****

**Софийски университет „Св. Климент Охридски”**

**Факултет по математика и информатика**

Курсов проект

по Системи за паралелна обработка

Тема:

„Многонишков чат клиент-сървър”

Изготвил: Иван Валериев Николов ФН: 81503, Компютърни науки,

курс 3, поток 2, група 5

Ръководител: проф. Васил Цунижев, ас. Христо Христов

Дата: 07.06.2019г. Проверка: .........................

**Съдържание**

[1. Условие на задачата](#Условие)

[2. Описание и интерфейс на програмата](#Описание)

[3. Архитектура на реализацията](#Архитектура)

[4. Алтернатива](#Алтернатива)

[5. Инструкции за изпълнение](#Инструкции)

1. **Условие на задачата**

Разглеждаме софтуерна архитектура от тип клиент-сървър, в един от конкретните ѝ варианти,а именно – чат сървър. Задачата е да се реализира клиент-сървър приложение, което спазва принципите на тази архитектура и служи за обмяна на малки съобщения между потребителите(клиентите). Принципите, които се стремим да спазим при реализация на въпросното приложение са:

* 1. За клиента:
* Подава заявки към сървъра – в случая команди за изпращане на прости съобщения до другите потребители
* Изчаква отговор – в рамките на разумно време се очаква сървърът да реагира на неговите команди и да му изпрати отговор.
* Получава съобщения – действайки като посредник, сървърът отговаря за това да доставя съобщения между потребителите
* Не е натоварен с много реализирана функционалност, единствено служи като средство за изискване на услуги от сървъра.
  1. За сървъра:
* Способен да предоставя различни услуги на клиенти. В случая да действа като посредник – да получава и изпраща съобщения на отделните потребители.
* Трябва да може да обслужва много клиенти наведнъж.
* Пасивен – не предприема никакви действия докато не му бъде указано от клиента.
* Получава заявки от голям брой потребители и е способен да ги обработи в разумно време. В случая всички свързани потребители трябва да могат едновременно да си изпращат съобщения един на друг и сървърът трябва да се грижи те да бъдат доставени във възможно най-кратко време.

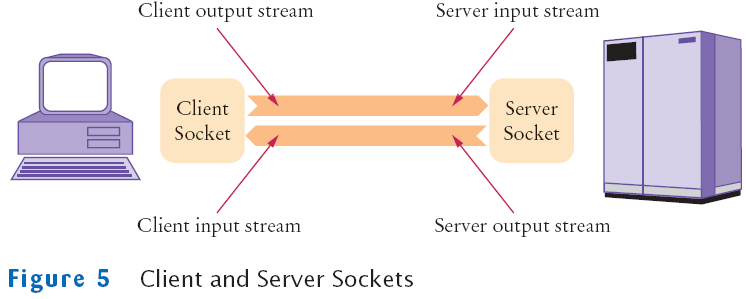
1. **Описание и интерфейс на програмата**

Приложението е реализирано като са написани две отделни програми, работещи независимо една от друга – клиент и сървър. За целта сме използвали езика Java, който предоставя много удобно API за работа със сокети и нишки. Двете програми са конзолни приложения – не предоставят графичен интерфейс за ползване от потребителя. Дефиниран е следния протокол за работа:

* 1. Команди,които клиентът изпраща към сървъра:
* user <username> - регистриране на потребител. Неуспешно ако името вече е заето от друг потребител.
* list – само при успешна регистрация, отговорът на сървъра е списък с регистрираните(активните) потребители на сървъра
* send\_to <username> <one\_line\_message> - само при успешна регистрация, изпраща private съобщение до друг потребител ако е регистриран.
* send\_all <one\_line\_message> – само при успешна регистрация, изпраща съобщение до всички регистрирани потребители
* <bye> - сигнализира за напускане на чат сървъра и бива изтрит от списъка с регистрирани потребители
  1. Съобщения,които сървърът изпраща до потребителите:
* съобщения от вида <status\_code> <response\_to\_user\_command>
* съобщения от вида <username> : <message>

1. **Архитектура на реализацията**

**3.1** За реализация на дефинирания протокол от съобщения, които се изпращат между сървъра и клиента е реализирана йерархия от класове-обвивки на типовете съобщения, които имплементират ***interface Serializable*** за да могат бъдат сериализирани при прехвърляне между сървъра и клиента и да бъдат съответно възстановявани при получаването им.За комуникация през сокетите се използват ***ObjectInputStream*** и ***ObjectOutputStream***, които предоставят удобен интерфейс за прехвърляне на обекти между клиента и сървъра през сокетите.



**3.2** **Реализация на клиента**

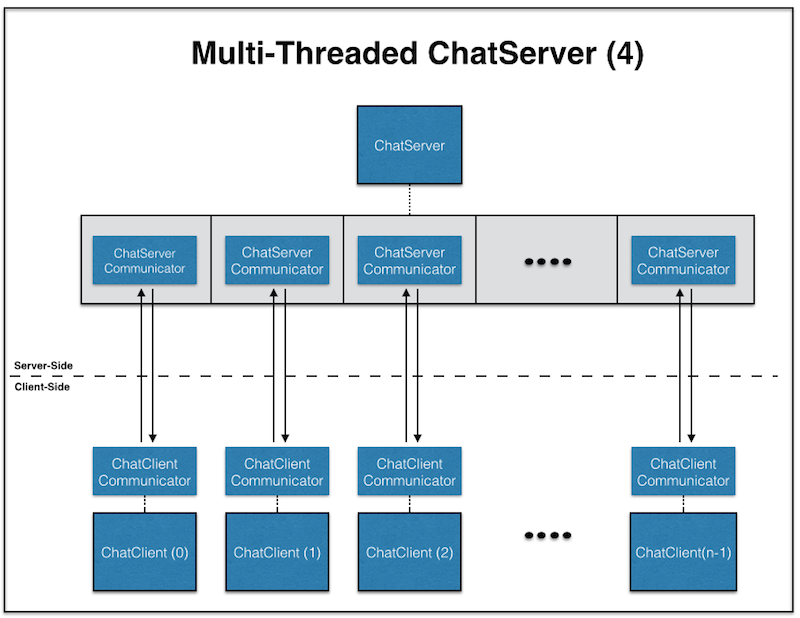
Като изискване е поставено да се осигури асинхронна работа на клиента. За при стартиране на програмата клиент се отваря ***Socket*** към сървъра и на отделна нишка се подава ***ObjectInputStream*** референция получена от сокета и на тази нишка, клиентът започва да слуша за съобщения изпратени от сървъра. Когато бъде получено съобщение – нишката го отпечатва на конзолата. Така клиентът може да получава съобщения независимо от командите, които подава към сървъра (В традиционните реализации на клиент-сървър приложения, клиентът подава заявка и след това изчаква за отговор от сървъра преди да подаде следваща заявка. В случая на нас това не ни върши работа тъй като искаме клиентът да може да получава съобщения от други потребители без да е подал заявка към сървъра.) В случай че за подадения адрес и порт не може да бъде установена връзка със сървъра, програмата съобщава за това и приключва работа. В случай че се подаде команда за напускане на сървъра, въпросния ***Socket*** към сървъра се затваря,а заедно с него и ***ObjectInputStream*** и ***ObjectOutputStream*** и нишката която се грижи за получаване на съобщения от сървъра приключва работа.

**3.3 Реализация на сървъра**

Тук сме поставени пред избор. Съществуват няколко варианта за реализиране на програмата на сървъра. Всички от тях включват използването на много нишки – интуитивно за всеки потребител на сървъра се заделя нова нишка, която отговаря за получаването и изпращането на съобщения през ***Socket***. Ще опишем варианта, който сме избрали за реализация и защо сме го избрали и след това ще опишем защо другия вариант не би бил подходящ за нашите цели.

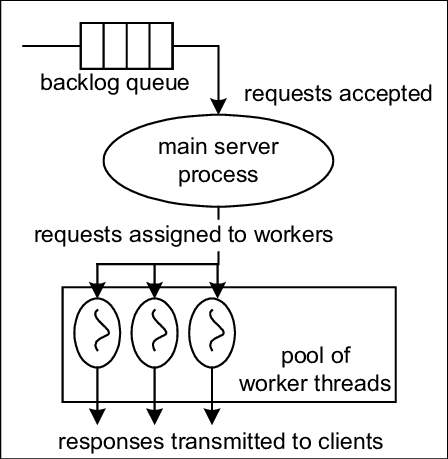
При стартиране на програмата с помощта на команден параметър(виж Инструкции за пускане на програмата) указваме максималният брой клиенти които може да обработва по едно и също време сървъра. Това е необходимо тъй като очевидно не е възможно сървъра да обслужва безброй потребители – ако заделя нишка, която се грижи за всеки потребител, ресурсите на сървъра ще свършат по някое време и това ще се отрази върху работата му. Сървърът започва да слуша на зададения порт за конекции.Щом някой се свърже към него се отваря нов ***Socket***, с който може да се комуникира с клиента. Сървърът за да може да продължи да обслужва и други клиенти стартира нова нишка, на която подава сокет обекта и подава референция към нея в ***HashTable*** обект където срещу ключовете(имената на потребителите) се пазят референции към всички нишки. Всяка нова нишка, която се стартира за клиент съдържа в себе си референция към ***ChatServer*** обекта, за да може през него щом се наложи да даде на друга нишка да изпрати съобщение към потребител. При напускане на сървъра на клиент – нишката, която се грижи за него казва на ***ChatServer*** да изтрие референцията към въпросната нишка, след която тя приключва работа (Това е важен аспект,тъй като щом една нишка приключи работа,тя продължава да живее като обект, докато има референции към нея – garbage collector-a не я трие. Освен това трябва да се изтрие от хеш таблицата за да остане консистентен броя на потребители, който се следи когато се отвори нов сокет към сървъра.). Единствената синхронизация, която е необходимо да реализираме в случая е достъпа до обекта ***HashTable*** и писането в ***ObjectOutputStream***. ***HashTable*** e thread-safe, а синхронизацията на писане през стрийма реализираме като правим метода който изпраща съобщения на клиента ***synchronized***.

https://contribute.geeksforgeeks.org/wp-content/uploads/JavaSocketProgramming.png



1. **Алтернативa**

Друг начин, по който бихме могли да подходим към задачата е ползването на ***ThreadPool***. По този начин работят класическите web сървъри. Този подход включва дефиниране на ***coreThreadSize*** и ***maximumThreadSize*** както и queue и още няколко други параметъра. Идеята на този подход е при достигане на лимита на броя нишки всеки нов ***Runnable*** (в нашия случай обслужване на нови потребители) подаден на ***threadpool***-a се добавя в ***blocking queue*** и се обработва чак когато някоя съществуваща нишка се освободи за работа.Този подход не е подходящ за нас тъй като не желаем потребителя да чака да се освободи място на сървъра за него,тъй като това може да продължи дълго време, а искаме да го уведомим ако е пълен сървъра да опита по-късно.



1. **Инструкции за изпълнение**

Програмите се стартират от командния ред с командата:

java –jar <името\_на\_пакетираната\_програма> <опции>  
5.1 Клиентската програма приема параметрите „-serverAddress”, „- serverPort” и „–testMode”, които могат да бъдат подадени като command line parameters и са optional. Ако не бъдат подадени на програмата, тя използва стойности по подразбиране. Подават се по следния начин:

-<parameter\_name> <value>

-testMode – използван за да се симулира ефективността на сървъра (тъй като ще е трудно за нашите цели да тестваме работата на сървъра с много на брой клиенти.) При стойност true на параметъра, клиентът ще бъде настроен да не чака за input от потребителя,а да генерира съобщения, които да бъдат доставени на всички регистрирани потребители на сървъра, което ще тества как той се справя с големия брой съобщения които да бъдат доставени на всички потребители.

-serverAddress и -serverPort – в комбинация, стойностите на тези два параметъра указват на програмата клиент да отвори socket към сървъра, през който ще се реализира комуникацията.

5.2 Сървърната програма приема параметрите “-serverPort”, “-maxNumUsers”. Приема параметрите по същия начин като клиента и има стойности по подразбиране за тях.

-serverPort – При стартиране на сървъра, той започва да слуша на този порт за клиенти,които да се свържат с него.

-maxNumUsers – Указва на сървъра колко най-много клиенти да може да обработва едновременно.