Code

1

import threading

import random

import time

#inheriting threading class in Thread module

class Philosopher(threading.Thread):

running = True #used to check if everyone is finished eating

#Since the subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.\_\_init\_\_()) before doing anything else to the thread.

def \_\_init\_\_(self, index, forkOnLeft, forkOnRight):

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

self.index = index

self.forkOnLeft = forkOnLeft

self.forkOnRight = forkOnRight

def run(self):

while(self.running):

# Philosopher is thinking (but really is sleeping).

time.sleep(30)

print ('Philosopher %s is hungry.' % self.index)

self.dine()

def dine(self):

# if both the semaphores(forks) are free, then philosopher will eat

fork1, fork2 = self.forkOnLeft, self.forkOnRight

while self.running:

fork1.acquire() # wait operation on left fork

locked = fork2.acquire(False)

if locked: break #if right fork is not available leave left fork

fork1.release()

print ('Philosopher %s swaps forks.' % self.index)

fork1, fork2 = fork2, fork1

else:

return

self.dining()

#release both the fork after dining

fork2.release()

fork1.release()

def dining(self):

print ('Philosopher %s starts eating. '% self.index)

time.sleep(30)

print ('Philosopher %s finishes eating and leaves to think.' % self.index)

def main():

forks = [threading.Semaphore() for n in range(5)] #initialising array of semaphore i.e forks

#here (i+1)%5 is used to get right and left forks circularly between 1-5

philosophers= [Philosopher(i, forks[i%5], forks[(i+1)%5])

for i in range(5)]

Philosopher.running = True

for p in philosophers: p.start()

time.sleep(100)

Philosopher.running = False

print ("Now we're finishing.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

2

import threading

import random

import time

# Dining philosophers, 5 Phillies with 5 forks. Must have two forks to eat.

#

# Deadlock is avoided by never waiting for a fork while holding a fork (locked)

# Procedure is to do block while waiting to get first fork, and a nonblocking

# acquire of second fork. If failed to get second fork, release first fork,

# swap which fork is first and which is second and retry until getting both.

#

# See discussion page note about 'live lock'.

class Philosopher(threading.Thread):

running = True

def \_\_init\_\_(self, xname, forkOnLeft, forkOnRight):

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

self.name = xname

self.forkOnLeft = forkOnLeft

self.forkOnRight = forkOnRight

def run(self):

while(self.running):

# Philosopher is thinking (but really is sleeping).

time.sleep( random.uniform(3,13))

print('%s is hungry.' % self.name)

self.dine()

def dine(self):

fork1, fork2 = self.forkOnLeft, self.forkOnRight

while self.running:

fork1.acquire(True)

locked = fork2.acquire(False)

if locked: break

fork1.release()

print('%s swaps forks' % self.name)

fork1, fork2 = fork2, fork1

else:

return

self.dining()

fork2.release()

fork1.release()

def dining(self):

print('%s starts eating '% self.name)

time.sleep(random.uniform(1,10))

print('%s finishes eating and leaves to think.' % self.name)

def DiningPhilosophers():

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

philosopherNames = ('Aristotle','Kant','Spinoza','Marx', 'Russel')

philosophers= [Philosopher(philosopherNames[i], forks[i%5], forks[(i+1)%5]) \

for i in range(5)]

random.seed(507129)

Philosopher.running = True

for p in philosophers: p.start()

time.sleep(100)

Philosopher.running = False

print ("Now we're finishing.")

DiningPhilosophers()