

操作系统第四章作业

计试 81 白思雨 2186123935

一、题目

编写一个 C/S 架构的分布式程序，Server 接受 Client 发来的请求，执行一个计算 $F(X)$ 并给 Client 返回结果；分别用进程与线程作为服务器 Server 实现，并比较服务器的开销。可以在一台机器上模拟。

二、实验过程

用 python 实现 Client 和 Server，实现 Client 发送一个数，Server 返回这个数的 4 次方。server_process 用进程实现，server_thread 用线程实现。client.py 中计算从发送第一个数据到接收完最后一个数据的时间差。

即， $F(x) = x^4$

三、源代码

1. client.py

```
1. import socket
2. import time
3.
4.
5. time1 = time.time()
6. for i in range(1,100):
7.     s = socket.socket()
8.     host = socket.gethostname()
9.     port = 1234
10.    s.connect((host,port))
11.    s.send(bytes('20', encoding = 'utf8'))
12.    print(s.recv(1024).decode('utf-8'))
13.
14. time2 = time.time()
15. print(time2 - time1)
```

2. server_thread.py

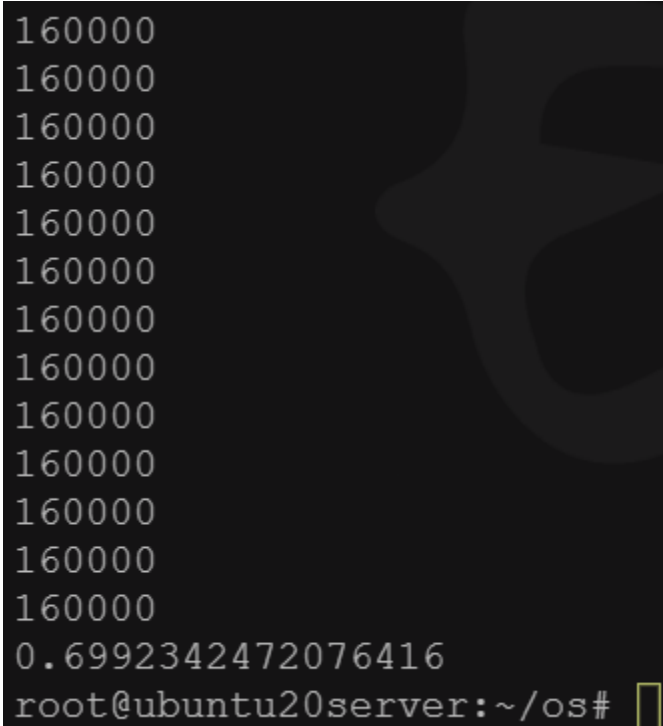
```
1. import socket
2. from threading import Thread
3.
4.
5. s = socket.socket()
6. host = socket.gethostname()
7. s.bind((host,1234))
8. s.listen(100)
9. print("Waiting for connection.....")
10.
11. def tcplink(sock,addr):
12.     print("Accept new connection from %s:%s" % addr)
13.     data = sock.recv(1024)
14.     i = int(data)
15.     sock.send(bytes(str(i*i*i*i), encoding = 'utf8'))
16.     sock.close()
17.     print("connection closed!")
18.
19. while True:
20.     sock, addr = s.accept()
21.     t = Thread(target = tcplink, args = (sock,addr))
22.     t.start()
```

3. server_process.py

```
1. import socket
2. from multiprocessing import *
3.
4.
5. s = socket.socket()
6. host = socket.gethostname()
7. s.bind((host,1234))
8. s.listen(100)
9. print("Waiting for connection.....")
10.
11. def tcplink(sock,addr):
12.     print("Accept new connection from %s:%s" % addr)
13.     data = sock.recv(1024)
14.     i = int(data)
15.     sock.send(bytes(str(i*i*i*i), encoding = 'utf8'))
16.     sock.close()
```

```
17.     print("connection closed!")
18.
19. while True:
20.     sock, addr = s.accept()
21.     t = Process(target = tcplink, args = (sock,addr))
22.     t.start()
```

四、实验结果

A terminal window with a dark background. It displays a series of 14 '160000' lines, followed by a decimal value '0.6992342472076416', and finally a command prompt 'root@ubuntu20server:~/os#' with a yellow cursor.

```
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
0.6992342472076416
root@ubuntu20server:~/os#
```

```
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
160000
0.8742377758026123
root@ubuntu20server:~/os#
```

从结果图中可以看到，多进程所用时间为 0.87424，而多线程所用时间为 0.69923，多进程比多线程慢了许多，因此可以得出结论，多进程在服务器中开销较大，使用多线程速度较快。