06-Concurrency

November 21, 2019

1 Concurrency

- multiprocessing.Process
- threading.Thread

Usage is similar: 1. Create a derived class of Process or Thread. 2. In the constructor call constructor of Process or Thread 3. Implement `run()` method. It will be called when your Process/Thread is `start()`ed.

```
In [14]: from multiprocessing import Process, Lock,RLock, Semaphore
         import time

         class Philosopher(Process):
             def __init__(self,i,left,right):
                 self.left = left
                 self.right = right
                 self.id = i
                 super().__init__()
             def run(self):

                 for i in range(0,5):
                     if self.id % 2 == 0:
                         self.left.acquire()
                         self.right.acquire()
                     else:
                         self.right.acquire()
                         self.left.acquire()
                     print("phil {} is eating".format(self.id))
                     time.sleep(0.2)
                     self.left.release()
                     self.right.release()
                     print("phil {} is thinking".format(self.id))
                     time.sleep(0.1)


         N = 5
```

```
locks = [Lock() for i in range(0,N)]

phils=[]
for i in range(0,N):
    p = Philosopher(i, locks[i], locks[i+1] if i+1 < N else locks[0])
    phils.append(p)

for phil in phils:
    phil.start()

for phil in phils:
    phil.join()
```

```
---------------------------------------------------------------------------

BlockingIOError                           Traceback (most recent call last)

<ipython-input-14-8825929b0a1d> in <module>()
     35
     36 for phil in phils:
---> 37     phil.start()
     38
     39 for phil in phils:


/usr/lib/python3.5/multiprocessing/process.py in start(self)
    103                'daemonic processes are not allowed to have children'
    104         _cleanup()
--> 105         self._popen = self._Popen(self)
    106         self._sentinel = self._popen.sentinel
    107         _children.add(self)


/usr/lib/python3.5/multiprocessing/context.py in _Popen(process_obj)
    210     @staticmethod
    211     def _Popen(process_obj):
--> 212         return _default_context.get_context().Process._Popen(process_obj)
    213
    214 class DefaultContext(BaseContext):


/usr/lib/python3.5/multiprocessing/context.py in _Popen(process_obj)
    265         def _Popen(process_obj):
```

```
    266             from .popen_fork import Popen
--> 267             return Popen(process_obj)
    268 
    269     class SpawnProcess(process.BaseProcess):


    /usr/lib/python3.5/multiprocessing/popen_fork.py in __init__(self, process_obj)
     18         sys.stderr.flush()
     19         self.returncode = None
---> 20         self._launch(process_obj)
     21 
     22     def duplicate_for_child(self, fd):


    /usr/lib/python3.5/multiprocessing/popen_fork.py in _launch(self, process_obj)
     65         code = 1
     66         parent_r, child_w = os.pipe()
---> 67         self.pid = os.fork()
     68         if self.pid == 0:
     69             try:


    BlockingIOError: [Errno 11] Resource temporarily unavailable
```

1.1 multiprocessing

`Process` is the main class. It includes synchronization related classes and data structures:
`Lock, Semaphore, Condition, Value, Array, Queue`
`Process(target=`function`, args=(`arguments`)` will create a new instance (not process yet)
calling `start()` method of the object will create the process and call the parameter function as the entry function.

```
In [ ]: from multiprocessing import Process
        import time

        counter = 10

        def hello(name):
            global counter
            for i in range(0,5):
                counter += 1
                time.sleep(0.5)
                print("hello " + name, counter)

        p = Process(target=hello, args=("world",))

        q = Process(target=hello, args=("myself",))
```

```
p.start()
q.start()
# p, q and main process are concurrent here

p.join() # wait for p to complete
q.join() # wait for q to complete
print('counter is :',counter)
# back to single process again
```

- Multiprocessing environment executes on a separate process. During process creation current set of global variables are copied in a new python interpreter and after that all work isolated.
- `multiprocess` classes are multi-process aware. They are shared. A global lock or locks passed as parameters will be on a shared environment.

```
In [ ]: ''' Simple communication among two processes.
            Locks are logical entities, process do not
            have to own the lock to release it'''



        # Watch this variable. There are two processes incrementing it, three reporting it
        # Each process have its own copy
        counter = 10
        def ping(name,memut,othmut):
            global counter
            for i in range(0,5):
                # wait until my turn
                memut.acquire()
                print(name,counter)
                # tell other end it is its turn
                othmut.release()
                counter += 1

        imut,omut = Lock(), Lock()

        pip = Process(target=ping, args=("ping",imut,omut))
        pop = Process(target=ping, args=("pong",omut,imut))

        # make sure only one (ping enters first)
        omut.acquire()
        pip.start()
        pop.start()
        pip.join()
        pop.join()
        omut.release()
        print("in main process: {}".format(counter))
```