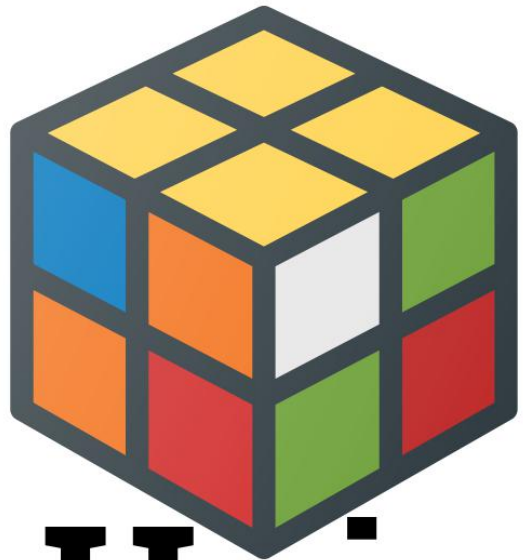


Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

GIL - Python Global Interpreter Lock

GIL - Python Global Interpreter Lock

O Python Global Interpreter Lock, ou simplesmente GIL, é um mutex (ou lock) que permite que apenas uma thread tome conta do interpretador Python.

Isso significa que somente uma thread pode estar em um estado de execução em qualquer ponto do tempo.

O impacto do GIL não é comumente visível para desenvolvedores que executam programas single-thread, mas pode ser uma dor de cabeça para programas que precisam de tempo de resposta em códigos multi-thread.

Desde que o GIL permite apenas uma thread a ser executada, mesmo em computadores multi-threads com arquitetura que permite utilizar mais de um CPU ou core, o GIL tem ganhado reputação como um recurso 'indecente' do Python.

Nesta aula, iremos aprender como o GIL afeta a performance do seu programa Python e também como a gente pode mitigar (diminuir) o impacto no nosso código.

GIL - Python Global Interpreter Lock

Como vimos na aula passada, Python utiliza reference counting para gerenciamento de memória.

Isso significa que para cada objeto criado Python mantém uma variável de contagem de referência que guarda quantas referências apontam para o objeto. Quando o contador de referências chega a zero, a memória ocupada é liberada.

GIL - Python Global Interpreter Lock

Como vimos na aula passada, Python utiliza reference counting para gerenciamento de memória.

Isso significa que para cada objeto criado Python mantém uma variável de contagem de referência que guarda quantas referências apontam para o objeto. Quando o contador de referências chega a zero, a memória ocupada é liberada.

```
>>> import sys

>>> a = []
>>> b = a
>>> sys.getrefcount(a)
3
```

GIL - Python Global Interpreter Lock

Como vimos na aula passada, Python utiliza reference counting para gerenciamento de memória.

Isso significa que para cada objeto criado Python mantém uma variável de contagem de referência que guarda quantas referências apontam para o objeto. Quando o contador de referências chega a zero, a memória ocupada é liberada.

```
>>> import sys

>>> a = []
>>> b = a
>>> sys.getrefcount(a)
3
```

No código ao lado, a contagem de referências deu 3.

O objeto lista foi referenciado por 'a', por 'b' e pelo argumento passado ao sys.getrefcount().

GIL - Python Global Interpreter Lock

O problema é que essa forma de gerenciamento de memória utilizando reference couting precisa de proteção para um fenômeno chamado 'race conditions', onde duas threads aumentam ou diminuem seu valor simultaneamente.

Se isso acontecer, poderá causar problemas de memória que nunca é liberada, ou ainda pior, liberação incorreta de memória enquanto ainda existe referência para o objeto.

E isso pode causar 'crashs' ou outros bugs esquisitos no seu programa Python.

GIL - Python Global Interpreter Lock

Este reference couting pode ser mantido seguro adicionando 'locks' em toda estrutura de dados que são compartilhadas via threads. Desta forma eles não são modificados de forma inconsistente.

O problema é que adicionar 'locks' em cada objeto ou grupo de objetos significa que múltiplos locks irão existir, e isso irá causar um outro problema - Deadlocks (deadlocks só podem existir se existe mais de um lock). Outro efeito colateral seria decaimento da performance causada pela repetida aquisição e liberação dos locks.

GIL - Python Global Interpreter Lock

O GIL aplica na regra de execução de qualquer código Python o single lock previnindo qualquer deadlock, o que por outro lado transforma qualquer código Python em single-thread.

Vale mencionar que o GIL apesar de ser usado também em outras linguagens de programação, como Ruby, não é a única solução.

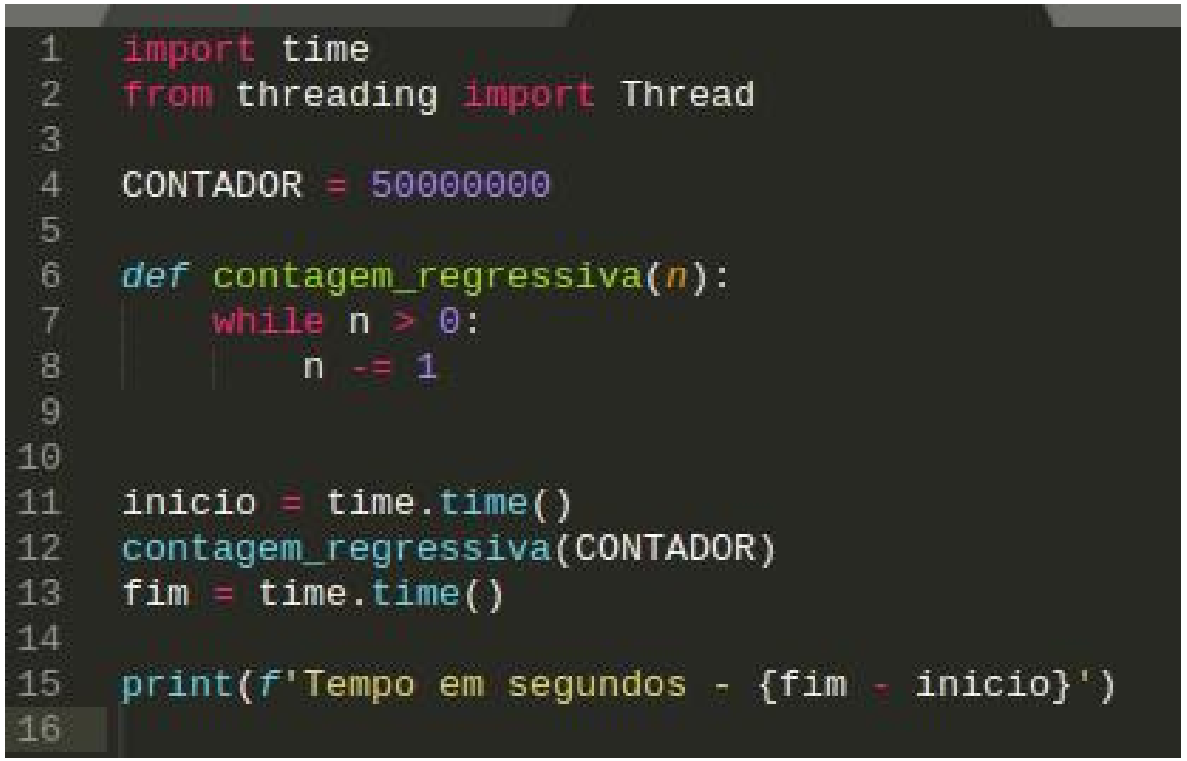
Algumas linguagens evitam o uso do GIL para gerenciamento de memória em thread utilizando abordagens diferentes do reference counting que o Python utiliza.

Por exemplo, uma das abordagens que outras linguagens utilizam é o compilador JIT - Just in Time, como o Java.

GIL - Python Global Interpreter Lock

O impacto em programas Python multi-thread

Exemplo single-thread:



```python
import time
from threading import Thread

CONTADOR = 50000000

def contagem_regressiva(n):
    while n > 0:
        n -= 1


inicio = time.time()
contagem_regressiva(CONTADOR)
fim = time.time()

print(f'Tempo em segundos - {fim - inicio}')

```

GIL - Python Global Interpreter Lock

O impacto em programas Python multi-thread

Exemplo single-thread:

```python
import time
from threading import Thread

CONTADOR = 50000000

def contagem_regressiva(n):
    while n > 0:
        n -= 1


inicio = time.time()
contagem_regressiva(CONTADOR)
fim = time.time()

print(f'Tempo em segundos - {fim - inicio}')

```

```
Tempo em segundos - 1.8489618301391602
```

GIL - Python Global Interpreter Lock

O impacto em programas Python multi-thread

Exemplo 2 threads em paralelo:

```python
import time
from threading import Thread

CONTADOR = 50000000

def contagem_regressiva(n):
    while n > 0:
        n -= 1


t1 = Thread(target=contagem_regressiva, args=(CONTADOR//2,))
t2 = Thread(target=contagem_regressiva, args=(CONTADOR//2,))

inicio = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
fim = time.time()


print(f'Tempo em segundos - {fim - inicio}')

```

GIL - Python Global Interpreter Lock

O impacto em programas Python multi-thread

Exemplo 2 threads em paralelo:

```python
import time
from threading import Thread

CONTADOR = 50000000

def contagem_regressiva(n):
    while n > 0:
        n -= 1


t1 = Thread(target=contagem_regressiva, args=(CONTADOR//2,))
t2 = Thread(target=contagem_regressiva, args=(CONTADOR//2,))

inicio = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
fim = time.time()


print(f'Tempo em segundos - {fim - inicio}')

```

```
Tempo em segundos - 1.7511544227600098
```

GIL - Python Global Interpreter Lock

Conforme pudemos ver, mesmo utilizando multi-thread o tempo de finalização não muda tanto.

A utilização do GIL prejudica a real utilização de multi-cores nas máquinas, o que torna os projetos Python lentos em alguns casos.

Por outro lado, o GIL torna as aplicações single-thread muito performaticas, e a grande maioria das aplicações não precisam utilizar multi-threads.

GIL - Python Global Interpreter Lock

Como lidar com o GIL?

GIL - Python Global Interpreter Lock

Como lidar com o GIL?

Caso você tenha problemas com o GIL, você pode utilizar multi-processing ao invés de multithreading.

Utilizando processos ao invés de threads cada processo Python ganha seu próprio interpretador Python e espaço em memória. Desta forma o GIL não será problema.

GIL - Python Global Interpreter Lock

Exemplo multi-processing

GIL - Python Global Interpreter Lock

Exemplo multi-processing

```python
from multiprocessing import Pool
import time

CONTADOR = 50000000

def contagem_regressiva(n):
    while n > 0:
        n -= 1


if __name__ == '__main__':
    pool = Pool(processes=2)
    inicio = time.time()
    r1 = pool.apply_async(contagem_regressiva, [CONTADOR//2])
    r2 = pool.apply_async(contagem_regressiva, [CONTADOR//2])
    pool.close()
    pool.join()
    fim = time.time()
    print(f'Tempo em segundos - {fim - inicio}')

```

GIL - Python Global Interpreter Lock

Exemplo multi-processing

```python
from multiprocessing import Pool
import time

CONTADOR = 50000000

def contagem_regressiva(n):
    while n > 0:
        n -= 1


if __name__ == '__main__':
    pool = Pool(processes=2)
    inicio = time.time()
    r1 = pool.apply_async(contagem_regressiva, [CONTADOR//2])
    r2 = pool.apply_async(contagem_regressiva, [CONTADOR//2])
    pool.close()
    pool.join()
    fim = time.time()
    print(f'Tempo em segundos - {fim - inicio}')

```

```
Tempo em segundos - 0.9025936126708984
```

Tivemos uma melhora na performance incrível, não?

GIL - Python Global Interpreter Lock

Que fique claro que multi-processing são mais 'pesados' que multi-threading.

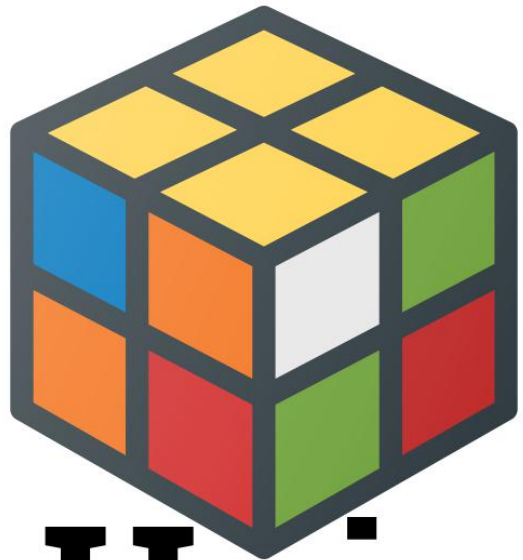
Ou seja, lembre-se que para cada processo, teremos um ambiente Python próprio.

GIL - Python Global Interpreter Lock

Interpretadores alternativos Python

Conforme mencionado antes, Python possui múltiplas implementações dentre as principais: CPython, Jython, IronPython e PyPy, escritos em C, Java, C# e Python respectivamente.

GIL só existe na implementação original (CPython). Então se seu programa estiver rodando em outra implementação, você não terá o problema que o GIL traz.



Geek University

Evolua seu lado geek!

www.geekuniversity.com.br