

Universidade Federal de Minas Gerais
Escola de Engenharia
DCC023 - Redes de Computadores
Trabalho prático 01: servidor de mensagens
Júlia de Backer Pacífico - 2019021476
Junho de 2021

I. Introdução

No vigésimo ano de 2021, entidades governamentais buscam combater a pandemia do coronavírus através da vacinação da população. Nesse contexto, o primeiro trabalho prático da disciplina de Redes de Computadores consistiu na implementação de um sistema servidor e clientes, que viabiliza a troca de mensagens para que um usuário encontre um ponto de vacinação próximo a uma localização fornecida.

Assim, as coordenadas esperadas na troca de mensagens deste programa serão chamadas genericamente de X e Y, de forma que ambas assumem valores maiores ou iguais àqueles entre 0 e 9999. Do cliente, quatro tipos de mensagem podem ser enviadas: comandos para listar as localizações disponíveis, para retorno da localização mais próxima a uma fornecida e para adição ou remoção de novos locais de vacinação. Poderão ser adicionados até 50 locais de vacinação, e o tamanho das mensagens trocadas não deve ultrapassar 500 bytes.

II. Implementação

O trabalho foi implementado por meio da ferramenta WSL -Windows Subsystem for Linux, em linguagem C. A primeira parte da produção deu-se pelo acompanhamento das aulas, disponibilizadas pelos professores da disciplina, de Introdução à Programação em Redes[2]. Assim, foi possível desenvolver um servidor que, por meio de threads, consegue receber mensagens de diferentes clientes, independente da ordem em que eles enviam as mensagens. O servidor criado também permite a conexão tanto por IPv4 quanto por IPv6, devido à implementação do método “socket parser”.

O servidor criado armazena os dados dos locais de vacinação em uma matriz 50x2, inicializada com valores negativos, uma vez que eles não constituem valores válidos para coordenadas geográficas. A primeira coluna da matriz se destina aos valores de X, e a segunda, aos de Y. Dessa forma, ela é preenchida ao comando do cliente, após feita uma verificação quanto à preexistência da localidade por ele fornecida. A mesma verificação é feita ao comando de se deletar um certo par ordenado. Assim, ao se remover uma localização especificada, a posição que ele anteriormente ocupava volta a ser preenchida por valores negativos.

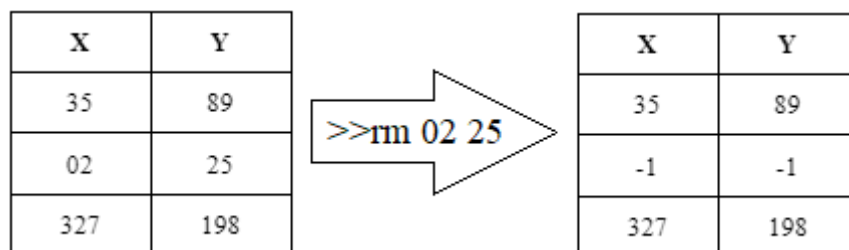


Figura 1 - Representação da matriz de dados após o comando de remoção.

O comando “query” retorna ao cliente a localidade mais próxima à uma por ele fornecida, inferida pelo algoritmo da distância Euclidiana. Assim, uma vez que o comando para remoção de uma localidade da matriz de dados pode, eventualmente, romper com a continuidade dos valores não nulos na matriz de dados, o algoritmo para lidar com o comando query percorre todas as suas 50 linhas para

calcular as distâncias das coordenadas conhecidas até o ponto fornecido pelo cliente, ignorando os valores negativos. Feitas as comparações de magnitude, a função implementada retorna o índice da linha da matriz que contém o par ordenado requisitado. Por último, o comando “list” faz uma simples busca por posições preenchidas na matriz de dados, e concatena os valores das localidades existentes em uma única string, que é então atribuída diretamente ao buffer de envio.

Todas as funções para tratamento de mensagens vindas do cliente foram implementadas em um arquivo modularizado, e são chamadas dentro de uma única função **handle()**, criada para preencher o buffer de envio conforme o comando requisitado. Essa função foi criada no intuito de minimizar o tamanho do código principal do servidor.

A implementação do programa cliente conta com um laço de repetição, que garante a troca de mensagens ininterruptas entre ele e o servidor. A conexão só é finalizada pelo servidor ao mando do cliente, pela solicitação do comando “kill”.

III. Desafios

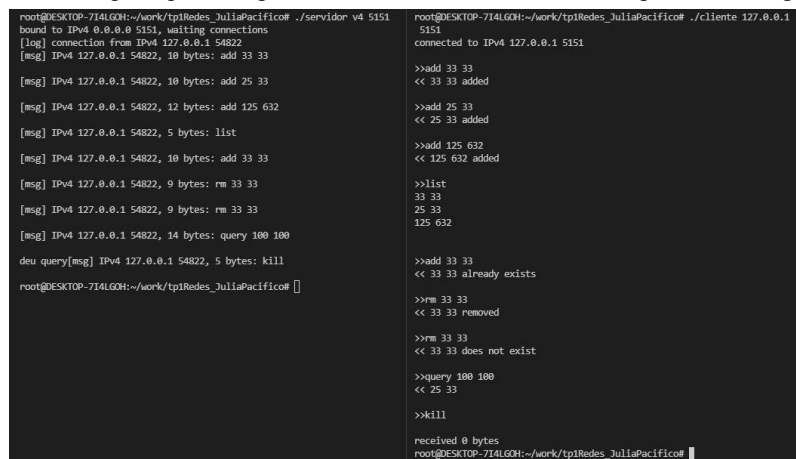
O maior desafio encontrado durante a implementação foi garantir o envio de múltiplos requisitos de um mesmo cliente ao servidor. Em dado momento da produção deste trabalho, o cliente implementado garantia o envio de várias mensagens ao servidor, mas apenas a primeira delas recebia uma resposta. Depois disso, o programa do servidor indicava falha de segmentação, ainda que preenchesse o buffer de envio corretamente. Descobriu-se posteriormente que o erro era causado pela incompatibilidade do tamanho da mensagem esperada com o da variável “total” no programa cliente, que indicava o tamanho de bytes de cada mensagem trocada. Ao se zerar a variável em cada nova passagem pelo laço de repetição do cliente, o problema foi sanado.

Dificuldades adicionais devido ao baixo nível da linguagem C incluíram a manipulação de strings e a criação de funções que retornam apenas valores únicos ao invés de arrays multidimensionais. Essas dificuldades foram contornadas de forma que a maioria das funções implementadas não possuem tipo algum de retorno. Adicionalmente, a falta da criação de um *backup* para os arquivos implementados, inicialmente armazenados apenas em um diretório local, foi custosa, pois em dado momento da produção deste trabalho, o arquivo principal foi irrecuperavelmente perdido. Assim, foi necessário implementá-lo do início duas vezes antes da entrega marcada.

Ademais, o servidor atual contém algumas falhas não sanadas; dentre elas: o não reconhecimento de mais de um comando por mensagem e o não tratamento de mensagens desconhecidas. Por isso, de modo geral, o projeto executado é limitado às estritas especificações do roteiro.

IV. Resultados e Conclusão

Os programas servidor e cliente implementados neste trabalho se comunicam com êxito quando utilizados quaisquer dos protocolos IPv6 e IPv4, como exemplificam as figuras 2 e 3.



```
root@DESKTOP-714L6QH:~/work/tp1Redes_JuliaPacifico# ./servidor v4 5151
bound to IPv4 0.0.0.0 5151, waiting connections
[log] connection from IPv4 127.0.0.1 54822
[msg] IPv4 127.0.0.1 54822, 10 bytes: add 33 33
[msg] IPv4 127.0.0.1 54822, 12 bytes: add 125 632
[msg] IPv4 127.0.0.1 54822, 5 bytes: list
[msg] IPv4 127.0.0.1 54822, 10 bytes: add 33 33
[msg] IPv4 127.0.0.1 54822, 9 bytes: rm 33 33
[msg] IPv4 127.0.0.1 54822, 9 bytes: rm 33 33
[msg] IPv4 127.0.0.1 54822, 14 bytes: query 100 100
deu query[msg] IPv4 127.0.0.1 54822, 5 bytes: kill
root@DESKTOP-714L6QH:~/work/tp1Redes_JuliaPacifico#
```

```
root@DESKTOP-714L6QH:~/work/tp1Redes_JuliaPacifico# ./cliente 127.0.0.1 5151
connected to IPv4 127.0.0.1 5151
>>add 33 33
<< 33 33 added
>>add 25 33
<< 25 33 added
>>add 125 632
<< 125 632 added
>>list
33 33
25 33
125 632
>>add 33 33
<< 33 33 already exists
>>rm 33 33
<< 33 33 removed
>>rm 33 33
<< 33 33 does not exist
>>query 100 100
<< 25 33
>>kill
received 0 bytes
root@DESKTOP-714L6QH:~/work/tp1Redes_JuliaPacifico#
```

Figura 2 - Teste para conexão em IPv4.

```
root@DESKTOP-7I4L6QH:~/work/tp1Redes_JuliaPacífico# ./servidor v6 51511
bound to IPv6 :: 51511, waiting connections
[log] connection from IPv6 ::1 57415
[msg] IPv6 ::1 57415, 13 bytes: add 1526 233
[msg] IPv6 ::1 57415, 12 bytes: add 152 222
[msg] IPv6 ::1 57415, 12 bytes: add 152 588
[msg] IPv6 ::1 57415, 14 bytes: query 500 255
deu query[msg] IPv6 ::1 57415, 11 bytes: rm 152 222
[msg] IPv6 ::1 57415, 13 bytes: quer 500 225
deu query[msg] IPv6 ::1 57415, 5 bytes: list
[msg] IPv6 ::1 57415, 5 bytes: kill
root@DESKTOP-7I4L6QH:~/work/tp1Redes_JuliaPacífico#

root@DESKTOP-7I4L6QH:~/work/tp1Redes_JuliaPacífico# ./cliente ::1 51511
connected to IPv6 ::1 51511
>>add 1526 233
<< 1526 233 added
>>add 152 222
<< 152 222 added
>>add 152 588
<< 152 588 added
>>query 500 255
<< 152 222
>>rm 152 222
<< 152 222 removed
>>quer 500 225
<< 152 588
>>list
1526 222
152 588
>>kill
received 0 bytes
root@DESKTOP-7I4L6QH:~/work/tp1Redes_JuliaPacífico#
```

Figura 3 - Teste para conexão em IPv6.

Concluiu-se, findada a execução deste trabalho, que uma das questões mais pertinentes na programação em redes é a compatibilidade do tamanho das mensagens esperadas pelo servidor e pelo cliente. Para além da prática com a linguagem C, a implementação deste trabalho também foi proveitosa para compreender de forma prática os conceitos trabalhados ao longo das aulas. Apesar das dificuldades supracitadas, o servidor implementado apresentou resultados satisfatórios.

V. Fontes

- [1] “Sockets In C”, Michael J. Donahoo, Kenneth L. Calvert;
- [2] “Introdução à Programação em Redes” - <https://www.youtube.com/watch?v=tJ3qNtv0HVs&list=PLyrH0CFXIM5Wzmbv-IC-qvoBejsa803Ok>
- [3] “Linguagem C: Completa e Descomplicada”, André Backs.