Dining Philosophers

E94106151 游宗謀 作業一

首先，描述一下哲學家就餐問題。有五位哲學家坐在同一個圓桌，他們每個人之間有一支筷子，並共享中間一大碗飯。他們只做兩件事，分別是思考和吃飯，不會跟其他人互動。他們偶爾會嘗試拿起左、右邊的兩支筷子，他們需要一雙筷子才能夾中間的飯，且當他們夾完之後會將兩支筷子放回原位。

接著，這個程式使用到的技術有：Thread, Random, Time。Random 和 Time 負責讓執行緒暫停(sleep)一個隨機的時間。而 Thread 使用到的則是 start, join, Semaphore。Thread.start()會 invoke 作業系統層級的執行緒，讓執行緒從”new” state 變成”runnable” state，接著 Python 直譯器就會開始執行執行緒 run()，而 Thread.join()則是會 block 執行緒直到它完成執行，會等到執行緒 terminate 才會執行接下來的程式。threading.Semaphore(1)則是負責建立筷子和 critical section 的 semaphore，1 代表著只有一個哲學家(執行緒)可以使用筷子或者執行 critical section 的程式碼。

最後，解釋程式運作原理。chopsticks = [threading.Semaphore(1) for i in range(num_chopsticks)]會建立每一隻筷子的 semaphore，控制一支筷子只能被一位哲學家(執行緒)取用，讓程式碼符合題意(deadlock 的互斥條件)；mutex = threading.Semaphore(1)會建立 critical section 的 semaphore，讓 chopsticks.acquire()的程式碼只能被一位哲學家執行。如此一來，才能解決 race condition，不會讓兩位以上的哲學家同時執行 acquire chopsticks，同時也無法達成 deadlock 中 hold and wait 的條件，讓 deadlock 無法發生。philosopher()是一個執行緒函式，定義一位哲學家會思考一段隨機長的時間、拿取筷子(左、右)、吃飯、放回筷子。程式碼最後的三個迴圈則是主程式，分別負責建立執行緒、開始(invoke)執行緒、等待執行緒完成。

我有稍微修改參考網址的程式碼，我將 fork 改成比較符合上課時老師所說的 chopsticks，我們通常不需要兩個叉子才能吃飯，但筷子需要一雙(兩支)。我還將 philosopher()中的”while True:”改成” for i in range(2):”，讓程式不會一直跑下去，陷入無限迴圈。若需要更多，則更改 range()的參數即可。

參考網址：https://www.geeksforgeeks.org/dining-philosopher-problem-using-semaphores/