PROGRAMACIÓN DE SERVICIOS Y PROCESOS – FEOE

📦 Tarea: Problema de los Filósofos Comensales

📖 Descripción del Problema

Esta tarea se centra en aplicar los conceptos de programación concurrente usando hilos (threading.Thread) en Python para simular el clásico **problema de la cena de los filósofos**.

Actividad principal

Completa el código proporcionado para implementar una simulación en la que cinco filósofos comparten palillos para comer tallarines. Cada filósofo alterna entre pensar y comer, necesitando ambos palillos para hacerlo.

- Tu programa debe cumplir con lo siguiente:Utilizar clases que hereden de *threading.Thread*.
- Gestionar los palillos con *threading.Lock*.
- Evitar condiciones de carrera e interbloqueos mediante una estrategia correcta de adquisición de recursos.
- Mostrar por consola mensajes que indiquen qué está haciendo cada filósofo (pensando, con hambre, comiendo).

🧩 Código base para completer.

```
import threading
import time
import random

class Filosofo(threading.Thread):
    def __init__(self, nombre, palillo_izq, palillo_der):
        super().__init__()
        self.nombre = nombre
        self.palillo_izq = palillo_izq
        self.palillo_der = palillo_der
    def run(self):
        while True:
            print(f"{self.nombre} está pensando.")
            time.sleep(random.uniform(1, 3))
            print(f"{self.nombre} tiene hambre.")
            self.comer()
    def comer(self):
        # COMPLETAR: Coger primero un palillo, luego el otro (usa with)
        # Asegúrate de evitar interbloqueo usando un orden consistente de adquisición
        pass
N = 5
palillos = [threading.Lock() for _ in range(N)]
filosofos = [
    Filosofo (f"Filósofo {i}", palillos[i], palillos[(i + 1) % N])
    for i in range(N)
]
for f in filosofos:
    f.start()
```

📊 Criterios de Calificación: Total 10 puntos

Criterio	Descripción	Puntos
Comprensión del problema	Describe adecuadamente el comportamiento de los filósofos.	1
Uso de hilos	Implementación de threading.Thread correctamente.	1
Sincronización	Uso adecuado de threading.Lock para los palillos.	1.5
Prevención de deadlock	Se implementa una estrategia para evitar bloqueos mutuos.	2
Simulación funcional	Los filósofos alternan correctamente entre pensar y comer.	2
Calidad del código	Código legible, ordenado y comentado.	1
Ejecución sin errores	El programa se ejecuta de forma estable.	1.5

🧰 Recursos necesarios

- Python 3.10 o superior.
- Editor de código (VSCode, PyCharm, Thonny, etc.).
- Conocimientos básicos de threading y sincronización.

✨ Consejos y Recomendaciones

- Lee bien el problema antes de comenzar.
- Recuerda que los bloqueos deben adquirirse en orden para evitar interbloqueos.
- Usa with para trabajar con Lock de forma segura.
- Añade print() para ver qué hace cada hilo.
- Puedes utilizar cualquier IA para que te ayude a la resolución del problema.

🛠️ Indicaciones de entrega

- Debes entregar un fichero comprimido (.zip) que contenga:
- El código fuente completo y funcional.
- Un documento (ODT, DOCX o PDF) con una breve explicación de la solución y el funcionamiento de la misma.
- Capturas de pantalla de la ejecución.
- El archivo comprimido debe seguir esta nomenclatura:

Resolución

Para la resolución del problema de los 5 filósofos, se ha planteado la siguiente estructura.

Una carpeta que contendrá una clase para representar el hilo Filósofo. En esta clase se agregará el constructor y los métodos necesarios para pensar, tener hambre y esta comiendo.

También, se desarrollará un main para ejecutar la aplicación y crear los hilos filósofos.

Clase filosofos.py

Contructor __init__

En esta clase se va a establecer un constructor que inicializará el objeto filósofo

- **Nombre:** este parámetro representará al nombre del filósofo
- **Palillo_izq_id:** recibirá el palillo izquierdo para poder comer.
- **Palillo_der_id:** recibirá el palillo derecho para poder comer.
- **Palillos:** lista compartida de objetos Look entre los filósofos.
- **Comidas:** se inicializará en cero, y contabilizará las veces que se ha comida.
- **Ejecutando:** variable para controla la parada del hilo de forma controlada.

```
def __init__(self, nombre, palillo_izq_id, palillo_der_id, palillos):
    """
    Constructor de la clase Filosofo.
    Args:
        nombre (str): Nombre identificativo del filósofo
        palillo_izq_id (int): Índice del palillo izquierdo
        palillo_der_id (int): Índice del palillo derecho
        palillos (list): Lista que se comparte entre los hilos de objetos Lock que representan a los palillos
    """
    super().__init__()
    self.nombre = nombre
    self.palillo_izq_id = palillo_izq_id
    self.palillo_der_id = palillo_der_id
    self.palillos = palillos
    self.comidas = 0
    # se contabilizará las veces que comen
    self.ejecutando = True
    # variable para controlar la parada del hilo de forma controlada.
```

Método detener

Con este método se detiene de forma segura la ejecución del filósofo desde el exterior. Permitirá cambiar el valor de la variable ejecutando a false. Esta se verifica en el bucle principal del método run.

```
def detener(self):
    #Método para detener la ejecución del filósofo de forma segura
    self.ejecutando = False
```

Método pensar

Este método genera el tiempo de pensamiento del filósofo, podrá pensar entre 1 y 3 segundos. esto se simula congelando el hilo, durante el resultado aleatorio entre 1 y 2.

```
def pensar(self):
    #Simula el tiempo que el filósofo pasa pensando
    print(f"{self.nombre} está pensando... 🤔")
    time.sleep(random.uniform(1, 3))
```

Método comer

Este método simulará la acción de comer del filósofo, además se asegurará de que coja correctamente un palillo sin producir bloqueos mutuos.

Al iniciar el proceso, se informará de que tiene hambre. Se organizarán los palillos para coger siempre el primero con el id más bajo. Lo que garantizará que todos los filósofos sigan el mismo orden de adquisición y evita bloqueos.

Una vez seleccionados se bloquearán los palillos que ha seleccionado el filósofo. Informará que ha cogido los dos palillos y que esta comiendo. Se congelará el hilo entre 1 y 3 segundos, para simular el tiempo que tarda en comer.

Por último, se suma a la variable comer un, para contabilizar las veces que come, y se finaliza con un mensaje de que ha comido.

```python
def comer(self):
    # Método para simular que el filósofo esta comiendo
    print(f"{self.nombre} tiene hambre y quiere comer... 😋")

    # Ordenamos los palillos por ID para evitar deadlocks
    primer_id = min(self.palillo_izq_id, self.palillo_der_id)
    segundo_id = max(self.palillo_izq_id, self.palillo_der_id)

    # Se obtiene el palillo primer_id, y se bloquea
    with self.palillos[primer_id]:
        print(f"{self.nombre} tomó el palillo {primer_id}")
        time.sleep(0.5)
        # Se espera 0.5 segundos para coger el siguiente palillo

        with self.palillos[segundo_id]:
        # Se obtiene el segundo palillo, y se bloquea
            print(f"{self.nombre} tomó el palillo {segundo_id} y está comiendo 🍝")
            # Se informa de que esta comiendo
            time.sleep(random.uniform(1, 3))
            # Se simula que esta comiendo
            self.comidas += 1
            # Se suma al contador
            print(f"{self.nombre} terminó de comer (comida #{self.comidas})")
            # Se informa que ha teminado de comer

    print(f"{self.nombre} soltó los palillos")
    # Se informa que el filóso solto los
```

Método run

Este método se utiliza para gestionar y definir el ciclo de vida del hilo. Esto hará que se ejecute de forma automática al llamar a .start sobre la instancia de un filósofo.

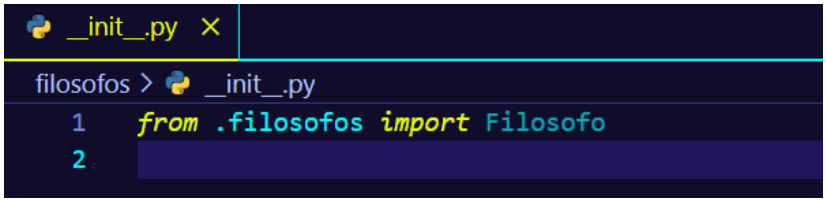
Lo primero que se hará es comprobar que variable ejecutando sea true, esto permitirá alternar entre pesar y comer. Seguidamente, antes de comer, se comprobará que ejecutando sea true, esto permitirá una salida segura si se detiene desde el exterior.

```python
def run(self):
    #Método principal del hilo que alterna entre pensar y comer mientras ejecutando sea True

    while self.ejecutando:
        # Bucle que se ejecuta mientras ejecutando sea true
        self.pensar()
        # se llama al metodo pensar()
        if not self.ejecutando:
        # Si ejecutando es false se para el bucle.
            break
        # Se llama al método.
        self.comer()
```

__init__.py

Como esta clase va dentro de un paquete para el proyecto, y de este modo se tiene mejor organizado el código, en agregado una __init__.py en el paquete para indicar que objetos se pueden compartir desde la clase filosofo, como se va a usar todo, simplemente se indica que se puede compartir el objeto complet.



Main.py

Por último, vas ver el archivo main, que ejecutará la aplicación, y gestionará la conexión de los hilos. También, se ha decidido utilizar la librería keyboard, que se ejecutará un hilo independiente que estará pendiente para cuando el usuario pulse la tecla “esc” escape del teclado. Esta tecla finalizará la aplicación y el proceso de los otros hilos.

Importante. Para poder usar este código, se ha de tener la librería instalada de keyboard. Para instalarla hay que utilizar el siguiente comando. Si se está utilizando Visual Studio, se puede ejecutar un terminal e introducir el siguiente comando “pip install keyboard”. En algunos equipos, debido al sistema operativo, se necesitarán permisos de administrador

En el main, encontraremos un variable global, que controlará si el programa sigue ejecutándose o debe de finalizar. Se comparte con todos los hilos.

Método monitor_teclado

Este método se ejecuta en un hilo independiente, y estará monitoreando la pulsación de la tecla “Esc”. Cuando se pulse la tecla se informará de que se va a cerrar el programa y se cambia el valor de la variable ejecutando, que finalizará los hilos filosofo de la aplicación.

```python
def monitor_teclado():
    # Método para monitorear si se pulsa la tecla "esc" del teclado.
    global ejecutando
    # Se obtiene la variable ejecutando
    keyboard.wait('esc')
    # Cuando se pulsa la tecla "Esc" se finaliza el proceso
    print("")
    print("########################")
    print("Tecla ESC detectada. Finalizando la simulación...")
    print("########################")
    print("")
    # Se informa que se ha pulsado la tecla y se ha finalizado el programa
    ejecutando = False
    # Se cambia el valor de ejecuntando para finalizar los hilos y la aplicación.
```

Método main

Este método contiene la lógica del programa, y creará los hilos filósofos. También ejecutará el hilo que estará a la escucha de si el usuario presiona la tecla “esc”. Cuando el usuario se finalizarán los hilos.

Se definirá en una variable el número de filósofos y palillos que se han de crear. Seguidamente se instanciarán los filósofos.

Se ejecutará el hilo para monitorizar si se pulsa la tecla “esc”. Y también se inicial los hilos de los filósofos.

Ahora ejecutaremos un bucle while que se ejecuta mientras ejecutando sea true. En el momento que este valor cambie se finalizará el proceso. No será de inmediato, porque dependen de los hilos y lo que tarden en acabar sus procesos.

Cuando se finalice se mostrará un mensaje por pantalla informando de que va a finalizar el programa.

```
def main():
    # Método que ejecuta la aplicación
    global ejecutando
    # Variable de control para finalizar el proceso.
    N = 5
    # Número de filósofos y palillos
    palillos = [threading.Lock() for _ in range(N)]
    # Crear los palillos como objetos Lock

    filosofos = [
    # Crear los filósofos
        Filosofo(f"Filósofo {i+1}", i, (i + 1) % N, palillos)
        for i in range(N)
    ]

    threading.Thread(target=monitor_teclado, daemon=True).start()
     # Iniciar el hilo que monitoreará la tecla ESC

    for f in filosofos:
    # Iniciar los hilos de los filósofos
        f.start()

    while ejecutando:
        # Mantener el programa principal ejecutándose hasta que se presione ESC
        time.sleep(1)

    print("Deteniendo filósofos...")
    # Detener cada filósofo de forma ordenada
    for f in filosofos:
        f.detener()

    for f in filosofos:
    # Esperar a que todos los hilos terminen (con timeout)
        f.join(timeout=2)

    print("")
    print("########################")
    print("Simulación finalizada.")
    print("########################")
    print("")
    # Se informa de que se ha finalizado la simulación

    exit(0)
    # Se finaliza el proceso
```

Ahora llamamos al método main, y se ejecutará la aplicación.

Ejecución de la aplicación.

A continuación, se muestra el proceso de ejecución de la aplicación.

```
Filósofo 1 está pensando... 🤔
Filósofo 2 está pensando... 🤔
Filósofo 3 está pensando... 🤔
Filósofo 4 está pensando... 🤔
Filósofo 5 está pensando... 🤔
Filósofo 1 tiene hambre y quiere comer... 😋
Filósofo 1 tomó el palillo 0
Filósofo 4 tiene hambre y quiere comer... 😋
Filósofo 4 tomó el palillo 3
Filósofo 2 tiene hambre y quiere comer... 😋
Filósofo 2 tomó el palillo 1
Filósofo 4 tomó el palillo 4 y está comiendo 🍝
Filósofo 2 tomó el palillo 2 y está comiendo 🍝
Filósofo 5 tiene hambre y quiere comer... 😋
Filósofo 3 tiene hambre y quiere comer... 😋
Filósofo 2 terminó de comer (comida #1)
Filósofo 2 soltó los palillos
Filósofo 1 tomó el palillo 1 y está comiendo 🍝
Filósofo 2 está pensando... 🤔
Filósofo 3 tomó el palillo 2
Filósofo 4 terminó de comer (comida #1)
Filósofo 4 soltó los palillos
Filósofo 4 está pensando... 🤔
Filósofo 3 tomó el palillo 3 y está comiendo 🍝
Filósofo 4 tiene hambre y quiere comer... 😋
Filósofo 1 terminó de comer (comida #1)
Filósofo 1 soltó los palillos
Filósofo 1 está pensando... 🤔
Filósofo 5 tomó el palillo 0
Filósofo 3 terminó de comer (comida #1)
Filósofo 3 soltó los palillos
Filósofo 3 está pensando... 🤔
Filósofo 4 tomó el palillo 3
Filósofo 2 tiene hambre y quiere comer... 😋
Filósofo 2 tomó el palillo 1
Filósofo 5 tomó el palillo 4 y está comiendo 🍝
Filósofo 2 tomó el palillo 2 y está comiendo 🍝

########################
Tecla ESC detectada. Finalizando la simulación...
########################

Deteniendo filósofos...
Filósofo 5 terminó de comer (comida #1)
Filósofo 5 soltó los palillos
Filósofo 4 tomó el palillo 4 y está comiendo 🍝
Filósofo 2 terminó de comer (comida #2)
Filósofo 2 soltó los palillos
Filósofo 4 terminó de comer (comida #2)
Filósofo 4 soltó los palillos

########################
Simulación finalizada.
########################
```

Como se puede ver, cuando el usuario pulse la tecla “esc” se muestra un mensaje informando que se ha pulsado la tecla, serán finalizando los filósofos, y por último, se informa de la finalización de la apliación.