**B2**

**1. Dining philosophers (Deadlocks)**

In computer science, the dining philosophers problem is an example problem often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them. It was originally formulated in 1965 by Edsger Dijkstra as a student exam exercise, in terms of computers competing for access to tape drive peripherals. Soon after, Tony Hoare gave the problem its present formulation.

**Problem Statement**

Five silent philosophers sit at a table around a bowl of spaghetti. A fork is placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. Eating is not limited by the amount of spaghetti left: assume an infinite supply. However, a philosopher can only eat while holding both the fork to the left and the fork to the right.

Each philosopher can pick up an adjacent fork, when available, and put it down, when holding it. These are separate actions: forks must be picked up and put down one by one.

The problem is how to design a discipline of behavior (a concurrent algorithm) such that each philosopher won't starve, i.e. can forever continue to alternate between eating and thinking.

Your task is to modify the [philosophers.py](https://python.g-node.org/python-summerschool-2011/_media/philosophers.py) which can be affected by a possible deadlock in a way that avoids it. Please make sure your philosophers obey the above rules in your solution!

Original:

import threading

import time

class Philosopher(threading.Thread):

def \_\_init\_\_(self, name, forkl, forkr, stopevent):

threading.Thread.\_\_init\_\_(self)

self.name = name

self.forkl = forkl

self.forkr = forkr

self.stopevent = stopevent

def run(self):

while not self.stopevent.is\_set():

self.think()

self.dine()

def think(self):

print self.name, 'is thinking.'

def dine(self):

print self.name, 'is hungry.'

self.forkl.acquire()

self.forkr.acquire()

print self.name, 'is dining.'

print self.name, 'finished dining.'

self.forkl.release()

self.forkr.release()

def main():

forks = [threading.Lock() for i in range(5)]

stopevent = threading.Event()

philosophers = [Philosopher('Philosopher %i' % i, forks[i%5], forks[(i+1)%5], stopevent) for i in range(5)]

for p in philosophers:

p.start()

time.sleep(10)

print '[Setting Stop-Event.]'

stopevent.set()

print '[Waiting for Philosophers to stop...',

for p in philosophers:

p.join()

print 'Done.'

if \_\_name\_\_ == '\_\_main\_\_':

main()

Para el informe:

**Planteamiento del problema:**

El problema de la cena de los filósofos representa uno de los problemas icónicos sobre sincronización de procesos. Este problema es acerca de cinco filósofos que se sientan en una misma mesa a cenar, donde hay cinco platos y cinco cubiertos, además de una ración indeterminada de espagueti.

Un filósofo puede realizar solamente una acción a la vez, la de pensar o la de comer.

Para comer, cada filósofo necesita un plato y dos cubiertos. Esto hace que se compita por dichos recursos por parte de los filósofos adyacentes, pero sólo uno de ellos lo logrará a la vez.

Es decir, si todos los filósofos tomaran un cubierto de un lado al mismo tiempo, los del otro lado, es decir todos, quedarían en estado de espera indefinidamente, ya que ninguno soltará el suyo y todos morirán de hambre.

Entre las soluciones posibles son que sólo un filósofo pueda comer a la vez, y cuando este termine, suelta el cubierto para que coma el siguiente filósofo, el problema es que los últimos filósofos pueden morir de hambre por la espera.

Otra posible solución es que se entreguen fichas a manera de turnos temporizados, una vez terminado se entregan dichos tokens a los siguientes filósofos, el problema con esto es la sincronización del tiempo de los tokens y esto generaría una cola de espera para obtener cubiertos.

**Solución propuesta y pseudocódigo:**

La solución adoptada se la realiza gracias al uso de semáforos, tomando en cuenta que un cubierto es el recurso por el cual se compite, además lo que puede realizar un filósofo en un determinado tiempo es: pensar, comer, tomar cubierto o soltar cubierto.

Cada filósofo tendrá un identificador único, que en este caso será un número del 0 al 4, dividiéndolos en un grupo de acuerdo con su paridad (id par o impar).

Al principio la acción que realizan todos los filósofos es pensar,

Para comenzar a tomar un cubierto, se revisa si el id del filósofo es par, y de ser así, tomará el cubierto de su derecha y espera la disponibilidad del cubierto de su izquierda, una vez libre, lo toma.

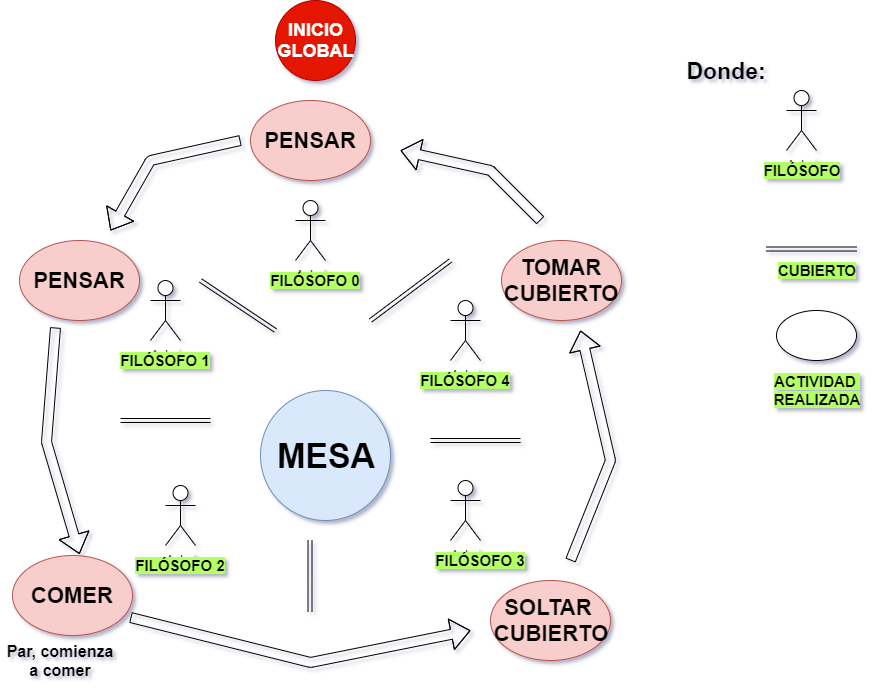
Turno de impar: Posteriormente se revisa si el id del filósofo es impar, y de ser así, tomará el cubierto de su izquierda y espera la disponibilidad del cubierto de su derecha, una vez libre, lo toma.

De esta manera se asegura que hasta dos filósofos estén comiendo a la vez.

Una vez hayan terminado de comer, los filósofos sueltan sus cubiertos uno a uno, regresando a su acción de pensar y volviendo a los cubiertos disponibles para ser usados en los siguientes turnos.

Estos pasos se repiten para los siguientes filósofos.

**Esquema gráfico del algoritmo con la solución:**

El siguiente esquema se lo realizó con una herramienta online, en la página web ([www.draw.io](http://www.draw.io)).

**Pseudocódigo del algoritmo con la solución:**

Con base en la solución planteada en este punto, se presenta el siguiente pseudocódigo:

Inicio: Todos los filósofos piensan

-Los filósofos quieren comer

-Mientras un filósofo quiera comer

Si el filósofo tiene id impar

-Filósofo busca cubierto a su izquierda

-Si cubierto a la izquierda está disponible -> Tomar cubierto

-Caso contrario-> Esperar disponibilidad de cubierto

-Filósofo busca cubierto a su derecha

-Si cubierto a la derecha está disponible -> Tomar cubierto

-Caso contrario-> Esperar disponibilidad de cubierto

Caso contrario (filósofo tiene id par)

-Filósofo busca cubierto a su derecha

-Si cubierto a la derecha está disponible -> Tomar cubierto

-Caso contrario-> Esperar disponibilidad de cubierto

-Filósofo busca cubierto a su izquierda

-Si cubierto a la izquierda está disponible -> Tomar cubierto

-Caso contrario-> Esperar disponibilidad de cubierto

-El filósofo terminó de comer

-El filósofo ya no tiene hambre

-El filósofo suelta los cubiertos

-El filósofo cambia de actividad a pensar

Fin