Working with Managers

This lesson describes how to implement basic security when working with managers, and caveats to be mindful of.

Working with Managers

Using managers we can start processes on separate hosts that can share data amongst themselves. Once connected, there is no difference in the way processes use shared resources whether running locally or remotely.

Secure Communication

A manager can be created by specifying an **authkey** which must be a byte string. **authkey** is the authentication key which will be used to check the validity of incoming connections to the server process. If **authkey** is None then `current_process().authkey` is used. The manager created in the consumer process must also use the same **authkey** to connect with the manager in the main process.

Objects are passed between client and server via pickling, which can introduce security holes as unpacking a pickle may cause code to be executed. Pickles from an unknown source should never be trusted. To mitigate this risk, all communication in the processing package can be secured with *digest authentication* using the `hmac` module from the standard library. Callers can pass authentication keys to the manager explicitly, but default values are generated if no key is passed.

We'll rewrite a basic example using the **authkey**.

```
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), authkey=b'MySecretKey')
    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), authkey=b'MySecretKey')

    # 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()
```

Notes

Following are a few salient points to consider when working with managers:

- Proxy objects are picklable and can be passed between processes.
- Several proxy objects can point to the same shared object.
- Arguments to proxy methods should be pickable.
- Proxy objects should be appropriately guarded with locks if accessed by multiple threads with the same process.
- A proxy object has a weakref callback that allows it to deregister itself from the manager containing its referent when the proxy object gets garbage collected.
- Shared objects get deleted from the manager process when no proxy refers to the object.

Back

SyncManager

Next

Namespace

Mark as Completed

Report an Issue

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