Relatório TP1
Redes de Computadores

João Marcos Machado Couto
Matrícula:201714421

Janeiro 2021

1 Introdução

Sob o desafio de implementar uma infraestrutura basica de comunicação entre um servidor e vários clientes concorrentes utilizando o paradigma Publish/Subscribe, este trabalho prático provou-se altamente desafiador sobretudo devido às restrições impostas na definição do TP acerca do tráfego de dados entre os vários endpoints da rede, que acabaram causando refactors tão severos em relação ao produto minimo viavel do TP que quase triplicaram o tempo de implementação.

2 Filosofia basica da implementação

Afim de permitir que os clientes consigam enviar e receber pacotes do servidor simultaneamente, fez-se necessária, na aplicação do cliente a criação de threads dedicadas para estas duas tarefas. De forma analoga, no servidor, após iniciar a thread "mãe" de cada cliente implementada no código do profesor Italo, fiz com que esta thread criasse duas novas threads, uma dedicada para receber as mensagens de cada cliente e outra dedicada para fazer o processamento de novas mensagens vindas de outros clientes.

Além disso, alterei a thread "mãe" de cada cliente de forma que ela aloque um vector de strings, que é então passado por referencia junto do parametro de dados para as threads filhas. Esse vector thread simplesmente armazena quais são as tags de interesse de cada clientes, adicionadas e removidas através de comands "+tag" e "-tag".

O broadcast de mensagens entre usuários se da da seguinte maneira: no momento em que a thread de recebimento de mensagens de um usuário recebe uma nova mensagem, essa mensagem é empilhada numa estructura global vector de strings que armazena todas as mensagens enviadas pelos vários clientes de forma ordenada. A modificação dessa estrutura via push-back de novas mensagens é controlada por um único mutex que garente que apenas uma thread esteja alterando a estrutura em cada ciclo do processador.

A ideia é então que a thread de processamento de novas mensagens criada para cada cliente vai manter um contador local no qual ela guarda o indice da última mensagem do vector global que ela processou e, assim, avaliando a cada iteração o tamanho do vector global de mensagens e comparando com esse contador local, ela consegue determinar se existem novas mensagens que ela deve processar e decidir se deve ou não envia-la para seu cliente.

Essa decisão acontece da seguinte maneira: cada nova mensagem a ser processada é divididas em tokens separados por espaço. Neste ponto basta percorrer o vector de tags de interesse do cliente e identificar se algum dos tokens obtidos da mensagem são da forma "#tagDeInteresse", se for o caso, a thread então envia a mensagem para o cliente e encerra a iteração sem verificar as tags restantes na lista para evitar o envio de mensagems duplicadas no caso da mensagem conter mais de uma tag de interesse do cliente.

3 Desafios e soluções

- *Validade de caracteres*: para conseguir utilizar uma solução única usavel em todos os pontos do programa que necessitassem de verificação de caracteres variando apenas o conjunto de caracteres válidos, optei pelo uso da função strspn. Ela recebe dois pointers para array de chars e retorna o tamanho da maior substring da primeira string que pode ser formada utilizando apenas os characteres presentes na segunda string. Assim, se o retorno for igual ao strlen da primeira string então ela é

integralmente composta apenas por caracteres presentes na segunda string. Bastou então criar os dois conjuntos de caracteres: um para os caracteres permitidos nas tags (letras) e outro para os caracteres permitidos nas mensagens (letras, números, os caracteres de pontuação ,.?!:;+-*/=@$%()[]{} e espaços)

- *\0 não podem ser transmitidos na rede*: para assegurar essa restrição, bastou restringir o número de bytes em cada transmissão send(): transmitindo um byte a menos do que o strlen() de cada mensagem, elimina-se o . Dai com um append() imediatamente antes da tranmissão asseguramos também que todas as mensagens terão seu ultimo byte identificado com um newline

- *Mensagens não podem ter mais de 500 bytes*: optei por manter os buffers no mesmo tamanho da implementação base do professor Italo e então processei o dado procurando por um entre os primeiros 500 bytes utilizando a função memchr. Na ausencia de um match, o servidor entende que o cliente entrou em um estado fatal de bug, portanto o loop central da thread de recebimento é quebrado. Neste momento, a thread mãe, que associou um pthread_join com a thread de recebimento de mensagens do cliente consegue seguir com sua execução, matando a thread de envio de mensagens, fechando o socket do cliente e por fim rodando um EXIT_SUCESS, terminando de vez todos os processos associados do cliente que mandou a mensagem acima do limite de bytes.

- *O comando kill*: para implementar essa feature da comunicação, optei por detectar o envio do comando diretamente na thread de recebimento de mensagens dos clientes rodando no servidor usando um strcmp, ao invés de deixa-la passar pelo pipeline de processamento primeiro. No momento em que o comando é tedectado, seto o int global "kill" para 0,faço um break no loop central para matar a thread de recebimentos e por fim faço uma chamada shell para abrir um novo cliente no intuito de fazer a main do servidor sair do accept() para que ela então reavalie a condição do loop while central da main, que, em cada iteração, avalia o valor de kill. Como ele estara em 0, o loop while não sera executado e isso fara com que a execução da main chegue em um exit que finalizara por consequencia todas as threads de servidor abertas apartir dela.

- *Recebimento de mensagens particionadas em múltiplas partes e multiplas mensagens em um único recv*:certamente o maior desafio do tp, pois exigiu um refactor extenso de minha implementação v0. Resolvi ambos os desafios com um mesmo mecanismo: ao receber um novo pacote de recv primeiramente a thread avalia se no buffer de recebimento temos um \n na ultima posição pois se não for o caso, isso significa que os bytes finais no buffer são na verdade o inicio de uma mensagem cuja continuação vira no próximo recv. Tendo feito isso, bastou então utilizar a função strtok para dividir o buffer em tokens separados por \n que, por definição do protocolo definido pelo TP, representam mensagens individuais.

    · Aqui vale ressaltar que em cada iteração, na verdade utilizo dois iteradores que recebem ponteiros do strtok: um deles, aponta para o local da mensagem corrente que será processada e o outro para a proxima mensagem. O que isso me permite fazer é que se o ponteiro para proxima mensagem for NULL isso significa que o token corrente ja é o ultimo segmento da mensagem, juntando este fato com o conhecimento de que os bytes finais do buffer são na verdade os bytes finais de uma mensagem que ainda estar por vir, basta fazer um break do loop de processamento e inicializar o buffer da proxima iteração do recv com os bytes apontados pelo ponteiro da mensagen corrente da iteração anterior. Assim consigo garantir a integredades de mensagens espalhadas e também o eventualmente processamento de multiplas mensagens dentro de um mesmo buffer de recv.

4 Conclusão

Optei por um approach mais holistico para este TP. Em se tretar de um trabalho que poderia ser implementado em C/C++ fiz questão de aproveitar cada desafio para exercitar a maior quantidade possivel de conceitos e patterns da linguagem buscando uma implementação limpa, sem pular nenhuma restrição mais complicada imposta pelo tp e com menor quantidade possivel meia-soluções "gambiarras". Ademais o TP me permitiu um primeiro contato bastante produtivo com um uso real para threads.