**Python多进程管理包multiprocessing包&pipe&queue的使用：**

<http://www.cnblogs.com/vamei/archive/2012/10/12/2721484.html>

**多线程包与多进程包的相似性：**

**1.创建线程：**threading.Thread（）对象 创建进程：multiprocessing.Process（）对象

2.创建好的线程/进程**都可以运行在Python程序内部编写的函数**。

3.创建的Process对象与Thread**对象的用法相同**：都有start(), run(), join()的方法

4.都有Lock/Event/Semaphore/Condition类，可以通过参数传递给各个线程/进程)，用以**同步**线程/进程。

所以，multiprocessing的很大一部份与threading使用同一套API。

**不同点：**

**1.在UNIX平台上，当多进程中某个进程终结之后，有必要对每个进程（Process）对象调用join()方法** (实际上等同于wait)，否则进程成为[僵尸进程](http://www.cnblogs.com/vamei/archive/2012/09/20/2694466.html)(Zombie)。**对于多线程来说，由于只有一个进程，所以不存在此必要性。**

2.multiprocessing提供了threading包中没有的IPC(比如Pipe和Queue)，效率上更高。**应优先考虑Pipe和Queue**，避免使用Lock/Event/Semaphore/Condition等同步方式 (因为它们占据的不是用户进程的资源)。

3.在多线程中，可以比较容易地共享资源，比如[使用全局变量或者传递参数](http://www.cnblogs.com/vamei/archive/2012/10/11/2720042.html)。在多进程情况下，由于**每个进程有自己独立的内存空间，应该避免共享资源**。此时我们可以通过[共享内存](http://www.cnblogs.com/vamei/archive/2012/10/10/2715398.html)和Manager的方法来共享资源。但这样做提高了程序的复杂度，并因为同步的需要而降低了程序的效率。

Process.PID中保存有PID，如果进程还没有start()，则PID为None。

**多线程和多进程的区别（**各个线程和进程都做一件事：打印PID**）：**

**#!/usr/bin/python**

# Similarity and difference of multi thread vs. multi process

import os

import threading

import multiprocessing

# worker function

def worker(sign, lock):

lock.acquire()

print(sign, os.getpid())

lock.release()

# Main

print('Main:',os.getpid())

# 多线程

record = []

lock = threading.Lock()

for i in range(5):

thread = threading.Thread(target=worker,args=('thread',lock))

thread.start()

record.append(thread)

for thread in record:

thread.join()

# 多进程

record = []

lock = multiprocessing.Lock()

for i in range(5):

process = multiprocessing.Process(target=worker,args=('process',lock))

process.start()

record.append(process)

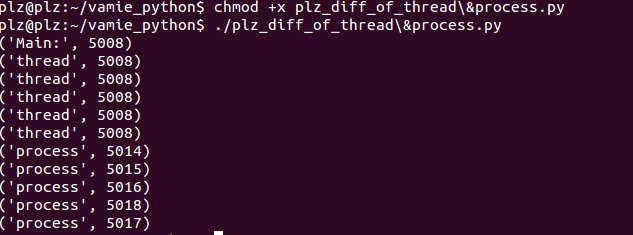
for process in record:

process.join()

刚开始运行的时候，一运行就卡死电脑，后来在

<http://my.oschina.net/liudiwu/blog/88573>，是因为码存放的目录里，多出来一个字节码的pyc文件，导致程序发生递归，结果就是不断启用python解释执行pyc文件而卡死的，与代码并没有关系。我看了一下我的代码存放目录下并没有pyc文件。。这时候突然想到我的代码前面忘记加python的绝对路径了，加上**#!/usr/bin/python**之后，再次运行就正常了。

可以看出多线程中各个线程的id与主程序的id相同，多进程中各个进程的id与主程序不同



**值得注意的是，多线程并不限制你一次只执行一个，如果你的线程之间不共享变量的话，是用不到线程锁的，虽然这么说，但是其实即使不用lock程序还是一次只执行一个线程。如果程序内部有阻塞现象（比如下载之类），使用线程很不错，但是如果没有的话，用不用线程的消耗差别不大（甚至因为要支持线程而减慢）。不使用lock其实是将lock交给了python负责，它会根据阻塞与否协调线程的执行。**

**Queue的使用**

#!/usr/bin/python

import os

import multiprocessing

import time

#==================

# input worker

def inputQ(queue):

info = str(os.getpid()) + '(put):' + str(time.time())

queue.put(info)

#将input process id （put）： 此时的时间戳 信息放入队列

# output worker

def outputQ(queue,lock):

info = queue.get() #从队列中获取信息

lock.acquire()

print (str(os.getpid()) + '(get):' + info)

#打印信息：output process id （get）： input process id （put）： 时间戳

lock.release()

#===================

# Main

record1 = [] # store input processes

record2 = [] # store output processes

lock = multiprocessing.Lock() #多进程共享一个stdout，同时向stdout输出的话，输出结果会混合在一起。用锁，可以让一个进程输出之后，再由另一个进程输出。

queue = multiprocessing.Queue(3)

# input processes（几乎同时产生，所以时间戳也几乎完全相同）

for i in range(10):

process = multiprocessing.Process(target=inputQ,args=(queue,))

process.start()

record1.append(process)

# output processes

for i in range(10):

process = multiprocessing.Process(target=outputQ,args=(queue,lock))

process.start()

record2.append(process)

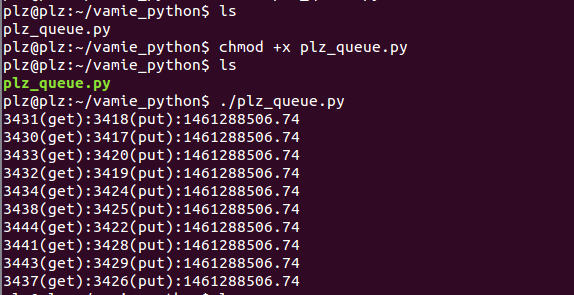
for p in record1:

p.join()

queue.close() # No more object will come, close the queue

for p in record2:

p.join()



Queue与Pipe相类似，都是先进先出的结构。但Queue允许多个进程放入，多个进程从队列取出对象。**Queue使用mutiprocessing.Queue(maxsize)创建**，maxsize表示队列中可以存放对象的最大数量。

**Pipe的使用**

#!/usr/bin/python

# Multiprocessing with Pipe

import multiprocessing as mul

def proc1(pipe):

pipe.send('hello')

print('proc1 rec:',pipe.recv())

def proc2(pipe):

print('proc2 rec:',pipe.recv())

pipe.send('hello, too')

# Build a pipe

pipe = mul.Pipe()

# Pass an end of the pipe to process 1

p1 = mul.Process(target=proc1, args=(pipe[0],))

#p1发送消息“hello”，并等待接受信息

# Pass the other end of the pipe to process 2

p2 = mul.Process(target=proc2, args=(pipe[1],))

#p2打印从p1接受的信息，并发送消息给p1

p1.start()

p2.start()

p1.join()

p2.join()选区_098

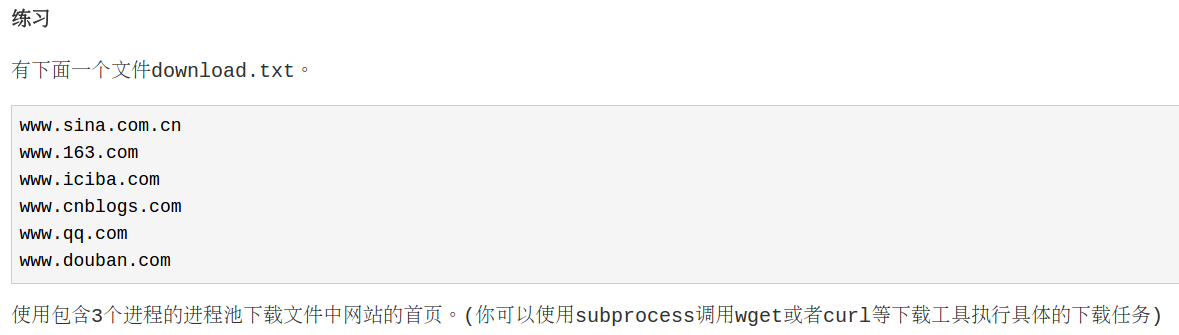
Pipe对象建立的时候，返回一个含有两个元素的表，每个元素代表Pipe的一端(Connection对象)。我们**对Pipe的某一端调用send()方法来传送对象，在另一端使用recv()来接收。**

**2016.4.25**

# **[Python multiprocessing包)](http://www.cnblogs.com/vamei/archive/2012/10/13/2722254.html)高级使用～**

<http://www.cnblogs.com/vamei/archive/2012/10/13/2722254.html>

进程池 (Process Pool)可以创建多个进程。



#!/usr/bin/python

import multiprocessing as mul

import subprocess

def download(x):

out = subprocess.call("wget x",shell=True)

**#参考out = subprocess.call("ls -l", shell=True)**

**http://www.cnblogs.com/vamei/archive/2012/09/23/2698014.html**

add1='www.baidu.com'

add2='www.163.com'

add3='www.iciba.com'

add4='www.cnblogs.com'

add5='www.qq.com'

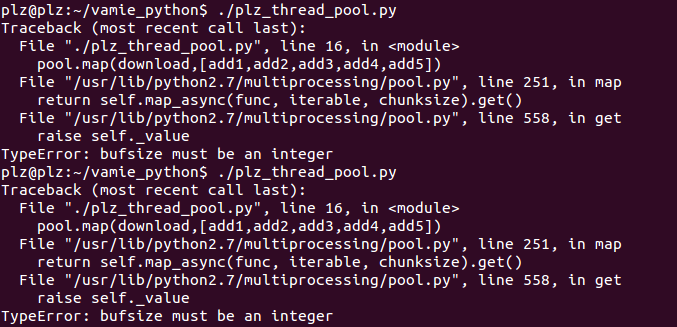
pool=mul.Pool(3)

pool.map(download,[add1,add2,add3,add4,add5])



将out = subprocess.call("wget x",shell=True) 改为：

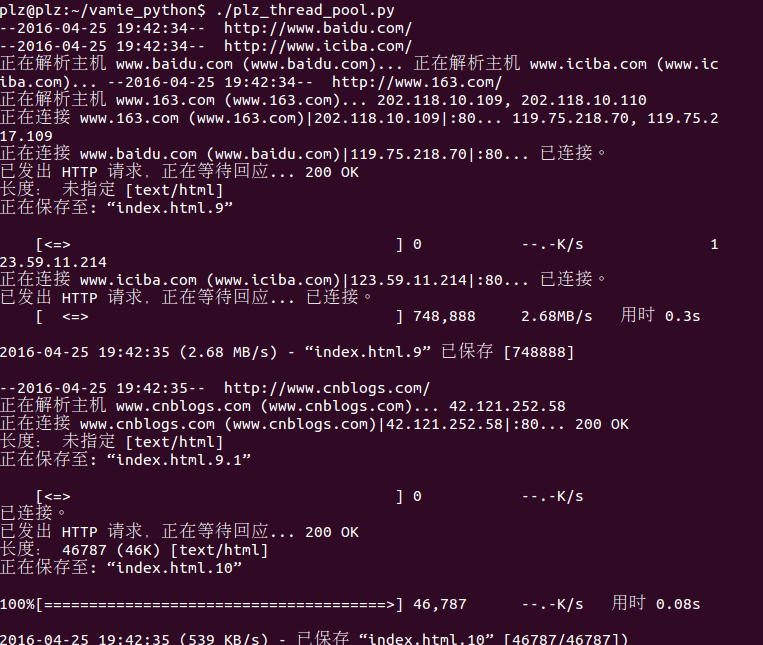
out = subprocess.call("wget",x,shell=True)



将out = subprocess.call("wget",x,shell=True)改为：

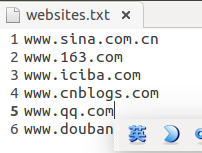
out = subprocess.call(["wget",x])

#参考rc = subprocess.call(["ls","-l"])



之后又将网址的表达方式改为“”，发现仍然可以正常运行～

然后将所有网址保存为一个文件websites.txt



采取按行读取文件内容的方式读入网址：

websites=[] #初始化

for line in open('websites.txt'):

websites.append(line.strip())

print websites #输出网址

此时整个程序变为：

import multiprocessing as mul

import subprocess

def download(x):

out = subprocess.call(['wget',x])

websites=[]

for line in open('websites.txt'):

websites.append(line.strip())

print websites

pool = mul.Pool(3) #创建了一个容许3个进程的进程池

pool.map(download,websites) #每个进程都执行f()函数，利用map()方法将download()函数作用到表的每个元素上。

**如果进程运行结束后，还有需要处理的元素，那么的进程会被用于重新运行f()函数**。

Pool的常用方法：

上面的map()方法

apply\_async(func,args)  从进程池中取出一个进程执行func，args为func的参数。它将返回一个AsyncResult的对象，你可以对该对象调用get()方法以获得结果。

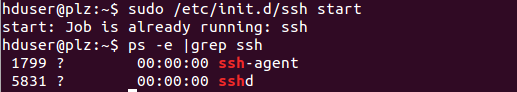
close()  进程池不再创建新的进程

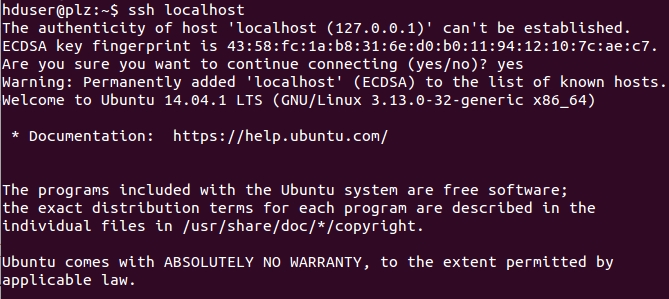
join()   wait进程池中的全部进程。必须对Pool先调用close()方法才能join。

**如何使用python编写一个mapreduce项目**

1. 安装java

启动服务。启动后，可以通过命令查看服务是否正确启动





**安装hadoop**

到https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/stable/下载hadoop

