Desenvolvimento de um sistema de chat

U.C. Redes de Computadores

Docentes: Pedro Salgueiro, Pedro Patinho
Discentes: Helder Godinho 42741, Guilherme Grilo 48921

5 de junho de 2022

1 Introdução

No presente trabalho foi-nos proposto desenvolver ainda mais o nosso sistema de chat, que foi feito como 1º trabalho, de forma a, para além das funcionalidades que já possuía, ainda ter a capacidade de reconhecer várias *Tags* diferentes e guardar vários *Posts* em cada uma delas, e implementar a funcionalidade da transmissão de ficheiros, seja entre utilizadores seja na forma de *Post* numa *Tag*.

De salientar que voltamos a fazer o trabalho em linguagem Java, devido à maior facilidade no que toca a trabalhar com esta mesma linguagem.

2 Implementação

Tal como no 1º trabalho, o nosso sistema de chat está subdividido em dois ficheiros, *Server.java*, que serve essencialmente para criar o servidor que posteriormente irá aceitar os diversos clientes e permitir que estes comuniquem entre si de variadas formas, recorrendo para tal ao uso de *threads*; *Client.java*, que, tal como o servidor, recorre também ao uso de *threads* de maneira a que, com apenas um ficheiro, seja possível criar vários clientes que se irão comunicar entre si.

2.1 Server.java

Este ficheiro serve essencialmente para criar o servidor que irá permitir que os diversos clientes comuniquem entre si.

Em suma, começamos por criar um *Server Socket* que que tem a função de aceitar os diversos *Sockets* dos vários clientes que se pretendam conectar ao servidor através da porta TCP especificada (1234). Tal como no 1ºtrabalho, foram implementadas as seguintes classes:

- `Server` - onde é aceite o *Socket* de cada cliente por parte do *Socket* do servidor e inicializa a *Thread* responsável por permitir que os diversos clientes comuniquem entre si. De referir ainda que esta classe possui ainda duas listas, a lista *clients* do tipo *AllClients*, que guarda todos os clientes ativos no servidor; e ainda a lista *tags* do tipo *Tag*, que guarda todas as tags presentes no servidor.
- `MultipleClients` - implementa a interface *Runnable* que permite que, através da função *run()*, as diversas *Threads* (clientes) consigam enviar a mensagem para o servidor que se encarrega de a encaminhar o(s) cliente(s) desejado(s).

Foi ainda necessário criar mais quatro classes novas com o objetivo de implementar tudo o que foi pedido no enunciado deste 2º trabalho:

- `AllClients` - guarda todas as informações acerca dos clientes ativos no servidor.
- `Tag` - guarda todas as tags que forem criadas pelos diferentes clientes no servidor. Possui duas listas, a primeira *subscribers* do tipo *AllClients*, que guarda os clientes subscritos numa determinada tag; e a lista *posts* do tipo *Post*, que guarda todos os *posts* já publicados numa determinada *tag*.
- `Post` - guarda todos os *posts* que forem feitos por clientes, isto é, guarda a mensagem que o cliente quer publicar e ainda o *username* do cliente emissor.
- `TheFile` - nesta classe é guardado o nome do ficheiro transferido de um cliente para o servidor, o username do próprio e ainda o *array* de *bytes* que contém o conteúdo do ficheiro mas transformado em *bytes* de forma a que o servidor consiga transferir o ficheiro para o cliente desejado ou colocá-lo num *post* na *tag* desejada.

2.2 Client.java

Este ficheiro foi criado com o objetivo de criar múltiplos clientes, recorrendo ao uso de *Threads*, que possam receber enviar e receber mensagens entre eles, utilizando o servidor como intermediário.

Devido ao uso das *Threads*, conseguimos criar vários clientes diferentes e conectá-los todos ao servidor, permitindo a troca de mensagens e ficheiros entre os mesmos.

Este ficheiro possui apenas uma única classe:

- `Client` - nesta classe é criado o *Socket* do cliente, o que irá permitir que o mesmo se conecte ao servidor e passe a poder enviar e receber mensagens de outros clientes que também estejam conectados ao servidor.

3 Funções

3.1 Servidor

- `main` - aqui é inicializado o *Socket* do servidor e ainda as listas *clients*, *tags* e *files*. Responsável ainda por aceitar o *Socket* de cada cliente e inicializar a *Thread* de cada cliente.

- `checkUsername` - verifica se a primeira palavra da mensagem enviada pelo cliente, quando este pretende definir o seu *username*, é a palavra "HELLO".
- `clientAlreadyExists` - verifica se o *username* escolhido por um novo cliente já existe no servidor.
- `addToGlobal` - adiciona o novo cliente à *tag* GLOBAL (que possui todos os clientes subscritos).
- `checkLength` - verifica se a mensagem enviada pelo cliente possui os requerimentos mínimos necessários do comando desejado.
- `getUsername` - esta função serve para analisar a primeira mensagem enviada pelo cliente e, no caso desta se encontrar no formato correto, retorna apenas o *username* escolhido.
- `findClient` - encontra o cliente que possua o *username* pretendido. No caso de não existir nenhum cliente com esse *username*, devolve *null*.
- `getMessage` - devolve apenas a parte da mensagem do cliente, já sem o comando nem o destinatário.
- `removeSubFromTag` - remove a subscrição do cliente da *tag* pretendida.
- `removeSubFromAllTags` - remove as subscrições de todas as *tags* onde o cliente esteja inserido. Usada apenas quando o cliente se pretende desconectar do servidor.
- `findTag` - devolve a *tag* que possua o mesmo nome dado como argumento. No caso de não existir nenhuma *tag* com esse nome, devolve *null*.
- `clientAlreadyInTag` - verifica se o cliente se encontra subscrito numa determinada *tag*.
- `addClientToTag` - adiciona o cliente à *tag* pretendida.
- `messageToSubs` - envia a mensagem para todos os subscritores de uma determinada *tag*, excepto para o remetente da mesma.
- `printAllPosts` - envia todos os *Posts* de uma determinada *tag* para o cliente, incluindo o nome dos ficheiros que estejam como *post* nessa *tag*.
- `printAllFileNames` - envia o nome dos ficheiros que estejam contidos num *post*.
- `printAllTags` - envia para o cliente o nome de todas as *tags* ativas no servidor.
- `printAllUsers` - envia para o cliente o *username* de todos os clientes ativos no servidor.
- `printAllSubs` - envia para o cliente o *username* de todos os subscritores de uma determinada *tag*.
- `receiveFile` - esta função é responsável por receber um *array* de *bytes*, que contém o texto de um determinado ficheiro convertido em *bytes*, e, consoante a vontade do cliente, envia para outro cliente ou guarda como *post* na *tag* escolhida.
- `sendFile` - esta função é responsável por enviar o conteúdo do ficheiro, convertido num *array* de *bytes*, para o cliente desejado.
- `run` - considerada a função mais importante, responsável por receber as mensagens de cada cliente, analisar as mesmas, e ainda executar o comando pretendido. Recorre às *threads* de forma a receber mensagens de diversos clientes.

- `addText` - adiciona a mensagem à lista *Text*, na classe *Post*
- `addFile` - adiciona as informações do ficheiro (conteúdo (convertido num *array* de *bytes*), remetente e nome) à classe *Post*.

3.2 Cliente

- `main` - responsável por inicializar, não só o *Socket* do cliente, como também todos os recursos necessários para ler e enviar mensagens para o servidor. Responsável ainda por ler as mensagens do terminal e enviar para o servidor, e possui ainda a *Thread* responsável por receber as mensagens enviadas pelo servidor.
- `sendFileToServer` - esta função serve para ler o conteúdo de um determinado ficheiro, converter esse mesmo conteúdo para um *array* de *bytes* e enviar para o servidor.
- `receiveFileFromServer` - recebe um *array* de *bytes* do servidor (que nada mais é que o conteúdo de um ficheiro), e converte-o para texto noutro ficheiro diferente.

4 Lista de comandos

Tal como pedido no enunciado, implementámos os seguintes comandos:

- `HELLO <nickname>\n` - permite ao cliente definir o *username* pelo qual passará a ser conhecido no servidor.
- `MSG <tag>/<username> <message>\n` - permite ao cliente enviar uma mensagem ou para uma *tag* (na forma de *post*) ou para um utilizador.
- `POST <tag> <message>\n` - envia uma mensagem na forma de *post* para a *tag* escolhida.
- `READ <tag>\n` - apresenta todos os *posts* já enviados e guardados numa determinada *tag*.

Para além destes comandos obrigatórios, decidimos implementar alguns comandos extras:

- `EXIT` - permite ao cliente se desconectar do servidor.
- `MULTIPOST <tag> <message>` - o cliente passa a poder enviar posts para uma determinada *tag* com mais do que 1 linha. Apenas quando o cliente enviar "END", o post é guardado e a transmissão é fechada.
- `SUB <tag>\n` - permite a um cliente subscrever uma *tag* já existente.
- `CREATE <tagName>\n` - permite que um cliente crie uma nova *tag*, desde que o nome pretendido para a mesma não coincida com o nome de nenhuma *tag* já existente no servidor.
- `SHOW TAGS/USERS/<tag>\n` - permite mostrar ao cliente o nome de todas as *tags* no servidor (no caso de ser "SHOW TAGS"), ou o *username* de todos os clientes conectados ao servidor (no caso de ser "SHOW USERS"), ou mostra o *username* de todos os subscritores de uma determinada *tag* (no caso de ser "SHOW tag").
- `COUNT TAGS/USERS/<tag>\n` - envia ao cliente o número de *tags* ativas no servidor (no caso de ser "COUNT TAGS"), ou envia o número de utilizadores que estejam ativos no servidor (no caso de ser "COUNT USERS"), ou mostra o número de clientes subscritos numa determinada *tag* (no caso de ser "COUNT tag").

- `UNSUB <tag>\n` - permite ao cliente remover a sua subscrição da *tag* desejada.
- `FILE <tag>/<username> <filename> <bytes>\n` - permite que o cliente envie um ficheiro, com tamanho bytes, ou para uma determinada *tag* ou para um determinado cliente.
- `DOWNLOAD <filename>\n` - permite a um cliente transferir um ficheiro que esteja guardado como *post* numa *tag*.

5 Conclusão

Após a realização deste trabalho, pensamos ter conseguido atingir o objetivo que nos foi proposto. Para além de termos conseguido aprofundar ainda mais o nosso conhecimento acerca de *Sockets* e *Threads*, também fomos capazes de melhorar ainda mais o nosso sistema de chat, tornando-o mais sofisticado.

Um dos problemas com que nos deparámos foi com uma exceção que ocorre quando o primeiro cliente criado (primeira *thread*) executa o comando Exit, isto é, se desconecta do servidor. De salientar que este problema só acontece quando existem mais do que 2 clientes conectados ao servidor e de salientar ainda que esta ocorrência não compromete em nada o bom funcionamento nem do servidor nem dos restantes clientes.

Referir também outro problema que o nosso sistema de chat é que, se for executado o atalho ctrl+c no terminal de um cliente, o servidor não reconhece que o mesmo se desconectou, pois não fomos capazes de implementar esta funcionalidade.

6 Bibliografia

https://www.geeksforgeeks.org/socket-programming-cc
https://www.youtube.com/watch?v=OTwp3xtd4dg
Patinho, Pedro in "Aulas teóricas de Redes de Computadores". 2022 at University of Évora.
Salgueiro, Pedro in "Aulas práticas de Redes de Computadores". 2022 at University of Évora.