Programação com memória partilhada (exemplos)

A classe queue.Queue

Classe que implementa uma fila (FIFO) **sincronizada**.

- Usa locks para bloquear threads que competem pelo acesso aos elementos da fila.

Constructor: Queue(*maxsize=0*)

- *maxsize* estabelece um limite máximo para o número de itens na fila

- Quando o limite é atingido, as operações de inserção serão bloqueadas, até ser consumido algum item.

- Se *maxsize <= 0* o tamanho da fila é infinito.

Outras classes do módulo queue:

class queue.LifoQueue(*maxsize=0*)

class queue.PriorityQueue(*maxsize=0*)

exception queue.Full

exception queue.Empty

Outros métodos das classes Queue:

Queue.qsize()

- Devolve o tamanho da fila.
- qsize() > 0 não garante que a operação seguinte de get não bloqueie
- qsize() < maxsize não garante que a operação seguinte de put não bloqueie.

Queue.empty()

- Devolve True se fila vazia, False caso contrário
- Se empty() devolve True, isso não garante que a operação seguinte de put não bloqueie.
- Se empty() devolve False, isso não garante que a operação seguinte de get não bloqueie.

Outros métodos das classes Queue:

Queue.full()

- Devolve True se a vila está cheia, False caso contrário.
- …

Queue.put(*item*, *block=True*, *timeout=None*)

- Insere um item na fila. Se *block* é true e *timeout* é None (valores de omissão), a operação bloqueia até que exista espaço para inserir o novo elemento

- Se *timeout* é um valor positivo a operação bloqueia no máximo por timeout segundos até que gera a exceção Full caso não exista espaço dentro desse tempo.

Queue.put_nowait(*item*) Equivalente a put(item, False).

Outros métodos das classes Queue:

Queue.get(*block=True*, *timeout=None*)

- Remove e devolve um item da queue. Se *block* é true e *timeout* é None (valores de omissão), a operação bloqueia se necessário até haver um item disponível.

- Se t*imeout* é um valor positivo a operação bloqueia no máximo *timeout* segundos, gerando a exceção Empty se não existe um item disponível nesse intervalo de tempo.

Queue.get_nowait() Equivalente a get(False).

Outros métodos das classes Queue:

Queue.task_done()

- Indica que a tarefa “retirada” anteriormente da fila está completa.
- Operação usada por threads que consomem elementos da Queue.
- Para cada get(), deve haver uma chamada de task_done(), assinalando à fila que o processamento da tarefa está completo.

Queue.join()

- Bloqueia até que todos os itens da fila tenham sido processados, O contador de itens é incrementado sempre que um item é adicionado e decrementado sempre que é invocado task_done(). Quando o contador chega a 0 o join() desbloqueia.

Exemplo: como processar numa Queue um conjunto de tarefas

```
def worker():
    while True:
        item = q.get()
        if item is None:
            break
        do_work(item)
        q.task_done()
```

Exemplo: como colocar numa Queue um conjunto de tarefas

```
q = queue.Queue()
threads = []
for i in range(num_worker_threads):
    t = threading.Thread(target=worker)
    t.start()
    threads.append(t)

for item in source():
    q.put(item)

# block until all tasks are done
q.join()
```

Exemplo: como colocar numa Queue um conjunto de tarefas

```
# stop workers
for i in range(num_worker_threads):
    q.put(None)

for t in threads:
    t.join()
```

Exemplo: produtor / consumidor

```
from queue import Queue
import time
import random

class producer(Thread):
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue
    def run(self) :
        for i in range(10):
            item = random.randint(0, 256)
            self.queue.put(item)
            print ('Producer notify : item N° %d appended to queue by %s
                    \n‘  % (item, self.name))
            time.sleep(1)
```

Exemplo: produtor / consumidor

```
class consumer(Thread):
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue

    def run(self):
        while True:
            item = self.queue.get()
            print ('Consumer notify : %d popped from queue by %s'\
                        % (item, self.name))
            self.queue.task_done()
```

Exemplo: produtor / consumidor

```
if __name__ == '__main__':
    queue = Queue()
    t1 = producer(queue)
    t2 = consumer(queue)
    t3 = consumer(queue)
    t1.start()
    t2.start()
    t3.start()
    t1.join()
    t2.join()
    t3.join()
```

Exercício: implementar, executar, modificar para terminar consumidor.

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

Cinco filósofos, Aristotle, Kant, Spinoza, Marx, e Russell (**tasks**) gastam o seu tempo a pensar e a comer esparguete.

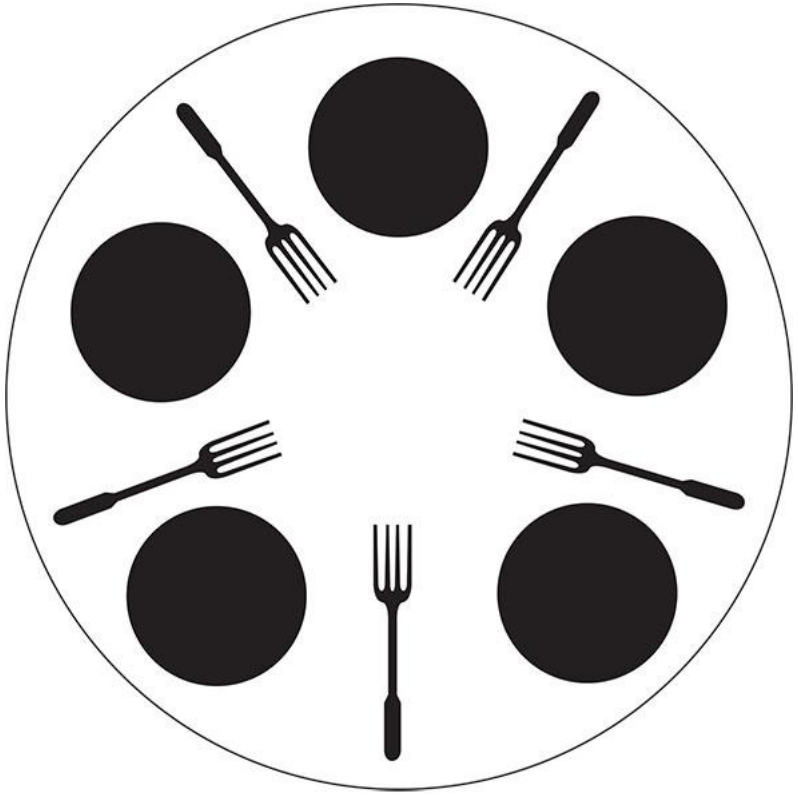
Comem numa mesa redonda com 5 lugares. Para comer, cada filósofo precisa 2 garfos (**resources**). Existem cinco garfos na mesa, um à direita e um à esquerda de cada lugar.

Quando um filósofo que vai comer não consegue pegar nos dois garfos, fica em espera. Comer demora um tempo aleatório, após o qual o filósofo pousa os dois garfos e vai pensar.

Após passar um tempo aleatório a pensar sobre a natureza do universo, ele vai novamente comer. ….

Exemplo: Dining Philosophers

(by Edsger Dijkstra)



Uma primeira hipótese de solução em python ➔

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

```
import threading
import random
import time

class Philosopher(threading.Thread):

    running = True   # variável de classe

    def __init__(self, name, forkOnLeft, forkOnRight):
        threading.Thread.__init__(self)
        self.name = name
        self.forkOnLeft = forkOnLeft
        self.forkOnRight = forkOnRight
```

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

```
def run(self):
    while(self.running):
        # Philosopher is thinking.
        time.sleep( random.uniform(3,13))
        print ('%s is hungry.' % self.name)
        self.dine()
```

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

```
def dine(self):
    self.forkOnLeft.acquire()
    self.forkOnRight.acquire()
    self.dining()
    self.forkOnLeft.release()
    self.forkOnRight.release()
```

O que acontece se cada Filósofo, pega no garfo à sua direita?

```
def dining(self):
    print ('%s starts eating '% self.name)
    time.sleep(random.uniform(1,10))
    print ('%s finishes eating and leaves to think.' % self.name)
```

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

```
def DiningPhilosophers():
    forks = [threading.Lock() for n in range(5)]
    philosopherNames = ('Aristotle','Kant','Buddha','Marx', 'Russel')
    philosophers= [Philosopher(philosopherNames[i], forks[i%5], \
                                                    forks[(i+1)%5]) for i in range(5)]
    random.seed(507129)
    Philosopher.running = True
    for p in philosophers: p.start()
    time.sleep(100)
    Philosopher.running = False
    Print(“    ***Now we're finishing.")

DiningPhilosophers()
```

➔ testar

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

Resolver o deadlock:

```
def dine(self):
    fork1, fork2 = self.forkOnLeft, self.forkOnRight
    while self.running:
        fork1.acquire(True)
        locked = fork2.acquire(False)
        if locked: break
        fork1.release()
        print ('%s swaps forks' % self.name)
        fork1, fork2 = fork2, fork1
    else:
        return
    self.dining()
    fork2.release()
    fork1.release()
```

Se conseguiu os 2 locks, segue para self.dining()

Se não conseguiu, o 2º lock, liberta o primeiro, e troca os locks, continuando a tentar

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

X = Lock().

x.acquire(*blocking=True*, *timeout=-1*)

- Adquire o lock, de forma bloqueante ou não bloqueante.

- Quando é invocado com *blocking* = True (valor de omissão) bloqueia até que o lock seja libertado e devolve True.

- Quando invocado com *blocking* = False, não bloqueia. Se consegue adquire o lock devolve True, caso contrário devolve False

Exemplo: Dining Philosophers

(by Edsger Dijkstra)

Estrutura:

```
while valor:

        if something: break
else:
        return
        # executado, quando valor == False se não ocorreu o break
```

Exemplo: Readers and Writers

a) Construa uma classe em RW, que possua um campo inteiro, XPTO, e dois métodos: ler e escrever. O método ler deve devolver o valor da variável XPTO; o método escrever deve adicionar o valor 100 à variável XPTO e seguidamente subtrair o mesmo valor à variável XPTO.

b) Pretende-se que um objecto da classe RW seja partilhado por vários processos (Threads) de dois tipos:

- processos Leitores – que lêem o valor da variável XPTO usando o método ler;
- processos Escritores – que alteram a variável XPTO usando o método escrever.
- Construa as classes Leitor e Escritor. Cada uma destas classes deve ter uma Thread de execução própria em que, num ciclo infinito, vão respectivamente lendo e alterando valores do objecto partilhado.

Exemplo: Readers and Writers

c) Construa uma classe de teste que crie um objecto do tipo RW, 3 objectos do tipo Leitor e 2 objectos do tipo Escritor. Estude o comportamento do seu programa.

Código … RW.py

d) Pretende-se que modifique as classes anteriores tal **que os vários processos Leitores possam executar concorrentemente o método ler, mas que quando um processo Escritor executar o método escrever o faça em exclusão mútua**. Isto é, quando um processo está a escrever, nenhum outro pode em simultâneo ler ou escrever a variável XPTO.

Exemplo: Readers and Writers

Código … rwSync.py

```
import threading
import time
import random

class RW():
    def __init__ (self):
    self.cond = threading.Condition()
    self.nl = 0
    self.xpto = 0

    def ler (self):
        time.sleep(random.uniform(1,3))
        return self.xpto
```

Exemplo: Readers and Writers

```
def escrever (self):
    self.cond.acquire()
    while ( self.nl > 0):
        self.cond.wait()
     self.xpto += 100
     time.sleep(random.uniform(1,3))
     self.xpto -= 100
     self.cond.release()
```

Exemplo: Readers and Writers

```
def startLer(self):
    self.cond.acquire()
    self.nl += 1
    self.cond.release()

def endLer (self):
    self.cond.acquire()
    self.nl -= 1
    if (self.nl == 0):
        self.cond.notifyAll()
    self.cond.release()
```

Exemplo: Readers and Writers

```
class Leitor(threading.Thread):
    def __init__ (self, rw):
        treading.Thread.__init__ (self)
        self.rw = rw

    def run (self):
        while (True):
            self.rw.startLer()
            if (self.rw.ler() != 0):
                print ("Secção crítica violada")
            self.rw.endLer()
```

Exemplo: Readers and Writers

```
class Escritor(threading.Thread):
    def __init__ (self, rw):
        threading.Thread.__init__ (self)
        self.rw = rw


    def run (self):
        while (True):
            self.rw.escrever()
```

Exemplo: Readers and Writers

```
if __name__ == "__main__":
    x = RW()
    l1 = Leitor(x)
    l2 = Leitor(x)
    l3 = Leitor(x)
    e1 = Escritor(x)
    e2 = Escritor(x)
    l1.start()
    l2.start()
    l3.start()
    e1.start()
    e2.start()
```

Performance

Execução sequencial versus multithreading

```
from threading import Thread
import urllib.request
import time

#ExecuçãoSequencial
class nothreads_object(object):
    def run(self):
        function_to_run()

def non_threaded(num_iter):
    funcs = []
    for i in range(int(num_iter)):
        funcs.append(nothreads_object())
    for i in funcs:
        i.run()
```

Performance

Execução sequencial versus multithreading

```
#execução multithreaded
class threads_object(Thread):
    def run(self):
        function_to_run()

def threaded(num_threads):
    funcs = []
    for i in range(int(num_threads)):
        funcs.append(threads_object())
    for i in funcs:
        i.start()
    for i in funcs:
        i.join()
```

Performance

Execução sequencial versus multithreading

```
def show_results(func_name, results):
    print ("%-23s %4.6f seconds"% (func_name, results))



def function_to_run():
    #Fibonacci
    a = 0
    b = 1
    for i in range (100000):
        a = b
        b = a + b
```

Performance

```
if __name__ == "__main__":
    num_threads = [ 1, 2, 4, 8 ]
    print('Starting tests')
    for i in num_threads:
        start = time.time()
        non_threaded (i)
        executionTime = time.time() - start
        show_results("non_threaded (%s iters)" % i, executionTime)

        start = time.time()
        threaded (i)
        executionTime = time.time() - start
        show_results("threaded (%s threads)" % i, executionTime)
    print ('Iterations complete')
```

Performance

- **Estudar o comportamento do programa com a função da página 33 (Fibonacci)**

- **Alterar o código para dado um número de execuções, o programa calcular o tempo médio de execução para cada uma das situações estudadas (1, 2, 4, 8 threads / iterações)**

- **Que código é mais rápido, sequencial ou multi-threading?**

Performance

Estudar os tempos de execução para as seguintes funções:
(substitua o URL por outro à sua escolha)
a)

```
def function_to_run():
    for i in range(10):
        with urllib.request.urlopen("https://google.com")as f:
            f.read(1024)
```

Performance

Estudar os tempos de execução para as seguintes funções:

b)

```
def function_to_run():
    file = open ("test.dat", "rb")
    size = 1024
    for i in range (1000):
        file.read(size)
```