**Государственный комитет Российской Федерации по**

**телекоммуникациям**

**Сибирский государственный университет**

**телекоммуникаций и информатики**

**КУРСОВАЯ РАБОТА**

**По дисциплине “Операционные системы”**

**Тема работы**

**Игра «Многопоточный чат на основе сокетов»**

Работу выполнил

студент 3 курса

группы ИП-412

Максимов А.С.

Работу проверил

Малков Е. А.

Работа защищена

«\_\_\_» \_\_\_\_\_\_\_\_\_\_2016г.

С оценкой «\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2016

Оглавление

[**Цель работы** 3](#_Toc468744476)

[**Поставленные задачи** 3](#_Toc468744477)

[**Реализация проекта** 4](#_Toc468744478)

[*Реализация программы-сервера:* 4](#_Toc468744479)

[**Оформление области серверного приложения** 6](#_Toc468744480)

[*Реализация программы-клиента:* 6](#_Toc468744481)

[**Оформление клиентской области** 7](#_Toc468744482)

# **Цель работы**

Целью нашего курсового проекта являлось создание многопользовательского многопоточного (создание потоков и их синхронизация) чата на основе сокетов (в частности использовался Net Socket).

Второстепенными целями проекта (поставленными нами) стали реализация клиентской области при помощи WinForms и создание полнофункционально сервера обработчика событий.

# **Поставленные задачи**

* Создание простой в использовании программы - клиента с помощью WinForms на языке C#.
* Создание стабильно функционирующего сервера, обменивающегося сообщениями с программой - клиентом и обрабатывающего описанные в них события.
* Создание канальной системы общения между программой-клиентом и программой-сервером используя socket и протокол TCP.

# **Реализация проекта**

## *Реализация программы-сервера:*

Сервер функционирует по принципу *получил – обработал - отправил*.

1. **Получил**: сервер работает в синхронном режиме (то есть ожидает приема сообщения от клиента). Поскольку для каждого клиента создается отдельный поток, то не возникает ошибки с приемом сообщения, так как однозначно известно кто прислал сообщение.
2. **Обработал / отправил**: для взаимодействия между клиентом и сервером используются следующие типы сообщений:
3. MSG\_AUTH\_CONNECT = 100, - Авторизация
4. MSG\_USER\_DISCONNECT, - Пользователь на сервере вышел
5. MSG\_USER\_CONNECT, - Пользователь на сервере вошел
6. MSG\_MESSAGE, - Пришло / ушло мировое сообщение
7. MSG\_MESSAGE\_WHISPER\_IN, - Текущему пользователю пришло личное сообщение
8. MSG\_MESSAGE\_WHISPER\_OUT - Текущий пользователь отправил кому то личное сообщение

Для хранения информации о пользователях реализован класс userInfo, имеющий следующую структуру:

class userInfo

{

public String name;

public Thread thread;

public userInfo(Thread th)

{

thread = th;

name = null;

}

public void SetName(string n)

{

name = n;

}

};

Как видно, класс содержит в себе информацию об имени пользователя, а также о потоке, в котором обрабатывается текущий пользователь.

Для последовательного хранения данных о пользователях используются словари «Dictionary» со следующей инициализацией:

Dictionary<Socket, userInfo> userList = new Dictionary<Socket, userInfo>();

В качестве ключей используются переменные с типом данных Socket.

Словарь (Dictionary) представляет собой сложную структуру данных, позволяющую обеспечить доступ к элементам по ключу. Главное свойство словарей – быстрый поиск на основе ключей. Можно также свободно добавлять и удалять элементы без накладных расходов производительности, связанных с необходимостью смещения последующих элементов в памяти.

В передаче данных используется специальный тип данных byte, один элемент которого способен хранить в себе значение от 0 до 255. Для передачи данных используется массив типа данных byte. Единичная запись зависит от типа передаваемого сообщения и выглядит следующим образом:

MSG\_AUTH\_CONNECT/ MSG\_USER\_CONNECT/ MSG\_USER\_DISCONNECT

|  |  |
| --- | --- |
| Содержимое байта | Тип сообщения |
| № байта | 0 |

|  |  |
| --- | --- |
| Содержимое байта | Имя пользователя |
| № байта | 1 -> размер в байтах(Имя пользователя) |

MSG\_MESSAGE

|  |  |
| --- | --- |
| Содержимое байта | Сообщение, исходящее от пользователя |
| № байта | 1 -> размер в байтах(сообщение) |

MSG\_MESSAGE\_WHISPER\_OUT

|  |  |  |
| --- | --- | --- |
| Содержимое байта | Имя пользователя, от которого исходит сообщение | Сообщение |
| № байта | 1 -> размер в байтах(сообщение) | 1 + размер в байтах(сообщение) -> размер в байтах(сообщение) |

Инициализация нового клиента происходит следующим образом:

1. На сервер приходит сообщение типа MSG\_AUTH\_CONNECT, содержащее в себе имя входящего пользователя.
2. Запускается отдельный поток для нового клиента, куда передается сокет нового клиента.
3. На сервере производится проверка на присутствие пользователя с аналогичным ником. Если таких пользователей на сервере нет, то на клиент отправляется сообщение типа MSG\_AUTH\_CONNECT с присутствием единичного байта, который сигнализирует об успешной аутентификации, также пользователь добавляется в список клиентов userList. В противном случае сообщение с типом MSG\_AUTH\_CONNECT будет содержать в себе нулевой байт, после этого соединение с этим клиентом прерывается, а поток завершается.

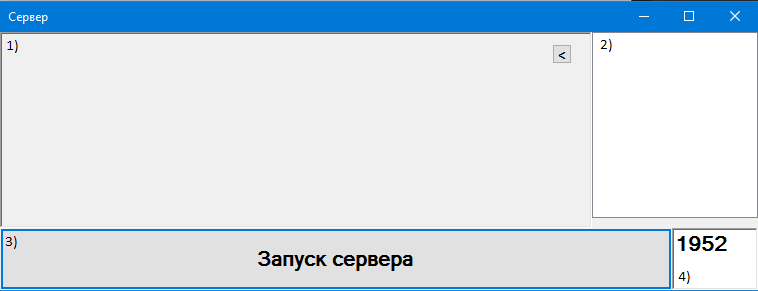
За беспрерывную связь с клиентом отвечает цикл while(true), который прерывается в случае, если размер получаемого сообщения будет равным или меньше нуля:

if (bytesRec1 <= 0)

break;

При выходе клиента с сервера, всем сокетам из списка userList отправляется сообщение типа MSG\_USER\_DISCONNECT, содержащее имя пользователя.

## **Оформление области серверного приложения**



Рабочая область сервера содержит в себе следующие компоненты:

* 1. richTextBox. Это текстовая область, отображающая весь процесс переписки между клиентами.
  2. listBox. Список клиентов. При двойном щелчке на пользователе, тот отключается от сервера.
  3. button. Кнопка, по нажатию которой создается сокет и начинается ожидание подключений.
  4. richTextBox. Текстовая область, позволяющая изменить порт подключения к серверу.

## *Реализация программы-клиента:*

Реализация клиента мало отличается от реализации сервера. Но в реализации клиента есть дополнительный тип сообщения, который он способен принимать от сервера:

|  |  |
| --- | --- |
| Содержимое байта | Тип сообщения |
| № байта | 0 |

MSG\_MESSAGE\_WHISPER\_IN

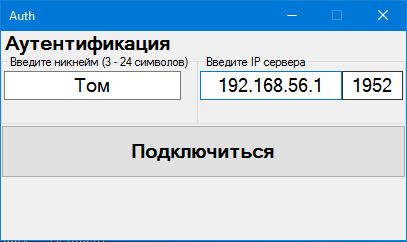
|  |  |  |  |
| --- | --- | --- | --- |
| Содержимое байта | Размер в байтах(пользователь, от которого исходит сообщение) | Имя пользователя, от которого исходит сообщение | Сообщение |
| № байта | 1 | 2-> Размер в байтах(пользователь, от которого исходит сообщение) | 2 + Размер в байтах(пользователь, от которого исходит сообщение)->2 + Размер в байтах(пользователь, от которого исходит сообщение) + Размер в байтах(сообщение) |

Этот тип сообщения приходит, когда к данному клиенту приходит личное сообщение.

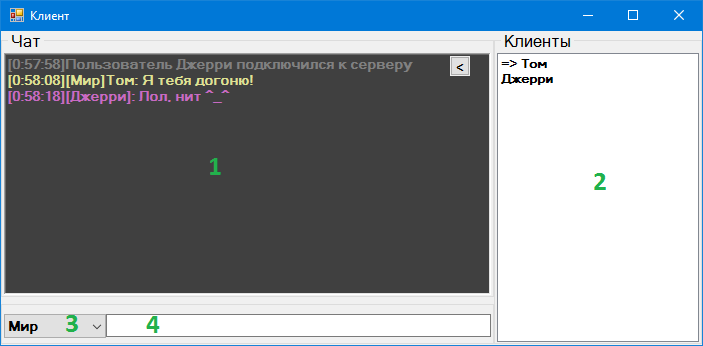
В остальном клиент работает по схожей с сервером логикой.

## **Оформление клиентской области**

Приложение клиента разделено на две формы: “Аутентификация” и “Клиент”. Первая форма представляет собой окно, в котором пользователь может ввести имя пользователя, IP сервера и порт.



Вторая форма представляет собой рабочее окно для общения между клиентами.



Рабочая область сервера содержит в себе следующие компоненты:

1. richTextBox. Это текстовая область, отображающая весь процесс переписки между пользователями.
2. listBox. Список пользователей. При двойном щелчке на пользователе, тот выбирается в качестве собеседника, которому можно отправить личное сообщение. ComboBox принудительно выбирает элемент с соответствующим пользователем.
3. comboBox. Выпадающий список пользователей.
4. textBox. Текстовая область ввода сообщений.

Сообщения окрашиваются в разные цвета:

Мировые сообщений окрашиваются в светло желтый. Личные сообщения окрашиваются в светло фиолетовый. Остальные сообщения окрашиваются в серый цвет.