07.09.2018.

The main reference is Python 3.7 online documentation:

<https://docs.python.org/3/library/multiprocessing.html>

Section “17.2.1.5. Sharing state between processes”, subtitle “Server process”

The example code is in the module w1\_manager.py.

Section “17.2.2.7. Managers”

One should read this section in detail.

Tryout this simple example: copy the following code into Python interpreter in the 1st terminal (Server):

|  |
| --- |
| from multiprocessing.managers import BaseManager  manager = BaseManager(address=('', 50000), authkey=b'abc')  server = manager.get\_server()  server.serve\_forever() |

Copy the following code in Python interpreter in the 2nd terminal (Client):

|  |
| --- |
| from multiprocessing.managers import BaseManager  m = BaseManager(address=('127.0.0.1', 50000), authkey=b'abc')  m.connect() |

Section “17.2.2.7.1. Customized managers”

The example is in the module w2\_customized\_manager.py.

Section “17.2.2.7.2. Using a remote manager”

Tryout this example: copy the following code into Python interpreter in the 1st terminal (Server):

|  |
| --- |
| from multiprocessing.managers import BaseManager  from queue import Queue  queue = Queue()  class QueueManager(BaseManager):  pass  QueueManager.register('get\_queue', callable=lambda:queue)  m = QueueManager(address=('', 50000), authkey=b'abracadabra')  s = m.get\_server()  s.serve\_forever() |

Copy the following code in Python interpreter in the 2nd terminal (Client 1) on the same computer (if you want to run it from different computer, set address below to address of Server computer, e.g. 'foo.bar.org'):

|  |
| --- |
| from multiprocessing.managers import BaseManager  class QueueManager(BaseManager):  pass  QueueManager.register('get\_queue')  m = QueueManager(address=('127.0.0.1', 50000), authkey=b'abracadabra')  m.connect()  queue = m.get\_queue()  queue.put('hello') |

Copy the following code in Python interpreter in the 3rd terminal (Client 2) on the same computer:

|  |
| --- |
| from multiprocessing.managers import BaseManager  class QueueManager(BaseManager):  pass  QueueManager.register('get\_queue')  m = QueueManager(address=('127.0.0.1', 50000), authkey=b'abracadabra')  m.connect()  queue = m.get\_queue()  queue.get() |

The second example is with the manager and the local client process at the same computer.

In the 1st terminal run w3\_manager\_and\_local\_client.py: python w3\_manager\_and\_local\_client.py (this Server and local Client)

Copy the following code in Python interpreter in the 2nd terminal (remote Client):

|  |
| --- |
| from multiprocessing.managers import BaseManager  class QueueManager(BaseManager):  pass  QueueManager.register('get\_queue')  m = QueueManager(address=('127.0.0.1', 50000), authkey=b'abracadabra')  m.connect()  queue = m.get\_queue()  queue.get() |

Expected result for the remote client: it should return the message send by the local client, which is 'local hello'.

Section “17.2.2.8. Proxy Objects”

Tryout this example: copy the following code into Python interpreter:

|  |
| --- |
| from multiprocessing import Manager  manager = Manager()  l = manager.list([i\*i for i in range(10)])  print(l)  print(repr(l))  l[4]  l[2:5] |

The following is the example with proxy nesting (b is nested in a). Continue the previous session without exiting from the interpreter:

|  |
| --- |
| a = manager.list()  b = manager.list()  a.append(b) # referent of a now contains referent of b  print(a, b)  b.append('hello')  print(a[0], b) |

Similarly, dict and list proxies may be nested inside one another. Continue the previous session:

|  |
| --- |
| l\_outer = manager.list([ manager.dict() for i in range(2) ])  d\_first\_inner = l\_outer[0]  d\_first\_inner['a'] = 1  d\_first\_inner['b'] = 2  l\_outer[1]['c'] = 3  l\_outer[1]['z'] = 26  print(l\_outer[0])  print(l\_outer[1]) |

Warning: Updates of container proxy must be made explicitly. Continue the previous session:

|  |
| --- |
| # create a list proxy and append a mutable object (a dictionary)  lproxy = manager.list()  lproxy.append({})  # now mutate the dictionary  d = lproxy[0]  d['a'] = 1  d['b'] = 2  # at this point, the changes to d are not yet synced, but by  # updating the dictionary, the proxy is notified of the change  lproxy[0] = d |

Note: The proxy types in multiprocessing do nothing to support comparisons by value. Continue with:

|  |
| --- |
| manager.list([1,2,3]) == [1,2,3] |

One should just use a copy of the referent instead when making comparisons.

Exercise: make a copy of the referent and compare it with [1,2,3]. Too simple, OK here it is:

|  |
| --- |
| cp = []  for i in range (0,3):  cp.append(l123[i])  print(cp)  cp == [1,2,3] |

Important: \_callmethod calls and returns the result of a method of the proxy’s referent. Continue:

|  |
| --- |
| l = manager.list(range(10))  l.\_callmethod('\_\_len\_\_')  l.\_callmethod('\_\_getitem\_\_', (slice(2, 7),)) # equivalent to l[2:7]  l.\_callmethod('\_\_getitem\_\_', (20,)) # equivalent to l[20] |

The end.