# Relatório Projeto Merge and sort

Nome: Dionys Dener Antonio RA: 119322

**Equipe Beta**

**Introdução**

O projeto da disciplina de Sistemas Operacionais do 2º semestre de 2019 visa a ordenação de valores obtidos através de N arquivos. A ordenação destes valores será realizada por meio de threads definidas na execução do programa.

Para a ordenação dos valores contidos nos arquivos .dat foi necessário seguir os passos abaixo:

**Algorítmo em Alto Nível**

- Checar a quantidade de linhas de cada arquivo através da função *numLinhas* e armazenar em uma variável chamada *size*.
- Criar um vetor com tamanho obtido ao contar a quantidade de linhas de cada arquivo.
- Passar o vetor criado anteriormente por parâmetro na função *lerArquivo* para realizar a leitura dos arquivos.
- É feito então a leitura de cada valor de cada arquivo e armazenado no vetor.
- Em seguida é feito a criação de threads para a ordenação do vetor.
- A thread recebe por parâmetro a função *mergeSort*, o vetor e o intervalo que será ordenado. A ideia é quanto mais threads forem criadas maior será a divisão do vetor.
- É exibido em tela o tempo de execução das threads.
- Em seguida o vetor é ordenado totalmente pois estava ordenado de acordo com a divisão realizada pela quantidade de threads.
  Imagine por exemplo que o vetor tem 5000 valores e eu ordenei utilizando 4 threads. No meu código, ele irá ordenar o vetor de 0 a 1250, de 1251 a 2500 e assim por diante. Nesta parte do código, eu ordeno o vetor gravando em *vetFinal* comparando 0 com 1251, 2501 e 3751. Se o valor de índice 0 for o menor, o próximo passo é comparar 1 com 1251,2501 e 3751.
- Após isto, os valores do vetor são gravados na mesma pasta que o arquivo compilado.

**Instruções para compilação:**

Para compilar o programa é necessário fazer o download do arquivo mergesort.c  e inserir os arquivos com valores para ser ordenados na pasta “arquivos” que deve ser criada no mesmo local que o arquivo mergesort.c se encontra. Esta pasta também se encontra no git.

Com isto, é necessário colocar os comandos no terminal para compilação:

- gcc -pthread -fno-stack-protector mergesort.c -o mergesort.o
- ./mergesort.o 4 arq1.dat arq2.dat arq3.dat arq4.dat arq5.dat

Abaixo segue link para download dos arquivos:

- Git: https://github.com/dionysantonio/projetoTT304_2019_2S

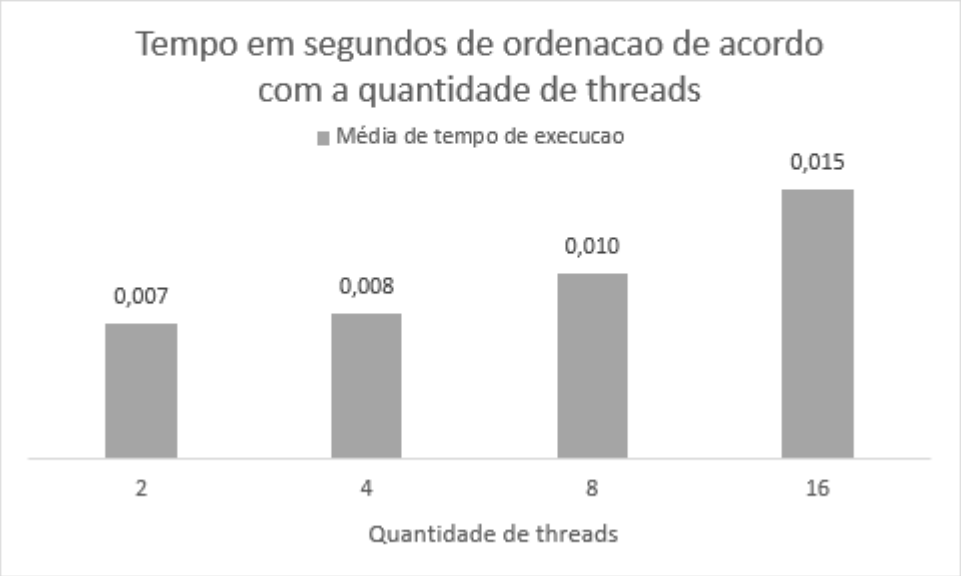
**Tempos de execução:**

Foram realizadas 200 execuções para 2, 4, 8 e 16 threads. Com isto, foram desconsiderados 14% destes dados que foram classificados como ruídos ao comparar os tempos com a média + desvio padrão ou média – desvio padrão.

| Qtd de threads | Execuções | Ruidos | % Ruidos |
|---|---|---|---|
| 2 | 50 | 1 | 2% |
| 4 | 50 | 13 | 26% |
| 8 | 50 | 1 | 2% |
| 16 | 50 | 13 | 26% |

| Qtd de threads | Media | Desvio Padrao | Media + DesvP | Media - DesvP |
|---|---|---|---|---|
| 2 | 0,027 | 0,137 | 0,164 | -0,110 |
| 4 | 0,009 | 0,004 | 0,013 | 0,005 |
| 8 | 0,030 | 0,138 | 0,168 | -0,108 |
| 16 | 0,013 | 0,004 | 0,017 | 0,009 |

| Media sem ruidos |
|---|
| 0,007 |
| 0,008 |
| 0,010 |
| 0,015 |




* Formato de tempo considerado em segundos.

**Conclusão**

Os ruídos desconsiderados da média de tempo de execução para cada cenário de quantidade de threads utilizadas na execução do programa entendemos que são casos onde houve interferência do escalonador do SO no processo. Com isto, a média pura para cada cenário de execução mostra que quanto mais threads há na ordenação do vetor, maior é o tempo de execução. Este comportamento nos mostra que para o cenário de ordenação do vetor não é indicado o uso de várias threads pois não há redução do tempo de execução. Pode-se entender também que como não há redução do tempo de execução, o tempo de alocação de memória para varias threads está fazendo com que o tempo de execução aumente conforme o cenário executado.