Relatório do Projeto SOS

Link do repositório no GitHub: https://github.com/DanielFriol/SOS.git

Link do vídeo: https://youtu.be/HdsLh_4FCYA

Algoritmo em alto nível:

Se o número de threads passado pelo usuário não for 2, 4, 8 ou 16, o programa retorna erro e encerra sua execução

Se o número de argumentos passado pelo usuário for menor que 2 (os argumentos se iniciam em 0), o programa retorna uma mensagem de erro e encerra a sua execução

Um vetor é iniciado com uma alocação usando “calloc”

Se inicia um laço e, enquanto o contador estiver na posição que correspondem a todos os arquivos de entrada, fará as seguintes operações:

O arquivo de entrada passado por parâmetro é aberto em modo de leitura

Enquanto não chega o fim do arquivo e, se o contador for menor que o tamanho máximo permitido multiplicado pelo tamanho do vetor, o número que está no arquivo é lido, guardado no vetor, e o contador avança uma posição

Se o contador for maior, é feito uma realocação usando o comendo “realloc” para que o novo tamanho do vetor caiba.

O processo se repete para todos os arquivos de entrada

É realocado novamente, para que o vetor fique do tamanho do numero de elementos lidos nos arquivos

Uma nova variável com o tamanho do vetor é definida, para que corresponda ao novo tamanho do vetor

É criado um identificador para cada thread, de acordo com o número de threads passado pelo usuário

O vetor com os números inteiros é dividido para que cada thread processe uma parte diferente das outras do vetor

É criada os threads, chamando a função para ordenar o vetor, contudo, cada thread irá ordenar uma parte do vetor, fazendo-se assim, vários blocos ordenados dentro do vetor

É dado o comando para que cada thread espero o fim da execução das outras threads

Em seguida, ordenamos os vários blocos que cada thread ordenou, deixando assim o vetor completamente ordenado

O tempo de processamento para toda a ordenação é pego e guardado em uma variável

Após isso, o arquivo de saída é aberto em modo de escrita, escrevendo o vetor completamente ordenado nele

Após isso, mostramos o tempo levado, na tela, e uma mensagem dizendo que foi gerado um novo arquivo com sucesso.

Solução do problema:

Como solução para esse problema, utilizamos um vetor que armazena todos os inteiros lidos de todos os arquivos de entrada, após armazená-los, o mesmo vetor se divide pelo número de threads que o usuário determinou, para que cada thread ordene uma parte do vetor. Logo após essa ordenação em blocos, foram ordenados os blocos em si, utilizando o algoritmo de ordenação "Merge Sort", que colocou cada bloco ordenado dentro do vetor, e ordenou o vetor com todos os blocos já dentro dele, deixando assim o vetor completamente ordenado, pronto para ser gravado no arquivo de saída que o programa resultará. Para a ordenação nos threads utilizamos o algoritmo de "Quick Sort" (Ordenação Rápida), que pode rodar utilizando-se 2, 4, 8 ou 16 threads, ficando a critério do usuário, o número de threads a ser utilizado dentro do programa.

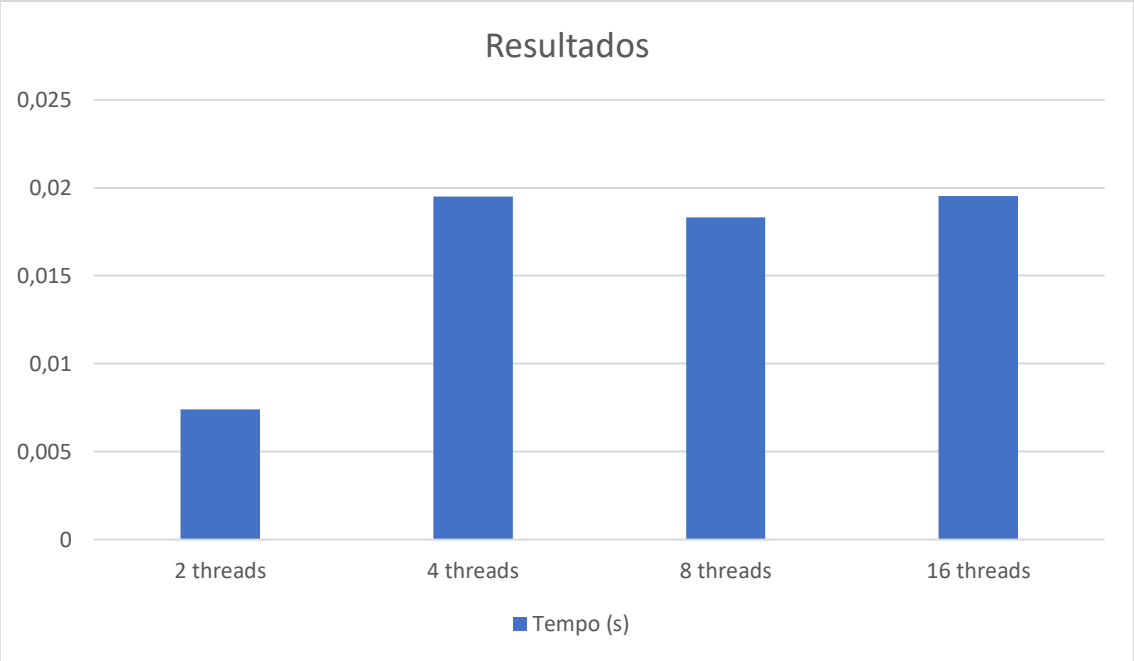
O comando para a compilação do programa em sistemas Linux é assim:

gcc SOS.c -lpthread -o "nome do programa"

E para a execução do programa, supondo que ele irá ser executado com **4** threads lendo os arquivos de entrada ***arq1.in***, ***arq2.in*** e ***arq5.in***, com isso formando um único arquivo resultante, chamado ***saida.out***, será dessa forma:

./"nome do programa" 4 arq1.in arq2.in arq5.in saida.out

O gráfico abaixo apresenta o tempo levado para processar com as opções de threads com todos os arquivos de entrada que está em nossa pasta do repositório no GitHub. As configurações da máquina utilizada para testes é um processador AMD E-300 2 cores de 1.3Ghz, Sistema Operacional Linux Mint 19 Cinnamon v 3.8.8 com 3.6 GB de memória RAM:



Com os resultados obtidos, concluímos que utilizando duas threads nesse processador o tempo é cerca de metade do tempo utilizado nas demais opções de threads. Possivelmente isso é devido ao fato de que o processador utilizado possui 2 cores, o que ajuda no processamento com duas threads somente.

Com tudo isso obtivemos a conclusão de que o ganho de velocidade para o processo utilizando threads vai depender muito da configuração do processador a ser utilizado. Pois o que ajuda a velocidade de processamento de cada thread são os números de cores de um processador.