
**Benemérita Universidad Autónoma de Puebla**

**Profesor: Adolfo Aguilar Rico**

**Integrantes:**

**Alvarez Vargas David**

**Cruz Pinedo Carlos Daniel**

**Díaz Cruz Diana Monserrat**

**Varela Coyotl Lizet Cristal**

# *Reporte Filósofos Comensales*

Tarea:

- Realizar un programa en python de los filósofos comensales, utilizando rlock de la librería threading.
- Evitar que se dé la condición de abrazo mortal (deadlock), para ello hacer que cada filósofo tome un tenedor si es posible.
- Si no está disponible el otro tenedor lo suelte y esta **hambriento**
- Si lo intenta un número de veces (por ejemplo 10), entonces muere por **inanición**
- Si tiene los dos tenedores puede **comer** en un tiempo finito.
- Después de comer pasa a **filosofar**.

## Módulos utilizados

```
import time
import random
import threading
```

**Time:** El módulo **time** de la librería estándar de Python nos proporciona herramientas para trabajar con fechas y/o horas, en este caso la necesitamos dado a que en el código se hace uso de la función **sleep()** incluida en ese modulo. La función primordial de sleep() en el código es la de determinar el tiempo en que el filósofo va a pensar.

**Random**: El módulo **random** de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números aleatorios. En el código es utilizada en conjunto con **sleep()** para darle un tiempo aleatorio al filósofo para filosofar.

**Threading:** Este módulo construye interfaces multihilo de alto nivel sobre el módulo de bajo nivel thread. En el programa es utilizado para el manejo de hilos.

## Variables globales

```
N = 5
TIEMPO_TOTAL = 3
```

La variable *N* es utilizada en la clase *main()* para definir el número de filósofos que entraran, en cambio la variable *TIEMPO_TOTAL* es el tiempo designado para que cada filosofo ejecute sus actividades.

## Clase principal “main”

Esta clase al ser la clase principal nos permite agregar un nuevo filósofo y ejecutar todas las funciones de este a través de *f.start()* que en el programa equivale a run, tales funciones son las de tomar, pensar, comer y soltar.

Por último se utiliza la función *f.join()* , la cual bloquea hasta que termina el hilo.

```
def main():
    lista=[]
    for i in range(N):
        lista.append(filosofo()) #agregamos un filosofo a la lista

    for f in lista:
        f.start() #equivale a run() nos sirve para que el programa empiec
e

    for f in lista:
        f.join() #bloquea hasta terminado el hilo
```

## Clase filosofo

```
semaforo = threading.RLock() #semaforo binario, el cual nos asegura la ex
clusion mutua
    estado = [] #guarda el estado de cada filosofo
    tenedores = [] #arreglo de semaforos
```

En primer lugar tenemos la utilización de la función *threading.RLock()* la cual está incluida en el módulo *threading*, La clase RLock es una versión de bloqueo simple que solo se bloquea si el bloqueo está retenido por otro hilo.

En segundo lugar, tenemos un arreglo de nombre estado, su función el código es la de almacenar el estado de cada filosofo.

En la siguiente línea se tiene otro arreglo de nombre tenedores, este es un arreglo de semáforos, el cual muestra quien está en cola del tenedor.

```
   #constructor
    def __init__(self):
        super().__init__()    #nos sirve para heredar
        self.id=filosofo.count #se le asigna el id al filosofo
        filosofo.count+=1 #incrementa el contador de filosofos
        filosofo.estado.append('PENSANDO') #se ejecuta el filosofo con es
tado "PENSANDO"
        filosofo.tenedores.append(threading.Semaphore(0)) #agrega el sema
foro de su tenedor (tenedor a la izquierda)
```

```
        print("FILOSOFO {0} -
 PENSANDO".format(self.id)) #imprimimos el numero de filosofo con el
estado "PENSANDO"
```

En el constructor lo que se hace es inicializar las variables que se utilizaran mas adelante en el programa, tales como el estado y el id de los filósofos.

El método destructor se llama cuando se han eliminado todas las referencias al objeto, es decir, cuando se recolecta basura un objeto, que en este caso es cuando el filósofo se levanta de la mesa.

En esta parte también se incluye una función llamada derecha lo que hace es buscar a la derecha. En la función denominada izquierda se busca a la izquierda.

Y en la función verificar se verifica si esta hambriento o comiendo en dado caso que este comiendo se asigna el estado “comiendo”.

```
#destructor
    def __del__(self):
        print("FILOSOFO {0} -
 Se para de la mesa".format(self.id))  #se necesita para saber cuando ter
mina el proceso o hilo

    def derecha(self,i):
        return (i-1)%N #buscamos a la derecha

    def izquierda(self,i):
        return(i+1)%N #buscamos a la izquierda

    def verificar(self,i):
        if filosofo.estado[i] == 'HAMBRIENTO' and
filosofo.estado[self.izquierda(i)] != 'COMIENDO' and
filosofo.estado[self.derecha(i)] != 'COMIENDO':
            filosofo.estado[i]='COMIENDO' #le asigna al filosofo el
estado "COMIENDO"
            filosofo.tenedores[i].release()  #verifica si los vecinos no
estan comiendo, si es asi aumenta el semaforo y cambia su estado a
"COMIENDO"
```

En esta parte se va a correr lo que es la función pensar, tomar, comer y soltar. Lo de pensar quiere decir que el filósofo está filosofando. Tomar, tomar el tenedor correspondiente. Comer, el filósofo come y en soltar, el filósofo suelta el tenedor.

```
    #funciones que se ejecutaran:
    def run(self):
```

```
        for i in range(TIEMPO_TOTAL):
            self.pensar() #filosofo filosofando
            self.tomar() #tomar los tenedores correspondientes
            self.comer() #el filosofo come
            self.soltar() #suelta tenedores
```

En la siguiente parte del código definimos la función tomar en cual se hace la exclusión mutua después se la cambia el estado del filósofo a hambriento.

De ahí se corrobora a sus vecinos y se cambia el arreglo de estado.

Y en la última función en caso de poder tomarlo esta bloquea con el estado “comiendo”.

```
def tomar(self):
        filosofo.semaforo.acquire() #exclusion mutua
        filosofo.estado[self.id] = 'HAMBRIENTO' #cambia el estado del
filosofo a "HAMBRIENTO"
        self.verificar(self.id) #verifica a sus vecinos
        filosofo.semaforo.release() #cambia el arreglo de estado (ya
intento de tomar los tenedores)
        filosofo.tenedores[self.id].acquire() #en caso de poder tomarlos,
este se bloqueara con estado "COMIENDO"
```

Este método es llamado desde la clase principal para ejecutar todas las funciones del código como pensar, tomar, comer y soltar.

En esta línea se define la función de pensar, utilizamos la librería time y después mandamos a dormir lo que es el random de randint el cual se le asigna un tiempo aleatorio a cada filosofo para que simulemos la acción de pensar.

```
def pensar(self):
        time.sleep(random.randint(0,5)) #se le asigna un tiempo aleatorio
a cada filosofo para pensar
```

En la siguiente parte del código definimos la función soltar, la acción que va hacer cada filósofo, cada que un filósofo suelta el tenedor lo siguiente es que el estado cambia a pensando y por consecuente verifica los movimientos.

Y por último se termina con la manipulación de tenedores.

```
def soltar(self):
        filosofo.semaforo.acquire() #soltara los tenedores
        filosofo.estado[self.id] = 'PENSANDO' #cambia estado a "PENSANDO"
        self.verificar(self.izquierda(self.id)) #verifica a la izquierda
        self.verificar(self.derecha(self.id)) #verifica a la derecha
```

```
        filosofo.semaforo.release() #fin de minipulacion de tenedores
```

En esta línea de código definimos la función comer la cual simulara la misma de cada filósofo, se le asigna un tiempo para comer entre un filósofo y otro, el cual es el tiempo de espera.

Después de que se termina de hacer la función se manda a imprimir que cada filósofo terminó de comer.

```
def comer(self):
      print("FILOSOFO {} COMIENDO".format(self.id))
      time.sleep(2) #tiempo dado para comer, es el tiempo de espera
      print("FILOSOFO {} TERMINO DE COMER".format(self.id))
```