

APP LAB-EX-6

Name: Rahul Goel
Reg no: RA1911030010094
Batch: CSE-O2

7)

```
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()
```

The screenshot shows a Replit environment with a Python script named `main.py` and its execution output in the console.

Code in `main.py`:

```
1 import threading
2 import random
3 import time
4
5 #inheriting threading class in Thread module
6 class Philosopher(threading.Thread):
7     running = True #used to check if everyone is
8     finished eating
9
10 #Since the subclass overrides the constructor, it must
11 #make sure to invoke the base class constructor
12 # (Thread.__init__()) before doing anything else to the
13 # thread.
14 def __init__(self, index, forkOnLeft, forkOnRight):
15     threading.Thread.__init__(self)
16     self.index = index
17     self.forkOnLeft = forkOnLeft
18     self.forkOnRight = forkOnRight
19
20 def run(self):
21     while(self.running):
22         # Philosopher is thinking (but really is
23         # sleeping).
24         time.sleep(30)
25         print ('Philosopher %s is hungry.' %
26               self.index)
27         self.dine()
28
29 def dine(self):
30     # if both the semaphores(forks) are free, then
31     # philosopher will eat
32     fork1, fork2 = self.forkOnLeft, self.forkOnRight
33     while self.running:
34         fork1.acquire() # wait operation on left fork
```

Console Output:

```
Philosopher 0 is hungry.
Philosopher 0 starts eating.
Philosopher 1 is hungry.
Philosopher 2 is hungry.
Philosopher 2 starts eating.
Philosopher 3 is hungry.
Philosopher 4 is hungry.
Philosopher 4 swaps forks.
Philosopher 0 finishes eating and leaves to think.
Philosopher 1 swaps forks.
Philosopher 4 starts eating.
Philosopher 2 finishes eating and leaves to think.
Philosopher 1 starts eating.
Philosopher 3 swaps forks.
Philosopher 0 is hungry.
Philosopher 4 finishes eating and leaves to think.
Philosopher 3 starts eating.
Philosopher 0 swaps forks.
Philosopher 1 finishes eating and leaves to think.
Philosopher 0 starts eating.
Philosopher 2 is hungry.
Philosopher 2 swaps forks.
Now we're finishing.
> Philosopher 4 is hungry.
Philosopher 3 finishes eating and leaves to think.
Philosopher 2 starts eating.
Philosopher 1 is hungry.
Philosopher 0 finishes eating and leaves to think.
Philosopher 2 finishes eating and leaves to think.
```