FUNDAÇÃO CENTRO DE ANÁLISE, PESQUISA E INOVAÇÃO TECNOLÓGICA FUCAPI

CURSO DE SISTEMAS DE INFORMAÇÃO

Título:

Ordenação externa

https://github.com/fabi0pc/OrdenacaoExternaMsort.git

Acadêmicos:

Fábio Pinheiro – Ra: 091389

Jeferson Rocha Silva – Ra: 081758

Rangel Ramos Ferreira - Ra: 134118

Robson Alexandre de Souza Olinger – Ra: 113224

Manaus, Am

03 de maio de 16

Breve descrição sobre os objetivos do trabalho:

Este trabalho tem como objetivo a implementação e execução de um programa utilizando o Merg-sort para fazer a ordenação de um arquivo de 6GB, utilizando de forma controlada a memória secundária e apresentando os possíveis resultados.

Uma breve descrição sobre sua implementação (linguagem, bibliotecas, estruturas de dados):

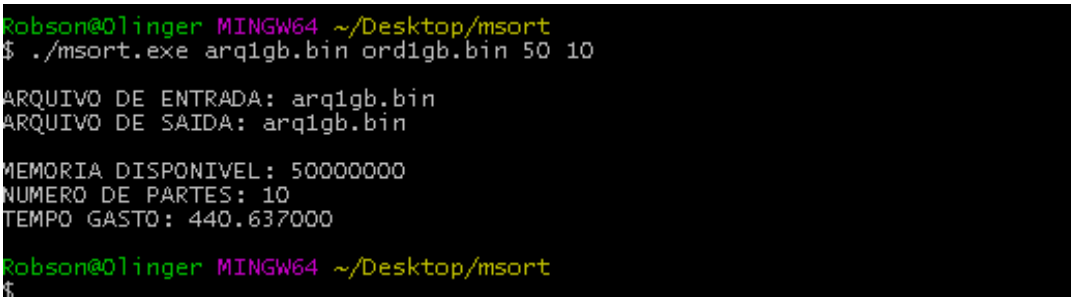
A implementação do código *Merge-sort* foi linguagem C por ela ter recursos que se mostram viáveis para a execução deste. As bibliotecas usadas neste programa são:

- **stdio.h** : Leitura de dados digitados no teclado e exibição de informações telado programa de computador.
- **stdlib.h:** Ela possui funções envolvendo alocação de memória, controle de processos, conversões e etc.
- **math.h:** Biblioteca própria para cálculos matemáticos um pouco mais complexos.
- **string.h:** Determina o comprimento duma string, Copia, Concatena n caracteres da string2 à string1.
- **unistd.h:** Define constante simbólicas diversas e tipos, declara funções auxiliares.
- **time.h:** Medir o tempo de execução duma operação, inicializar geradores de números aleatórios
- **stdint.h:** Conjuntos de tipos inteiros com larguras especificadas, e definirá conjuntos de macros correspondente.

Uma breve tutorial de compilação e execução de sua implementação, que apresentar o que se pede na Questão 1, Questão2 e Questão 4:

Para rodar o programa temos que executar no compilador o código de geração de arquivos (gerador_arquivo), e executar o aplicativo criado na pasta de destino do programa. Para criar o arquivo temos que digitar o nome do aplicativo digitar –w que induz que é pra escrever o programa e digitar o nome do arquivo a ser criado e seu tamanho.

Depois de gerado o arquivo, compilamos o segundo programa que é o de ordenação (ordenador_binario), para ordenar o arquivo criado anteriormente, digitando seu nome da aplicação o nome do arquivo que irá criar, seu tamanho de memória e quantas partes que deseja que faça o arquivo.



Ao final da execução são retornados os paramentos de entrada e saída, os números de memória disponível, números de partes e o tempo gasto.

Sucinta descrição sobre os resultados obtidos:

Tam_Arquivo	Tempo de execução	N° Partes	Plataforma	CPU/Memória
1GB	120.475	10	Windows 8.1	Core i3/4GB
3GB	412.769	10	Windows 8.1	Corei3/4GB
6GB	983.097	10	Windows 8.1	Corei3/4GB

Questão 1 - Descreva em alto nível um algoritmo de ordenação externa por intercalação *(Merge-Sort*) multivias para ordenar um arquivo de dados fornecido como entrada.

O ***Merg-Sort*** é um exemplo de algoritmo de ordenação do tipo dividir para conquistar. Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e Conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas). Como o algoritmo *Merge-sort* usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas. Os três passos úteis dos algoritmos dividir para conquistar se aplicam ao *merge sort* são:

- Dividir: Dividir os dados em subsequências pequenas;
- Conquistar: Classificar as metades recursivamente aplicando o *merge sort*; e
- Combinar: Juntar as metades em um único conjunto já classificado.

Questão 2 - Apresente a complexidade de tempo do seu algoritmo considerando o ambiente de execução em memória secundária.

Aplique a função mergesort a um vetor v[0..n-1]. O tamanho do vetor é reduzido à metade a cada passo da recursão. Na primeira rodada, a instância original do problema é reduzida a duas menores: v[0..n/2-1] e v[n/2..n-1]. Na segunda rodada, temos quatro instâncias: v[0..n/4-1], v[n/4..n/2-1], v[n/2..3n/4-1] e v[3n/4..n-1]. E assim por diante, até que, na última rodada, cada instância tem no máximo 1 elemento. O número total de rodadas é aproximadamente $\log n$. Em cada rodada, a função intercala executa $2n$ movimentações de elementos do vetor v[0..n-1]. Assim, o número total de movimentações para ordenar v[0..n-1] é aproximadamente $2n \log n$. É fácil constatar que o consumo de *tempo* da função mergesort é proporcional ao número total de movimentações, e portanto proporcional a **$n \log n$.**

Questão 3 - Baseado em seu algoritmo, implemente um programa para ordenar arquivos cujos registros são inteiros de 32 bits. O programa resultante (msort) deve receber os seguintes parâmetros:
a) arquivo de entrada - arquivo que contém os dados a serem ordenados;
b) arquivo de saída - arquivos que contém os dados ordenados;
c) memória - a quantidade total de memória disponível para a ordenação. O programa não deve alocar mais memória do que o especificado neste parâmetro;
d) K - número de vias usadas pelo Merge-sort.
`Link do repositório GitHub:`

https://github.com/fabi0pc/OrdenacaoExternaMsort.git

Questão 4 - Usando o programa implementado, execute e apresente gráficos dos seguintes experimentos de desempenho (tempo e execução) (Válida somente com a entrega da questão 3).

Tempo de Ordenação	
Tam_Arquivo	Tempo de execução em milissegundos
1GB	120.475
3GB	412.769
6GB	983.097