



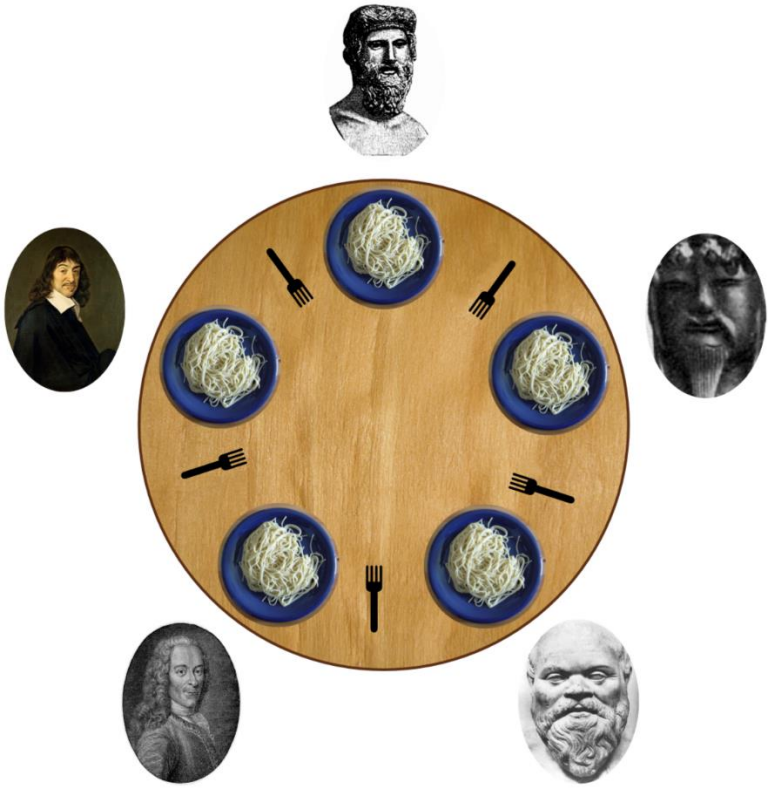
# Actividad 3: Problema de la cena de los Filósofos

Matías Navia Barrientos
Ingeniería Civil en Computación e Informática.
Asignatura: Sistemas Operativos
Profesora: Claudia Contreras G.
Fecha: 22 de mayo del 2024.

# Introducción:

El problema de los 5 filósofos o más conocido como el problema de la cena de los filósofos fue propuesto por el científico de la computación Edsger Dijkstra en 1965, que representa el problema de la sincronización de procesos en un sistema operativo.

El problema es simple pero desafiante, consiste en una mesa redonda, con 5 filósofos sentados alrededor de ella y cada uno tiene si propio plato de fideos y un tenedor a la izquierda de su plato.



En el momento de que un filósofo quiera comer, es necesarios utilizar dos tenedores, por lo que sólo puede tomar los que están a su izquierda y derecha suya, dándonos así una serie de situaciones donde puede que algunos filósofos coman o ninguno come y se mueren de hambre.

# Explicación solución:

Para este problema se utilizo la técnica del semáforo y hilos en un programa en lenguaje de Python:

```python
import threading
import time
import random

num_filosofos = 5
tenedores = [False] * num_filosofos
condicion = threading.Condition()
def coger_tenedores(i):
    with condicion:
        while tenedores[i] or tenedores[i - 1]:
            condicion.wait()
        tenedores[i] = tenedores[i - 1] = True

def dejar_tenedores(i):
    with condicion:
        tenedores[i] = tenedores[i - 1] = False
        condicion.notify_all()

def filosofo(i):
    while True:
        print(f"Filosofo {i + 1} está pensando")
        time.sleep(random.uniform(0, 4))
        coger_tenedores(i)
        print(f"Filosofo {i + 1} está comiendo")
        time.sleep(random.uniform(0, 4))
        print(f"Filosofo {i + 1} deja de comer, tenedores libres {i + 1}, {i if i > 0 else num_filosofos}")
        dejar_tenedores(i)

filosofos = []
for i in range(num_filosofos):
    hilo = threading.Thread(target=filosofo, args=(i,))
    filosofos.append(hilo)
    hilo.start()
```

y por consola se desplegaba las siguientes acciones:

```
Filosofo 1 está pensando
Filosofo 2 está pensando
Filosofo 3 está pensando
Filosofo 4 está pensando
Filosofo 5 está pensando
Filosofo 2 está comiendo
Filosofo 5 está comiendo
Filosofo 2 deja de comer, tenedores libres 2, 1
Filosofo 2 está pensando
Filosofo 3 está comiendo
Filosofo 5 deja de comer, tenedores libres 5, 4
Filosofo 5 está pensando
Filosofo 1 está comiendo
Filosofo 3 deja de comer, tenedores libres 3, 2
Filosofo 3 está pensando
Filosofo 4 está comiendo
Filosofo 1 deja de comer, tenedores libres 1, 5
Filosofo 1 está pensando
Filosofo 2 está comiendo
Filosofo 4 deja de comer, tenedores libres 4, 3
Filosofo 4 está pensando
Filosofo 5 está comiendo
Filosofo 2 deja de comer, tenedores libres 2, 1
Filosofo 2 está pensando
Filosofo 3 está comiendo
Filosofo 5 deja de comer, tenedores libres 5, 4
Filosofo 5 está pensando
Filosofo 1 está comiendo
Filosofo 1 deja de comer, tenedores libres 1, 5
Filosofo 1 está pensando
Filosofo 5 está comiendo
Filosofo 3 deja de comer, tenedores libres 3, 2
Filosofo 3 está pensando
Filosofo 2 está comiendo
Filosofo 5 deja de comer, tenedores libres 5, 4
Filosofo 5 está pensando
```

Como se ve en la consola, lo que hace este código es que todos tengan la oportunidad de comer y no se mueran de hambre. Además de eso solo 2 filósofos pueden comer a la vez tomando en cuenta estos requerimientos, que estén a menos un filósofo de distancia para no ocupar el tenedor que ocupa el que está comiendo

porque al final hay 5 tenedores y siempre se va a ocupar 4 tenedores, el que queda con 1 deberá esperar hasta que termine de comer el filosofo que esta ya sea el lado derecho o izquierdo, siempre respetando el lado que hace falta un tenedor. Y así mismo con los otros 2 que esperan a diferencia que ellos no tiene tenedor por lo que se demoraran el doble de tiempo hasta que esté disponible uno.

A continuación, se explicará cada parte del Código para su mejor entendimiento:

En esta primera parte del Código se declaran diferentes módulos para su utilización, y luego se definieron variables como lo es el número de filósofos, tenedores y condición que se utiliza para sincronizar partes de un programa que necesitan esperar a que se cumpla una cierta condición.

```
1  import threading
2  import time
3  import random
4
5  num_filosofos = 5
6  tenedores = [False] * num_filosofos
7  condicion = threading.Condition()
```

El método coger tenedores es utilizado por un filósofo para intentar coger dos tenedores. Si los tenedores a su izquierda y derecha están siendo utilizados, el filósofo espera. Cuando ambos tenedores están disponibles, los coge y los marca como utilizados. Todo esto se realiza dentro de un bloque “with” (línea 9) para sincronizar el acceso a los tenedores compartidos.

```
8  def coger_tenedores(i):
9      with condicion:
10         while tenedores[i] or tenedores[i - 1]:
11             condicion.wait()
12         tenedores[i] = tenedores[i - 1] = True
```

El método dejar tenedores es utilizado por un filósofo para liberar los tenedores que ha usado. Marca ambos tenedores como no utilizados y notifica a todos los demás filósofos que los tenedores están disponibles ahora.

```
14  def dejar_tenedores(i):
15      with condicion:
16          tenedores[i] = tenedores[i - 1] = False
17          condicion.notify_all()
```

El método filosofo representa el ciclo de vida de un filósofo. El filósofo piensa, intenta coger dos tenedores, come si los tenedores están disponibles, y luego libera los tenedores para que los otros filósofos que esperaban puedan comer. Este ciclo se repite continuamente. Además se definió los tiempos que se utiliza para pensar y para comer.

```
19  def filosofo(i):
20      while True:
21          print(f"Filosofo {i + 1} está pensando")
22          time.sleep(random.uniform(0, 4))
23          coger_tenedores(i)
24          print(f"Filosofo {i + 1} está comiendo")
25          time.sleep(random.uniform(0, 4))
26          print(f"Filosofo {i + 1} deja de comer, tenedores libres {i + 1},
    {i if i > 0 else num_filosofos}")
27          dejar_tenedores(i)
```

Por último, se crea una lista de filósofos, donde cada filósofo es un hilo. Para cada filósofo, se crea un nuevo hilo que ejecuta la función filosofo, donde i es el índice del filósofo. Este hilo se añade a la lista de filósofos y luego se inicia, lo que significa que el filósofo comienza su ciclo de vida.

```
filosofos = []
for i in range(num_filosofos):
    hilo = threading.Thread(target=filosofo, args=(i,))
    filosofos.append(hilo)
    hilo.start()
```

## Conclusión:

Hay diferentes soluciones para este tipo de problema, pero cada solución garantiza que todos los filósofos tengan la oportunidad de comer sin riesgo de interbloqueo, esto relacionándolo con los sistemas operativos es algo tedioso, ya que puede provocar una parálisis en el sistema y afectar negativamente su capacidad para ejecutar tareas de manera eficiente, y más aún cuando se utilizan hilos, ya que son más propensos a enfrentar problemas de interbloqueos.

En resumen, la resolución del problema de los filósofos comensales destaca la importancia de la sincronización en sistemas operativos, la gestión de recursos compartidos y la necesidad de soluciones robustas para evitar interbloqueos en entornos de multiprocesamiento y multihilo.