*Simple chat client-server*

Разглеждаме софтуерната архитектура за разпределени системи клиент-сървър. Най-общо можем да характеризираме тази архитектура като услуги изисквани от клиентите и предоставяни от сървърите. Сървърите доставящи услугите се държат и проектират различно от клиентите изискващи услугите. Някои от по важните характеристики на двете страни в тази архитектура са както следва:

*Характеристики на Клиента*

- Подава заявки

- Изчаква отговор

- Свързва се до малък брой сървъри едновременно

- Взаимодейства си с крайните потребители чрез графичен интерфейс или интерфейс с команден ред

*Характеристика на Сървъра*

- Пасивност

- Чака за заявки от клиенти

- При получаване на заявки, ги обработва и след това отговаря

- Получава заявки от голямо количество клиенти

- Не контактува директно с крайният потребител

Задачата ни се състои да моделираме определен тип клиент-сървър приложение, а именно такова за обмяна на малки текстови съобщения. Приложението се състои от две части (програми) работещи напълно независимо една от друга - chat сървър и chat клиент. По предварително дефиниран протокол за работа на клиента със сървъра трябва да се реализират двете части на приложението.

*Протокол на chat сървър (команди които той разбира):*

1) Регистриране на потребител:

**user <username>\r\n**

Веднага след установяване на връзка със сървъра клиента изпраща команда, регистрираща потребителя <username>. Възможностите на сървъра са две – регистрира съответния потребител или вече има такъв регистриран потребител. За двата случая сървъра връща подходящо съобщение на клиента.

* **200 ok <username> successfully registerred\r\n**

при успешна регистрация

или

* **100 err <username> already taken!\r\n**

ако вече има такъв регистриран потребител.

2) Съобщение до потребител:

**send\_to <username> <single line message>\r\n**

След като потребителя е регистриран, той може да изпраща текстови съобщения (от един ред) до други (вече регистрирани) потребители. Възможностите за отговор на сървъра отново са две:

* **200 ok message to <username> sent successfully.**

или

**100 err <username> does not exists!**

3) Съобщение до всички потребители:

**send\_all <single line message>\r\n**

След като потребителя е регистриран, той може да изпраща текстови съобщения (от един ред) до всички други (вече регистрирани) потребители. Възможностите за отговор на сървъра отново са две:

**200 ok message sent successfully.**

или

**100 err server error!**

След като даден клиент е изпратил съобщение и адресата му съществува то последния може да получи това съобщение, като сървъра изпрати следната команда:

**300 msg\_from <username> <single line message>\r\n**

4) Списък на потребителите ползващи в момента сървъра (регистрирани):

**list\r\n**

Отново само при успешна регистрация, клиента може да поиска от сървъра списък на потребителите които в момента обменят съобщения. Възможностите за отговор:

**200 ok <username1> ... <usernameN>\r\n**

или

**100 err server error!**

5) Преустановяване на работа на потребител (напускане)

**bye\r\n**

При завършване на работата си със сървъра (преустановяване процеса на обмяна на съобщения) клиентите изпращат команда уведомяваща сървъра за това. Клиентски страни се уведомяват от сървъра за напускането на даден клиент чрез командата:

**400 user gone <username>\r\n**

Изискванията към програмите са следните:

(o) Порт, на който ще слуша и/или максимален брой клиенти които могат да обменят съобщения през този сървър могат да бъдат подадени през конзолата. Ако потребителя не укаже такива се избират стойности по подразбиране;

(о) Адреси и порт, на който ще се свърже клиента, за да ползва услугите на сървъра се подават от потребителя на main() метода на стартовия клас. Ако потребителя не укаже такива се избират стойности по подразбиране;

(о) Клиентът може да получава съобщения, без потребителя да изпраща съобщения (тоест получаването на съобщение не е пряко обвързано изпращането);.

(о) Командните аргументи (параметри) на терминална (конзолна) Java програма, получаваме във масива String args[] на main() метода, на стартовия клас.

Работата може да се опише по следния начин.

Включваме сървъра, който има ограничение на броя клиенти , които може да обслужи наведнъж (зададен от командния ред на main() метода на класа **MultiServer** (по подразбиране е **10**)). Той започва да слуша на определен порт (зададен от командния ред на main() метода на класа **MultiServer** (по подразбиране е **4455**)) за клиент, който да се опита да се свърже с него.

При появата на такъв, се създава нишка на сървъра чрез конструктор на класа **MultiServerThread** (тя реално представлява сървър за точно този определен клиент, тъй като от нея се изпращат и получават съобщения към и от клиента), тази нишка се добавя в **registry** на user-ите(масив **users** – поле на класа **MultiThreadServer**), съставено от всички активни такива нишки, и се изпраща съобщение, което приканва потребителя да се регистрира.

След успешна регистрация потребителя има следните права:

- изпращане на съобщение до определен регистриран потребител

- изпращане на съобщение до всички потребители

- получаване на съобщения от други потребители

- получаване на списък от всички (активни) потребители в registry-то

- излизане от системата (придружено със затваряне на клиента)

Начините за постигане на горните са обяснени по-нагоре в документацията (в описанието на протокол, чрез който си общуват клиента и сървъра). Накратко, потребителя праща през клиента съобщение в определен формат, което е разбраемо за сървъра, и сървърът изпълнява посоченото. Съобщенията, пратени от клиента се обработват от метода **processInput()** на класа **ChatProtocol**, като по резултата от неговото изпълнение се използва като input от съответната нишка на сървъра, която от своя страна на базата на този input изпълнява определената задача.

След изпращане на съобщение “bye” от страна на клиента, нишката на сървъра изпраща съобщение, че приключва работа, спира работа и бива премахната от registry-то .