10-Server-Models

November 21, 2019

1 Server Models

- Problem: concurrent connections. How to process them
- One solution is single thread/process:
  - Block on multiple sockets. This is possible but hard to write and maintain
- We can create a concurrent agent per connection. Workers.

1. Process per connection
2. Thread per connection

1.1 Thread per connection

```
s.listen(10)
while True:
    ns,peer = s.accept()
    t = Thread(target=service, args=(ns,...))
    t.start()
```

Advantages: * Threads are light * Already shared data. Creating a shared object is easy.

Disadvantages: * GIL cannot utilize multiple processes (specific to Python) * Resource limits apply to process, all threads share them (file descriptors, memory, CPU, stack) * Bugs causing exceptions and memory leakage in one thread will affect all connections

1.2 Process per connection

```
s.listen(10)
while True:
    ns,peer = s.accept()
    t = Process(target=service, args=(ns,...))
    ns.close()
    t.start()
```

Disadvantages: * Shared data should be on shared memory. Use `Value`, `Array`, `Queue`, `Manager`,.. for shared information. * IPC synchronization is more expensive. * Creating a process is more expensive. Memory, starting cost.

Advantages: * Each connection has independent resources * Each connection can get only its exceptions and errors. Others isolated. * Python can use multiple processors.

What if we need to put an upper bound. What if we like to get rid of startup cost for threads and processes.

1.3 Pool based services

We create threads/processes in advance. They serve multiple connections. Reuse service objects for multiple connections. Increase #of conections that can be handled in a short period.

Also it is possible to grow and shrink number of processes/threads.

```
class Service(Process):
    def __init__(self,sock,...):
        self.sock = sock
        super().__init__()
    def run(self):
            while True:
                ns, peer = self.sock.accept()
                echoservice(ns)
                # connection over, ready to get next connection


s=socket(AF_INET,...)
s.bind(.;..)
s.listen(10)

poolsize = 40
pool = [Service(s) for i in range(poolsize)]
for p in pool:
    p.start()

....
```

- If initialization cost is significant with respect to responsiveness of your protocol, creating a thread/process per connection is too expensive. so create in advance and reuse existing threads/processes will be more practical.