

# ARQUITECTURA Y SISTEMA OPERATIVO

Tecnicatura Universitaria en Programación (TUP) – UTN FRBB

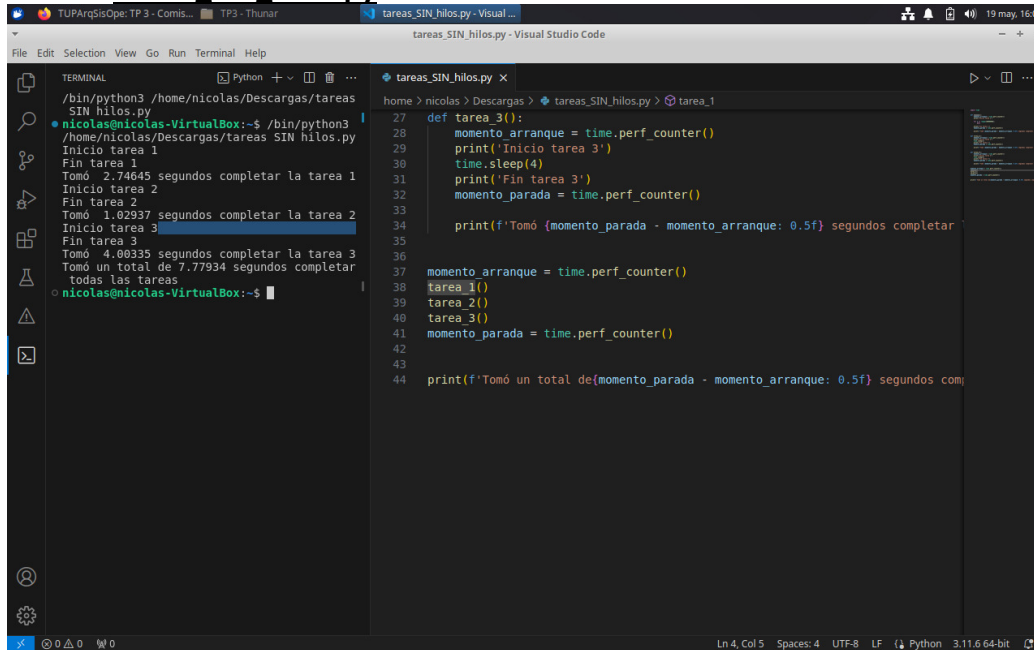
Profesor: Gustavo Ramoscelli

Ayudantes: Leandro M. Regolf - Sergio Antozzi

## Trabajo Práctico Nº 3

Procesos multihilados (multi-threads)

- **tareas SIN hilos.py**



```
terminal
/home/nicolas/Descargas/tareas SIN hilos.py
nicolas@nicolas-VirtualBox:~/Descargas$ python3 tareas SIN hilos.py
Inicio tarea 1
Fin tarea 1
Tomó 2.74645 segundos completar la tarea 1
Inicio tarea 2
Fin tarea 2
Tomó 1.02937 segundos completar la tarea 2
Inicio tarea 3
Fin tarea 3
Tomó 4.00335 segundos completar la tarea 3
Tomó un total de 7.77934 segundos completar todas las tareas
nicolas@nicolas-VirtualBox:~/Descargas$

tareas_SIN_hilos.py
27 def tarea_3():
28     momento_arranque = time.perf_counter()
29     print('Inicio tarea 3')
30     time.sleep(4)
31     print('Fin tarea 3')
32     momento_parada = time.perf_counter()
33
34     print(f'Tomó {momento_parada - momento_arranque: 0.5f} segundos completar')
35
36
37 momento_arranque = time.perf_counter()
38 tarea_1()
39 tarea_2()
40 tarea_3()
41 momento_parada = time.perf_counter()
42
43
44 print(f'Tomó un total de {momento_parada - momento_arranque: 0.5f} segundos completar todas las tareas')
```

-¿Qué se puede notar con respecto al tiempo de ejecución? ¿Es predecible?

Los tiempos con respecto a la ejecución van variando respecto a la maquina. los tiempo de tarea\_2 y tarea\_3 son predecibles ya que contando los segundo logra ejecutarse a tiempo, en cambio la ejecución de la tarea\_1 puede termina siendo impredecible, al ejecutar la mayoría de las veces dicha tarea aveces tarda 2 segundo y otras veces puede tardar entre 4 a 5 segundo según el funcionamiento de la maquina.

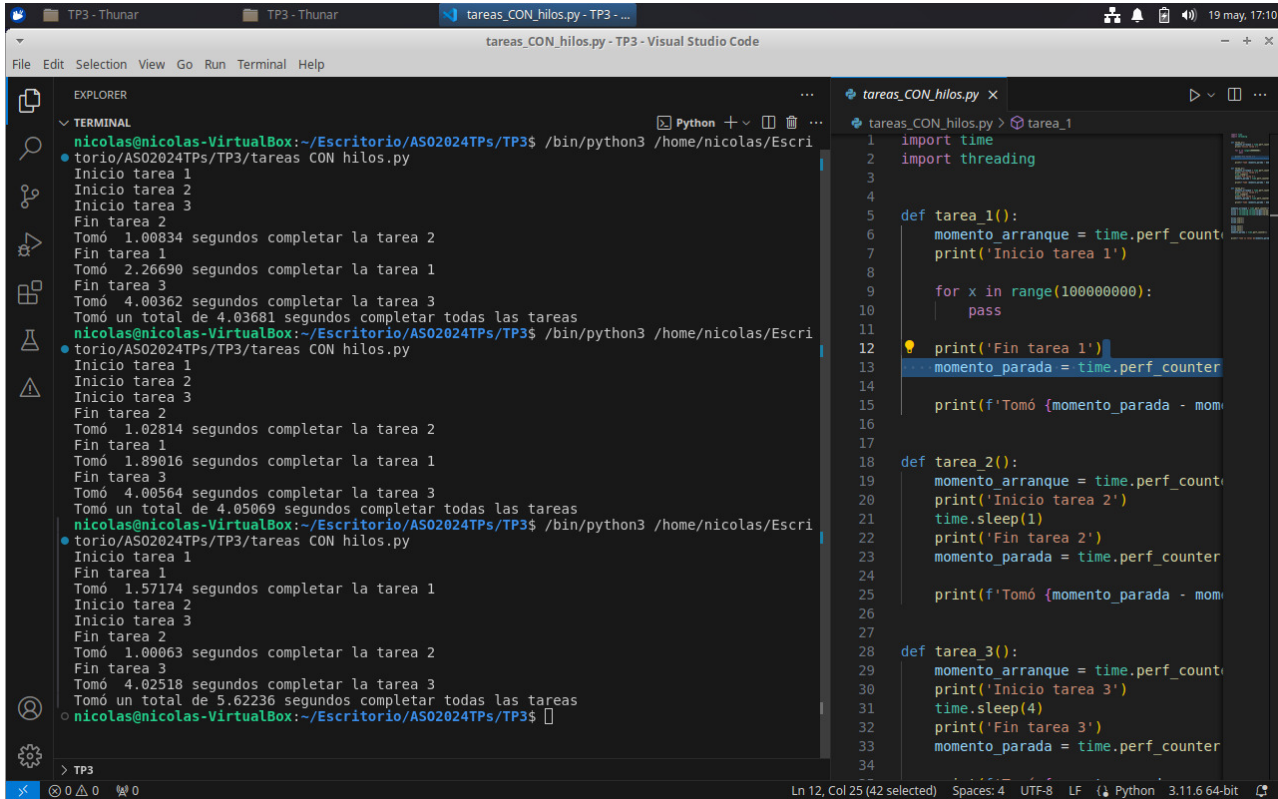
-Nombrar un proceso o función de la vida real que pueden ser considerados procesos de “máxima velocidad posible” que dependen casi exclusivamente de la velocidad de la máquina que los ejecuta (ej.Ordenar una lista)

Un proceso que puede ser considerado de máxima velocidad posible que solo dependa de la maquina que los ejecute seria el de abrir un archivo o carpeta que se encuentre dentro de una computadora o el uso exclusivo de una aplicación dentro de un teléfono celular. tanto el uso de una aplicación o el ingreso a un archivo o carpeta dependerán de la velocidad máxima que pueda ofrecer ambas maquinas.

-Nombrar un proceso o función de la vida real que pueden ser considerados procesos de “velocidad de respuesta no dependiente de la velocidad de procesamiento” o que sea de naturaleza impredecible o externa (ej. Leer un archivo externo).

Un proceso o función de la vida real que pueda ser considerado proceso de velocidad de respuesta no dependiente de la velocidad de procesamiento seria el de la conexión a Internet. este mismo dependerá de no solo de la condición de red, sino también, de la distancia, las interferencias, los problemas de los proveedores,etc.

## tareas\_CON\_hilos.py



The screenshot shows a Visual Studio Code window with a Python file named `tareas_CON_hilos.py` and its terminal output. The terminal shows the execution of the script multiple times, demonstrating the execution time of three tasks (tarea 1, tarea 2, tarea 3) and the total time taken. The output shows that the tasks are executed in parallel, and the total time is significantly reduced compared to sequential execution.

```
1 import time
2 import threading
3
4
5 def tarea_1():
6     momento_arranque = time.perf_counter()
7     print('Inicio tarea 1')
8
9     for x in range(100000000):
10         pass
11
12     print('Fin tarea 1')
13     momento_parada = time.perf_counter()
14
15     print(f'Tomó {momento_parada - momento_arranque} segundos completar la tarea 1')
16
17 def tarea_2():
18     momento_arranque = time.perf_counter()
19     print('Inicio tarea 2')
20     time.sleep(1)
21     print('Fin tarea 2')
22     momento_parada = time.perf_counter()
23
24     print(f'Tomó {momento_parada - momento_arranque} segundos completar la tarea 2')
25
26 def tarea_3():
27     momento_arranque = time.perf_counter()
28     print('Inicio tarea 3')
29     time.sleep(4)
30     print('Fin tarea 3')
31     momento_parada = time.perf_counter()
32
33     print(f'Tomó {momento_parada - momento_arranque} segundos completar la tarea 3')
34
35 if __name__ == '__main__':
36     t1 = threading.Thread(target=tarea_1)
37     t2 = threading.Thread(target=tarea_2)
38     t3 = threading.Thread(target=tarea_3)
39
40     t1.start()
41     t2.start()
42     t3.start()
43
44     t1.join()
45     t2.join()
46     t3.join()
47
48     momento_arranque = time.perf_counter()
49     print('Inicio tareas')
50
51     for x in range(100000000):
52         pass
53
54     momento_parada = time.perf_counter()
55     print(f'Tomó {momento_parada - momento_arranque} segundos completar todas las tareas')
```

Terminal output (first run):

```
Inicio tarea 1
Inicio tarea 2
Inicio tarea 3
Fin tarea 2
Tomó 1.00834 segundos completar la tarea 2
Fin tarea 1
Tomó 2.26690 segundos completar la tarea 1
Fin tarea 3
Tomó 4.00362 segundos completar la tarea 3
Tomó un total de 4.03681 segundos completar todas las tareas
```

Terminal output (second run):

```
Inicio tarea 1
Inicio tarea 2
Inicio tarea 3
Fin tarea 2
Tomó 1.02814 segundos completar la tarea 2
Fin tarea 1
Tomó 1.89016 segundos completar la tarea 1
Fin tarea 3
Tomó 4.00564 segundos completar la tarea 3
Tomó un total de 4.05069 segundos completar todas las tareas
```

Terminal output (third run):

```
Inicio tarea 1
Inicio tarea 2
Inicio tarea 3
Fin tarea 2
Tomó 1.57174 segundos completar la tarea 1
Fin tarea 1
Tomó 1.00063 segundos completar la tarea 2
Fin tarea 3
Tomó 4.02518 segundos completar la tarea 3
Tomó un total de 5.62236 segundos completar todas las tareas
```

-Ejecutar varias veces el código

-¿Qué se puede notar con respecto al tiempo de ejecución? ¿Se mejoró el tiempo de respuesta con respecto al mismo programa sin hilos?

*el tiempo de ejecución es un poco mas rapido ya que logro mejorar el tiempo de respuestas con respecto al mismo programa sin hilos.*

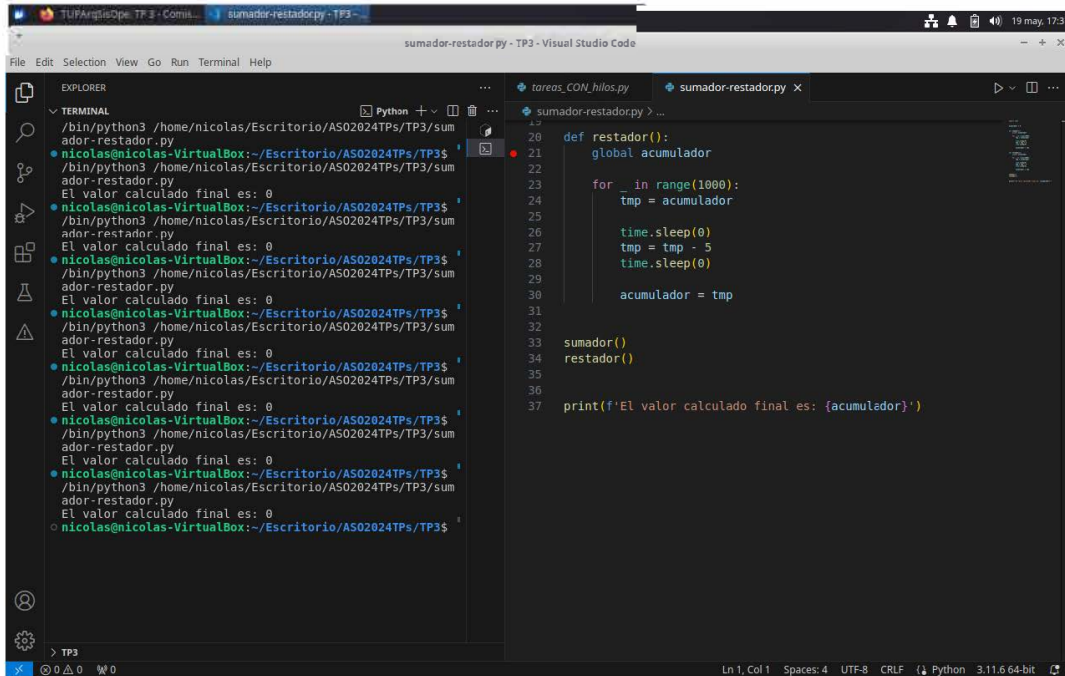
-¿Completan las funciones su ejecución en el orden establecido?

*Las funciones de su ejecución son completadas, pero su orden establecido no. ya que se ejecuta en orden diferente.*

-Nombrar un escenario real donde el multi-hilado puede mejorar considerablemente el tiempo de respuesta de un sistema (ej. Carga de una página WEB en un navegador)

*Un escenario real donde el multi-hilado puede mejorar es en las ediciones de vídeos. ya que podemos optimizar el procesamiento en tiempo real como también las operaciones de exportación.*

## Sumador-Restador.py



The screenshot shows the Visual Studio Code interface with the file `sumador-restador.py` open. The code defines a `restador()` function that uses a global `acumulador` variable. It iterates 1000 times, sleeping for 0 seconds, then subtracting 5 from the accumulator and sleeping again for 0 seconds. The `sumador()` function calls `restador()` and prints the final value of the accumulator.

```
def restador():
    global acumulador

    for _ in range(1000):
        tmp = acumulador

        time.sleep(0)
        tmp = tmp - 5
        time.sleep(0)

        acumulador = tmp

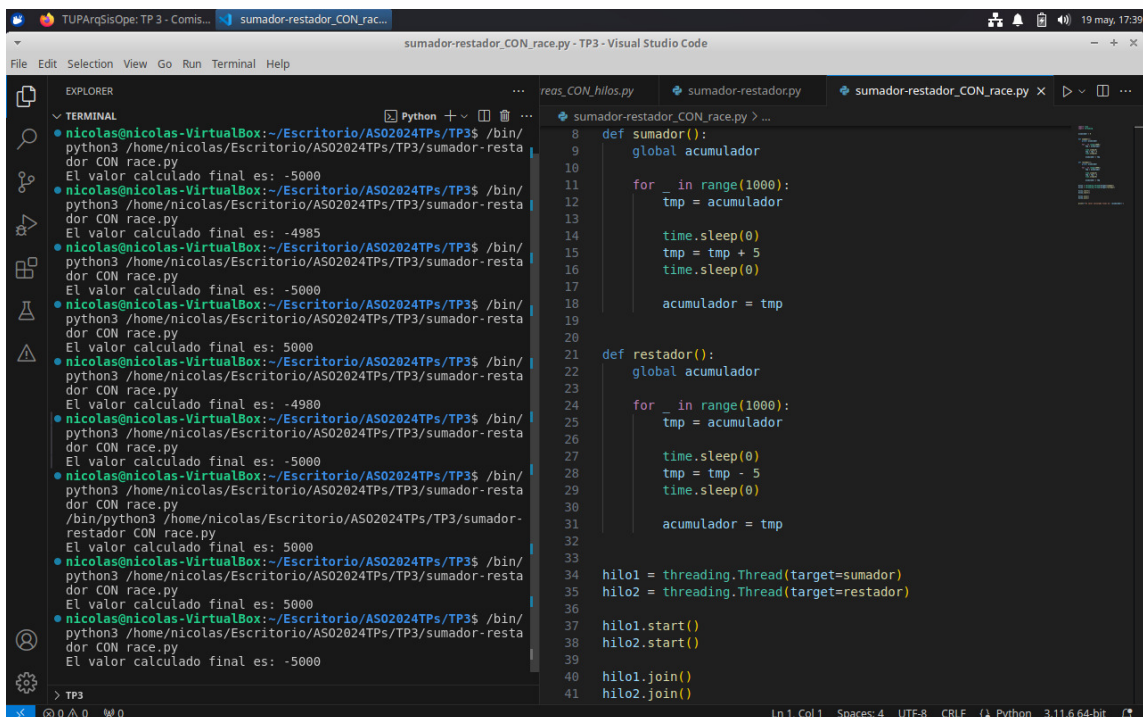
sumador()
restador()

print(f'El valor calculado final es: {acumulador}')
```

The terminal output shows the execution of the script, displaying the final value of the accumulator as 0.

Ejecutar varias veces el código: *Este código fue ejecutado un total de 7 veces.*

## sumador-restador\_CON\_race.py



The screenshot shows the Visual Studio Code interface with the file `sumador-restador_CON_race.py` open. The code defines two functions: `sumador()` and `restador()`. Both functions use a global `acumulador` variable. The `sumador()` function iterates 1000 times, sleeping for 0 seconds, then adding 5 to the accumulator and sleeping again for 0 seconds. The `restador()` function iterates 1000 times, sleeping for 0 seconds, then subtracting 5 from the accumulator and sleeping again for 0 seconds. The script creates two threads, `hilo1` and `hilo2`, which run `sumador()` and `restador()` respectively. The script then joins the threads and prints the final value of the accumulator.

```
def sumador():
    global acumulador

    for _ in range(1000):
        tmp = acumulador

        time.sleep(0)
        tmp = tmp + 5
        time.sleep(0)

        acumulador = tmp

def restador():
    global acumulador

    for _ in range(1000):
        tmp = acumulador

        time.sleep(0)
        tmp = tmp - 5
        time.sleep(0)

        acumulador = tmp

hilo1 = threading.Thread(target=sumador)
hilo2 = threading.Thread(target=restador)

hilo1.start()
hilo2.start()

hilo1.join()
hilo2.join()
```

The terminal output shows the execution of the script, displaying the final value of the accumulator as 5000.

-Ejecutar varias veces el código

-¿Qué se puede notar con respecto al tiempo de ejecución?

*respecto al tiempo de ejecución se puede notar que el tiempo es demasiado rápido.*

-¿Qué sucede con el valor final del acumulador?

*Al final del acumulador el valor varia entre 5000 a -5000, pero también sin previo aviso puede cambiar como vemos en la imagen que paso de -5000 a -4985 y de 5000 a -4980.*

-¿Por sucede esto?

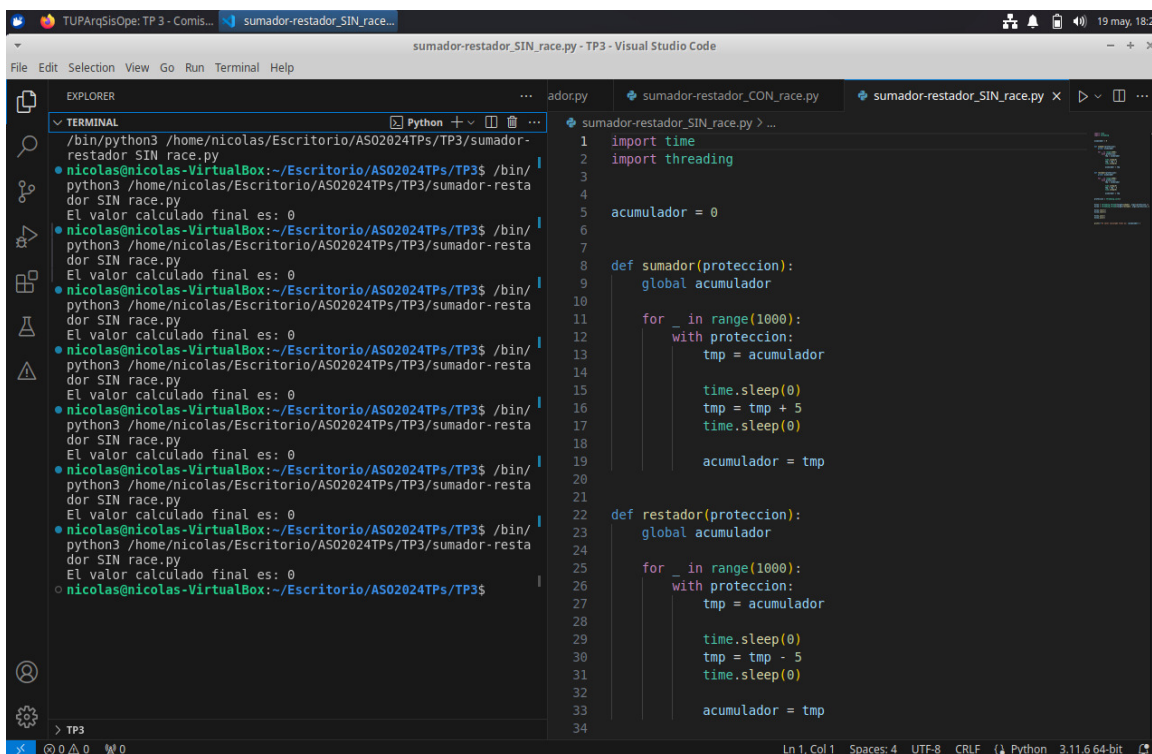
*Esto sucede porque los hilos no se encuentran sincronizados y acceden al valor en cualquier momento y es por eso que los valores son diferentes todo el tiempo.*

**TIP: cambios de contexto en medio de acceso a zona crítica y actualización de valor de variable**

-¿Cómo se puede corregir esta condición de carrera sin dejar de utilizar hilos?

*Para corregir una condición de carrera sin dejar de utilizar hilos, considera usar técnicas de programación concurrente como transacciones atómicas, estructuras de datos inmutables, colas de comunicación, bloqueos de lectura/escritura y mecanismos de entronización.*

*sumador-restador\_SIN\_race.py*



```
sumador-restador_SIN_race.py
1 import time
2 import threading
3
4 acumulador = 0
5
6 def sumador(proteccion):
7     global acumulador
8     for _ in range(1000):
9         with proteccion:
10             tmp = acumulador
11             time.sleep(0)
12             tmp = tmp + 5
13             time.sleep(0)
14             acumulador = tmp
15
16 def restador(proteccion):
17     global acumulador
18     for _ in range(1000):
19         with proteccion:
20             tmp = acumulador
21             time.sleep(0)
22             tmp = tmp - 5
23             time.sleep(0)
24             acumulador = tmp
25
26 if __name__ == '__main__':
27     proteccion = threading.Lock()
28     t1 = threading.Thread(target=sumador, args=(proteccion,))
29     t2 = threading.Thread(target=restador, args=(proteccion,))
30     t1.start()
31     t2.start()
32     t1.join()
33     t2.join()
34     print("El valor calculado final es: ", acumulador)
```

-Ejecutar varias veces el código

-¿Qué sucede con el valor final del acumulador?

*Lo que sucede con el valor final del acumulador es que nos da un total de 0.*

-¿Qué se puede notar con respecto al tiempo de ejecución?

*Respecto al tiempo de ejecución esta es mas tardía.*