

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Tipos de Programação

Para cada uma das alternativas, faça a correspondência com o modelo de programação que é mais corretamente definido pela característica:

Operações são feitas em simultâneo	Programação paralela	✓
Nunca tira partido de CPUs multi-core	Programação concorrente	✗
Necessita de hardware compatível	Programação paralela	✓
As instruções de um programa são executadas uma após a outra.	Programação sequencial	✓
Pode permitir resolver em simultâneo sub-problemas de um problema maior	Programação paralela	✓
Preocupa-se principalmente com a eficiência da computação	Programação paralela	✓
Preocupa-se em como as computações a executar trocam informações	Programação paralela	✗
Pode ou não fazer múltiplas execuções ao mesmo tempo	Programação concorrente	✓
Preocupa-se em gerir qual tarefa vai ser executada a seguir	Programação sequencial	✗

A resposta correcta é:

Operações são feitas em simultâneo → Programação paralela,

Nunca tira partido de CPUs multi-core → Programação sequencial,

Necessita de hardware compatível → Programação paralela,

As instruções de um programa são executadas uma após a outra. → Programação sequencial,

Pode permitir resolver em simultâneo sub-problemas de um problema maior → Programação paralela,

Preocupa-se principalmente com a eficiência da computação → Programação paralela,

Preocupa-se em como as computações a executar trocam informações → Programação concorrente,

Pode ou não fazer múltiplas execuções ao mesmo tempo → Programação concorrente,

Preocupa-se em gerir qual tarefa vai ser executada a seguir → Programação concorrente

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Sobre programação distribuída e programação paralela, faça a correspondência adequada:

Sistemas autônomos diferentes	Programação distribuída	✓
Geralmente usa sistemas operativos diferentes	Programação distribuída	✓
Usa servidores remotos que podem por sua vez dividir o trabalho para vários processadores	Programação paralela	✗
A meta é processar o problema total o mais rápido possível	Programação paralela	✓
Geralmente usa sistemas operativos iguais	Programação paralela	✓
Um problema é particionado por vários processadores	Programação paralela	✓

A resposta correcta é:

Sistemas autônomos diferentes → Programação distribuída,

Geralmente usa sistemas operativos diferentes → Programação distribuída,

Usa servidores remotos que podem por sua vez dividir o trabalho para vários processadores → Programação distribuída,

A meta é processar o problema total o mais rápido possível → Programação paralela,

Geralmente usa sistemas operativos iguais → Programação paralela,

Um problema é particionado por vários processadores → Programação paralela

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Para cada uma das alternativas, faça a correspondência com o modelo de programação que é mais corretamente definido pela característica:

Nunca tira partido de CPUs multi-core	Programação sequencial	✓
Pode ou não fazer múltiplas execuções ao mesmo tempo	Programação concorrente	✓
Necessita de hardware compatível	Programação paralela	✓
Operações são feitas em simultâneo	Programação paralela	✓
Preocupa-se em como as computações a executar trocam informações	Programação concorrente	✓
Pode permitir resolver em simultâneo sub-problemas de um problema maior	Programação concorrente	✗
As instruções de um programa são executadas uma após a outra.	Programação sequencial	✓
Preocupa-se em gerir qual tarefa vai ser executada a seguir	Programação sequencial	✗
Preocupa-se principalmente com a eficiência da computação	Programação paralela	✓

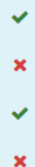
A resposta correcta é:

- Nunca tira partido de CPUs multi-core → Programação sequencial,
- Pode ou não fazer múltiplas execuções ao mesmo tempo → Programação concorrente,
- Necessita de hardware compatível → Programação paralela,
- Operações são feitas em simultâneo → Programação paralela,
- Preocupa-se em como as computações a executar trocam informações → Programação concorrente,
- Pode permitir resolver em simultâneo sub-problemas de um problema maior → Programação paralela,
- As instruções de um programa são executadas uma após a outra. → Programação sequencial,
- Preocupa-se em gerir qual tarefa vai ser executada a seguir → Programação concorrente,
- Preocupa-se principalmente com a eficiência da computação → Programação paralela

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Assinale as razões que tornam a computação paralela mais complexa que a computação sequencial.

- ☒ a. Dificuldade em dividir uma computação sequencial em sub-computações paralelas
- ☒ b. Exigência de mais de um computador
- ☒ c. Aumento de erros potenciais na programação paralela
- ☒ d. Exigência de hardware auto-configurável
- ☐ e. Exigência de linguagens de programação específicas



Respostas corretas:

Dificuldade em dividir uma computação sequencial em sub-computações paralelas,  
Aumento de erros potenciais na programação paralela

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Programação Sequencial

Assinale as razões que tornam a computação paralela mais complexa que a computação sequencial.

- ☐ a. Exigência de linguagens de programação específicas
- ☒ b. Exigência de hardware auto-configurável
- ☒ c. Aumento de erros potenciais na programação paralela
- ☒ d. Dificuldade em dividir uma computação sequencial em sub-computações paralelas
- ☐ e. Exigência de mais de um computador

✗

✓

✓

Respostas corretas:

Dificuldade em dividir uma computação sequencial em sub-computações paralelas,

Aumento de erros potenciais na programação paralela

## Programação Concorrente

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Faça a correspondência adequada, segundo o que foi visto na aula sobre Introdução à Computação Paralela:

Simulação do comportamento de fluidos	Computação paralela	✓
Servidores Web	Computação concorrente	✓
Interfaces do utilizador	Computação concorrente	✓
Processamento de <i>big data</i>	Computação paralela	✓
Multiplicação de matrizes	Computação paralela	✓
Renderização de gráficos 3D	Computação paralela	✓
Bases de dados	Computação concorrente	✓

A resposta correcta é:

Simulação do comportamento de fluidos → Computação paralela,

Servidores Web → Computação concorrente,

Interfaces do utilizador → Computação concorrente,

Processamento de *big data* → Computação paralela,

Multiplicação de matrizes → Computação paralela,

Renderização de gráficos 3D → Computação paralela,

Bases de dados → Computação concorrente

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Programação Paralela

### Seções críticas

Sobre **seções críticas** num programa paralelo, faça a correspondência adequada:

Os processos aguardam até que uma condição esteja satisfeita

Sincronização condicional



O acesso a uma variável é efetuado à vez pelos processos

Serialização



Garante que as seções críticas não vão ser intercaladas com nenhum outro código

Sincronização com exclusão mútua



Somente um processo executa num certo momento

Atomicidade



A resposta correcta é:

Os processos aguardam até que uma condição esteja satisfeita → Sincronização condicional,

O acesso a uma variável é efetuado à vez pelos processos → Sincronização com exclusão mútua,

Garante que as seções críticas não vão ser intercaladas com nenhum outro código

→ Atomicidade,

Somente um processo executa num certo momento → Serialização

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## VLIW

O uso nas CPUs de palavras de instrução muito longas, ou VLIW, como um mecanismo de melhoria do desempenho através do processamento paralelo de instruções, requer que os compiladores reconheçam quais instruções são independentes umas das outras e as organize na *instruction word*.

Selecione uma opção:

- ☒ Verdadeiro ✓
- ☐ Falso

A resposta correta é: Verdadeiro

## Pipelining

Na disciplina de Arquitetura, viram como funciona o *pipelining* dentro da CPU. Assinale a alternativa menos correta:

- ☒ a. O *pipelining* pode ser utilizado em processadores que permitem ILP (*instruction level parallelism*) ✖
- ☐ b. O *pipelining* a nível de instrução foi habilitado com a evolução das CPUs de 32 para 64 bits
- ☐ c. O *pipelining* reduz o tempo médio de execução por instrução
- ☐ d. Os diferentes estágios do *pipelining* podem executar em paralelo no mesmo core
- ☐ e. O *pipelining* interno da CPU decompõe o processamento das instruções em estágios de *pipelining*

A resposta correta é:

O *pipelining* a nível de instrução foi habilitado com a evolução das CPUs de 32 para 64 bits



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Se eu dividir uma computação entre meu desktop com placa NVIDIA e meu laptop, conectados à mesma rede local, que classe de sistemas paralelos vou estar a usar?

- ☐ a. Todo o processamento será feito na GPU da placa NVIDIA
- ☐ b. Cluster homogêneo de computadores
- ☐ c. Multiprocessadores simétricos, se a CPU do laptop tiver pelo menos 2 cores físicos
- ☐ d. Construí um FPGA
- ☒ e. Um cluster heterogêneo de computadores



A resposta correta é:

Um cluster heterogêneo de computadores

## Distribuição de dados

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Sobre **distribuição de dados** para processamento paralelo, faça a correspondência adequada:

Todos os dados ficam na mesma memória

Memória partilhada ⇅



É necessário um padrão para evitar que cada core ou processador aceda a dados que deveriam ser manipulados por outra unidade de processamento

Memória partilhada ⇅



Cada processador só pode aceder aos dados que lhe foram atribuídos

Memória distribuída ⇅



Somente os dados atribuídos a uma unidade de processamento ficam na memória local

Memória distribuída ⇅



A resposta correcta é:

Todos os dados ficam na mesma memória → Memória partilhada,

É necessário um padrão para evitar que cada core ou processador aceda a dados que deveriam ser manipulados por outra unidade de processamento → Memória partilhada,

Cada processador só pode aceder aos dados que lhe foram atribuídos → Memória distribuída,

Somente os dados atribuídos a uma unidade de processamento ficam na memória local → Memória distribuída

## Overclock

O *overclock* do processador aumenta o consumo de energia proporcionalmente ao aumento da frequência de relógio.

Selecione uma opção:

☐ Verdadeiro

☒ Falso ✓

A resposta correta é: Falso

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

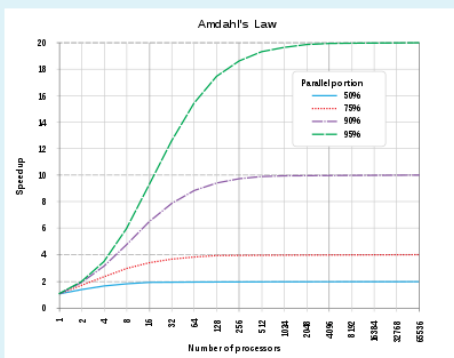
Gene Amdahl propôs uma fórmula para o limite teórico

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Gene Amdahl propôs uma fórmula para o limite teórico de ganho em rapidez de um programa executado de forma paralela em múltiplos processadores. A fórmula baseia-se na percentagem (em tempo) do programa que pode ser paralelizada e resulta no seguinte gráfico, retirado da Wikipedia:



Segundo este gráfico, e sendo  $p$  a percentagem paralelizável de um programa, quais das alternativas abaixo estão corretas?

- ☒ a. Para qualquer número de processadores utilizados pelo programa, o ganho de eficiência sempre será  $\frac{1}{1-p}$  ✗
- ☐ b. O ganho incremental em eficiência decresce com o número de unidades de processamento acrescentadas
- ☐ c. O ganho de eficiência tem uma correspondência linear com o número de unidades de processamento utilizadas
- ☐ d. O ganho incremental em eficiência não tem a ver com o número de unidades de processamento acrescentadas
- ☐ e. Para qualquer número de processadores utilizados pelo programa, o ganho de eficiência nunca ultrapassa  $\frac{1}{1-p}$

Respostas corretas:

O ganho incremental em eficiência decresce com o número de unidades de processamento acrescentadas ,

Para qualquer número de processadores utilizados pelo programa, o ganho de eficiência nunca ultrapassa  $\frac{1}{1-p}$

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Granularidade

Sobre a **granularidade** das aplicações de computação paralela, faça a correspondência adequada:

Numa multiplicação de matrizes, divide-se as matrizes em blocos menores que são multiplicados em paralelo

Escolha...

O servidor web faz multi-threading e usa cores físicas diferentes para atender diferentes utilizadores

Paralelismo bit-level

Ao aumentar o tamanho da palavra de memória, o processamento é realizado com menos instruções

Paralelismo bit-level

A resposta correcta é:

Numa multiplicação de matrizes, divide-se as matrizes em blocos menores que são multiplicados em paralelo → Paralelismo instruction-level,

O servidor web faz multi-threading e usa cores físicas diferentes para atender diferentes utilizadores → Paralelismo task-level,

Ao aumentar o tamanho da palavra de memória, o processamento é realizado com menos instruções → Paralelismo bit-level

## Multithreading

Sobre **multithreading**, faça a correspondência adequada:

Alterna entre processadores virtuais somente nas maiores interrupções, tal como um *miss* na cache de nível 2.

Fine-grained multithreading

Alterna implicitamente para o próximo processador virtual após cada instrução.

Fine-coarse multithreading

Oculta a demora no acesso à memória ao simular um número fixo de processadores virtuais para cada processador físico.

Coarse-grained multithreading

A resposta correcta é:

Alterna entre processadores virtuais somente nas maiores interrupções, tal como um *miss* na cache de nível 2. → Coarse-grained multithreading,

Alterna implicitamente para o próximo processador virtual após cada instrução. → Fine-grained multithreading,

Oculta a demora no acesso à memória ao simular um número fixo de processadores virtuais para cada processador físico. → Fine-grained multithreading

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Hyperthreading

Sobre *Hyperthreading*, marque a alternativa menos correta:

- ☐ a. Um processador ou core físico é visto pelo sistema como duas unidades lógicas de processamento
- ☐ b. O sistema operativo pode atribuir dois processos ou duas threads ao processador com um core físico
- ☐ c. O aumento total de área do chip habilitado ao hyperthreading é de 5%
- ☒ d. Duplica a região utilizada para processamento
- ☐ e. Os threads ou processos atribuídos pelo sistema operativo podem vir da mesma aplicação ou de aplicações diferentes

A resposta correta é: Duplica a região utilizada para processamento

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Sistemas multi-processadores

Sobre o espaço de memória para sistemas multi-processadores, faça a correspondência adequada:

Espaço de endereçamento de memória distribuído	Troca de dados pelo uso de mensagens	✓
Espaço de endereçamento de memória partilhado	Troca de dados pelo uso de variáveis	✓

A resposta correcta é:

Espaço de endereçamento de memória distribuído → Troca de dados pelo uso de mensagens,

Espaço de endereçamento de memória partilhado → Troca de dados pelo uso de variáveis

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

## Redes de Interconexão

Sobre Redes de Interconexão, faça a correspondência adequada:


Número mínimo de nós que devem ser removidos para desconectar a rede

Conectividade de nós  

Distância máxima entre quaisquer dois nós da rede

Diâmetro  

Número de nós adjacentes a um nó

Grau do nó  

A resposta correcta é:

Número mínimo de nós que devem ser removidos para desconectar a rede → Conectividade de nós,

Distância máxima entre quaisquer dois nós da rede → Diâmetro,

Número de nós adjacentes a um nó → Grau do nó

Sobre Redes de Interconexão, faça a correspondência das características com o valor desejado:

Escalabilidade e expandabilidade

Grande  

Grau de cada nó

Pequeno  


Diâmetro

Grande  

Largura de banda para bisseccionamento

Grande  

Conectividade

Pequeno  





```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## VLSI

VLSI significa:

- ☒ a. Integração em muito larga escala
- ☐ b. Interface de software de carga virtual
- ☐ c. Sistemas integrados de carga virtual
- ☐ d. Informação em muito larga escala (Big Data)
- ☐ e. Sistemas de informação em muito larga escala



A resposta correta é:  
Integração em muito larga escala

## Estratégia de switching

A estratégia de *switching*, ou chaveamento, determina se uma mensagem deve ser dividida em pacotes menores para transmissão.

Selecione uma opção:

- ☒ Verdadeiro
- ☐ Falso

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Taxonomia de Flynn

Sobre a taxonomia de Flynn, faça a correspondência adequada:

Em cada passo do processamento, várias instruções são aplicadas sobre vários valores	MIMD	✓
Em cada passo do processamento, múltiplas instruções são aplicadas sobre um só valor	MISD	✓
Em cada passo do processamento, a mesma instrução é aplicada sobre vários valores	SIMD	✓
Em cada passo do processamento, uma instrução é aplicada sobre um só valor	SISD	✓

A resposta correcta é:

- Em cada passo do processamento, várias instruções são aplicadas sobre vários valores → MIMD,
- Em cada passo do processamento, múltiplas instruções são aplicadas sobre um só valor → MISD,
- Em cada passo do processamento, a mesma instrução é aplicada sobre vários valores → SIMD,
- Em cada passo do processamento, uma instrução é aplicada sobre um só valor → SISD

## Memory Wall

A integração de caches no chip da CPU não melhora o problema da barreira de memória (*Memory Wall*), porque continua sendo memória com desvantagem no tempo de acesso em relação à velocidade da CPU.

Selecione uma opção:

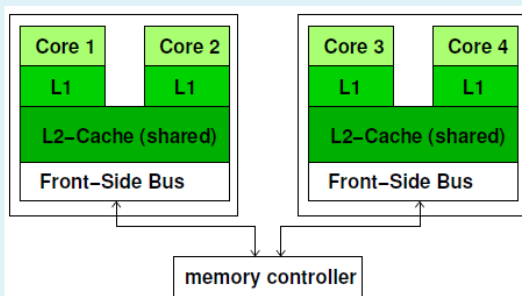
- ☐ Verdadeiro
- ☒ Falso ✓

A resposta correta é: Falso

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Tipos de arquitetura

A figura mostra uma parte da arquitetura Intel Quad-Core Xeon. Esta arquitetura segue um desenho:



- ☒ a. Misto
- ☐ b. Hierárquico
- ☐ c. Network based
- ☐ d. Pipeline
- ☐ e. Cache based

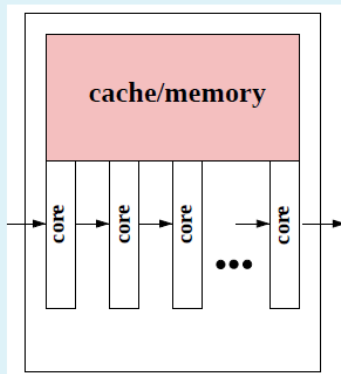
A resposta correta é:  
Hierárquico

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

A arquitetura mostrada na figura um desenho:



- ☐ a. Hierárquico
- ☒ b. Cache based
- ☐ c. Misto
- ☐ d. Pipeline
- ☐ e. Network based

✗

A resposta correta é:  
Pipeline

## Processamento ILP (Instruction-Level Parallelism)

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

A lógica de controlo para um processamento ILP (*Instruction-Level Parallelism*) não requer hardware de alta complexidade.

Selecione uma opção:

- ☒ Verdadeiro ✖
- ☐ Falso

A resposta correta é: Falso

## Localidade no acesso à memória

Sobre a Localidade no acesso à memória, assinale a alternativa menos correta

- ☐ a. Localidade temporal diz respeito ao uso repetido da mesma instrução ou mesmo dado num dado intervalo
- ☐ b. O carregamento de blocos de instruções ou dados da RAM para a cache é transparente para as aplicações
- ☐ c. Localidade espacial diz respeito ao acesso a instruções ou dados próximos uns dos outros
- ☒ d. Linha de cache e bloco de cache são conceitos diferentes ✓
- ☐ e. A localidade espacial ou temporal implica menor números de leitura da RAM

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

# Caches

Assinale a alternativa menos correta sobre caches:

- ☐ a. O processador sempre verifica se o dado requisitado está na cache antes de tentar aceder à memória RAM
- ☐ b. Caches são memórias DRAM mais pequenas e embebidas no chip do processador
- ☐ c. Em sistemas multi-núcleos, cada acesso de leitura deve retornar o valor mais recentemente escrito
- ☒ d. Tipicamente são utilizados vários níveis de cache, onde o L1 é o nível mais pequeno, rápido e caro e está mais próximo do processamento ✗
- ☐ e. Em sistemas multi-núcleos, cada núcleo deve ter uma visão coerente do sistema de memória (cache + RAM)

A resposta correta é:

Caches são memórias DRAM mais pequenas e embebidas no chip do processador

Assinale as duas alternativas mais corretas:

- ☒ a. O tempo de acesso à memória DRAM não acompanhou o crescimento na velocidade dos processadores e isso cria uma barreira de desempenho ✓
- ☐ b. O desempenho medido de um processador pode ser diferente se estiver a trabalhar com números reais ou com números inteiros
- ☒ c. O tempo de acesso à memória SRAM não acompanhou o crescimento na velocidade dos processadores e isso cria uma barreira de desempenho ✗
- ☐ d. O crescente número de transístores cria barreiras à paralelização ao nível de instruções (ILP)
- ☐ e. Antes de 2003 não era possível fazer *overclock* ao processador

Respostas corretas:

O desempenho medido de um processador pode ser diferente se estiver a trabalhar com números reais ou com números inteiros,

O tempo de acesso à memória DRAM não acompanhou o crescimento na velocidade dos processadores e isso cria uma barreira de desempenho

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Algoritmos de encaminhamento

Sobre algoritmos de encaminhamento, faça a correspondência adequada:

Estabelecimento de um caminho concreto	Algoritmo mínimo	✗
Escolha do menor caminho entre A e B	Algoritmo determinístico	✗
Considera a utilização dinâmica	Algoritmo adaptativo	✓
Escolha do caminho observando a utilização da rede	Algoritmo não-mínimo	✓
Source based algorithm	Algoritmo determinístico	✓

A resposta correcta é:

Estabelecimento de um caminho concreto → Algoritmo determinístico,

Escolha do menor caminho entre A e B → Algoritmo mínimo,

Considera a utilização dinâmica → Algoritmo adaptativo,

Escolha do caminho observando a utilização da rede → Algoritmo não-mínimo,

Source based algorithm → Algoritmo determinístico



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Processos e *Threads*

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Sobre processos e *threads*, faça a correspondência adequada:

Processos concorrentes mas não paralelos	Processos	✗
Têm um espaço partilhado de endereçamento	Threads	✓
Têm um espaço próprio de endereçamento	Processos	✓
Fazem <i>context switch</i> através do escalonamento do sistema	Multitasking	✗
Programas em execução	Processos	✓
Fluxos de computações do mesmo programa	Multitasking	✗

A resposta correcta é:

Processos concorrentes mas não paralelos → Multitasking,  
Têm um espaço partilhado de endereçamento → Threads,  
Têm um espaço próprio de endereçamento → Processos,  
Fazem *context switch* através do escalonamento do sistema → Processos,  
Programas em execução → Processos,  
Fluxos de computações do mesmo programa → Threads

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Overhead

O overhead para criar *threads* e para criar processos é semelhante.

Selecione uma opção:

- ☐ Verdadeiro
- ☒ Falso ✓

A resposta correta é: Falso

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Nível de abstração

Qual a ordem correta desde o nível de abstração mais baixo até ao mais elevado?

- ☐ a. Modelos de programação paralela, Modelos computacionais paralelos, Modelos de arquiteturas paralelas, Modelos de máquinas paralelas
- ☐ b. Modelos de programação paralela, Modelos de máquinas paralelas, Modelos computacionais paralelos, Modelos de arquiteturas paralelas
- ☐ c. Modelos computacionais paralelos, Modelos de programação paralela, Modelos de máquinas paralelas, Modelos de arquiteturas paralelas
- ☐ d. Modelos de máquinas paralelas, Modelos computacionais paralelos, Modelos de arquiteturas paralelas, Modelos de programação paralela
- ☒ e. Modelos de máquinas paralelas, Modelos de arquiteturas paralelas, Modelos computacionais paralelos, Modelos de programação paralela ✓

A resposta correta é:

Modelos de máquinas paralelas, Modelos de arquiteturas paralelas, Modelos computacionais paralelos, Modelos de programação paralela

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## IO-bound

Tem um programa *IO-bound* que leva 2 segundos para executar e é lançado uma vez por semana. Qual estratégia de concorrência deve utilizar?

- ☐ a. *Threading*
- ☒ b. Múltiplos processos com múltiplos *threads* cada um
- ☐ c. Não deve usar concorrência
- ☐ d. *Asyncio*
- ☐ e. *Multiprocessing*



A resposta correta é:  
Não deve usar concorrência

## Loops paralelos

Sobre *loops* paralelos, faça a correspondência adequada:

As instruções sem inter-dependência são separadas e podem ser paralelizadas

Loop distribuído ✓

Todas as instruções podem ser executadas paralelamente

Paralelismo DOALL ✓

A resposta correcta é:

As instruções sem inter-dependência são separadas e podem ser paralelizadas → Loop distribuído,

Todas as instruções podem ser executadas paralelamente → Paralelismo DOALL

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Extra

Assinale a alternativa menos correta:

- ☐ a. Devido à crescente disparidade entre a redução do tempo de acesso à RAM e o aumento da frequência de relógio da CPU, o tempo de acesso à RAM contado em ciclos de relógio aumentou entre 1990 e 2003 mais de 30 vezes
- ☐ b. A distância entre os componentes de um chip é uma preocupação dos projetistas
- ☒ c. O processamento de uma instrução não é afetado pelo tamanho do chip da CPU ✓
- ☐ d. O *overclock* pode causar que uma instrução demore mais ciclos de relógio para executar
- ☐ e. O aumento do número de transístores por área de um chip provoca maior desperdício de energia

A resposta correta é:

O processamento de uma instrução não é afetado pelo tamanho do chip da CPU

GPUs são baseadas na classe onde, em cada passo do processamento, a mesma instrução é aplicada a vários valores.

Selecione uma opção:

- ☐ Verdadeiro
- ☒ Falso ✗

A resposta correta é: Verdadeiro

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Na paralelização de ciclos `for`, a complexidade resultante (e portanto o ganho de eficiência) é afetada pela interdependência das operações realizadas em cada iteração.

Selecione uma opção:

- ☒ Verdadeiro ✓
- ☐ Falso

A resposta correta é: Verdadeiro

O número de transistores num processador é uma indicação grosseira do seu desempenho e complexidade

Selecione uma opção:

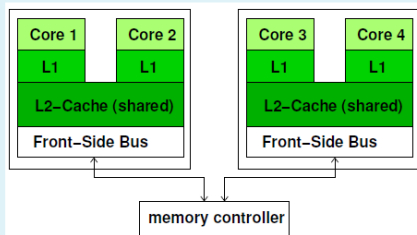
- ☒ Verdadeiro ✓
- ☐ Falso

A resposta correta é: Verdadeiro

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

# 1º TESTE

A figura mostra uma parte da arquitetura Intel Quad-Core Xeon. Esta arquitetura segue um desenho:



- ☐ a. Pipeline
- ☐ b. Network based
- ☐ c. Misto
- ☒ d. Hierárquico
- ☐ e. Cache based

A sua resposta está correta.

A resposta correta é:  
Hierárquico



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Faça a correspondência adequada entre o tipo de processamento e a biblioteca a utilizar:

Programa que calcula se um número é primo	Concurrencying	✗
Programa que calcula a inversa e o determinante de uma matriz 1000x1000	Threading	✗
Programa de acesso aleatório a uma base de dados	Multiprocessing	✗
Programa de processamento de ficheiros de imagens	Threading	✓

A sua resposta está parcialmente correta.

Selecionou 1 respostas corretas.

A resposta correcta é:

Programa que calcula se um número é primo → Multiprocessing,

Programa que calcula a inversa e o determinante de uma matriz 1000x1000 → Multiprocessing,

Programa de acesso aleatório a uma base de dados → Threading,

Programa de processamento de ficheiros de imagens → Threading

O *pipelining* da CPU é uma característica normal nos processadores hoje em dia.

Selecione uma opção:

- ☒ Verdadeiro ✓
- ☐ Falso

A resposta correta é: Verdadeiro

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Por que temos feito muitos exemplos e exercícios nas práticas de Computação Paralela e Distribuída usando a multiplicação de matrizes (assinale a melhor resposta):

- ☐ a. Porque é muito simples de paralelizar e diminuir o tempo de execução usando *threads* no Python.
- ☐ b. Porque o algoritmo é complexo.
- ☐ c. Porque podemos utilizar muitas matrizes pré-definidas no Python e no NumPy.
- ☐ d. Podíamos ter escolhido qualquer outra estrutura de dados, matrizes e multiplicação de matrizes são assuntos somente teóricos.
- ☒ e. Porque tem numerosas aplicações em várias áreas científicas e é uma ferramenta básica da álgebra linear.



A sua resposta está correta.

A resposta correta é:

Porque tem numerosas aplicações em várias áreas científicas e é uma ferramenta básica da álgebra linear.

Associe corretamente:

Reutilização de dados utilizados recentemente	Localidade temporal	✓
Uso de dados em proximidade de dados utilizados recentemente	Localidade temporal	✗
Salvar em cache valores frequentemente utilizados	Localidade espacial	✗
Ler blocos inteiros de memória com dados que vão ser todos utilizados	Localidade espacial	✓

A sua resposta está parcialmente correta.

Selecionou 2 respostas corretas.

A resposta correcta é: Reutilização de dados utilizados recentemente → Localidade temporal, Uso de dados em proximidade de dados utilizados recentemente → Localidade espacial, Salvar em cache valores frequentemente utilizados → Localidade temporal, Ler blocos inteiros de memória com dados que vão ser todos utilizados → Localidade espacial

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Que é o GIL do Python?

- ☒ a. É um bloqueio para permitir que apenas um thread tenha o controlo do interpretador Python por vez
- ☐ b. São as iterações globais e locais do interpretador Python que permitem declarar o contexto de uma variável
- ☐ c. É o mecanismo que o Python utiliza para correr múltiplas threads em simultâneo
- ☐ d. São as garantias locais do interpretador Python que evitam hacking do código
- ☐ e. São as bibliotecas interativas globais do Python que permitem correr o nosso código em qualquer máquina



A sua resposta está correta.

A resposta correcta é:

É um bloqueio para permitir que apenas um thread tenha o controlo do interpretador Python por vez

Há situações favoráveis à paralelização e há situações inibidoras da paralelização. Faça a correspondência:

Dependência das instruções	Candidato à paralelização	✗
Ciclo	Inibidor da paralelização	✗
Dependência dos dados	Candidato à paralelização	✗

A sua resposta está incorreta.

A resposta correcta é: Dependência das instruções → Inibidor da paralelização, Ciclo → Candidato à paralelização, Dependência dos dados → Inibidor da paralelização

Faça a correspondência adequada entre o tipo de processamento e a biblioteca a utilizar:

Processamento <i>coarse-grained</i>	Multiprocessing	✓
Programas <i>CPU-bound</i>	Multiprocessing	✓
Processamento <i>fine-grained</i>	Threading	✓
Programas <i>IO-bound</i>	Concurrencying	✗

A sua resposta está parcialmente correta.

Selecionou 3 respostas corretas.

A resposta correcta é:

Processamento *coarse-grained* → Multiprocessing,

Programas *CPU-bound* → Multiprocessing,

Processamento *fine-grained* → Threading,

Programas *IO-bound* → Threading

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Como se entra e como se sai da consola Python interativa (o REPL)?

- ☐ a. Para entrar: `python --i`  
Para sair: `Ctrl-C`
- ☐ b. Para entrar: `python`  
Para sair: `quit`
- ☐ c. Para entrar: `python`  
Para sair: `exit()`
- ☐ d. Para entrar: `python`  
Para sair: `exit`
- ☒ e. Para entrar: `python --repl`  
Para sair: `Ctrl-C`



A sua resposta está incorreta.

A resposta correta é:

Para entrar: `python`

Para sair: `exit()`

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

A estratégia de *switching* tem influência sobre o tempo de transmissão de uma mensagem?

- ☐ a. Não
- ☐ b. Sim, mas a influência é pequena
- ☒ c. Sim, grande influência



A sua resposta está correta.

A resposta correta é:

Sim, grande influência

Assinale a alternativa INCORRETA:

Selecione uma opção de resposta:

- ☐ a. Nem todas as aplicações têm potencial de paralelismo que faça valer a pena o esforço de paralelizar o código
- ☒ b. O tamanho das sub-tarefas em computação paralela é importante para permitir que caibam na memória
- ☐ c. Uma dificuldade para executar programas paralelos é escolher a máquina e o sistema operativo, dado que há poucas opções no mercado
- ☐ d. A estimativa de custo/benefício da paralelização deve considerar os tempos necessários para mover os dados na hierarquia de memória.
- ☐ e. Podemos configurar nossa aplicação paralelizada para que reajuste dinamicamente as tarefas conforme a disponibilidade dos processadores envolvidos



A sua resposta está incorreta.

A resposta correta é: Uma dificuldade para executar programas paralelos é escolher a máquina e o sistema operativo, dado que há poucas opções no mercado

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

O *pipelining* não tem paralelo com uma linha de montagem, é um processo em que só uma tarefa usa os recursos de cada vez.

Selecione uma opção:

☐ Verdadeiro

☒ Falso ✓

A resposta correta é: Falso

A frequência de relógio dos processadores tende a continuar aumentando sem limite porque novas técnicas de fabrico permitem arrefecer o equipamento.

Selecione uma opção:

☒ Verdadeiro ✗

☐ Falso

A resposta correta é: Falso

No código Python abaixo, qual o resultado?

```
frase = "Ouço...esqueço; Leio...aprendo; Faço...entendo!"
print ("aprendo" in frase)
```

☐ a. frase[23:29]

☐ b. frase[23]

☐ c. 23:29

☒ d. True ✓

☐ e. 23

A sua resposta está correta.

A resposta correta é:

True

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Faça as correspondências adequadas:

Dados particionados	Modelo de memória partilhada ✖
A coordenação é feita através das variáveis partilhadas	Modelo de memória partilhada ✔
A aplicação é uma coleção de processos	Comunicação por mensagens ✔
A aplicação é uma coleção de <i>threads</i>	Modelo de memória partilhada ✔
Diversos processos em diversas máquinas	Comunicação por mensagens ✔

A sua resposta está parcialmente correta.

Selecioneu 4 respostas corretas.

A resposta correcta é: Dados particionados → Comunicação por mensagens, A coordenação é feita através das variáveis partilhadas → Modelo de memória partilhada, A aplicação é uma coleção de processos → Comunicação por mensagens, A aplicação é uma coleção de *threads* → Modelo de memória partilhada, Diversos processos em diversas máquinas → Comunicação por mensagens

A Lei de Moore, de 1965, previa que:

- ☐ a. O número de computadores desktop por empresa dobraria a cada 18 a 24 meses
- ☐ b. O número de circuitos integrados por transistor dobraria a cada 18 a 24 meses
- ☒ c. O número de transistores por circuito integrado dobraria a cada 18 a 24 meses
- ☐ d. O número de fabricantes de computadores desktop dobraria a cada 18 a 24 meses
- ☐ e. O tamanho dos computadores diminuiria pela metade a cada 18 a 24 meses



A sua resposta está correta.

A resposta correta é:

O número de transistores por circuito integrado dobraria a cada 18 a 24 meses

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Considerando o código a seguir que corre sem erros, assinale o output mais provável:

```

def multBlock(pars):
    M = pars[0]
    blockSize = int(pars[1])
    block = int(pars[2])
    V = pars[3]
    R = pars[4]
    for i in range(blockSize): # número de linhas da matriz passadas para esta iteração
        temp = 0
        for j in range(len(V)): # número de elementos no vetor
            temp += M[i,j]*V[j]
        R[i+(block*blockSize)] = temp

if __name__ == '__main__':
    rnd = np.random
    n = 3000
    vetor = rnd.uniform(0,100,size=n)
    matriz = rnd.uniform(0,100,size=(n,n))
    resultado = np.zeros(n)
    cpu_cores = 4
    bSize = n/cpu_cores # para simplificar vamos assumir que a divisão é exata...
    pool = Pool(processes=4)
    starttime = timeit.default_timer()
    for i in range(cpu_cores): # número de linhas da matriz
        params = [matriz,bSize,i,vetor,resultado]
        pool.apply_async(multBlock, [params])
    pool.close()
    pool.join()
    print(f"Tempo multi-processos: {timeit.default_timer() - starttime}")
    print(f"Primeiro elemento do vetor resultado: {resultado[0]}")
    starttime = timeit.default_timer()
    for i in range(len(matriz)): # número de linhas da matriz
        temp = 0
        for j in range(len(vetor)): # número de elementos no vetor
            temp += matriz[i,j]*vetor[j]
        resultado[i] = temp
    print(f"Tempo sequencial: {timeit.default_timer() - starttime}")
    print(f"Primeiro elemento do vetor resultado: {resultado[0]}")

```



```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

- ☐ a. Tempo multi-processos: 1.7997834  
Primeiro elemento do vetor resultado: 0.0  
Tempo sequencial: 4.212884799999999  
Primeiro elemento do vetor resultado: 0.0
- ☐ b. Tempo multi-processos: 1.7997834  
Primeiro elemento do vetor resultado: 7462941.620096451  
Tempo sequencial: 4.212884799999999  
Primeiro elemento do vetor resultado: 7462941.620096451
- ☒ c. Tempo multi-processos: 1.7997834  
Primeiro elemento do vetor resultado: 0.0  
Tempo sequencial: 4.212884799999999  
Primeiro elemento do vetor resultado: 7462941.620096451
- ☐ d. Tempo multi-processos: 1.7997834  
Primeiro elemento do vetor resultado: 7462941.620096451  
Tempo sequencial: 4.212884799999999  
Primeiro elemento do vetor resultado: 0.0
- ☐ e. Tempo multi-processos: 4.212884799999999  
Primeiro elemento do vetor resultado: 7462941.620096451  
Tempo sequencial: 1.7997834  
Primeiro elemento do vetor resultado: 7462941.620096451

Como se pode correr código concorrente na mesma máquina usando Python? Assinale todas as alternativas que se aplicarem.

- ☐ a. Lançando vários interpretadores Python
- ☒ b. Utilizando a biblioteca *multiprocessing*
- ☒ c. Só é possível se houverem pelo menos 2 cores de processamento
- ☐ d. Não é possível correr código concorrentemente com programação Python
- ☒ e. Utilizando a biblioteca *threading*

A sua resposta está parcialmente correta.

Selecionou 2 respostas corretas.

Respostas corretas:

Utilizando a biblioteca *threading*,

Utilizando a biblioteca *multiprocessing*,

Lançando vários interpretadores Python

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Assinale a alternativa incorreta:

- ☐ a. O GIL (*Global Interpreter Lock*) do Python protege contra condições de corrida em *threads* e em processos.
- ☒ b. O Python protege contra condições de corrida na referência aos objetos em memória.
- ☐ c. A gestão de memória é feita no Python através da contagem de referências.
- ☐ d. Qualquer execução Python demanda a aquisição do bloqueio do interpretador.
- ☐ e. O Python liberta a memória associada a um objeto não referenciado.

A sua resposta está incorreta.

A resposta correta é:

O GIL (*Global Interpreter Lock*) do Python protege contra condições de corrida em *threads* e em processos.

Na multiplicação da matriz **A** pelo vetor **B**, usando programação sequencial iterativa, o código abaixo é ineficiente por quê?

```

18
19 for i in range(len(A)):
20     resultado[i] = 0
21     for j in range(len(B)):
22         resultado[i] += A[i,j]*B[j]
23

```

- ☐ a. Porque as variáveis *i* e *j* não foram declaradas dentro dos *for*
- ☒ b. Porque precisa calcular *len(A)* e *len(B)* a cada iteração
- ☐ c. Porque falta um ciclo *for*, tem de ser acrescentado mais tarde
- ☐ d. Porque ele tem de ir buscar o valor de *resultado[i]* em cada iteração e é menos eficiente que usar uma variável para acumular.
- ☐ e. Porque está a multiplicar linha-coluna, quando o mais eficiente é coluna-linha

A sua resposta está incorreta.

A resposta correta é:

Porque ele tem de ir buscar o valor de *resultado[i]* em cada iteração e é menos eficiente que usar uma variável para acumular.

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

6  from threading import Thread
7  from PIL import Image, ImageFilter, ImageEnhance
8  import os, queue
9  from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## 2º TESTE

SOAP REST , mecanismos de comunicação , coordenação ou cooperação

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 1

Não respondida

Nota: 1,0

Faça a correspondência adequada:

Requer XML	Escolha...
Não requer XML	Escolha...
Requer HTTP	Escolha...

A resposta correcta é: Requer XML → SOAP,

Não requer XML → REST,

Requer HTTP → REST

Pergunta 2

Não respondida

Nota: 1,0

Assinale as duas alternativas incorretas. Um conector provê mecanismos de comunicação, coordenação ou cooperação para:

- ☐ a. Debugging dos componentes
- ☐ b. Streaming de dados entre processos
- ☐ c. Passagem de mensagens entre processos
- ☐ d. Chamada de procedimentos remotos
- ☐ e. Partilha de variáveis

Respostas corretas:

Debugging dos componentes,

Partilha de variáveis

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

tabelas centralizadas , nomeação plana



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for arquivo in os.listdir(dir):
```

Pergunta 3

Não respondida

Nota: 1,0

Podemos usar tabelas centralizadas para associar nomes a endereços em sistemas distribuídos de pequena dimensão.

Selecione uma opção:

- ☐ Verdadeiro
- ☐ Falso

A resposta correta é: Verdadeiro

Pergunta 4

Não respondida

Nota: 1,0

Não são soluções para a nomeação plana:

- ☐ a. Broadcast
- ☐ b. Multicast
- ☐ c. Tabelas de dispersão distribuídas
- ☐ d. Servidores DNS
- ☐ e. Resolução iterativa

Respostas corretas:

Servidores DNS,

Resolução iterativa

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

nomeação estruturada

Pergunta 5

Não respondida

Nota: 1,0

Sobre nomeação estruturada, faça a correspondência adequada:

Onde estão os recursos propriamente ditos

Escolha...

Forma de subdividir um nome em sub-nomes

Escolha...

Nome de mais alto nível

Escolha...

A resposta correcta é:

Onde estão os recursos propriamente ditos → Folhas,

Forma de subdividir um nome em sub-nomes → Diretórios,

Nome de mais alto nível → Raiz

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

protocolo HTTP , endereço físico ou endereço MAC

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta **6**

Não respondida

Nota: 1,0

Há quantos anos foi lançado o protocolo HTTP?

- ☐ a. 30
- ☐ b. 20
- ☐ c. 6
- ☐ d. 40
- ☐ e. 10

A resposta correta é:

30

Pergunta **7**

Não respondida

Nota: 1,0

O endereço físico de destino de cada pacote, ou endereço MAC, muda em cada trecho do caminho percorrido por esse pacote.

Selecione uma opção:

- ☐ Verdadeiro
- ☐ Falso

A resposta correta é: Verdadeiro

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Escalabilidade , Containers , Virtualização, Load balancers , encapsulamento

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for arquivo in os.listdir(dir):
```

Pergunta 8

Não respondida

Nota: 1,0

Faça a correspondência adequada:

Aumento do número de recursos físicos ou virtuais.

Aumento do poder de processamento, memória ou espaço de armazenamento de uma máquina.

É feita uma divisão de recursos ao nível do Sistema Operativo.

Particionamento dos recursos de um sistema físico em múltiplos recursos virtuais.

Distribuir os pedidos dos clientes pelos vários servidores para permitir a máxima utilização dos recursos físicos e virtuais.

Escolha...

Escolha...

Escolha...

Escolha...

Escolha...

A resposta correcta é:

Aumento do número de recursos físicos ou virtuais. → Escalabilidade horizontal,

Aumento do poder de processamento, memória ou espaço de armazenamento de uma máquina. → Escalabilidade vertical,

É feita uma divisão de recursos ao nível do Sistema Operativo. → Containers,

Particionamento dos recursos de um sistema físico em múltiplos recursos virtuais. → Virtualização,

Distribuir os pedidos dos clientes pelos vários servidores para permitir a máxima utilização dos recursos físicos e virtuais. → Load balancers

Pergunta 9

Não respondida

Nota: 1,0

Segundo o conceito de encapsulamento, cada protocolo conversa com o protocolo equivalente através da adição de campos de controle no pacote que vai ser transmitido.

Selecione uma opção:

☐ Verdadeiro

☐ Falso

A resposta correta é: Verdadeiro



```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

Wrapper , Adapter , pacote

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 10

Não respondida

Nota: 1,0

Como se chama o componente que faz a interface com um sistema legado? (marque 2 respostas corretas)

- ☐ a. Connector
- ☐ b. Adopter
- ☐ c. Wrapper
- ☐ d. Waffle
- ☐ e. Adapter

Respostas corretas:

Wrapper,

Adapter

Pergunta 11

Não respondida

Nota: 1,0

É impossível saber qual a rota que nosso pacote seguiu até seu destino.

Selecione uma opção:

- ☐ Verdadeiro
- ☐ Falso

A resposta correta é: Falso

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

IP

Pergunta **12**

Não respondida

Nota: 1,0

Qual o protocolo que evita que um pacote fique eternamente a circular na rede se não encontrar seu destino?

Selecione uma opção de resposta:

- ☐ a. HTTP
- ☐ b. Ethernet
- ☐ c. FTP
- ☐ d. TCP
- ☐ e. IP

A resposta correta é: IP

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

guerra fria (cold war), acessos redundantes , IP

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 13

Não respondida

Nota: 1,0

A guerra fria foi determinante na decisão de criar acessos redundantes entre computadores da agência de defesa americana.

Selecione uma opção:

- ☐ Verdadeiro
- ☐ Falso

A resposta correta é: Verdadeiro

Pergunta 14

Não respondida

Nota: 1,0

A máscara de rede indica os bits de um endereço IP versão 4 que designam uma rede. Os restantes designam cada hospedeiro nesta rede.

Suponha que a máscara diz que há 24 bits no endereço de rede de um computador cujo endereço IP é 192.168.10.20.

Portanto, seu endereço de rede e seu endereço IP são:

Selecione uma opção de resposta:

- ☐ a. 192.168.10.255 e 255.255.255.20
- ☐ b. 192.168.10 e 20
- ☐ c. 192.168.10.0 e 192.168.10.20
- ☐ d. 192.160.0.0 e 10.20

A resposta correta é: 192.168.10.0 e 192.168.10.20

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

GET , PATCH , PUT , POST

Pergunta **15**

Não respondida

Nota: 1,0

Faça a correspondência adequada:

Solicita um recurso passando uma URL

Escolha...

Permite aplicar alterações parciais a recursos no servidor

Escolha...

Permite adicionar ou modificar recursos no servidor

Escolha...

Envia um conteúdo ao servidor

Escolha...

A resposta correcta é:

Solicita um recurso passando uma URL → GET,

Permite aplicar alterações parciais a recursos no servidor → PATCH,

Permite adicionar ou modificar recursos no servidor → PUT,

Envia um conteúdo ao servidor → POST

SOAP , REST , AJAX



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta **16**  
Não respondida  
Nota: 1,0

Faça a correspondência adequada:

Protocolo de mensagens que usa XML para trocar informação com um servidor

Escolha...

Arquitectura para interoperabilidade entre computadores na Internet

Escolha...

Permite aos browsers obterem informação de um servidor de forma assíncrona sem reler a página toda novamente

Escolha...

A resposta correcta é:

Protocolo de mensagens que usa XML para trocar informação com um servidor → SOAP,

Arquitectura para interoperabilidade entre computadores na Internet → REST,

Permite aos browsers obterem informação de um servidor de forma assíncrona sem reler a página toda novamente → AJAX

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Nomeação estruturada , baseada em atributos , plana

Pergunta 17

Não respondida

Nota: 1,0

Sobre Naming, faça a correspondência adequada:

O espaço de nomes pode agrupar os vários componentes do nome num formato tipo árvore

Escolha...

Os identificadores tendem a ser compostos por nomes simples e legíveis, organizados num "espaço de nomes" (*namespace*).

Escolha...

Usa diretórios de serviços

Escolha...

Pode-se indexar os nomes dos recursos por Serviços que oferecem e Capacidades que possuam

Escolha...

Sistemas onde identificadores são nomes não-estruturados

Escolha...

Não contém informação nenhuma sobre como acessar fisicamente o recurso

Escolha...

A resposta correcta é:

O espaço de nomes pode agrupar os vários componentes do nome num formato tipo árvore → Nomeação estruturada,

Os identificadores tendem a ser compostos por nomes simples e legíveis, organizados num "espaço de nomes" (*namespace*). → Nomeação estruturada,

Usa diretórios de serviços → Nomeação baseada em atributos,

Pode-se indexar os nomes dos recursos por Serviços que oferecem e Capacidades que possuam → Nomeação baseada em atributos,

Sistemas onde identificadores são nomes não-estruturados → Nomeação plana,

Não contém informação nenhuma sobre como acessar fisicamente o recurso → Nomeação plana

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

socket , Hipermedia , Hipertexto

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for arquivo in os.listdir(dir):
```

Pergunta **18**

Não respondida

Nota: 1,0

A identificação de um socket envolve:

Selecione uma opção de resposta:

- ☐ a. O protocolo de rede, o endereço IP e o número da porta associado à aplicação
- ☐ b. O protocolo de rede e o endereço IP
- ☐ c. O protocolo de transporte, o endereço IP e o número da porta associado à aplicação
- ☐ d. O protocolo de transporte e o endereço IP

A resposta correta é: O protocolo de transporte, o endereço IP e o número da porta associado à aplicação

Pergunta **19**

Não respondida

Nota: 1,0

Faça a correspondência adequada:

Permite ligações a outras páginas e a elementos gráficos, filmes, áudio, etc.

Escolha...

Somente permite ligações a outras páginas

Escolha...

A resposta correcta é:

Permite ligações a outras páginas e a elementos gráficos, filmes, áudio, etc. → Hipermédia,

Somente permite ligações a outras páginas → Hipertexto

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

estilo de arquitetura

Pergunta 20

Não respondida

Nota: 1,0

Um estilo de arquitetura para sistemas distribuídos é formulado em que termos? Assinale a alternativa incorreta.

- ☐ a. Como cada componente conecta-se aos outros
- ☐ b. Quais linguagens de programação são utilizadas
- ☐ c. Componentes reutilizáveis com interfaces bem definidas
- ☐ d. Dados trocados entre os componentes
- ☐ e. Como componentes e conectores formam um sistema

A resposta correta é:

Quais linguagens de programação são utilizadas

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Recursos Web , Web service , Aplicação

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 21

Não respondida

Nota: 1,0

Recursos na web são somente websites, documentos e ficheiros.

Selecione uma opção:

- ☐ Verdadeiro
- ☐ Falso

A resposta correta é: Falso

Pergunta 22

Não respondida

Nota: 1,0

Faça a correspondência adequada:

Solicita uma operação	Escolha...
Oferece operações	Escolha...
Efetua o processamento	Escolha...
Envia a resposta	Escolha...

A resposta correcta é:

Solicita uma operação → Aplicação,

Oferece operações → Web service,

Efetua o processamento → Web service,

Envia a resposta → Web service



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

trocar mensagens via UDP

Pergunta 23

Não respondida

Nota: 1,0

Antes de trocar mensagens via UDP, é necessário o estabelecimento de uma sessão subjacente.

Selecione uma opção:

- ☐ Verdadeiro
- ☐ Falso

A resposta correta é: Falso

Elasticidade , Serviço mensurável , Serviço on-demand , Agrupamento de recursos , Acesso amplo à rede

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 24

Não respondida

Nota: 1,0

Faça a correspondência adequada:

Os recursos podem ser atribuídos e libertados de uma forma rápida, dando a sensação de serem ilimitados.

Escolha...

Os utilizadores usam serviços na *cloud* e são cobrados usando uma métrica específica.

Escolha...

Os recursos na *cloud* podem ser provisionados à medida das necessidades dos utilizadores.

Escolha...

Os recursos disponibilizados pelos *providers* são disponibilizados usando recursos físicos e virtuais atribuídos dinamicamente conforme as necessidades.

Escolha...

Os recursos na *cloud* são acessíveis usando métodos standard de acesso às redes, por vários dispositivos diferentes.

Escolha...

A resposta correcta é:

Os recursos podem ser atribuídos e libertados de uma forma rápida, dando a sensação de serem ilimitados. → Elasticidade,

Os utilizadores usam serviços na *cloud* e são cobrados usando uma métrica específica. → Serviço mensurável,

Os recursos na *cloud* podem ser provisionados à medida das necessidades dos utilizadores. → Serviço on-demand,

Os recursos disponibilizados pelos *providers* são disponibilizados usando recursos físicos e virtuais atribuídos dinamicamente conforme as necessidades. → Agrupamento de recursos,

Os recursos na *cloud* são acessíveis usando métodos standard de acesso às redes, por vários dispositivos diferentes. → Acesso amplo à rede

processos diferentes uma máquina

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 25

Não respondida

Nota: 1,0

Teoricamente, quantos processos diferentes uma máquina pode ter que estejam a comunicar em rede ao mesmo tempo?

Selecione uma opção de resposta:

- ☐ a. 1024
- ☐ b. 65.536, pois o número de porta usa 16 bits
- ☐ c. 65.536, pois o número de porta usa 32 bits
- ☐ d. 65.536 para cada protocolo de transporte

A resposta correta é: 65.536 para cada protocolo de transporte

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Flooding , Random walk

Pergunta 26

Não respondida

Nota: 1,0

Relativamente a redes peer-to-peer não estruturadas, faça a correspondência adequada:

Mais rápido a encontrar o resultado

Escolha...

O nó passa a solicitação para um vizinho aleatoriamente escolhido

Escolha...

O nó passa a solicitação para todos os vizinhos

Escolha...

Mais eficiente na comunicação

Escolha...

A resposta correcta é:

Mais rápido a encontrar o resultado → Flooding,

O nó passa a solicitação para um vizinho aleatoriamente escolhido → Random walk,

O nó passa a solicitação para todos os vizinhos → Flooding,

Mais eficiente na comunicação → Random walk

browsers da Internet

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

Pergunta 27

Não respondida

Nota: 1,0

Marque a alternativa INCORRETA. Antes de surgirem os browsers da Internet, esta era utilizada principalmente para:

Selecione uma opção de resposta:

- ☐ a. Email
- ☐ b. Ligações remotas à linha de comando
- ☐ c. Jogos
- ☐ d. Troca de ficheiros

A resposta correta é: Jogos

## Arquiteturas e Modelos de Sistemas Distribuídos

### Traditional three-layered view

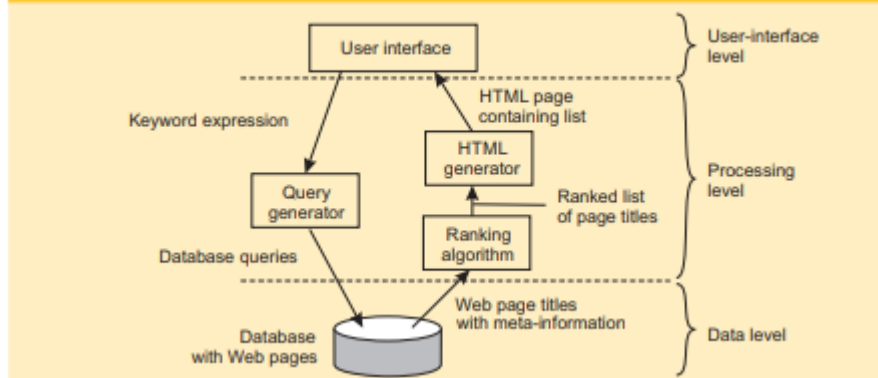
- **Application-interface layer** contains units for interfacing to users or external applications
- **Processing layer** contains the functions of an application, i.e., without specific data
- **Data layer** contains the data that a client wants to manipulate through the application components

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

### Example: a simple search engine



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

REST

## RESTful architectures

### Essence

View a distributed system as a collection of resources, individually managed by components. Resources may be added, removed, retrieved, and modified by (remote) applications.

- 1 Resources are identified through a single naming scheme
- 2 All services offer the same interface
- 3 Messages sent to or from a service are fully self-described
- 4 After executing an operation at a service, that component forgets everything about the caller

### Basic operations

Operation	Description
PUT	Create a new resource
GET	Retrieve the state of a resource in some representation
DELETE	Delete a resource
POST	Modify a resource by transferring a new state

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## On interfaces

### Simplifications

Assume an interface `bucket` offering an operation `create`, requiring an input string such as `mybucket`, for creating a bucket "mybucket."

### SOAP

```
import bucket
bucket.create("mybucket")
```

### RESTful

```
PUT "http://mybucket.s3.amazonaws.com/"
```



```

6  from threading import Thread
7  from PIL import Image, ImageFilter, ImageEnhance
8  import os, queue
9  from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

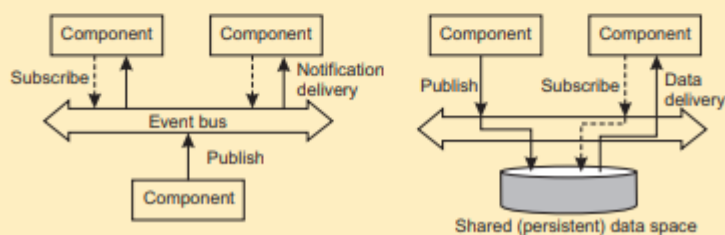
## Coordenação

### Coordination

#### Temporal and referential coupling

	Temporally coupled	Temporally decoupled
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

#### Event-based and Shared data space



```

6  from threading import Thread
7  from PIL import Image, ImageFilter, ImageEnhance
8  import os, queue
9  from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

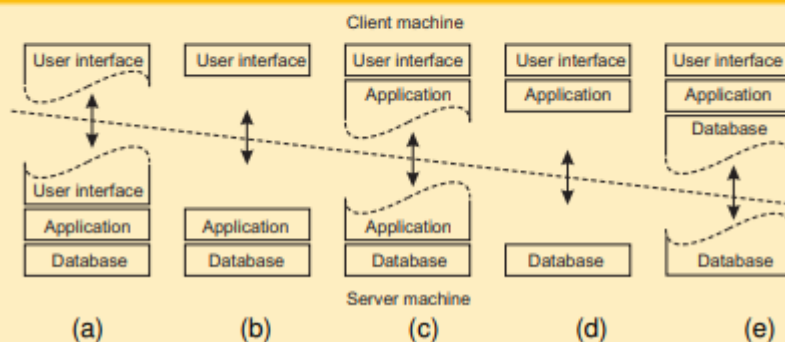
## Multi-tiered

### Multi-tiered centralized system architectures

#### Some traditional organizations

- **Single-tiered:** dumb terminal/mainframe configuration
- **Two-tiered:** client/single server configuration
- **Three-tiered:** each layer on separate machine

#### Traditional two-tiered configurations



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Prática

### Pipelining(lab5)

O ganho de tempo no pipelining existe pois o processamento é feito por “trabalhador” e não por processo, dando assim oportunidade de aplicar vários filtros com maior rapidez.

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

## Pipelining

Processamento em *pipelining* é semelhante a uma linha de montagem: um *worker* (*thread* ou processo) aplica alguma manipulação aos dados e passa-os para o próximo *worker*. Assim que passar os dados manipulados, carrega nova leva de dados e assim todas as etapas das pipeline trabalham concorrentemente.

Uma *pipeline* é composta de uma série de elementos de processamento encadeados, onde o output de um elemento é o input do próximo. A Figura 1 ilustra o conceito:

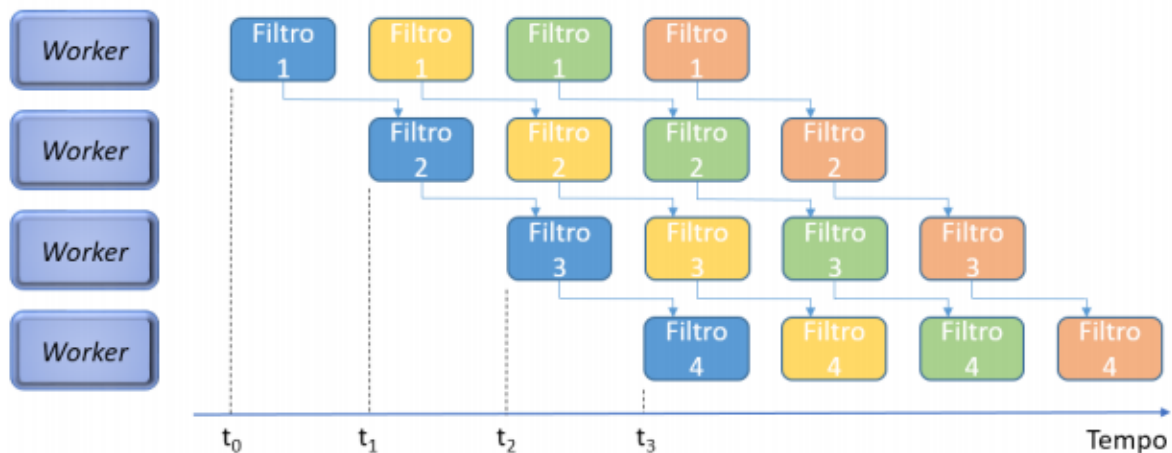


Figura 1 – Processamento em Pipelining

O processamento em *pipelining* procura evitar que unidades de processamento estejam desocupadas enquanto houver trabalho a realizar.

No Python, se usarmos *threads* será sempre melhor em circunstâncias de processamento *fine-grained*, onde a atividade de IO é significativa. De modo contrário, será melhor utilizar processos com algum mecanismo de passagem de dados entre eles (por exemplo o Manager visto na prática anterior).

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Processamento de Imagens no Python

Há diversos *frameworks* ou bibliotecas *open-source* disponíveis para processamento de imagens com Python:

- scikit-image
- NumPy (uma imagem é uma matriz...)
- SciPy
- PIL/Pillow
- OpenCV
- SimpleCV
- Mahotas
- SimpleITK
- pgmagik
- Pycairo

Nesta prática vamos utilizar o PIL/Pillow, que permite manipular muitos tipos de imagens. Inicialmente é necessário instalar o módulo:

```
pip install pillow
```

E de seguida podemos usar as funcionalidades no nosso programa com a importação dos módulos desejados. A documentação completa pode ser encontrada neste [link](#). No nosso caso usaremos 3 módulos:

- **Image**: fornece uma classe com o mesmo nome para representar uma imagem PIL. Tem também várias funções para carregar imagens e criar novas imagens.
- **ImageFilter**: contém definições para um conjunto pré-definido de filtros, que podem ser utilizados com o método `Image.filter()`:
  - BLUR
  - CONTOUR
  - DETAIL
  - EDGE\_ENHANCE
  - EDGE\_ENHANCE\_MORE
  - EMBOSS
  - FIND\_EDGES
  - SHARPEN
  - SMOOTH
  - SMOOTH\_MORE

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
34         # Read image
35         imagem = Image.open(dir + "/" + str(file))
36         # Display image
37         imagem.show()
38         queue_inicial.put(imagem)
39     trabalhador_1 = Thread(target=trabalhador, args=('cinzentos', queue_inicial, queue_12, num_trabalhos))
40     trabalhador_1.start()
41     trabalhador_2 = Thread(target=trabalhador, args=('contraste', queue_12, queue_123, num_trabalhos))
42     trabalhador_2.start()
43     trabalhador_3 = Thread(target=trabalhador, args=('arestas', queue_123, queue_1234, num_trabalhos))
44     trabalhador_3.start()
45     trabalhador_4 = Thread(target=trabalhador, args=('arestas', queue_1234, queue_final, num_trabalhos))
46     trabalhador_4.start()

```



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

```
ine2ThreadsQueue.py x pipeline4ThreadsQueue.py x processImagensSequencial2.py x pro
# Computação Paralela e Distribuída 2020/2021 - Lab 01
# Aluno1: (Gonçalo Fernandes Costa) (180221083) (2ºL_EI-SW-01)
# Aluno2: (Carlos Oliveira)(190221084)(2ºL_EI-SW-01)
#####
import checkSys as check
from PIL import Image, ImageFilter, ImageEnhance
import os
from timeit import default_timer as timer

def cinzentos(imagem):
    enh = ImageEnhance.Color(imagem)
    imagem = enh.enhance(0.0)
    return imagem

def contraste(imagem):
    enh = ImageEnhance.Contrast(imagem)
    imagem = enh.enhance(1.8)
    return imagem

if __name__ == '__main__':
    print(check.get_processor_info())
    tic = timer()
    dir = "./imagens" # As imagens devem estar na pasta imagens
    for file in os.listdir(dir):
        im = Image.open(dir + "/" + str(file))
        im.show()
        im = cinzentos(im)
        im = contraste(im)
        im.show("B&W e 30% mais contraste")
        tac = timer()
    print(f"Tempo para tratamento sequencial: {tac - tic:21.8f}")
```



```
pipeline2ThreadsQueue.py x pipeline4ThreadsQueue.py x processaImagensSequencial2.py x processaImagensSequencial4.py x
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34         img = Image.open(dir + "/" + str(file))
35         img.show()
36         img = cinzentos(img)
37         img = contraste(img)
38         img = arestas(img)
39         img = esbacamento(img)
40         img.show("B&W , 30% mais contraste, Arestas e Esbacamento")
41         tac = timer()
42         print(f"Tempo para tratamento sequencial: {tac - tic:21.8f}")
```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

## Threads POSIX(lab6)

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Pré-requisitos:

O aluno deve ter o Linux com o ambiente de compilação C instalado, ou o Windows com o MinGW. Entretanto, **não tentem compilar os códigos com PThreads em Windows** – não sendo bibliotecas nativas, a sua instalação e configuração é demasiado complicada.

Num terminal do Linux ou numa linha de comando do Windows deve ser capaz de executar o comando a seguir com resultado semelhante:

```
L:\cpd\c>gcc --version
gcc (MinGW.org GCC-6.3.0-1) 6.3.0
```

Caso o GCC não seja de versão 4.4 ou superior, deverá ser feita a atualização – no Windows através da atualização do MinGW.

Para este exercício, sugerimos um editor de texto tal como o Notepad++ para editar os códigos e o terminal em Linux ou linha de comandos em Windows para o executar.

## PThreads

Os Threads POSIX são referidos como **pthreads**. Trata-se de um modelo de execução paralela independente da linguagem. A criação e controlo dos *threads* dá-se através de chamadas à API respetiva POSIX. Os procedimentos têm todos o prefixo **pthread\_** e dividem-se em quatro grupos:

- Gestão de *threads* (criação, *join*, etc.)
- Mecanismos de sincronização tipo *mutexes*
- Variáveis de condição
- Outros mecanismos de sincronização tais como *locks* de leitura e escrita e barreiras

Na página a seguir mostramos um código usando PThreads.

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function( void *i);
int main(){
    printf("Hello World!\n");
    pthread_t thread[4];
    for (int i = 0; i < 4; i++) {
        int *arg = malloc(sizeof(*arg));
        *arg = i;
        pthread_create( &thread[i], NULL, print_message_function, arg);
    }
    for (int i = 0; i < 4; i++) {
        pthread_join( thread[i], NULL);
    }
    printf("Viva Portugal\n");
    return 0;
}
void *print_message_function( void *i ){
    int a = *((int *) i);
    free(i);
    printf("Thread %d\n",a);
}

```

Para compilar este código é necessário usar a opção *-lpthread*. Abaixo mostramos o resultado da compilação e execução no Linux (não tentar compilar no Windows...):

```

aluno@cad01:~/pthreads$ gcc exemploPThreads.c -o ePT -lpthread
aluno@cad01:~/pthreads$ ./ePT
Hello World!
Thread 0
Thread 2
Thread 1
Thread 3
Viva Portugal

```

PThreads permitem criar código portátil mas considera-se que é pesado e lento face a alternativas como o OpenMP.

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## OpenMP(lab6)

### OpenMP (Open Multiprocessing) – [www.openmp.org](http://www.openmp.org)

OpenMP é uma especificação aberta para processamento paralelo, uma API de alto nível para definir programas *multi-threaded* em memória compartilhada.

O padrão OpenMP permite:

- Que o programador possa separar o programa em regiões sequenciais e regiões paralelas
- Confiar na gestão automática das necessidades de memória
- Usar mecanismos de sincronização já disponíveis

O padrão OpenMP NÃO permite:

- Paralelismo automático
- Garantias de melhor desempenho
- Evitar por completo inconsistências no acesso compartilhado aos dados

O OpenMP é completamente suportado para C e C++ desde o GCC 6.

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

A seguir está um código equivalente ao anterior mas utilizando OpenMP:

```
#include <stdio.h>
#include <omp.h>
int main() {
    omp_set_num_threads(4);
    int i = 0;
    printf("Hello World!\n");
    // faça em paralelo
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        printf("Thread %d\n", tid);
    }
    printf("Viva Portugal\n");
    return 0;
}
```

A figura abaixo demonstra compilação e execução em ambiente Windows:

```
L:\cpd\c>gcc exemploOpenMP.c -o eOMP.exe -fopenmp
L:\cpd\c>eOMP
Hello World!
Thread 1
Thread 2
Thread 0
Thread 3
Viva Portugal
```



```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

## Prática: Calcular o pi (Tim Mattson)

O código a seguir calcula numericamente a integral  $\int_0^1 \frac{4}{1+x^2}$ , cujo resultado é o número pi:

```

#include <stdio.h>
#include <time.h>
static long num_steps = 100000000;
double step;
int main()
{
    int i;
    double x, pi, sum = 0.0;
    clock_t tic = clock();

    step = 1.0 / (double)num_steps;
    for (i = 1; i <= num_steps; i++)
    {
        sum = 0.0;
        {
            for (i = 1; i <= num_steps; i++)
            {
                x = (i - 0.5) * step;
                sum = sum + 4.0 / (1.0 + x * x);
            }
        }
        pi = step * sum;
        clock_t tac = clock();
        double elapsed = (double)(tac-tic)/CLOCKS_PER_SEC;
        printf("\n pi vale %f e o tempo foi: %f\n", pi, elapsed);
    }
}

```

Copie o código com o nome de **pi\_loopSeq.c** e inclua o cabeçalho da prática (atenção que é linguagem C, os comentários são diferentes do Python). Compile-o e execute. Aponte o resultado.



```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

Crie o programa **pi\_loopOMP.c** copiando o código a seguir e inserindo o cabeçalho da prática:

```

#include <stdio.h>
#include <omp.h>
static long num_steps = 1000000000;
double step;
int main()
{
    int i, j;
    double x, pi, sum = 0.0;
    double start_time, run_time;

    step = 1.0 / (double)num_steps;
    for (j = 1; j <= 4; j++) // lança execuções com 1, 2, 3 e 4 threads
    {
        sum = 0.0;
        omp_set_num_threads(j);
        start_time = omp_get_wtime();
        #pragma omp parallel default(none) private(i, x) \
            shared(j, num_steps, step, sum) \
            num_threads(j)
        {
            int tid = omp_get_thread_num();
            #pragma omp single
            printf(" num_threads = %d", omp_get_num_threads());

            double somaParcial = 0.0;
            for (i = tid*(num_steps/j)+1; i <= (tid+1)*(num_steps/j); i++)
            {
                x = (i - 0.5) * step;
                somaParcial = somaParcial + 4.0 / (1.0 + x * x);
            }
            #pragma omp barrier
            sum += somaParcial;
        }
        pi = step * sum;
        run_time = omp_get_wtime() - start_time;
        printf("\n pi is %.15f in %f seconds and %d threads\n", pi, run_time, j);
    }
}

```

Compile-o com o comando

```
gcc pi_loopOMP.c -o pi_loopOMP -fopenmp
```

Execute-o e veja que utiliza funções da API OpenMP para obter o tempo. Aponte os resultados.

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Sockets(lab7)

### Parte I – Introdução aos Sockets

Nesta primeira parte da ficha iremos implementar um pequeno serviço de rede cujo objetivo é fornecer o cálculo de operações matemáticas. O trabalho consiste na implementação de um pequeno servidor em Python que será responsável por fornecer um serviço e um programa cliente que irá consumir o serviço. Exemplo de utilização na aplicação cliente:

```
> 1+1
2
```

A comunicação entre o servidor e os clientes é feita recorrendo a *sockets* TCP/IP. Um *socket* de rede é um ponto de comunicação entre dois programas a correr sobre uma rede de computadores, e pode ser considerado como o acesso mais *low-level* que um programador tem ao subsistema de rede. Para construirmos o servidor e o cliente do serviço, teremos de definir como iremos utilizar os sockets.

**Servidor:** O servidor terá inicialmente de definir um **endereço IP** e uma **porta** onde irá estar à escuta de comunicações. Em termos gerais, o servidor tratará de:

1. Criar um *socket* do tipo TCP/IP que lidará com endereços IPv4 (do tipo 127.0.0.1).
2. Associar o *socket* a um par (endereço, porto) usando o comando *bind()*.
3. Ficar à escuta por ligações usando o comando *listen()*.

```

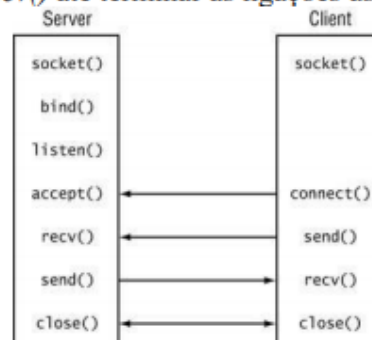
6  from threading import Thread
7  from PIL import Image, ImageFilter, ImageEnhance
8  import os, queue
9  from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

**Cliente:** O cliente apenas terá de se ligar ao servidor (sabendo qual o endereço IP e a porta) e enviar mensagens:

1. Criar um socket do tipo TCP/IP semelhante ao do servidor.
2. Ligar-se ao servidor usando o comando *connect()*.

Após o cliente se ligar ao servidor usando o comando *connect()* o servidor terá de aceitar a ligação usando o comando *accept()*. As comunicações propriamente ditas serão feitas usando os comandos *send()* (ou *sendall*) e *recv()* até terminar as ligações usando o comando *close()*.



Este laboratório tem por base o ficheiro **7-1-sockets-base.zip** que contém uma implementação inicial do servidor (*server.py*) e do cliente (*client.py*).

Crie um projeto no seu IDE com estes dois ficheiros, e execute cada um deles em separado (primeiro o servidor e depois o cliente). Leia o código de ambos os ficheiros, tente compreender como funcionam e verifique que o servidor faz o *print* da mensagem enviada pelo cliente.

```
pipeline2ThreadsQueue.py x pipeline4ThreadsQueue.py x processaImagensSequencial2.py x processaImagensSequencial4.py x
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34         pass

Project
Lab07 C:\Users\gonca\PycharmProjects\Lab07
  7-1-sockets-final
    checkSys.py
    client.py
    server.py
  7-2-chat-final
    checkSys.py
    client.py
    server.py
  lab7.pdf
  lab7.zip
External Libraries
Scratches and Consoles
client.py x server.py x
1 """
2 Implements a simple socket client
3
4 """
5 # Computação Paralela e Distribuída 2020/2021 - Lab 01
6 # Aluno1: (Gonçalo Fernandes Costa) (180221083) (2ºL_EI-SW-01)
7 # Aluno2: (Carlos Oliveira) (190221084) (2ºL_EI-SW-01)
8 #####
9 import ...
11 print(check.get_processor_info())
12
13 # Define socket host and port
14 SERVER_HOST = '127.0.0.1'
15 SERVER_PORT = 8000
16
17 # Create socket
18 msg = ''
19 while msg != 'exit':
20     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21
22     # Connect to server
23     client_socket.connect((SERVER_HOST, SERVER_PORT))
24
25     # Send message
26     client_socket.sendall("Hello from the client application!".encode())
27     msg = input("> ")
28     client_socket.sendall(msg.encode())
29     msgree = client_socket.recv(1024).decode()
30     print('<', msgree)
31     # Close socket
32
33     client_socket.close()
34
```

```
pipeline2ThreadsQueue.py x pipeline4ThreadsQueue.py x processaImagensSequencial2.py x processaImagensSequencial4.py x
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34         img = Image.open(os.path.join(dir, file))
35         queue_inicial.put(img)
36
37 Project
38 Lab07 C:\Users\gonca\PycharmProjects\Lab07
39 7-1-sockets-final
40 checkSys.py
41 client.py
42 server.py
43 7-2-chat-final
44 checkSys.py
45 client.py
46 server.py
47 lab7.pdf
48 lab7.zip
49 External Libraries
50 Scratches and Consoles
51 client.py x server.py x
52 """
53 # Computação Paralela e Distribuída 2020/2021 - Lab 01
54 # Aluno1: (Gonçalo Fernandes Costa) (180221083) (2º L_EI-SW-01)
55 # Aluno2: (Carlos Oliveira) (190221084) (2º L_EI-SW-01)
56 #####
57 import ...
58 print(check.get_processor_info())
59
60 # Define socket host and port
61 SERVER_HOST = '0.0.0.0'
62 SERVER_PORT = 8000
63
64 # Create socket
65 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
66 server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
67 server_socket.bind((SERVER_HOST, SERVER_PORT))
68 server_socket.listen(1)
69 print('Listening on port %s ...' % SERVER_PORT)
70
71 while True:
72     # Wait for client connections
73     client_connection, client_address = server_socket.accept()
74     # Print message from client
75     msg = client_connection.recv(1024).decode()
76     try:
77         client_connection.send(str(eval(msg)).encode())
78         if (msg == 'exit'):
79             break
80         print(eval(msg))
81     except NameError:
82         errorMessage = "Message not valid"
83         print(errorMessage)
84         client_connection.send(errorMessage.encode())
85     except:
86         print("Something else went wrong")
87
88     # Close client connection
89     client_connection.close()
90     # Close socket
91     server_socket.close()
92 """
```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

HTTP(lab8)



```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34
SERVER_HOST = '0.0.0.0'
SERVER_PORT = 8000

# Create socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((SERVER_HOST, SERVER_PORT))
server_socket.listen(1)
print('Listening on port %s ...' % SERVER_PORT)

def responseHTTP(request, response):
    #Split in lines of all the request
    s = request.splitlines()
    #Add's a breakspace to all lines
    for ele in s:
        ele += '<br>'
        response += ele
    client_connection.sendall(response.encode())

def handleRequest(request):
    # Get the first line of the request (Ex: GET /ipsum.html HTTP/1.1)
    fl = request.splitlines()
    # Split the content of first line (Ex: ['GET', '/ipsum.html', 'HTTP..'])
    nameFileLine = fl[0]
    nameFileSplit = nameFileLine.split()[1]
    nameFile = nameFileSplit.split('/')[1]
    content = ''
    print(nameFile)
    if(nameFile == ""):
        response = 'HTTP/1.0 200 OK\n\n'
        # Nivel 2
        # Open file
        f = open("htdocs\\index.html", "r")

```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```

else:
    content = 'HTTP/1.0 200 OK\n\n'
    # Open the file
    direct = f'hdocs\\{nameFile}'
    #print(direct)
    f = open(direct, "r")
    # Read contents
    for x in f:
        content += x
    # Close file
    f.close()
    # Return contents
    return content

```

```

while True:
    # Wait for client connections
    client_connection, client_address = server_socket.accept()

    # Handle client request
    request = client_connection.recv(1024).decode()
    content = handleRequest(request)

    # Send HTTP response
    #response = 'HTTP/1.0 200 OK\n\n<H1>Hello World</H1>\n\n'

    #Nivel 1
    #responseHTTP(request, response)

    #response = 'HTTP/1.0 200 OK\n\n'
    #Nivel 2
    # Open file

```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```

#Nivel 3
#response = handleRequest(request)

# Send HTTP response
#response = 'HTTP/1.0 200 OK\n\n'
#print(content)

client_connection.sendall(response.encode())

client_connection.close()

# Close socket
server_socket.close()

```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```

1 """
2     Implements a simple HTTP/1.0 Server
3 """
4 """
5 """
6 import socket
7 import time
8
9
10 def handle_request(request):
11     """Returns file content for client request"""
12
13     # Parse headers
14     print(request)
15     headers = request.split('\n')
16     get_content = headers[0].split()
17
18     # Get filename
19     filename = get_content[1]
20     if filename == '/':
21         filename = '/index.html'
22
23     try:
24         # Return file contents
25         if(filename.split(" ")[1] == ".jpg"):
26             f = open('htdocs' + filename, 'rb')
27             return f.read()
28         with open('htdocs' + filename) as fin:
29             return fin.read()
30     except FileNotFoundError:
31         return None

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):
34          #print("img")
35          response += f'Content-Length: {contentSize}\n\n'.encode()
36          response += f'Content-Type: {contentType}\n\n'.encode()
37          response += content
38          contentType = 'img/jpeg'
39      else:
40          response += content.encode()
41          contentType = 'text/html'
42      else:
43          response = 'HTTP/1.0 404 NOT FOUND\n\nFile Not Found'.encode()
44
45      print(str(contentSize))
46
47      #response += f'Content-Length: {contentSize}\n\n'.encode()
48      #response += ('Content-Type: ' + contentType + '\n\n').encode()
49      # Return encoded response
50      return response
51
52
53
54
55
56
57
58
59
60
61
62      # Define socket host and port
63      SERVER_HOST = '0.0.0.0'
64      SERVER_PORT = 8000
65
66      # Create socket
67      server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
68      server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
69      server_socket.bind((SERVER_HOST, SERVER_PORT))
70      server_socket.listen(1)
71      print('Listening on port %s ...' % SERVER_PORT)
72
73      while True:
74          # Wait for client connections
75          client_connection, client_address = server_socket.accept()
76

```

```
pipeline2ThreadsQueue.py x pipeline4ThreadsQueue.py x processaImagensSequencial2.py x processaImagensSequencial4.py x
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
34         # Return HTTP response
35         client_connection.sendall(response)
36         time.sleep(5)
37         client_connection.close()
38
39     # Close socket
40     server_socket.close()
41
```

```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Web Services(lab9)

```
"""
    Basic webservice application
"""

# Computação Paralela e Distribuída 2020/2021 - Lab 01
# Aluno1: (Gonçalo Fernandes Costa) (180221083) (2ºL_EI-SW-01)
# Aluno2: (Carlos Oliveira)(190221084)(2ºL_EI-SW-01)
#####

import json

USERS = [
    {
        "id": 1,
        "name": "Ana",
        "age": 22,
    },
    {
        "id": 2,
        "name": "Paulo",
        "age": 25,
    }
]
```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```

def controller(request):
    """Handles a request and returns a response."""

    # #Nivel 1
    # if (request['url']== '/users/') | (request['url']== '/users'):
    #     return {
    #         'status': '200 OK',
    #         'headers': {
    #             'Content-Type': 'application/json'
    #         },
    #         'body': json.dumps(USERS),
    #     }
    #
    # if (len(request['url'])>1):
    #     len(request['url'].split('/'))
    #     user_id= request['url'].split('/')[2]
    #
    #     if len(request['url'].split('/')) == 3:
    #         user= get_user_by_id(user_id)
    #
    #         if user != None:
    #             return {
    #                 'status': '200 OK',
    #                 'headers': {
    #                     'Content-Type': 'application/json'
    #                 },
    #                 'body': json.dumps( user),
    #             }
    #
    # return {
    #     'status': '404 Not Found',
    #     'headers': {},

```



```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34
35 if (request['url'] == '/users/') | (request['url'] == '/users'):
36     if request['method'] == 'GET':
37         return response_ok(USERS)
38
39     if request['method'] == 'POST':
40         # Handle all posts to users
41         newUser = json.loads(request['body']) # converts a str to a dict
42         USERS.append(newUser)
43         return response_ok(USERS)
44
45     else: return not_found('Method unavailable')
46
47 if (len(request['url']) > 1):
48     len(request['url'].split('/'))
49     user_id = request['url'].split('/')[2]
50
51 if len(request['url'].split('/')) == 3:
52     user = get_user_by_id(user_id)
53
54     #the userId does exists
55     if user != None:
56
57         if request['method'] == 'GET':
58             return response_ok(user)
59
60         elif request['method'] == 'DELETE':
61             USERS.remove(get_user_by_id(user_id))
62             return response_ok(USERS)
63
64         elif request['method'] == 'PUT':
65
66             userData = json.loads(request['body']) # converts a str to a dict
67
68             user['name'] = userData['name']

```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):

```

```

        else:
            return not_found('Method unavailable')

```

```

    else: return not_found('Not Found')

```

```

def get_user_by_id(user_id):

```

```

    for u in USERS:
        if u['id'] == int(user_id):
            return u
    return None

```

```

def response_ok(body):

```

```

    """Returns 200 OK"""
    return {
        'status': '200 OK',
        'headers': {
            'Content-Type': 'application/json'
        },
        'body': json.dumps(body)
    }

```

```

def not_found(body):

```

```

    """Returns 404 Not Found"""
    return {
        'status': '404 Not Found', 'headers': {},
        'body': body
    }

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      # colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      #colocar imagens na queue inicial
33      for file in os.listdir(dir):

```

```

14
15  @app.route("/api/users/<int:pk>/", methods=['GET', 'PUT', 'DELETE'])
16  def single_user(pk):
17      if request.method == 'DELETE':
18          userToDelete = db.get_user(pk)
19          if userToDelete:
20              db.remove_user(userToDelete)
21              return make_response(jsonify(), "200 OK")
22          else:
23              return make_response(jsonify(), "404 Not Found")
24
25      elif request.method == 'PUT':
26          userToUpdate = db.get_user(pk)
27          if userToUpdate:
28              data = request.get_json()
29              updateUser = db.update_user(userToUpdate, data)
30              return make_response(jsonify(updateUser))
31          else:
32              return make_response(jsonify(), 404)
33
34      elif request.method == 'GET':
35          singleUser = db.get_user(pk)
36          if singleUser == None:
37              return make_response(jsonify(), 404)
38          return make_response(jsonify(singleUser))
39
40
41  @app.route('/')
42  def index():
43      return app.send_static_file('index.html')

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      #colocar imagens na queue inicial
33      for file in os.listdir(dir):
34
41  @app.route('/')
42  def index():
43      return app.send_static_file('index.html')
44
45
46  @app.route("/api/users/", methods=['GET', 'POST'])
47  def all_users():
48      if request.method == 'GET':
49          # return all users
50          users = db.get_users()
51          return make_response(jsonify(users))
52      elif request.method == 'POST':
53          # add a user
54          userData = request.get_json()
55          # print(userData)
56          user = db.add_user(userData)
57          # print(user)
58          return make_response(jsonify(user), "201 Created")
59
60
61  db.recreate_db()
62  app.run(host='0.0.0.0', port=8000, debug=True)

```

```
pipeline2ThreadsQueue.py x pipeline4ThreadsQueue.py x processaImagensSequencial2.py x processaImagensSequencial4.py x
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34
app.py x script.js x
*.js files are supported by IntelliJ IDEA Ultimate
1 /**
2  * REST Client
3  *
4  */
5 ✓ /*
6  # Computação Paralela e Distribuída 2020/2021 - Lab 01
7  # Aluno1: (Gonçalo Fernandes Costa) (180221083) (2ºL_EI-SW-01)
8  # Aluno2: (Carlos Oliveira) (190221084) (2ºL_EI-SW-01)
9  #####
10 */
11 function getUsers() {
12     var req = new XMLHttpRequest();
13     req.open("GET", "/api/users/");
14     req.addEventListener("load", function() {
15         var users = JSON.parse(this.responseText);
16         var ul = document.getElementById('users');
17         var form = document.getElementById("form");
18         ul.innerHTML = '';
19         for (var i in users) {
20             var li = document.createElement('li');
21             li.innerHTML = users[i].name + ' (' + users[i].age + ')';
22             li.innerHTML += " <button onclick='updateUser(" + users[i].id + ")'>Update</button>";
23             li.innerHTML += " <button onclick='deleteUser(" + users[i].id + ")'>Delete</button>";
24             ul.appendChild(li);
25         }
26     });
27     req.send();
28 }
```

```

6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     #colocar imagens na queue inicial
33     for file in os.listdir(dir):
34
35
36 function addUser() {
37     var form = document.getElementById("form");
38     var name = form.name.value;
39     var age = form.age.value;
40     const userText = {"name" : name , "age" : age };
41
42     var req = new XMLHttpRequest();
43     req.open("POST", "/api/users/");
44     req.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
45     console.log(userText);
46     req.send(JSON.stringify(userText));
47     req.addEventListener("load", function() {
48         getUsers();
49     });
50     //console.log("Add: " + name + " - " + age);
51 }

```

```

6   from threading import Thread
7   from PIL import Image, ImageFilter, ImageEnhance
8   import os, queue
9   from timeit import default_timer as timer
10  def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11      i = 0
12      while (i < qtd_trabalhos):
13          imagem = input_queue.get()
14          if tarefa == 'cinzentos':
15              enh = ImageEnhance.Color(imagem)
16              imagem = enh.enhance(0.0)
17          elif tarefa == 'contraste':
18              enh = ImageEnhance.Contrast(imagem)
19              imagem = enh.enhance(1.8)
20          #imagem.show()
21          output_queue.put(imagem)
22          i = i + 1
23  if __name__ == '__main__':
24      print(check.get_processor_info())
25      tic = timer()
26      dir = "./imagens"
27      #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28      queue_inicial = queue.Queue()
29      queue_12 = queue.Queue()
30      queue_final = queue.Queue()
31      num_trabalhos = len(os.listdir(dir))
32      #colocar imagens na queue inicial
33      for file in os.listdir(dir):
34
47  function updateUser(id) {
48      var form = document.getElementById("form");
49      var name = form.name.value;
50      var age = form.age.value;
51
52      const userText = {"name" : name , "age" : age };
53
54      var req = new XMLHttpRequest();
55      req.open("PUT", "/api/users/" + id + "/");
56      req.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
57      console.log(userText);
58      req.send(JSON.stringify(userText));
59      req.addEventListener("load", function() {
60          getUsers();
61      });
62
63      console.log("Update: " + name + " " + age);
64  }
65
66  function deleteUser(id) {
67      var req = new XMLHttpRequest();
68      req.open("DELETE", "/api/users/" + id + "/");
69      req.addEventListener("load", function() {
70          getUsers();
71      });
72      req.send();
73      console.log("Delete: " + id)
74  }
75
76  getUsers();

```



```
6 from threading import Thread
7 from PIL import Image, ImageFilter, ImageEnhance
8 import os, queue
9 from timeit import default_timer as timer
10 def trabalhador(tarefa, input_queue, output_queue, qtd_trabalhos):
11     i = 0
12     while (i < qtd_trabalhos):
13         imagem = input_queue.get()
14         if tarefa == 'cinzentos':
15             enh = ImageEnhance.Color(imagem)
16             imagem = enh.enhance(0.0)
17         elif tarefa == 'contraste':
18             enh = ImageEnhance.Contrast(imagem)
19             imagem = enh.enhance(1.8)
20         #imagem.show()
21         output_queue.put(imagem)
22         i = i + 1
23 if __name__ == '__main__':
24     print(check.get_processor_info())
25     tic = timer()
26     dir = "./imagens"
27     #lista_tarefas = ['cinzentos', 'contraste', 'arestas', 'esbacamento']
28     queue_inicial = queue.Queue()
29     queue_12 = queue.Queue()
30     queue_final = queue.Queue()
31     num_trabalhos = len(os.listdir(dir))
32     # colocar imagens na queue inicial
33     for file in os.listdir(dir):
```

## Favicon

Pequeno ícone que aparece ao lado da barra de endereços ou no canto da aba de navegação do navegador quando acessamos um site, ou ainda quando adicionamos um endereço aos favoritos ou à barra de links.