Projeto - “SO quero me formar logo”

 ***:*** **https://github.com/VXisto/ProjetoSO**

Video do projeto*:* **https://github.com/VXisto/ProjetoSO/blob/master/gravacao.mp4**

Resumo do algoritmo:

O algoritmo tem como objetivo a criação de um programa multithreaded (utilizando a biblioteca POSIX Threads | pthreads) que lê valores inteiros dentre 5 arquivos de entrada, onde cada um possui 1000 registros não ordenados e não necessariamente todos os arquivos são usados na execução do arquivo, sendo parametrizável a questão, para poder ordenar os registros, mostrar em tela e também é registrado o tempo levado para a execução do arquivo.

Algoritmo em alto nível:

O programa tem uma parametrização específica de threads e números de argumentos. Se as threads informadas pelo usuário não estiverem dentre 2, 4, 8 ou 16 e os argumentos passados pelo usuário forem menores que dois (sendo que eles se iniciam em zero), o programa vai retornar um erro e irá encerrar a execução.

O ínicio do programa é dado por um vetor iniciado e alocado com “calloc”, após isso é iniciado um laço e enquanto o contador estiver na posição que correspondem a todos os arquivos de entrada, irá realizar as seguintes operações:

Em modo de apenas leitura, o arquivo de entrada que foi enviado via parâmetro é aberto e enquanto não chega o fim deste e, se o contador for menor que o tamanho máximo permitido multiplicado pelo tamanho do vetor, o número que está no arquivo é lido e guardado no vetor iniciado anteriormente, e assim o contador avança uma posição. Caso ele for maior, é realocado utilizando o comando “realloc” para que o novo tamanho do vetor seja de acordo e assim o processo se repete para todos os arquivos passados através do parâmetro e assim é feita novamente a realocação, para que o vetor fique do tamanho do número de elementos lidos nos arquivos.

Após isso, é definida uma nova variável com o tamanho do vetor, para que seja correspondida ao novo tamanho gerado pela função anterior. Assim, para cada thread é criado um identificador de acordo com o número de threads passado pelo usuário pela parametrização.

Cada thread então é criada com o intuito de processar um vetor de números inteiros, divididos para melhor otimização e poder atingir partes diferentes do vetor, assim, é chamado a função para ordenar o vetor em partes, criando blocos ordenados dentro do vetor. Após isso, é chamado o comando para o final de cada thread espere o fim da execução das outras threads. Finalizando com a ordenação dos blocos feita por cada thread, tornando o vetor 100% ordenado.

Finalizando com a gravação do tempo de processamento de todo o programa em uma variável, completando com o arquivo de saída aberto em modo de escrita jogando todo o vetor ordenado nele. Assim o tempo é mostrado em tela e o programa se encerra com uma mensagem informando que um novo arquivo foi gerado com sucesso.

Solução do problema:

Como o problema levantado foi ordenar N arquivos de entrada em 4 soluções de threads, trouxemos como solução, um vetor que armazena todos os inteiros de todos os arquivos de entrada e após armazenar os valores, o vetor é dividido pelo número de threads determinadas pelo usuário na parametrização do programa, para que cada um realize sua ordenação. Logo após a ordenação dividida em blocos e estes então ordenados e utilizados o algoritmo “Merge & Sort”, que coloca em ordem cada bloco dentro do vetor, os ordenando cada um e ordenando o próprio vetor com todos os blocos, tornando completamente ordenado para ser gravado no arquivo de saída de resultado. Já na ordenação de threads, foi utilizado o mesmo algoritmo, que pode se aproveitar da execução multithreaded, rodando tanto em 2, 4, 8 ou 16 threads, ficando a critério do usuário através da parametrização do programa na execução.

Para realizar a compilação do programa, utilizaremos o makefile criado por nós para o GCC realizar a compilação e execução. Ficando assim apenas:

```
$ make all
```

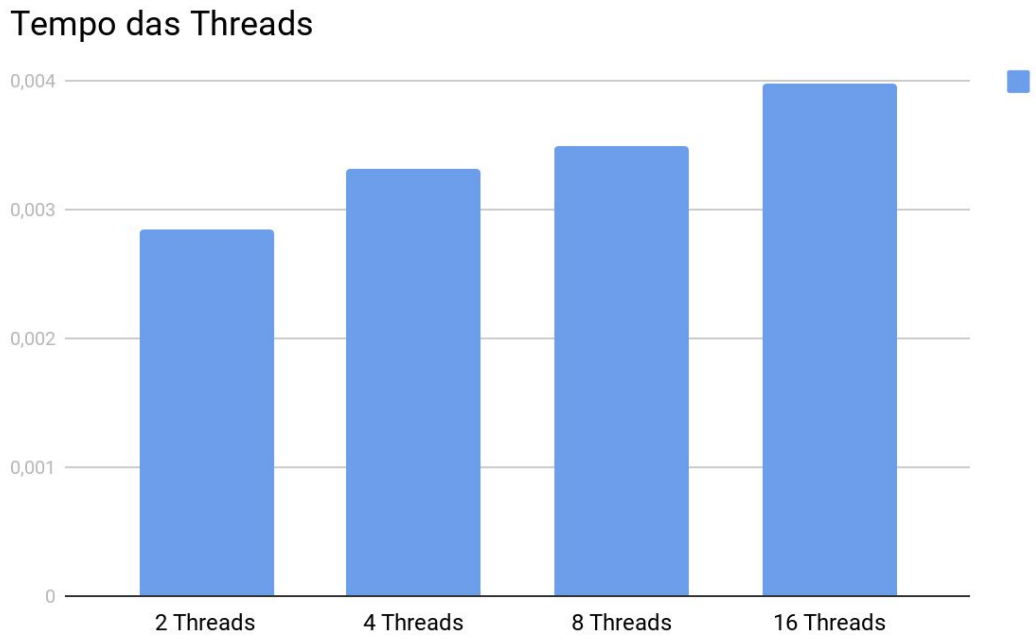
Mas caso opte por não usar o makefile, só utilizar manualmente com:

```
$ gcc projeto.c -lpthread -o “SOqueromeformar”
```

Para executar o programa, vamos supor que você gostaria de executar ele com duas threads e lendo 3 arquivos (dentre os 5 criados) para gerar o arquivo saida.dat, você irá executar desta forma:

```
$ ./SOqueromeformar 2 arq1.in arq2.in arq5.in saida.dat
```

Abaixo, em anexo, apresentamos o gráfico de tempo levado para realização do projeto com as opções de thread solicitadas no problema e os arquivos de entrada, que se encontram no repositório do projeto. As especificações da máquina onde foram realizados os testes foram: Processador Intel Core i3-4130 (2 cores Hyperthreaded / 4 threads) com 8GB de RAM e o SO é Ubuntu 18.04.3 LTS.



Valores do gráfico:

2 Threads	0.002842
4 Threads	0.003312
8 Threads	0.003486
16 Threads	0.003981

Concluímos com os testes realizados que quanto mais threads parametrizadas influencia diretamente no tempo de execução baseado no número de threads do processador. Quanto mais núcleos físicos ou threads, melhor é o processamento de tempo das multi-threads.