多进程

1452次阅读

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：

```
# multiprocessing.py
import os

print 'Process (%s) start...' % os.getpid()
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服务器就是由父进程监听端口，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 'Process will start.'
    p.start()
    p.join()
    print '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。

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()
    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 1.91 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。

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):
    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):
    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:

```
Put A to queue...
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。

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。

Links