Programação em sistemas de memória distribuída

(Message-Passing Computing)

Exercício:

- Testar o programa anterior, analisando os tempos de execução.

- Exercício para férias: Como obter o valor parcial calculado por cada thread e cada processo?

- Quais os tempos de execução, se na versão multiprocessador, só passarmos para cada Processo a parte da lista que vai ser somada?

Python multiprocessing

Comunicação de dados entre processos

Queues – permite vários produtores e vários recetores.

Pipes – comunicar entre dois processos

Python multiprocessing
Comunicação de dados entre processos

#Queue - exemplo

```
import multiprocessing
import random
import time

class producer(multiprocessing.Process):
    def __init__(self, queue):
        multiprocessing.Process.__init__(self)
        self.queue = queue
```

Python multiprocessing
Comunicação de dados entre processos

```
def run(self) :
    for i in range(10):
        item = random.randint(0, 256)
        self.queue.put(item)
        print ("Process Producer : item %d appended to queue
                                                %s“  % (item,self.name))
        time.sleep(1)
        print ("The size of queue is %s“ % self.queue.qsize())
```

Python multiprocessing
Comunicação de dados entre processos

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

Python multiprocessing
Comunicação de dados entre processos

```
def run(self):
    while True:
        if (self.queue.empty()):
            print("the queue is empty")
            break;
        else :
            time.sleep(2)
            item = self.queue.get()
            print ('Process Consumer : item %d popped from by %s
                                                \n‘ % (item, self.name))
            time.sleep(1)
```

Método task_done() não existe

Python multiprocessing
Comunicação de dados entre processos

```
if __name__ == '__main__':
    queue = multiprocessing.Queue()
    process_producer = producer(queue)
    process_consumer = consumer(queue)
    process_producer.start()
    process_consumer.start()
    process_producer.join()
    process_consumer.join()
```

➔ O que acontece??

Python multiprocessing

Comunicação de dados entre processos

Substituir no produtor:

```
class producer(multiprocessing.Process):
 …
    def run(self) :
        for i in range(10):
            item = random.randint(0, 256)
            self.queue.put(item)
            print ('Process Producer : item %d appended to queue
                                                    %s' % (item,self.name))
            time.sleep(1)
            print ('The size of queue is %s' % self.queue.qsize())
        self.queue.put('DONE')
```

Python multiprocessing
Comunicação de dados entre processos

Substituir no consumidor:

```
class consumer(multiprocessing.Process):
 …
        def run (self):
            while True:
                time.sleep (2)
                item = self.queue.get()
                if (item == 'DONE'):
                    break
                print ('Process Consumer : item %d popped from by
                                                %s \n' % (item, self.name))
```

Python multiprocessing
Comunicação de dados entre processos

```
#Pipe – exemplo

import multiprocessing

class producer(multiprocessing.Process):
    def __init__(self, pipe):
        multiprocessing.Process.__init__(self)
        self.pipe = pipe
```

Python multiprocessing
Comunicação de dados entre processos

```
def run(self) :
    output_pipe, input_pipe = self.pipe
    for item in range(10):
        output_pipe.send(item)
    output_pipe.send(99)
    output_pipe.close()
```

#Pipe() devolve um par de “connection objects” ligados por um pipe bidirecional.

Python multiprocessing
Comunicação de dados entre processos

```
class consumer(multiprocessing.Process):
    def __init__(self, pipe):
        multiprocessing.Process.__init__(self)
        self.pipe = pipe

    def run(self):
        close, input_pipe = self.pipe
        close.close()
        while (True):
            item = input_pipe.recv()
            if (item == 99):
                break
                input_pipe.close()
            print ("1: " , str(item * item))
```

Python multiprocessing

Comunicação de dados entre processos

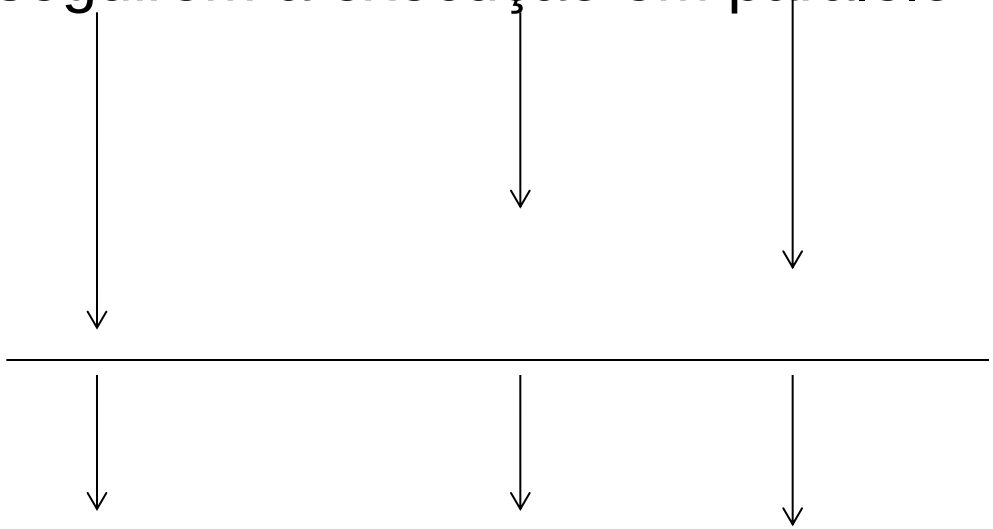
```
if __name__ == '__main__':
    pipe = multiprocessing.Pipe()
    process_producer = producer(pipe)
    process_consumer = consumer(pipe)
    process_producer.start()
    process_consumer.start()
    process_producer.join()
    process_consumer.join()
```

Python multiprocessing
Sincronização de processos

Lock, Rlock, Semaphore, Condition, Event

Barrier – ponto de sincronização entre um conjunto de processos

Todos os processos têm de atingir um determinado ponto antes de prosseguirem a execução em paralelo



Python multiprocessing
Sincronização de processos

```
import multiprocessing
from multiprocessing import Barrier, Lock, Process
from time import time
from datetime import datetime

if __name__ == '__main__':
    synchronizer = Barrier(2)  “”” ← número de processo na barreira: 2 “””
    serializer = Lock()
    Process(name='p1 - test_with_barrier‘ , target=test_with_barrier,\
                                        args=(synchronizer, serializer)).start()
    Process(name='p2 - test_with_barrier', target=test_with_barrier,\
                                         args=(synchronizer,serializer)).start()

    Process(name='p3 - test_without_barrier‘, target=test_without_barrier).start()
    Process(name='p4 - test_without_barrier‘ ,target=test_without_barrier).start()
```

Python multiprocessing
Sincronização de processos

```
def test_with_barrier(synchronizer, serializer):
    name = multiprocessing.current_process().name
    synchronizer.wait()  # barrier
    now = time()

    with serializer:
        print("process %s ----> %s" %(name,datetime.fromtimestamp(now)))

def test_without_barrier():
    name = multiprocessing.current_process().name
    now = time()
    print("process %s ----> %s" %(name ,datetime.fromtimestamp(now)))
```

Python multiprocessing
Sincronização de processos

Output1:

process p3 - test_without_barrier ----> 2018-04-01 19:27:49.878603
process p2 - test_with_barrier ----> 2018-04-01 19:27:49.878603
process p1 - test_with_barrier ----> 2018-04-01 19:27:49.878603
process p4 - test_without_barrier ----> 2018-04-01 19:27:49.931988

Output2:

process p3 - test_without_barrier ----> 2020-04-02 19:20:29.842714
process p2 - test_with_barrier ----> 2020-04-02 19:20:29.870697
process p1 - test_with_barrier ----> 2020-04-02 19:20:29.870697
process p4 - test_without_barrier ----> 2020-04-02 19:20:29.872697

Instrução with

```
import threading
import logging

logging.basicConfig(level=logging.DEBUG, format='(%(threadName)-10s)
   !!                                                    %(message)s',)
if __name__ == '__main__':
    lock = threading.Lock()
    t1 = threading.Thread(target=threading_with, args=(lock,))
    t2 = threading.Thread(target=threading_not_with, args=(lock,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

#Instrução with

```
def threading_with(statement):
    with statement:
        logging.debug('%s acquired via with' %statement)

def threading_not_with(statement):
    statement.acquire()
    try:
      logging.debug('%s acquired directly' %statement )
    finally:
        statement.release()
```

#Instrução with

Output:

•(Thread-5) <_thread.lock object at 0x000000000689ADC8> acquired via with
•(Thread-6) <_thread.lock object at 0x000000000689ADC8> acquired directly

Com with o lock é adquirido, só existe dentro do âmbito do with e é libertado quando termina o bloco do with

Python multiprocessing

Process Pool

- **Permite criar um conjunto de processos**

Algumas funções:

- map () – divide um conjunto iterável de dados em pedaços que submete aos processos da pool como tarefas individuais; bloqueia até à obtenção do resultado;

- **map_async() – map não bloqueante; devolve um objeto**

Python multiprocessing

Process Pool - exemplo

```
import multiprocessing

def function_square(data):
result = data*data
return result
```

Python multiprocessing

Process Pool - exemplo

```
if __name__ == '__main__':
    inputs = list(range(0,100))
    pool = multiprocessing.Pool(processes=4)
    pool_outputs = pool.map(function_square, inputs)
    print ('Pool :', pool_outputs)

    inputs2 = list(range(100,200))
    pool2 = multiprocessing.Pool(processes=4)
    x = pool2.map_async(function_square, inputs2)
    #some work ....
    x.wait()
    print ('Pool :', x.get())
```