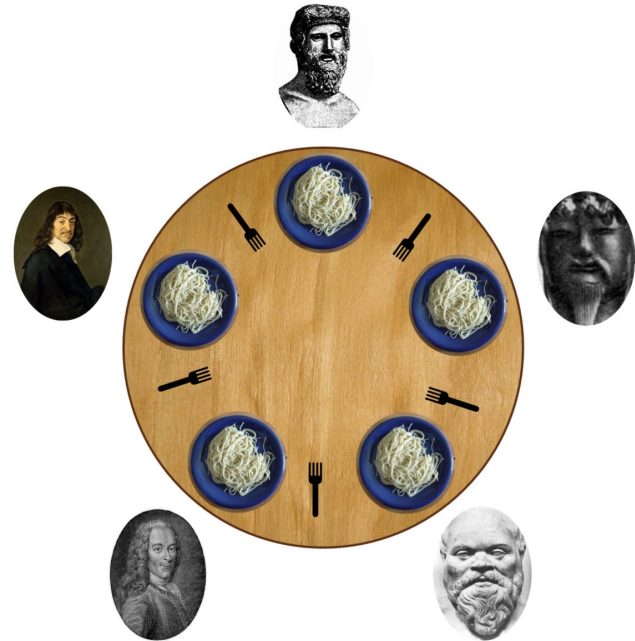


Problema de los *filósofos* *hambrientos*



El problema

- 5 Filósofos alrededor de una mesa
- Tenedores a la izquierda y a la derecha.
- Cuando no piensan, comen con ambos tenedores.
- Si no los pueden usar, esperan.

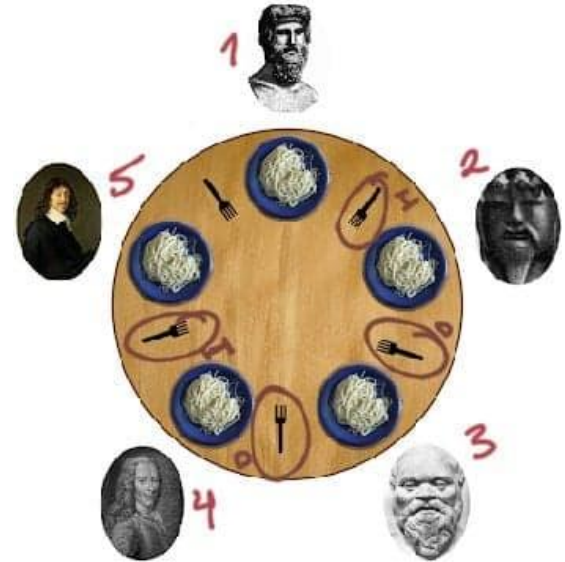


Propuesta de solución

Identificar filósofos no contiguos, como ejemplo numerando a partir de alguno arbitrario y separa en pares e impares.

Reglas para la toma del tenedor:

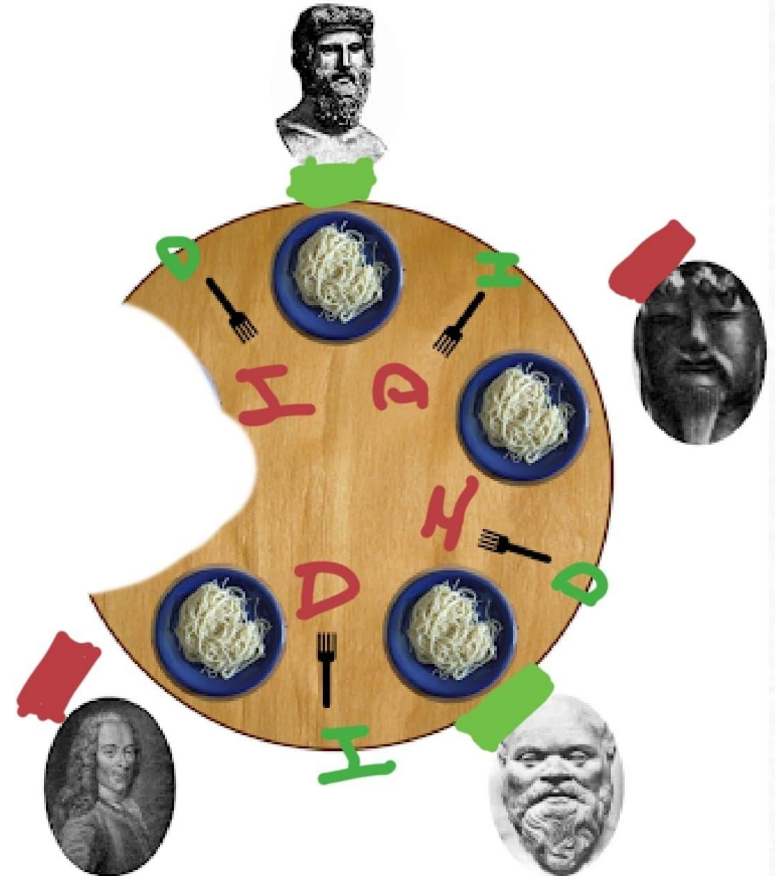
1. Todos los filósofos **pares** tomarán su tenedor derecho primero, y luego el izquierdo.
2. Todos los filósofos **impares** tomarán su tenedor izquierdo primero, y luego el derecho.



Un caso sencillo

Filósofos pares ($2n$)

Ahora existen n filósofos en cada subgrupo, y $2n$ cubiertos en la mesa, podemos hacer que coman en dos turnos (uno en cada grupo).



Ejemplificación casos

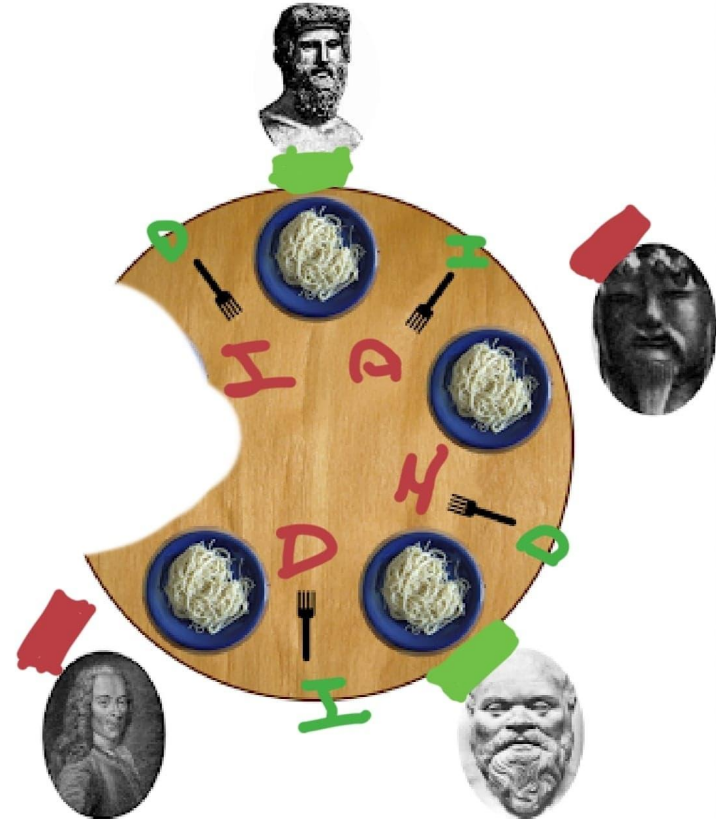
Los cuatro filósofos quieren comer a la vez:

El tercer filósofo agarra su tenedor izquierdo.

El segundo filósofo agarra su tenedor derecho.

El cuarto filósofo intenta agarrar su tenedor derecho, pero el tercero ya lo tiene, espera.

El primer filósofo intenta agarrar su tenedor izquierdo pero el segundo ya lo tiene, así que espera.



Tienen:



1° Filósofo: Sin tenedor. Quiere comer

2° Filósofo: Tenedor derecho.

3° Filósofo: Tenedor izquierdo.

4° Filósofo: Sin tenedor. Quiere comer

.El tercer filósofo agarra su tenedor derecho.

El segundo filósofo intenta agarrar su tenedor izquierdo, pero lo tiene el tercero, espera.

El tercer filósofo termina de comer y deja sus tenedores.

Tienen:



1° Filósofo: Sin tenedor. Quiere comer

2° Filósofo: Tenedor derecho.

3° Filósofo: Ya comió.

4° Filósofo: Sin tenedor. Quiere comer

El cuarto filósofo agarra el tenedor derecho.

El cuarto filósofo agarra su tenedor izquierdo.

El segundo filósofo ya puede agarrar el otro tenedor, come y los deja.

Tienen:



1º Filósofo: Sin tenedor. Quiere comer

2º Filósofo: Ya comió.

3º Filósofo: Ya comió.

4º Filósofo: . Ambos tenedores

El primer filósofo agarra su tenedor izquierdo.

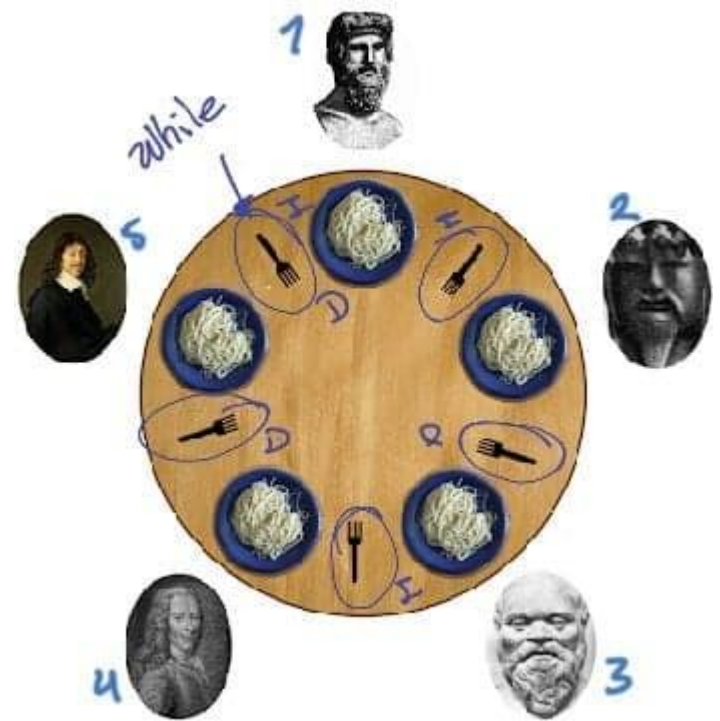
El cuarto filósofo come y deja los tenedores.

El primer filósofo agarra su tenedor derecho y puede comer.

Ya todos comieron :)

El peor de los casos

En caso de que sean un número impar, el primero o el último no comerán por lo que usaremos un condicional que haga que el primero espere un segundo turno, es decir, cuando el último termine.



Mutex en Python

- Es un candado compartido entre los procesos.
- Solo un estado (bloqueado / desbloqueado) está asociado con él.



Exclusión Mutua (Mutual Exclusion or Mutex)

“Mientras haya un proceso en la sección crítica, no entra nadie más.

Como el candado solo lo puede tomar un proceso a la vez. Esto soluciona la propiedad de exclusión mutua

Mutex en Python

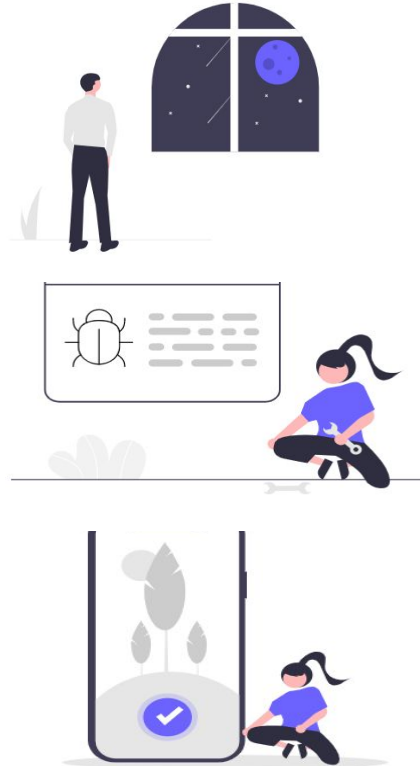
```
wait (mutex);
```

...

Sección Crítica


...

```
free (mutex);
```



Exclusión Mutua (Mutual Exclusion or Mutex)

Ejemplo Mutex



```
#----- Ejemplo Mutex-----  
import time  
from threading import Thread  
from threading import Lock  
  
def myfunc(i, mutex):  
    try:  
        mutex.acquire(1)  
        time.sleep(1)  
        print("Thread:", i)  
    except Exception as e:  
        print(e)  
    finally:  
        mutex.release()  
  
mutex = Lock()  
for i in range(0,10):  
    t = Thread(target=myfunc, args=(i,mutex))  
    t.start()  
    print("main loop", i)
```

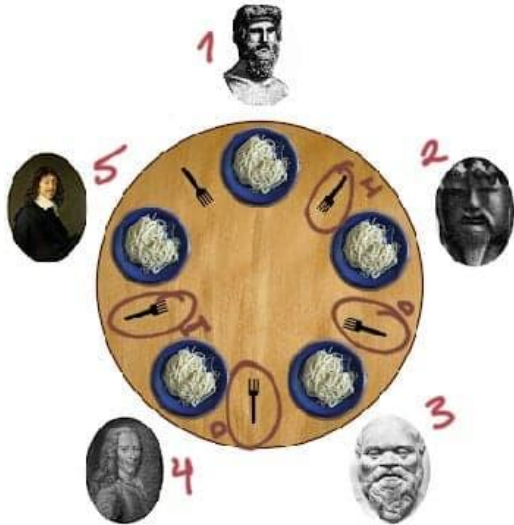
Out:

```
main loop 0  
main loop 1  
main loop 2  
main loop 3  
main loop 4  
main loop 5  
main loop 6  
main loop 7  
main loop 8  
main loop 9  
Thread: 0  
Thread: 1  
Thread: 2  
Thread: 3  
Thread: 4  
Thread: 5  
Thread: 6  
Thread: 7  
Thread: 8  
Thread: 9
```

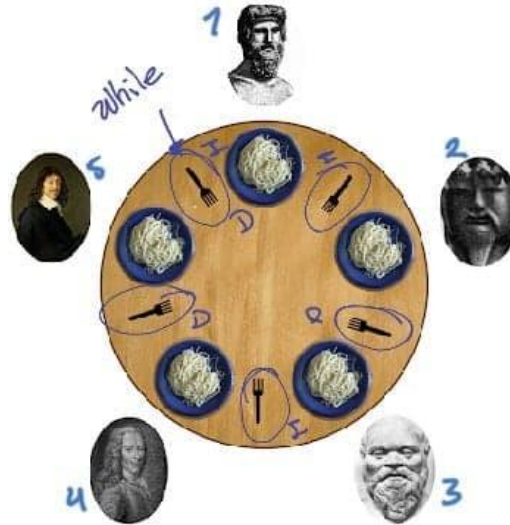
Aplicación (Solución mutua)

Los whiles esperan hasta la disponibilidad del recurso, ya que solo una persona puede tomar el tenedor al mismo tiempo. Tenemos que usar un candado para poder solucionar que solo un proceso se apropie del recurso.

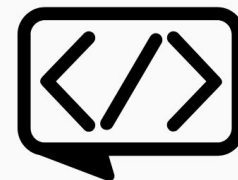
PARES



IMPARES



Pseudocódigo



```
def filosofo comer(i):  
    pname = currentProcessName()  
    if (pname%2):  
        try:  
            mesa[i].agarrarTenedor(izquierda)  
            while (!mesa[i].tenedorLibre(derecha)):  
                wait()  
            mesa[i].agarrarTenedor(derecha)  
            mesa[i].comer()  
            mesa[i].dejarTenedor(derecha)  
            mesa[i].dejarTenedor(izquierda)  
        else:  
            try:  
                mesa[i].agarrarTenedor(derecha)  
            while (!mesa[i].tenedorLibre(izquierda)):  
                wait()  
            mesa[i].agarrarTenedor(izquierda)  
            mesa[i].comer()  
            mesa[i].dejarTenedor(derecha)  
            mesa[i].dejarTenedor(izquierda)
```

fin

—