Utilizamos semaforo = threading.Lock() para el semaforo binario que asegura la exclusion mutua.

Despues conoceremos el estado de cada filosofo, es decir, si se encuentra comiendo, hambriento , pensando o agarrando los tenedores utilizando

la variable estado = []

tenedores = [] ------> ARRAY DE SEMAFOROS PARA SINCRONIZAR ENTRE FILOSOFOS, MUESTRA QUIEN ESTA EN COLA DEL TENEDOR

def pensar(self):

time.sleep(random.randint(0,5)) -------> CADA FILOSOFO SE TOMA DISTINTO TIEMPO PARA PENSAR, ALEATORIO

Definimos Verificar para saber si los vecinos de cada filoso no estan comiendo, en ese caso aumenta el semafo del tenedor y cambia su estado a comiendo.

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'

filosofo.tenedores[i].release() ---------> SUS VECINOS NO ESTAN COMIENDO AUMENTA EL SEMAFORO DEL TENEDOR Y CAMBIA SU ESTADO A COMIENDO

def tomar(self):

filosofo.semaforo.acquire() -------> SEÑALA QUE TOMARA LOS TENEDORES (EXCLUSION MUTUA)

filosofo.estado[self.id] = 'HAMBRIENTO'

self.verificar(self.id) -------> VERIFICA SUS VECINOS, SI NO PUEDE COMER NO SE BLOQUEARA EN EL SIGUIENTE ACQUIRE

filosofo.semaforo.release() -------> SEÑALA QUE YA DEJO DE INTENTAR TOMAR LOS TENEDORES (CAMBIAR EL ARRAY ESTADO)

filosofo.tenedores[self.id].acquire() --------> SOLO SI PODIA TOMARLOS SE BLOQUEARA CON ESTADO COMIENDO

Utilizamos la siguiente funcion para bloquear hasta que termina el thread

for f in lista:

f.join()

def comer(self):

print("FILOSOFO {} COMIENDO".format(self.id))

time.sleep(2) ---> Lo utilizamos para contar el tiempo arbitrario que tiene un filosofo para comer

print("FILOSOFO {} TERMINO DE COMER".format(self.id))