操作系统实验报告

冯子越
计算机 52
2150500067

使用 Python 语言编写一个 C/S 架构的程序，从客户端发送请求，服务器端接受请求后进行计算，返回结果。分别使用进程和线程实现。

程序由 3 个文件组成，分别是服务端（进程），服务端（线程）和客户端。它们之间的通信由 Python 的 Socket 模块发送 TCP 包来实现。线程使用 Threading 模块，进程使用 Multiprocessing 模块。代码如下：

~/Desktop/Process-and-Thread-speed/source/server_process.py - Sublime Text (UNREGISTERED)

client.py | server_process.py | server_thread.py

```python
from multiprocessing import Process,Queue
from socket import *
import time
import os

def compute(q):

    print('compute')
    x=q.get()
    x=x*x
    for i in range(10):
        x=x+i
        x=x-i
    q.put(x)
    print('compute done')

sock=socket(AF_INET,SOCK_STREAM)
sock.bind(('127.0.0.1',8007))
sock.listen(5)

while 1:
    print('waiting for connection')
    tcpClientSock,addr = sock.accept()
    print('connect from ',addr)
    data=tcpClientSock.recv(1024)
    data=int(data)
    print(data)

    start=time.time()
    q=Queue()
    q.put(data)
    p=Process(target=compute,args=(q,))
    p.start()
    p.join()
    data=q.get()
    end=time.time()

    s=str(data)
    t=str(end-start)
    s='result='+s+'and time is '+t
    tcpClientSock.send(s.encode('utf8'))
tcpClientSock.close()
sock.close()
```

Line 22, Column 18 | Tab Size: 4 | Python

~/Desktop/Process-and-Thread-speed/source/server_thread.py - Sublime Text (UNREGISTERED)

client.py | server_process.py | server_thread.py

```python
from socket import *
import threading
import time

x=0

def compute():
    global x
    x=x*x
    for i in range(10):
        x=x+i
        x=x-i
    print('compute done')

sock=socket(AF_INET,SOCK_STREAM)
sock.bind(('127.0.0.1',8007))
sock.listen(5)

while 1:
    print('waiting for connection')
    tcpClientSock,addr = sock.accept()
    print('connect from ',addr)
    data=tcpClientSock.recv(1024)
    data=int(data)
    print(data)

    start=time.time()
    global x
    x=data
    th=threading.Thread(target=compute)
    th.start()
    th.join()
    data=x
    end=time.time()

    s=str(data)
    t=str(end-start)
    s='result='+s+'and time is '+t
    tcpClientSock.send(s.encode('utf8'))
tcpClientSock.close()
sock.close()
```

Line 17, Column 15 Tab Size: 4 Python

~/Desktop/Process-and-Thread-master/source/client.py - Sublime Text (UNREGISTERED)

client.py | server_process.py | server_thread.py

```python
from socket import *

c=socket(AF_INET,SOCK_STREAM)
c.connect(('127.0.0.1',8009))


data='9'
c.send(data)
data=c.recv(1024)
print(data)

c.close()
```

客户端发送数字 9，服务器端将其平方后再进行一些运算，返回。下面测试进程和线程两种实现方式的负载差异。

首先分析速度。

线程实现的速度：

```
fzyue@fzyue: ~/Desktop/Process-and-Thread-speed/source
fzyue@fzyue:~/Desktop/Process-and-Thread-speed/source$ python server_process.py
waiting for connection
('connect from ', ('127.0.0.1', 42556))
9
compute
compute done
waiting for connection
```

```
fzyue@fzyue: ~/Desktop/Process-and-Thread-speed/source
fzyue@fzyue:~/Desktop/Process-and-Thread-speed/source$ python client.py
result=81and time is 0.000208854675293
fzyue@fzyue:~/Desktop/Process-and-Thread-speed/source$
```

使用了 0.0002 秒完成。

进程实现的速度：

```
fzyue@fzyue: ~/Desktop/Process-and-Thread-speed/source
fzyue@fzyue:~/Desktop/Process-and-Thread-speed/source$ python server_process.py
waiting for connection
('connect from ', ('127.0.0.1', 42556))
9
compute
compute done
waiting for connection
```

```
fzyue@fzyue: ~/Desktop/Process-and-Thread-speed/source
fzyue@fzyue:~/Desktop/Process-and-Thread-speed/source$ python client.py
result=81and time is 0.0100021362305
fzyue@fzyue:~/Desktop/Process-and-Thread-speed/source$
```

使用 0.01 秒完成。可见线程实现更快，因为计算量很小，所以主要反映了进程和线程在创建的时候的速度大约有两个数量级的差异。

接下来比较两种实现方式对内存占用的差异。将上面代码进行修改，每次创建两个进程或线程，并传入一个 7MB 左右的字符串。代码如下

进程：

~/Desktop/Process-and-Thread-master/source/server_process.py - Sublime Text (UNREGISTERED)

client.py | server_process.py | server_thread.py

```python
from multiprocessing import Process,Queue
from socket import *
import time
import os

def compute(q,st):
    x=q.get()
    st+='s'
    x=x*x
    print('process id :')
    print(os.getpid())
    time.sleep(40)
    q.put(x)

sock=socket(AF_INET,SOCK_STREAM)
sock.bind(('127.0.0.1',8009))
sock.listen(5)

while 1:
    print(os.getpid())
    print('waiting for connection')
    tcpClientSock,addr = sock.accept()
    print('connect from ',addr)
    data=tcpClientSock.recv(1024)
    data=int(data)
    st='sssssss'
    for i in range(100000):
        st=st+'sssssss'
    start=time.time()
    q=Queue()
    q.put(data)
    q.put(data)
    p1=Process(target=compute,args=(q,st,))
    p2=Process(target=compute,args=(q,st,))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
    data=q.get()
    end=time.time()

    s=str(data)
    t=str(end-start)
    s='result='+s+'and time is '+t
    tcpClientSock.send(s.encode('utf8'))
tcpClientSock.close()
sock.close()
```

Line 21, Column 36 | Tab Size: 4 | Python

线程：

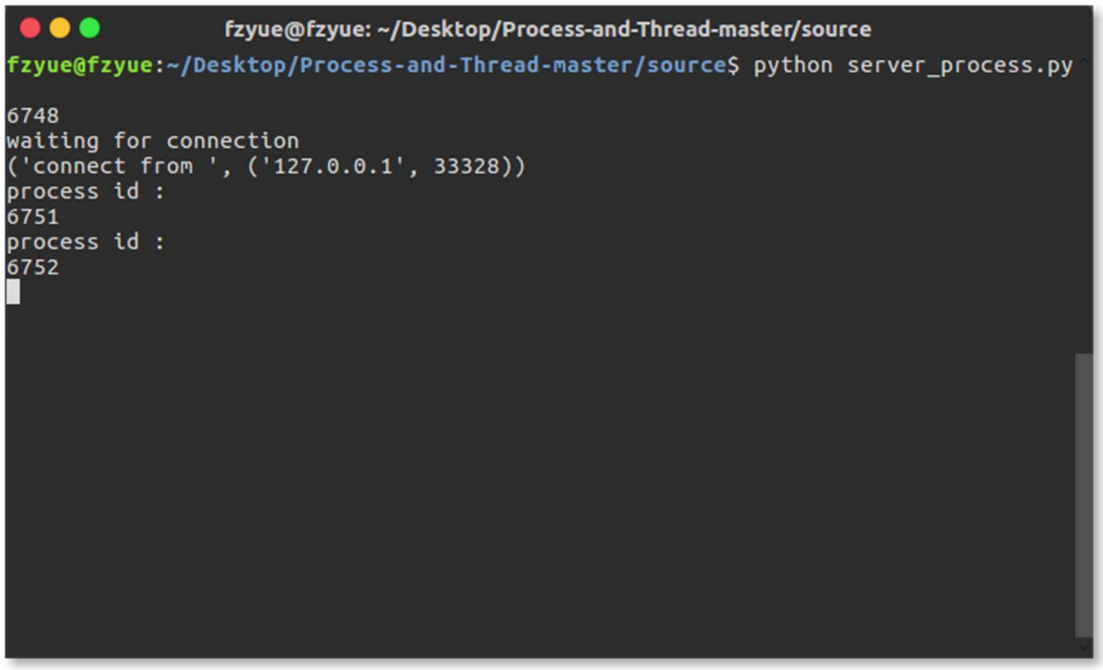
~/Desktop/Process-and-Thread-master/source/server_thread.py - Sublime Text (UNREGISTERED)

client.py | server_process.py | server_thread.py

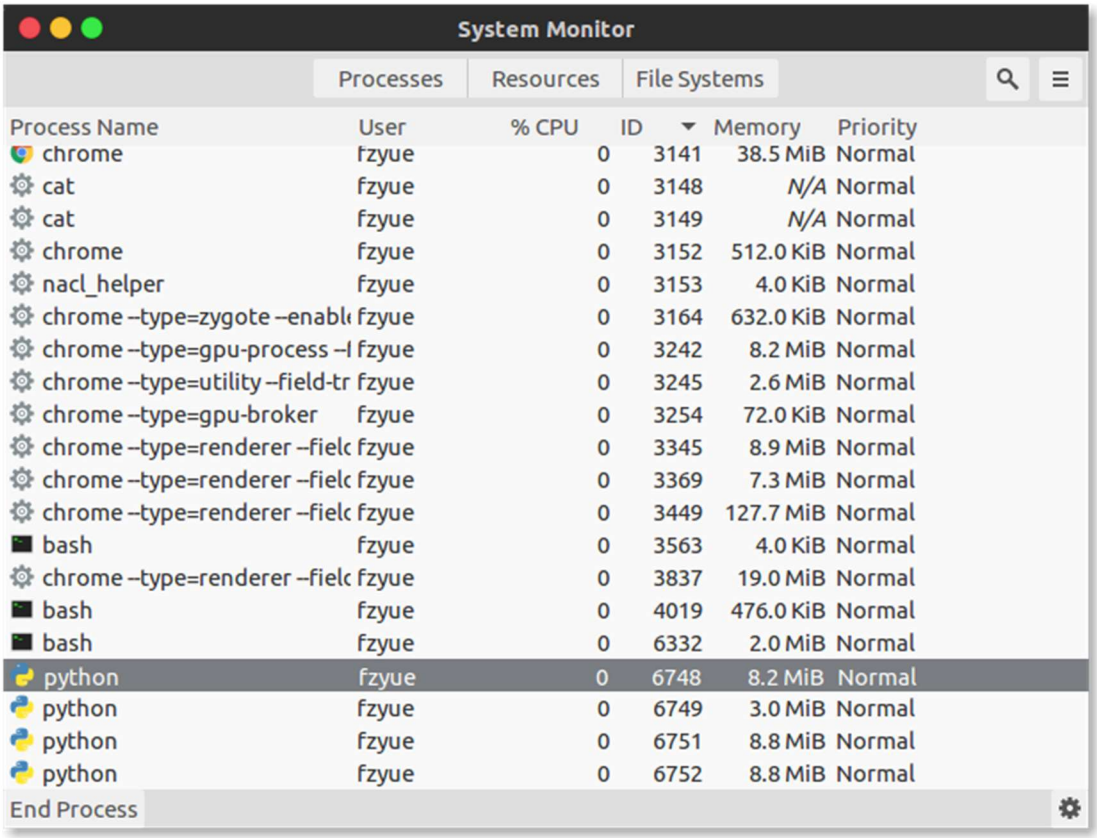
```python
from socket import *
import threading
import time
import os

x=0
lock=threading.Lock()

def compute(st):
    global x
    lock.acquire()
    xx=x
    lock.release()
    xx=xx*xx
    st+='sssssss'
    time.sleep(40)
    lock.acquire()
    x=xx
    lock.release()

sock=socket(AF_INET,SOCK_STREAM)
sock.bind(('127.0.0.1',8009))
sock.listen(5)

while 1:
    print('process id :')
    print(os.getpid())
    print('waiting for connection')
    tcpClientSock,addr = sock.accept()
    print('connect from ',addr)
    data=tcpClientSock.recv(1024)
    data=int(data)
    st='sssssss'
    for i in range(100000):
        st=st+'sssssss'
    start=time.time()
    global x
    x=data
    th1=threading.Thread(target=compute,args=(st,))
    th2=threading.Thread(target=compute,args=(st,))
    th1.start()
    th2.start()
    th1.join()
    th2.join()
    data=x
    end=time.time()

    s=str(data)
    t=str(end-start)
    s='result='+s+'and time is '+t
    tcpClientSock.send(s.encode('utf8'))
tcpClientSock.close()
sock.close()
```

Line 18, Column 9 | Tab Size: 4 | Python

实际运行，观察内存占用。
进程实现：



可以看到新建的两个进程的 id 分别是 6751 和 6752.



可以看到 6751 和 6752 两个进程各占用了 8.8MB 内存空间。

线程实现：

```
fzyue@fzyue: ~/Desktop/Process-and-Thread-mem/source
fzyue@fzyue:~/Desktop/Process-and-Thread-mem/source$ python server_thread.py
server_thread.py:37: SyntaxWarning: name 'x' is assigned to before global declar
ation
  global x
process id :
2180
waiting for connection
('connect from ', ('127.0.0.1', 51888))
```

只有一个进程，id 为 2180，两个线程都在这一个进程下。

System Monitor

Processes | Resources | File Systems

Process Name	User	% CPU	ID	Memory	Priority
gvfs-afc-volume-monitor	fzyue	0	1931	3.0 MiB	Normal
gvfsd-trash	fzyue	0	1941	732.0 KiB	Normal
gvfsd-metadata	fzyue	0	1959	544.0 KiB	Normal
gvfsd-network	fzyue	0	1985	2.7 MiB	Normal
gvfsd-dnssd	fzyue	0	2004	764.0 KiB	Normal
gnome-terminal-server	fzyue	0	2010	9.1 MiB	Normal
bash	fzyue	0	2016	2.0 MiB	Normal
bash	fzyue	0	2026	2.0 MiB	Normal
zeitgeist-datahub	fzyue	0	2037	4.4 MiB	Normal
sh	fzyue	0	2044	80.0 KiB	Normal
zeitgeist-daemon	fzyue	0	2048	1.3 MiB	Normal
zeitgeist-fts	fzyue	0	2055	3.9 MiB	Normal
unity-scope-home	fzyue	0	2095	3.9 MiB	Normal
unity-scope-loader	fzyue	0	2105	7.6 MiB	Normal
unity-files-daemon	fzyue	0	2107	4.3 MiB	Normal
gnome-system-monitor	fzyue	1	2140	16.3 MiB	Normal
python	fzyue	0	2180	7.8 MiB	Normal
update-notifier	fzyue	0	2212	3.6 MiB	Normal
python	fzyue	0	2265	3.0 MiB	Normal
deja-dup-monitor	fzyue	0	2298	2.5 MiB	Normal

End Process

可以看到它总共占用了 7.8MB 内存。可见线程实现时不会开辟新的内存空间，内存占用少很多。

综上所述，线程实现比进程实现的速度更快，内存更省。

上述代码已上传到我的 GitHub 主页：github.com/fengziyue