Санкт-Петербургский политехнический университет

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторным работам

Курс: «Сети ЭВМ и телекоммуникации»

Тема: «Программирование сокетов протоколов TCP и UDP»

Выполнил: студент группы 43501/3

БояркинНикитаСергеевич

Подпись: _____

Принял: Зозуля Алексей Викторович

Подпись: _____

Дата: «__» _____ 2016 г.

Санкт-Петербург

2016

Оглавление

Цель работы

Ознакомиться с принципами программированиясобственных протоколов, созданных на основе TCPи UDP.

Программа работы

TCP:

1. Реализация простейшего TCP сервера и клиента на ОС Linux и Windows соответственно
2. Реализация многопоточного обслуживания клиентов на сервере
3. Реализация собственного протокола на основе TCP для индивидуального задания
4. Реализация синхронизации, с помощью мьютексов

UDP:

1. Модификация сервера и клиента для протокола UDP на ОС Windows и Linux соответственно
2. Обеспечение надежности протокола UDP, посредством нумерации пакетов и посылки ответов

Индивидуальное задание

Разработать распределенную систему, состоящую из приложений клиента и сервера, для распределенного оповещения о событиях по принципу подписки. Подразумевается разовая или многократная генерация событий сервером (с некоторым заданным интервалом), с информированием подписанных клиентов об этих событиях.Информационная система должна обеспечивать параллельную работу нескольких клиентов.

Протокол оповещения о событиях и подписки на них

Форматы команд

Для получения или изменения информации на сервере клиент или администратор сервера посылают текстовые команды. Набор команд для клиента или администратора различен, однаковсе они должны удовлетворять следующим требованиям:

1. Полная длина команды не превышает 1000 символов
2. Длина каждого аргумента команды не превышает 30 символов
3. Регистр не имеет значения, все переводится в нижний регистр
4. Множественные пробелы в команде воспринимаются как один

Если команда не удовлетворяет требованиям, то выводится/отправляется сообщение об ошибке, если команда была успешно выполнена, также выводится/отправляется сообщение об успешной операции.

Команды оперируют некоторыми сущностями, которые тоже имеют ограничения:

Мнемоника	Описание	Ограничения
<username>	Имя пользователя. В большинстве случаев может быть заменено уникальным идентификатором пользователя.	ASCIIсимволы в диапазоне [a-z] и [1-9].
<#UID>	Уникальный идентификатор пользователя.	Всегда начинается с символа “#”. Идентификатор должен быть неотрицательным числом и содержаться в таблице пользователей.
<eventname>	Название события. В большинстве случаев может быть заменено уникальным идентификатором события.	ASCIIсимволы в диапазоне [a-z] и [1-9].
<#EID>	Уникальный идентификатор события.	Всегда начинается с символа “#”. Идентификатор должен быть неотрицательным числом и содержаться в таблице событий.
<password>	Пароль пользователя. Используется для регистрации и подключения клиента.	ASCIIсимволы в диапазоне [a-z] и [1-9].
<filename>	Название файла для сохранения/восстановления набора пользователей/событий/подписок. По умолчанию имеет значение “server.data”.	UTF-8 символыза исключением \/:*?”<>|
<timestamp>	Момент начала события.	Временная метка в формате dd/mm/yyyy|hh:mm:ss Момент должен быть позже текущей даты и времени.
<period>	Период события в секундах.	Период должен быть целым числом не менее 10 и не более 864000 (от 10 секунд до 10 суток).

Список команд, которыми оперирует администратор сервера:

Команды состояния сервера	
filename <filename>	Установка текущего имени файла (по умолчанию “server.data”).
save	Сохранение всех таблиц сервера в файл.
load	Загрузкавсехтаблицсервераизфайла. При запуске сервера вызывается эта функция, поэтому в каталоге с программой сервера должен быть файл “server.data” с набором значений.
exit	Завершает работу сервера, закрывает

	все соединения и сохраняет все таблицы сервера в файл “server.backup”.
Командыполученияинформации	
help	Получение информации о всех доступных командах.
info user <username | UID>	Получение информации о конкретном присоединившихся пользователе.
info users	Получение информации о всех присоединившихся пользователях.
info accounts	Получение информации о всех существующих пользователях.
info events	Получение информации о всех событиях.
info subscriptions	Получение информации о всех подписках.
Команды работы с пользователями	
register <username><password>	Создание нового пользователя.
detach <username | UID>	Принудительное отсоединение присоединившегося пользователя.
delete <username>	Удаление пользователя (и отсоединение, если он присоединился).
Команды работы с событиями	
event create single <eventname><timestamp>	Создание нового однократного события.
event create multi <eventname><timestamp><period>	Создание периодического события.
event drop <eventname | EID>	Удаление события.
event subscribe <eventname | EID><username | UID>	Принудительная подписка пользователя на событие.
event unsubscribe <eventname | EID><username | UID>	Принудительная отписка пользователя от события.
event notify <eventname | EID>	Уведомление всех подписавшихся пользователей о событии.

Список команд, которыми оперирует клиент:

Команды работы с сервером	
connect <username><password>	Команда подсоединения к серверу, без этой команды большинство последующих не будут работать.
exit	Полное отключение от сервера и завершение работы программы. Работает без команды connect.
Командыполученияинформации	
help	Получение информации о всех доступных командах.Работает без команды connect.

infoself	Получение информации о текущем пользователе.
infoevents	Получение информации о всех событиях.
Команды работы с пользователями	
register <username><password>	Создание нового пользователя. Работает без команды connect.
Команды работы с событиями	
event create single <eventname><timestamp>	Создание нового однократного события.
event create multi <eventname><timestamp><period>	Создание периодического события.
event drop <eventname | EID>	Удаление события.
event subscribe <eventname | EID>	Подписка текущего пользователя на событие.
event unsubscribe <eventname | EID>	Отписка текущего пользователя от события.

Подключение к серверу, аргументы командной строки

Сервер использует порт 65100. Клиент по умолчанию подключается к 127.0.0.1:65100, что обусловлено отладочными целями. Для того, чтобы задать адрес и порт подключения клиента используются аргументы командной строки. Первый аргумент принимает IP адрес, второй - порт.

Пример для ОС Linux:

```
./client 192.168.0.2 65100
```

Для того чтобы изменить имя файла для начальной загрузки (аналогкоманды filename), сервер принимает это имя в качестве аргумента командной строки.

ПримердляОСLinux:

```
./server filename.in
```

Формат загрузочного/сохраняемого файла

Серверная программа позволяет сохранять и загружать состояние всех коллекций в файл, в специальном формате. Рассмотрим формат загрузочного и сохраняемого файла:

<количество событий>
    <название события 1>
    <стартовая дата события 1 в миллисекундах>
    <период события 1 (если нулевой период то однократное событие)>
    <название события 2>
    <стартовая дата события 2 в миллисекундах>
    <период события 2 (если нулевой период то однократное событие)>
    …
<количество пользователей>
    <имя пользователя 1>
    <пароль пользователя 1>

<количество подписок пользователя 1>
<название события 1 пользователя 1>
<название события 2 пользователя 1>
…
<имя пользователя 2>
<пароль пользователя 2>
<количество подписок пользователя 2>
<название события 1 пользователя 2>
<название события 2 пользователя 2>
…
…

Описание и коды ошибок

Рассмотримошибкифункционированиясервера. Большинство из этих ошибок уведомляют о неправильном запуске/завершении/подключениях сервера и являются внештатными:

Исключение	Код	Описание
COULD_NOT_CREATE_SOCKET	0x1	Не удалось создатьсерверный сокет (ошибка функции socket).
COULD_NOT_BIND	0x2	Не удалось забиндить сокет (ошибка функции bind).Как правило означает, что на этом порте сервер уже работает.
COULD_NOT_SET_NON_BLOCKING	0x3	Не удалось установить флаг запрета блокировки (ошибка функций fcntlили ioctlsocket).
COULD_NOT_ACCEPT	0x4	Не удалось присоединить клиента (ошибка функции accept).
COULD_NOT_RECEIVE_MESSAGE	0x5	Неудалосьпринятьсообщение (ошибкафункцииrecv). Является штатной ситуацией и означает, что TCPклиент отсоединился от сервера.
COULD_NOT_SHUT_SOCKET_DOWN	0x6	Не удалось завершить работу сокета (ошибка функции shutdown).
COULD_NOT_CLOSE_SOCKET	0x7	Не удалось завершить закрыть сокет (ошибка функций closeили closesocket).
COULD_NOT_SEND_MESSAGE	0x8	Неудалосьпослатьсообщение. Используется только в UDP, если не приходит ответ от клиента.
COULD_NOT_STARTUP	0x9	Только для Windows (ошибка функции WSAStartup).
COULD_NOT_RESOLVE_ADDRESS	0xA	Только для Windows (ошибка функции getaddrinfo).

Рассмотрим ошибки серверного парсера. Эти ошибки возникают результате распознавания команды от администратора сервера или клиента:

Исключение	Код	Описание
COULD_NOT_RESOLVE_COMMAND	0x1	Не удалось распознать команду.
COULD_NOT_RESOLVE_ARGUMENT	0x2	Не удалось распознать аргумент команды.
ARGUMENT_LOGIC_ERROR	0x3	Ошибка в логике команды. Например, когда дата начала события уже давно завершилась.
COMMAND_OR_ARGUMENT_TOO_LONG	0x4	Слишком длинная команда или аргумент.

Рассмотрим ошибки серверного контроллера. Эти ошибки возникают при выполнении команд. Они обычно связаны с доступом к коллекциями ошибками логики:

Исключение	Код	Описание
COULD_NOT_OPEN_FILE	0x1	Не удалось открыть файл для чтения или записи.
COULD_NOT_PARSE_FILE	0x2	Не удалось распарсить загружаемый файл (ошибка функции load).
USER_IS_ALREADY_EXISTS	0x3	Пользователь уже содержится в коллекции.
USER_IS_NOT_EXISTS	0x4	Пользователь не содержится в коллекции.
USER_IS_NOT_CONNECTED_YET	0x5	Пользователь еще не подключился (не использовал функциюconnect).
USER_IS_ALREADY_CONNECTED	0x6	Пользователь не может быть подключен повторно.
EVENT_IS_ALREADY_EXISTS	0x7	Событие уже содержится в коллекции.
EVENT_IS_NOT_EXISTS	0x8	Событие не содержится в коллекции.
COULD_NOT_GET_CLIENT_BY_SOCKET	0x9	Не удалось получить информацию о клиенте (IP&port) по его сокету.
WRONG_PASSWORD	0xA	Введен неправильный пароль.

Рассмотрим ошибки функционирования клиента. Большинство из этих ошибок уведомляют о неправильном запуске/завершении/подключениях клиента и являются внештатными:

Исключение	Код	Описание
COULD_NOT_STARTUP	0x1	Только для Windows (ошибка функции WSAStartup).
COULD_NOT_RESOLVE_ADDRESS	0x2	Только для Windows (ошибка функции getaddrinfo).
COULD_NOT_CREATE_SOCKET	0x3	Не удалось создатьсерверный сокет (ошибка функции socket).
COULD_NOT_CREATE_CONNECTION	0x4	Не удалось установить соединение. Используется только в UDP, если не приходит ответ от сервера на сообщение о присоединении.
COULD_NOT_SEND_MESSAGE	0x5	Неудалосьпослатьсообщение. Используется только в UDP, если не приходит ответ от клиента.

COULD_NOT_RECEIVE_MESSAGE	0x6	Не удалось принять сообщение (ошибка функции recv). Является штатной ситуацией и означает, что TCP сервер отсоединил клиента или завершил работу.
COULD_NOT_SHUT_SOCKET_DOWN	0x7	Не удалось завершить работу сокета (ошибка функции shutdown).
COULD_NOT_CLOSE_SOCKET	0x8	Не удалось завершить закрыть сокет (ошибка функций closeили closesocket).

Реализация программы

Структура проекта

Для реализации клиента и сервера была использована среда разработки CLion, которая удовлетворяет всем требованиям кроссплатформенной разработки на языке C++. Кроме того, в ней поддерживается возможность параллельной отладки двух проектов одновременно, что очень удобно для отладки клиент-серверного соединения.

Учитывая кроссплатформенные возможности среды разработки, было принято решение использовать один проект для клиента и один клиент для сервера и с помощью директив препроцессора разрабатывать кроссплатформенный код. TCP и UDP реализации тоже разграничиваются директивами препроцессора (см. Приложение 1).

Сетевая частьTCP

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

На сервере, в первую очередь, происходит инициализацияWinSock (на Windows), создание сокета (функция socket), привязка сокета к конкретному адресу (функция bind), подготовка сокета к принятию сообщений (функция listen), установка флага запрета блокировки (функция fcntlили ioctlsocket). Реализация инициализации сервера представлена в Приложении 2.

После этого ожидаем подключения клиентов в бесконечном цикле, с помощью функции accept. Если функция возвращает положительное значение, которое является клиентским сокетом, то создаем новый поток, в котором обрабатываем клиентские сообщения. Реализация подключения клиентов представлена в Приложении 3.

Клиентский поток вызывает функцию считывания фиксированного количества символов в бесконечном цикле. Если функция не вернула исключение, то посылаем команду на обработку, в противном случае это обозначает отключение клиента. Также отключение клиента может быть произведено извне обработчика клиентского потока, посредством закрытия клиентского сокета (функция считывания в этом случае сразу же вернет исключение). Реализация клиентского потока представлена в Приложении 4.

Функция считывания фиксированного количества символов - это оболочка для функции recv. Считывание происходит по одному символу и записывается в результирующую строку. Если соединение с клиентом разорвано, то функция возвращает исключение.Реализация функции считывания фиксированного числа символов представлена в Приложении 5.

Закрытие всех сокетов и как следствие завершение всех клиентских потоков производится в деструкторе серверного класса. Реализация деструктора представлена в Приложении 6.

Клиентские реализации вышеописанных функций концептуально не отличаются от серверных, поэтому не приводятся в приложениях.

Сетевая часть UDP

Клиентское приложение в UDP не может быть заменено сторонним приложением по типу telnet, потому что используется нумерация пакетов и посылка ответов.

Реализация сетевой части UDPсервера похожа на TCP, за исключением следующих отличий: функция создания сокета выполняется с параметрами SOCK_DGRAM, IPPROTO_UDP; отсутствует функция подготовки к принятию сообщений listen; отсутствует функцияустановления соединения accept, вместо нее используется функция recvfromи создание клиентского сокета функцией socket; вместо функций sendи recvиспользуются функции sendtoи recvfrom с явным указанием адресной структуры.

Реализации инициализации сервера и подключения новых клиентов представлены в Приложениях 7 и 8 соответственно.

Так как протокол UDPненадежный и без установления соединения некоторые пакеты могут затеряться в сети. Во избежание этого была реализована нумерация пакетов и посылка ответных пакетов:

Пакет	Ответ	Описание
@S0@@A	@R0	Запрос на присоединение к серверу.
@S1@@D	@R1	Запрос на отсоединение от сервера.
@S2@@C	@R2	Пакет проверки соединения.
@SN@MESSAGE	@RN	Пакет с сообщением, номер пакета N≥3.

Если ответ на пакет не приходит в течение определенного времени, то пакет посылается еще 4 раза. Если ни один ответ на этот пакет не пришел за это время, сообщение считается потерянным. Реализации функций отправки и приема сообщений представлены в Приложениях 9 и 10 соответственно.

Также в UDPотсутствует установление соединения, поэтому невозможно понять отключился ли клиент или сервер. Для проверки соединения был созданпоток, который с определенной периодичностью отправляет пакет проверки соединения всем клиентам. Если ответ не был получен, то клиент отключается. Реализация потока проверки соединения представлена в Приложении 11.

Клиентские функции отправки, приема сообщений и потока проверки соединения практически абсолютно симметричны серверным, поэтому не приводятся в приложениях.

Логика протокола

Парсер команд и серверный контроллер находятся только в серверном приложении, и их реализация одинакова и для TCP, и для UDP.

Класс ServerController является внутренним классом класса Server. Методы класса ServerController позволяют осуществлять полный контроль над коллекциями, методами и другими составляющими класса Server. Когда приходит команда от клиента создается экземпляр класса ClientCommand, которому передается указатель на экземпляр класса ServerController. Таким образом ClientCommand может вызывать методы реализующие действия команд.

Когда команду вводит администратор сервера, создается экземпляр класса ServerCommand. Классы ClientCommand и ServerCommand унаследованы от класса Command, что позволяет им

использовать общие функции, такие как парсер команды, функции проверки на допустимость аргументов.

СамиклассыClientCommandиServerCommand распознают действие команды, вызывают функции проверки на допустимость аргументов, и если все верно, то вызывают соответствующую функцию на указателе класса ServerController, например, удаление пользователя.

Сигнатура класса ServerController представлена в Приложении 12. Реализация методов класса ServerController слишком большая для отчета, поэтому не включена в приложения.

РеализациюклассовCommand, ServerCommandиClientCommand представлены в Приложениях 13, 14, 15 соответственно.

Для реализации уведомлений о наступлении событий был разработан поток с таймером и функция обновления для события. Как только событие наступает, посылается уведомление и вычисляется следующее время события для периодических событий и удаляется для однократных событий. Реализация таймера представлена в Приложении 16.

Тестирование

Тестирование проводилось наоперационных системах Windows 10 и Ubuntu 16.04 для сервера и клиента соответственно.Были проверены все команды, многопоточность, уведомления, UDP-пакеты, отключение от сервера по таймауту, правильное завершение всех потоков.

В этом пункте были приведены скриншоты некоторых экспериментов:

Команда helpна сервере:

```
D:\afiles\student\7\networks\result\windowsTCP>server.exe
Socket has been successfully created.
Socket has been successfully bind.
Listen socket.
help

COMMAND                                                                 DESCRIPTION

Server state commands:

filename <filename^[\/:*?"<>|]>                                         Set current filename (default "server.data").
save                                                                    Save program data into current file.
load                                                                    Load program data from current file.
exit                                                                    Close all connections and save backup data (into file "server.backup").

Info commands:

info user <login[a-z][0-9] | #UID[0-9]>                                 Print info about connected user.
info users                                                              Print info about all connected users.
info accounts                                                           Print info about all existing accounts.
info events                                                             Print info about all existing events.
info subscriptions                                                      Print info about all existing subscriptions.

Account commands:

register <user[a-z][0-9]> <password[a-z][0-9]>                          Register new user in accounts table.
detach <user[a-z][0-9] | #UID[0-9]>                                     Disconnect user from server.
delete <user[a-z][0-9]>                                                 Detach user and delete him from accounts table.

Event commands:

event create single <event[a-z][0-9]> <[dd/mm/yyyy|hh:mm:ss]>           Create single event.
event create multi <event[a-z][0-9]> <[dd/mm/yyyy|hh:mm:ss]> <period[int]>  Create repeating event with period in seconds.
event drop <event[a-z][0-9] | #EID[0-9]>                                Remove event by name or id.
event subscribe <event[a-z][0-9] | #EID[0-9]> <user[a-z][0-9] | #UID[0-9]>  Sign user on event.
event unsubscribe <event[a-z][0-9] | #EID[0-9]> <user[a-z][0-9] | #UID[0-9]> Unsubscribe user from event.
event notify <event[a-z][0-9] | #EID[0-9]>                              Notify all subscribers about the event immediately.

Command has been successfully executed.
_
```

Команда help на клиенте:

```
Connect commands:

connect <user[a-z][0-9]> <password[a-z][0-9]>                              Connect to server by username and p
assword.
exit                                                                       Disconnect from server and remove s
ession.

Info commands:

info self                                                                  Print info about current connected
user.
info events                                                                Print info about all existing event
s.

Account commands:

register <user[a-z][0-9]> <password[a-z][0-9]>                             Register new user in accounts table
.

Event commands:

event create single <event[a-z][0-9]> <[dd/mm/yyyy|hh:mm:ss]>              Create single event.

event create multi <event[a-z][0-9]> <[dd/mm/yyyy|hh:mm:ss]> <period[int]>  Create repeating event with period
in seconds.
event drop <event[a-z][0-9] | #EID[0-9]>                                   Remove event by name or id.

event subscribe <event[a-z][0-9] | #EID[0-9]>                              Sign current connected user on even
t.
event unsubscribe <event[a-z][0-9] | #EID[0-9]>                            Unsubscribe current connected user
from event.

Command has been successfully executed.
```

Многопоточность на сервере:

```
D:\afiles\student\7\networks\result\windowsTCP>server.exe
Socket has been successfully created.
Socket has been successfully bind.
Listen socket.
Thread 0x0. Client has been connected.
Thread 0x1. Client has been connected.
Thread 0x2. Client has been connected.
```

Уведомления о наступлении события (event4 каждые 20 сек, event5 каждые 30 сек):

```
connect u1 p1
Command has been successfully executed.
20:23:39 Notify about the event "event4".
20:23:59 Notify about the event "event4".
20:23:59 Notify about the event "event5".
20:24:19 Notify about the event "event4".
20:24:29 Notify about the event "event5".
```

Команда принудительного отсоединения клиента detachco стороны сервера:

```
D:\afiles\student\7\networks\result\windowsTCP>server.exe
Socket has been successfully created.
Socket has been successfully bind.
Listen socket.
Thread 0x0. Client has been connected.
detach #0
User isn't connected yet.
detach #0
Thread 0x0. Socket is down.
Thread 0x0. Socket is closed.
Thread 0x0. Client disconnected.
Command has been successfully executed.
```

Команда принудительного отсоединения клиента detachco стороны клиента:

```
connect u1 p1
Command has been successfully executed.
20:23:39 Notify about the event "event4".
20:23:59 Notify about the event "event4".
20:23:59 Notify about the event "event5".
20:24:19 Notify about the event "event4".
20:24:29 Notify about the event "event5".
20:24:39 Notify about the event "event4".
20:24:59 Notify about the event "event4".
20:24:59 Notify about the event "event5".
Socket is closed.
nikita@nikita-pc:~/Desktop/ezsem/networks/result/linuxTCP$
```

Команда завершения работы сервераexit (Thread 0x-1 это основной поток с серверным сокетом):

```
D:\afiles\student\7\networks\result\windowsTCP>server.exe
Socket has been successfully created.
Socket has been successfully bind.
Listen socket.
Thread 0x0. Client has been connected.
exit
Command has been successfully executed.
Thread 0x0. Socket is down.
Thread 0x0. Socket is closed.
Thread 0x0. Client disconnected.
Thread 0x-1. Socket is down.
Thread 0x-1. Socket is closed.

D:\afiles\student\7\networks\result\windowsTCP>
```

Команда завершения работы клиента exit:

Все эксперименты завершились успешно.

Вывод

В ходе работы был реализован собственный протокол для оповещения о событиях и подписки на них на основе TCPи UDP. Протокол был реализован на языке C++ для операционных систем Windowsи Linuxпри помощи APIсокетов Беркли и Windows Sockets API.

Протоколы TCPи UDPимеют свои сильные и слабые стороны, поэтому не совсем очевидно какой из них лучше использовать для данной задачи. С одной стороны, надежность передачи данных это критично при уведомлении о событии. С другой стороны, при большом количестве подписчиков структура TCPне выдержала бы столь большой нагрузки. Хорошим решением я считаю компромисс: реализация протокола на основе UDP, однако с собственным механизмом обеспечения надежности (нумерацией пакетов, посылкой ответов, отключением по таймауту).

Приложения

Для упрощения представления результатов, нижепредставленный код был представлен только для ОС Windows.

Приложение 1, структура проекта

```
#ifndef NETWORKS_GLOBAL_H
#define NETWORKS_GLOBAL_H

// #define _LINUX_
#define _WIN_

#define _TCP_
// #define _UDP_

#endif //NETWORKS_GLOBAL_H
```

Приложение 2, инициализация сервера TCP

```
Server::Server(std::ostream* out, std::istream* in,
std::ostream* error, const char* port, const char* filename) throw(ServerException) {

    this->out = out;
    this->in = in;
    this->error = error;
    this->filename = std::string(filename);

    WSADATA wsaData;
    auto wsaStartup = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if(wsaStartup != 0)
        throw ServerException(COULD_NOT_STARTUP);

    struct addrinfo hints;
    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE;

    struct addrinfo *addressResult = nullptr;
    auto wsaAddress = getaddrinfo(NULL, port, &hints, &addressResult);
    if (wsaAddress != 0)
        throw ServerException(COULD_NOT_RESOLVE_ADDRESS);

    generalSocket = socket(addressResult->ai_family, addressResult->ai_socktype, addressResult-
>ai_protocol);
    if(generalSocket == INVALID_SOCKET)
        throw ServerException(COULD_NOT_CREATE_SOCKET);

    *this->out << "Socket has been successfully created." << std::endl;

    generalBind = bind(generalSocket, addressResult->ai_addr, (int) addressResult->ai_addrlen);
    if(generalBind == SOCKET_ERROR)
        throw ServerException(COULD_NOT_BIND);

    *this->out << "Socket has been successfully bind." << std::endl;

    listen(generalSocket, BACKLOG);

    *this->out << "Listen socket." << std::endl;

    unsigned int iMode = 1;
    generalFlags = ioctlsocket(generalSocket, 0x8004667E, &iMode);
    if(generalFlags == SOCKET_ERROR)
        throw ServerException(COULD_NOT_SET_NON_BLOCKING);
}
```

Приложение 3, подключение клиентов TCP

```
const void Server::start() throw(ServerException, ServerController::ControllerException) {
    this->events = std::map<int, Server::Event*>();
    this->users = std::map<int, Server::User*>();
    this->accounts = std::map<std::string, std::string>();
    this->timings = std::vector<std::pair<int, std::chrono::milliseconds>>();
    this->subscriptions = std::vector<std::pair<std::string, std::string>>();

    this->generalInterrupt = false;
    this->timerInterrupt = false;

    this->controller = new ServerController(this);
    this->controller->load();

    auto bindCommand = std::bind(&Server::commandThreadInitialize, std::placeholders::_1);
    commandThread = std::make_shared<std::thread>(bindCommand, this);

    while(!this->generalInterrupt) {
        auto clientAddress = new sockaddr_in;
        auto size = sizeof(struct sockaddr_in);

        auto clientSocket = accept(generalSocket, (sockaddr *) clientAddress, (socklen_t *) &size);

       if (clientSocket != INVALID_SOCKET)
            createClientThread(clientSocket, clientAddress);

        else
            delete clientAddress;
    }
}
```

Приложение 4, поток обработки клиентских сообщений TCP

```
void* Server::clientThreadInitialize(void *thisPtr, const int threadId, const SOCKET clientSocket) {

    auto serverPtr = ((Server*)thisPtr);

    bool lockOut, lockError;

    try {
        serverPtr->acceptClient(threadId, clientSocket);
    }
    catch (const ServerException& exception) {
        lockError = serverPtr->mutexError.try_lock();
        *serverPtr->error << "Thread 0x" << threadId << ". " << exception.what() << std::endl;
        if(lockError)
            serverPtr->mutexError.unlock();
    }
    try {
        serverPtr->removeClientThread(threadId);

        lockOut = serverPtr->mutexOut.try_lock();
        *serverPtr->out << "Thread 0x" << threadId << ". Client disconnected." << std::endl;
        if(lockOut)
            serverPtr->mutexOut.unlock();
    } catch (const ServerException& exception) {
        lockError = serverPtr->mutexError.try_lock();
        *serverPtr->error << "Thread 0x" << threadId << ". " << exception.what() << std::endl;
        if(lockError)
            serverPtr->mutexError.unlock();
    }

    return NULL;
}
```

```
const void Server::acceptClient(const int threadId, const SOCKET
clientSocket) throw(ServerException) {

    bool lockOut = this->mutexOut.try_lock();
    *this->out << "Thread 0x" << threadId << ". Client has been connected." << std::endl;
    if(lockOut)
        this->mutexOut.unlock();

    while(!this->generalInterrupt) {
        std::string message;
        try{
            message = readLine(threadId, clientSocket);
        }
        catch (const ServerException& exception) {
            if(exception.code() == COULD_NOT_RECEIVE_MESSAGE)
                break;
        }

        auto stream = new std::stringstream();
        try {
            ClientCommand(&message, this->controller, clientSocket).parseAndExecute(stream);
            *stream << "Command has been successfully executed." << std::endl;
        }
        catch(const Command::CommandException& exception) {
            *stream << exception.what() << std::endl;
        }
        catch(const Server::ServerController::ControllerException& exception) {
            *stream << exception.what() << std::endl;
        }
        catch(const std::exception& exception) {
            *stream << exception.what() << std::endl;
        }
        catch(...) {
            *stream << "Strange resolve command error." << std::endl;
        }
        writeLine(stream->str(), clientSocket);
  }
}
```

Приложение 5, функция считывания фиксированного числа символов TCP

```
const std::string Server::readLine(const int threadId, const SOCKET
socket) const throw(ServerException) {

    auto result = std::string();

    char resolvedSymbol = ' ';

    while(!this->generalInterrupt) {
        auto userFind = this->users.find(threadId);
        if(userFind == this->users.end() || userFind->second->clientInterrupt || userFind->second-
>serverInterrupt)
            throw ServerException(COULD_NOT_RECEIVE_MESSAGE);

        auto readSize = recv(socket, &resolvedSymbol, 1, EMPTY_FLAGS);
        if(readSize == 0)
            throw ServerException(COULD_NOT_RECEIVE_MESSAGE);
        else if(readSize < 0)
            continue;
        else if(result.size() > MESSAGE_SIZE || resolvedSymbol == '\n')
            break;
        else if(resolvedSymbol != '\r')
            result.push_back(resolvedSymbol);
    }

    return result;
}
```

Приложение 6, деструктор, освобождение ресурсов

```
Server::~Server() {
    stop();
}

const void Server::stop() throw(ServerException){
    this->generalInterrupt = true;
    this->timerInterrupt = true;

    bool tryLockArray[9];
    lockAll(tryLockArray);

    if(generalSocket != INVALID_SOCKET) {
        unsigned int iMode = 0;
        auto ioctlResult = ioctlsocket(generalSocket, 0x8004667E, &iMode);
        if(ioctlResult == SOCKET_ERROR)
            throw ServerException(COULD_NOT_SET_NON_BLOCKING);
    }

    if(timerThread != nullptr && timerThread.get()->joinable())
        timerThread.get()->join();

    if(commandThread != nullptr && commandThread.get()->joinable())
        commandThread.get()->join();

    for (auto &current: this->users) {
        current.second->clientInterrupt = false;
        current.second->serverInterrupt = true;

        if (current.second->thread != nullptr && current.second->thread->joinable())
            current.second->thread->join();
    }

    clearSocket(-1, generalSocket);
    WSACleanup();

    unlockAll(tryLockArray);

    this->controller->finit(BACKUP_FILENAME);
    try { this->controller->save(); }
    catch (const ServerController::ControllerException& exception) {  }
}

const void Server::clearSocket(const int threadId, const SOCKET socket) throw(ServerException) {

    if(socket == INVALID_SOCKET)
        return;

    if(threadId != -1) {
        auto socketShutdown = shutdown(socket, SD_BOTH);
        if(socketShutdown == SOCKET_ERROR)
            throw ServerException(COULD_NOT_SHUT_SOCKET_DOWN);
    }

    bool lockOut = this->mutexOut.try_lock();
    *this->out <<  "Thread 0x" << threadId << ". Socket is down." << std::endl;
    if(lockOut)
        this->mutexOut.unlock();

    auto socketClose = closesocket(socket);
    if(socketClose == SOCKET_ERROR)
        throw ServerException(COULD_NOT_CLOSE_SOCKET);

    lockOut = this->mutexOut.try_lock();
    *this->out <<  "Thread 0x" << threadId << ". Socket is closed." << std::endl;
    if(lockOut)
        this->mutexOut.unlock();
}
```

Приложение 7, инициализация сервера UDP

```
Server::Server(std::ostream* out, std::istream* in,
std::ostream* error, const uint16_t port, const char* filename) throw(ServerException) {

    this->out = out;
    this->in = in;
    this->error = error;
    this->filename = std::string(filename);

    WSADATA wsaData;
    auto wsaStartup = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if(wsaStartup != 0)
        throw ServerException(COULD_NOT_STARTUP);

    generalSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if(generalSocket == INVALID_SOCKET)
        throw ServerException(COULD_NOT_CREATE_SOCKET);

    *this->out << "Socket has been successfully created." << std::endl;

    struct sockaddr_in serverAddress;
    ZeroMemory(&serverAddress, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(port);

    generalBind = bind(generalSocket, (struct sockaddr *) &serverAddress, sizeof(serverAddress));
    if(generalBind == SOCKET_ERROR)
        throw ServerException(COULD_NOT_BIND);

    *this->out << "Socket has been successfully bind." << std::endl;

    unsigned int iMode = 1;
    generalFlags = ioctlsocket(generalSocket, 0x8004667E, &iMode);
    if(generalFlags == SOCKET_ERROR)
        throw ServerException(COULD_NOT_SET_NON_BLOCKING);
}
```

Приложение 8, подключение клиентов UDP

```
const void Server::start() throw(ServerException, ServerController::ControllerException) {
    this->events = std::map<int, Server::Event*>();
    this->users = std::map<int, Server::User*>();
    this->accounts = std::map<std::string, std::string>();
    this->timings = std::vector<std::pair<int, std::chrono::milliseconds>>();
    this->subscriptions = std::vector<std::pair<std::string, std::string>>();

    this->generalInterrupt = false;
    this->timerInterrupt = false;
    this->checkInterrupt = false;

    this->controller = new ServerController(this);
    this->controller->load();

    auto bindCommand = std::bind(&Server::commandThreadInitialize, std::placeholders::_1);
    commandThread = std::make_shared<std::thread>(bindCommand, this);

    const auto attachMessage = std::string(SEND_STRING) + "0@" +  std::string(ATTACH_STRING);
    char connectionBuffer[MESSAGE_SIZE];

    while(!this->generalInterrupt) {
        auto clientAddress = new sockaddr_in;
        auto size = sizeof(struct sockaddr_in);

        memset(connectionBuffer, 0, sizeof(connectionBuffer));
        auto clientSocket = recvfrom(generalSocket, connectionBuffer, MESSAGE_SIZE,
```

```
EMPTY_FLAGS, (sockaddr *) clientAddress, (socklen_t *)&size);

       if (clientSocket != INVALID_SOCKET) {
           auto connectionString = std::string(connectionBuffer);

           std::remove(connectionString.begin(), connectionString.end(), '\r');
           if(connectionString.back() == '\n')
               connectionString.pop_back();

           if(connectionString == attachMessage) {
               auto resultSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
               auto responseString = std::string(RESPONSE_STRING) + "0";
               sendto(resultSocket, responseString.data(), MESSAGE_SIZE,
EMPTY_FLAGS, (struct sockaddr *) clientAddress, sizeof(structsockaddr_in));
               createClientThread(resultSocket, clientAddress);
           }
       }
       else
           delete clientAddress;
   }
}
```

Приложение 9, функция отправки сообщений UDP

```
const void Server::writeLine(const std::string& message, const SOCKET
socket, const sockaddr_in* clientAddress,

 int* currentPackageNumber, int* clientPackageNumber, int* progressivePackageNumber, bool* responseArri
ved,
                            const bool special, const bool waitThreadRead) throw(ServerException) {

    if(message.empty())
        return;

    auto result = std::string(message);

    if(special) {
        *currentPackageNumber = (message == std::string(ATTACH_STRING)) ? 0 :
                                ((message == std::string(DETACH_STRING)) ? 1 :
                                 ((message == std::string(CHECK_STRING)) ? 2 : -1));

        if(*currentPackageNumber == -1)
            return;
    }
    else {
        *currentPackageNumber = *progressivePackageNumber;
        ++(*progressivePackageNumber);
    }

    std::remove(result.begin(), result.end(), '\r');
    if(result.back() != '\n')
        result.push_back('\n');

    result.insert(0, "@");
    result.insert(0, std::to_string(*currentPackageNumber));
    result.insert(0, SEND_STRING);

    *responseArrived = false;

    for(auto tryIndex = 0; tryIndex < TRIES_COUNT; ++tryIndex) {
        sendto(socket, result.data(), MESSAGE_SIZE,
EMPTY_FLAGS, (struct sockaddr *) clientAddress, sizeof(struct sockaddr_in));

        if(!waitThreadRead && readLine(socket, clientAddress, currentPackageNumber,
clientPackageNumber, responseArrived, true) ==RESPONSE_STRING)
            return;
        else {
            auto iterationsWait = ITERATIONS_COUNT;
```

```
            while (!(*responseArrived) && iterationsWait != 0)
                --iterationsWait;

            if (*responseArrived)
                return;
        }
    }

    throw ServerException(COULD_NOT_SEND_MESSAGE);
}
```

Приложение 10, функция чтения сообщений UDP

```
const std::string Server::readLine(const SOCKET socket, const sockaddr_in* clientAddress,

 int* currentPackageNumber, int* clientPackageNumber, bool* responseArrived, bool responseExecutor)thro
w(ServerException) {

    auto result = std::string();

    auto input = new char[MESSAGE_SIZE];
    while(!this->generalInterrupt && result.empty()) {
        memset(input, 0, MESSAGE_SIZE);

        int iterationIndex = 0;
        while(!this->generalInterrupt) {
            if(responseExecutor && iterationIndex == 1) {
                result.clear();
                break;
            }

            ++iterationIndex;

            ZeroMemory(input, sizeof(input));

            auto size = sizeof(struct sockaddr_in);
            recvfrom(socket, input, MESSAGE_SIZE,
EMPTY_FLAGS, (struct sockaddr *) clientAddress, (socklen_t *) &size);
            result = input;

            auto find = result.find_last_of('\n');
            if(find != std::string::npos)
                result.erase(find);

            std::remove(result.begin(), result.end(), '\r');

            if(result.size() < 3)
                continue;

            auto prefix = result.substr(0, 2);

            if(prefix == std::string(SEND_STRING)) {
                auto response = result.substr(2, result.size() - 2);
                find = response.find_first_of('@', 0);

                if(find == std::string::npos || find >= response.size() - 1)
                    continue;

                auto stream = std::stringstream(response.substr(0, find));

                int packageNumber;
                stream >> packageNumber;
                if(stream.fail())
                    continue;

                if(packageNumber == 0)
                    continue;
```

```
                bool continueNeeded = false;
                if(packageNumber >= *clientPackageNumber)
                    *clientPackageNumber = packageNumber + 1;
                else
                    continueNeeded = true;

                auto message = std::string(result);

                result = response.substr(find + 1, response.size() - find - 1);

                response = std::string(RESPONSE_STRING) + std::to_string(packageNumber);
                sendto(socket, response.data(), MESSAGE_SIZE,
EMPTY_FLAGS, (struct sockaddr *) clientAddress,  sizeof(struct sockaddr_in));

                if(packageNumber == 2)
                    continue;

                if(packageNumber == 1) {
                    if(result == std::string(DETACH_STRING)) {
                        sendto(socket, message.data(), MESSAGE_SIZE,
EMPTY_FLAGS, (struct sockaddr *) clientAddress, sizeof(structsockaddr_in));
                        throw ServerException(COULD_NOT_RECEIVE_MESSAGE);
                    }
                    else
                        continue;
                }

                if(continueNeeded)
                    continue;

                break;
            }
            else if(prefix == std::string(RESPONSE_STRING)) {
                auto stream = std::stringstream(result.substr(2, result.size() - 2));

                int packageNumber;
                stream >> packageNumber;
                if(stream.fail())
                    continue;

                if(packageNumber != *currentPackageNumber)
                    continue;

                *responseArrived = true;

                if(responseExecutor) {
                    result = RESPONSE_STRING;
                    break;
                }

                continue;
            }
            else
                continue;
        }
    }

    delete[] input;
    return result;
}
```

Приложение 11, проверка соединения, отключение по таймауту UDP

```
void* Server::checkThreadInitialize(void *thisPtr) {
    ((Server*)thisPtr)->checkExecutor();
    return NULL;
}
```

```
const void Server::checkExecutor() {
    while(!this->generalInterrupt && !this->checkInterrupt) {
        Sleep((DWORD)(CHECK_INTERVAL * 1e3));
        bool lockUsers = this->mutexUsers.try_lock();

        std::vector<int> eraseUsers = std::vector<int>();

        for(auto& current: users) {
            try {
                writeLine(CHECK_STRING, current.second->socket, current.second->address,
                          &(current.second->currentPackageNumber), &(current.second-
>clientPackageNumber), &(current.second->progressivePackageNumber), &(current.second-
>responseArrived),
                          true, true);
            }
            catch (const ServerException& exception) {
                clearSocket(current.first, current.second->socket);
                current.second->thread->detach();
                eraseUsers.push_back(current.first);
            }
        }

        for(auto& current: eraseUsers)
            users.erase(current);

        eraseUsers.clear();

        if(lockUsers)
            this->mutexUsers.unlock();
    }
}
```

Приложение 12, сигнатура класса ServerController

```
class ServerController {
    private:
        Server* serverPtr;

    public:
        enum Error {
            COULD_NOT_OPEN_FILE = 0x1,
            COULD_NOT_PARSE_FILE = 0x2,
            USER_IS_ALREADY_EXISTS = 0x3,
            USER_IS_NOT_EXISTS = 0x4,
            USER_IS_NOT_CONNECTED_YET = 0x5,
            USER_IS_ALREADY_CONNECTED = 0x6,
            EVENT_IS_ALREADY_EXISTS = 0x7,
            EVENT_IS_NOT_EXISTS = 0x8,
            COULD_NOT_GET_CLIENT_INFO_BY_SOCKET = 0x9,
            WRONG_PASSWORD = 0xA,
        };

        class ControllerException: public std::exception {
        private:
            Error error;
        public:
            explicit ControllerException(const Error);
            const char* what() const noexcept override;
            const int code() const;
        };

        ServerController(Server* serverPtr);

        const void reg(const char* userName, const char* password) const throw(ControllerException);
        const void del(const char* userName) const throw(ControllerException);
        const void detach(const char* userName) const throw(ControllerException);

        const std::pair<std::string, std::string> getAddressInfoBySocket(const SOCKET
```

```
socket) const throw(ControllerException);
        const void close(const SOCKET socket) const throw(ControllerException);
        const void connect(const char* userName, const char* password, const SOCKET
clientSocket) const throw(ControllerException);
        const char* getUserNameBySocket(const SOCKET clientSocket) const throw(ControllerException);

        const void finit(const char* filename) const;
        const void save() const throw(ControllerException);
        const void load() const throw(ControllerException);
        const void exit() const;

  const void eventCreate(const char* eventName, const std::chrono::milliseconds& start, const std::chro
no::seconds& period) constthrow(ControllerException);
        const void eventDrop(const char* eventName) const throw(ControllerException);

  const void eventSubscribe(const char* eventName, const char* userName) const throw(ControllerExceptio
n);
        const void eventUnsubscribeAll(const char* eventName) const throw(ControllerException);

  const void eventUnsubscribe(const char* eventName, const char* userName) const throw(ControllerExcept
ion);
        const void eventNotify(const char *eventName) const;
        const void help(std::ostream* out) const;
        const void printSubscriptionsInfo() const;
        const void printSelfInfo(std::ostream* out, const char* userName) const;
        const void printUsersInfo() const;
        const void printEventsInfo(std::ostream* out) const;
        const void printAccountsInfo() const;

        const int getThreadIdByUserName(const char* userName) const throw(ControllerException);
        const char* getUserNameByThreadId(const int threadId) const throw(ControllerException);
        const int getEventIdByEventName(const char* eventName) const throw(ControllerException);
        const char* getEventNameByEventId(const int eventId) const throw(ControllerException);
};
```

Приложение 13, реализация класса Command

```
#include <algorithm>
#include <sstream>
#include <iomanip>
#include "../headers/Command.h"

Command::Command(std::string* expr,
Server::ServerController* controller) throw(std::invalid_argument) {
    if(expr == nullptr || controller == nullptr)
        throw std::invalid_argument("Pointer couldn't be null.");

    this->expr = expr;
    this->controller = controller;
}

Command::~Command() { }

const std::vector<std::string> Command::prepareCommand() const throw(CommandException){
    auto result = std::vector<std::string>();

    auto iterator = this->expr->begin();
    auto string = std::string();

    while(iterator != this->expr->end()) {
        if(*iterator == ' ' && !string.empty()){
            if(string.length() > MAX_LENGTH_OF_ARGUMENT)
                throw CommandException(COMMAND_OR_ARGUMENT_IS_TOO_LONG);
            std::transform(string.begin(), string.end(), string.begin(), ::tolower);
            result.push_back(string);
            string.clear();
        }
        else if(*iterator != ' ')
```

```
            string.push_back(*iterator);

        ++iterator;
    }

    if(!string.empty()) {
        if(string.length() > MAX_LENGTH_OF_ARGUMENT)
            throw CommandException(COMMAND_OR_ARGUMENT_IS_TOO_LONG);
        result.push_back(string);
    }

    return result;
}

char* Command::getNameFromCommand(const std::string& command, const bool isEvent) const throw(CommandEx
ception, Server::ServerController::ControllerException) {
    char* result;

    try {
        result = (isEvent) ? const_cast<char*>(this->controller-
>getEventNameByEventId(parseId(command))) :
                             const_cast<char*>(this->controller-
>getUserNameByThreadId(parseId(command)));
    }
    catch (const Server::ServerController::ControllerException& exception)
    { throw Server::ServerController::ControllerException(exception); }
    catch (const std::invalid_argument &exception)
    { result = const_cast<char*>(checkASCII(command).data()); }
    catch (...)
    { throw CommandException(COULD_NOT_RESOLVE_ARGUMENT); }

    return result;
}

const std::chrono::milliseconds Command::parseDate(const std::string &string) throw(CommandException) {
    std::tm time = {};
    std::stringstream stream(string);
    stream >> std::get_time(&time, "%d/%m/%Y|%H:%M:%S");

    if(stream.fail())
        throw CommandException(COULD_NOT_RESOLVE_ARGUMENT);


  auto result = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::from_t
ime_t(std::mktime(&time)).time_since_epoch());

  auto now = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().tim
e_since_epoch());
    auto buffer = EVENT_BUFFER;

    if(result <= now + std::chrono::seconds(buffer))
        throw CommandException(ARGUMENT_LOGIC_ERROR);

    return result;
}

const std::string& Command::checkASCII(const std::string &value) throw(CommandException) {
    for(auto& current: value)
        if(current < '0' || current > 'z' || (current < 'a' && current > '9'))
            throw CommandException(COULD_NOT_RESOLVE_ARGUMENT);

    return value;
}

const void Command::checkFilename(const std::string &filename) throw(CommandException) {
    if(filename.find_first_of("\\/:*?\"<>|") != std::string::npos)
        throw CommandException(COULD_NOT_RESOLVE_ARGUMENT);
}
```

```
const int Command::parseId(const std::string &stringId) throw(std::invalid_argument,
std::out_of_range) {
    if(stringId[0] != PREFIX)
        throw std::invalid_argument(stringId);

    if(stringId.size() < 2)
        throw std::out_of_range(stringId);

    auto string = stringId.substr(1, stringId.length());
    if(string.find_first_not_of("0123456789") != std::string::npos)
        throw std::out_of_range(stringId);

    int result;
    try { result = std::stoi(string); }
    catch(...) { throw std::out_of_range(stringId); }

    return result;
}

Command::CommandException::CommandException(const Error error) {
    this->error = error;
}

const char* Command::CommandException::what() const noexcept {
    switch(this->error){
        case COULD_NOT_RESOLVE_COMMAND:
            return "It's impossible to resolve command.";

        case COULD_NOT_RESOLVE_ARGUMENT:
            return "It's impossible to resolve argument.";

        case ARGUMENT_LOGIC_ERROR:
            return "Argument logic exception.";

        case COMMAND_OR_ARGUMENT_IS_TOO_LONG:
            return "It's impossible to resolve so long command or argument.";
    }

    return "Unknown exception.";
}

const int Command::CommandException::code() const { return this->error; }
```

Приложение 14, реализация класса ServerCommand

```
#include "../headers/ServerCommand.h"

ServerCommand::ServerCommand(std::string* expr, Server::ServerController* controller) : Command(expr,
controller) { }

ServerCommand::~ServerCommand() { }

const void ServerCommand::parseAndExecute() const throw(CommandException,
Server::ServerController::ControllerException) {
    auto commandPartition = prepareCommand();

    char* eventName;
    char* userName;
    std::chrono::milliseconds start;
    std::chrono::seconds period;

    switch(commandPartition.size()){
        case 1:
            if(commandPartition[0] == "help")
                this->controller->help(nullptr);
            else if(commandPartition[0] == "exit")
                this->controller->exit();
            else if(commandPartition[0] == "save")
```

```
                this->controller->save();
            else if(commandPartition[0] == "load")
                this->controller->load();
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 2:
            if(commandPartition[1] == "events" && commandPartition[0] == "info")
                this->controller->printEventsInfo(nullptr);
            else if(commandPartition[1] == "users" && commandPartition[0] == "info")
                this->controller->printUsersInfo();
            else if(commandPartition[1] == "accounts" && commandPartition[0] == "info")
                this->controller->printAccountsInfo();
            else if(commandPartition[1] == "subscriptions" && commandPartition[0] == "info")
                this->controller->printSubscriptionsInfo();
            else if(commandPartition[0] == "filename") {
                checkFilename(commandPartition[1]);
                this->controller->finit(commandPartition[1].data());
            }
            else if(commandPartition[0] == "detach") {
                userName = getNameFromCommand(commandPartition[1], false);
                this->controller->detach(userName);
            }
            else if(commandPartition[0] == "delete")
                this->controller->del(checkASCII(commandPartition[1]).data());
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 3:
            if(commandPartition[1] == "user" && commandPartition[0] == "info") {
                userName = getNameFromCommand(commandPartition[2], false);
                this->controller->printSelfInfo(nullptr, userName);
            }
            else if(commandPartition[0] == "register")
                this->controller->reg(checkASCII(commandPartition[1]).data(),
checkASCII(commandPartition[2]).data());

  else if(commandPartition[0] == "event" && (commandPartition[1] == "drop" || commandPartition[1] == "n
otify")) {
                eventName = getNameFromCommand(commandPartition[2], true);
                if(commandPartition[1] == "drop") this->controller->eventDrop(eventName);
                else this->controller->eventNotify(eventName);
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 4:

  if(commandPartition[0] == "event" && (commandPartition[1] == "subscribe" || commandPartition[1] == "u
nsubscribe")) {
                eventName = getNameFromCommand(commandPartition[2], true);
                userName = getNameFromCommand(commandPartition[3], false);
                if(commandPartition[1] == "subscribe") this->controller->eventSubscribe(eventName,
userName);
                else this->controller->eventUnsubscribe(eventName, userName);
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 5:

  if(commandPartition[0] == "event" && commandPartition[1] == "create" && commandPartition[2] == "singl
e") {
                eventName = const_cast<char*>(checkASCII(commandPartition[3]).data());
                start = parseDate(commandPartition[4]);
                this->controller->eventCreate(eventName, start, std::chrono::seconds(0));
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
```

```
            break;
        case 6:

  if(commandPartition[0] == "event" && commandPartition[1] == "create" && commandPartition[2] == "multi
") {
                eventName = const_cast<char*>(checkASCII(commandPartition[3]).data());
                start = parseDate(commandPartition[4]);
                try { period = std::chrono::seconds(std::stoi(commandPartition[5])); }
                catch(...) { throw CommandException(COULD_NOT_RESOLVE_ARGUMENT); }

  if(period < std::chrono::seconds(static_cast<int>(MIN_EVENT_PERIOD)) || period > std::chrono::seconds
(static_cast<int>(MAX_EVENT_PERIOD)))
                    throw CommandException(ARGUMENT_LOGIC_ERROR);
                this->controller->eventCreate(eventName, start, period);
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        default:
            throw CommandException(COULD_NOT_RESOLVE_COMMAND);
    }

}
```

Приложение 15, реализация класса ClientCommand

```
#include "../headers/ClientCommand.h"

#ifdef _LINUX_
ClientCommand::ClientCommand(std::string* expr,
Server::ServerController* controller, const int clientSocket)throw(Server::ServerController::Controller
Exception)  : Command(expr, controller){
#endif
#ifdef _WIN_
ClientCommand::ClientCommand(std::string* expr, Server::ServerController* controller, const SOCKET
clientSocket)throw(Server::ServerController::ControllerException)  : Command(expr, controller){
#endif
    this->clientSocket = clientSocket;
}

ClientCommand::~ClientCommand() { }

const void ClientCommand::parseAndExecute(std::ostream* out) const throw(CommandException,
Server::ServerController::ControllerException) {
    auto commandPartition = prepareCommand();

    char* eventName;
    char* userName;
    std::chrono::milliseconds start;
    std::chrono::seconds period;

    switch(commandPartition.size()) {
        case 1:
            if(commandPartition[0] == "help")
                this->controller->help(out);
            else if(commandPartition[0] == "exit")
                this->controller->close(this->clientSocket);
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 2:
            if(commandPartition[1] == "events" && commandPartition[0] == "info")
                this->controller->printEventsInfo(out);
            else if(commandPartition[1] == "self" && commandPartition[0] == "info")
                this->controller->printSelfInfo(out, this->controller->getUserNameBySocket(this-
>clientSocket));
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
```

```
            break;
        case 3:
            if(commandPartition[0] == "register")
                this->controller->reg(checkASCII(commandPartition[1]).data(),
checkASCII(commandPartition[2]).data());
            else if(commandPartition[0] == "connect")
                this->controller->connect(checkASCII(commandPartition[1]).data(),
checkASCII(commandPartition[2]).data(), this->clientSocket);
            else if(commandPartition[0] == "event" && commandPartition[1] == "drop") {
                this->controller->getUserNameBySocket(this->clientSocket);
                eventName = getNameFromCommand(commandPartition[2], true);
                this->controller->eventDrop(eventName);
            }

  else if(commandPartition[0] == "event" && (commandPartition[1] == "subscribe" || commandPartition[1]
== "unsubscribe")) {
                userName = const_cast<char*>(this->controller->getUserNameBySocket(this-
>clientSocket));
                eventName = getNameFromCommand(commandPartition[2], true);
                if(commandPartition[1] == "subscribe") this->controller->eventSubscribe(eventName,
userName);
                else this->controller->eventUnsubscribe(eventName, userName);
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 4:
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
        case 5:

  if(commandPartition[0] == "event" && commandPartition[1] == "create" && commandPartition[2] == "singl
e") {
                this->controller->getUserNameBySocket(this->clientSocket);
                eventName = const_cast<char*>(checkASCII(commandPartition[3]).data());
                start = parseDate(commandPartition[4]);
                this->controller->eventCreate(eventName, start, std::chrono::seconds(0));
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        case 6:

  if(commandPartition[0] == "event" && commandPartition[1] == "create" && commandPartition[2] == "multi
") {
                this->controller->getUserNameBySocket(this->clientSocket);
                eventName = const_cast<char*>(checkASCII(commandPartition[3]).data());
                start = parseDate(commandPartition[4]);
                try { period = std::chrono::seconds(std::stoi(commandPartition[5])); }
                catch(...) { throw CommandException(COULD_NOT_RESOLVE_ARGUMENT); }

  if(period < std::chrono::seconds(static_cast<int>(MIN_EVENT_PERIOD)) || period > std::chrono::seconds
(static_cast<int>(MAX_EVENT_PERIOD)))
                    throw CommandException(ARGUMENT_LOGIC_ERROR);
                this->controller->eventCreate(eventName, start, period);
            }
            else
                throw CommandException(COULD_NOT_RESOLVE_COMMAND);
            break;
        default:
            throw CommandException(COULD_NOT_RESOLVE_COMMAND);
    }

}
```

Приложение 16, таймер

```
void* Server::timerThreadInitialize(void *thisPtr) {
    ((Server*)thisPtr)->eventTimer();
```

```
    return NULL;
}

const void Server::eventTimer() {
    while(!this->generalInterrupt && !this->timerInterrupt) {
        bool lockTimings = this->mutexTimings.try_lock();

        if(this->timings.empty()) {
            if(lockTimings)
                this->mutexTimings.unlock();

            continue;
        }


  auto now = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().tim
e_since_epoch());

        auto resultVector = std::vector<int>();

        for(auto& current: this->timings) {
            if (now >= current.second)
                resultVector.push_back(current.first);
            else
                break;
        }

        if(lockTimings)
            this->mutexTimings.unlock();

        for(auto& current: resultVector) {
            this->controller->eventNotify(this->controller->getEventNameByEventId(current));
            refreshTiming(current);
        }
    }
}
```