Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Remote Shared Objects
Python – multiprocessing.Manager

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Objectives

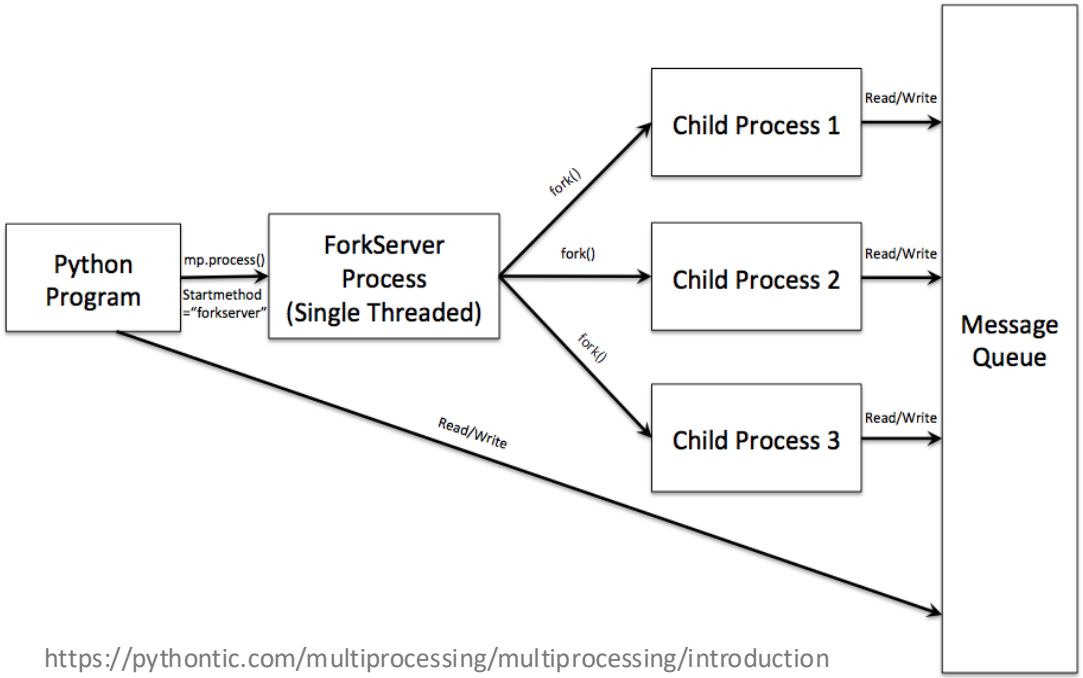
- Understand the concept of Python's multiprocessing.Manager() for Remote Objects
- Understand the concept and the limitations of remote shared objects
- Examples of Python multiprocessing.Manager()
- Exercises

Multiprocessing Manager

- Managers provide a way to create data which can be **shared** between different processes
  - Including sharing over a network between processes running on different machines
- A manager object controls a server process which manages *shared objects*. Other processes can access the shared objects by using proxies

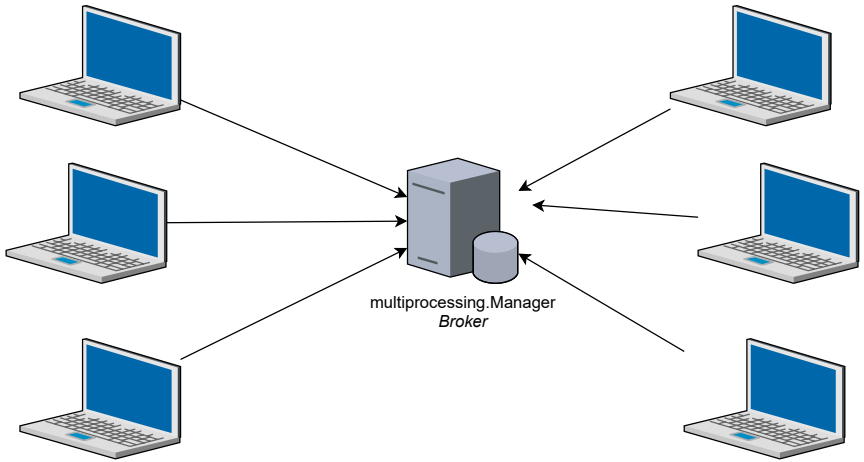
Multiprocessing - the processes

Method "forkserver" creates a separate single threaded server process to handle the process creation requests of the parent process. Since the server process is a single threaded process, multithreaded processes can be created without any side effects.



https://pythontic.com/multiprocessing/multiprocessing/introduction

Remote Shared Objects – Python



The remote manager – broker

```
queue = Queue()

class MyRemoteObjectManager(BaseManager): pass

mgr = MyRemoteObjectManager(
     address=('', MANAGER_PORT),
     authkey=MANAGER_PW)

mgr.register('my_queue', callable=lambda: queue)

s = mgr.get_server()
s.serve_forever()
```

A new class for the remote object Manager, derived from BaseManager

Create a Manager object. Port and PW must be provided

Registering all needed remote objects, with a specific name → 'my_queue'

Finally get a remote object server from the created manager and start it (serving)

The clients, using the remote manager

```
m = BaseManager(
    address=(MANAGER_ADDRESS, MANAGER_PORT), authkey=MANAGER_PW)

m.register('my_queue')

try:
    m.connect()
except :...

remote_queue = m.my_queue()

# use the queue:
remote_queue.put(data)
data = remote_queue.get()
```

Create a Manager object. Address, port and PW must be provided

Registering all needed remote objects, with the name specified in the broker → 'my_queue'

Connect to the broker

Create the (proxy) object

Use the object and its method as usual

Key take-away messages

- Manager is the orchestrator for the shared objects
- Objects can be registered and used “on the fly”
- Distributed computing with Python’s on-board tools is quite low-level
  - Broker/Queueing services exists and offers a lot of additional features
    - RabbitMQ, ZeroMQ, KubeMQ, …

Exercise

- **Ex21 – Distributed async. queue / semaphores**

- A remote manager (broker) is installed on a VM at HEIA-FR, accessible from Internet. The broker offers a queue and a semaphore
- Play around with the proposed test scripts
- You should work in groups to create a small test program that acquires and releases this shared-remote semaphore

https://concurp2-lecture.kube.ext.isc.heia-fr.ch/exercises/ex21/