AULA 3

PI PARALELO + PIPE

```
from multiprocessing import
Process, Queue
import time
PROCS = 2
a,b = Pipe()

def pi(start, end, step, sums):
  print "Start: " + str(start)
  print "End: " + str(end)
  sum = 0.0
  for i in range(start, end):
    x = (i+0.5) * step
    sum = sum + 4.0/(1.0+x*x)
  sums.send(sum)

if __name__ == "__main__":
  num_steps = 100000000
  sum = 0.0
  step = 1.0/num_steps
  proc_size = num_steps / PROCS
  tic = time.time()
  workers = []
  for i in range(PROCS):
    worker = Process( target=pi , args
=(i*proc_size, (i+1)*proc_size - 1,
step, a))
    workers.append(worker)

  for worker in workers :
    worker.start()
  for worker in workers :
    worker.join()
  toc = time.time()

  for i in range(PROCS):
    sum += b.recv()
  pi = step * sum
  print pi
  print "Pi: %.8f s" %(toc-tic)
```

EXERCÍCIO

Transforme o programa Pi em paralelo + PIPES!

Atenção: Os dados de um pipe podem ficar corrompidos se dois processos (ou threads) tentarem ler ou gravar na mesma extremidade do pipe ao mesmo tempo.

PI PARALELO + PIPE

```
from multiprocessing import
Process, Queue
import time
PROCS = 2
a,b = Pipe()

def pi(start, end, step, sums):
  print "Start: " + str(start)
  print "End: " + str(end)
  sum = 0.0
  for i in range(start, end):
    x = (i+0.5) * step
    sum = sum + 4.0/(1.0+x*x)
  sums.send(sum)
```

Condição de corrida!!!

```
if __name__ == "__main__":
  num_steps = 100000000
  sum = 0.0
  step = 1.0/num_steps
  proc_size = num_steps / PROCS
  tic = time.time()
  workers = []
  for i in range(PROCS):
    worker = Process( target=pi , args
=(i*proc_size, (i+1)*proc_size - 1,
step, a))
    workers.append(worker)

  for worker in workers :
    worker.start()
  for worker in workers :
    worker.join()
  toc = time.time()

  for i in range(PROCS):
    sum += b.recv()
  pi = step * sum
  print pi
  print "Pi: %.8f s" %(toc-tic)
```

SINCRONIZAÇÃO E REGIÕES CRÍTICAS

A principal causa da ocorrência de erro na programação de processos está relacionada com o fato do uso de estruturas de dados compartilhadas.

Apesar de este ser um aspectos mais poderosos da utilização de processos, também pode ser um dos mais problemáticos.

O problema existe quando dois ou mais processos tentam acessar/alterar as mesmas estruturas de dados (race conditions)**.**

EXEMPLO

Tempo	P1	P2	Saldo
T0			$200

EXEMPLO

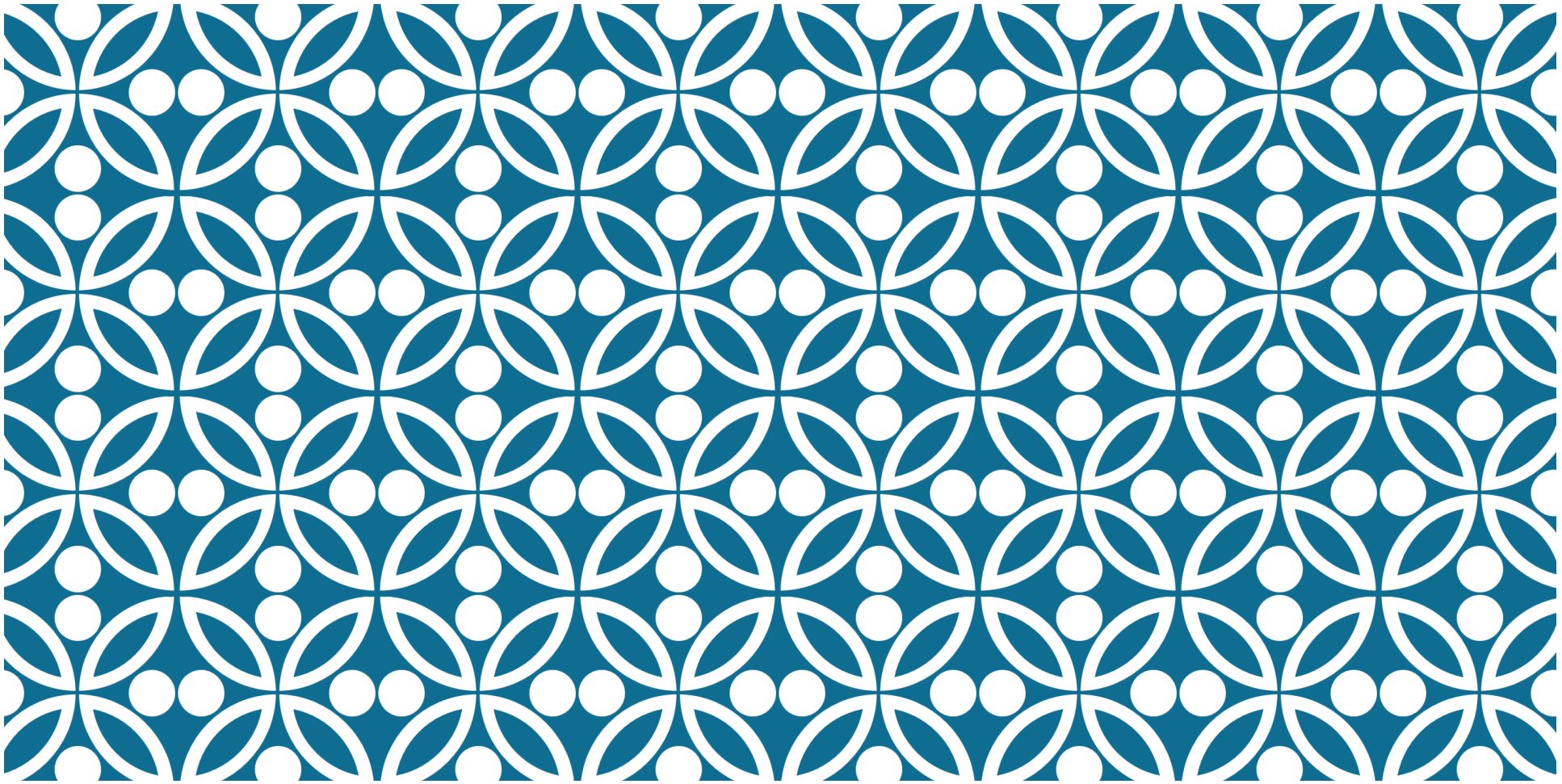
Tempo	P1	P2	Saldo
T0			$200
T1	Leia Saldo $200		$200
T2		Leia Saldo $200	$200

EXEMPLO

Tempo	P1	P2	Saldo
T0			$200
T1	Leia Saldo $200		$200
T2		Leia Saldo $200	$200
T3		Some $100 $300	$200
T4	Some $150 $350		$200

EXEMPLO

Tempo	P1	P2	Saldo
T0			$200
T1	Leia Saldo $200		$200
T2		Leia Saldo $200	$200
T3		Some $100 $300	$200
T4	Some $150 $350		$200
T5		Escreva Saldo $300	**$300**
T6	Escreva Saldo $350		**$350**



CONTROLANDO ACESSO A RECURSOS USANDO **LOCKS**

LOCKS

Em situações em que um único recurso precisa ser compartilhado entre vários processos, um bloqueio pode ser usado para evitar acessos conflitantes.

acquire(block=True, timeout=None)

- Adquira um bloqueio, bloqueando ou não bloqueando.
- Com o argumento block definido como True (padrão), a chamada do método será bloqueada até que o bloqueio esteja em um estado desbloqueado, em seguida, defina-o como bloqueado e retorne True.
- Com o argumento block definido como False, a chamada do método não é bloqueada. Se o bloqueio estiver atualmente em um estado bloqueado, retorne False; caso contrário, defina o bloqueio para um estado bloqueado e retorne True.
- Quando chamado com um valor de ponto flutuante positivo para o tempo limite, bloqueie no máximo o número de segundos especificado pelo tempo limite, desde que o bloqueio não possa ser adquirido. Invocações com um valor negativo para tempo limite são equivalentes a um tempo limite de zero. Invocações com um valor de tempo limite de Nenhum (o padrão) definem o período de tempo limite como infinito.

release()

- Solte um bloqueio. Isso pode ser chamado de qualquer processo ou thread, não apenas o processo ou thread que originalmente adquiriu o bloqueio.

CONTROLANDO ACESSO A RECURSOS

```
import multiprocessing
import sys
def worker_with(lock, stream):
  with lock:
    stream.write('Lock acquired via with\n')

def worker_no_with(lock, stream):
  lock.acquire()
  try:
    stream.write('Lock acquired directly\n')
  finally:
    lock.release()

lock = multiprocessing.Lock()
w = multiprocessing.Process(target=worker_with, args=(lock, sys.stdout))
nw = multiprocessing.Process(target=worker_no_with, args=(lock, sys.stdout))
w.start()
nw.start()
w.join()
nw.join()
```

Neste exemplo, as mensagens impressas no console podem ser misturadas se os dois processos não sincronizarem o acesso do fluxo de saída ao bloqueio.

Saída:
Lock acquired via with
Lock acquired directly

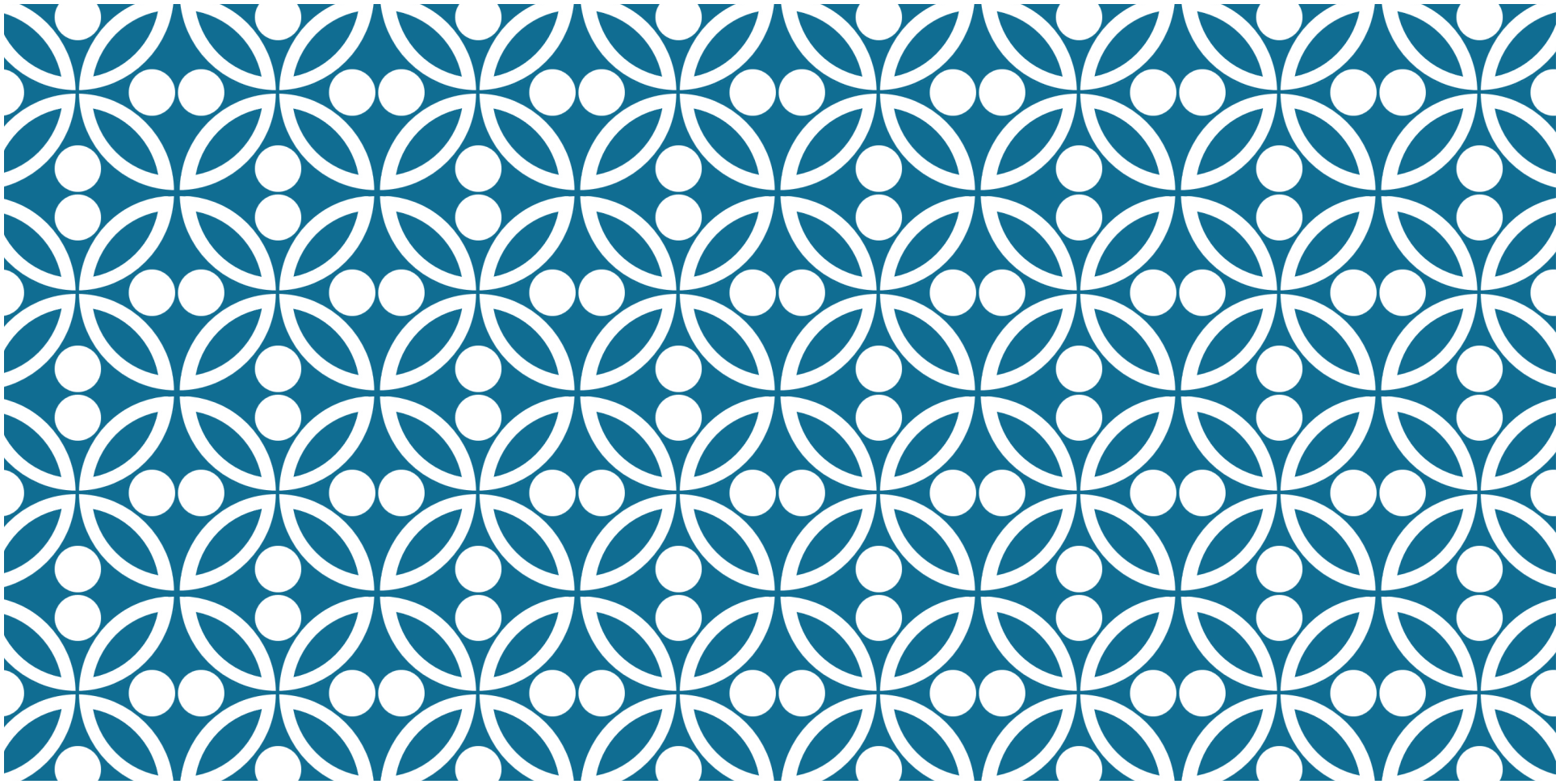
EXERCÍCIO

Transforme o programa Pi em paralelo + PIPES!

Agora vamos utilizer locks para que nosso programa seja **multiprocess-safe**

Vamos precisar:

- **Criar um lock**
- **Enviar o lock como parametron**
- **Antes de inserir no pipe, vamos usar o lock….. “ *with lock:* “**



VALUE E ARRAY COMPARTILHADOS

VALUE

Value(typecode, value)

Crie um objeto com um atributo de valor gravável e retorne um proxy para ele.

Retorna um objeto ctypes alocado na memória compartilhada. Por padrão, o valor de retorno é, na verdade, um wrapper sincronizado para o objeto. O objeto em si pode ser acessado através do atributo value de um valor.

typecode_or_type determina o tipo do objeto retornado: ele é um tipo de ctypes ou um caractere de um caractere do tipo usado pelo módulo de matriz. * args é passado para o construtor para o tipo.

EXEMPLO COM VALUE

```
import time
from multiprocessing import Process, Value

def func(val):
    for i in range(50):
        time.sleep(0.01)
        val.value += 1

if __name__ == '__main__':
    v = Value('i', 0)
    procs = [Process(target=func, args=(v,)) for i in range(10)]

    for p in procs: p.start()
    for p in procs: p.join()

    print v.value
```

Funciona?

EXEMPLO COM VALUE

```
import time
from multiprocessing import Process, Value

def func(val):
    for i in range(50):
        time.sleep(0.01)
        val.value += 1

if __name__ == '__main__':
    v = Value('i', 0)
    procs = [Process(target=func, args=(v,)) for i in range(10)]

    for p in procs: p.start()
    for p in procs: p.join()

    print v.value
```

Temos uma condição de corrida!!!

VALUE

Value(typecode, value)

Se lock for True (o padrão), um novo objeto de bloqueio recursivo será criado para sincronizar o acesso ao valor. Se lock é um objeto Lock ou RLock, isso será usado para sincronizar o acesso ao valor. Se o bloqueio for Falso, o acesso ao objeto retornado não será automaticamente protegido por um bloqueio, portanto, ele não será necessariamente "seguro para o processo".

Operações como + = que envolvem leitura e gravação não são atômicas. Então, se, por exemplo, você quiser incrementar atomicamente um valor compartilhado, é insuficiente fazer apenas

```
contador.valor += 1
```

Mas, você pode fazer:

```
with counter.get_lock():
    contador.valor + = 1
```

EXEMPLO COM VALUE

```
import time
from multiprocessing import Process, Value

def func(val):
    for i in range(50):
        time.sleep(0.01)
        val.value += 1

if __name__ == '__main__':
    v = Value('i', 0, lock=True)
    procs = [Process(target=func, args=(v,)) for i in range(10)]

    for p in procs: p.start()
    for p in procs: p.join()

    print v.value
```

Definir um lock para um VALUE não significa que ele usará automaticamente!!!

EXEMPLO COM VALUE

```
import time
from multiprocessing import Process, Value

def func(val):
    for i in range(50):
        time.sleep(0.01)
        with val.get_lock():
            val.value += 1

if __name__ == '__main__':
    v = Value('i', 0, lock=True)
    procs = [Process(target=func, args=(v,)) for i in range(10)]

    for p in procs: p.start()
    for p in procs: p.join()

    print v.value
```

Yes! Agora temos uma função confiável

ARRAY

Array(typecode, sequence)

Crie uma matriz e retorne um proxy para ela.

Retorna uma matriz ctypes alocada da memória compartilhada. Por padrão, o valor de retorno é, na verdade, um wrapper sincronizado para a matriz.

typecode_or_type determina o tipo dos elementos da matriz retornada: é um tipo ctypes ou um tipo de caractere de um caractere do tipo usado pelo módulo da matriz. Se size_or_initializer for um inteiro, ele determinará o comprimento da matriz e a matriz será inicialmente zerada. Caso contrário, size_or_initializer é uma sequência que é usada para inicializar o array e cujo comprimento determina o comprimento do array.

ARRAY

Array(typecode, sequence)

Se o bloqueio for True (o padrão), um novo objeto de bloqueio será criado para sincronizar o acesso ao valor. Se lock é um objeto Lock ou RLock, isso será usado para sincronizar o acesso ao valor. Se o bloqueio for Falso, o acesso ao objeto retornado não será automaticamente protegido por um bloqueio, portanto, ele não será necessariamente “seguro para o processo”.

Observe que uma matriz de ctypes.c_char tem valor e atributos brutos que permitem usá-lo para armazenar e recuperar strings.

C_TYPES

ctypes type	C type	Python type
c_bool	_Bool	bool (1)
c_char	char	1-character bytes object
c_wchar	wchar_t	1-character string
c_byte	char	int
c_ubyte	unsigned char	int
c_short	short	int
c_ushort	unsigned short	int
c_int	int	int
c_uint	unsigned int	int
c_long	long	int
c_ulong	unsigned long	int
c_longlong	__int64 or long long	int
c_ulonglong	unsigned __int64 or unsigned long long	int
c_size_t	size_t	int
c_ssize_t	ssize_t or Py_ssize_t	int
c_float	float	float
c_double	double	float
c_longdouble	long double	float
c_char_p	char * (NUL terminated)	bytes object or None
c_wchar_p	wchar_t * (NUL terminated)	string or None
c_void_p	void *	int or None

EXERCÍCIO

Transforme o programa Pi em paralelo + (VALUE ou ARRAY)!