

# Sistemas Distribuídos

## *Atividade SpeedUP*

*Prof. Dr. Ricardo Destro*  
*2º Semestre de 2021*

# Lista de exercícios

Atividade #2

*Entrega até 30/08*



## SpeedUp

*SpeedUp pode ser definido como a relação entre o tempo gasto para executar uma tarefa com um único processador e o tempo gasto com P processadores.*

$$S = \frac{T_1}{T_P}$$

S -> SpeedUp

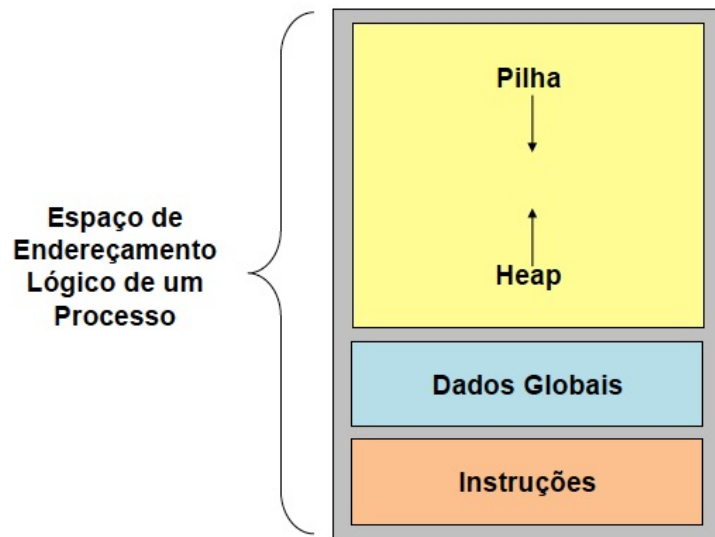
T<sub>1</sub> -> Tempo com 1 processador

T<sub>P</sub> -> Tempo com P processadores

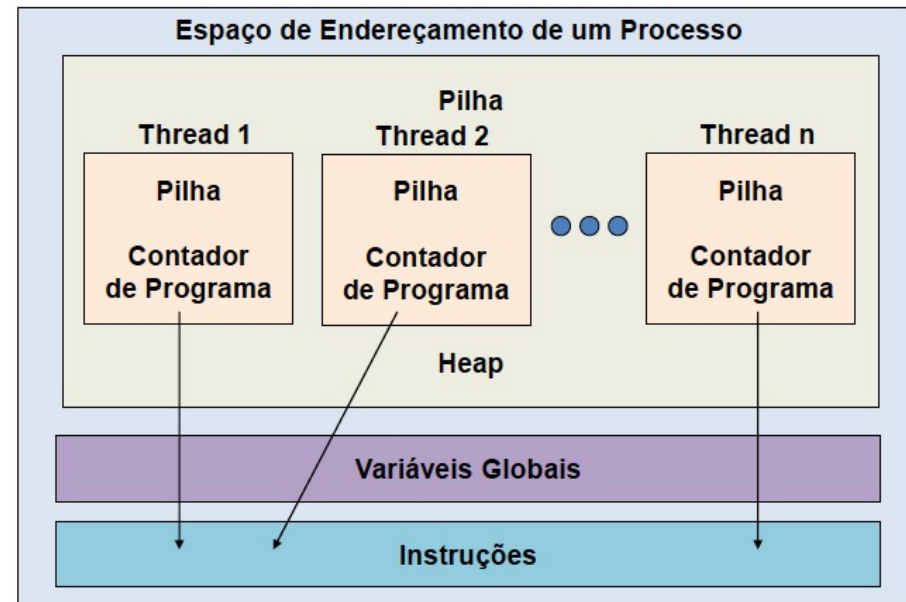
Vamos colocar em prática.....

## Multi-Thread ou Multi-Processo

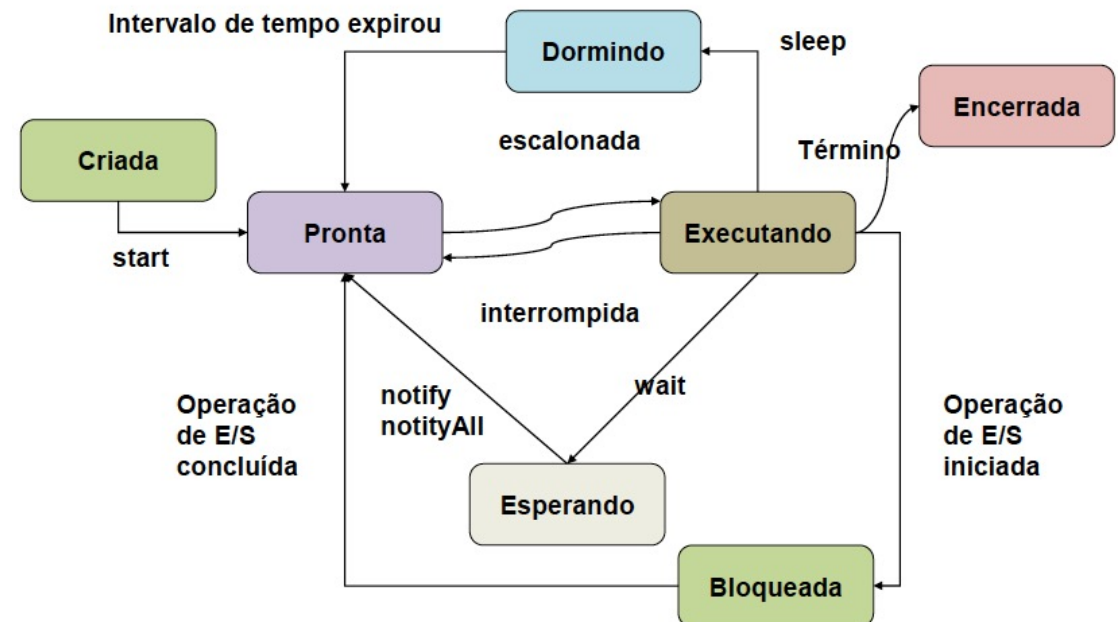
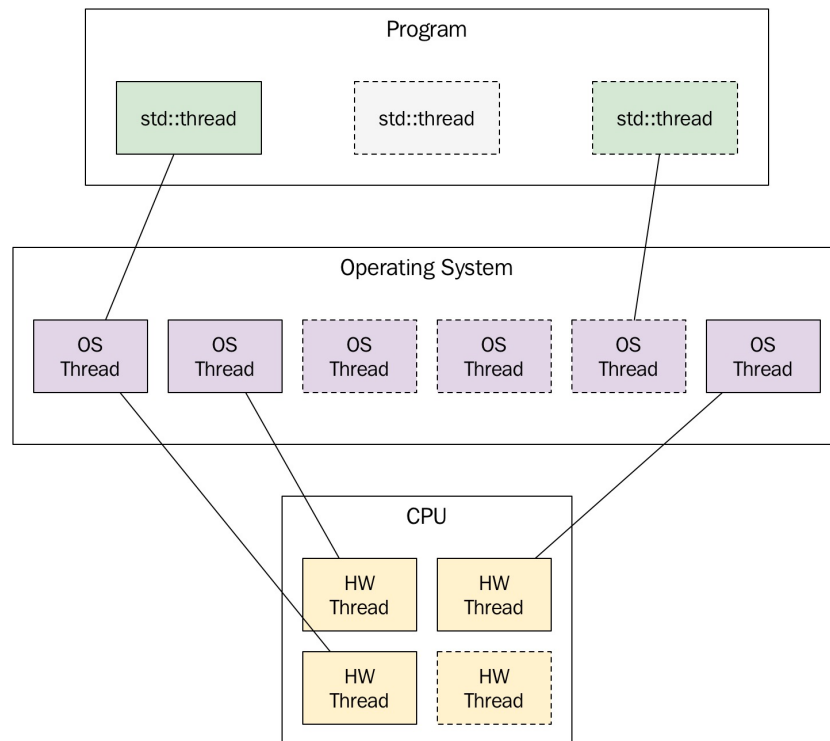
### Processos



### Threads



## Multi-Thread



## 1. **JAVA:**

1. [https://www.dca.ufrn.br/~affonso/DCA2401/2004\\_1/aulas/threads.pdf](https://www.dca.ufrn.br/~affonso/DCA2401/2004_1/aulas/threads.pdf)
2. <http://docs.fct.unesp.br/docentes/dmec/olivete/java/arquivos/Aula07.pdf>

## 2. **C/C++**

1. [http://www2.dcc.ufmg.br/disciplinas/aeds3\\_turmaN/pthreads.pdf](http://www2.dcc.ufmg.br/disciplinas/aeds3_turmaN/pthreads.pdf)
2. [https://www.dcc.fc.up.pt/~ricroc/aulas/1516/cp/apontamentos/slides\\_pthreads.pdf](https://www.dcc.fc.up.pt/~ricroc/aulas/1516/cp/apontamentos/slides_pthreads.pdf)

## 3. **Python**

1. <https://www.cin.ufpe.br/~dfp/files/CITi - Curso Python/Aula6ThreadsSocket.ppt>

1. Sincronização entre elas
2. Condições de corrida (race conditions)
3. Deadlock's
4. Localização de erros
5. Difícil garantia de correção dos programas  
(modelos analíticos e verificação formal)
6. Imprevisibilidade

1. É realmente um desafio
2. Comece pequeno... uma ou duas threads, poucos dados.
3. Coloque Logs/Prints
  1. Lembre-se de identificar a thread
4. **VALIDE OS RESULTADOS** – compare com a versão simples sempre que possível



Uma versão de exemplo está disponível para os alunos em:

*[https://github.com/rdestro/CC7261\\_AtividadeSpeedUp](https://github.com/rdestro/CC7261_AtividadeSpeedUp)*

É apenas uma referência que pode (e deve) ser melhorada!

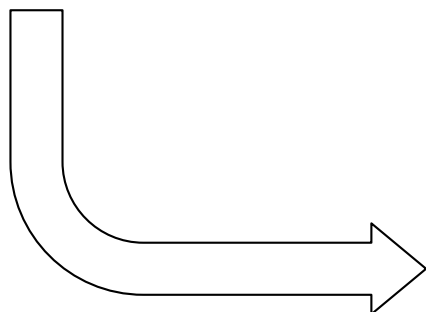
**A versão final pode ser desenvolvida em qualquer linguagem**

# PROJETO DE EXEMPLO

```
print('\n\analyse de %d valores\n\n'%(len(data)))
start1 = perf_counter_ns()
primo_sp = sp.resolve_simples(data)
finish1 = perf_counter_ns()

start2 = perf_counter_ns()
primo_mt = mt.resolve_trhread(data)
finish2 = perf_counter_ns()

print('simples          > threads')
print('%f ms    > %f ms    : tempo execucao'%((finish1-start1)/1000000,(finish2-start2)/1000000))
print('%d          > %d          :numeros primos encontrados'%(primo_sp,primo_mt))
print('SpeedUP = %f'%(((finish1-start1)/(finish2-start2))))
```



analise de 1500 valores

Valores apenas  
de exemplo

```
simples          > threads
91.113400 ms    > 41.855583 ms    : tempo execucao
1201          > 1201          :numeros primos encontrados
SpeedUP = 2.176852
```

# PROJETO DE EXEMPLO

```
start1 = perf_counter_ns()
primo_sp = sp.resolve_simples(data)
finish1 = perf_counter_ns()
```


```
def resolve_simples(data):
    tamanho_lista = len(data)
    primos = 0
    for i in range(tamanho_lista):
        if sympy.isprime(data[i]):
            primos += 1
    return primos
```

```
start2 = perf_counter_ns()
primo_mt = mt.resolve_trhread(data)
finish2 = perf_counter_ns()
```

```
def resolve_trhread(data):
    ThreadsQtdd = 5
    tamanho_lista = len(data)
    index = range(0, tamanho_lista+(tamanho_lista//ThreadsQtdd), tamanho_lista//ThreadsQtdd)
    primos = 0
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = []
        for i in range(ThreadsQtdd):
            futures.append(executor.submit(tCalculaPrimo, data=data[index[i]:index[i+1]]))
        for future in concurrent.futures.as_completed(futures):
            primos += future.result()
    return primos
```

# PROJETO DE EXEMPLO

```
def resolve_trhread(data):  
    ThreadsQtdd = 5  
    tamanholista = len(data)  
    index = range(0, tamanholista+(tamanholista//ThreadsQtdd), tamanholista//ThreadsQtdd)  
    primos = 0  
    with concurrent.futures.ThreadPoolExecutor() as executor:  
        futures = []  
        for i in range(ThreadsQtdd):  
            futures.append(executor.submit(tCalculaPrimo, data=index[i]:index[i+1]))  
        for future in concurrent.futures.as_completed(futures):  
            primos += future.result()  
    return primos
```



```
def tCalculaPrimo(data):  
    primos = 0  
    for i in range(len(data)):  
        if sympy.isprime(data[i]):  
            primos += 1  
    return primos
```

Foi fornecida uma base com 250.000 números.

Precisamos descobrir quais são primos e quais não são.

Simple assim! ;)

1. Vamos adotar a solução sem threads como sendo a execução sem melhorias.
2. Depois, acrescentamos threads.....  
Calculamos o SpeedUP
3. Quanto mais threads, o SpeedUp melhora?  
Qual a melhor quebra para ser feita?
4. Por que se aumentamos MUITO a quantidade de threads, perdemos o SpeedUp

A cada execução, obtemos um tempo diferente, certo?

Para simplificar:

- Faça 50 execuções de cada caso/cenário.
- Calcule a média.
- Automatize estes cenários/cálculos

1. Responda as questões apresentadas
2. Coloque um link para o seu código no GitHub
3. Faça melhorias, inclua gráficos... É o seu relatório!
4. Para pensar: é possível melhorar o “solução” simples apresentada? Pode melhorar.... Coloque no relatório.