

Relatório do grupo: “Is Lazy” - Projeto 1

TT304A (Sistemas Operacionais)

João Gabriel N. D. de Oliveira, 199617
Leonardo F. Soares, 201164

Limeira, 21 de Junho de 2018

Repositório GitHub	3
Solução do problema	3
Instruções de compilação	4
Gráficos de tempo de execução	5
Conclusão	5

1. Repositório GitHub

Segue o link para o repositório do GitHub contendo todos os itens pedidos:

<https://github.com/joaognoliveira/IsLazy>

2. Solução do problema

O projeto desenvolvido pelo grupo visava a criação de um programa que utilizasse múltiplas threads para procurar um determinado valor em uma matriz $M \times N$ (com M linhas e N colunas) e posteriormente realizar a análise do desempenho desse programa com 2, 4, 8 e 16 threads.

O ponto de partida adotado pelo grupo foi o desenvolvimento das funções que levariam ao resultado final esperado. Um dos insights fundamentais no processo foi a abordagem da matriz como um novo tipo, uma struct *mDouble* que tinha como membros: a própria matriz e variáveis auxiliares que seriam futuramente usadas nas diferentes linhas de processo. Abordamos o problema proposto em etapas, e tratando do mesmo com níveis de complexidade crescentes, a primeira sendo a implementação de um programa que efetua a leitura de uma matriz qualquer, sabendo apenas sua quantidade de linhas e colunas. A próxima etapa foi a passagem e leitura dessa matriz por meio da abertura de um arquivo, a função *leMatrizDouble*, que tem como parâmetros o número de linhas, colunas e o nome de um arquivo, é responsável por essa etapa, “pegando” essa matriz de um arquivo qualquer e a alocando corretamente, efetuando verificações de arquivo vazio e espaço de memória necessários. A próxima etapa foi aproximar-se de um raciocínio que permitiria as threads serem o mais eficientes possíveis durante a procura de um elemento na matriz, utilizando material de apoio e obtendo conhecimento sobre o funcionamento de threads, acabamos por optar, via tentativas, pelo simples algoritmo em que cada thread se responsabiliza por ler uma quantidade “x” de linhas ou colunas, encontrando esse valor como resultado da divisão da quantidade de linhas pela própria quantidade de threads utilizada para execução do programa, obtendo conflitos na utilização de com “M” linhas, sendo “M” menor que dezesseis enquanto “T” threads assumem valor superior ao número de linhas, por exemplo.

O resultado foi a obtenção de um software que atingiu o objetivo proposto, efetuando de forma correta a partição da matriz em “T” linhas de processo e realizando a saída de acordo com a posição na matriz, por meio da espera e “sincronização” utilizando a função *join*, pertencente a biblioteca POSIX threads, no caso de elementos repetidos.

3. Instruções de compilação

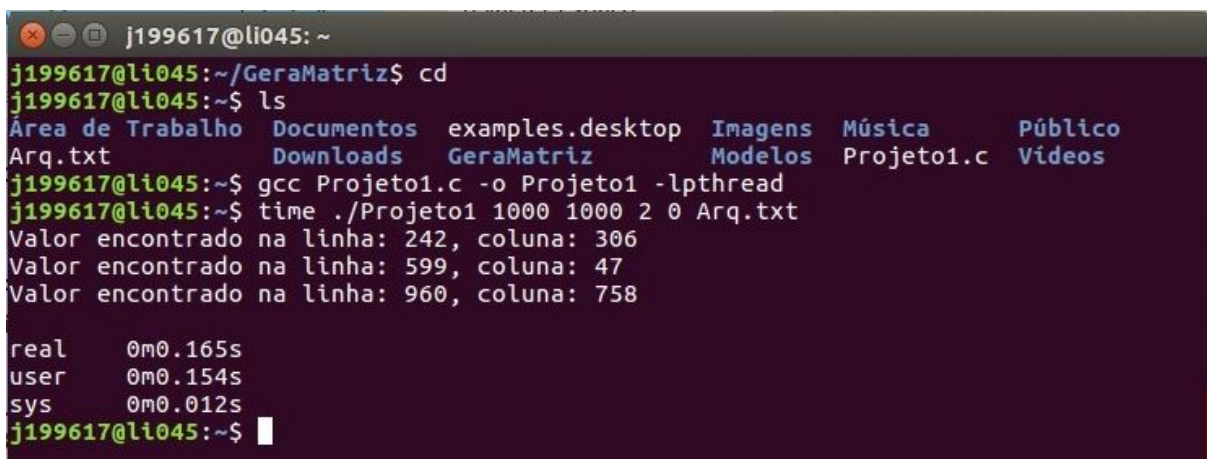
A pasta Projeto1 no repositório contém uma subpasta “GeraMatriz” e o projeto1.c (Código fonte principal do projeto). Para obtenção da matriz aleatória para testes no programa deve-se acessar a pasta GeraMatriz no terminal e executar o generateRandomMatrix, seguido do número de linhas, número de colunas, valor mínimo, valor máximo, e nome do arquivo que será gerado. conforme imagem 1.



```
j199617@li045: ~/GeraMatriz
j199617@li045:~$ cd GeraMatriz
j199617@li045:~/GeraMatriz$ ls
generateRandomMatrix      generateRandomMatrixDouble.o  MatrixIO.h
generateRandomMatrix.c    generateRandomMatrix.o        MatrixIO.o
generateRandomMatrixDouble makefile
generateRandomMatrixDouble.c MatrixIO.c
j199617@li045:~/GeraMatriz$ ./generateRandomMatrix 1000 1000 0 10000000 Arq.txt
Gerando uma matriz 1000 x 1000 com os limites [0, 10000000] para o arquivo Arq.txt
j199617@li045:~/GeraMatriz$
```

Imagem 1: gerando uma Matriz Aleatória

Então, deve-se compilar o projeto 1, por meio do compilador padrão gcc e escrevendo -lpthread ao final da linha de comando, incluindo assim biblioteca POSIX Threads no resultado da compilação. Logo em seguida, devemos executar o programa, passando os parâmetros na própria linha de comando, em seguida da entrada <M> <N> <T> <V> <filename>, conforme a imagem 2. Sendo M= número de linhas, N= número de colunas, T=número de threads e filename= nome do arquivo de onde será puxada a matriz para procura.



```
j199617@li045: ~
j199617@li045:~/GeraMatriz$ cd
j199617@li045:~$ ls
Área de Trabalho  Documentos  examples.desktop  Imagens  Música  Público
Arq.txt           Downloads  GeraMatriz        Modelos  Projeto1.c  Vídeos
j199617@li045:~$ gcc Projeto1.c -o Projeto1 -lpthread
j199617@li045:~$ time ./Projeto1 1000 1000 2 0 Arq.txt
Valor encontrado na linha: 242, coluna: 306
Valor encontrado na linha: 599, coluna: 47
Valor encontrado na linha: 960, coluna: 758

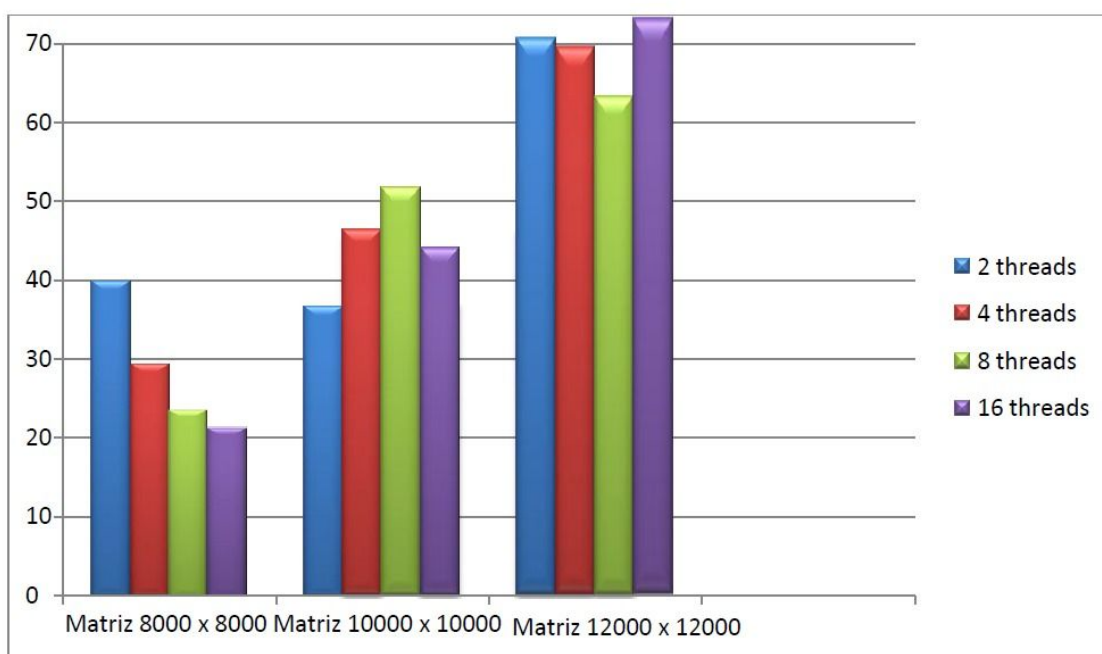
real    0m0.165s
user    0m0.154s
sys     0m0.012s
j199617@li045:~$
```

Imagem 2: compilação e exemplo de execução do programa Projeto1

4. Gráficos de tempo de execução

Devo ressaltar que os vídeos e a comparação de tempo realizados para chegada na conclusão do projeto foram feitos em uma Virtual Machine que utilizava 2048 Mb de RAM e apenas uma partição dos processadores do hospedeiro, tornando assim a execução mais lenta que o normal (comparação com computadores dos laboratórios, que possuem Dual boot, dito mais eficiente).

	8000 x 8000	10000 x 10000	12000 x 12000
2	40.038 s	37.049 s	1m11.886 s
4	30.192 s	47.614 s	1m10.120 s
8	24.222 s	52.357 s	1m4.223 s
16	22.812 s	44.573 s	1m16.479 s



5. Conclusão

A programação multithreading, se define execução de procedimentos independentes ao programa principal, por meio da divisão dessas “linhas de processo” ou threads, cada uma podendo possuir diferentes atributos e executar diferentes rotinas, logo, se aplicadas corretamente e seguindo uma lógica eficiente, podem melhorar o desempenho de um programa em muitas vezes, como por exemplo a própria proposta do projeto 1, a leitura de uma matriz com um número de linhas e colunas maiores ou iguais a 100, se torna comprovadamente mais rápida pela utilização de mais threads. Apesar de uma diferenciação desses resultados pela utilização de um Sistema Operacional em uma Virtual Machine com poder de processamento reduzido, fica evidente, em certas ocasiões (deve-se ressaltar eventos de entrada e saída entre outros), o benefício de multithreading.