UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

RELATÓRIO DO PROJETO DE ORDENAÇÃO DE ARQUIVOS COM MÚLTIPLAS THREADS

Disciplina Sistemas Operacionais
Prof. André Leon S. Gradvohl, Dr

Grupo Coda Fofo
Guilherme Bastos Leone - 176199
Otávio Marques Cruz - 281443

Limeira, 12 de novembro de 2024

1 OBJETIVO

Este projeto visa ordenar números inteiros, extraídos de arquivos, em ordem crescente utilizando múltiplas threads. O usuário fornecerá os nomes dos arquivos de entrada, a quantidade de threads a ser utilizada e o nome do arquivo de saída onde os valores ordenados serão gravados. O programa será desenvolvido para o sistema operacional Linux e fará uso obrigatório da biblioteca POSIX Threads.

2 DESCRIÇÃO DO PROBLEMA

Considerando que serão fornecidos quatro arquivos desorganizados, cada um contendo pelo menos 1.000 números inteiros, o programa deverá ler os números de cada arquivo e mesclá-los em um array global. Em seguida, o programa criará threads com base na quantidade especificada pelo usuário, que poderá ser 1, 2, 4 ou 8, garantindo que não haja threads ociosas. Cada thread será responsável por ordenar uma parte do vetor. Após a ordenação, os resultados serão combinados em um único array ordenado, que será gravado no arquivo de saída.

3 INSTRUÇÕES SOBRE O PROGRAMA

3.1 DESCRIÇÃO DA SOLUÇÃO

Inicialmente, o programa valida as entradas fornecidas pelo usuário, verificando a quantidade de argumentos, o número de threads e a presença de um arquivo de saída. Após confirmar que as informações são válidas, o programa realiza alocações dinâmicas de memória para armazenar os argumentos, os dados dos arquivos e os IDs das threads. Cada alocação é verificada para evitar estouro de pilha. Para facilitar a comunicação entre as partes do código, foram definidos tipos de dados específicos, que representam os argumentos, os dados dos arquivos, as saídas das threads e as informações enviadas para elas.

Com os dados carregados, cada posição no vetor de dados dos arquivos passa a representar um arquivo distinto. Em seguida, o programa inicia o processamento desses dados. Neste procedimento, os dados são

segmentados em blocos aproximadamente iguais, que serão distribuídos para cada thread. O programa calcula quantos elementos cada thread deve processar e ajusta a quantidade de dados, assegurando que nenhuma thread fique sobrecarregada.

Posteriormente, o programa cria as threads em um laço de repetição, onde cada thread executa a função responsável pela ordenação dos números. Para facilitar a passagem dos argumentos para cada thread, é definido um tipo de dado que contém os dados a serem processados, o ID da thread e o nome do arquivo de saída.

A função de ordenação utilizada é a `qsort`, que faz parte da biblioteca padrão da linguagem C (`stdlib`). Ela recebe como argumentos o ponteiro para os números desordenados, a quantidade de números, o tamanho de um inteiro e uma função de comparação.

Após cada thread ordenar seu bloco de dados, todos os resultados são combinados em um loop de repetição. Essa junção é realizada a partir do retorno de cada thread, armazenando os números em um array global, que contém todos os números ordenados e a quantidade total deles. Finalizando assim, o array global é ordenado utilizando a função `qsort`, e os dados são então impressos no arquivo de saída.

Para fornecer informações sobre desempenho, durante a execução do programa, o tempo total de execução é calculado, assim como os tempos de execução de cada thread.

A fim de ilustrar melhor o funcionamento do programa, a imagem a seguir ilustra o fluxo principal do processamento dos números desordenados, mostrando como ele lê os arquivos até a impressão dos dados no arquivo de saída.

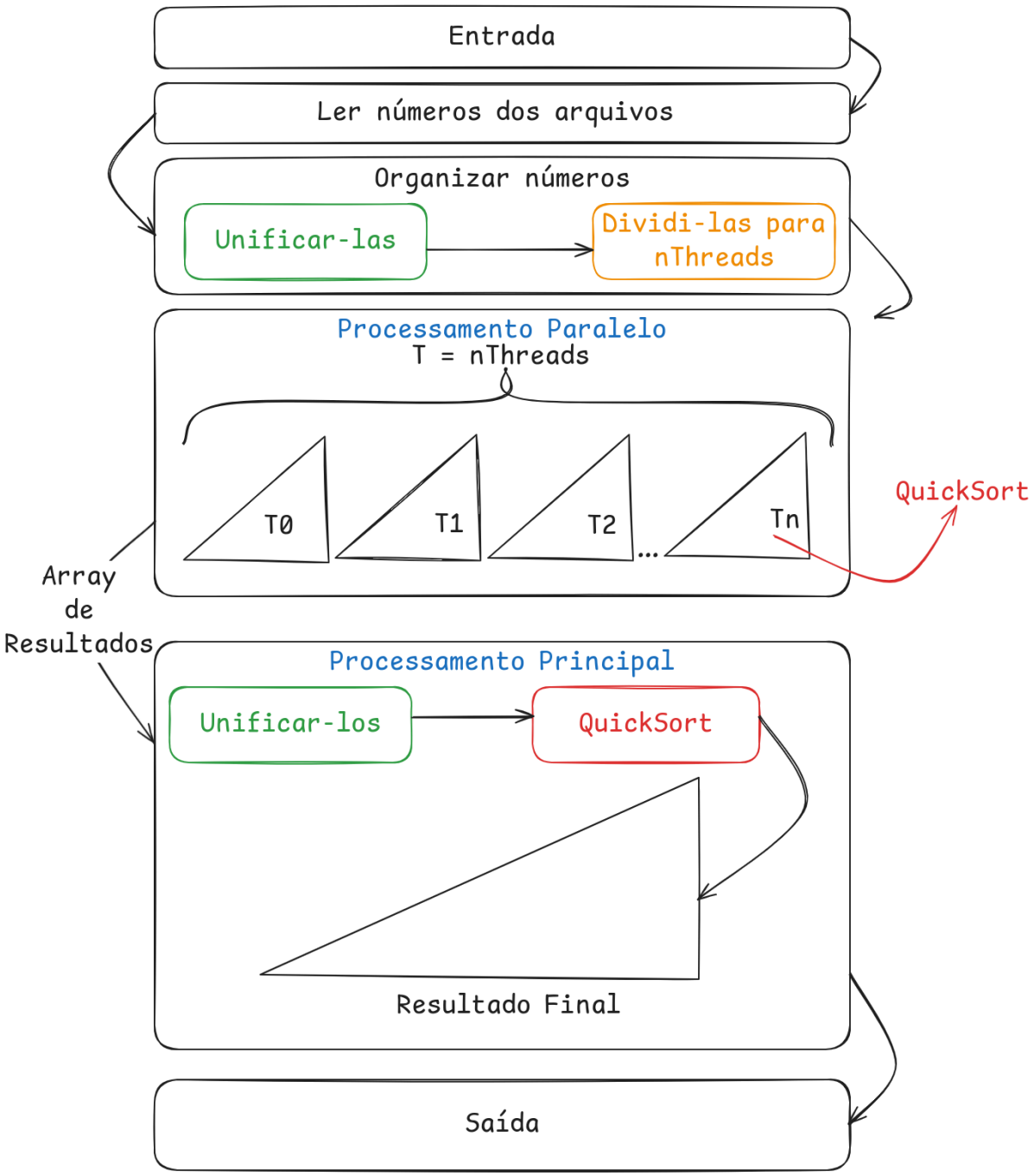

Figura 1 - Representação gráfica do programa.

3.2 COMO COMPILAR O PROGRAMA

Para compilar o programa, é necessário utilizar o sistema operacional Linux. A compilação pode ser realizada seguindo os passos abaixo:

1. Acesse o link do GitHub: https://github.com/guibleone/trabalho-final-so.
2. Clone o repositório ou faça o download dos arquivos.
3. Acesse o diretório contendo os arquivos.

4. Use o comando `make` para compilar o programa em um único executável.
5. Execute o programa com o comando: `./mergesort <n_threads> <arquivo1> <arquivo2> ... <arquivo_n> -o <arquivo_saida>`.

Passos Opcionais:

- Execute: `./auxPrograms/criador.o` para criar arquivos de teste para o organizador.
- Execute: `./auxPrograms/verificador.o` para verificar se o arquivo de saída está devidamente ordenado.

3.3 VÍDEO EXPLICATIVO

O vídeo explicando todas as partes do código, a lógica por traz do programa e os testes feitos, pode ser acessado através da plataforma *YouTube* com o seguinte link: https://youtu.be/ziDxezBxQAI.

4 RESULTADOS OBTIDOS

Os resultados obtidos foram medidos pelo programa, registrando o tempo de processamento em diferentes cenários. As medições consideraram as seguintes situações:

- **Número de threads de processamento (T):** os testes foram realizados com 1, 2, 4 e 8 threads.
- **Tamanho dos arquivos:** cada arquivo utilizado continha 1000 números inteiros.
- **Número de arquivos**: 5 arquivos para cada experimento.

Portanto, foram feitos 4 experimentos misturando essas configurações que permitiram avaliar o desempenho do programa em diferentes cenários, analisando como o número de threads influencia no desempenho.

4.1 GRÁFICOS E TABELAS

Os gráficos e tabelas a seguir apresentam os tempos individuais e totais de execução com base na quantidade de threads. Os tempos estão expressos em frações de segundo.

Quantidade Threads	Thread 0	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Tempo Total
1	0.000455	-	-	-	-	-	-	-	0.001022
2	0.000156	0.000217	-	-	-	-	-	-	0.000658
4	0.000105	0.000074	0.000072	0.000114	-	-	-	-	0.000657
8	0.000088	0.000054	0.000054	0.000044	0.000040	0.000038	0.000058	0.000036	0.001069

Tabela 1 - Tempo de execução total e individual das threads.

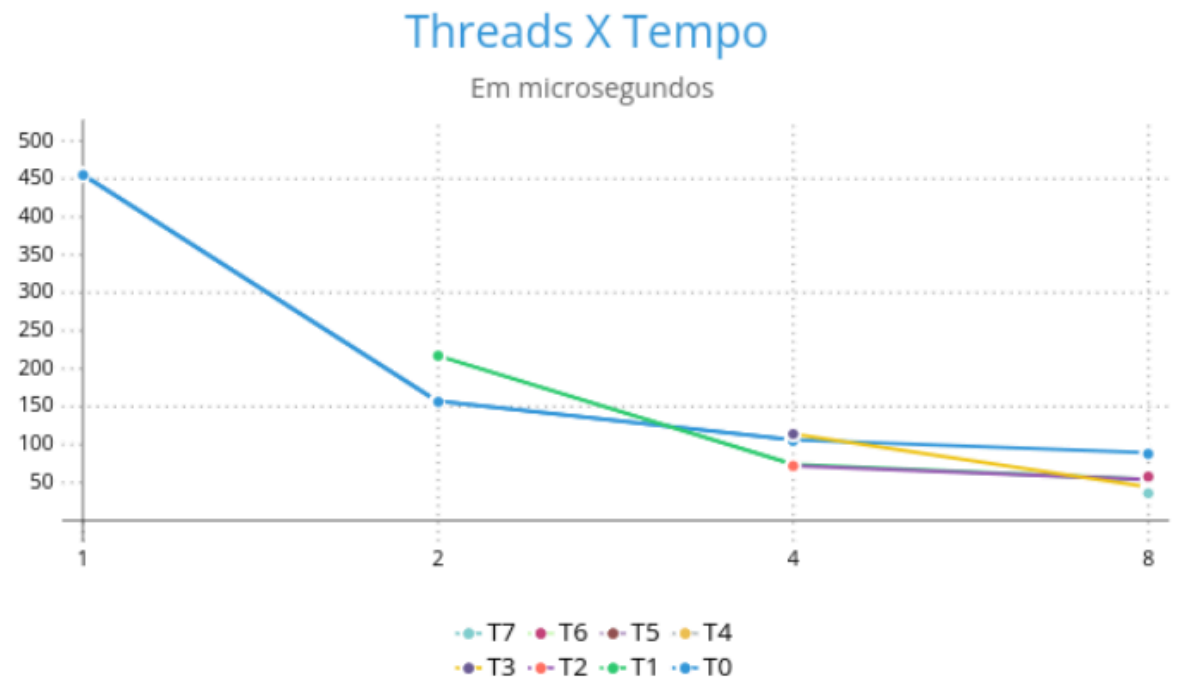

Gráfico 1 - Relação de tempo de processamento de cada thread com o número de Threads

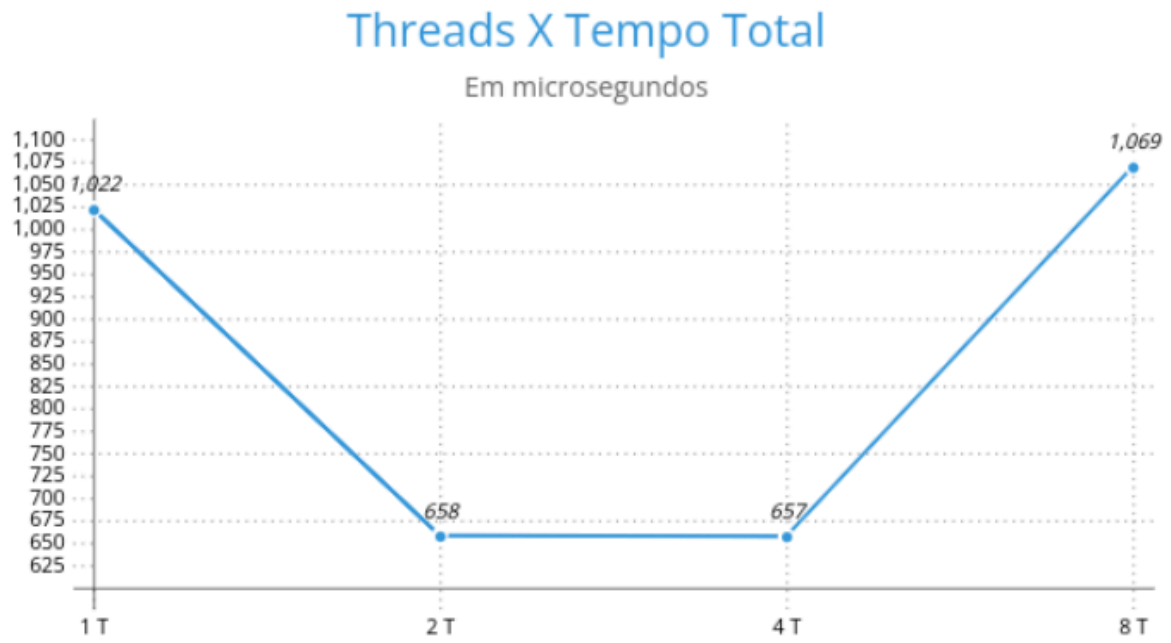

Gráfico 2 - Relação de tempo de total de processamento de cada thread com o número de Threads

5 CONCLUSÃO

Com base nos experimentos feitos, concluímos que o número de threads afeta o desempenho individual e total da execução. Com isso, afirmamos que, quanto maior a quantidade de threads, menor é o tempo individual de processamento, como visto no gráfico 1. Isto se deve ao fato de que os blocos de dados distribuídos são menores.

Entretanto, o tempo total de execução tende a crescer com aumento da quantidade de threads, como visto no gráfico 2. Isto ocorre porque o tempo de processamento necessário para criar, juntar e excluir threads requer muitas instruções da CPU. Por isso, criar mais threads deve levar em consideração quantos números serão processados. Este balanceamento, entre o número de threads e a carga de processamento, é fundamental para o funcionamento correto do programa.