Зад. 3 (Simple chat client-server, with file transfer)

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.

Някой 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

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. Например:

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!

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) Обмяна на файл между потребители

Тук е необходимо да направим някой уточнения. Принципно съществуват различни подходи за реализация на подобна функционалност. Ще изредим два такива, непосредствено приложими за нашите цели.

(о) Първият подход реализира обмяната на файлове, като изпраща съдържанието на самия файл към сървъра, който известява (по подходящ начин) клиента (потребителя) за когото е предназначен файла. Основен недостатък на този подход е че съдържанието на файловете минава през сървъра. Като плюс можем да отчетем факта че е сравнително лесен за реализация;

(о) Вторият подход разчита на сървъра като страна която единствено сигнализира, докато реалния обмен на данни се извършва между клиентите (разновидност на т.нар. peer-to-peer комуникация). Като недостатък на този поход, можем да отчетем това че клиентските програми се натоварват с допълнителна функционалност, която не е особено лесна за реализация;

Като пример за комуникация между клиента и сървъра, реализираща първия подход, можем да разгледаме следните команди:

Клиента изпраща файл към сървъра използвайки командата (много-редова в този случай):

send_file_to <username> <file_name> <communication_pkg_size>\r\n
file-row-1-base64-encoded-with-max-length-512-bytes\r\n
...
file-row-N-base64-encoded-with-max-length-512-bytes\r\n

Възможните отговори на сървъра са (след като е получил подходящо потвърждение от клиента получател):

200 file transferred sucessfully\r\n

или

100 server transfer error\r\n

Сървърът от своя страна известява клиента получател, използвайки командата:

500 file_from <username> <file_name> <communication_pkg_size>\r\n
file-row-1-base64-encoded-with-max-length-512-bytes\r\n
...
file-row-N-base64-encoded-with-max-length-512-bytes\r\n

От своя страна клиента получател известява сървъра, относно получавания файл, използвайки командите:

200 file accepted sucessfully\r\n

или

100 client transfer error\r\n

Бележка (обяснение)

Редовете от вида:

file-row-k-base64-encoded-with-max-length-512-bytes\r\n

означават следното. Данните на файла се обменят на блокове, като всеки блок представлява един ред от комуникацията с максимална дължина от 512 байта. Блоковете (редовете) са кодирани, чрез т.нар. Base64 encoding. С други думи, всеки блок (ред) от файлови данни се представя използвайки единствено следните символи: **A-Z a-z 0-9 + /**

6) 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. Отговора на сървъра е

пожелателен в този случай.

NOTE: Получаване на съобщение от клиент, изпратено през сървъра (от друг клиент).

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

на клиентския канал за връзка със сървъра.

Изискванията към програмите са следните:

(о) Да се осигурят подходящи командни параметри за сървъра. Например порт на който ще слуша и/или максимален брой клиенти които могат да обменят съобщения през този сървър. Ако потребителя не укаже такива се избират стойности по подразбиране;
(о) Да се осигурят подходящи командни параметри за клиента. Например адреси и порт на който ще се свърже за да ползва услугите на сървъра. Ако потребителя не укаже такива се избират стойности по подразбиране;
(о) Да се осигури асинхронна работа на клиента. Тоест той може да получава съобщения, без потребителя да изпраща съобщения (тоест получаването на съобщение да не е пряко обвързано с изпращането);
(о) Да се осигури възможност активните клиентски страни да бъдат уведомявани от сървъра за напускането на даден клиент. Това може да става например чрез командата:

400 user gone <username>\r\n

изпращана от сървъра на клиентския канал за връзка.

(о) При желание да се реализира извеждане на различните етапи от работата на сървъра , тоест подходящи текстови съобщения във неговия стандартен изход. Да се осигури възможност за „quiet“ режим на работа.
(о) Аналогично изискване за клиента, като при неговия „quiet“ режим се извеждат само евентуалните съобщения получавани от други потребители

ЗАБЕЛЕЖКА:

(о) При желание за направата на подходящ графичен потребителски интерфейс (GUI) с помощта на класовете от пакета **javax.swing** задачата може да се изпълни от **двама души**;
(о) Задачата може да се реши и с помощта на RMI (**java.rmi**). За целта сървъра може да се оформи като Remote Object а клиентите като подходящи негови ползватели.

Уточнения (hints) към задачата:

(о) Командните аргументи (параметри) на терминална Java програма, получаваме във масива **String args[]** на **main()** метода, намиращ се в стартовият клас. За „разбирането“ им (анализирането им) може да ползвате и външни библиотеки писани специално за тази цел . Един добър пример за това е: **Apache Commons CLI (http://commons.apache.org/cli/)**.