Python的Multiprocessing模块。 (基于进程的模块， 可以充分发挥出多核处理器的功用, 同时也是弥补使用多线程threading（以前的multithreading模块的劣势)

1. Process类

p = Process (target=f, args=('bob',)) 🡪 target指明此进程所要执行的函数，args 表明输入的参数

p.start() 🡪 调用对象的start方法开始执行

p.join()

🡪在进程中，以os.getppid()获取父进程id, 以getpid()获取自己进程的id

2. 进程之间的交流。Python给与了两种方式，提供进程之间的交流。

队列(Queue)

Queue的功能是将每个核或线程的运算结果放在队里中， 等到每个线程或核运行完毕后再从队列中取出结果。我们可以在一个进程中使用Queue对象的put() 方法像队列中放入数据，然后可以在任何进程中通过Queue对象的get()方法取出队尾的一个数据。

管道(Pipes)

3. 效率上，多进程比多线程快许多，多线程和普通执行过程相比差别不大，甚至还会比普通执行过程更长。

4.

Multiprocessing模块中的Pool类可以提供指定数量的并发进程，当有新的请求提交到池中时，如果池子的限额没有满，则会加入池中，否则请求等待。

---调用Pool实例的join方法，将会等待池中所有的子进程完毕。

---调用join方之前必须调用close方法

---当close以后，池子不能再加入新的请求。

---Pool的apply\_async方法是向池中加入异步请求，池中的请求可以并发执行。

---Pool的apply方法是向池中加入同步请求，池中的请求完成后才可以加入下一个请求。该方法只能允许一个进程进入池子，在一个进程结束之后，另外一个进程才可以进入池子。

---Pool的apply\_async方法执行后将返回异步结果对象，将对象get方法将获得原方法的结果。

---Pool对象的map方法集合了基础函数map的功能以及Pool的apply\_async()函数的功能。

基础Python函数map的功能是：

第一个参数 function 以参数序列中的每一个元素作为参数传入 function 函数，并直接function函数，map函数的返回为每次 function 函数执行的返回值所形成的新列表。

map(function, iterable, ...)

基于同样原理Pool中的map函数的功能是根据给定的iterable，每次将一个元素作为job的参数（所以map函数只允许job接受一个参数），放入pool池中并且异步执行。将各进程返回的结果放入队列中，作为map函数的返回值。

map(job, iterable, ...)

例如：

def multicore():

pool = mp.Pool()

res = pool.map(job, range(10)) 🡪直接用pool.map函数发起任务并且获取结果，结果为map函数将各进程执行job函数的的返回值放入队列中，将队列作为map的返回结果

print(res)

res = pool.apply\_async(job, (2,)) 🡪也可以不用map函数获取结果，使用apply\_async,然后从返回对象中get()方法获取这个进程的返回结果。

# 用get获得结果

print(res.get())

5.

from multiprocessing import Pool 🡪 这个pool是基于进程的Pool，一般使用这个

from multiprocessing.pool import ThreadPool🡪 这个pool是基于线程的Pool

6.

def generate\_data(data):  
 for i in range(1, 4):  
 data.append(i)  
 return data  
  
  
data = []  
pool = Pool(processes=cpu\_count())  
for i in range(3):  
 async\_result = pool.apply\_async(generate\_data, args=(data,))  
pool.close()  
pool.join()  
print("All Processes End")  
print("Data is:" + str(data))  
  
data = []  
pool = ThreadPool(processes=cpu\_count())  
for i in range(3):  
 async\_result = pool.apply\_async(generate\_data, args=(data,))  
pool.close()  
pool.join()  
print("All threading End")  
print("Data is:" + str(data))

从以上例子我们可以看出，如果我们使用线程Threading那么最后data中会有数据，此数据又线程中提供。如果我们使用多进程，那么一般情况下变量将依据进程分割，最后的data中没有任何数据。