Reporte: “El problema de los filósofos comensales”.

Integrantes del equipo:

Oswaldo Cordoba Jimenez 201627085.
Cesar Ruiz Rodrigues Flores 201624846.
Oscar Arzeta Rodriguez 201629369.

Introduccion.

El **Porblema de los filósofos** es un problema tipico en las ciencias de la computacion, ya que este representa el problema de la sincronización de procesos en un sistema operativo. Cinco filósofos se sientan alrededor de una mesa y pasan su vida cenando y pensando, cada filośofo tiene un plato de spaguetti y un tenedor a la izquierda y uno a la derecha. Para poder comer, tiene que tomar ambos tenedores y cada filósofo sólo puede tomar los que estan a su izquierda y derecha. Si cualquier filósofo toma un tenedor y el otro está ocupado, se quedará esperando, con el tenedor en la mano, hasta que pueda tomar el otro tenedor, para luego empezar a comer.

Si dos filósofos adyacentes intentan tomar el mismo tenedor a una vez, se produce una condición de carrera: ambos compiten por tomar el mismo tenedor, y uno de ellos se queda sin comer.

Si todos los filósofos toman el tenedor que está a su derecha al mismo tiempo, entonces todos se quedarán esperando eternamente, porque alguien debe liberar el tenedor que les falta. Nadie lo hará porque todos se encuentran en la misma situación (esperando que alguno deje sus tenedores). Entonces los filósofos se morirán de hambre. Este bloqueo mutuo se denomina interbloqueo o deadlock.

El problema consiste en encontrar un algoritmo que permita que los filósofos nunca se mueran de hambre.

Codigo (Python).

Para poder modelar este problema y su solución usaremos hilos(Threads) para simular a los filósofos y los tenedores seran los recursos compartidos, seran la zona critica en el programa, ya que 2 filósofos no pueden usar el mismo tenedor al mismo tiempo, y a su vez deben liberarlos para que otro pueda usarlo.
Para emprezar, importaremos las siguientes bibliotecas:

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

Des pues creamos la clase “Filosofos” y en su constructor declaramos su nombre, estado, tenedor derecho y tenedor izquierdo.

```
class Filosofos(threading.Thread):


    def __init__(self, name, state,  leftfork, rightfork):
        threading.Thread.__init__(self)
        self.leftfork = leftfork
        self.rightfork = rightfork
        self.name = name
        self.state = state
```

En su metodo “run()” declaramos un ciclo “while” infinito y dos bloques “Try/Except” anidados en el primer bloque, el hilo(filosofo) intentara tomar primero su tenedor derecho, si lo logra, procedera a tomar su tenedor izquierdo, de no lograr tomar su primer tenedor, pasa a su estado “Hambriento”, suma 1 a su numero de intentos para comer y espera un tiempo aleatorio de 1 a 3 segundos hasta volver a intentar tomar de nuevo su

tenedor derecho.

```
try:
    self.rightfork.acquire()

except:
    self.state = "hambriento"
    print("El filosofo {} se encuentra {}".format(self.name, self.state))
    intentosC += 1
    time.sleep(random.randint(1,3)
```

En el segundo bloque de "Try/Except" el hilo(filósofo) intentara tomar su segundo tenedor(izquierdo), de lograrlo, reinicia sus intentos para comer, cambia a su estado "Comiendo" y se queda en ese estado de 1 a 5 segundos aleatorios mientras "come", uan vez termina de comer, pasa a su estado "Pensando", libera sus recursos compartidos(tenedores), y se queda "Pensando" un tiempo aleatorio de 1 a 10 segundos. Si no logra tomar su segundo tenedor, suelta su primer tenedor, incrementa su numero de intentos para comer y pasa a su estado de "Hambriento" durante un tiempo aleatorio de 1 a 3 segundos hasta vover a intentar tomar sus dos tenedores.

```
try:
    self.leftfork.acquire()
    self.state = "comiendo"
    print("El filosofo {} se encuentra {}".format(self.name, self.state))
    intentosC = 0
    time.sleep(random.randint(1,5))
    self.state = "pensando"
    print("El filosofo {} se encuentra {}".format(self.name, self.state))
    self.rightfork.release()
    self.leftfork.release()
    time.sleep(random.randint(1,10))
except:
    self.rightfork.release()
    self.state = "hambriento"
    print("El filosofo {} se encuentra {}".format(self.name, self.state))
    intentosC += 1
    time.sleep(random.randint(1,3))
```

Finalmete si el hilo(filósofo) no consigue "comer en almenos 10 intentos, y este permanece en su estado "Hambriento", pasa a su estado "Muerto de Hambre" donde termina el cilo "while" y ya no puede hacer nada.

```
if intentosC > 10:
    self.state = "Muerto de Hambre"
    print("El filosofo {} ha {}".format(self.name, self.state))
    break
```

Tecnicamente hablando, los bloques "Try/Except" antes mencionados, son suficientes para que todos los hilos puedan hacer uso de los recursos compartidos y estos nunca "Mueran de hambre"
De forma completa, la funcion "run()" se ve asi:

```
def run(self):
    while True:
        try:
            self.rightfork.acquire()
            try:
                self.leftfork.acquire()
                self.state = "comiendo"
                print("El filosofo {} se encuentra {}".format(self.name, self.state))
                intentosC = 0
                time.sleep(random.randint(1,5))
                self.state = "pensando"
                print("El filosofo {} se encuentra {}".format(self.name, self.state))
                self.rightfork.release()
                self.leftfork.release()
                time.sleep(random.randint(1,10))
            except:
                self.rightfork.release()
                self.state = "hambriento"
                print("El filosofo {} se encuentra {}".format(self.name, self.state))
                intentosC += 1
                time.sleep(random.randint(1,3))

        except:
            self.state = "hambriento"
            print("El filosofo {} se encuentra {}".format(self.name, self.state))
            intentosC += 1
            time.sleep(random.randint(1,3)
            if intentosC > 10:
                self.state = "Muerto de Hambre"
                print("El filosofo {} ha {}".format(self.name, self.state))
                break
```

En el programa pincipal, se declaran las variables de los tenedores.

```
t1 = threading.RLock()
t2 = threading.RLock()
t3 = threading.RLock()
t4 = threading.RLock()
t5 = threading.RLock()
```

Se construyen los hilos(Filosofos) donde se les da su nombre, su estado inicial por defecto es "Pensando" y se le dice cual es su tenedor izquierdo y su tenedor derecho.

```
f1 = Filosofos("1", "pensando", t1, t2)
f2 = Filosofos("2", "pensando", t2, t3)
f3 = Filosofos("3", "pensando", t3, t4)
f4 = Filosofos("4", "pensando", t4, t5)
f5 = Filosofos("5", "pensando", t5, t1)
```

Finalmente se inician los hilos para empezar a trabajar.

```
f1.start()
f2.start()
f3.start()
f4.start()
f5.start()
```