关于作者
廖雪峰 V 北京 朝阳区
+ 加关注

多进程

阅读: 851167

要让Python程序实现多进程（multiprocessing），我们先了解操作系统的相关知识。

Unix/Linux操作系统提供了一个`fork()`系统调用，它非常特殊。普通的函数调用，调用一次，返回一次，但是`fork()`调用一次，返回两次，因为操作系统自动把当前进程（称为父进程）复制了一份（称为子进程），然后，分别在父进程和子进程内返回。

子进程永远返回`0`，而父进程返回子进程的ID。这样做的理由是，一个父进程可以fork出很多子进程，所以，父进程要记下每个子进程的ID，而子进程只需要调用`getppid()`就可以拿到父进程的ID。

Python的`os`模块封装了常见的系统调用，其中就包括`fork`，可以在Python程序中轻松创建子进程：

```
import os

print('Process (%s) start...' % os.getpid())
# Only works on Unix/Linux/Mac:
pid = os.fork()
if pid == 0:
    print('I am child process (%s) and my parent is %s.' % (os.getpid(), os.getppid()))
else:
    print('I (%s) just created a child process (%s).' % (os.getpid(), pid))
```

运行结果如下：

```
Process (876) start...
I (876) just created a child process (877).
I am child process (877) and my parent is 876.
```

由于Windows没有`fork`调用，上面的代码在Windows上无法运行。由于Mac系统是基于BSD（Unix的一种）内核，所以，在Mac下运行是没有问题的，推荐大家用Mac学Python！

有了`fork`调用，一个进程在接到新任务时就可以复制出一个子进程来处理新任务，常见的Apache服务器就是由进程监听端口，每当有新的http请求时，就fork出子进程来处理新的http请求。

multiprocessing

如果你打算编写多进程的服务程序，Unix/Linux无疑是正确的选择。由于Windows没有`fork`调用，难道在Windows上无法用Python编写多进程的程序？

由于Python是跨平台的，自然也应该提供一个跨平台的多进程支持。`multiprocessing`模块就是跨平台版本的多进程模块。

`multiprocessing`模块提供了一个`Process`类来代表一个进程对象，下面的例子演示了启动一个子进程并等待其结束：

```
from multiprocessing import Process
import os

# 子进程要执行的代码
def run_proc(name):
    print('Run child process %s (%s)...' % (name, os.getpid()))

if __name__=='__main__':
    print('Parent process %s.' % os.getpid())
    p = Process(target=run_proc, args=('test',))
    print('Child process will start.')
    p.start()
    p.join()
    print('Child process end.')
```

执行结果如下：

```
Parent process 928.
Process will start.
Run child process test (929)...
Process end.
```

创建子进程时，只需要传入一个执行函数和函数的参数，创建一个`Process`实例，用`start()`方法启动，这样创建进程比`fork()`还要简单。

`join()`方法可以等待子进程结束后再继续往下运行，通常用于进程间的同步。

Pool

如果要启动大量的子进程，可以用进程池的方式批量创建子进程：

```
from multiprocessing import Pool
import os, time, random

def long_time_task(name):
    print('Run task %s (%s)...' % (name, os.getpid()))
    start = time.time()
    time.sleep(random.random() * 3)
    end = time.time()
    print('Task %s runs %0.2f seconds.' % (name, (end - start)))

if __name__=='__main__':
    print('Parent process %s.' % os.getpid())
    p = Pool(5)
    for i in range(5):
        p.apply_async(long_time_task, args=(i,))
    print('Waiting for all subprocesses done...')
    p.close()
    p.join()
    print('All subprocesses done.')
```

执行结果如下：

```
Parent process 669.
Waiting for all subprocesses done...
Run task 0 (671)...
Run task 1 (672)...
Run task 2 (673)...
Run task 3 (674)...
Task 2 runs 0.14 seconds.
Run task 4 (673)...
Task 1 runs 0.27 seconds.
Task 3 runs 0.86 seconds.
Task 0 runs 1.41 seconds.
Task 4 runs 2.91 seconds.
All subprocesses done.
```

代码解读：

对`Pool`对象调用`join()`方法会等待所有子进程执行完毕，调用`join()`之前必须先调用`close()`，调用`close()`之后就不能继续添加新的`Process`了。

请注意输出的结果，task `0`，`1`，`2`，`3`是立刻执行的，而task `4`要等待前面某个task完成后才执行，这是因为`Pool`的默认大小在我的电脑上是4，因此，最多同时执行4个进程。这是`Pool`有意设计的限制，并不是操作系统的限制。如果改成：

```
p = Pool(5)
```

就可以同时跑5个进程。

由于`Pool`的默认大小是CPU的核数，如果你不幸拥有8核CPU，你要提交至少9个子进程才能看到上面的等待效果。

子进程

很多时候，子进程并不是自身，而是一个外部进程。我们创建了子进程后，还需要控制子进程的输入和输出。

`subprocess`模块可以让我们非常方便地启动一个子进程，然后控制其输入和输出。

下面的例子演示了如何在Python代码中运行命令`nslookup www.python.org`，这和命令行直接运行的效果是一样的：

```
import subprocess

print('$ nslookup www.python.org')
r = subprocess.call(['nslookup', 'www.python.org'])
print('Exit code:', r)
```

运行结果：

```
$ nslookup www.python.org
Server:         192.168.19.4
Address:        192.168.19.4#53

Non-authoritative answer:
www.python.org  canonical name = python.map.fastly.net.
Name:   python.map.fastly.net
Address: 199.27.79.223

Exit code: 0
```

如果子进程还需要输入，则可以通过`communicate()`方法输入：

```
import subprocess

print('$ nslookup')
p = subprocess.Popen(['nslookup'], stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
output, err = p.communicate(b'set q=mx\npython.org\nexit\n')
print(output.decode('utf-8'))
print('Exit code:', p.returncode)
```

上面的代码相当于在命令行执行命令`nslookup`，然后手动输入：

```
set q=mx
python.org
exit
```

运行结果如下：

```
$ nslookup
Server:         192.168.19.4
Address:        192.168.19.4#53

Non-authoritative answer:
python.org      mail exchanger = 50 mail.python.org

Authoritative answers can be found from:
mail.python.org internet address = 82.94.164.166
mail.python.org has AAAA address 2001:888:2000:d::a6


Exit code: 0
```

进程间通信

`Process`之间肯定是需要通信的，操作系统提供了很多机制来实现进程间的通信。Python的`multiprocessing`模块包装了底层的机制，提供了`Queue`、`Pipes`等多种方式来交换数据。

我们以`Queue`为例，在父进程中创建两个子进程，一个往`Queue`里写数据，一个从`Queue`里读数据：

```
from multiprocessing import Process, Queue
import os, time, random

# 写数据进程执行的代码:
def write(q):
    print('Process to write: %s' % os.getpid())
    for value in ['A', 'B', 'C']:
        print('Put %s to queue...' % value)
        q.put(value)
        time.sleep(random.random())

# 读数据进程执行的代码:
def read(q):
    print('Process to read: %s' % os.getpid())
    while True:
        value = q.get(True)
        print('Get %s from queue.' % value)

if __name__=='__main__':
    # 父进程创建Queue，并传给各个子进程：
    q = Queue()
    pw = Process(target=write, args=(q,))
    pr = Process(target=read, args=(q,))
    # 启动子进程pw，写入：
    pw.start()
    # 启动子进程pr，读取：
    pr.start()
    # 等待pw结束：
    pw.join()
    # pr进程里是死循环，无法等待其结束，只能强行终止：
    pr.terminate()
```

运行结果如下：

```
Process to write: 50563
Put A to queue...
Process to read: 50564
Get A from queue.
Put B to queue...
Get B from queue.
Put C to queue...
Get C from queue.
```

在Unix/Linux下，`multiprocessing`模块封装了`fork()`调用，使我们不需要关注`fork()`的细节。由于Windows没有`fork`调用，因此，`multiprocessing`需要“模拟”出`fork`的效果，父进程所有Python对象都必须通过pickle序列化再传到子进程去，所有，如果`multiprocessing`在Windows下调用失败了，要先考虑是不是pickle失败了。

小结

在Unix/Linux下，可以使用`fork()`调用实现多进程。

要实现跨平台的多进程，可以使用`multiprocessing`模块。

进程间通信是通过`Queue`、`Pipes`等实现的。

参考源码

do_fork.py

multi_processing.py

pooled_processing.py

do_subprocess.py

do_queue.py

读后有收获可以请作者喝咖啡，读后有疑问请加群讨论：

还可以分享给朋友：

分享到微博

上一页 下一页

评论

进程池的数据通信
liyuhuiadmin created at May 24, 2019 1:01 PM, Last updated at May 24, 2019 1:01 PM

使用Pool创建多进程，想要数据通信时，需要使用multiprocessing.Manager().Queue()，multiprocessing下的Queue是不行的

全部评论 回复

multiprocessing没有pool怎么回事呢？
Lilinxa created at April 26, 2019 7:23 PM, Last updated at May 23, 2019 10:22 AM

我重新了一遍上文中的 multiprocessing import Pool的那一段，但是在python交互模式运行的时候提示：AttributeError：Module 'multiprocessing' has no attribute Pool。