Dining philosophers

11.2.2017

1 Solución centralizada

```
import time
from random import random
from threading import Condition, Thread


class Table:
    inuse = set()
    table = Condition()

    def acquire(self, fork1, fork2):
        with self.table:
            while fork1 in self.inuse or fork2 in self.inuse:
                self.table.wait()
            self.inuse.add(fork1)
            self.inuse.add(fork2)

    def release(self, fork1, fork2):
        with self.table:
            self.inuse.remove(fork1)
            self.inuse.remove(fork2)
            self.table.notifyAll()


table = Table()
eating = set()


def philosopher(me, need1, need2):
    global table, eating
    while True:
        # print(me, "thinking")
        time.sleep(random())
        table.acquire(need1, need2)
        eating.add(me)
        # print(me, "eating")
        print("eating", eating)
        time.sleep(random())
        table.release(need1, need2)
        eating.remove(me)
```

```
"""
test suite
"""

philo1 = Thread(target=philosopher, args=(1, 1, 2))
philo2 = Thread(target=philosopher, args=(2, 2, 3))
philo3 = Thread(target=philosopher, args=(3, 3, 4))
philo4 = Thread(target=philosopher, args=(4, 4, 5))
philo5 = Thread(target=philosopher, args=(5, 5, 1))

philo1.start()
philo2.start()
philo3.start()
```

```
philo4.start()
philo5.start()
```

2 Solución descentralizada

```
import time
from random import random
from threading import Condition, Thread


class Chopstick:
    _condition = Condition()
    _busy = False

    def take(self):
        with self._condition:
            while (self._busy):
                self._condition.wait()
            self._busy = True

    def put(self):
        with self._condition:
            self._busy = False
            self._condition.notify_all()

eating = set()

def philosopher(me, need1, need2):
    while True:
        # print(me, "thinking")
        time.sleep(random())
        if id(need1) < id(need2):
            need1.take()
            need2.take()
        else:
            need2.take()
            need1.take()
        eating.add(me)
        # print(me, "eating")
        print("eating", eating)
        time.sleep(random())
        need1.put()
        need2.put()
        eating.remove(me)
```

```
"""
test suite
"""

F1 = Chopstick()
F2 = Chopstick()
F3 = Chopstick()
F4 = Chopstick()
F5 = Chopstick()

philo1 = Thread(target=philosopher, args=(1, F1, F2))
philo2 = Thread(target=philosopher, args=(2, F2, F3))
philo3 = Thread(target=philosopher, args=(3, F3, F4))
```

```
philo4 = Thread(target=philosopher, args=(4, F4, F5))
philo5 = Thread(target=philosopher, args=(5, F5, F1))

philo1.start()
philo2.start()
philo3.start()
philo4.start()
philo5.start()
```

3 Creación de un lobby

2 o más se ponen de acuerdo para que siempre haya 1 de ellos comiendo.

```
import time
from random import random
from threading import Condition, Thread
```

```
class Table:
    inuse = set()
    table = Condition()

    def acquire(self, fork1, fork2):
        with self.table:
            while fork1 in self.inuse or fork2 in self.inuse:
                self.table.wait()
            self.inuse.add(fork1)
            self.inuse.add(fork2)

    def release(self, fork1, fork2):
        with self.table:
            self.inuse.remove(fork1)
            self.inuse.remove(fork2)
            self.table.notifyAll()
```

```
class Lobby:
    inhouse = set()
    cond = Condition()

    def enter(self, id):
        with self.cond:
            self.inhouse.add(id)
            self.cond.notifyAll()

    def leave(self, id):
        with self.cond:
            while len(self.inhouse) < 2:
                self.cond.wait()
            self.inhouse.remove(id)
```

```
table = Table()
eating = set()


def philosopher(me, need1, need2, cartel=None):
    global table, eating
```

```
    while True:
        # print(me, "thinking")
        time.sleep(random())
        table.acquire(need1, need2)
        if cartel:
            cartel.enter(me)
        eating.add(me)
        # print(me, "eating")
        print("eating", eating)
        time.sleep(random())
        if cartel:
            cartel.leave(me)
        table.release(need1, need2)
        eating.remove(me)
```

```
"""
test suite
"""

lobby = Lobby()

philo1 = Thread(target=philosopher, args=(1, 1, 2, lobby))
philo2 = Thread(target=philosopher, args=(2, 2, 3))
philo3 = Thread(target=philosopher, args=(3, 3, 4, lobby))
philo4 = Thread(target=philosopher, args=(4, 4, 5))
philo5 = Thread(target=philosopher, args=(5, 5, 1))

philo1.start()
philo2.start()
philo3.start()
philo4.start()
philo5.start()
```

4 Fair: un poquito de orden

Lo que quieren comer se apuntan a una cola. Tiene prioridad, de entre los que pueden comer, el que lleva más tiempo (FIFO)

```
import time
from random import random
from threading import Condition, Thread
```

```
class Table:
   inuse = set()
   table = Condition()

   queue = list()

   def acquire(self, it, fork1, fork2):
      with self.table:
         request = (it, fork1, fork2)
         self.queue.append(request)
         while self.have_to_wait(it):
            self.table.wait()
         self.inuse.add(fork1)
         self.inuse.add(fork2)
         self.queue.remove(request)

   def release(self, fork1, fork2):
      with self.table:
         self.inuse.remove(fork1)
         self.inuse.remove(fork2)
         self.table.notifyAll()

   def have_to_wait(self, candidate):
      for it, fork1, fork2 in self.queue:
         if fork1 in self.inuse:
            continue
         if fork2 in self.inuse:
            continue
         return not it == candidate
      return True
```

```
table = Table()
eating = set()

def philosopher(me, need1, need2):
   global table, eating
   lunchs = 0
   while True:
      # print(me, "thinking")
      time.sleep(random())
      table.acquire(me, need1, need2)
      eating.add(me)
      # print(me, "eating")
      # print("eating", eating)
      lunchs+= 1
      print(me, lunchs)
      time.sleep(random())
      table.release(need1, need2)
      eating.remove(me)
```

```
"""
test suite
"""

philo1 = Thread(target=philosopher, args=(1, 1, 2))
philo2 = Thread(target=philosopher, args=(2, 2, 3))
philo3 = Thread(target=philosopher, args=(3, 3, 4))
philo4 = Thread(target=philosopher, args=(4, 4, 5))
philo5 = Thread(target=philosopher, args=(5, 5, 1))

philo1.start()
philo2.start()
philo3.start()
philo4.start()
philo5.start()
```