UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIA DA COMPUTAÇÃO

Artur Luiz Rizzato Toru Soda
Davi Menegaz Junkes
Felipe Elton Pavini Savi

Trabalho 1 - Sistema Peer-to-Peer

Florianópolis
2024

1. INTRODUÇÃO

Este relatório tem como propósito demonstrar as principais decisões e estratégias tomadas durante o primeiro trabalho da disciplina de computação distribuída (INE5418), cujo objetivo é simular uma rede P2P não estruturada de compartilhamento de arquivos. Nesse sentido, cada peer possui arquivos armazenados localmente disponíveis para serem transferidos para outros peers, além disso, devem estar sempre preparados para requisitar ou receber requisições de recursos a qualquer momento.

Sendo assim, para fazer a busca deve-se utilizar o método de procura flooding, e é preciso que a transferência de recursos seja feita em chunks, sendo possível obter cada chunk de peers diferentes.

2. ESTRUTURA DO CÓDIGO

Este trabalho foi dividido nos seguintes arquivos: main.cpp, parse_files.cpp, utils.cpp, parse_files.hpp, utils.hpp, structs.hpp.

Os arquivos parse_files.hpp e parse_files.cpp possuem funções para ler os arquivos topologia.txt e config.txt.

Os arquivos utils.hpp e utils.cpp possuem funções utilitárias, como por exemplo funções para imprimir estruturas de dados na tela, serializar e desserializar os pacotes e funções auxiliares na interface com a API do Linux.

O arquivo struct.hpp possui estruturas fundamentais à lógica do programa, como as estruturas dos pacotes de requisição e resposta UDP, a estrutura File que armazena os dados de arquivo.p2p e a estrutura Data, que armazena os dados utilizados ao longo do programa.

O arquivo main.cpp, por sua vez, possui a estrutura geral do programa, chamando as funções do módulo parse_files, salvando seus retornos na estrutura Data, utiliza as funções do módulo utils para auxiliar na lógica do programa.

Logo no início do programa, são criadas cinco threads principais, responsáveis pela funcionamento geral do programa:

2.1. RECEIVE_UDP_PACKETS_THREAD

Esta thread tem como objetivo receber todos os pacotes UDPs encaminhados ao peer, e, dependendo do cabeçalho do pacote, o qual pode indicar um pacote de requisição ou de resposta.

Dessa forma, caso for de requisição, a thread respond_discovery_thread é notificada para responder-la (se não for o mesmo peer que fez a requisição) e o pedido é então posto na fila de retransmissão, caso for de resposta o pacote é armazenado no vetor de respostas recebidas.

2.2. RETRANSMIT_UDP_PACKETS_THREAD

O objetivo desta thread é transmitir os pacotes de requisições UDP recebidos pela thread receive_udp_packets_thread para seus vizinhos um segundo após ter sido adquirido.

Para isso, ela itera infinitamente pelo vetor de requisições recebidas, e para cada requisição verifica se já fez 1 segundo desde que foi obtida. Nesse sentido, ao alcançar o tempo desejado o pedido é removido da lista, e então retransmitindo para todos seus vizinhos.

2.3. REQUEST_FILE_THREAD

Esta thread faz a requisição de um arquivo digitado pelo usuário.

Nesse sentido, primeiro é aguardado que se informe o caminho do arquivo .p2p que possui as informações de qual arquivo deve ser requisitado. Em seguida, o pacote de pedido é montado e enviado para todos seus vizinhos. Com isso, a thread espera um tempo suficiente para que já tenha recebido todas as respostas enviadas pelos peers alcançados.

Depois de ter aguardado, a lista de respostas recebidas é organizada por chunk em outro vetor, e para cada chunk requisitado é escolhido a resposta com maior taxa de transmissão e iniciado a thread receive_chunk_thread para recebê-lo.

Por fim, após o término das transferências é verificado se foi recebido todos os chunks, caso verdadeiro todos os chunks são juntados para formar o arquivo requisitado.

2.4. RESPOND_DISCOVERY_THREAD

Nesta thread as requisições recebidas são respondidas.

Para esse propósito, todas vez que é recebido notificação de que uma requisição foi recebida (pela thread receive_udp_packets), é lido as informações do pedido, e então tendo o arquivo requisitado, a thread verifica os chunks inteiros (chunks que estão sendo recebidos não são enviados) que o peer que ela representa possui.

Nesse sentido, para cada chunk verificado é montado um pacote de resposta que é enviado diretamente ao peer que o requisitou, e também é iniciado uma thread responsável pela transição de envio: send_chunk_thread.

2.5. MANAGE_BYTES_TO_SEND_THREAD

Esta thread controla a taxa de transmissão do peer que ela representa.

Para esse objetivo, a cada segundo a thread carrega a variável que indica a capacidade de transmissão do peer no atual segundo.

2.6. RECEIVE_CHUNK_THREAD

Esta thread, criada pela request_file_thread, é responsável pelo recebimento de um dos chunks do arquivo requisitado.

Assim, ela age como um cliente em uma conexão TCP, onde o servidor TCP é o peer que respondeu à requisição com melhor taxa de transmissão para o chunk responsável por esta thread.

Desse modo, inicialmente se conecta ao servidor TCP, e então, entra em loop lendo os bytes recebidos e salva no respectivo arquivo de chunk. Após todos os bytes terem sido recebidos, finaliza sua execução.

2.7. SEND_CHUNK_THREAD

Nesta última thread, criada pela respond_discovery_thread, é feito o envio de um dos chunks do arquivo requisitado, tendo cuidado com a taxa de transmissão do peer que ela representa.

Para essa finalidade, ela age como um servidor TCP, onde o cliente é o peer que irá receber o chunk, ou seja, aquele que fez a requisição. Sabendo disso, a thread abre uma

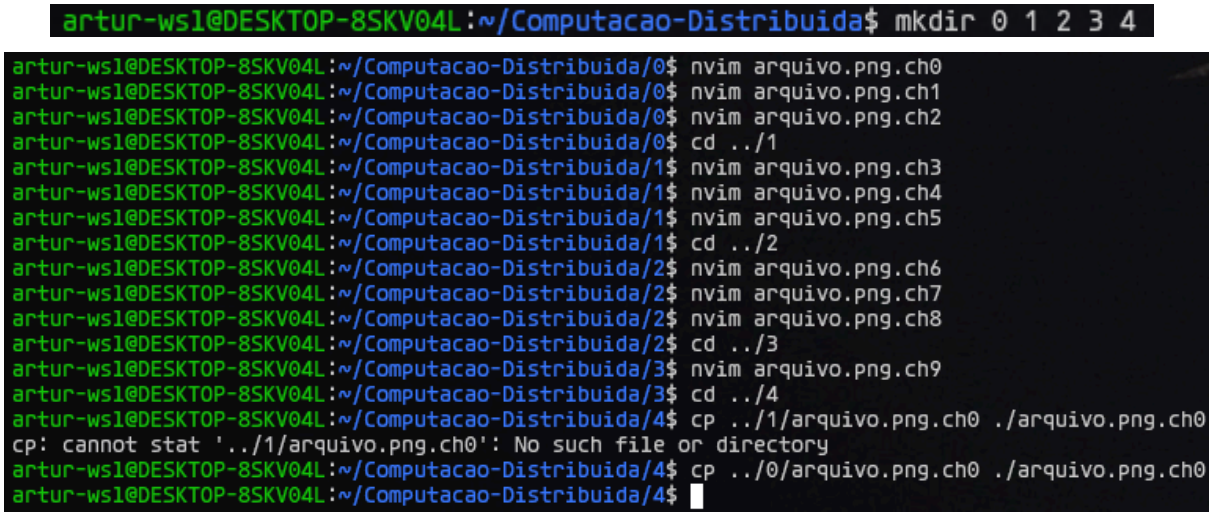
conexão TCP e aguarda que o requerente se conecte, até que um timeout seja alcançado (o timeout é o suficiente para que o request_file_thread já tenha terminado de esperar por respostas). Caso a conexão seja mal sucedida a thread simplesmente termina sua execução.

Com o sucesso de conexão, é iniciado um looping em que em todo início de iteração é aguardado por uma notificação de atualização da capacidade de transmissão do peer (feita a cada segundo pela manage_bytes_to_send_thread), e em seguida é enviado a quantidade de bytes possíveis de ser enviado no atual segundo, este loop se repete até que todos os bytes do chunk que a thread é responsável seja enviado.

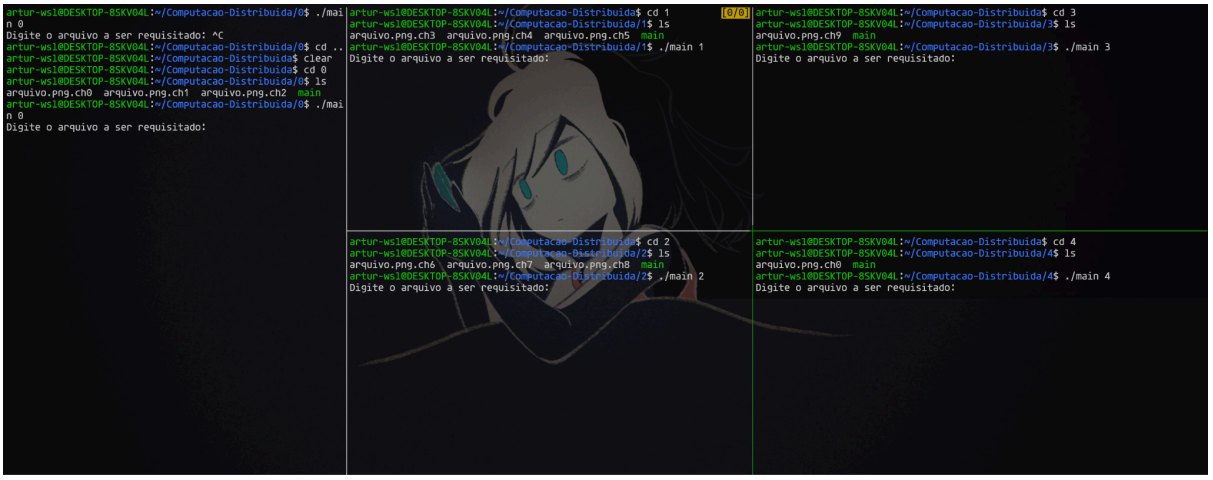
3. EXECUÇÃO DO CÓDIGO

Para compilar o código, é necessário seguir as seguintes etapas:

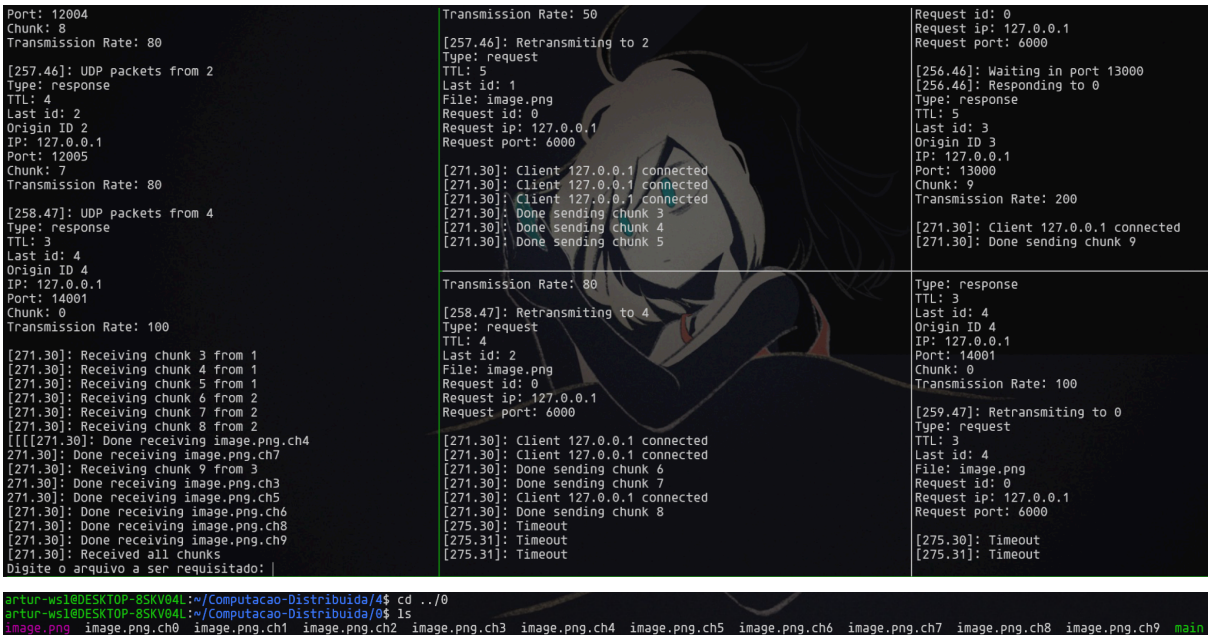
1. Criar as pastas dos nós (0, 1, 2, 3, 4) e popula-las com seus chunks



2. Executar o Makefile com o comando “make all” na pasta base. Isso criará a pasta build, com os arquivos parse_files.o, utils.o e main. O arquivo main vai ser copiado para as respectivas pastas dos nós (0, 1, 2, 3, 4)
3. Se for rodar localmente, abrir um terminal separado e rodar o comando “cd” para a pasta do respectivo nó e rodar o comando “main”, passando como argumento o nó em questão. Por exemplo, para o nó 0, rodar o comando “cd 0” seguido de “./main 0”.



4. O programa imprimirá a mensagem “Digite o arquivo a ser requisitado: “, pedindo para digitar o arquivo p2p. Quando respondido com o respectivo arquivo, ele iniciará a requisição.



4. CONCLUSÃO

Considerando os pontos discutidos, podemos afirmar que o projeto realizado foi uma ótima prática para um melhor conhecimento de diferentes áreas da computação distribuída, incluindo redes P2P, gerenciamento de threads e utilização de sockets com protocolos TCP e UDP.

Durante a implementação do sistema, foi utilizado estratégias como flooding, que provou ser eficaz na identificação de recursos, permitindo que as requisições se espalhassem de maneira ampla entre os peers.

Ademais, a escolha dos protocolos UDP para as requisições e TCP para a transferência de dados, garantiu a agilidade nas solicitações e a integridade na transferência de arquivos, respectivamente.

Em resumo, a simulação demonstra a viabilidade e a eficiência de uma rede P2P não

estruturada, destacando a importância de técnicas adequadas para busca e transferência de dados.