IUT Robert Schuman
Institut universitaire de technologie
Département informatique
Université de Strasbourg

S4x/ Programmation concurrente (en Python)
Série 4 : Des études de cas

Partie I : Encore une partie de tennis de table...

D'abord nous allons revoir et étudier de plus près le script *ping_pong_cond.py* qui a été traité dans la série3.
Dans ce programme, la synchronisation entre les *ping* et les *pong* était réalisée avec deux variables de type *Condition* (dotées du même *Lock*) et une variable d'état « bascule » (*Switch*).

Principe : On change la valeur de *Switch* lorsqu'un *ping* (ou un *pong*) a été effectué ; le thread (*ping* ou *pong*) **notifie** alors, via la variable de type *Condition*, ce changement d'état à l'un des threads en **attente** de l'autre catégorie.

```
#!/usr/bin/python3
# ping_pong_cond.py

from threading import Thread, Lock, Condition
from random import random
from time import sleep
from sys import argv, stderr

class Ping(Thread) :
    def run(self) :
        global Switch
        sleep(random())         # pour espacer aléatoirement les démarrages des threads
        Cond_ping.acquire()     # idem => Cond_pong.acquire()
        while Switch == 1 :
            Cond_pong.wait()    # en attente d'une notification de pong
        print("ping ...", end=' ')
        Switch = 1
        Cond_ping.notify()
        Cond_ping.release()     # idem => Cond_pong.release()

class Pong(Thread) :
    def run(self) :
        global Switch
        sleep(random())         # pour espacer les démarrages des threads
        Cond_pong.acquire()     # idem => Cond_ping.acquire()
        while Switch == 0 :
            Cond_ping.wait()    # en attente d'une notification de ping
        print("pong")
        Switch = 0
        Cond_pong.notify()
        Cond_pong.release()     # idem => Cond_ping.release()

if argv[1:] :    # si liste des paramètres non vide
    N = int(argv[1])
else :
    N = 100      # 100 ping pong par défaut
Switch = 0 # pour bloquer pong au départ
Mutex = Lock()  # Lock commun aux 2 sortes de threads, en paramètre des 2 variables Condition
Cond_ping = Condition(Mutex)
Cond_pong = Condition(Mutex)
Threads_ping = [ Ping() for i in range(N) ]
Threads_pong = [ Pong() for i in range(N) ]
for t in Threads_ping: t.start()
for t in Threads_pong: t.start()
for t in Threads_ping : t.join()
for t in Threads_pong : t.join()
print("\nFin de partie !")
```

Questions

Rem. : Lisez les deux premières questions « en même temps » ; la deuxième peut donner une piste pour répondre à la première...

1. Le programme « tourne » t-il encore si on remplace la partie de code ① par ② ?

```
Mutex = Lock()
Cond_ping = Condition(Mutex) ; Cond_pong = Condition(Mutex)
```
①

```
Cond_ping = Condition() ; Cond_pong = Condition()
```
②

ce qui est équivalent à :

```
Cond_ping = Condition(lock1) ; Cond_pong = Condition(lock2)
```

2. Quel objet précisément est verrouillé quand on effectue *acquire* sur une variable *Condition* ?
3. Quelles actions « invisibles » sont automatiquement faites lorsqu'un thread entre dans *wait* puis en sort après une notification ?
4. Un *notify* envoie t-il un « signal » à un thread en particulier ? Si oui, lequel ?
   Si un thread effectue une notification et qu'il n'y a pas de thread de l'autre catégorie qui est en attente *wait* à ce moment-là, que devient le signal ?
   Est-ce un problème dans ce script ?
5. Pourquoi la boucle `while Switch == 0 (ou 1)` ne peut-elle pas être remplacée par une "simple" conditionnelle `if Switch == 0 (ou 1)` ?

Exercice `ping_pong_cond_withPriority.py`

Il s'agit de mettre en place un algorithme de type « Premier arrivé, premier servi ».
Un thread *Pong* ne peut pas faire son 'pong' avant un *Pong* qui attendait de pouvoir faire son 'pong'. Autrement dit : un nouvel arrivant *Pong* se met en file d'attente si un *Pong* arrivé avant lui est en file d'attente.
Un thread *Ping* ne peut pas faire son 'ping' avant un *Ping* qui attendait de pouvoir faire son 'ping' .

Indication : une variable de classe supplémentaire dans chacune des deux classes (`waiting`) pour compter le nombre de threads *ping* et *pong* qui sont dans leur file d'attente.

Supplément possible pour les « affamés de code » ...

Repartir de la version *ping_pong_cond.py* (donc sans la gestion de priorité qui vient d'être demandée).
Contrainte : utiliser **une** seule variable de type *Condition* (oui, c'est possible !)
Avertissement : Il y a aussi des changements à faire dans les notifications...

Suite après les vacances...

Partie II : Cas « L'Igloo Bar »