FIELD_SIZE  = 2;
       static const int SENDER_NAME_FIELD_SIZE  = 2;
       static const int TIMESTAMP_FIELD_SIZE  = 2;
       static const int MESSAGE_FIELD_SIZE  = 3;
       static const int SERVICE_FIELDS_SIZE = DEST_FIELD_SIZE + SENDER_NAME_FIELD_SIZE + TIME-
STAMP_FIELD_SIZE + MESSAGE_FIELD_SIZE;
public:
       MyMessage(void);
       MyMessage(string sender_name, string receiver_name, string message,string timestamp);
       ~MyMessage(void);
       string receiver_name;
       string sender_name;

       string message;
       string timestamp;

       bool correct_data;
       bool correct_receiver;

       static MyMessage ConvertDataToMessage(string);
};


MyMessage.cpp
#include "stdafx.h"
#include "MyMessage.h"
#include "StringsHelper.h"


MyMessage::MyMessage(void)
{
}

MyMessage::MyMessage(string sender_name, string receiver_name, string message,string timestamp)
```

```
{
	this->message = message;
	this->sender_name = sender_name;
	this->receiver_name = receiver_name;
	this->timestamp = timestamp;
}


MyMessage::~MyMessage(void)
{
}

MyMessage MyMessage::ConvertDataToMessage(string data)
{
	MyMessage message;
	int shift = 0;

	//parse receiver name
	string s_dest_len = data.substr(shift,DEST_FIELD_SIZE);
	shift +=DEST_FIELD_SIZE;
	int dest_len = atoi(s_dest_len.c_str());
	string dest = data.substr(shift,dest_len);
	shift+=dest_len;


	//parse sender name
	string s_sender_len = data.substr(shift,SENDER_NAME_FIELD_SIZE);
	shift+=SENDER_NAME_FIELD_SIZE;
	int sender_len = atoi(s_sender_len.c_str());
	string sender = data.substr(shift,sender_len);
	shift+=sender_len;


	//parse timestamp
	string s_timestamp_len = data.substr(shift,TIMESTAMP_FIELD_SIZE);
	shift+=TIMESTAMP_FIELD_SIZE;
	int timestamp_len = atoi(s_timestamp_len.c_str());
	string timestamp = data.substr(shift,timestamp_len);
	shift+=timestamp_len;

	//parse message
	string s_message_len = data.substr(shift,MESSAGE_FIELD_SIZE);
	shift +=MESSAGE_FIELD_SIZE;
	int message_len = atoi(s_message_len.c_str());
	string message_text = data.substr(shift,message_len);

	//add data to message structure and return structure
	message.receiver_name = dest;
	message.sender_name = sender;
	message.timestamp = timestamp;
	message.message = message_text;
	return message;

}
```

Myserver.h

```
#pragma once



//*****************************************
const int TCPregime = 1;                                //
const int UDPregime = 2;                                //
const int CurrentRegime = TCPregime;            //
```

```
//*****************************************


#include "CrossPlatformDefines.h"
#include "MyLogger.h"
#include "Package.h"
#include "UserList.h"


struct SendMetaInfo //tcp, common userlist
{
    mysocket socket;
       UserList *userlist;
};

class MyServer
{
public:
       //constants
    static const int DEFAULT_PORT  = 34343;

    static const int USER_TIMEOUT= (1*60 + 20)*1000;  //inactive time to disconnect users=  1 min
20 sec
       static const int USER_WARNING_TIME = (0*60 + 20)*1000;//time remains to warning message = 0
min 20 sec
    static const int USER_UPDATE_TIME_S = 3;    //update users every 3 sec  (linux)
    static const int USER_UPDATE_TIME_MS = USER_UPDATE_TIME_S*1000; // update users every 3000
msec (win)

       static const int ADVERT_SEND_PAUSE_S = 60;      //send advert message every 60 sec (linux)
       static const int ADVERT_SEND_PAUSE_MS = ADVERT_SEND_PAUSE_S*1000; //send advert message
every 1000 ms (windows)



    //constr /destr
       MyServer(void);
       ~MyServer(void);

    //
    UserList userlist;

       //crossplatform thread functions
#ifdef _WIN32
       static returnType WINAPI Communicate(void  * socket_param); //send/receive data function
       static returnType WINAPI TimerUserTimeOut(void * user_list_arg);    //delete timeout users
       static returnType WINAPI AdvertMessage(void * user_list_arg);       //send all active users
advert`s
#elif __linux
       static returnType Communicate(void  * socket_param); //send/receive data function
       static returnType TimerUserTimeOut(void * user_list_arg);    //delete timeout users
       static returnType AdvertMessage(void * user_list_arg); //send all active users advert`s
#endif
       //functions
    sockaddr_in createListenAddr();
    bool runTCP();
       bool runUDP();
    bool SetupLibrary();
    void CleanUpLibrary();
    Command ParceIncomingPackage(char * packagedata);

       static void SendData(Package package, mysocket socket,string info_message);
       static void SendDataUDP(Package package,sockaddr_in addr, mysocket socket,string in-
fo_message);
```

```
	static void closeSocket(mysocket socket);


};
```

Myserver.cpp

```
#include "stdafx.h"
#include "MyServer.h"

#include "commandFactory.h"
#include "StringsHelper.h"
#include "MyMessage.h"
#include "AdvertMessageList.h"

#ifdef _WIN32
#pragma comment(lib, "WS2_32.lib")
#elif __linux
#include <unistd.h>
#include <string.h>
#endif



MyServer::MyServer(void)
{
}


MyServer::~MyServer(void)
{
}

bool MyServer::SetupLibrary()
{
#ifdef _WIN32 //Windows

	WSADATA wsaData;
	int errorcode = WSAStartup(MAKEWORD(2,2),&wsaData);
	if (errorcode)
	{
		MyLogger::WriteFailNetworkEvent("setup lib error");
		return false;
	}
	else
	{
		MyLogger::WriteSuccessNetworkEvent("setub lib complete");
		return true;
	}
#endif
	return true;
}

void MyServer::CleanUpLibrary()
{
#ifdef _WIN32
	WSACleanup();
#endif
}

sockaddr_in MyServer::createListenAddr()
{
	sockaddr_in serv;
	serv.sin_family = AF_INET;
```

```
	serv.sin_addr.s_addr = INADDR_ANY;
	serv.sin_port = htons( DEFAULT_PORT );
	return serv;

}

bool MyServer::runTCP()
{
	mysocket s;
	s = socket(AF_INET,SOCK_STREAM,0);
	if(s == INVALID_SOCKET)
	{
		MyLogger::WriteFailNetworkEvent("invalid socket");
		return false;
	}

	sockaddr_in server_addr = createListenAddr();
	int servaddr_len = sizeof(server_addr);
	if (bind(s,(struct sockaddr * )&server_addr,servaddr_len) < 0)
	{
		MyLogger::WriteFailNetworkEvent("bind error\n");
		return false;
	}
	sockaddr_in in_addr;
	sock_len addr_len = sizeof(in_addr);
	listen(s , SOMAXCONN);
	int read_size = 0;
	mysocket s_in;
	printf("wait connections\n");

	//*************************
	#ifdef _WIN32
	CreateThread(NULL,0,&TimerUserTimeOut, &userlist,0,NULL); //timer thread
	#elif __linux
    pthread_t timerthread;
    pthread_create(&timerthread,NULL,TimerUserTimeOut,reinterpret_cast<void*>(&userlist));
	#endif
	//************************


	//*****************
	#ifdef _WIN32
	CreateThread(NULL,0,&AdvertMessage, &userlist,0,NULL); //advert thread
	#elif __linux
    pthread_t advertthread;
    pthread_create(&advertthread,NULL,AdvertMessage,reinterpret_cast<void*>(&userlist));
	#endif

	//*******************


	while( s_in = accept(s , (struct sockaddr * )&in_addr,&addr_len))
	{

		printf(" connection accepted\n");
		SendMetaInfo info = {s_in,&userlist}; //common userlist
#ifdef _WIN32
		CreateThread(NULL,0,&Communicate, &info,0,NULL); //thread
#elif __linux
		pthread_t sendthread;
		pthread_create(&sendthread,NULL,Communicate,reinterpret_cast<void*>(&info));
		//pthread_join(sendthread,NULL);
#endif
	}
```

```
	MyServer::closeSocket(s);
	return true;
}

bool MyServer::runUDP()
{
	mysocket s;
	s = socket(AF_INET,SOCK_DGRAM,0);
	int send_size = 0;
	if(s == INVALID_SOCKET)
	{
		MyLogger::WriteFailNetworkEvent("invalid socket");
		return false;
	}
	struct sockaddr_in servaddr,cliaddr;
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = INADDR_ANY;
	servaddr.sin_port = htons( DEFAULT_PORT );

	if (bind(s,(struct sockaddr * )&servaddr,sizeof(servaddr)) < 0)
	{
		MyLogger::WriteFailNetworkEvent("bind error\n");
		return false;
	}
	char* buffer_in = StringsHelper::getCleanBuffer(Package::MAX_PACKAGE_SIZE);
	mysocket s_in;
	printf("waiting connection\n");
	for(;;)
	{
	 sock_len len = sizeof(cliaddr);
		int rec_size = recvfrom(s,buffer_in,Package::MAX_PACKAGE_SIZE,0,(struct sock-
addr*)&cliaddr,&len); //cliaddr-incomming address
		printf("get %d byte\n",rec_size);
		buffer_in[rec_size] = 0;
		Command cmd = Package::Parse(buffer_in);
		if(cmd.commandID == CommandsIDs::ECHO_REQUEST)
		{
			MyLogger::WritePackageInfo("echo request","receive");
			EchoAnswerCommand echoAnswer;
			Package EchoPack = Package::Generate(echoAnswer);
			SendDataUDP(EchoPack,cliaddr,s,"echo request");
		}
		if(cmd.commandID == CommandsIDs::LOGIN_REQUEST)
		{
			MyLogger::WritePackageInfo("login request","receive");
			string login = User::ParseName(cmd.data);
			bool isUnique = userlist.isNameUnique(login);
			bool isRight  = userlist.isNameRight(login);

			bool Answer = isUnique && isRight;
			if(Answer == true)
			{
				User* newUser = new User(login,cliaddr);
				userlist.tryAddNewUser(newUser);
				MyLogger::WriteSuccessNetworkEvent("add new us-
er",(char*)login.c_str());
			}
			else
			{
				if(isUnique == false)
				{
					MyLogger:: WriteFailEvent("bad login:not unique [" + login +
"]");
```

```
                        }
                        if(isRight == false)
                        {
                                MyLogger:: WriteFailEvent("bad login: incorrect lenght or sym-
bols ["+login + "]");
                        }
                }
                LoginAnswerCommand loginAnswer(Answer);
                Package LoginPack = Package::Generate(loginAnswer);
                SendDataUDP(LoginPack,cliaddr,s,"login answer");
                if(send_size > 0)
                {
                        MyLogger::WritePackageInfo("login answer","send");
                }
        }
        if(cmd.commandID == CommandsIDs::USERS_REQUEST)
        {
                if(userlist.isAuthorizedUser(cliaddr)==false)
                {
                        //send unauthorized action package
                        MyLogger::WriteFailEvent("unauthorized user action");
                }
                MyLogger::WritePackageInfo("users request","receive");
                UsersAnswerCommand usersanswer(&userlist);
                Package UserPack = Package::Generate(usersanswer);
                SendDataUDP(UserPack,cliaddr,s,"user answer");
                if(send_size > 0)
                {
                        MyLogger::WritePackageInfo("users answer","send");
                }
        }
        if(cmd.commandID == CommandsIDs::MESSAGE_SEND)
        {
                MyLogger::WritePackageInfo("message","receive");

                MyMessage incoming_message = MyMessage::ConvertDataToMessage(cmd.data);
                //mysocket s_out = userlist.getSocketByName(incoming_message.receiver_name);
                sockaddr_in addr = userlist.getAddrByName(incoming_message.receiver_name);
                send_size = sendto(s,buffer_in,strlen(buffer_in),0,(struct sock-
addr*)&addr,sizeof(addr));
                if(send_size > 0)
                {
                        MyLogger::WritePackageInfo("message","send");
                }
                else
                {
                        MyLogger::WriteFailEvent("fail to delivery incoming message");
                }
        }
        if(cmd.commandID == CommandsIDs::ECHO_ANSWER)
        {
                MyLogger::WritePackageInfo("echo answer","receive");
        }
        if(cmd.commandID == CommandsIDs::QUIT_REQUEST)
        {
                MyLogger::WritePackageInfo("quit request","receive");
                userlist.deleteQuitUser(cliaddr);
        }
    }
    return true;
}

returnType MyServer::TimerUserTimeOut(void * user_list_arg) //timer thread function
{
```

```
	UserList* userlist = (UserList*)(user_list_arg);
	while(1)
	{


		for(unsigned int i = 0; i<userlist->users.size() ; i++)
		{
			User* curUser = &userlist->users[i];
			if(curUser->getLiveTime() <= 0)
			{
				QuitCommand quitCmd;
				quitCmd.GenerateData();
				Package packquit = Package::Generate(quitCmd);
				SendData(packquit,curUser->getUserSocket(),"quit timeout command to " +
curUser->getUserName());
			}
		}

		for(unsigned int i = 0; i<userlist->users.size() ; i++)
		{
			User* curUser = &userlist->users[i];
			if(curUser->getLiveTime() <=  MyServer::USER_WARNING_TIME && curUser-
>getWarning() == false)
			{
				curUser->setWarning(true);
				EchoRequestCommand echoRequestCmd;
				echoRequestCmd.GenerateData();
				Package packEchoReq = Package::Generate(echoRequestCmd);
				SendData(packEchoReq,curUser->getUserSocket(),"timeout warning to " +
curUser->getUserName());
			}
		}
#ifdef __linux
		sleep(MyServer::USER_UPDATE_TIME_S);
#elif _WIN32
		Sleep(MyServer::USER_UPDATE_TIME_MS);
#endif
		userlist->DecreaseTimeUsers(MyServer::USER_UPDATE_TIME_MS); //decrease lifetime for
all users 3sec, sleep 3sec
		userlist->deleteTimeOutedUsers(); //delete users (lifetime < 0)
	}
	return NULL;
}

returnType MyServer::Communicate(void  * socket_param)
{
	SendMetaInfo info = *(SendMetaInfo*)socket_param;
	mysocket s_in = info.socket;
	UserList * userlist = info.userlist;

	char* buffer_in = StringsHelper::getCleanBuffer(Package::MAX_PACKAGE_SIZE);
	//string buffer_in;
	int read_size = 0;
	int send_size = 0;
	while(read_size = recv(s_in,buffer_in,Package::MAX_PACKAGE_SIZE,0) > 0)
	{
		Command cmd = Package::Parse(buffer_in);
		userlist->RestoreTimeBySocket(s_in);
		if(cmd.commandID == CommandsIDs::ECHO_REQUEST)
		{
			MyLogger::WritePackageInfo("echo request","receive");
			EchoAnswerCommand echoAnswer;
			Package EchoPack = Package::Generate(echoAnswer);
			SendData(EchoPack,s_in,"echo request");
```

```
		}
		if(cmd.commandID == CommandsIDs::LOGIN_REQUEST)
		{
			MyLogger::WritePackageInfo("login request","receive");
			string login = User::ParseName(cmd.data);
			bool isUnique = userlist->isNameUnique(login);
			bool isRight  = userlist->isNameRight(login);

			bool Answer = isUnique && isRight;
			if(Answer == true)
			{
				userlist->tryAddNewUser(login,s_in);
		  SockDescr descr = (int)s_in;
				string newUserInfo = "["+login+" , " + to_string(descr) + "]";
				MyLogger::WriteSuccessNetworkEvent("add new us-
er",(char*)newUserInfo.c_str());
			}
			else
			{
				if(isUnique == false)
				{
					MyLogger:: WriteFailEvent("bad login:not unique [" + login +
"]");
				}
				if(isRight == false)
				{
					MyLogger:: WriteFailEvent("bad login: incorrect lenght or sym-
bols ["+login + "]");
				}
			}
			LoginAnswerCommand loginAnswer(Answer);
			Package LoginPack = Package::Generate(loginAnswer);
			send_size =  send(s_in,LoginPack.data.c_str(),LoginPack.packagesize,0);
			if(send_size > 0)
			{
				MyLogger::WritePackageInfo("login answer","send");
			}
		}
		if(cmd.commandID == CommandsIDs::USERS_REQUEST)
		{
			if(userlist->isAuthorizedUser(s_in)==false)
			{
				//send unauthorized action package
				MyLogger::WriteFailEvent("unauthorized user action");
			}
			MyLogger::WritePackageInfo("users request","receive");
			UsersAnswerCommand usersanswer(userlist);
			Package UserPack = Package::Generate(usersanswer);
			send_size =  send(s_in,UserPack.data.c_str(),UserPack.packagesize,0);
			if(send_size > 0)
			{
				MyLogger::WritePackageInfo("users answer","send");
			}
		}
		if(cmd.commandID == CommandsIDs::MESSAGE_SEND)
		{
			MyLogger::WritePackageInfo("message","receive");

			MyMessage incoming_message = MyMessage::ConvertDataToMessage(cmd.data);
			mysocket s_out = userlist->getSocketByName(incoming_message.receiver_name);
			send_size =  send(s_out,buffer_in,strlen(buffer_in),0);
			if(send_size > 0)
			{
				MyLogger::WritePackageInfo("message","send");
```

```
                        }
                        else
                        {
                                MyLogger::WriteFailEvent("fail to delivery incoming message");
                        }
                }
                if(cmd.commandID == CommandsIDs::ECHO_ANSWER)
                {
                        MyLogger::WritePackageInfo("echo answer","receive");
                }
                if(cmd.commandID == CommandsIDs::QUIT_REQUEST)
                {
                        MyLogger::WritePackageInfo("quit request","receive");
                        userlist->deleteQuitUser(s_in);
                }
        }
        return NULL;
}

void MyServer::closeSocket(mysocket socket)
{
#ifdef _WIN32
        shutdown(socket,SD_BOTH);
        closesocket(socket);
#elif __linux
        shutdown(socket,SHUT_RDWR);
        close(socket);
#endif
}

void MyServer::SendData(Package package, mysocket socket,string info_message)
{
        int send_size = 0;

        send_size =  send(socket,package.data.c_str(),package.packagesize,0);
        if(send_size > 0)
        {
                MyLogger::WritePackageInfo((char*)info_message.c_str(),"send");
        }
        else
        {
                string fail_text = "fail to send " + info_message;
                MyLogger::WriteFailEvent((char*)fail_text.c_str());
        }
}

void MyServer::SendDataUDP(Package package,sockaddr_in addr, mysocket socket,string info_message)
{
        int send_size = 0;
        sock_len addr_len = sizeof(addr);
        send_size = sendto(socket,package.data.c_str(),package.packagesize,0,(struct sock-
addr*)&addr,addr_len);
}

returnType MyServer::AdvertMessage(void * user_list_arg)
{
        UserList* userlist = (UserList*)(user_list_arg);
        AdvertMessageList advert_list; //advertmess
        while(1)
        {
                string text = advert_list.getNextAdvert();
                AdvertCommand advertCmd(text);
                advertCmd.GenerateData();
                Package packAdvert = Package::Generate(advertCmd);
```

```
		for(unsigned int i = 0; i<userlist->users.size() ; i++)
		{
			User* curUser = &userlist->users[i];
			SendData(packAdvert,curUser->getUserSocket(),"advert to " + curUser-
>getUserName());
		}
		#ifdef __linux
		sleep(MyServer::ADVERT_SEND_PAUSE_S);
		#elif _WIN32
		Sleep(MyServer::ADVERT_SEND_PAUSE_MS);
		#endif
	}
}
```

ServerApp.cpp

```
#include "stdafx.h"
#include "ApplicationOptions.h"
#include "MyServer.h"


int main(int argc, _TCHAR* argv[])
{
	ApplicationOptions::setServerAppOptions();
	MyServer* server = new MyServer();
	server->SetupLibrary();
    if(CurrentRegime == TCPregime)
    {
		printf("TCP\n");
	  server->runTCP();
    }
    if(CurrentRegime == UDPregime)
    {
		printf("UDP\n");
		server->runUDP();
    }
	server->CleanUpLibrary();
}
```

Advertmessagelist.h

```
#pragma once

#include <vector>
#include <string>
using namespace std;

class AdvertMessageList
{
private:
	vector<string>advertList;
	int Counter;
public:
	AdvertMessageList(void);
	~AdvertMessageList(void);
	string getNextAdvert();
};
```

Advertmessagelist.cpp

```
#include "stdafx.h"
#include "AdvertMessageList.h"


AdvertMessageList::AdvertMessageList(void)
{
	advertList.push_back("http://habrahabr.ru - be smart");
	advertList.push_back("http://ideone.com    Online coding. Be mobile!");
	advertList.push_back("https://www.google.ru - we know");
	this->Counter = 0;
}

AdvertMessageList::~AdvertMessageList(void)
{
}


string AdvertMessageList::getNextAdvert() //choose advert
{

	if( advertList.size() == 0 )
	{
		return "";
	}
	if(Counter >= advertList.size())
	{
		Counter = 0;
	}
	Counter++;
	return advertList[Counter - 1];
}
```

Код для клиента:

My Client.h

```
#ifndef _MYCLIENT_H_
#define _MYCLIENT_H_



#include "PackageStructure.h"
#include "CrossPlatformDefines.h"
#include "MessageCollection.h"
#include "User.h"

//*****************************************
const int TCPregime = 1;                                //
const int UDPregime = 2;                                //
const int CurrentRegime = TCPregime;            //
//*****************************************

#ifdef __linux__
    #include <sys/socket.h>
    #include <arpa/inet.h>
    typedef int mysocket;
       typedef unsigned int addr_size;
    typedef void * returnType;
```

```
#elif _WIN32
	#include <WinSock2.h>
	#include <windows.h>
	typedef SOCKET mysocket;
	typedef int addr_size;
	typedef DWORD  returnType;
	typedef int socklen_t;

#else

#endif

#include "MyLogger.h"



struct SendMetaInfo //for IO thread
{
	char * login;
	UserList * userlist;
	mysocket *s;
	MessageCollection * collection;
	int CurRegime;
};
enum ActionID
{
	ACTION_SHOW_MESSAGE  =1,
	ACTION_SEND_MESSAGE  =2,
	ACTION_SHOW_CONTACTS = 3,
	ACTION_QUIT = 4,
};


static const int DEFAULT_PORT = 34343;

static const int MAX_PACKAGE_SIZE = 512;

bool SetUpLib();
void CleanLib();

sockaddr_in createSendAddr();
mysocket createSocket();
bool runTCP();
bool runUDP();
void Listen(void);
void * Answer( void * arg);
static void closeSocket(mysocket socket);
int sendData(mysocket s,char *data);
int sendDataUDP(mysocket s,sockaddr_in addr,char *data);

#ifdef _WIN32
	static returnType WINAPI InputText(void  * info_param);
#elif __linux
	static returnType InputText(void  * info_param);
#endif

#endif

char * inputLogin();
int  getAction(SendMetaInfo info);
char * inputMessage();
User GetUser(UserList* list);
void closeSocket(mysocket socket);
```

Myclient.cpp

```
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>


#include "MyClient.h"
#include "CommandFactory.h"
#include "CommandsIDs.h"
#include "MyLogger.h"
#include "MyCreator.h"
#include "GetCurrentTime.h"
#include "CrossPlatformDefines.h"



#ifdef _WIN32
       #pragma comment(lib, "WS2_32.lib")
#elif __linux
    #include <unistd.h>
#endif

sockaddr_in createSendAddr()
{
       sockaddr_in address;

       address.sin_family = AF_INET;
    //address.sin_addr.s_addr = inet_addr("192.128.0.13"); //server adress
       address.sin_addr.s_addr = inet_addr("127.0.0.1"); //server adress
    address.sin_port = htons( DEFAULT_PORT );
       return address;
}

bool SetUpLib()
{
       #ifdef _WIN32
              WSADATA wsaData;
              int errorcode = WSAStartup(MAKEWORD(2,2),&wsaData);
        if (errorcode)
           {
               //printf("fail setup lib\n");
                       WriteFailNetworkEvent("fail to setup lib");
               return false;
           }
           else
           {
                       WriteSuccessNetworkEvent("setup lib");
               //printf("setup lib\n");
               return true;
           }
       #endif
}

void CleanLib()
{
#ifdef _WIN32
    WSACleanup();
#endif
}

mysocket createSocket()
{
    return socket(AF_INET,SOCK_STREAM,0);
}
```

```
bool runTCP()
{
    mysocket s = createSocket();
       UserList list = initEmptyList();
       char *login = getCleanBuffer(1);
#ifdef _WIN32
    MessageCollection *collection = &initEmptyCollection();
#elif __linux
    MessageCollection *collection = initEmptyCollectionPointer();
#endif

    if(s == INVALID_SOCKET)
    {
              WriteFailNetworkEvent("bad socket");
        //printf("bad socket\n");
        return false;
    }
    sockaddr_in adr = createSendAddr();
    addr_size addr_len = sizeof(adr);

    int connectionResult = connect(s,(struct sockaddr *)&adr, addr_len); //connect to server
    if(connectionResult < 0)
    {
              WriteFailNetworkEvent("bad connection");
        //printf("bad connection\n");
        return false;
    }

   // sockaddr_in incomingAddr;
    int send_size = 0;
       int rec_size = 0;
    bool isOnWork = true; //client works - listen from server
       char * receivebuffer = getCleanBuffer(MAX_PACKAGE_SIZE);

       SendMetaInfo info = {login,&list,&s,collection};
       #ifdef _WIN32
              CreateThread(NULL,0,&InputText, &info,0,NULL);
       #elif __linux
         pthread_t sendthread;
         pthread_create(&sendthread,NULL,InputText,reinterpret_cast<void*>(&info));
         //pthread_join(sendthread,NULL);
       #endif

    CommandStructure EchoRequestCmd = GenerateEchoRequestCommand(); //client sends echo request
    PackageStructure package_echo_req = CreatePackage(EchoRequestCmd);
    char *senddata = GeneratePackageData(package_echo_req);
       send_size = sendData(s,senddata);
       if(send_size < 0)
       {
              printf("fail to send echo request\n");
       }
       if(send_size > 0)
       {
              printf("send echo request\n");
       }
       printf("wait server answers\n");

    while(isOnWork) // client listen
    {

        while (recv (s,receivebuffer,MAX_PACKAGE_SIZE,0) > 0)
        {
                    CommandStructure cmd = Parse(receivebuffer);
```

```
                        if(cmd.CommandId == IDs::ECHO_ANSWER)
                        {
                                printf("receive echo answer. server online\n");
                                char *inplogin = inputLogin(); //client inputs login
                                memcpy(login,inplogin,strlen(inplogin));
                                CommandStructure LoginRequestCmd = GenerateLoginRequestCom-
mand(inplogin); //command = type+data
                                PackageStructure package_login_req = CreatePackage(LoginRequestCmd);
                                char *sendlogindata = GeneratePackageData(package_login_req); //convert
(6 0 1 -> 006001)
                                send_size = sendData(s,sendlogindata);
                                if(send_size < 0)
                                {
                                        printf("fail to send login request\n");
                                }
                                if(send_size > 0)
                                {
                                        printf("send login request\n");
                                }
                                //free(sendlogindata);
                        }
                        if(cmd.CommandId == IDs::LOGIN_ANSWER)
                        {
                                if(cmd.data[0] == '1')
                                {
                                        printf("login accepted\n");
                                        CommandStructure UsersRequestCmd = GenerateUsersRequestCom-
mand();
                                        PackageStructure package_users_req = CreatePack-
age(UsersRequestCmd);
                                        char *sendusersdata = GeneratePackageData(package_users_req);
                                        send_size =sendData(s,sendusersdata);
                                        if(send_size < 0)
                                        {
                                                printf("fail to send users request\n");
                                        }
                                        if(send_size > 0)
                                        {
                                                printf("send users request\n");
                                        }

                                }
                                else
                                {
                                        printf("login rejected. Try again\n");
                                        char *inplogin_r = inputLogin(); //client inputs login
                                        memcpy(login,inplogin_r,strlen(inplogin_r));
                                        CommandStructure LoginRequestCmd = GenerateLoginRequestCom-
mand(login); //command = type+data
                                        PackageStructure package_login_req = CreatePack-
age(LoginRequestCmd);
                                        char *sendlogindata = GeneratePackageData(package_login_req);
//convert (6 0 1 -> 006001)
                                        send_size = sendData(s,sendlogindata);
                                        if(send_size < 0)
                                        {
                                                printf("fail to send login request\n");
                                        }
                                        if(send_size > 0)
                                        {
                                                printf("send login request\n");
                                        }
                                }
                        }
```

```
                        if(cmd.CommandId == IDs::UNAUTHORIZED_ACTION)
                        {
                                printf("unauthorized action answer\n");
                        }
                        if(cmd.CommandId == IDs::USERS_ANSWER)
                        {
                                printf("receive users answer\n");
                                list = ParseUsers(cmd.data);
                        }
                        if(cmd.CommandId == IDs::MESSAGE_SEND)
                        {
                                MessageStructure message =  parseData(cmd.data);
                                AddNewUnReadedMessage(collection,message);
                        }

                        if(cmd.CommandId == IDs::ECHO_REQUEST)
                        {
                                CommandStructure EchoAnswerCmd = GenerateEchoAnswerCommand(); //client
sends echo answer
                                PackageStructure package_echo_ans = CreatePackage(EchoAnswerCmd);
                                char *senddata = GeneratePackageData(package_echo_ans);
                                send_size = sendData(s,senddata);
                        }
                        if(cmd.CommandId == IDs::QUIT_REQUEST)
                        {
                                WriteFailEvent("Quit command: close app");
                                closeSocket(s);
                                exit(2);
                        }
                        if(cmd.CommandId == IDs::ADVERT_MESSAGE)
                        {
                                MessageStructure message = convertAdvertToMessage(cmd.data);
                                AddNewUnReadedMessage(collection,message);
                        }
                        free(receivebuffer);
                        receivebuffer = getCleanBuffer(MAX_PACKAGE_SIZE);
                }
        }

        return true;
}


int sendData(mysocket s,char *data)
{
        int len = strlen(data)+1;
        return send(s,data,len,0);
}

int sendDataUDP(mysocket s,sockaddr_in addr,char *data)
{
        int len = strlen(data)+1;
        return sendto(s,data,len,0, (struct sockaddr *)&addr,sizeof(addr));
}

char * inputLogin()
{
        char * login = getCleanBuffer(256);

        printf("input your name\n");
        scanf("%255s",login);
        return login;
}
```

```
char * inputMessage()
{
	char * message = getCleanBuffer(256);
	printf("input message:\n");
    scanf("%255s",message);//scanf ignore symbols after whitespace
	scanf("%255[^\n]", message);
	//scanf("%255[0-9a-zA-Z ]s", message);
	/*
	fgets (message, 256, stdin);
	if ((strlen(message)>0) && (message[strlen (message) - 1] == '\n'))
	{
		message[strlen (message) - 1] = '\0';
	}
	*/
	return message;
}

int  getAction(SendMetaInfo info) //from console
{
	int action_id = 1;
	do
	{
		ClearConsole();
		PrintfInfoTitleText(info.login,info.userlist->list_len,info.collection-
>unreaded_size);
		printf("choose action:\n");
		printf("%d. show messages\n", ACTION_SHOW_MESSAGE);
		printf("%d. send message\n",ACTION_SEND_MESSAGE);
		printf("%d. show contacts\n",ACTION_SHOW_CONTACTS);
		printf("%d. quit\n",ACTION_QUIT);
		scanf("%d",&action_id);
	}
	while(action_id < ACTION_SHOW_MESSAGE || action_id > ACTION_QUIT);
	return action_id;
}

User GetUser(UserList* list)
{
	int user_number = -1;
	int own_id = -1;
	do
	{
		printf("choose message receiver:\n");
		for(int i = 0; i < list->list_len;i++)
		{
			//char * username;
			//username = getCleanBuffer(list.users[i].name_len + 1);
			//memcpy(username,list.users[i].name,list.users[i].name_len);
			//username[list.users[i].name_len] = '\0';
			printf("%d) %s\n",i+1,list->users[i].name);
		}
		printf("0) back\n");
		scanf("%d",&user_number);
	}
	while(user_number < 0 || user_number - 1 >= list->list_len|| user_number == own_id);

	if(user_number == 0)
	{
		return EmptyUser();
	}
	return list->users[user_number - 1];
}

void PrintMessages(MessageCollection* collection)
```

```
{
	ClearConsole();

	WriteAllMessages(collection); //show all messages
	//MoveAllUnReadToRead(collection);
	printf("\n<- press any key to return\n");
	int any_key = 0;
	scanf("%s");
}


#ifdef __linux
returnType InputText(void  * info_param) //thread function
#elif _WIN32
 returnType WINAPI InputText(void  * info_param) //thread function
#endif

{
	SendMetaInfo info = *(SendMetaInfo*)info_param;

	char* login = info.login;
	mysocket s = *info.s;
	UserList* list = info.userlist;
	MessageCollection* collection = info.collection;
	int curRegime = info.CurRegime;

	sockaddr_in addr = createSendAddr();


	while(1)
	{
		if(strlen(info.login) <=0   || info.userlist->list_len <= 0) //condition
		{
			#ifdef __linux
			usleep(500*1000);
			#elif _WIN32
			Sleep(500); //uncorrect login or userlist, sleep
			#endif
		}
		else
		{

			int action = 0;
			action = getAction(info);
			if(action == ACTION_SEND_MESSAGE)
			{
				int send_size = 0;
				int rec_size = 0;
				int len = 0;
				User user = GetUser(list);
				if(isEmptyUser(user)== false)
				{
					char * message_text;
					message_text = inputMessage();
					MessageStructure message = generateMes-
sage(message_text,login,user,getCurrentTime());
					CommandStructure MessageSendCommand = GenerateMessageSendCom-
mand(message);
					PackageStructure package_message_send = CreatePack-
age(MessageSendCommand);
					char *sendmessagedata = GeneratePackageDa-
ta(package_message_send);
					if(CurrentRegime == TCPregime)
					{
```

```
                        send_size = sendData(s,sendmessagedata);
                }
                if(CurrentRegime == UDPregime)
                {
                        send_size = sendDataUDP(s,addr,sendmessagedata);
                }
                if(send_size < 0)
                {
                        printf("fail to send message\n");
                }
                if(send_size > 0)
                {
                        printf("send message\n");
                }
        }
        else
        {
                printf("empty user\n");
        }

        continue;
}

if(action == ACTION_SHOW_CONTACTS)
{
        //send users request [view actual userlist]
        CommandStructure UsersRequestCmd = GenerateUsersRequestCommand();
        PackageStructure package_users_req = CreatePackage(UsersRequestCmd);
        char *sendusersdata = (char*)UsersRequest;
        if(CurrentRegime == TCPregime)
        {
                int send_size = sendData(s,sendusersdata);
        }
        if(CurrentRegime == UDPregime)
        {
                sendDataUDP(s,addr,sendusersdata);
        }
        #ifdef __linux
        usleep(500*1000);
        #elif _WIN32
        Sleep(500);
        #endif
        GetUser(list);
        continue;
}
if(action == ACTION_SHOW_MESSAGE)
{
        PrintMessages(collection);
        continue;
}
if(action == ACTION_QUIT)
{
        char *sendquitdata = (char*)quitPackage;
        if(CurrentRegime == TCPregime)
        {
                sendData(s,sendquitdata);
                closeSocket(s);
                exit(0);
        }
        if(CurrentRegime == UDPregime)
        {
                sendDataUDP(s,addr,sendquitdata);
                exit(0);
        }
```

```
                        }
                }

        }
        return 0;
}


bool runUDP()
{
        mysocket s = socket(AF_INET,SOCK_DGRAM,0);
        UserList list = initEmptyList();
        char *login = getCleanBuffer(1);
#ifdef _WIN32
    MessageCollection *collection = &initEmptyCollection();
#elif __linux
    MessageCollection *collection = initEmptyCollectionPointer();
#endif


    if(s == INVALID_SOCKET)
    {
            WriteFailNetworkEvent("bad socket");
        //printf("bad socket\n");
        return false;
    }
    struct sockaddr_in servaddr,cliaddr;
       //servaddr.sin_family = AF_INET;
       //servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
       //servaddr.sin_port=htons(DEFAULT_PORT);
       servaddr = createSendAddr();

       //bind(s,(struct sockaddr *)&adr,addr_len);

    int send_size = 0;
       int rec_size = 0;
    bool isOnWork = true; //client works - listen from server
       char * receivebuffer = getCleanBuffer(MAX_PACKAGE_SIZE);

       SendMetaInfo info = {login,&list,&s,collection,CurrentRegime};
       #ifdef _WIN32
            CreateThread(NULL,0,&InputText, &info,0,NULL);
       #elif __linux
         pthread_t sendthread;
         pthread_create(&sendthread,NULL,InputText,reinterpret_cast<void*>(&info));
         pthread_join(sendthread,NULL);
       #endif

    CommandStructure EchoRequestCmd = GenerateEchoRequestCommand(); //client sends echo request
    PackageStructure package_echo_req = CreatePackage(EchoRequestCmd);
    char *senddata = GeneratePackageData(package_echo_req);
       send_size = sendDataUDP(s,servaddr,senddata);
       if(send_size < 0)
       {
            printf("fail to send echo request\n");
       }
       if(send_size > 0)
       {
            printf("send echo request\n");
       }
       printf("wait server answers\n");
    socklen_t len = sizeof(cliaddr);
    while(isOnWork) // client listen
```

```
    {

            while (recvfrom(s,receivebuffer,MAX_PACKAGE_SIZE,0,(struct sockaddr *)&cliaddr,&len)
> 0)
            {
                    CommandStructure cmd = Parse(receivebuffer);
                    if(cmd.CommandId == IDs::ECHO_ANSWER)
                    {
                            printf("receive echo answer. server online\n");
                            char *inplogin = inputLogin(); //client inputs login
                            memcpy(login,inplogin,strlen(inplogin));
                            CommandStructure LoginRequestCmd = GenerateLoginRequestCom-
mand(inplogin); //command = type+data
                            PackageStructure package_login_req = CreatePackage(LoginRequestCmd);
                            char *sendlogindata = GeneratePackageData(package_login_req); //convert
(6 0 1 -> 006001)
                            send_size = sendDataUDP(s,servaddr,sendlogindata);
                            if(send_size < 0)
                            {
                                    printf("fail to send login request\n");
                            }
                            if(send_size > 0)
                            {
                                    printf("send login request\n");
                            }
                            //free(sendlogindata);
                    }
                    if(cmd.CommandId == IDs::LOGIN_ANSWER)
                    {
                            if(cmd.data[0] == '1')
                            {
                                    printf("login accepted\n");
                                    CommandStructure UsersRequestCmd = GenerateUsersRequestCom-
mand();
                                    PackageStructure package_users_req = CreatePack-
age(UsersRequestCmd);
                                    char *sendusersdata = GeneratePackageData(package_users_req);
                                    send_size =sendDataUDP(s,servaddr,sendusersdata);
                                    if(send_size < 0)
                                    {
                                            printf("fail to send users request\n");
                                    }
                                    if(send_size > 0)
                                    {
                                            printf("send users request\n");
                                    }

                            }
                            else
                            {
                                    printf("login rejected. Try again\n");
                                    char *inplogin_r = inputLogin(); //client inputs login
                                    memcpy(login,inplogin_r,strlen(inplogin_r));
                                    CommandStructure LoginRequestCmd = GenerateLoginRequestCom-
mand(login); //command = type+data
                                    PackageStructure package_login_req = CreatePack-
age(LoginRequestCmd);
                                    char *sendlogindata = GeneratePackageData(package_login_req);
//convert (6 0 1 -> 006001)
                                    send_size = sendDataUDP(s,servaddr,sendlogindata);
                                    if(send_size < 0)
                                    {
                                            printf("fail to send login request\n");
                                    }
```

```
                                if(send_size > 0)
                                {
                                        printf("send login request\n");
                                }
                        }
                }
                if(cmd.CommandId == IDs::UNAUTHORIZED_ACTION)
                {
                        printf("unauthorized action answer\n");
                }
                if(cmd.CommandId == IDs::USERS_ANSWER)
                {
                        printf("receive users answer\n");
                        list = ParseUsers(cmd.data);
                }
                if(cmd.CommandId == IDs::MESSAGE_SEND)
                {
                        MessageStructure message =  parseData(cmd.data);
                        AddNewUnReadedMessage(collection,message);
                }

                if(cmd.CommandId == IDs::ECHO_REQUEST)
                {
                        CommandStructure EchoAnswerCmd = GenerateEchoAnswerCommand(); //client
sends echo answer
                        PackageStructure package_echo_ans = CreatePackage(EchoAnswerCmd);
                        char *senddata = GeneratePackageData(package_echo_ans);
                        send_size = sendDataUDP(s,servaddr,senddata);
                }
                if(cmd.CommandId == IDs::QUIT_REQUEST)
                {
                        WriteFailEvent("Quit command: close app");
                        closeSocket(s);
                        exit(2);
                }
                free(receivebuffer);
                receivebuffer = getCleanBuffer(MAX_PACKAGE_SIZE);
         }
    }
        return true;
}

void closeSocket(mysocket socket)
{
#ifdef _WIN32
        shutdown(socket,SD_BOTH);
        closesocket(socket);
#elif __linux
        shutdown(socket,SHUT_RDWR);
        close(socket);
#endif
}
```