Universidade Federal
de São João del-Rei

Servidor de Mensagens Publish/Subscribe

Joel Silva Campos das Chagas

Neste trabalho iremos implementar um servidor e clientes para troca de mensagens de forma similar ao serviço da plataforma Twitter.

São João del-Rei

Maio de 2022

Sumário

1 Introdução

Neste trabalho implementou-se um simples servidor de mensagens, com funcionamento similar ao da plataforma Twitter. O programa foi estruturado no formato cliente-servidor, em que vários clientes podem se conectar ao servidor para enviar e receber mensagens. As mensagens dos clientes podem conter tags que indicam que outros clientes com a mesma tag também devem recebê-las. Para isso, foi necessário implementar a comunicação cliente-servidor, guardar uma tabela com cada cliente e suas tags cadastradas e permitir que clientes cadastrem e descadastrem tags.

2 Implementação

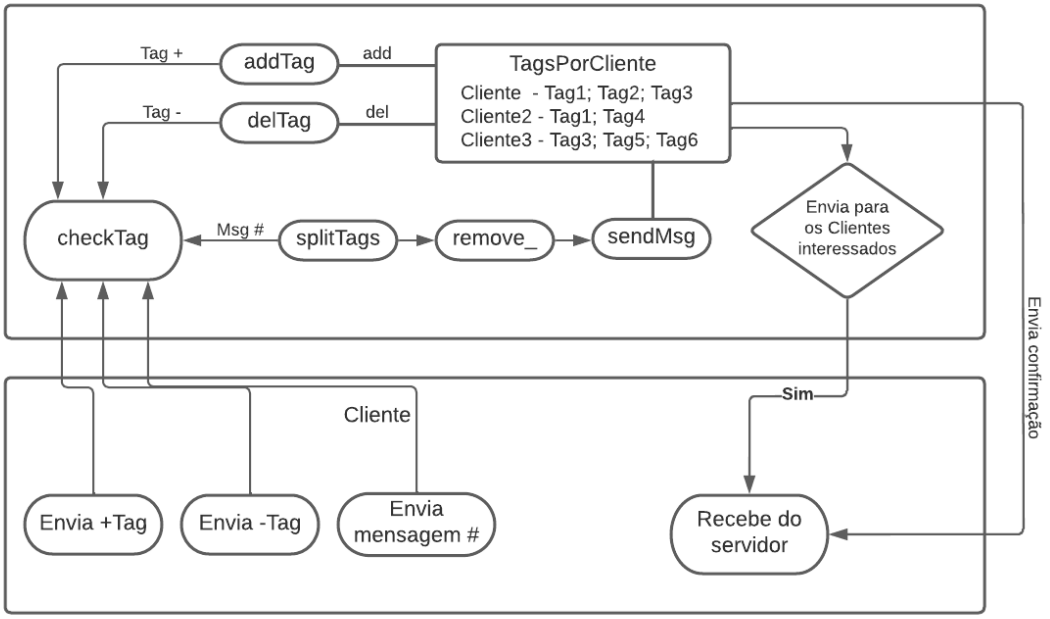
O trabalho foi implementado utilizando o protocolo TCP, protocolo de transporte fim a fim, orientado a conexão, que fornece um serviço de transferência confiável de dados entre Serviços de Transporte na Internet aplicações parceiras.

Garante que os dados são entregues livres de erro, em sequência e sem perdas ou duplicação.

3 Lógica do programa

O servidor utiliza a função threading para aceitar multiplos usuários, para ser capaz de receber e enviar multiplas mensagens ao mesmo tempo. Sendo efetuada uma conexão com o cliente ele entra na função receive, onde é recebido os dados do cliente e a thread é iniciada. As mesagens recebidas podem ser de inserção ou remoção de tags ou podem ser mensagens que devem ser transmitidas aos outros clientes de acordo com as tags contidas nelas. Mensagens de inserção de tags têm um caractere (+) seguido da tag a ser cadastrada, enquanto mensagens de remoção têm um (-), então elas podem ser distinguidas de mensagens de texto normais. Estas mensagens são enviadas pelo cliente ao servidor e podem conter citações de tags, o que é feito através de um caractere () seguido da tag de interesse. As mensagens com citação de tag são enviadas para todos os clientes que cadastraram interesse nessa tag, menos para o cliente que enviou a mensagem. Os passos para tratar cada caso são detalhados a seguir.

Figura 1: Exemplo de fluxograma das funções do programa.



4 Recebendo informações

A estrutura utilizada para armazenar as informações das tags de cada cliente foi um dicionário chamado tags-per-client. As chaves desse dicionário são tuplas contendo a conexão de cada cliente. E para cada chave é armazenada uma lista com as tags para as quais determinado cliente tenha se cadastrou. A seguir um exemplo mostrando como o programa armazena tais informações.

Figura 2: Exemplo armazenamento de informações no dicionário.

A função para adicionar tags é chamada addTag. Caso o cliente ainda não esteja no dicionário, o que acontece quando o cliente cadastra sua primeira tag, ele é inserido no dicionário e sua palavra é cadastrada. Se o cliente já estiver no dicionário, verifique se a tag não foi inserida para esse cliente antes de adicioná-la à lista de tags para esse cliente. O cliente recebe então uma confirmação de que a tag foi adicionada e qual é a tag.

A função de exclusão de tags é chamada delTag. Ele primeiro verifica se a tag a ser removido já está registrado para o cliente relevante e, caso contrário, retorna uma mensagem avisando que a tag não existe para esse cliente. Se a tag existir, ela será removida do dicionário do cliente. Neste caso, uma mensagem é enviada ao cliente informando que a tag solicitada foi removida, e uma mensagem é impressa na tela do servidor indicando a operação de remoção da tag.

5 Tratamento das mensagens

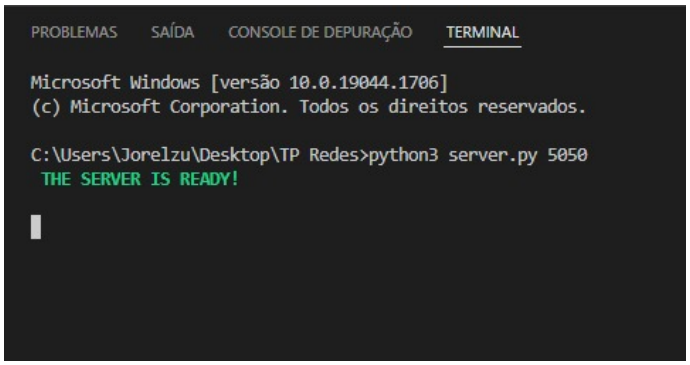
No momento em que o servidor recebe uma mensagem, ela é repassada à função check-Tag. A função é responsável por verificar se é uma mensagem de inserção ou remoção de tags, ao procurar pelos símbolo de mais ou de menos, ou se é uma mensagem normal. Toda mensagem normal é impressa na tela do servidor para indicar que ele a recebeu.

Uma mensagem que não contém as tags(+ e -) pode ou não conter citações, que são tags na mensagem que vêm antecedidas de um caractere hashtag (#). Assim que uma mensagem normal é identificada, ela é repassada à função splitTags, que tem o objetivo de identificar citações em uma mensagem. Ela encontra todas as tags presentes em uma mensagem através de todos os caracteres hashtag() presentes na mensagem. Essas tags identificadas, juntamente com a mensagem e o cliente que a enviou, são passadas à função que envia as mensagens a outros clientes, que é a função sendMsg. Isso é feito através da implementação da função sendMsg. Quando essa função recebe as tags presentes em na mensagem, ela chama a função remove-tag-duplicate, que remove as tags repetidas dentre as tags identificadas. Depois disso, ela percorre a tabela tags-per-client e verifica quais clientes cadastraram cada uma das tags identificadas. Caso um cliente tenha cadastrado uma tag presente na mensagem e esse cliente não foi o cliente que enviou a mensagem, a mensagem é enviada para ele. E para indicar o processo na tela do servidor, é impressa uma mensagem que informa que a mensagem foi enviada e também imprime a mensagem.

6 Execução

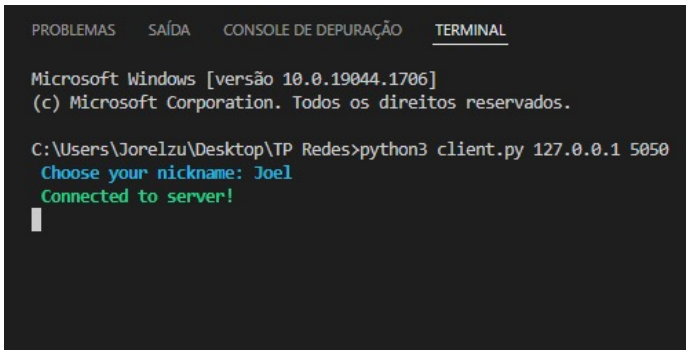
python3 server.py 5050

Figura 3: Execução do servidor.



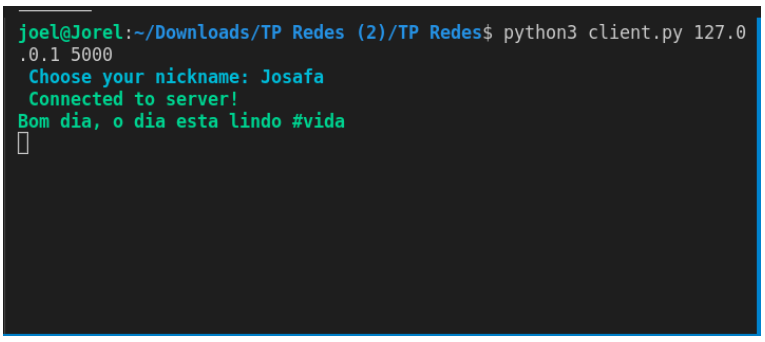
python3 client.py 127.0.0.1 5050

Figura 4: Execução do servidor.



O cliente se conecta ao servidor e logo em seguida envia uma mensagem contendo uma hastag.

Figura 5: Cliente enviando mensagem contendo hastag.



O segundo cliente que está cadastrado na tag +vida, recebe a mensagem contendo determinada tag e em seguida realiza o unsubiscribe para remover a tag de seu dicionário.

Figura 6: Cliente recebendo mensagem contendo hastag.

```
joel@Jorel:~/Downloads/TP Redes (2)/TP Redes$ python3 client.py 127.
0.0.1 5000
 Choose your nickname: Joel
 Connected to server!
+vida
 > subscribed vida
Bom dia, o dia esta lindo #vida
-vida
 > unsubscribed vida
```

O servidor printa cada ocorrência de adição, remoção de tags e envio de mensagens sem as tags(+ e -), servindo para a vizualização de ocorrências durante a execução do programa.

Figura 7: Ocorrências no servidor.

```
joel@Jorel:~/Downloads/TP Redes (2)/TP Redes$ python3 server.py 5000
 THE SERVER IS READY

 Connected with ('127.0.0.1', 46232)
 Nickname is Joel
 Add "vida" para cliente <socket.socket fd=4, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, p
roto=0, laddr=('127.0.0.1', 5000), raddr=('127.0.0.1', 46232)>
 Connected with ('127.0.0.1', 46234)
 Nickname is Josafa
 Mensagem recebida: "Bom dia, o dia esta lindo #vida"
Mensagem "Bom dia, o dia esta lindo #vida" enviada para cliente <socket.socket fd=4, family=AddressFamily.
AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5000), raddr=('127.0.0.1', 46232)>
 Deletando tag "vida" para cliente <socket.socket fd=4, family=AddressFamily.AF_INET, type=SocketKind.SOCK
 STREAM, proto=0, laddr=('127.0.0.1', 5000), raddr=('127.0.0.1', 46232)>
```

7 Conclusão

No decorrer do desenvolvimento da aplicação tive alguns problemas com as implementações das Threads tanto no client, quanto no servidor, visto que não é tão simples de imaginar os processos trabalhando em paralelo. Após a implementação do cliente e servidor utilizando Threads, o primeiro desafio foi criar um dicionário contendo todas as tags, o segundo desafio foi desenvolver funções que tratavam as mensagens que chagavam ao servidor, dado que fora necessário adicionar e remover determinadas tags. Foi muito interessante e desafiador desenvolver este projeto.

8 Referências

https://github.com/Rishija/python_chatServer

https://docs.python.org/3/library/socket.html

https://www.geeksforgeeks.org/socket-programming-python/