

**PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ**  
**FACULTAD DE CIENCIAS E INGENIERÍA**

**Guía para el tercer laboratorio de**  
**INF239 Sistemas Operativos**

**Tema: Concurrencia en Python**

El presente material pretende mostrar con algunos ejemplos el uso de las herramientas de concurrencia que proporciona Python. No pretende explicar cómo funcionan, en Internet, usted puede encontrar la documentación correspondiente.

La documentación se encuentra en: <https://docs.python.org/3/library/threading.html>

Todos los ejemplos serán escritos como *scripts* y la ejecución siempre será desde la terminal.

**Problema 1**

```
problema1.py x
1 import threading as th
2
3 def hilo(num):
4     print(f"Hi, congratulations you are running your first threading program, i am thread {num}.")
5     print(f"Python threads are used in cases where the execution of a task involves some waiting, i am thread {num}.")
6     print(f"Threading allows python to execute other code while waiting, i am thread {num}.")
7
8 hilos = []
9 for n in range(4):
10     h = th.Thread(target=hilo, args=(n,))
11     hilos.append(h)
12
13 for n in range(4):
14     hilos[n].start()
15
```

```
alejandro@abdebian:Concurrencia$ python3 problema1.py
Hi, congratulations you are running your first threading program, i am thread 0.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 0.
Hi, congratulations you are running your first threading program, i am thread 1.
Threading allows python to execute other code while waiting, i am thread 0.
Hi, congratulations you are running your first threading program, i am thread 2.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 2.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 1.
Threading allows python to execute other code while waiting, i am thread 1.
Hi, congratulations you are running your first threading program, i am thread 3.
Threading allows python to execute other code while waiting, i am thread 2.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 3.
Threading allows python to execute other code while waiting, i am thread 3.
alejandro@abdebian:Concurrencia$ █
```

Se crean 4 hilos (de 0 a 3) y cada uno de ellos debe de imprimir 3 líneas de texto. Se puede observar en la salida que esto no sucede. Cada hilo, al imprimir su mensaje, ha sido interrumpido por los otros hilos. Esto es por que, la pantalla también es un recurso compartido. Lo interesante es que esta situación no es constante, usted puede ejecutar varias veces y obtendrá distintos resultados.

Lo que se desea en este caso es que cada hilo imprima su mensaje completo (las tres líneas) antes de que otro hilo haga uso de la pantalla.

## Solución 1 – Empleando Locks

```
problema1.py  solucion1.py x
1 import threading as th
2
3 def hilo(num):
4     l.acquire()
5     print(f"Hi, congratulations you are running your first threading program, i am thread {num}.")
6     print(f"Python threads are used in cases where the execution of a task involves some waiting, i am thread {num}.")
7     print(f"Threading allows python to execute other code while waiting, i am thread {num}.")
8     l.release()
9
10 hilos = []
11 l = th.Lock()
12 for n in range(4):
13     h = th.Thread(target=hilo,args=(n,))
14     hilos.append(h)
15
16 for n in range(4):
17     hilos[n].start()
```

```
alejandro@abdebien:Concurrencia$ python3 solucion1.py
Hi, congratulations you are running your first threading program, i am thread 0.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 0.
Threading allows python to execute other code while waiting, i am thread 0.
Hi, congratulations you are running your first threading program, i am thread 1.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 1.
Threading allows python to execute other code while waiting, i am thread 1.
Hi, congratulations you are running your first threading program, i am thread 2.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 2.
Threading allows python to execute other code while waiting, i am thread 2.
Hi, congratulations you are running your first threading program, i am thread 3.
Python threads are used in cases where the execution of a task involves some waiting, i am thread 3.
Threading allows python to execute other code while waiting, i am thread 3.
alejandro@abdebien:Concurrencia$ █
```

Observe ahora la salida, ningún hilo es interrumpido. Todos imprimen su mensaje antes que otro lo haga. Al ejecutarlo varias veces, siempre obtendrá el mismo resultado

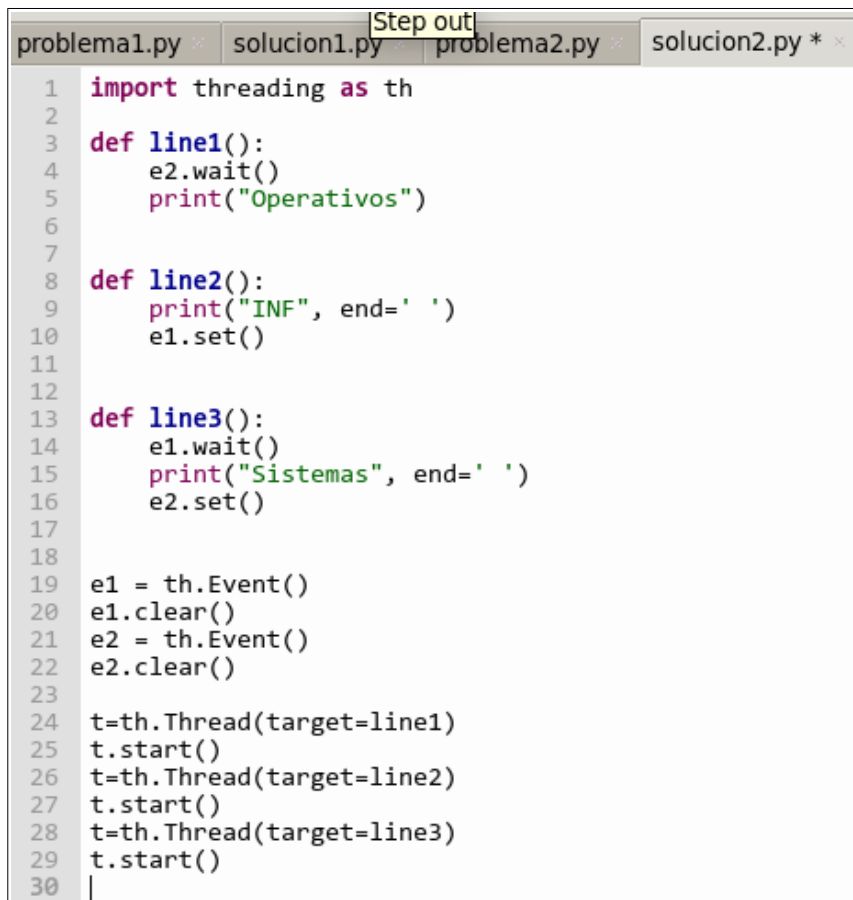
## Problema 2

```
problema1.py  solucion1.py  problema2.py x
1 import threading as th
2
3 def line1():
4     print("Operativos")
5
6 def line2():
7     print("INF", end=' ')
8
9 def line3():
10    print("Sistemas", end=' ')
11
12 t=th.Thread(target=line1)
13 t.start()
14 t=th.Thread(target=line2)
15 t.start()
16 t=th.Thread(target=line3)
17 t.start()
```

```
alejandro@abdebien:Concurrencia$ python3 problema2.py
Operativos
INF Sistemas alejandro@abdebien:Concurrencia$ █
```

El objetivo es sincronizar los tres hilos para que impriman: **INF Sistemas Operativos**

**Solución 2** - Empleando Eventos.



```
1 import threading as th
2
3 def line1():
4     e2.wait()
5     print("Operativos")
6
7
8 def line2():
9     print("INF", end=' ')
10    e1.set()
11
12
13 def line3():
14     e1.wait()
15     print("Sistemas", end=' ')
16     e2.set()
17
18
19 e1 = th.Event()
20 e1.clear()
21 e2 = th.Event()
22 e2.clear()
23
24 t=th.Thread(target=line1)
25 t.start()
26 t=th.Thread(target=line2)
27 t.start()
28 t=th.Thread(target=line3)
29 t.start()
30 |
```

```
alejandro@abdebien:Concurrencia$ python3 solucion2.py
INF Sistemas Operativos
alejandro@abdebien:Concurrencia$ █
```

### Problema 3

El objetivo es simular el problema del productor-consumidor usando *buffer* limitado. En Python los arreglos no están definidos nativamente, pero pueden ser sustituidos por listas. Una lista de Python puede ser accedida por un índice, como en los arreglos. Una lista puede crecer si se emplean los métodos `insert` y `append`. No hay otra forma de modificar su tamaño. Nosotros definiremos su tamaño al inicio y no usaremos los métodos mencionados, de esta forma se comportará como un arreglo de tamaño estático. Por este motivo protegeremos el acceso a la lista, exigiendo que el índice esté entre los límites de la lista.

El comportamiento del productor-consumidor debe ser el siguiente: siempre que haya espacio el productor puede producir un elemento y almacenarlo en el *buffer*. Cuando el *buffer* se encuentra lleno debe esperar. Por otro lado el consumidor debe consumir elementos del *buffer*, siempre y cuando haya elementos que consumir, en caso contrario debe de esperar.

problema1.py	solucion1.py	problema2.py	solucion2.py	problema3.py
--------------	--------------	--------------	--------------	--------------

```

1  import threading as th
2
3  indice = -1
4  buffer = [None, None, None, None, None]
5
6  def productor():
7      for n in range(20):
8          global indice
9          global buffer
10         item = n*n
11         if indice < 5:
12             indice += 1
13             buffer[indice]=item
14             print(f"productor {indice} {item} {buffer}", flush=True)
15
16
17  def consumidor():
18      item = None
19      for _ in range(20):
20          global indice
21          global buffer
22          if indice > -1:
23              item = buffer[indice]
24              buffer[indice]=None
25              print(f"consumidor {indice} {item} {buffer}", flush=True)
26              indice -= 1
27
28  t=th.Thread(target=productor)
29  t.start()
30  t=th.Thread(target=consumidor)
31  t.start()
32

```

```

alejandro@abdebian:Concurrencia$ python3 problema3.py
productor 0 0 [0, None, None, None, None]
productor 1 1 [None, 1, None, None, None]
consumidor 0 0 [None, None, None, None, None]
productor 2 4 [None, 1, 4, None, None]
productor 2 9 [None, None, 9, None, None]
consumidor 1 1 [None, None, 4, None, None]
productor 3 16 [None, None, 9, 16, None]
consumidor 2 9 [None, None, None, 16, None]
productor 3 25 [None, None, None, 25, None]
consumidor 2 None [None, None, None, 25, None]
productor 3 36 [None, None, None, 36, None]
consumidor 2 None [None, None, None, 36, None]
productor 3 49 [None, None, None, 49, None]
consumidor 2 None [None, None, None, 49, None]
productor 3 64 [None, None, None, 64, None]
consumidor 2 None [None, None, None, 64, None]
productor 3 81 [None, None, None, 81, None]
consumidor 2 None [None, None, None, 81, None]
productor 3 100 [None, None, None, 100, None]
consumidor 2 None [None, None, None, 100, None]
productor 3 121 [None, None, None, 121, None]
consumidor 2 None [None, None, None, 121, None]
productor 3 144 [None, None, None, 144, None]
consumidor 2 None [None, None, None, 144, None]
productor 3 169 [None, None, None, 169, None]
consumidor 2 None [None, None, None, 169, None]
productor 3 196 [None, None, None, 196, None]
consumidor 2 None [None, None, None, 196, None]
productor 3 225 [None, None, None, 225, None]
consumidor 2 None [None, None, None, 225, None]
productor 3 256 [None, None, None, 256, None]
consumidor 2 None [None, None, None, 256, None]
productor 3 289 [None, None, None, 289, None]
consumidor 2 None [None, None, None, 289, None]
productor 3 324 [None, None, None, 324, None]
consumidor 2 None [None, None, None, 324, None]
productor 3 361 [None, None, None, 361, None]
consumidor 2 None [None, None, None, 361, None]
consumidor 1 None [None, None, None, 361, None]
consumidor 0 None [None, None, None, 361, None]
alejandro@abdebian:Concurrencia$

```

```

alejandro@abdebian:Concurrencia$ python3 problema3.py
productor 0 0 [0, None, None, None, None]
productor 1 1 [0, 1, None, None, None]
productor 2 4 [0, None, 4, None, None]
productor 3 9 [0, None, 4, 9, None]
productor 4 16 [0, None, 4, 9, 16]
consumidor 1 1 [0, None, None, None, None]
consumidor 4 16 [0, None, 4, 9, None]
consumidor 3 9 [0, None, 4, None, None]
consumidor 2 4 [0, None, None, None, None]
consumidor 1 None [0, None, None, None, None]
consumidor 0 0 [None, None, None, None, None]
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.7/threading.py", line 917, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.7/threading.py", line 865, in run
    self._target(*self._args, **self._kwargs)
  File "problema3.py", line 13, in productor
    buffer[indice]=item
IndexError: list assignment index out of range

```

Se puede observar que el comportamiento es errático, y en algunos casos produce un error al querer acceder fuera del rango de la lista. Por ejemplo en la primera salida, en la cuarta línea el productor coloca 4 en la posición 2, pero también coloca el 4, en la misma posición, sin que se haya consumido. Entre otros comportamientos errados.

### Solución 3 - Empleando semáforos

problema1.py	solucion1.py	problema2.py	solucion2.py	problema3.py	solucion3.py
<pre> 1  import threading as th 2 3  indice = -1 4  buffer = [None, None, None, None, None] 5 6  def productor(): 7      for n in range(20): 8          global indice 9          global buffer 10         item = n*n 11         libre.acquire() 12         mutex.acquire() 13         indice += 1 14         buffer[indice]=item 15         print(f"productor {indice} {item} {buffer}", flush=True) 16         mutex.release() 17         ocupado.release() 18 19  def consumidor(): 20      item = None 21      for _ in range(20): 22          global indice 23          global buffer 24          ocupado.acquire() 25          mutex.acquire() 26          item = buffer[indice] 27          buffer[indice]=None 28          print(f"consumidor {indice} {item} {buffer}", flush=True) 29          indice -= 1 30          mutex.release() 31          libre.release() 32 33  libre = th.Semaphore(value=5) 34  ocupado = th.Semaphore(value=0) 35  mutex = th.Semaphore(value=1) 36 37  t=th.Thread(target=productor) 38  t.start() 39  t=th.Thread(target=consumidor) 40  t.start() </pre>					

```

alejandro@abdebian:Concurrency$ python3 solucion3.py
productor 0 0 [0, None, None, None, None]
productor 1 1 [0, 1, None, None, None]
productor 2 4 [0, 1, 4, None, None]
productor 3 9 [0, 1, 4, 9, None]
productor 4 16 [0, 1, 4, 9, 16]
consumidor 4 16 [0, 1, 4, 9, None]
consumidor 3 9 [0, 1, 4, None, None]
consumidor 2 4 [0, 1, None, None, None]
consumidor 1 1 [0, None, None, None, None]
consumidor 0 0 [None, None, None, None, None]
productor 0 25 [25, None, None, None, None]
productor 1 36 [25, 36, None, None, None]
productor 2 49 [25, 36, 49, None, None]
productor 3 64 [25, 36, 49, 64, None]
productor 4 81 [25, 36, 49, 64, 81]
consumidor 4 81 [25, 36, 49, 64, None]
consumidor 3 64 [25, 36, 49, None, None]
consumidor 2 49 [25, 36, None, None, None]
consumidor 1 36 [25, None, None, None, None]
consumidor 0 25 [None, None, None, None, None]
productor 0 100 [100, None, None, None, None]
productor 1 121 [100, 121, None, None, None]
productor 2 144 [100, 121, 144, None, None]
productor 3 169 [100, 121, 144, 169, None]
productor 4 196 [100, 121, 144, 169, 196]
consumidor 4 196 [100, 121, 144, 169, None]
consumidor 3 169 [100, 121, 144, None, None]
consumidor 2 144 [100, 121, None, None, None]
consumidor 1 121 [100, None, None, None, None]
consumidor 0 100 [None, None, None, None, None]
productor 0 225 [225, None, None, None, None]
productor 1 256 [225, 256, None, None, None]
productor 2 289 [225, 256, 289, None, None]
productor 3 324 [225, 256, 289, 324, None]
productor 4 361 [225, 256, 289, 324, 361]
consumidor 4 361 [225, 256, 289, 324, None]
consumidor 3 324 [225, 256, 289, None, None]
consumidor 2 289 [225, 256, None, None, None]
consumidor 1 256 [225, None, None, None, None]
consumidor 0 225 [None, None, None, None, None]
alejandro@abdebian:Concurrency$

```

### Problema 4

```

1 import threading as th
2
3 contador = [0,0]
4
5 def hilo(num):
6     for i in range(1,11):
7         t = num
8         print(f"Hilo {num} valor de {i}",flush=True)
9         other = contador[1-t]
10        contador[t] = other +1
11
12 ths = []
13 for i in range(2):
14     t=th.Thread(target=hilo, args=(i,))
15     ths.append(t)
16
17 for i in range(2):
18     ths[i].start()
19
20 for i in range(2):
21     ths[i].join()
22
23 print(f"contador de hilo 0 {contador[0]}")
24 print(f"contador de hilo 1 {contador[1]}")

```



El objetivo en este programa es que cada hilo  $i$ , tuviera a `contador[i]` como su contador de vueltas, el que está marcado por el lazo `for`. Pero la cuenta se hace en función del contador del otro hilo y esto funciona sólo si la ejecución es estrictamente alternante. Es decir uno a uno. De lo contrario los resultados serán inesperados, observe abajo dos ejecuciones:

```
alejandro@abdebien:Concurrencia$ python3 problema4.py
Hilo 0 valor de 1
Hilo 0 valor de 2
Hilo 0 valor de 3
Hilo 0 valor de 4
Hilo 0 valor de 5
Hilo 0 valor de 6
Hilo 1 valor de 1
Hilo 1 valor de 2
Hilo 1 valor de 3
Hilo 1 valor de 4
Hilo 1 valor de 5
Hilo 0 valor de 7
Hilo 1 valor de 6
Hilo 1 valor de 7
Hilo 0 valor de 8
Hilo 1 valor de 8
Hilo 0 valor de 9
Hilo 1 valor de 9
Hilo 0 valor de 10
Hilo 1 valor de 10
contador de hilo 0 9
contador de hilo 1 10
```

```
alejandro@abdebien:Concurrencia$ python3 problema4.py
Hilo 0 valor de 1
Hilo 0 valor de 2
Hilo 0 valor de 3
Hilo 0 valor de 4
Hilo 0 valor de 5
Hilo 0 valor de 6
Hilo 0 valor de 7
Hilo 0 valor de 8
Hilo 1 valor de 1
Hilo 1 valor de 2
Hilo 1 valor de 3
Hilo 1 valor de 4
Hilo 1 valor de 5
Hilo 1 valor de 6
Hilo 1 valor de 7
Hilo 1 valor de 8
Hilo 1 valor de 9
Hilo 0 valor de 9
Hilo 0 valor de 10
Hilo 1 valor de 10
contador de hilo 0 5
contador de hilo 1 6
```

## Solución 4 – Empleando Barreras

problema1.py	solucion1.py	problema2.py	solucion2.py	problema3.py	solucion3.py	problema4.py	solucion4.py
--------------	--------------	--------------	--------------	--------------	--------------	--------------	--------------

```
1 import threading as th
2
3 contador = [0,0]
4
5 def hilo(num):
6     for i in range(1,11):
7         t = num
8         print(f"Hilo {num} valor de {i}",flush=True)
9         other = contador[1-t]
10        b1.wait()
11        contador[t] = other + 1
12        b2.wait()
13
14 b1 = th.Barrier(2)
15 b2 = th.Barrier(2)
16
17 ths = []
18 for i in range(2):
19     t=th.Thread(target=hilo, args=(i,))
20     ths.append(t)
21
22 for i in range(2):
23     ths[i].start()
24
25 for i in range(2):
26     ths[i].join()
27
28 print(f"contador de hilo 0 {contador[0]}")
29 print(f"contador de hilo 1 {contador[1]}")
```

```
alejandro@abdebian:Concurrencia$ python3 solucion4.py
Hilo 0 valor de 1
Hilo 1 valor de 1
Hilo 0 valor de 2
Hilo 1 valor de 2
Hilo 0 valor de 3
Hilo 1 valor de 3
Hilo 0 valor de 4
Hilo 1 valor de 4
Hilo 0 valor de 5
Hilo 1 valor de 5
Hilo 1 valor de 6
Hilo 0 valor de 6
Hilo 1 valor de 7
Hilo 0 valor de 7
Hilo 1 valor de 8
Hilo 0 valor de 8
Hilo 1 valor de 9
Hilo 0 valor de 9
Hilo 1 valor de 10
Hilo 0 valor de 10
contador de hilo 0 10
contador de hilo 1 10
```

### Tarea

- 1) Resuelva algunos de los problemas presentados anteriormente con variables de condición (vea enlace, al inicio de este material).
- 2) Algunas herramientas son más apropiadas que otras para resolver algún tipo de problema, sin embargo esto no significa que haya solución única. Resuelva los problemas clásicos (problema del filósofo, problema del barbero dormilón, problema lectores-escriitores) empleando las diferentes herramientas que proporciona Python. También puede hacer uso del libro The Little Book of Semaphores, 2016 by A. Downey, que se encuentra en Intranet (Bibliografía)