**Relatório para o projeto 2- “Integer MultiFind”**

**Grupo:** Mochileiros das Galáxias
**Integrantes:**
Andrey Toshiro Okamura 213119
Gustavo Adrien Polli 217357
Mateus Pim Santos 222094
**TADS** - Disciplina: T_TT304A_2018S2 - Sistemas Operacionais

**Repositório Github:** https://github.com/mateuspim/IntegerMultifind
**Playlist com vídeos:**

https://www.youtube.com/watch?v=Fixx3QXrIxw
https://drive.google.com/open?id=1R4MA-Zr0WK3n2JSDCfFjHxiTgHlTdolK

**Problema:**

Há vários (n≥1) arquivos que contém números inteiros aleatórios e não ordenados. O programa deverá utilizar múltiplos threads para, em todos os n arquivos, encontrar um determinado valor, e informar em qual linha está o número.

**Algoritmo em alto nível para solução do problema:**

Aqui será apresentado o algoritmo utilizado em “main.c”.

1. Etapa de leitura de arquivos:
   a. Separação e interpretação do array de entrada;
   b. Recebe os inteiros dos arquivos e aloca em um vetor;
   c. Para verificação posterior, a cada passo de leitura dos arquivos é inserido -1 para dividir o array;
   d. Fornece os valores início e fim de busca para as structs;
2. Etapa de construção/execução de threads:
   a. Inicialização das threads;
   b. Criação de threads de acordo com o número fornecido na entrada, para executar a função “IntegerMultifind”, que verifica a existência do valor procurado dentro do array e setando o elemento a -2.
3. Etapa de exibição dos resultados; nela é chamado a função “printNumArray”, que:
   a. Procura todos os valores de 0 ao valor top (último elemento do vetor).
   b. Caso o valor verificado for -1, foi passado para outro arquivo;
   c. Caso o valor seja -2, dado pelo “IntegerMultifind”, o número procurado foi encontrado e o printa na tela;

**d.** Caso o próximo valor do array for -1 e o número não foi encontrado, informa a inexistência do número dentro do arquivo.

**Instruções de Compilação:**

Está sendo disponibilizado duas formas de compilação, automático e manual.
Automático:

No caso automático, o comando  permite a criação de um dado número de arquivos e a execução do  programa, além da compilação.

Em um terminal ou interpretador de comando do linux com git instalado, digite:

“sh automate.sh”

Manual:

O caso manual pode ser usado se desejar realizar todo o procedimento passo a passo, ou se não utilizou o *automate.sh*. Comando:

“make”

Caso queira limpar a saída, é possível utilizar o comando:

“make clean”

Default:

gcc main.c -o main.o -c -lpthread

gcc foo.c -o foo.o -c -lpthread

gcc main.o foo.o -o multifind -lpthread

if(random_numbersNeedsToBeUsed())

gcc -c random.c -o random

Observações:

O sub-módulo *random* é um programa auxiliar para o projeto, e pode ser encontrado na pasta random no repositório Github, incluindo um guia de uso. *Se está usando o automate.sh, não é necessário acessar os módulos manualmente.*

**Instruções de Uso:**

Em um terminal ou interpretador de comando, digite:

“./multifind 16 123 arq1.in arq2.in arq3.in”

Onde:

16 é o número de threads a serem utilizados;

123 é o inteiro a ser procurado;

arq1.in, arq2.in, arq3.in são os arquivos a serem buscados.

**Gráfico com tempo de execução:**

Para este projeto foi utilizado o processador Intel Core i5-3570 @ 3.40GHz. Os dados do tempo foram obtidos através do retorno fornecido pelo próprio programa*, desconsiderando o tempo de leitura e o tempo de exibição dos resultados.

*Utilizando a diretiva <sys/time.h> -> struct timeval e inicializando com gettimeofday(&start,NULL);

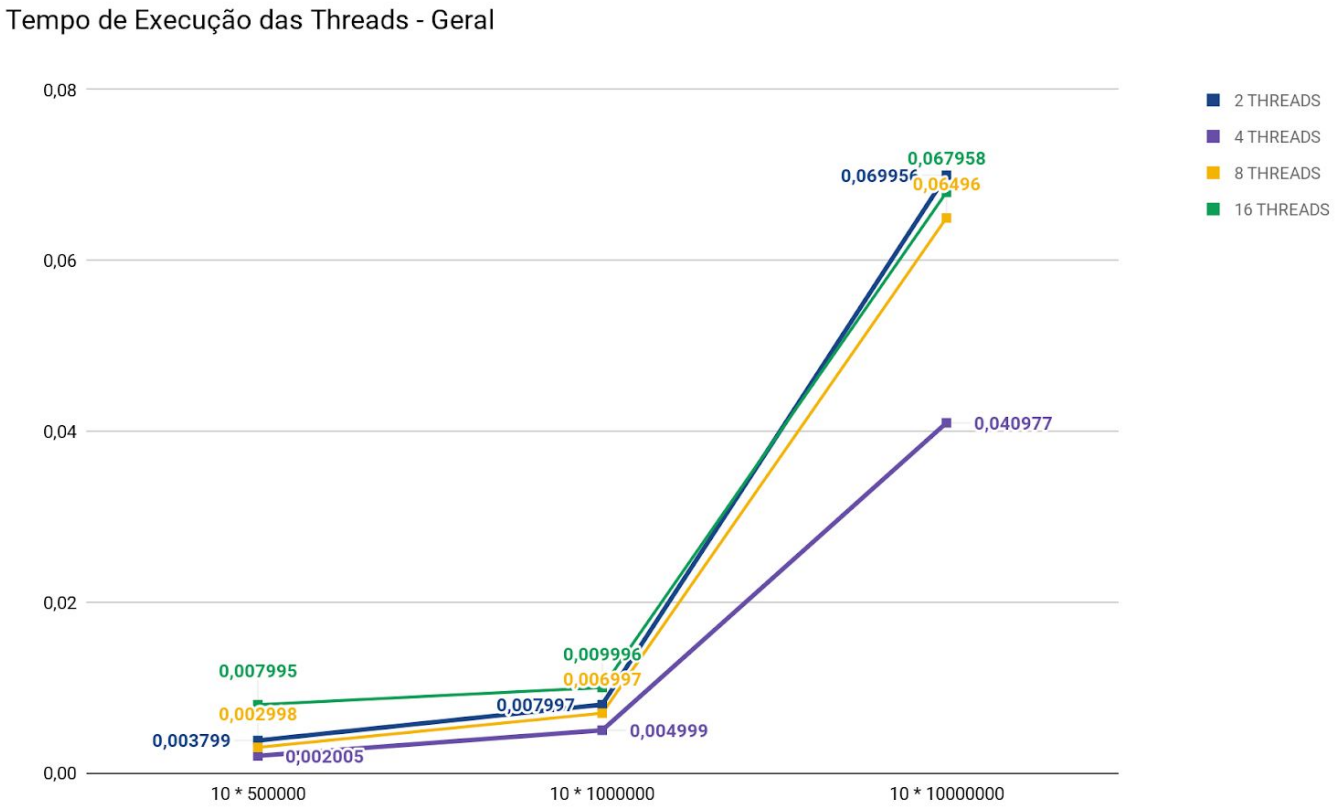


Gráfico que mostra o aumento do tempo de execução das threads

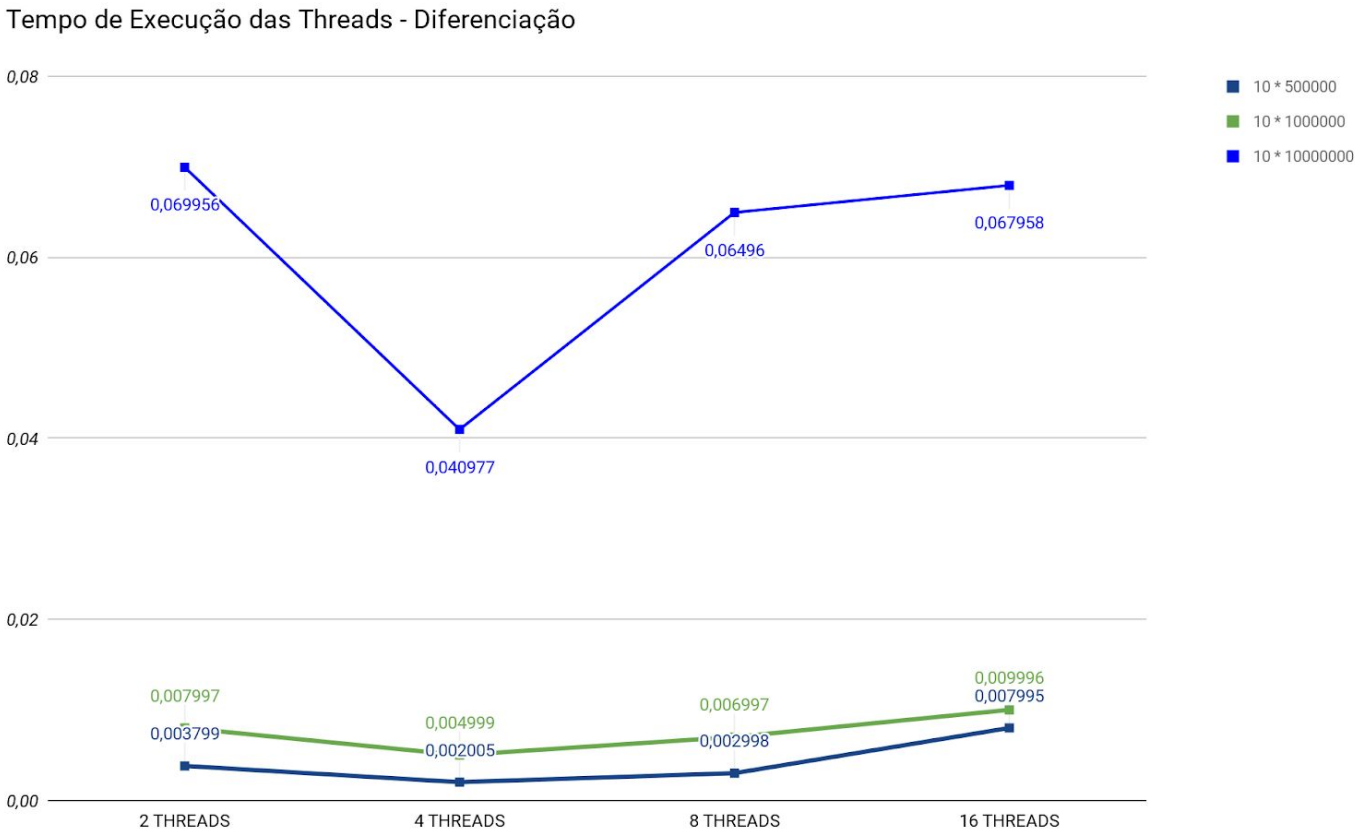


Gráfico que mostra a diferença no tempo de execução das threads

| NUM_ARQ * NUM_INT | 10*500000 | 10*1000000 | 10*10000000 |
|---|---|---|---|
| 2 Threads | 0.003779s | 0.007997s | 0.069956s |
| 4 Threads | 0.002005s | 0.004999s | 0.040977s |
| 8 Threads | 0.002998s | 0.006997s | 0.064960s |
| 16 Threads | 0.007995s | 0.009996s | 0.067958s |

Tabela com os resultados finais

* Tempo escolhido foi o menor possível entre 10 execuções utilizando os mesmos arquivos de entrada para as execuções de todas as threads.
** NUM_ARQ: Número de arquivos utilizados / NUM_INT: Número de inteiros
*** Considerar tempo no gráfico na ordem de segundos

**Conclusões:**

Os testes foram realizados em 3 casos e utilizando arquivos grandes de inteiros para realmente notar alguma diferença, assim, utilizando 10 arquivos em cada com diferentes números de inteiros:

- Primeiro teste: 10 Arquivos com 500000 números inteiros em cada;
- Segundo teste: 10 Arquivos com 1000000 números inteiros em cada;
- Terceiro teste: 10 Arquivos com 10000000 números inteiros em cada;
- Quarto teste: 10 Arquivos com 100000000 números inteiros em cada, não foi possível realizar devido ao fato de que cada arquivo de texto tinha 370MB;

Testes com quantidades muito pequenas de números inteiros dentro do array impossibilitava a amostragem dos resultados, uma vez que, sempre retornava ao tempo total 0.00000s mesmo sendo executada por 2 threads ou 16 threads, por ser um processamento muito rápido dos dados.

É possível notar que o melhor tempo possível de execução em todos os casos se dá ao utilizar o número de threads do processador para fazer a busca pelo array, neste teste realizado o processador tem somente 4 núcleos e 4 threads, o que verifica pelo baixo tempo de execução quando se utiliza 4 threads para fazer todo o trabalho.

Caso utilize 2 threads, já se percebe uma perda no tempo em relação ao uso de 4 threads, porém, ainda sim é mais vantajoso em alguns casos (uso de 16 threads), mas quando se tem 4 threads para processamento da CPU e, somente utilizar 2 delas para fazer todo o trabalho acaba por prejudicar o tempo total de execução na qual o que poderia ser feito em menos tempo e mais rápido quando todas as threads do processador estejam em execução em não ociosas.

Nos casos dos testes feitos com 8 threads o tempo é melhor do que quando se utiliza 2 threads, entretanto com 16 threads ainda há uma diferença significativa ao uso de 2 threads e, ambas, ao mesmo tempo são piores do que quando se utiliza 4 threads. Um dos motivos seria a espera das threads para poderem ter tempo de processamento a elas consecutivamente a troca de contexto das threads antigas pela nova o que acarreta em uma diferença de tempo perceptível ao ser comparada com o processamento ideal (utilizando todas as threads da CPU).