工科 112 E94084032 林耕澤 作業二 程式說明

```
1  import threading
2  import random
3  import time
```

首先先介紹 import 的模組，
第一個為 threading ，是 Python 標準函式庫裡面的模組，為撰寫多執行續平行化程式最基本的方式。
第二個為 random，也是 Python 標準函式庫裡面的模組，用來製造隨機數，不過這裡沒用到(後來改掉了)。
第三個為 time，也是 Python 標準函式庫裡面的模組，是一種有許多與時間有相關處理模組，其中我們這邊會使用到 sleep，用以製造時間段(短暫暫停執行)。

```
4
5  #繼承執行緒-哲學家的class
6  class Philosopher(threading.Thread):
```

再來，創建一個繼承 threading.Thread 的 class。

```
class Philosopher(threading.Thread):
    #確認大家是不是都吃了
    running = True

    def __init__(self, index, chopstick_Left, chopstick_Right):
        threading.Thread.__init__(self)
        self.index = index
        self.chopstick_Left = chopstick_Left
        self.chopstick_Right = chopstick_Right
```

先給定 running 一個初始值 True，再來要設定初始化 method _init_，其中包括三個參數，分別為 index(用來給定號碼，ex:哲學家 1 號等等)、左邊的筷子以及右邊的筷子。

```
def run(self):
    while(self.running):
        #哲學家在思考
        time.sleep(5)
        print ('Philosopher'+str(self.index)+' is hungry.' )
        self.dine()
```

創建一個 run method，如果 run 是 True 的話，就表示哲學家還可以吃，然後呼叫 dine 方法(稍後會提及)。

```python
def dine(self):
    # 如果兩邊筷子沒人用哲學家就會吃
    chopstick1, chopstick2 = self.chopstick_Left, self.chopstick_Right
    while self.running:
        #acquire左邊的筷子
        chopstick1.acquire(True)
        locked = chopstick2.acquire(False)
        #如果右邊的筷子被locked的話,就把左邊的筷子也放回去
        if locked: break
        chopstick1.release()
        print ('Philosopher'+str(self.index)+ ' swaps chopsticks.')
        chopstick1, chopstick2 = chopstick2, chopstick1
    else:
        return
    self.dining()
    #吃完飯後把兩邊筷子都放回去
    chopstick2.release()
    chopstick1.release()
```

```python
def dining(self):
    print ('Philosopher' +str(self.index)+' starts eating. ')
    time.sleep(10)
    print ( 'Philosopher'+str(self.index)+' finishes eating and leaves to think.')
```

再來創建一個 dine method ，用來判斷哲學家是否能進食，給定兩個變數來儲存 class 的參數左邊筷子以及右邊筷子。

當 running 是 True 時(還未結束)，哲學家會先拿起左手邊的筷子(如果沒有人先 acquire 的話，否則他會 wait 到 acquire 到為止)，再來會去 acquire(設定為 false)右邊的筷子,如果早有人先 acquire 的話就會就會直接 release 左邊的筷子，然後 swap 筷子換先去 acquire 另一邊的筷子，一直循環。

如果左右邊筷子都有 acquire 到的話就會呼叫 dining method 印出這位哲學家開始吃飯了，然後給一個時間間隔再印出該哲學家吃完飯了，吃完之後哲學家會把手上的筷子都 release(放下)，回去思考。

```
def main():
    #初始化Semaphore一共五支筷子
    chopsticks = [threading.Semaphore() for n in range(5)]
    print('total chopsticks :',len(chopsticks))
    #初始化哲學家LIST一共五個哲學家（1~5號）每個哲學家會拿起他右手邊的筷子和左手邊的筷子
    philosophers= [Philosopher(i, chopsticks[i%5], chopsticks[(i+1)%5])
            for i in range(1,6)]
    print('total philosophers :',len(philosophers))
    #random.seed(2105)
    Philosopher.running = True
    for i in philosophers: i.start()
    time.sleep(50)
    Philosopher.running = False
    print ("finishing.")


main()
```

建立一個 main function，來實現整個程式 ：
首先先利用 Semaphore 的方式來表示筷子(一共五支筷子，五個 Semaphore)。
再來一共有五個哲學家(1~5 號)，因此創建五個 Philosopher 的物件(再把所有 Philosopher 物件寫成一個 List)，他們分別只會拿起他們左右邊的筷子([i%5]對應其左邊的筷子，[(i+1)%5])對應到其右邊的筷子。
最後利用設定 running 來控制開始與結束。

最重要的技術:
我認為此程式最重要的技術在於下方圖示區塊:

```
chopstick1.acquire(True)
locked = chopstick2.acquire(False)
#如果右邊的筷子被locked的話,就把左邊的筷子也放回去
if locked: break
```

就是說，它不讓只 acquire 到一邊筷子的哲學家繼續拿著那一邊的筷子，如果另一邊有人在使用，那就先放下 acquire 到的那一邊，也就因為如此，可以避免掉死結(如下圖)。