Projeto de SO: Multicat Integer Sort

Grupo: Desesperados

Integrantes:

Alexandre Queiroz 212344

Leonardo Peterlevitz Cavichia 220150

Link para o repositório git: https://github.com/Peterlevitz/Projeto-de-SO-grupo-Desesperados

Descrição da solução do problema/algoritmo em alto nível:

Se o número de argumentos passados pelo usuário for menor do que 4, o programa mostra uma mensagem explicando o modo de uso do mesmo

Após isso, é feita uma alocação com o comando “calloc”, com o propósito de inicializar o vetor que será utilizado

Um laço é iniciado, onde, enquanto um contador estiver nas posições que correspondem aos arquivos de entrada, serão realizadas as operações seguintes:

O arquivo de entrada é aberto em modo de leitura

Enquanto não se chega ao final desse arquivo, se o contador for menor do que o tamanho máximo permitido multiplicado pelo tamanho do vetor, o número que está no arquivo de entrada é lido, e o contador avança

Se o contador for maior, o tamanho do vetor é aumentado, e é feita uma realocação de memória, a fim de comportar o novo tamanho do vetor. A memória alocada agora vai ser igual a: o tamanho máximo do vetor, multiplicado pelo tamanho do vetor, multiplicado pelo tamanho de um inteiro

Após terminada a leitura de um arquivo, o processo se repete até que acabem os arquivos de entrada

A memória é novamente realocada, para ficar com o tamanho correspondente ao número de elementos que foram lidos dos arquivos

Um novo tamanho de vetor é definido, novamente para ficar com o tamanho correspondente ao número de elementos que foram lidos dos arquivos

O próximo passo é ordenar todos esses valores, para isso, será utilizado o algoritmo de ordenação Radix Sort

Existem diversas funções que colaboram para fazer a ordenação ser possível, sendo elas:

radix_sort_thread: que coloca os valores nas posições certas

thread_work: que obtém qual porção do vetor cada thread irá ordenar, e também executa o radix sort em cada uma dessas partes

radix_sort: que é responsável por criar as threads, inicializar a estrutura utilizada pra ordenar as mesmas, e unir as threads após elas acabarem

Também foi incluído no programa um header chamado barrier.h, para auxiliar com a sincronização das threads

O programa deve ser compilado da seguinte forma:

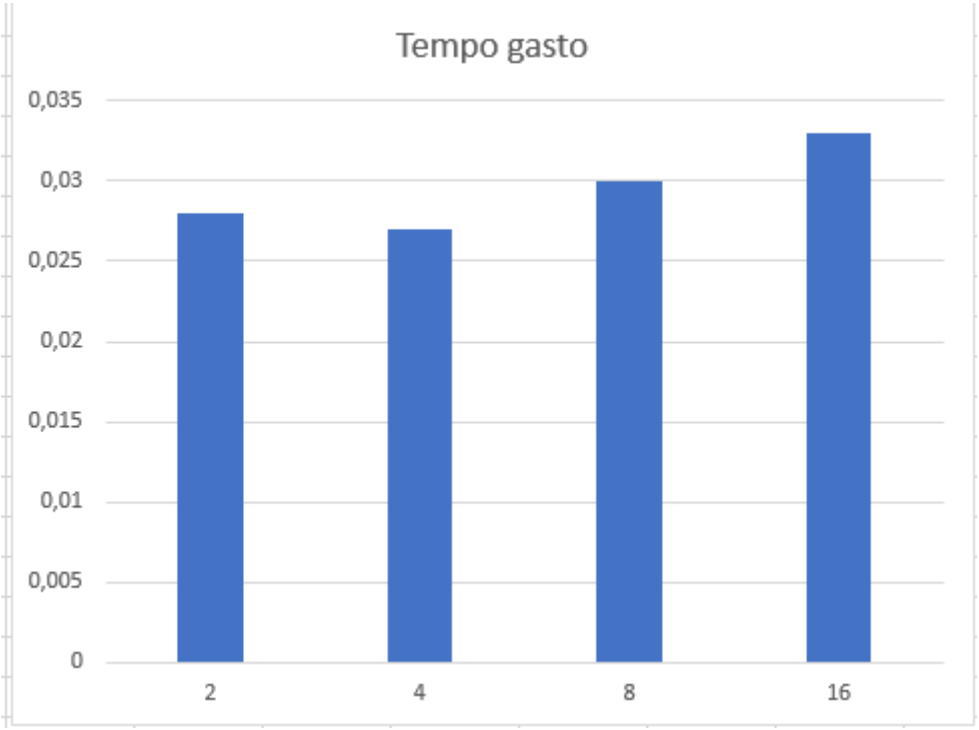
gcc -g -Wall -w multicat.c -o <nome_do_executavel> -pthread

E executado da seguinte forma:

./<nome_do_executável> <número_de_threads> (podendo ser 2,4,8,16) <arquivos_de_entrada> (no mínimo 1) <arquivo_de_saida>

Gráfico do tempo de execução:

O gráfico abaixo demonstra o tempo de execução do programa com 2,4,8 e 16 threads. O programa foi executado no sistema operacional Ubuntu, com 2 processadores Intel Core i5-2450 cpu, cada um com 2.5 GHZ, e 6 gigabytes de memória RAM. O programa foi executado 5 vezes para cada número de threads, e foi retirada uma média do tempo de execução.



Ao observar o gráfico, podemos ver que ouve uma pequena diferença de 2 para 4 threads, e uma diferença mais acentuada de 4 para 8 e 8 para 16. O programa teve o menor tempo de execução com 4 threads, muito provavelmente devido ao fato do computador em que o programa foi executado possuir 4 núcleos no total.

Então, após analisar esses resultados, podemos chegar a conclusão de que a efetividade de se dividir um programa em threads depende muito de em qual máquina o programa está sendo executado.