

# UT 2 - Multihilos

Programación de Servicios y Procesos  
Curso 2025-26

Profesor: Agustín González-Quel



Material bajo licencia Creative Commons Reconocimiento-NoComercial-SinObraDerivada



# Procesos e hilos

- Como hemos visto, a nivel del SO el proceso es la unidad mínima de gestión de recursos de ejecución.
- Sin embargo, desde hace años los SO y los lenguajes de programación han incorporado mecanismos para la ejecución de **procesos ligeros** o **hilos** (*multithreading*)

## Qué es multithreading

El *multithreading* es un modelo de ejecución de programas que permite la creación de múltiples hilos dentro de un proceso, que se ejecutan de forma independiente pero que comparten simultáneamente los recursos del proceso. Dependiendo del hardware, los hilos pueden ejecutarse totalmente en paralelo si se distribuyen en su propio núcleo de la CPU.

## Ejemplos

- procesosOO.py vs hilosOO.py

# Programación multihilo

## Ventajas

- Explotan las capacidades multi-core de los procesadores modernos.
- Ofrece al usuario una sensación de paralelismo absoluta
  - Descarga de un fichero mientras navegas
- Mejora el consumo de recursos por cambio de contexto más eficiente.

## Inconvenientes

- La programación es más compleja
  - Depurado de programas multihilo es más complicada
  - La sincronización mal resuelta puede bloquear el programa.
- Comparten los recursos del proyecto padre (no siempre ...)
  - 4 hilos de un proceso tienen los mismos recursos que 1 hilo de un proceso.

# Cómo se implementa

- El **sistema operativo** debe tener soporte para la ejecución de multi-hilos de proceso
  - Aunque parte de la información es común al proceso, es necesario que el SO identifique qué segmentos de programa hay que ejecutar en cada momento.
- El **lenguaje de programación**, debe ofrecer mecanismos necesarios
- Desarrolladores de programas,
  - Diseño del sistema
    - Identificando en qué partes es necesario/beneficioso implementar concurrencia.
    - Estableciendo mecanismos de sincronización, recursos compartidos, bloqueo, etc.
  - Programación
    - Implementación y depurado
    - Sobre todo, depurado.

# Implementación en Python

## threading

<https://docs.python.org/3/library/threading.html>

- Desde V3.7 ya viene instalada por defecto.

### Características:

- Paralelismo
- No determinismo
- Variables compartidas

```
import threading

def hilo():
    for i in range(10):
        print ('Hola Mundo')

t = threading.Thread(target=hilo)

t.start()
for i in range(10):
    print ("hola hilo")
```

Ejemplo no determinismo : 01-HolaMundo.py

# Clase Thread

- Thread

```
t = threading.Thread(target=hiloFun)
t = threading.Thread(target=hiloFun, args=(3,))
t = threading.Thread(target=hiloFun, kwargs = {'n':3})
t.start()
```

t.start() crea un nuevo hilo

t.run() ejecuta hiloFun sin crear un nuevo hilo

```
t.join()
```

```
t.name
```

```
t.is_alive()
```

```
threading.current_thread()
```

```
threading.active_count()
```

```
threading.enumerate()
```

## Variables locales

- Se crea una instancia con el método local y se almacenan datos en ese espacio:

```
mydata = threading.local()
mydata.x = 1
```

# Espacio de memoria compartido

- Todos los hilos comparten el mismo espacio de memoria.
- Si hay variables globales o variables de clase cualquier hilo puede modificarla.
- Este aspecto puede usarse para sincronizar la ejecución de los hilos

Ejemplo: 02-variableCompartida.py  
03-pasoParametros.py

```
import threading

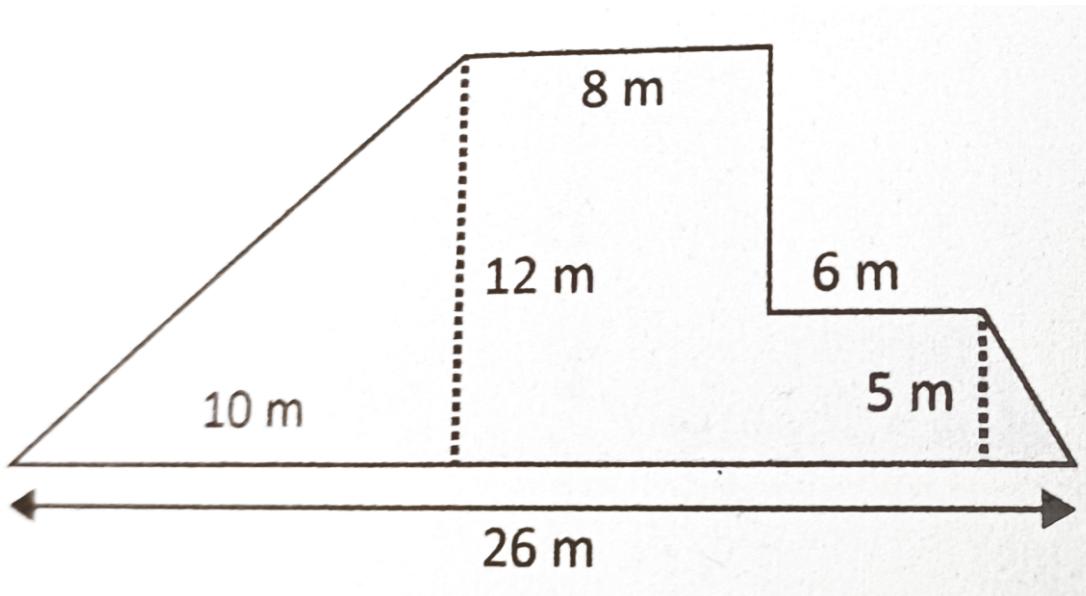
def resto():
    global Contador
    global numIter
    while numIter < 200:
        numIter +=1
        if Contador > 0:
            Contador -= 1
            print("Resto", Contador)
    return

def sumo():
    global Contador
    global numIter
    while numIter < 200:
        numIter +=1
        if Contador < 10 :
            Contador +=1
            print ("Sumo ", Contador)
    return

Contador = 5
numIter = 0
t1 = threading.Thread(target=sumo)
t2 = threading.Thread(target=resto)
t1.start()
t2.start()
```

# Ejercicio

- Área de Polígono: Hacer un programa que calcula el área de la figura descomponiéndola en hilos para optimizar.



# Repaso de ejemplos

- 01- HolaMundo → Ver hilos activos
- 02 – Variable compartida
  - Acceso a dato global
  - Print “thread-safe”  
Es conveniente forzar que se vacíe el buffer de salida del print.
- 03 – pasoParametros.py
- 04 – LocalStorage.py: Almacenamiento global vs local.

```
midato = threading.local()  
midato.x = 10  
midato.name = "Ana"
```

## Cálculo del polígono

- Almacenamiento global
- Almacenamiento local
- Uso de Join

# Sincronización

## Join

- El hilo que ejecuta el join (`current_thread()`) espera hasta que el hilo referido termina.

`t1.join()` espera a que acabe t1

## Lock

- Permite el acceso único a un recurso compartido.
- Implementa el concepto de acceso a una región crítica  
[https://es.wikipedia.org/wiki/Secci%C3%B3n\\_cr%C3%ADtica](https://es.wikipedia.org/wiki/Secci%C3%B3n_cr%C3%ADtica)

### Funcionamiento

- Se crea un lock local (o varios)    **`ml = threading.Lock()`**
- Un hilo adquiere el lock                         **`ml.acquire()`**
  - Accede a la región crítica
  - Libera el lock                                         **`ml.release()`**
- Los demás hilos pueden consultar si está libre o no  
    **`ml.locked()`**    → False: Está libre, podemos adquirirlo  
    → True: lock ocupado

# Semáforos

## Semaphore

- Similar a un lock pero permite que varios hilos de control accedan a la región crítica.
- ¿Cuántos? Los que definamos en su creación

```
mys = threading.Semaphore(4)  
mys.acquire()  
mys.release(n)
```

### Funcionamiento:

- Al crearse el semáforo se inicia un contador con el número dado.
- Cada llamada a acquire:
  - Si el contador es igual a 0, se bloquea el hilo
  - Si es mayor que cero, permite el acceso y decrementa en 1 el contador.
- Cada llamada a release:
  - Incrementa en n el contador.
- Ejemplos:
  - 06-VariableCompartidaLock.py
  - 10-VariableCompartidaSem.py

# Comunicación y sincronización entre hilos: Colas (Queue)

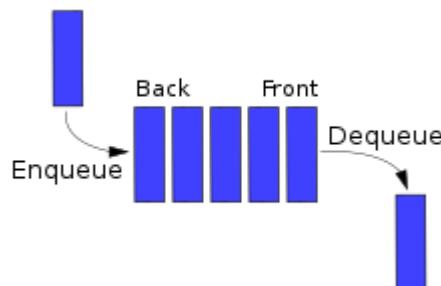
## Colas: Estructura de datos FIFO (First-In / First-Out)

Estándar en Python 3.

<https://docs.python.org/3/library/queue.html>

Ofrece:

- Colas compartidas de forma Segura entre diferentes hilos
- Sincroniza de forma automática el acceso (bloqueo) de productores-consumidores.
- Tres tipos de colas: FIFO, LIFO, con prioridad.
- Existe un modelo simplificado: SimpleQueue



```
import queue
```

```
miCola = queue.Queue()
```

```
miPila = queue.LifoQueue()
```

```
miColaPrio = queue.PriorityQueue()
```

```
miColaSimple = queue.SimpleQueue()
```

Funciones

```
micola.put(elem)
```

```
micola.get()
```

```
micola.qsize()
```

```
micola.empty() / .full()
```

## Fin Unidad 2 - Multihilos