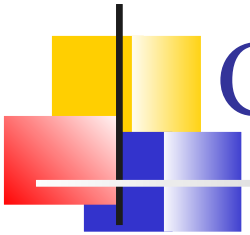


Concurrence : les threads en Python

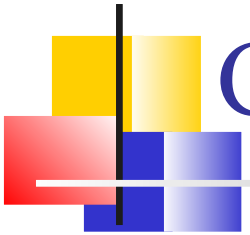
Luc Courtrai

Introduction sur les threads en Python

par les exemples.

Complément sur le cours

"Java threads"



Concurrence : les threads en Python

Python3.XX

- Processus lourds
  - appels système Linux (**os**.fork,os.wait,)
  - module **multiprocessing** (process,semaphore,lock,segment partagé ….
- Processus légers (threads)
  - module threading

Concurrence : les threads en Python

Création d'un thread thread_ex.py
Threading,Thread,start,join,current_thread()

```
import threading

def thread_function( ): # code du thread
    print(f"Thread {threading.current_thread()} start  "   )

if __name__ == "__main__":
    print("Main    : crée et lance le thread")
    x = threading.Thread(target=thread_function)
    x.start()
    print("Main    : join ")
    x.join()
```

```
Main    : crée et lance le thread
Thread <Thread(Thread-8, started 140395857061632)>: start
Main    : join
```

Concurrence : les threads en Python

Création d'un thread argument de la fonction thread

thread_param.py

Thread(args=

```
import threading

def thread_function( arg1,arg2,arg3):

    print("Thread %s: starting   arg",arg1,arg2,arg3  )

if __name__ == "__main__":
    x = threading.Thread(target=thread_function, args=(1,2,3))
    x.start()
    x.join(
```

```
Thread %s: starting   arg 1 2 3
```

Concurrence : les threads en Python

Timer : lancement d'un thread en différé thread_timer.py
Timer

```
from threading import Timer
def thread_function(arg):
    print(f"Thread {threading.current_thread()}: start " )

if __name__ == "__main__":
    print("Main lance thread dans 10 second")
    t=Timer(10.,thread_function,args=(1,))
    t.start()
```

```
Main lance thread dans 10 second
Thread <Timer(Thread-15, started 139990080149248)>: start
```

Concurrence : les threads en Python

Messages (log) des threads : thread_logging.py

```
import threading,logging,time
def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)
if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO,
                        datefmt="%H:%M:%S")
    logging.info("Main    : before creating thread")
    x = threading.Thread(target=thread_function, args=(1,))
    logging.info("Main    : before running thread")
    x.start()
    ...
```

```
10:25:11: Main    : before creating thread
10:25:11: Main    : before running thread
10:25:11: Thread 1: starting
10:25:11: Main    : wait for the thread to finish
10:25:13: Thread 1: finishing
10:25:13: Main    : all done
```

Concurrence : les threads en Python

Verrou thread_lock.py

Lock(acquire realease) + time.sleep

```
import threading,time
cpt=0 # global
mutex=threading.Lock() # un verou
def thread_function( ):
    global cpt, # compteur global
    print(f"Thread {threading.current_thread()}: start " )
    while True :
        mutex.acquire()
        cpt+=1
        print(threading.current_thread(),cpt)
        mutex.release()
        time.sleep(0.25) # leep sur le thread courant
if __name__ == "__main__":
    print("Main ", threading.current_thread())
    for i in range(4) :  # creation de 4 threads
        t=threading.Thread(target=thread_function)
        t.start()
```

Concurrence : les threads en Python

Verrou thread_lock.py

Lock(acquire realease) + time.sleep

```
Main <_MainThread(MainThread, started 140669740459200)>
Thread <Thread(Thread-1, started 140669729257216)>: start
<Thread(Thread-1, started 140669729257216)> 1
Thread <Thread(Thread-2, started 140669720864512)>: start
<Thread(Thread-2, started 140669720864512)> 2
Thread <Thread(Thread-3, started 140669641291520)>: start
Thread <Thread(Thread-4, started 140669632898816)>: start
<Thread(Thread-3, started 140669641291520)> 3
<Thread(Thread-4, started 140669632898816)> 4
<Thread(Thread-1, started 140669729257216)> 5
<Thread(Thread-2, started 140669720864512)> 6
<Thread(Thread-3, started 140669641291520)> 7
<Thread(Thread-4, started 140669632898816)> 8
<Thread(Thread-1, started 140669729257216)> 9
<Thread(Thread-2, started 140669720864512)> 10
```

Concurrence : les threads en Python

Verrou thread_lock.py
with Lock

```
import threading,time
cpt=0 # global
mutex=threading.Lock() # un verou
def thread_function( ):
    global cpt, # compteur global
    print(f"Thread {threading.current_thread()}: start " )
    while True :
        with mutex : #  mutex.release()
            cpt+=1
            print(threading.current_thread(),cpt)
        #  mutex.release()
        time.sleep(0.25) # leep sur le thread courant
if __name__ == "__main__":
    print("Main ", threading.current_thread())
    for i in range(4) :  # creation de 4 threads
        t=threading.Thread(target=thread_function)
        t.start()
```

Concurrence : les threads en Python

Verrou réentrant thread_rlock_.py

RLock(acquire realease)

```
cpt=0 # global
r_mutex=threading.RLock() # un verou réentrant
def getCpt():
    r_mutex.acquire()
    val=cpt
    r_mutex.release()
    return val
def incCpt(inc):
    global cpt
    r_mutex.acquire()
    cpt=getCpt()+inc  #appel getCpt réentrant (getCpt nécessite le verrou)
    r_mutex.release()
def thread_function( ):
    global cpt # compteur global
    while True :
        incCpt(1)
        print(threading.current_thread(),getCpt())
```

Concurrence : les threads en Python

Les Sémaphores : thread_semaphore.py

threading.Semaphore acquire,realease # PAS D'ORDRE FIFO

```
import threading,time
cpt=0 # variable global
mutex=threading.Semaphore(1) #un sémaphore initialisé à 1
                                    #Semaphore(n) n >=0
def thread_function( ):
    global cpt,
    print(f"Thread {threading.current_thread()} starting"   )
    while True :
        mutex.acquire()
        cpt+=1
        print(threading.current_thread(),cpt)
        mutex.release()
        time.sleep(0.25)
if __name__ == "__main__":
    for i in range(4) :
        t=threading.Thread(target=thread_function, args=(1,))
        t.start()
```

Concurrence : les threads en Python

Les Sémaphores : thread_semaphore.py
threading.Semaphore acquire,realease

```
Main <_MainThread(MainThread, started 140424686875840)>
Thread <Thread(Thread-1, started 140424675673856)> starting
<Thread(Thread-1, started 140424675673856)> 1
Thread <Thread(Thread-2, started 140424667281152)> starting
<Thread(Thread-2, started 140424667281152)> 2
Thread <Thread(Thread-3, started 140424658888448)> starting
<Thread(Thread-3, started 140424658888448)> 3
Thread <Thread(Thread-4, started 140424650495744)> starting
<Thread(Thread-4, started 140424650495744)> 4
<Thread(Thread-1, started 140424675673856)> 5
<Thread(Thread-2, started 140424667281152)> 6
<Thread(Thread-3, started 140424658888448)> 7
<Thread(Thread-4, started 140424650495744)> 8
<Thread(Thread-1, started 140424675673856)> 9
```

Concurrence : les threads en Python

Les Variable condition : thread_prod_cons.py
threading.Condition acquire,realease,wait,notify

```
import threading
article=threading.Condition() # objet condition intégrant un verrou
def extraire( liste):
    article.acquire() # demande le verrou  .. synchronized(o){ de java
    while len(liste) ==0:
        article.wait() # bloque le thread courant
    result=liste[0]
    del liste[0]
    article.release() # libère le verrou  .. } du synchronized(o) de java
    return result
def deposer(liste,elt):
    article.acquire() # demande le verrou  .. synchronized(o) de java
    liste.append(elt)
    article.notify()  # réveille un des threads bloquée sur le wait
                      # ou personne
    article.release() # demande le verrou  } du synchronized(o) de java
```

Concurrence : les threads en Python

Les Variable condition : thread_prod_cons.py

threading.Condition acquire,realease,wait,notify

```
def thread_prod( liste): # producteur
    cpt =0
    while True :
        cpt+=1
        print("Producteur ",cpt)
        deposer(liste,cpt)
def thread_cons(  liste): # consommateur
   while True :
        x=extraire (liste)
        print("Consomateur ",x)

if __name__ == "__main__":
    liste=[]
    p=threading.Thread(target=thread_prod, args=(liste,))
    p.start()
    c=threading.Thread(target=thread_cons, args=(liste,))
    c.start()
```

Concurrence : les threads en Python

```
Main    : before creating thread
Thread Producteur   starting    <Thread(Thread-1, started
140602589009664)>
Prod  1
-> deposer [] 1
<- deposer [1] 1
Thread Consomateur   starting    <Thread(Thread-2, started
140602580616960)>
->extraire [1]
<-extraire [] 1
Consomateur  1
Prod  2
-> deposer [] 2
<- deposer [2] 2
->extraire [2]
<-extraire [] 2
Consomateur  2
```

Concurrence : les threads en Python

Les Variable condition : thread_prod_cons.py

threading.Condition avec un Rlock partagé

```
import threading
buffer=[]
MAX=10
rlock=threading.Rlock() #verrou réentrant
article=threading.Condition(rlock) # spécifie le verrou
place=threading.Condition(rlock)  # le même verrou
# evite le double synchronized nécessaire en java wait notify

def extraire( buffer)
    rlock.acquire()
    while len(buffer) ==0:
        article.wait()
    result=buffer[0]
    place.notify() #producteur
    rlock.release()
    return result
```

```
def deposer(buffer,elt):
    rlock.acquire()
    while len(buffer) > MAX: #  place max
        place.wait()
    buffer.append(elt)
    article.notify() #notify consommateur
    rlock.release()
```

Concurrence : les threads en Python

La Barrère : Barrère cyclique synchronisée thread_barrier.py
Barrier,wait

```
from threading import Barrier
def thread_function(bar) :
    print(f"Thread {threading.current_thread()}: start " )
    cpt=0
    while True :
        print(threading.current_thread(),cpt)
        cpt+=1
        bar.wait() # synchronize les 4 threads

if __name__ == "__main__":
    bar = Barrier(4)
    print("Main    : crée et lance le thread")
    for _ in range(4):
       x=threading.Thread(target=thread_function,args=(bar,) )
       x.start()
```

Concurrence : les threads en Python

La Barrère : Barrère cyclique synchronisée thread_barriere.py

```
Thread <Thread(Thread-1, started 140480297232128)>: start
<Thread(Thread-1, started 140480297232128)> 0
Thread <Thread(Thread-2, started 140480288839424)>: start
<Thread(Thread-2, started 140480288839424)> 0
Thread <Thread(Thread-3, started 140480280446720)>: start
Thread <Thread(Thread-4, started 140480272054016)>: start
<Thread(Thread-4, started 140480272054016)> 0
<Thread(Thread-3, started 140480280446720)> 0
<Thread(Thread-1, started 140480297232128)> 1
<Thread(Thread-4, started 140480272054016)> 1
<Thread(Thread-2, started 140480288839424)> 1
<Thread(Thread-3, started 140480280446720)> 1
<Thread(Thread-3, started 140480280446720)> 2
<Thread(Thread-2, started 140480288839424)> 2
<Thread(Thread-4, started 140480272054016)> 2
<Thread(Thread-1, started 140480297232128)> 2
<Thread(Thread-1, started 140480297232128)> 3
```

Concurrence : les threads en Python

Evennement signal : synchronisation simple entre threads

thread_event.py

Event, wait,set

```
cpt=0 # global
def thread_function( synchr):
    global cpt # compteur globa
    while True :
        synchr.wait() # attend l évennement (signal)
        cpt+=1
        print(threading.current_thread(),cpt)
        synchr.set()  # envoie le signal de type synchr

if __name__ == "__main__":
    synchr=threading.Event()
    for i in range(4) :  # creation de 4 threads
        t=threading.Thread(target=thread_function,args=(synchr,))
        t.start()
    synchr.set()  # envoie le signal de type synchr
```

Concurrence : les threads en Python

Evennement signal : synchronisation simple entre threads

thread_event.py

```
Thread <Thread(Thread-1, started 140207157921536)>: start
Thread <Thread(Thread-2, started 140207149528832)>: start
Thread <Thread(Thread-3, started 140207141136128)>: start
Thread <Thread(Thread-4, started 140206925674240)>: start
<Thread(Thread-4, started 140206925674240)> 1
<Thread(Thread-1, started 140207157921536)> 2
<Thread(Thread-3, started 140207141136128)> 3
<Thread(Thread-2, started 140207149528832)> 4
<Thread(Thread-4, started 140206925674240)> 5
<Thread(Thread-3, started 140207141136128)> 6
<Thread(Thread-1, started 140207157921536)> 8
<Thread(Thread-2, started 140207149528832)> 7
<Thread(Thread-4, started 140206925674240)> 9
```

Concurrence : les threads en Python

queue: blocking queue (get FIFO) thread_blocking_queue.py
Queue. queue put get

```
import threading,queue
def thread_prod( aqueue):
    cpt =0
    while True :
        Cpt+=1
        print("Prod ",cpt)
        aqueue.put(cpt)
if __name__ == "__main__":
    aQueue=queue.Queue(
    threading.Thread(target=thread_prod, args=(aQueue,)).start()
    threading.Thread(target=thread_cons, args=(aQueue,)).start()
```

```
def thread_cons(  aqueue):
    while True :
        x=aqueue.get()
        print("Consomateur ",x)
```

```
Thread Producteur   <Thread(Thread-1, started 140044818827008)>
Produteur  1
Thread Consommateur <Thread(Thread-2, started 140044810434304)>
Consommateur  1
Produteur  2
Consommateur  2
Produteur  3
```