

Sistemas Operacionais

Trabalho 1 - Implementação de multiplicação de matrizes

Profª: Valeria Menezes Bastos

Grupo : Felipe Magalhães Duarte - 109119027

Lucas Roque Bernardo de Miranda - 112015703

Objetivo

Implementar a multiplicação de matrizes em duas versões, uma utilizando subprocessos e a outra com threads na linguagem de programação C. As matrizes envolvidas (**A**, **B** e **C** - **AxB=C**) são quadradas e com tamanho **N** = {3, 10, 50, 100, 200}.

Na versão usando subprocessos é necessário que se utilize alguma técnica de comunicação entre eles, pois normalmente um processo filho não passa suas informações para o processo pai. Nós utilizamos o método de pipe, em que os processos filhos enviam as linhas das matrizes por um lado do pipe e o processo pai recebe essas linhas para obter o resultado.

Instruções de compilação/execução

Threads

Para compilar o código que utiliza threads para o cálculo da multiplicação das matrizes, abra o terminal e vá até a pasta que contenha o arquivo *thread.c*.

Execute o seguinte comando:

```
gcc matrix.c -o matrix -lpthread
```

Para executar o código compilado, digite:

```
./matrix [tamanho-da-matriz] -p [numero-de-threads]
```

Obs: Em nossos testes o número de threads sempre foi igual ao tamanho da matriz.

Subprocessos

Para compilar o código que utiliza subprocessos para o cálculo da multiplicação das matrizes, abra o terminal e vá até a pasta que contenha o arquivo *subprocess.c*.

Execute o seguinte comando:

```
gcc subprocess.c -o subprocess
```

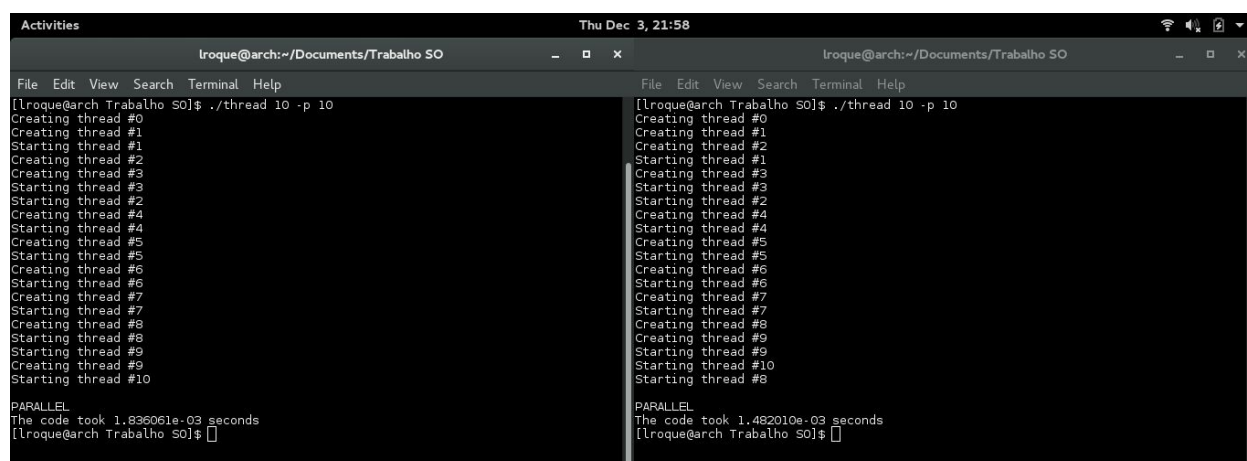
Para executar o código compilado, digite:

```
./subprocess [tamanho-da-matriz]
```

Resultados

Em nossos testes o número de threads/subprocessos foram iguais aos valores de **N** (tamanho da matriz quadrada). A estratégia usada nos nossos códigos foi de calcular uma linha da matriz resposta **C** em cada thread/subprocesso. Assim, há independência dos resultados, já que a escrita dos mesmos será sempre em regiões distintas.

Para os valores da matriz, deixamos um valor fixo para **A** e **B**: **A** foi preenchida somente com valor 1 e **B** com valor 2. Tal decisão foi feita para diminuir os fatores que pudessem influenciar durante as diversas execuções, porém os códigos estão aptos a resolver o problema para qualquer valor de **A** e **B**. A ordem de execução das threads varia a cada vez que rodamos o programa, porém nas execuções com subprocessos eles sempre mantiveram a mesma ordem. Para não ficar extenso, registramos essa variação de ordem no exemplo de 10 threads, como mostra a figura abaixo.



```
Activities Thu Dec 3, 21:58
lroque@arch:~/Documents/Trabalho SO
File Edit View Search Terminal Help
[lroque@arch Trabalho SO]$ ./thread 10 -p 10
Creating thread #0
Creating thread #1
Starting thread #1
Creating thread #2
Creating thread #3
Starting thread #3
Starting thread #2
Creating thread #4
Starting thread #4
Creating thread #5
Starting thread #5
Creating thread #6
Starting thread #6
Creating thread #7
Starting thread #7
Creating thread #8
Starting thread #8
Creating thread #9
Starting thread #9
Starting thread #10
PARALLEL
The code took 1.836061e-03 seconds
[lroque@arch Trabalho SO]$

lroque@arch:~/Documents/Trabalho SO
File Edit View Search Terminal Help
[lroque@arch Trabalho SO]$ ./thread 10 -p 10
Creating thread #0
Creating thread #1
Creating thread #2
Starting thread #1
Creating thread #3
Starting thread #3
Starting thread #2
Creating thread #4
Starting thread #4
Creating thread #5
Starting thread #5
Creating thread #6
Starting thread #6
Creating thread #7
Starting thread #7
Creating thread #8
Starting thread #8
Creating thread #9
Starting thread #9
Starting thread #10
Starting thread #8
PARALLEL
The code took 1.482010e-03 seconds
[lroque@arch Trabalho SO]$
```

Tempos de execução:

A seguinte máquina foi a usada para compilar e executar os códigos:

SO: Arch Linux 64-bit

Memória: 4GB RAM

Processador: Intel® Core™ i5-2410M CPU @ 2.30GHz × 4

Subprocessos:	Threads:
- 3x3 ~ 1,529 ms	- 3x3 ~ 0,607 ms
- 10x10 ~ 4,914 ms	- 10x10 ~ 2,742 ms
- 50x50 ~ 27,57 ms	- 50x50 ~ 5,369 ms
- 100x100 ~ 73,99 ms	- 100x100 ~ 12,66 ms
- 200x200 ~ 230 ms	- 200x200 ~ 43,51 ms
- 500x500 ~ 1252 ms	- 500x500 ~ 501,6 ms

Conclusão

Para valores pequenos de N , apesar de a execução com threads ser mais rápida, a estratégia de subprocessos pode ser viável. Porém, se considerarmos N muito grande, o cenário muda. Esse resultado é esperado, visto que os subprocessos são cópias independentes do processo principal (contexto de software, contexto de hardware e espaço de endereçamento próprios) e, por isso, a criação deles e as trocas de contexto são mais demoradas. Para exemplificação, a criação de um subprocesso obedece as mesmas etapas de um processo comum:

- Atribuir um PID
- Alocar uma entrada na tabela de processos
- Alocar espaço para o processo
- Inicializar o PCB
- Colocar o processo na fila apropriada
- Criar estruturas auxiliares

Além disso, os subprocessos não se comunicam nativamente. É preciso implementar alguma estratégia para que essa comunicação possa ser feita e, no nosso caso, foi a de pipe.

A implementação com threads é muito mais direta, pois elas compartilham memória e outros recursos do processo, o que torna menos custosa a sua criação e troca de contexto. Em termos de programação, a estratégia com threads foi mais simples e mais intuitiva.

Unindo essas vantagens das threads e as dificuldades e desvantagens percebidas com o uso de subprocessos, fica fácil perceber que se fosse preciso escolher entre as duas maneiras para resolver problemas mais complicados ou que necessitem menor tempo de resposta a escolha iria para threads.