10-Server-Models

January 7, 2026

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,...))
    t.start()
    ns.close()
```

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.

[4]:
```
from multiprocessing import Process, Lock, RLock, Queue, Value, Array
import random
import time
from socket import *

def echoservice(sock,i):
    ''' echo uppercase string back in a loop'''
    try:
        while True:
            ns, peer = sock.accept()
            req = ns.recv(1000)
            while req and req != '':
                # remove trailing newline and blanks
                req = req.rstrip()
                print(i,'serving',req.decode())
                ns.send(req.decode().upper().encode())
```

```
                req = ns.recv(1000)
            print(peer, ' closing')
    finally:
        pass


def client(n, port):
    # send n random request
    # the connection is kept alive until client closes it.
    mess = ['hello', 'bye', 'why', 'yes', 'no', 'maybe', 'are you sure', 'why␣
 ↪not?']
    c = socket(AF_INET, SOCK_STREAM)
    c.connect(('127.0.0.1', port))
    for i in range(n):
        time.sleep(random.random()*3)
        c.send(random.choice(mess).encode())
        reply = c.recv(1024)
        print(c.getsockname(), reply)
    c.close()


s = socket(AF_INET, SOCK_STREAM)
s.bind(('',20446))
s.listen(1)    # 1 is queue size for "not yet accept()'ed connections"

# create 3 workers
workers = [Process(target = echoservice, args=(s,i)) for i in range(3)]
for w in workers: w.start()

# create 10 clients
clients = [Process(target = client, args=(5, 20446)) for i in range(30)]
# start clients
for cl in clients: cl.start()

# just for python book not to keep port open. Kill all processes and close the␣
 ↪socket
time.sleep(20)
for w in workers: w.terminate()
s.close()
```

```
1 serving why not?
('127.0.0.1', 33412) b'WHY NOT?'
1 serving bye0
('127.0.0.1', 33412)  servingb'BYE' bye

('127.0.0.1', 33410) b'BYE'
1 serving maybe
```

```
('127.0.0.1', 33412) b'MAYBE'
1 serving why not?
('127.0.0.1', 33412) b'WHY NOT?'
1 serving hello
('127.0.0.1', 33412) b'HELLO'
('127.0.0.1', 33412)  closing
1 serving are you sure
('127.0.0.1', 33432) b'ARE YOU SURE'
0 serving why not?
('127.0.0.1', 33410) b'WHY NOT?'
2 serving why
('127.0.0.1', 33418) b'WHY'
1 serving no
('127.0.0.1', 33432) b'NO'
0 serving why not?
('127.0.0.1', 33410) b'WHY NOT?'
2 serving maybe
('127.0.0.1', 33418) b'MAYBE'
1 serving yes
('127.0.0.1', 33432) b'YES'
0 serving no
('127.0.0.1', 33410) b'NO'
2 serving no
('127.0.0.1', 33418) b'NO'
1 serving hello
('127.0.0.1', 33432) b'HELLO'
2 serving hello
('127.0.0.1', 33418) b'HELLO'
0 serving yes
('127.0.0.1', 33410) b'YES'
('127.0.0.1', 33410)  closing
0 serving why
('127.0.0.1', 33438) b'WHY'
1 serving why
('127.0.0.1', 33432) b'WHY'
('127.0.0.1', 33432)  closing
1 serving why
('127.0.0.1', 33456) b'WHY'
0 serving why
('127.0.0.1', 33438) b'WHY'
2 serving hello
('127.0.0.1', 33418) b'HELLO'
('127.0.0.1', 33418)  closing
2 serving maybe
('127.0.0.1', 33450) b'MAYBE'
1 serving why
('127.0.0.1', 33456) b'WHY'
2 serving yes
```

```
('127.0.0.1', 33450) b'YES'
0 serving are you sure
('127.0.0.1', 33438) b'ARE YOU SURE'
1 serving yes
('127.0.0.1', 33456) b'YES'
2 serving why not?
('127.0.0.1', 33450) b'WHY NOT?'
0 serving why not?
('127.0.0.1', 33438) b'WHY NOT?'
1 serving maybe
('127.0.0.1', 33456) b'MAYBE'
2 serving why not?
('127.0.0.1', 33450) b'WHY NOT?'
0 serving yes
('127.0.0.1', 33438) b'YES'
('127.0.0.1', 33438)  closing
0 serving yes
('127.0.0.1', 33464) b'YES'
2 serving hello
('127.0.0.1', 33450) b'HELLO'
('127.0.0.1', 33450)  closing
2 serving hello
('127.0.0.1', 33650) b'HELLO'
1 serving maybe
('127.0.0.1', 33456) b'MAYBE'
('127.0.0.1', 33456)  closing
1 serving maybe
('127.0.0.1', 33658) b'MAYBE'
0 serving why
('127.0.0.1', 33464) b'WHY'
2 serving yes
('127.0.0.1', 33650) b'YES'
2 serving bye
('127.0.0.1', 33650) b'BYE'
2 serving why not?
('127.0.0.1', 33650) b'WHY NOT?'
2 serving why
('127.0.0.1', 33650) b'WHY'
('127.0.0.1', 33650)  closing
2 serving hello
('127.0.0.1', 33586) b'HELLO'
1 serving hello
('127.0.0.1', 33658) b'HELLO'
0 serving hello
('127.0.0.1', 33464) b'HELLO'
('127.0.0.1', 33586) b''
('127.0.0.1', 33658) b''
('127.0.0.1', 33464) b''
```

```
Process Process-71:
Traceback (most recent call last):
  File "/usr/lib/python3.13/multiprocessing/process.py", line 313, in _bootstrap
    self.run()
    ~~~~~~~~^^
  File "/usr/lib/python3.13/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
    ~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/tmp/ipykernel_3853590/514158599.py", line 31, in client
    c.send(random.choice(mess).encode())
    ~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
BrokenPipeError: [Errno 32] Broken pipe
Process Process-63:
Traceback (most recent call last):
  File "/usr/lib/python3.13/multiprocessing/process.py", line 313, in _bootstrap
    self.run()
    ~~~~~~~~^^
  File "/usr/lib/python3.13/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
    ~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/tmp/ipykernel_3853590/514158599.py", line 31, in client
    c.send(random.choice(mess).encode())
    ~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
BrokenPipeError: [Errno 32] Broken pipe
Process Process-50:
Traceback (most recent call last):
  File "/usr/lib/python3.13/multiprocessing/process.py", line 313, in _bootstrap
    self.run()
    ~~~~~~~~^^
  File "/usr/lib/python3.13/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
    ~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/tmp/ipykernel_3853590/514158599.py", line 31, in client
    c.send(random.choice(mess).encode())
    ~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
BrokenPipeError: [Errno 32] Broken pipe
```

[5]:
```
''' Thread pool example. Using slightly complex mechanism to transfer socket
    to threads in the pool (through a shared variable and condition variables).
    This way, load balancing and better pool control is possible. (not␣
 ↪implemented here
     but possible)'''
from socket import *
import time
import random
from threading import Thread, Condition, Lock
```

```
terminate = False

def echoservice(sockets, myid, mycond, done):
    ''' echo uppercase string back in a loop'''
    while not terminate:
        with mycond:
            while not terminate and  type(sockets[myid]) != socket:
                mycond.wait()
            if terminate:
                break
        print(myid, "serving", sockets[myid].getpeername())
        req = sockets[myid].recv(1000)
        while req and req != '':
            # remove trailing newline and blanks
            req = req.rstrip()
            sockets[myid].send(req.decode().upper().encode())
            req = sockets[myid].recv(1000)
        print(sockets[myid].getpeername(), ' closing')
        sockets[myid].close()
        with mycond:
            sockets[myid] = 'FREE'
            done.notify()
    if type(sockets[myid]) == socket:
        sockets[myid].close()



def client(n, port):
    # send n random request
    # the connection is kept alive until client closes it.
    mess = ['hello', 'bye', 'why', 'yes', 'no', 'maybe', 'are you sure', 'why␣
 ↪not?']
    c = socket(AF_INET, SOCK_STREAM)
    c.connect(('127.0.0.1', port))
    for i in range(n):
        time.sleep(random.random()*3)
        c.send(random.choice(mess).encode())
        reply = c.recv(1024)
        print(c.getsockname(), reply)
    c.close()

def checkfree(s):
    try:
        r = s.index('FREE')
        return r
    except:
        return None
```

```
N = 3

sockets = ['FREE'] * N
slock = Lock()
conditions = [ Condition(slock) for i in range(N)]
clientready = Condition(slock)

# create 3 workers
workers = [Thread(target = echoservice, args=(sockets, i, conditions[i],␣
 ↪clientready)) for i in range(N)]
for w in workers: w.start()

s = socket(AF_INET, SOCK_STREAM)
s.bind(('',20447))
s.listen(5)    # 1 is queue size for "not yet accept()'ed connections"

# create 5 clients
clients = [Process(target = client, args=(5, 20447)) for i in range(5)]
# start clients
for cl in clients: cl.start()

s.settimeout(20)     # special to python notebook. if no incoming connection␣
 ↪for 20 seconds, terminate!!
# otherwise python notebook kernel stays alive occupying the port
try:
    while True:
        ns, peer = s.accept()
        with slock:
            r = checkfree(sockets)
            while r == None:
                clientready.wait()
                r = checkfree(sockets)
            sockets[r] = ns
            conditions[r].notify()
except Exception as e:
    print("exitting", str(e), ". Will wait for ongoing connections to close")

terminate = True

for i in range(N):    # let all workers terminate
    with slock:
        conditions[i].notify()


for w in workers: w.join()
```

```
s.close()
# just for python book not to keep port open. Kill all processes and close the␣
  ↪socket
for cl in clients:
    cl.terminate()
    cl.join()
print("complete")
```

```
1 serving ('127.0.0.1', 55736)
0 serving ('127.0.0.1', 55722)
2 serving ('127.0.0.1', 55740)
('127.0.0.1', 55740) b'WHY NOT?'
('127.0.0.1', 55722) b'WHY'
('127.0.0.1', 55736) b'BYE'
('127.0.0.1', 55740) b'WHY NOT?'
('127.0.0.1', 55736) b'HELLO'
('127.0.0.1', 55722) b'ARE YOU SURE'
('127.0.0.1', 55736) b'ARE YOU SURE'
('127.0.0.1', 55722) b'YES'
('127.0.0.1', 55740) b'WHY'
('127.0.0.1', 55740) b'WHY'
('127.0.0.1', 55722) b'NO'
('127.0.0.1', 55722) b'MAYBE'
('127.0.0.1', 55756) b'HELLO'
('127.0.0.1', 55722)  closing
0 serving ('127.0.0.1', 55756)
('127.0.0.1', 55736) b'WHY NOT?'
('127.0.0.1', 55740) b'WHY'
('127.0.0.1', 55758) b'WHY NOT?'
('127.0.0.1', 55740)  closing
2 serving ('127.0.0.1', 55758)
('127.0.0.1', 55758) b'YES'
('127.0.0.1', 55756) b'ARE YOU SURE'
('127.0.0.1', 55736) b'MAYBE'
('127.0.0.1', 55756) ('127.0.0.1', 55736)  closing
b'YES'
('127.0.0.1', 55756) b'BYE'
('127.0.0.1', 55756) b'WHY'
('127.0.0.1', 55756)  closing
('127.0.0.1', 55758) b'YES'
('127.0.0.1', 55758) b'YES'
('127.0.0.1', 55758) b'ARE YOU SURE'
('127.0.0.1', 55758)  closing
exitting timed out . Will wait for ongoing connections to close
complete
```

[ ]: