Servidor de Mensagens Publish/Subscribe

Nome: Carlos Eduardo da Silva

Introdução

Neste trabalho iremos implementar um servidor e clientes para troca de mensagens de forma similar ao serviço da plataforma Twitter. Clientes enviam mensagens para o servidor informando em quais mensagens estão interessados. O servidor recebe mensagens de clientes e repassa cada mensagem para todos os clientes que estão interessados naquela mensagem. O tópico de uma mensagem é definido pelos *tags* que ela contém. Cada cliente envia ao servidor em quais *tags* está interessado, e o servidor irá repassar ao cliente todas as mensagens que contém pelo menos uma *tag* de seu interesse.

Protocolo

Usando o protocolo TCP, os clientes devem comunicar com o servidor. Os clientes devem informar quais tags estão interessado usando um ‘ + ‘ antes da tag. E clientes podem informar desinteresse por tags usando ‘ - ‘ antes da tag. Exemplo: +musica ou -jogos. E o servidor deve responder com confirmando a inscrição ou a desinscrição na tag com, exemplo: “subscribed +musica” ou “unsubscribed -tag”. Além disso, clientes podem enviar mensagens com um *tag* colocando um caractere ’ # ‘ seguido do identificador do *tag*. Uma mensagem pode ter mais de um *tag*. O servidor replica uma mensagem para todos os clientes que informaram interesse em qualquer um dos *tags* de uma mensagem. Caso um cliente informe interesse em um *tag* pro qual já tenha informado interesse antes, o servidor deve responder com uma mensagem contendo “already subscribed +tag”. Caso um cliente declare desinteresse por uma *tag* pra qual não tenha informado interesse, o servidor deve responder com uma mensagem “not subscribed -tag”. Uma mensagem sem um *tag* pode ser enviada ao servidor, mas não será repassada a nenhum cliente.

Especificações

- O servidor deve enviar no máximo uma cópia de cada mensagem a um cliente independente do número de *tags* na mensagem nas quais o cliente informou interesse.
- As mensagens de interesse (+tag) e desinteresse (-tag) devem ter o sinal (+ ou -) no primeiro caractere e apenas um *tag*, sem nenhum texto adicional.

- O servidor deve esquecer todas as *tags* de interesse de um cliente quando o cliente desconectar-se do sistema. (Em outras palavras, quando um cliente conecta-se ao servidor, ele precisa enviar para os servidor em quais *tags* tem interesse.)
- *Tags* e especificações de interesse e desinteresse devem estar precedidas e sucedidas por espaço, início da mensagem, ou término da mensagem. Em outras palavras, um string como “#dota#overwatch” não será considerado como *tags*.
- As mensagens são terminadas com um caractere de quebra de linha ‘\n’.
- O servidor deve descartar mensagens com caractere(s) inválido(s). O servidor pode desconectar o cliente que enviou a mensagem com caractere inválido, mas precisa continuar a operação sem impacto para os demais clientes conectados. Em outras palavras, o servidor não deve fechar ao receber uma mensagem com caracteres inválidos. Clientes podem simplesmente abortar operação caso recebam mensagens com caractere(s) inválido(s) do servidor.
- mensagens podem ter até 500 bytes e o fim de uma mensagem é identificado com um caractere ‘\n’.
- seu servidor deve fechar todas as conexões e terminar execução ao receber uma mensagem contendo apenas “##kill” de qualquer um dos clientes.
- Seu cliente deve receber mensagens do teclado e imprimir mensagens do servidor na tela. O cliente precisa ser capaz de ler mensagens do teclado e receber mensagens da rede simultaneamente.
- O servidor deve suportar um número arbitrário de clientes. Os clientes podem enviar mensagens a qualquer momento.
- O servidor deve imprimir na saída padrão todas as mensagens recebidas de clientes.
- Seu servidor deve receber um número de porta na linha de comando especificando em qual porta ele vai receber conexões
- Seu cliente deve receber o endereço IP e a porta do servidor para estabelecimento da conexão. (exemplo de execução dos programas em dois terminais distintos)

Desafios e dificuldades

Um dos maiores desafios iniciais, era lidar com vários sockets de cliente enviando requisições para o servidor e pensar em um algoritmo para administrar tudo, porém, como eu teria que salvar quais deles estavam cadastrados em cada tag, ficou um pouco mais claro, como poderia administrar cada um. Tomando um dicionario em python, com a chave sendo a tag e o conteúdo sendo uma lista de sockets que estavam inscritos na tag, ficou bem claro como iria transmitir a mensagem para cada cliente, caso a viesse uma frase com a tag em questão.

Outro grande imprevisto encontrado, foi para encerramento de todos os clientes e servidor a partir de um texto do cliente para o servidor, já que, além do socket de todos os clientes, também teria que lidar (fechar) com a thread em loop dos clientes que estavam recebendo mensagens de texto e as enviando para o servidor. E no servidor o grande problema foi o loop aceitando novas conexões que também deveria ser interrompido.

E para o recebimento de mensagens particionadas em múltiplas partes e recebimento de múltiplas mensagens em uma única, já que todas as mensagens deviam terminar com ‘ \n ‘, foi preciso reestruturar todo o código do programa. E alguns conflitos com o método de ler

carácter/string do teclado ‘ input() ’ já que o ‘ \n ‘ digitado manualmente pelo teclado, não era reconhecido como o ‘ \n ‘ que é a quebra de linha (que estava sendo usado para determinar o fim do texto). Caso o fim do texto recebido pelo cliente não fosse um ‘ \n ‘ o servidor deveria esperar até chegar a outra parte da mensagem, sendo assim um loop, até o fim da mensagem ser um ‘ \n ‘, para sim, transmitir a mensagem a todos os clientes que estivesse inscrito na(s) tag(s).