

SISTEMAS OPERACIONAIS  
PROF. CALEBE CONCEIÇÃO

## Aula 6 - Exercícios sobre Processos e Threads

*Atividade avaliativa, conforme plano de ensino apresentado. Atente-se aos prazos descritos no AVA.*

Aluna(o): Guilherme Menezes de Azevedo

Turma: T02

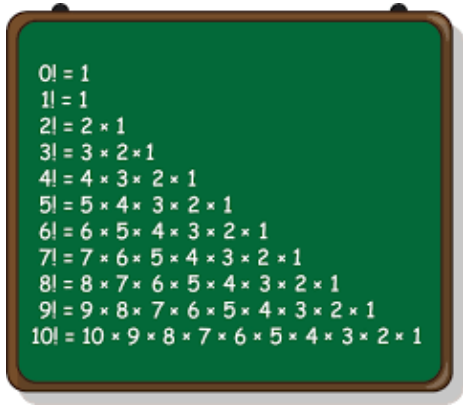
Nesta atividade você deve implementar em ambiente Linux diferentes soluções para o cálculo fatorial de um número. Você deve implementar e executar 4 soluções. Uma solução direta com laços de repetição (fornecida), outra usando processos que se comunicam por meio de memória compartilhada, uma terceira que dispara o uso de 2 usando threads, e uma quarta que dispara 10 threads. A seguir explico mais detalhes sobre como as unidades de processamento devem ser estruturadas.

Ao final, você deve comparar o tempo de execução de cada solução e reportar. Na entrega final devem constar a listagem do código fonte e suas impressões a respeito do tempo de execução de cada solução, respondendo às seguintes questões: os tempos de processamento convergem com o previsto em teoria? Se não convergem, o que pode ter acontecido? Quais fatores influenciam no tempo de processamento de cada solução?

### INSTRUÇÕES GERAIS

O cálculo do fatorial de um número  $N$  consiste na multiplicação e acumulação dos números  $N$  por todos os seus antecessores até 1, conforme ilustrado ao lado. Nesta atividade, sua tarefa é colocar duas unidades de execução (processos ou threads) a cooperar para solução do problema para alguns valores de  $N$  pré-definidos nesta atividade. Os programas devem ser implementados em C/C++, usando a biblioteca PThreads quando necessário. Para todas as variações de implementação, use o comando **hyperfine** para medir o tempo de execução, conforme segue:

`hyperfine --warmup 3 -m 10 ./seu_codigo_binario_aqui`



```
0! = 1
1! = 1
2! = 2 × 1
3! = 3 × 2 × 1
4! = 4 × 3 × 2 × 1
5! = 5 × 4 × 3 × 2 × 1
6! = 6 × 5 × 4 × 3 × 2 × 1
7! = 7 × 6 × 5 × 4 × 3 × 2 × 1
8! = 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1
9! = 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1
10! = 10 × 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1
```

**Nota 1)** o `./` na frente do seu binário é importante para o comando funcionar.

**Nota 2)** mande mensagem no grupo da turma se precisar de ajuda com os comandos necessários para compilação

Este comando realiza a execução de uma tarefa 10 vezes, coleta e apresenta o resultado do processamento, contendo mediana, desvio padrão, e os valores máximo e mínimo. Anote esses valores na tabela do verso da folha.

A entrega dessa atividade consiste no preenchimento da Tabela 1 - Resultados Obtidos no verso desta folha com as saídas do comando **hyperfine** para cada cenário, juntamente com suas considerações sobre os experimentos realizados e o envio desta folha de atividades preenchida juntamente com os códigos-fonte das aplicações indicadas.

*Obs.: Ignore o resultado da saída de execuções individuais para  $N$  muito grandes. Estamos mais interessados em fazer o computador trabalhar, e por isso considere só a execução do **hyperfine**. ;-)*

#Bons estudos.

**Tabela 1 - Resultados obtidos no WSL2**

Valor de N	Nome do Programa	Mediana	Des. padrão ( $\sigma$ )	(user)	(system)	max	min
1.000	fatorial_simples	2.0 ms	0.6 ms	0.4 ms	0.0 ms	4.6 ms	1.4 ms
	fatorial_dois_processos_memoria_comp	4.2 ms	0.3 ms	1.4 ms	0.1 ms	5.3 ms	3.7 ms
	fatorial_2_threads	2.3 ms	0.3 ms	0.5 ms	0.1 ms	3.7 ms	1.6 ms
	fatorial_10_threads	3.1 ms	0.3 ms	1.2 ms	0.2 ms	4.5 ms	2.5 ms
1.000.000	fatorial_simples	5.6 ms	0.3 ms	2.7 ms	0.1 ms	7.2 ms	4.7 ms
	fatorial_dois_processos_memoria_comp	5.9 ms	0.4 ms	3.7 ms	0.2 ms	8 ms	5 ms
	fatorial_2_threads	6.5 ms	1.3 ms	3.0 ms	0.2 ms	8.6 ms	3.3 ms
	fatorial_10_threads	6.8 ms	0.8 ms	5.0 ms	0.4 ms	9.8 ms	4.5 ms
1.000.000. 000	fatorial_simples	2.217 s	0.007 s	2.214 s	0.000 s	2.228 s	2.207 s
	fatorial_dois_processos_memoria_comp	1.120 s	0.006 s	2.224 s	0.002 s	1.128 s	1.113 s
	fatorial_2_threads	2.320 s	1.237 s	4.590 s	0.001 s	3.677 s	0.930 s
	fatorial_10_threads	2.899 s	0.716 s	16.550 s	0.011 s	3.844 s	1.710 s

**Tabela 2 - Resultados obtidos usando o Kali Linux**

Valor de N	Nome do Programa	Mediana	Des. padrão ( $\sigma$ )	(user)	(system)	max	min
1.000	fatorial_simples	942.2 $\mu$ s	77 $\mu$ s	823.7 $\mu$ s	235.8 $\mu$ s	1.4 ms	735.1 $\mu$ s
	fatorial_dois_processos_memoria_comp	1.5 ms	0.1 ms	1.5 ms	0.4 ms	2.2 ms	1.3 ms
	fatorial_2_threads	1.3 ms	0.1 ms	0.8 ms	0.4 ms	1.8 ms	1 ms
	fatorial_10_threads	1.7 ms	0.1 ms	1.4 ms	0.7 ms	3.1 ms	1.4 ms
1.000.000	fatorial_simples	5.2 ms	0.1 ms	4.8 ms	0.5 ms	5.7 ms	4.9 ms
	fatorial_dois_processos_memoria_comp	1.5 ms	0.1 ms	1.5 ms	0.4 ms	2.2 ms	1.2 ms
	fatorial_2_threads	6.5 ms	1.9 ms	8.3 ms	0.8 ms	8.7 ms	3 ms
	fatorial_10_threads	5.8 ms	1.1 ms	9.5 ms	1.1 ms	10.1 ms	3.6 ms
1.000.000. 000	fatorial_simples	4.275 s	0.003 s	4.273 s	0.001 s	4.28 s	4.27 s
	fatorial_dois_processos_memoria_comp	1.5 ms	0.1 ms	1.5 ms	0.4 ms	2.1 ms	1.2 ms
	fatorial_2_threads	5.097 s	2.291 s	9.989 s	0.001 s	7.246 s	1.83 s
	fatorial_10_threads	3.06 s	0.259 s	11.262 s	0.008 s	3.525 s	2.76 s

### Suas considerações

Apresente no quadro abaixo suas considerações sobre os experimentos realizados. Atente-se às questões apresentadas no enunciado da atividade.

Toda essa atividade foi realizada no OS Windows 10 Pro com WSL2 Ubuntu, dentro do WSL2 foi feito todo o benchmark pedido na atividade. Além disso, todos os processos foram finalizados e apenas foi usado o terminal do PowerShell para acessar a CLI do WSL2 Ubuntu. OBS.: Foi desativado todos os processos em segundo plano dos aplicativos, exceto os processos Zombie (Que deu tela Azul no Windows quando fui finalizar kkkkk) e minha máquina tem as seguintes configurações: Intel I5-8400K com 16 GB de RAM 2400MHZ em Dual-Channel.

### 1. Aplicação 1.:

- No fatorial\_simples com  $N = 1.000$  é rápido com baixa variação de tempo de execução, com  $N = 1.000.000$  o tempo aumenta quase que linearmente, mas mesmo assim mantém a consistência, com  $N = 1.000.000.000$ , o tempo de cálculo é muito maior devido a quantidade de cálculos que devem ser feitas na escala de bilhão.

Código-Fonte.: [fatorial\\_simples.c](#)

### 2. Aplicação 2.:

- No fatorial\_dois\_processos\_memoria\_comp com  $N = 1000$  é mais lento que o fatorial\_simples devido ao processo de compartilhamento de tarefas na memória, com  $N = 1.000.000$  aumenta-se um pouco a demanda em relação ao fatorial\_simples, entretanto nada que seja diferente do  $N = 1000$ , com  $N = 1.000.000.000$  ele é mais rápido que o fatorial\_simples, ou seja podemos concluir então que para  $N$  pequenos o fatorial\_simples é mais eficiente, entretanto para  $N$  grandes a memória compartilhada divide a carga de processamento em mais de um processo aumentando o rendimento se fosse apenas com um único processo.
- ☐ Entendimento sobre o código.: Este algoritmo divide o código em dois processos responsáveis por calcular uma parte do Fatorial, a função Fork() é responsável por criar esses dois processos permitindo acontecer um paralelismo oferecido pelo sistema de multiprocessamento como apresentado no livro e em sala de aula, tornando esse algoritmo mais eficiente para valores altos.

Código-Fonte.: [fatorial\\_memoria\\_compartilhada\\_main.c](#) // [fatorial\\_auxiliar.c](#)

### 3. Aplicação 3.:

- No fatorial\_2\_threads com  $N = 1000$  é quase a mesma coisa do fatorial\_simples, com  $N = 1.000.000$  temos um tempo de execução um pouco maior demonstrando que o trabalho sincronizado entre threads impactou no desempenho do algoritmo e com  $N = 1.000.000.000$  deu um tempo maior que o fatorial\_simples e o fatorial\_memoria\_compartilhada sugerindo que foi ineficiente em relação a esses dois métodos.

Código-Fonte.: [fatorial\\_2\\_threads.c](#)

### 4. Aplicação 4.:

- No fatorial\_10\_threads com  $N = 1000$  é mais lento que o fatorial\_simples e que o fatorial\_2\_threads, com  $N = 1.000.000$  ele continua sendo mais lento que os outros métodos e com  $N = 1.000.000.000$  a mesma coisa. Portanto, podemos concluir que o gerenciamento com 10 threads sobrecarrega a execução do algoritmo.

Código-Fonte.: [fatorial\\_10\\_threads.c](#)

### 5. Conclusão

- Fatorial\_Simples.: Eficiente com  $N$  pequeno e ineficiente com  $N$  muito grande;
- Fatorial\_memoria\_compartilhada.: Eficiente com  $N$  muito grande, sendo a melhor a opção para esse caso;

- Fatorial\_2\_Threads.: Péssimo para N pequeno e N muito grande;
- Fatorial\_10\_Threads.: Muito Péssimo para N pequeno e N muito grande;

## DESCRIÇÃO DAS APLICAÇÕES

### Aplicação 1 - Fatorial simples

Para ajudar, o código fonte desta aplicação em C está listado a seguir. Você pode alterar o valor de N no corpo do programa para realizar os experimentos.

#### Código-fonte: *fatorial.c*

```
#include <stdio.h>

// funcao recursiva que calcula o fatorial de n
unsigned long long int fatorial(int inicio, int n) {
    unsigned long long int resultado = 1;
    for (int i = inicio; i <= n; i++) {
        resultado *= i;
    }
    return resultado;
}

int main() {
    int numero = 1000; // defina AQUI o valor de N
    printf("%llu\n", fatorial(1, numero));
    return 0;
}
```



```
}
```

### **Aplicação 2 - Fatorial com 2 processos, comunicação com memória compartilhada.**

Nessa implementação, são executados 3 processos distintos. O processo main, que controla a criação de outros dois processos filhos, juntando seus resultados ao final; e dois processos filhos, responsáveis por executar metade do cálculo fatorial de N. O código dessa atividade está disponível nos arquivos `fatorial_memoria_compartilhada_main.c` e `fatorial_auxiliar.c`. Os códigos encontram-se sem qualquer comentário, sua tarefa é ler o código e entender. A partir desse entendimento, inferir como rodar, pois vai demandar uma ordem de compilação e nomenclatura dos binários. Ao final, você deverá adicionar comentários explicativos para os comandos utilizados, e reportar os resultados na tabela.

### **Aplicações 3 e 4 - Fatorial com 2 threads e com 10 threads**

Da mesma forma que o problema foi dividido pela metade na aplicação anterior, você deve dividir a tarefa em duas threads e em 10 threads nessa aplicação, usando Pthreads. Na aplicação 3 cada thread fica responsável por 50% do trabalho, na aplicação 4 cada thread fica responsável por 10% do trabalho. Em ambas, mantém-se a thread principal, controlando as demais e juntando os resultados. Tome por base o código de exemplo de uso de Pthreads apresentado no Capítulo 4 do livro Fundamentos de Sistemas Operacionais: princípios básicos para implementar essa solução. Faça então as devidas medições e anotações.

Qualquer dúvida, perguntem! Estou à disposição no telegram: @calebemicael

Bons estudos!