UNIVERSIDADE FEDERAL
DE MINAS GERAIS

CARLOS HENRIQUE BRITO MALTA LEÃO
MATRÍCULA: 2021039794

TRABALHO PRÁTICO 02

Redes de Computadores

Belo Horizonte
2024

INTRODUÇÃO

Este trabalho prático visa explorar a comunicação entre clientes e servidores em redes de computadores, utilizando a linguagem de programação C. Inspirado pelo funcionamento de aplicativos de streaming de filmes e séries, foi desenvolvida uma aplicação console que simula esse processo básico. Por meio dessa simulação, é objetivado não apenas exercitar, mas também compreender os fundamentos práticos da comunicação cliente-servidor usando conexão UDP.

Além de exercitar a implementação de sistemas distribuídos em C, os objetivos específicos deste projeto incluem familiarizar-se com o uso de sockets para comunicação entre processos, simular um sistema de solicitação e resposta em tempo real e explorar os detalhes da conexão UDP. Ademais, também é explorado o âmbito da conexão de múltiplos clientes no servidor ao mesmo tempo, com a utilização de threads.

Descrição do Projeto

A aplicação console replica o fluxo básico de interação entre usuários e o servidor, que realiza o streaming de vídeos. Os clientes desempenham o papel dos usuários que utilizam os serviços de transmissão, enquanto o servidor realiza essa transmissão. A interação é estabelecida por meio de uma comunicação cliente-servidor baseada em UDP, permitindo uma troca confiável e ordenada de mensagens. Além disso, a aplicação aceita conexões IPV4 ou IPV6, basta informar durante a execução.

A implementação da comunicação cliente-servidor neste projeto baseia-se no uso de sockets. Os sockets são mecanismos que permitem a comunicação entre processos, sejam eles locais ou remotos. No contexto deste projeto, eles são essenciais para estabelecer a conexão entre o cliente e o servidor, facilitando a troca de mensagens em tempo real.

O uso de sockets com conexão UDP proporciona uma comunicação rápida e eficiente, porém sem garantia de entrega ou ordem dos dados. Enquanto o UDP oferece uma abordagem não orientada à conexão, que pode ser ideal para aplicações que requerem baixa latência e uma comunicação leve, o TCP, por outro lado, poderia ser mais adequado para cenários que exigem confiabilidade e garantia de entrega ordenada.

Nesse âmbito, o processo de comunicação entre cliente e servidor segue um fluxo de comunicação sem a necessidade de estabelecimento de conexões. Primeiramente, o servidor configura um socket e fica aguardando pacotes de dados. O cliente então envia um pacote diretamente para o servidor, utilizando o endereço IP e a porta adequados. Como o UDP é um protocolo sem conexão, não há uma etapa de estabelecimento de conexão ou negociação. Cada mensagem enviada é independente, e a comunicação pode ocorrer de forma contínua enquanto ambos os processos permanecerem ativos. Para finalizar a comunicação, o servidor envia uma mensagem com o código ‘END’.

DESCRIÇÃO DO CÓDIGO

O código fonte desse projeto foi desenvolvido na linguagem de programação C e utiliza bibliotecas padrões da linguagem compatíveis com o sistema operacional Linux. Os códigos do Servidor e do Cliente estão implementados em arquivos separados, respectivamente *server.c* e *client.c*. O projeto também apresenta um arquivo Makefile, que gerencia a compilação dos arquivos fonte e a geração dos executáveis para o servidor e o cliente. Para compilar o programa, basta acessar a pasta raiz do projeto e executar o comando *make*.

Servidor

O servidor deve receber alguns parâmetros para realizar suas configurações de conexão. Para isso, durante a execução, é preciso informar dois parâmetros. Primeiramente, é necessário especificar o tipo de IP que o servidor utilizará. O segundo parâmetro deverá ser a porta utilizada. Seguem exemplos de comandos de execução:

- ./server ipv4 50501
- ./server ipv6 50501

Com os parâmetros informados, o cliente pode definir o tipo de IP a ser utilizado e também a porta do servidor. Isso é possível graças à estrutura *addrCriteria*, que é utilizada para definir os critérios para a configuração do endereço do servidor. Essa estrutura pertence à família *addrinfo*, que auxilia na criação de sockets de rede. A *addrinfo* fornece um formato padrão para especificar informações de endereçamento, o que facilita a inicialização de sockets tanto para IPv4 quanto para IPv6.

Com essa estrutura definida, é possível utilizar a função getaddrinf, que é chamada com essa estrutura para obter uma lista de endereços que correspondem aos critérios especificados. Estes são: a porta que o servidor utilizará e a própria estrutura addrCriteria, que contém todas as opções e critérios de endereçamento configurados. Com isso, a função armazena uma lista de endereços que correspondem aos critérios estabelecidos em uma estrutura chamada addrinfo.

Após obter a lista de endereços, o socket é criado e vinculado ao endereço apropriado. É possível criar o socket do servidor utilizando a função socket. Essa função cria um ponto final de comunicação e recebe três parâmetros: a família de protocolos (já definida anteriormente), o tipo de socket (no nosso caso,

SOCK_DGRAM que indica que o socket será usado para datagramas, que é o modo de operação do UDP) e o protocolo de rede que o socket usará (IPPROTO_UDP para o UDP).

Em seguida, com o socket criado, é possível dar continuidade no fluxo de comunicação, associando o socket do servidor ao endereço IP e porta. Para isso, utiliza-se a função bind recebendo como parâmetro a referência ao socket previamente criado, a especificação do endereço a ser associado (a estrutura de endereço criada anteriormente) e o tamanho da estrutura apontada pelo endereço, em bytes. Essa função associa o socket criado a um endereço específico. Isso é necessário para que o servidor possa receber pacotes enviados a esse endereço.

Por fim, basta realizar a liberação da lista de endereços, objetivando a liberação de memória alocada para a estrutura de endereços que foi preenchida pela função getaddrinfo. Após a criação do socket e a vinculação ao endereço, a lista de endereços retornada por essa função não é mais necessária.

Após toda essa configuração, o programa servidor cria a printThread, uma thread que é responsável por imprimir no console o número de clientes a cada quatro segundos. A biblioteca utilizada para toda a criação de threads do lado do servidor é a biblioteca padrão pthread.

Em seguida, o programa entra em seu loop principal, em que aguarda por conexões e gerencia a comunicação com os clientes. Ele funciona continuamente para receber e criar threads para processar cada solicitação de cliente. Em cada iteração, o programa aloca memória para uma nova estrutura clientInfo. Essa estrutura armazenará informações sobre o cliente, como o descritor do socket, a opção escolhida pelo cliente, a última frase enviada, e o endereço do cliente. Segue a composição da estrutura clientInfo:

```
struct clientInfo {
  int socket; // Socket do cliente
  int choice; // Opção escolhida pelo cliente
  int lastPhrase; // Última frase exibida
  struct sockaddr_storage sockaddr; // Endereço do cliente
};
```

O servidor então prepara para receber dados do cliente. Ele define o comprimento da estrutura de endereço do cliente (sockaddr) e aloca um buffer para

armazenar a mensagem recebida. A função **recvfrom** é chamada para ler uma mensagem enviada por um cliente. Essa função espera receber pacotes do socket, armazenando a mensagem recebida no buffer e preenchendo a estrutura de endereço do cliente com a origem do pacote. A função recebe como parâmetros: descritor do socket de onde os dados serão lidos; buffer onde os dados recebidos serão armazenados; tamanho máximo que pode ser armazenado no buffer; ponteiro para a estrutura que será preenchida com o endereço do remetente; tamanho desta estrutura.

Se a mensagem for recebida com sucesso, o programa define o socket, que é armazenado na estrutura de informações do cliente. Em seguida, a mensagem é interpretada, ou seja, é recebida e convertida para um número inteiro, que representa a escolha de filme do cliente.

Com todas as informações do cliente configuradas, o programa cria uma nova thread para processar a solicitação do cliente, através da função clientHandler. Assim, é possível que o servidor estabeleça a conexão com vários clientes ao mesmo tempo, processando toda a comunicação em uma thread totalmente isolada e independente.

Com a conexão com um cliente estabelecida e sua respectiva thread criada, começa o funcionamento da função clientHandler. Primeiramente, é verificada se a opção escolhida pelo cliente é uma opção válida, e logo após incrementa o número de clientes conectados no servidor. Para realizar o incremento dessa variável global, é preciso realizar um tratamento de sincronização utilizando um mutex, para garantir que o incremento do contador é realizado de forma segura, prevenindo condições de corrido devido ao multithreading.

Em seguida, a função entra em um loop, que envia as frases do filme escolhido para o cliente de três em três segundos. Para o envio dessas mensagens, é utilizada a função **sendTo**, que envia dados para o cliente através do socket UDP. Esta recebe como parâmetros: descritor do socket usado para enviar os dados; buffer contendo a mensagem a ser enviada (no caso, a frase do filme); tamanho da mensagem; endereço do cliente para o qual dados são enviados; tamanho da estrutura do endereço. Se a função executar corretamente, ela retorna o número de bytes enviados.

Esse processo ocorre até que todo o conteúdo do filme seja enviado. Em seguida, a função **sendTo** é utilizada novamente para enviar uma mensagem para o

cliente que estabelece o fim da conexão. Essa mensagem, apresenta o código ‘END’. Por fim, ao finalizar a conexão, o número de clientes conectados é decrementado, novamente utilizando o mutex para fazer a alteração segura.

Em conclusão, o servidor UDP implementado gerencia a transmissão de frases de filmes para vários clientes, usando threads para processar cada conexão de forma independente. Utilizando sockets e a função **sendto**, o servidor envia mensagens de maneira eficiente e sem conexão, mantendo a sincronização com mutexes para o controle seguro do contador de clientes. Essa arquitetura permite que o servidor lide com múltiplas solicitações simultâneas, garantindo uma comunicação eficaz e escalável.

Cliente

O cliente é responsável por solicitar filmes ao servidor UDP. Durante a execução, ele deve receber três parâmetros na linha de comando: o tipo de IP a ser utilizado (ipv4 ou ipv6), o endereço IP do servidor e a porta utilizada pelo servidor. Abaixo estão exemplos de comandos de execução:

- ./client ipv4 127.0.0.1 50501
- ./client ipv6 ::1 50501

Após o início da execução, o cliente coleta os parâmetros fornecidos na linha de comando: o tipo de IP (ipv4 ou ipv6), o endereço IP do servidor e a porta. Com base no tipo de IP, ele configura a estrutura addrinfo para especificar que deseja usar o protocolo UDP (SOCK_DGRAM) e o tipo de endereço apropriado (AF_INET para IPv4 ou AF_INET6 para IPv6). A função *getaddrinfo* é então utilizada para converter o endereço IP e a porta do servidor em uma estrutura de endereço utilizável pelo socket.

O cliente exibe um menu interativo ao usuário, permitindo a escolha de três opções de filmes ou a saída do programa. As opções são apresentadas em um loop contínuo, onde o usuário pode optar por uma das seguintes ações:

- **0 - Sair**: O programa é encerrado.
- **1 - Senhor dos Anéis**
- **2 - O Poderoso Chefão**
- **3 - Clube da Luta**

Quando o usuário escolhe um filme, o cliente cria um socket para comunicação UDP. Este socket é criado usando a função socket, que leva em consideração a família de endereços (IPv4 ou IPv6), o tipo de socket (SOCK_DGRAM para datagramas, que são usados em UDP), e o protocolo (IPPROTO_UDP para UDP). Caso a criação do socket falhe, um erro é reportado, e o programa tenta continuar ou encerrar conforme apropriado.

A escolha do filme é convertida em uma string e enviada ao servidor utilizando a função **sendto**. Essa é a mesma função utilizada no código do servidor, então apresenta o mesmo funcionamento e recebe os mesmos tipos de parâmetros. Após enviar a solicitação, o cliente entra em um loop para receber as respostas do servidor. A função **recvfrom** é usada para isso, permitindo a recepção de mensagens do servidor sem uma conexão prévia. Também é a mesma utilizada e descrita na seção de descrição do código do servidor.

O buffer com a mensagem recebida é então lido e a frase do filme é exibida na tela do usuário. O loop continua até que o cliente receba a mensagem com o código que determina o fim da conexão, que indica que todo o filme já foi transmitido. Nesse sentido, após a recepção de todas as frases do filme, o cliente finaliza a comunicação fechando o socket. Isso é feito com a função **close**, que libera os recursos associados ao socket e encerra a comunicação. O cliente então retorna ao menu inicial, permitindo ao usuário escolher outro filme ou sair do programa.

Esta arquitetura permite uma comunicação eficiente e sem conexão entre o cliente e o servidor. A utilização do UDP facilita a implementação de um protocolo simples e rápido para a transmissão de mensagens, adequado para o envio de frases de filmes, onde a ordem e a confiabilidade estrita de entrega não são críticas.

EXEMPLOS DE USO

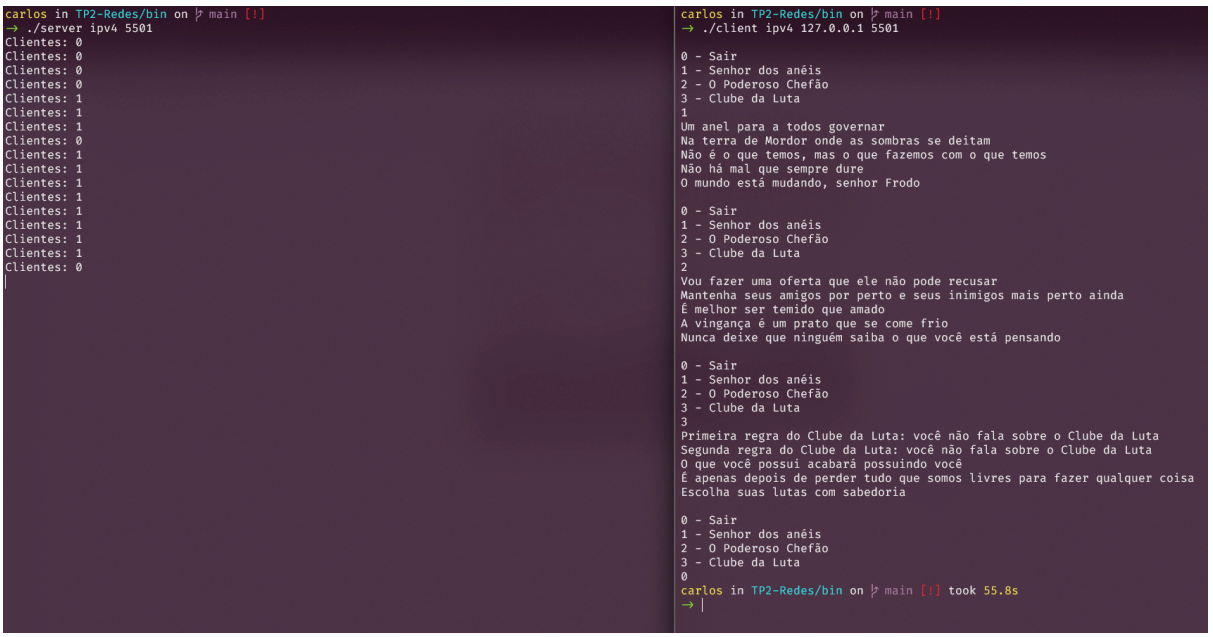
Nesta seção, serão apresentados exemplos de uso do programa em funcionamento, demonstrando sua interação entre cliente e servidor. Por meio de exemplos práticos, será ilustrado como executar o programa, fornecer os parâmetros necessários e acompanhar a comunicação entre o cliente e o servidor durante a transmissão de um filme. Cada exemplo será acompanhado por prints que destacam as principais etapas da execução, proporcionando uma compreensão visual do processo. Esses exemplos ajudarão os usuários a entenderem melhor como utilizar e interagir com a aplicação em diferentes cenários de uso.

Exemplo 1 - Cliente assiste filmes e encerra programa (IPv4)

Nesse caso de uso, o servidor será executado para receber conexões IPV4 na porta 5501. Nesse âmbito, o cliente receberá o endereço do servidor (endereço local: 127.0.0.1), a porta e o tipo IPV4.

Nesse exemplo, o usuário inicia a aplicação e escolhe o filme ‘1 - Senhor dos Anéis’. Dessa forma, as frases do filme começam a ser impressas na aplicação do cliente, e, enquanto isso, do lado do servidor, o número de clientes conectados muda para 1. Em seguida, quando o filme termina, o número de clientes conectados volta para 0. Em seguida, o usuário repete todo esse processo, escolhendo os outros dois filmes: ‘2 - O Poderoso Chefão’ e ‘3 - Clube da Luta’. Por fim, o usuário escolher a opção ‘0 - Sair’ e o programa, na parte do cliente finaliza, e o servidor volta a informar o número de clientes conectados igual a 0.

Na imagem a seguir, é possível visualizar todo esse fluxo do funcionamento tanto do servidor quanto da aplicação do cliente. Nesse sentido, o programa do servidor aparece no console da esquerda e o programa do cliente está disposto no console da direita. Importante relembrar que as mensagens dispostas do lado do servidor são enviadas de 4 em 4 segundos, enquanto as frases dos filmes que aparecem para o cliente, são recebidas de 3 em 3 segundos.



Exemplo 1 - Output Servidor e Cliente

Exemplo 2 - Múltiplos clientes conectados simultaneamente (IPv6)

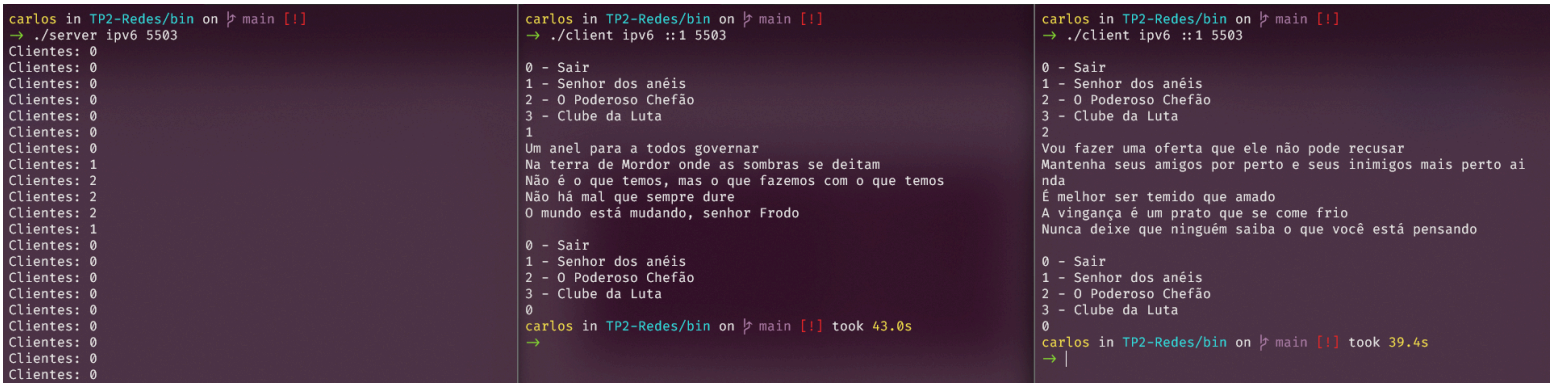
Nesse caso de uso, o servidor será executado para receber conexões IPv6 na porta 5503. Nesse âmbito, o cliente receberá o endereço do servidor (endereço local: ::1), a porta e o tipo IPv6.

Nesse exemplo, o programa do servidor é inicializado e começa a imprimir o número de clientes conectados, que, inicialmente, apresenta o valor 0. Em seguida, os programas de dois clientes são inicializados ao mesmo tempo, ambos conectados no mesmo endereço e porta do servidor em execução. O primeiro cliente solicita para o servidor a transmissão do filme ‘1 - Senhor dos Anéis’, dessa forma, o servidor inicia a transmissão e o cliente começa a receber as frases do filme, e, além disso, o servidor começa a imprimir o número de clientes igual a 1.

Ao mesmo tempo, o segundo cliente também solicita um filme, no caso, ‘2 - O Poderoso Chefão’. De forma similar, o segundo cliente também começa a receber as mensagens do filme, ao mesmo tempo que o primeiro, demonstrando a concorrência do envio de mensagens do servidor. Além disso, o servidor imprime agora que apresenta 2 clientes conectados. Quando o filme do primeiro cliente termina, o servidor identifica apenas 1 cliente conectado, que no caso, é o segundo. Quando o filme do segundo cliente também termina sua transmissão, o servidor identifica 0 clientes conectados. Por fim, ambos os clientes finalizam a execução do programa escolhendo a opção ‘0 - Sair’.

A seguir temos novamente uma imagem que demonstra o fluxo de execução dos três programas ao mesmo tempo. O programa em execução do servidor, primeiro cliente e

segundo cliente são representados na imagem pelo console da esquerda, do meio e da direita, respectivamente.



Exemplo 2 - Output Servidor e Clientes

CONCLUSÃO

Este trabalho prático demonstrou a aplicação de conceitos fundamentais na comunicação em redes de computadores, utilizando a linguagem de programação C e o protocolo UDP. Inspirado em sistemas de streaming, desenvolvemos uma aplicação console que simula a interação básica entre clientes e um servidor para a transmissão de frases de filmes. Esta abordagem proporcionou uma exploração prática de sockets, uma tecnologia essencial para a comunicação entre processos em sistemas distribuídos.

Durante o desenvolvimento, os objetivos foram alcançados com êxito. Implementamos a comunicação entre cliente e servidor, utilizando sockets para criar um canal de transmissão de dados sem a necessidade de uma conexão pré-estabelecida. O uso de UDP permitiu uma comunicação rápida e eficiente, adequada para o cenário proposto, onde a integridade e a ordem de entrega das mensagens não eram prioritárias. Além disso, a aplicação foi desenhada para suportar tanto IPv4 quanto IPv6, demonstrando flexibilidade no gerenciamento de diferentes protocolos de rede.

A estrutura do servidor, utilizando threads para processar múltiplas solicitações simultâneas, mostrou-se eficaz para lidar com a carga de trabalho, garantindo um desempenho adequado ao transmitir mensagens para diversos clientes ao mesmo tempo. O uso de mutexes para sincronização de dados críticos, como o contador de clientes conectados, assegurou a consistência em um ambiente multithreading.

Por outro lado, o cliente implementou um menu interativo simples, permitindo a escolha de diferentes filmes e solicitando suas frases ao servidor. A recepção e exibição das mensagens, utilizando o protocolo UDP, reforçaram a compreensão das diferenças entre comunicação orientada a conexão (TCP) e sem conexão (UDP).

Em termos educacionais, este projeto proporcionou um aprendizado valioso sobre a programação de redes, enfatizando os desafios e soluções na comunicação de dados em tempo real. A experiência de lidar com sockets, protocolos de rede, e gerenciamento de threads ampliou a compreensão prática das tecnologias subjacentes à comunicação cliente-servidor. As competências adquiridas serão aplicáveis em contextos mais complexos e reais de desenvolvimento de sistemas distribuídos.

Este exercício também destacou algumas limitações do UDP, como a falta de garantias de entrega e ordem dos pacotes, o que pode ser crítico em aplicações que exigem maior confiabilidade. Apesar dessas limitações, o projeto alcançou seus objetivos dentro do escopo definido, fornecendo uma base sólida para futuras explorações e aprimoramentos em comunicação de redes e programação de sistemas distribuídos.