TP 2 - Rede de Computadores

Daniel Augusto Costa de Sá - 2019041515
daniel.augusto191@gmail.com
UFMG - 2023

Introdução

O trabalho visa implementar uma aplicação de troca de mensagens entre multiplos clientes por socket tanto para IPV4 quanto para IPV6, ultilizando a linguagem C e multithreading.

Um Cliente pode se conectar a um servidor de troca de mensanges, enviar conversas privadas para outros clientes que usem o mesmo servidor, e enviar mensagens para todos os outros clientes que usam o servidor.

Implementação e execução

Para implementar tanto servidor quando cliente, fiz uso das bibliotecas padrao de C, (stdio, stdlib), a biblioteca string.h para manipulação dos arquivos a serem enviados, e unistd, types.h, socket.h, inet.h, para a implementação da comunicação cliente-servidor, time.h para tratamento de tempo entre mensagens, e pthread.h para permitir multiplas conecções ao servidor.

No mais, a implementação segue um script simples, lê do teclado no cliente, o comando a ser executado, faz um parser na mensagem enviada e executa o perspectivo comando.

Para simular os comandos usei algumas estruturas:

* Client_data - Um struct que armazena dados de um cliente como ID, socket de mensagens para ele e endereço.
* Client_list - Uma lista de Client_data que armazena todos os clientes conectados a rede no momento.
* Client_check - Uma lista de inteiros que guarda se o i-ésimo cliente está conectado. Não nescessario mas achei a implementação mais eficiente em tempo de execução, trocando o espaço que é no max O(15) constante.
* mutex, closed - Usei para sincronização de multithreads, explicado mais adiante.
* MAX_CONN - Número máximo de conecções.
* BUFSZ - 2048 buffer de mensagens.

Para a parte de conecção foi simples, apenas adicionais uma thread para cada conecção que chegava, assim o escalador por si só, executa as trheads e trata cada cliente no servidor. No cliente também fiz o uso multithreads para lidar, como as mensangens que envio, e as que rescebo, provavelmente havia um jeito melhor com broadcast, pórem acredito que essa com multithreads, seja valida, dado a especificação.

Basicamente, quando um cliente conecta ao servidor uma thread é spawnada que fica ouvindo o servidor em caso de uma mensagem de outro cliente, enquando que a thread main, fica esperando o input do proximo comando, no momento que o input é feito, para não gerar troca de dados entre as threads durando a troca de contexto, uso a variavel mutex, mensionada mais cedo, para travar a thread que sempre está esperando o servidor, após o comando ser executado libera mutex, e continua. Ao fim da execução a variavel closed é setada e assim a thread sabe que pode ser liberada.

Na abertura de conecção é verificado qual o primeiro ID que ainda não foi ultilizado para ser atribuido a aquele que o chamou.

Caso o cliete mande "close connection" o servidor simplesmente seta o cliente_check no id desse user como 0 (vantagem dessa implementação, não precisar remover da lista de clientes original), e envia o confimamento de fechamento de conecção para o cliente, que muda closed para 1 para remover a outra thread e fecha o socket.

Caso mande "list users", o servidor passa pelo cliente_check olhando quais clientes estão ativos com a excessão de quem pediu (conclui assim baseado nos exemplos da especificação), atribui eles a uma mensagem que é mandada de volta para o cliente, que a imprime.

Caso mande "send to IdUser {Message}", verifica se o IdUser esta setado em cliente_check (tempo constante na lista), e se não estiver, mando um erro para quem enviou. Se estiver confirma a mensagem mandando de volta para o cliente, e encaminha para o cliente com o ID que foi passado, fazendo um parser de hora e minuto que chegou ao servidor.

Caso mande "send all {Message}", manda a confirmação de volta para quem enviou a mensagem, e depois passa por toda a lista verificando os clientes ativos, se estiver ativo envia para aquele socket.

No mais de implementação é isso, acho que o ponto chave aqui foi achar o ponto entre multithreads e a conecção.

Notas e observações

Novamente a implementação não foi tão trabalhosa da maneira que fiz, a playlist e o livro disponibilizados ajudaram muito. Apenas disso eu não sei se havia uma unica maneira de fazer esse trabalho, ou se a maneira que era requirida tem melhor desempenho ou algo a mais. Eu por exemplo usei multi-threads no cliente não sei se era nescessario, se havia algo em socket que ajudava nisso, mas dei o meu jeito para implementar, ele passou em todos os 4 exemplos e mais alguns testes que fiz então acho que confio na respoosta. Essa foi com certeza a parte mais dificil, tive que pensar nessa solução e acho que acabou sendo o caminho mais dificil.

A Especificação do trabalho também me deixou com algumas duvidas, estava bem explicado mas algumas coisas me pareciam um pouco complicadas, por exemplo eu não usei aquela tabela de tipos de comandos de controle, fiz o parser por mim mesmo dos comandos. No mais ficam algumas observações da implementação:

- Usei o caracter \0 como identificador de fim de mensagem mesmo, como não foi especificado igual no TP1 e se não me engano houve uma resposta no forum de que podia ser assim. Então deixei como o \0 mesmo.

- Imagino que a biblioteca socket, tenha algum recurso para enviar a todos de uma vez uma mensagem, porem implementei por mim mesmo usando threads. Espero que não mude muito na questão do resultado do trabalho, já que meio que fiz um broadcast na mão o que não foi muito esperto, mas acho ser válido.

- As mensagem, tanto privadas quanto pro all, assumi que estão entre aspas e a mensagem deve ser mostrada na tela sem eles, segundo o exemplo da documentação.

- Com forme foi discutido no forum, quando há um comando errado, simplemente ignoro.

Referencias

- man pthreads
- linux.die.net/man
- man linux
- Cap. 2,3 do livro programação de sockets, sugerido na especificação.
- Playlist sugerida na especificação
- Notas de aula