... continued

Continues the discussion on using a manager to share objects among processes.

In the previous section we retrieved the server object from the manager and then let the server serve forever. There's also another way to run the server and doesn't require us to retrieve the server object first. We can invoke the **`start()`** method on the manager. The code from the previous section is refactored below:

Using manager's start method

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

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


def process_task():
    manager = BaseManager(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))


if __name__ == '__main__':
    manager = BaseManager(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()

    p = Process(target=process_task)
    p.start()

    time.sleep(10)
    print("Exiting main process")
    manager.shutdown()
```

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

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


def process_task():
    manager = BaseManager(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(proxy.isdigit())
    print(proxy.capitalize())


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

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

    p = Process(target=process_task)
    p.start()

    time.sleep(3)
    print("Exiting main process")
    manager.shutdown()
```

The **`start()`** call is not a blocking call, therefore we add a sleep of a few seconds before shutting down the manager. We'll see in later sections how we can avoid using the sleep call.

BaseManager

The base manager class, as the name implies is the base class of all managers. We can also extend it to create our own customized managers. When we start the manager or retrieve the server object and invoke serve method on it, another python server process is created internally. You can verify it by running any of the two scripts on your machine and then executing **ps -aef | grep python** in the terminal to see two or three processes. If the child process hasn't exited yet you'll see three processes otherwise two.

The astute reader would have noticed that we have not used the `_callmethod()` to invoke `capitalize()` and `isdigit()` methods on the proxy object in the above example. Instead we invoked the string methods directly on the proxy object. The proxy allows us to invoke public or exposed methods directly.

We can also share far more complex than simple types when using managers. The next section shows how can we share classes.

Base Proxy

The proxy object returned in the above example is of type `AutoProxy` which derives from the base class `BaseProxy`. The base proxy is the base class of various types of proxies e.g. `ListProxy`, `AcquirerProxy`, `EventProxy` and `IteratorProxy` etc.

In the next section, we'll see how we can specify the manager what proxy to return.

Custom Manager

We can also create a customized manager by derving from the `BaseManager` class. Consider the following `MyManager` class:

Creating custom manager

```
class MyManager(BaseManager):
    pass
```

We can register types with the manager class itself instead of an instance of the manager. Let's say there's a utility class with functionality for manipulating strings. We can register it like so:

```
class Utility:
    def capitalize(self, name):
        return name.capitalize()


class MyManager(BaseManager):
    pass


MyManager.register('UtilityClass', Utility)
```

The complete runnable code appears below:

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

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


class Utility:
    def capitalize(self, name):
        return name.capitalize()


class MyManager(BaseManager):
    pass


MyManager.register('UtilityClass', Utility)


def process_task():
    my_manager = MyManager(address=('127.0.0.1', port_num))
    my_manager.register('UtilityClass')
    my_manager.connect()

    utility = my_manager.UtilityClass()
    print(utility.capitalize("hello"))


if __name__ == '__main__':
    my_manager = MyManager(address=('127.0.0.1', port_num))

    my_manager.start()
    Process(target=process_task).start()

    time.sleep(3)
    print("Main process exiting")
```

← Back
Manager

Next →
... continued

✓ Mark as Completed

Report an Issue

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