SyncManager

This lesson discusses the SyncManager from the multiprocessing module which offers proxies for shared objects.

Sync Manager

So far we have been using **`time.sleep()`** to delay the server manager from shutting down before other processes had a chance to retrieve shared objects from it. Obviously, this isn't a good strategy and we need a better way to coordinate among the different processes. Fortunately, there's a subclass of **`BaseManager`** called the **`SyncManager`** which offers proxies of synchronization constructs from the **`threading`** module. Let's revisit the example we discussed under the manager section where we used a timer to delay shutting down the manager. We'll replace the base manager with a sync manager and create a semaphore proxy using the sync manager. The main process would wait on the semaphore while the child process after accessing the shared objects would set the semaphore. The code appears below:

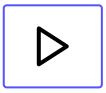
SyncManager implements tools for synchronizing inter-process communication in the style of threaded programming.

Using sync manager

```
from multiprocessing.managers import SyncManager
from multiprocessing import Process
import time

# port number on which the manager runs and another can
# connect at
port_num = 55555


def process_task(sem):
    manager = SyncManager(address=('127.0.0.1', port_num))
    manager.register('get_my_string')
    manager.connect()
    proxy = manager.get_my_string()

    print(repr(proxy))
    print(str(proxy))
    print("child process exiting in 5 seconds")

    time.sleep(5)
    sem._callmethod('release')

    # invoking methods on the proxy's referent
    print(proxy._callmethod('isdigit'))
    print(proxy._callmethod('capitalize'))


if __name__ == '__main__':
    manager = SyncManager(address=('127.0.0.1', port_num))

    # Register our type
    my_string = "educative"
    manager.register('get_my_string', callable=lambda: my_stri
ng)
    manager.start()

    # get a proxy for a Semaphore
    sem = manager.Semaphore(0)
```

```
# pass the semahore to the other process
p = Process(target=process_task, args=(sem,))
p.start()

# wait for the semaphore to be set
sem._callmethod('acquire')

print("Main process exiting")
```


```
from multiprocessing.managers import SyncManager
from multiprocessing import Process
import time

# port number on which the manager runs and another can
# connect at
port_num = 55555


def process_task(sem):
    manager = SyncManager(address=('127.0.0.1', port_num))
    manager.register('get_my_string')
    manager.connect()
    proxy = manager.get_my_string()

    print(repr(proxy))
    print(str(proxy))
    print("child process exiting in 5 seconds")

    time.sleep(5)
    sem._callmethod('release')

    # invoking methods on the proxy's referent
    print(proxy._callmethod('isdigit'))
    print(proxy._callmethod('capitalize'))


if __name__ == '__main__':
    manager = SyncManager(address=('127.0.0.1', port_num))

    # Register our type
    my_string = "educative"
    manager.register('get_my_string', callable=lambda: my_string)
    manager.start()

    # get a proxy for a Semaphore
    sem = manager.Semaphore(0)

    # pass the semahore to the other process
    p = Process(target=process_task, args=(sem,))
    p.start()

    # wait for the semaphore to be set
    sem._callmethod('acquire')

    print("Main process exiting")
```




The **`SyncManager`** also has equivalent methods that return proxies for **`condition`**, **`Event`**, **`Lock`** and **`BoundedSemaphore`** objects.

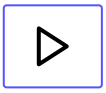
The **`SyncManager`** is more versatile and also offers arrays, value, dictionary and list proxies. Below is a program that creates a list proxy and shares it with two processes. The first process, changes the element in the first index and the change is visible to the second process. Similarly, the second process makes a change that becomes visible to the first process. Follow the print messages to understand how the list gets manipulated between the two processes.

Using a list proxy

```
from multiprocessing.managers import SyncManager
from multiprocessing import Process

# port number on which the manager runs and another can
# connect at
port_num = 55555


def process1_task(sem1, sem2, lst):
    lst[0] = 1
    sem1.release()
    sem2.acquire()
    print("process 1 sees list as " + str(lst))
    print("process 1 exiting")


def process2_task(sem1, sem2, lst):
    sem1.acquire()
    print("process 2 sees list as " + str(lst))
    lst[0] = 2
    sem2.release()
    print("process 2 exiting")


if __name__ == '__main__':
    manager = SyncManager(address=('127.0.0.1', port_num))
    manager.start()

    # get a proxy for a Semaphore
    sem1 = manager.Semaphore(0)
    sem2 = manager.Semaphore(0)
    lst = manager.list([0, 0, 0])
    # lst = list([0, 0, 0])

    # create first process
    p1 = Process(target=process1_task, args=(sem1, sem2, lst))
    p1.start()
```

```
# create second process
p2 = Process(target=process2_task, args=(sem1, sem2, lst))
p2.start()

p1.join()
p2.join()

print("Main process exiting")
```


```
from multiprocessing.managers import SyncManager
from multiprocessing import Process

# port number on which the manager runs and another can
# connect at
port_num = 55555


def process1_task(sem1, sem2, lst):
    lst[0] = 1
    sem1.release()
    sem2.acquire()
    print("process 1 sees list as " + str(lst))
    print("process 1 exiting")


def process2_task(sem1, sem2, lst):
    sem1.acquire()
    print("process 2 sees list as " + str(lst))
    lst[0] = 2
    sem2.release()
    print("process 2 exiting")


if __name__ == '__main__':
    manager = SyncManager(address=('127.0.0.1', port_num))
    manager.start()

    # get a proxy for a Semaphore
    sem1 = manager.Semaphore(0)
    sem2 = manager.Semaphore(0)
    lst = manager.list([0, 0, 0])
    # lst = list([0, 0, 0])

    # create first process
    p1 = Process(target=process1_task, args=(sem1, sem2, lst))
    p1.start()

    # create second process
    p2 = Process(target=process2_task, args=(sem1, sem2, lst))
    p2.start()

    p1.join()
    p2.join()

    print("Main process exiting")
```




In the above program, if you uncomment **line#33** and use a vanilla list, you'll see that the changes made by the processes aren't visible to each other.

Nested Lists

When working with lists, it might be of interest to wonder what will be the behavior of the proxy if we manipulate nested lists. Consider the program below:

```
from multiprocessing.managers import SyncManager
from multiprocessing import Process
import multiprocessing

# port number on which the manager runs and another can
# connect at
port_num = 55555


def process1_task(lstProxy):
    lstProxy[0] = 1
    print("Child process sees list as " + str(lstProxy))
    nested_list = list({})
    lstProxy.append(nested_list)

    nested_list.append(99)
    nested_list.append(98)
    nested_list.append(97)
    print("Child process sees nested list as: " + str(nested_list))
    print("Child process sees lstProxy as: " + str(lstProxy))



if __name__ == '__main__':
    multiprocessing.set_start_method("spawn")
    manager = SyncManager(address=('127.0.0.1', port_num))
    manager.start()
    lst = manager.list([0, 0, 0])

    # create first process
    p1 = Process(target=process1_task, args=(lst,))
```

```
p1.start()
p1.join()
```

We append a nested list to the list proxy object, which is reflected in the original list in the main process. However, when we manipulate the nested list by adding elements to it, the change isn't reflected in the original list. One way to fix the situation would be to reassign the list to the same index as below:

```
lstProxy[3] = nested_list
```

The program works correctly with the change as follows:

```
from multiprocessing.managers import SyncManager
from multiprocessing import Process
import multiprocessing

# port number on which the manager runs and another can
# connect at
port_num = 55555


def process1_task(lstProxy):
    lstProxy[0] = 1
    print("process 1 sees list as " + str(lstProxy))
    nested_list = list({})
    lstProxy.append(nested_list)

    nested_list.append(99)
    nested_list.append(98)
    nested_list.append(97)
    print("Child process sees lstProxy as: " + str(lstProxy))
    lstProxy[3] = nested_list
    print("Child process sees lstProxy as: " + str(lstProxy))


if __name__ == '__main__':
    multiprocessing.set_start_method("spawn")
    manager = SyncManager(address=('127.0.0.1', port_num))
    manager.start()
    lst = manager.list([0, 0, 0])

    # create first process
    p1 = Process(target=process1_task, args=(lst,))
    p1.start()
    p1.join()
```

Yet another way would be to nest two managed list proxies together. Shared object containers such as lists and dictionaries can contain other managed container objects which will be synchronized and managed by the **`SyncManager`**.

Back

Next

... continued

Working with Managers

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/syncmanager__multiprocessing__python-concurrency-for-senior-engineering-interviews)