UNIVERSIDADE FEDERAL DO MARANHÃO – UFMA

CIÊNCIA DA COMPUTAÇÃO/DCCET/CCET - SÃO LUÍS - TN

Relatório sobre o projeto programando com múltiplas threads

Herson Felipe Martins Sousa

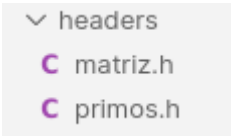
São Luís - MA
2022

DETALHES DO PROJETO

O programa tem como objetivo gerar uma matriz com números aleatórios de forma sequencial e concorrente. Para realizar a geração dos números aleatórios, foi utilizado a função rand(). O valor dos números aleatórios deve ser gerado entre 0 e 29999. Ademais, o programa deve realizar buscas sequenciais e concorrentes, contando o número de primos encontrados e o tempo de que levou para encontrar esses números.

Organização dos arquivos: os arquivos foram organizados em pastas, facilitando uma melhor visualização dos arquivos.

Arquivos cabeçalho: headers/



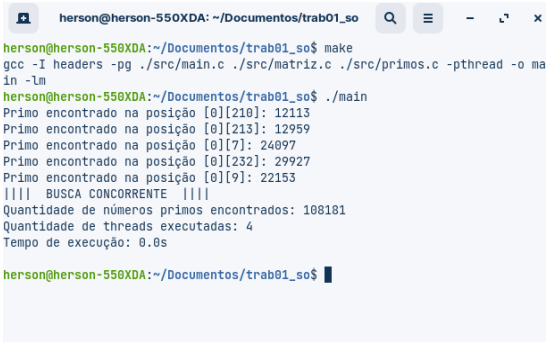
Código-Fonte em C: src/

Compilação do projeto: o compilador utilizado para compilar o projeto durante os testes foi o GCC, automatizado pela implementação de um Makefile contendo as diretivas de compilação do projeto.

Forma correta de compilar o projeto:

- make
- ./main

Saída esperada:



Sistema operacional utilizado: o projeto foi desenvolvido no sistema operacional GNU Linux, através da distro Zorin OS 16.01 Core, baseada majoritariamente no Ubuntu e com o ambiente do Gnome. A escolha de se utilizar uma distro Linux são as ferramentas disponíveis para auxiliar o desenvolvedor, além da facilidade de trabalhar com a linguagem C comparado a outros sistemas operacionais como o Windows. Além disso, o sistema operacional Linux é compatível com posix, o que facilitou bastante o processo de desenvolvimento.

REALIZAÇÃO DE TESTES

Esse projeto foi desenvolvido em uma máquina com sistema operacional GNU Linux, utilizando a distro Zorin OS 16.01, com 8gb de memória ram e um processador Intel Core i3 13ª geração, com 4 núcleos, com 4 threads.

MATRIZ 1000 X 1000

THREADS	TEMPO DE EXECUÇÃO
1	0.0s
2	0.1s
3	0.0s
4	0.0s

Analisando os dados gerados pelo gprof, podemos perceber que o algoritmo que mais realiza chamadas ao sistema é a função para checar se um número é primo (ehPrimo). Isso se justifica pelo fato de essa função checa todos os números da matriz e de sua submatrizes, ocupando quase todo o tempo de execução do programa.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
100.42      0.02     0.02   458238    43.83    43.83  ehPrimo
  0.00      0.02     0.00    53987     0.00     0.00  adicionarPrimo
  0.00      0.02     0.00       11     0.00     0.00  procuraSubmatriz
  0.00      0.02     0.00        1     0.00     0.00  criarLista
  0.00      0.02     0.00        1     0.00     0.00  criarMatriz
  0.00      0.02     0.00        1     0.00     0.00  cronExecucao
  0.00      0.02     0.00        1     0.00     0.00  divideMatriz
  0.00      0.02     0.00        1     0.00     0.00  freeMatriz
  0.00      0.02     0.00        1     0.00     0.00  iniciarBusca
  0.00      0.02     0.00        1     0.00     0.00  popularMatriz
```

MATRIZ 5000 X 5000

THREADS	TEMPO DE EXECUÇÃO
1	0.9s
2	0.8s
3	1.3s
4	1.8s

Nesse caso, analisando o relatório do gprof com uma matriz 5000 X 5000 e 4 threads foi possível notar ainda mais as chamadas de função principalmente da função ehPrimo e da função adicionarPrimo. A função ehPrimo ocupou 63.9% do tempo de execução e a função adicionarPrimo ocupa 29.39% do tempo de execução do programa.

profile-data.txt

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 63.90      0.56     0.56 11132651     0.00     0.00  ehPrimo
 27.39      0.80     0.24                             functionThread
  6.85      0.86     0.06        1    60.25    60.25  popularMatriz
  2.28      0.88     0.02  1337037     0.00     0.00  adicionarPrimo
  0.00      0.88     0.00      283     0.00     0.00  procuraSubmatriz
  0.00      0.88     0.00        1     0.00     0.00  criarLista
  0.00      0.88     0.00        1     0.00     0.00  criarMatriz
  0.00      0.88     0.00        1     0.00     0.00  cronExecucao
  0.00      0.88     0.00        1     0.00     0.00  divideMatriz
  0.00      0.88     0.00        1     0.00     0.00  freeMatriz
  0.00      0.88     0.00        1     0.00     0.00  iniciarBusca
```

MATRIZ 10000 X 10000

THREADS	TEMPO DE EXECUÇÃO
1	3.6s
2	4.4s
3	4.2s
4	4.9s

A análise feita pelo gprof em uma matriz 10000 X 10000 mostra que agora as chamadas de função estão mais distribuídas entre as funções, aumentando o nível de processamento necessário.

profile-data.txt

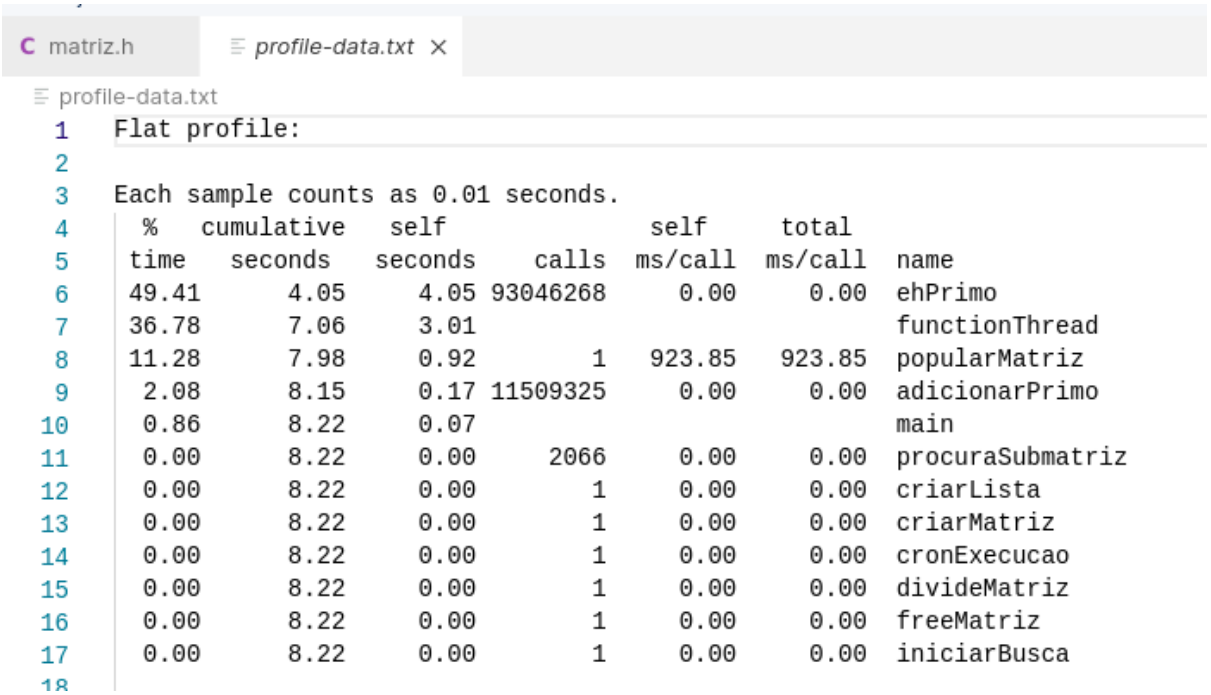
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 53.69      2.09     2.09 41090847     0.00     0.00  ehPrimo
 33.56      3.39     1.31                             functionThread
 11.10      3.83     0.43        1   431.80   431.80  popularMatriz
  1.68      3.89     0.07  5051435     0.00     0.00  adicionarPrimo
  0.26      3.90     0.01                             main
  0.13      3.91     0.01        1     5.02     5.02  freeMatriz
  0.00      3.91     0.00      965     0.00     0.00  procuraSubmatriz
  0.00      3.91     0.00        1     0.00     0.00  criarLista
  0.00      3.91     0.00        1     0.00     0.00  criarMatriz
  0.00      3.91     0.00        1     0.00     0.00  cronExecucao
  0.00      3.91     0.00        1     0.00     0.00  divideMatriz
  0.00      3.91     0.00        1     0.00     0.00  iniciarBusca
```

MATRIZ 15000 X 15000

THREADS	TEMPO DE EXECUÇÃO
1	8.8s
2	9.2s
3	9.3s
4	9.6s

A análise feita pelo gprof em uma matriz 15000 X 15000 mostra que usou ainda mais recursos do sistema operacional, pois executou



matriz.h | profile-data.txt

profile-data.txt

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 49.41      4.05     4.05 93046268     0.00     0.00  ehPrimo
 36.78      7.06     3.01                             functionThread
 11.28      7.98     0.92        1   923.85   923.85  popularMatriz
  2.08      8.15     0.17 11509325     0.00     0.00  adicionarPrimo
  0.86      8.22     0.07                             main
  0.00      8.22     0.00     2066     0.00     0.00  procuraSubmatriz
  0.00      8.22     0.00        1     0.00     0.00  criarLista
  0.00      8.22     0.00        1     0.00     0.00  criarMatriz
  0.00      8.22     0.00        1     0.00     0.00  cronExecucao
  0.00      8.22     0.00        1     0.00     0.00  divideMatriz
  0.00      8.22     0.00        1     0.00     0.00  freeMatriz
  0.00      8.22     0.00        1     0.00     0.00  iniciarBusca
```

MATRIZ 20000 X 20000

THREADS	TEMPO DE EXECUÇÃO
1	8.2s
2	8.8s
3	9.3s
4	16.0s

Nesse teste com 4 threads em uma matriz 20000 X 20000 tem um custo muito maior que os outros, pois as funções ehPrimo, adicionarPrimo e procurarSubmatriz são as mais executadas.

relatorio_gprof.txt

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 68.20     10.71    10.71 300547292     0.00     0.00  ehPrimo
 19.11     13.72     3.00                             functionThread
 11.06     15.45     1.74        1     1.74     1.74  popularMatriz
  1.02     15.62     0.16 34391049     0.00     0.00  adicionarPrimo
  0.96     15.77     0.15                             main
  0.06     15.78     0.01        1     0.01     0.01  freeMatriz
  0.00     15.78     0.00     7041     0.00     0.00  procuraSubmatriz
  0.00     15.78     0.00        1     0.00     0.00  criarLista
  0.00     15.78     0.00        1     0.00     0.00  criarMatriz
  0.00     15.78     0.00        1     0.00     0.00  cronExecucao
  0.00     15.78     0.00        1     0.00     0.00  divideMatriz
  0.00     15.78     0.00        1     0.00     0.00  iniciarBusca
```

DIFICULDADES ENFRENTADAS

Durante a elaboração desse projeto, algumas dificuldades surgiram, como a implementação de pthreads. Nunca havia desenvolvido nenhum projeto em que tivesse que lidar diretamente com a criação e gerenciamento das threads. Outro ponto a salientar é o uso da linguagem C, onde eu particularmente tenho algumas dificuldades.

Pude aplicar algumas técnicas de estruturas de dados que deixaram meus algoritmos com custos menores e que por consequência deixaram a execução do programa mais rápida.

Como se trata de threads, tudo pode acontecer. O programa pode funcionar na primeira execução ou acontecer alguns problemas, principalmente a alocação de memória.

CONCLUSÃO

A elaboração desse projeto foi de suma importância para me aperfeiçoar em linguagem C e aprender a utilizar a biblioteca pthreads em C. Além desse aperfeiçoamento e aprendizado, consegui aprender um pouco mais sobre o sistema operacional que utilizo atualmente - Linux. Pude entender melhor como os programas consomem a memória e como é importante uma boa implementação dos algoritmos para que o programa se comporte bem também em máquinas com menor capacidade de processamento e memória.

Muitas vezes o meu sistema operacional ficou mais lento, principalmente nos testes com matrizes 20000 X 20000, onde foi consumido aproximadamente 5gb de memória RAM. Entender como são implementadas as threads com a linguagem C me fez tomar mais cuidado com o custo dos meus algoritmos e pensar ainda mais no desempenho deles.