Python语言多进程与多线程设计探究

肖明魁 / 江苏经贸职业技术学院

摘 要：作为一种新型面向对象，解释型程序设计语言，python在近年来被广泛运用于系统管理和web开发，其特点是语法简洁，清晰，具有丰富强大的类库，且易于理解和维护。随着计算机硬件技术的迅速发展，多核CPU及并行计算技术的大规模应用，在程序设计中，如何处理好进程和线程优化管理问题，是提高程序运行效率的关键。

关键词：面向对象；Python；进程和线程；程序设计

1 进程与线程简介

进程是指一个具有独立功能的程序关于某个数据集合的一次运行活动，相对于程序而言，它是一个动态的概念，是指“执行中的程序”。而线程则是进程中的某个单一顺序控制流，是程序执行和CPU调度的基本单位，进程和线程的区别在于：

（1）一个进程中至少包含一个线程，线程是进程的子集；

（2）进程间相互独立，系统为每个进程分配单独数据空间，而同一进程中的线程间数据共享。

在单核CPU时代，操作系统为完成多任务程序处理，通常采用时间片轮转调度算法，即为每个进程或线程分配一定长度的时间段，将所要执行任务按优先级排成队列，依次执行，并通过反复切换进程块，完成虚拟多任务操作。而多核CPU的出现，可以使得程序进程在多个硬件内核中并行处理，从而让系统真正实现了多任务管理。在Python开发语言可以同时支持多进程，多线程程序设计，通常采用三种设计方案：

（1）多进程设计。

（2）多线程设计。

（3）多里程+多线程设计。下面逐一介绍其原理。

2 多进程设计

在一般程序设计当中，通常只有一个进程，即父进程，为了实现多任务程序开发，就必须增加进程块，即子进程。增加进程块一般采用调用Python类库模块的方式实现，而对于不同的操作系统，所使用的调用方法也有所不同，在Unix/Linux系统中，提供了一个fork()函数的系统调用，用于创建子进程，代码如下：

```
import os
print'Process(%s)start...'%os.getpid()
pid=os.fork()
if pid==0:
print（'this is a child process(%s)\n this is a parent process is (%s).'%(os.getpid(),os.getppid())）
else:
print（'this (%s)have created a child process (%s).'%(os.getpid(),pid)）
```

该段程序中，利用os.fork()创建子进程pid,而后通过os.getpid()获取当前进程号，os.getppid()获取父进程号，运行结果如下：

```
Process(1123)start...
this(1123) have created a child process(1124).
this is child process(1124)
this is a parent progerss(1123).
```

而对于windows系统下运行的Python而言，由于没有fork()调用，因此，通常会利用multiprocessing模块完成任务，作为一个跨平台版本的多进程模块，multiprocessing可以提供一个process类，并以此创建子进程，代码如下：

```
from multiprocessing import Process
import os
def proc(name):
print('this is a child process %s(%s)...'%(name,os.getpid()))
if__name__=='__main__':
print('this is a Parent process %s.'% os.getpid())
p=Process(target=proc,args=('test',))
print('Process will start.')
p.start()
p.join()
print('Process end.')
```

该段代码中，利用process创建一个实例p,通过函数调用和参数传递，并用start()方法启动，建立子进程，而后用join()方法，在函数调用结束后继续运行下一步运行，程序运行结果如下：

```
this is a Parent process 1228.
Process will start.
this is a child process test(1229)...
Process end.
```

3 多线程设计

Python不仅支持多进程，同时也支持多线程设计，Python标准库提供两个模块：thread和threading，其中，thread是低级模块，而threading是高级模块，是对thread进行二次封装，使用api接口函数处理线程。绝大多数情况下，我们只需要使用threading这个高级模块，创建子线程

中图分类号：TP393.01

的方法类似于多进程设计，先是创建thread实例，而后通过函数调用参数传递，并用start（）启动,代码如下：

```
import time,threading
def loop():
print'%s start...'%threading.current_thread().name
for n in range(5):
print'%s…%s'%(threading.current_thread().name,n)
time.sleep(1)
print'%s ended.'%threading.current_thread().name
print'%s start…'%threading.current_thread().name
t=threading.Thread(target=loop,name='Sub_Thread')
t.start()
t.join()
print'%s ended.'%threading.current_thread().name
```

任何进程在运行时，都会启动一个默认线程，称为主线程，主线程可以启动多个子线程，在本段代码当中，先是利用threading.current.thread().name获取当前线程，即主线程，而后通过threading.thread()创建一个实例t，在函数体内，利用循环语句，一共创建五个子线程。

由于多线程设计需要共享数据，因此，在进程运行过程中，变量如果被其中某个线程修改，会影响该进程中所有线程对该变量值的读取，为防止这一状况发生，Python利用threading.lock()方法实现对线程的锁定，代码如下:

```
import time,threading
a=0
lock=threading.Lock()
def change_it(n):
global a
a=a+n
a=a-n
def run_thread(n):
for i in range(100000):
lock.acquire()
try:
change_it(n)
finally:
lock.release()
TR1=threading.Thread(target=run_thread,args=(5,))
TR2=threading.Thread(target=run_thread,args=(8,))
TR1.start()
TR2.start()
TR1.join()
TR2.join()
print a
```

在本段代码中，创建了两个线程TR1，TR2，为了防止两个线程在运行过程中，变量值改变，影响运行结果，在代码中加入了threading.Lock()，并且通过acquire()方法，锁定当前运行线程，同时让另一线程处于等待状态，直到当前线程运行结束，并使用release()方法释放后，才可以继续执行下一线程。

4 多进程与多线程设计的比较

设计多任务处理程序，通常会采用Master-worker模式，即并行设计，核心思想是将原来串行的逻辑并行化，并将逻辑拆分成很多独立模块并行执行，Master负责分配任务，Worker则负责执行任务，主进程或多线程为master，其它进程或线程为worker，多进程和多线程设计是最为常用的两种方式，两者各有利蔽：

多进程设计中，数据共享复杂，需要用IPC；数据是分开的，同步较简单；而对于多线程设计，数据共享简单，由此却增加了同步的难度。

多进程代码设计和调试较方便，但是创建和释放速度较慢，cpu及内存占用率高；多线程系统占用资料少，运行效率高，但是设计和调试相对复杂。

多进程设计最大优势在于高可靠性，进程间相互独立，一个进程出现问题不会对其它进程运行造成影响；而多线程因为数据共享特性，当一个线程错误，会导致整个进程运行失败。

因此，在程序设计当中，设计者应当根据具体任务使用多进程+多线程的方法，对于计算量大，消耗资源较高的任务采用多线程设计，而对于诸如I/O输出，web应用，多机分布式计算等稳定性要求较高的任务多采用多进程设计。

参考文献：

[1]Wesley J.Chen(美).宋吉广,译.Python核心编程(第二版)[M].北京:人民邮电出版社,2008.

[2]Swaroop,C.H.沈吉元,译.简明Python教程[M/OL],2005.

[3]Noah Gift.使用Python实现多进程[M],2009.

作者简介：肖明魁（1979.04-），男、初级职称、大专、研究方向：计算机。

作者单位：江苏经贸职业技术学院、南京 210000