进程与线程

进程：运行中的程序

线程：可由一个进程派生出多个线程，同一进程下的线程共享该进程的变量等资源。

在调度上的不同：

进程是由cpu调度的，当一个进程在cpu上运行了足够长的时间之后，cpu就会保存他运行所需要的一些变量到寄存器中，切换到下一个进程中。由此，在单核的cpu上制造出有多个程序在“同时运行”的假象（称为并行）。

而线程则不是由cpu调度的，因为线程都是由程序员在程序中实现的，因此一般认为没有必要在cpu层面对线程也进行调度，而是由程序员自己控制(在程序中控制)线程的运行时间。当然如果线程所在的进程用完了属于他自己的cpu时间时，也是会被调度下cpu的。

关于python GIL机制

在正常运行时GIL存在，此时python只有一个线程.

一旦python程序执行了有IO的操作如从磁盘读取文件，网络爬虫等，或者sleep了，python都会将GIL锁解开，此时其他线程可以尝试获取锁，线程获取锁之后即可运行。

io密集型任务中的多线程与多进程

首先对书上说的这个做一个实验。

每个实验做多次，去掉有明显阻塞的某些情况，取平均值出现频率最高的一组。

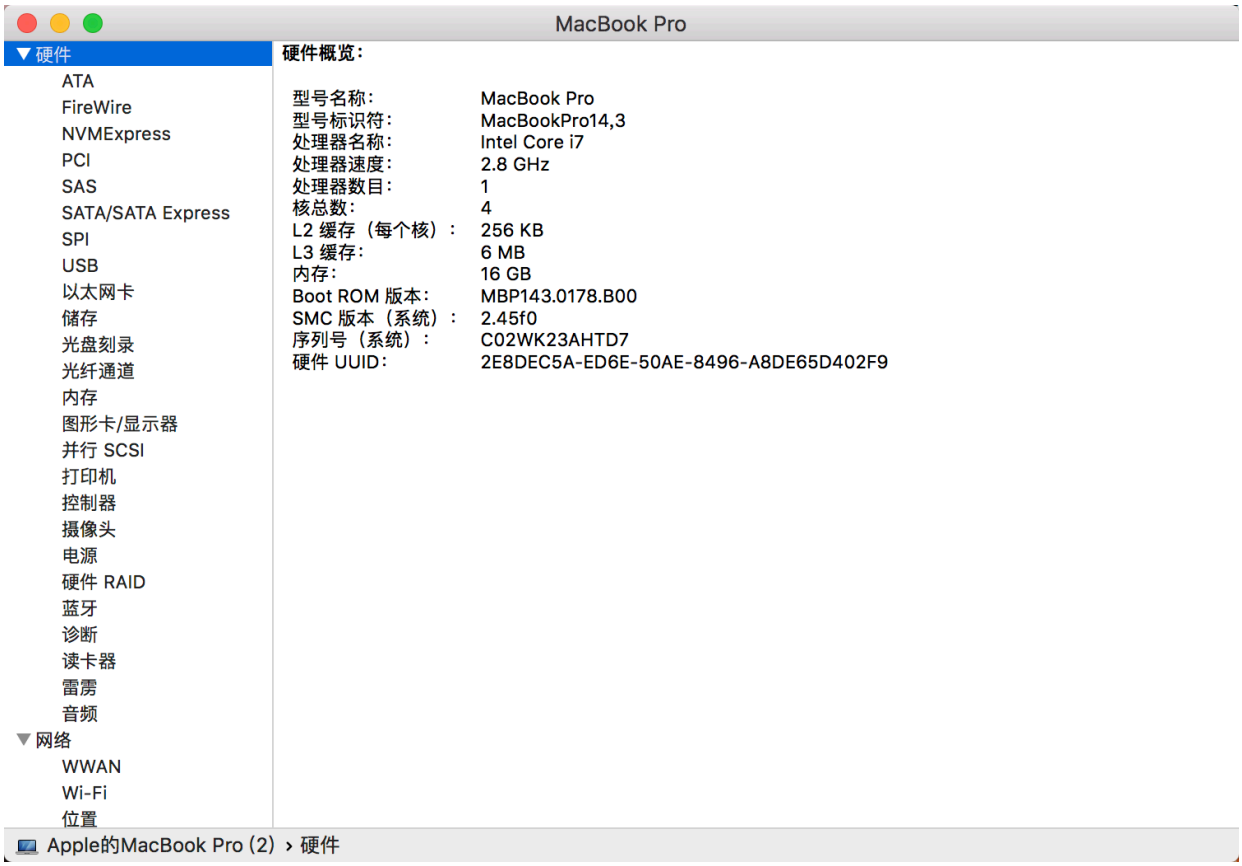
下载国旗的示例或其他 I/O 密集型作业使用 ProcessPoolExecutor 类得不到任何好处。这一点易于验证，只需把示例 17-3 中下面这几行：

```
def download_many(cc_list):
    workers = min(MAX_WORKERS, len(cc_list))
    with futures.ThreadPoolExecutor(workers) as executor:
```

改成：

```
def download_many(cc_list):
    with futures.ProcessPoolExecutor() as executor:
```

实验环境：



实验0: 单进程单线程基准情况

```
try:0
BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
20 flags downloaded in 3.64s
try:1
BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
20 flags downloaded in 3.84s
try:2
BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
20 flags downloaded in 3.73s
try:3
BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
20 flags downloaded in 3.79s
try:4
BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
20 flags downloaded in 3.60s
avarage time: 6.20284
```

实验0.5: 多线程 worker=2

```
try:0
FR CN BD BR DE EG ID ET JP IN NG PK CD MX PH IR RU TR VN US

20 flags downloaded in 0.45s
try:1
CN BR BD DE EG FR JP ET IN ID PK NG PH CD MX IR RU TR VN US

20 flags downloaded in 0.48s
try:2
BD DE BR CN EG FR IN ID ET JP PK NG CD MX PH IR RU VN TR US

20 flags downloaded in 0.46s
try:3
BR BD CN EG FR IN JP ET ID DE NG PK PH MX CD IR RU TR VN US

20 flags downloaded in 0.45s
try:4
BR BD JP ID CN EG NG FR IN DE PK ET PH MX CD IR RU VN TR US

20 flags downloaded in 0.45s
avarage time: 0.76302
```

实验1: 多线程 worker=4

```
try:0
DE JP BR BD IN NG EG ID RU ET PK FR CN TR PH MX VN US CD IR
20 flags downloaded in 0.26s
try:1
BD BR DE CN EG ID IN ET JP PK FR NG RU TR MX VN CD IR US PH
20 flags downloaded in 0.28s
try:2
BD DE BR EG CN FR ID NG JP IN RU VN TR PK PH MX CD IR US ET
20 flags downloaded in 0.27s
try:3
BD BR CN DE IN EG ET ID JP PK MX VN FR NG RU TR PH CD IR US
20 flags downloaded in 0.31s
try:4
EG BD BR CN DE FR ID IN JP ET PK NG RU TR MX CD IR US PH VN
20 flags downloaded in 0.34s
avarage time: 0.48705
```

实验2:多线程 worker=8

```
try:0
PK BD JP BR MX CN ID ET RU IN DE FR PH TR VN US NG IR EG CD
20 flags downloaded in 0.40s
try:1
BR CN ID NG JP ET EG FR BD RU TR US MX DE CD PH IN IR PK VN
20 flags downloaded in 0.33s
try:2
BD BR CN ID NG JP IN EG PH PK FR CD TR VN RU MX IR US ET DE
20 flags downloaded in 0.61s
try:3
BR EG DE FR ID IN BD JP ET NG CN RU TR PH VN IR US CD MX PK
20 flags downloaded in 0.25s
try:4
BD BR IN EG ET ID JP VN FR NG TR MX CD RU IR DE PH PK US CN
20 flags downloaded in 0.31s
avarage time: 0.37950
```

实验3:多线程 worker=16

```
try:0
BD CN NG FR EG BR DE JP ID VN MX PK RU PH TR CD IN IR US ET
20 flags downloaded in 0.32s
try:1
BR BD DE CN FR EG ID IN ET NG RU TR PK JP VN MX PH CD US IR
20 flags downloaded in 0.27s
try:2
BD DE BR CN FR EG NG JP ID RU MX ET CD PH TR US PK VN IR IN
20 flags downloaded in 0.27s
try:3
BD CN EG DE JP ET RU FR ID BR TR MX PK VN CD PH IR IN NG US
20 flags downloaded in 0.31s
try:4
BR CN EG FR ID DE IN ET JP BD PK NG MX RU VN CD PH TR IR US
20 flags downloaded in 0.29s
avarage time: 0.29106
```

实验4:多进程 worker=2

```
try:0
BD BR CN CD DE EG ET FR ID IN JP IR MX NG PH PK RU TR VN US
20 flags downloaded in 2.12s
try:1
BD BR CN CD DE EG FR ET ID IN JP IR MX NG PH PK RU TR US VN
20 flags downloaded in 2.17s
try:2
BR BD CN CD DE EG ET FR ID IN JP IR NG MX PK PH RU TR VN US
20 flags downloaded in 2.14s
try:3
BD BR CN CD DE EG ET FR IN ID JP IR MX NG PH PK RU TR VN US
20 flags downloaded in 2.15s
try:4
BD BR CN CD DE EG ET FR ID IN JP IR NG MX PK PH RU TR VN US
20 flags downloaded in 2.21s
avarage time: 3.59709
```

实验5:多进程 使用默认核数（4）

```
try:0
BD EG DE CN FR BR ET CD NG ID JP IN MX PH PK IR TR VN RU US
20 flags downloaded in 0.71s
try:1
DE BR ET BD EG FR CN CD ID NG IN JP PH MX PK IR RU VN TR US
20 flags downloaded in 0.72s
try:2
FR EG BR BD DE CN ET CD JP ID IN NG PH MX IR PK TR RU VN US
20 flags downloaded in 0.70s
try:3
CN BD FR EG BR ET CD DE IN ID NG JP MX IR PH PK TR VN RU US
20 flags downloaded in 0.73s
try:4
ET CN BR EG FR BD DE CD ID JP IN NG PK PH MX IR RU TR VN US
20 flags downloaded in 0.72s
avarage time: 1.19615
```

实验6: 多进程 worker=8

```
try:0
EG BR BD FR DE CN ET CD IN ID JP MX NG PK IR PH RU TR VN US
20 flags downloaded in 0.77s
try:1
BR DE CN EG FR BD ET CD ID IN JP NG MX PH IR PK RU TR VN US
20 flags downloaded in 0.74s
try:2
BR EG FR ET CN BD DE CD ID IN NG JP PK IR PH MX RU TR VN US
20 flags downloaded in 0.72s
try:3
CN FR EG BR ET BD DE CD IN JP NG ID PH MX IR PK TR VN RU US
20 flags downloaded in 0.72s
try:4
BR EG FR DE ET BD CD CN ID IN JP NG MX IR PK PH TR RU VN US
20 flags downloaded in 0.71s
avarage time: 1.21968
```

实验7：多进程 worker=16

```
try:0
EG FR BR DE BD ET CD CN ID JP NG IN MX IR PK PH RU TR VN US

20 flags downloaded in 0.70s
try:1
DE BD CN FR BR EG ET CD ID IN JP NG MX PH IR PK TR RU VN US

20 flags downloaded in 0.69s
try:2
ET BD FR BR CN EG DE CD JP IN ID NG MX PH PK IR RU TR VN US

20 flags downloaded in 0.71s
try:3
DE BD BR FR EG ET CN CD ID IN JP NG MX PH PK IR RU TR VN US

20 flags downloaded in 0.72s
try:4
EG ET BR DE BD CN CD FR ID JP IN NG MX IR PH PK RU TR VN US

20 flags downloaded in 0.69s
avarage time: 1.16917
```

由此我们得出以下数据：

核数	线程	平均时间（秒）
1	1	6.20284
1	2	0.76302
1	4	0.48705
1	8	0.37950
1	16	0.29106
2	1	3.59709
4	1	1.19615
8	1	1.21968
16	1	1.16917

我们可以看出，最慢的情况为单进程单线程的情况，平均使用时间为6.20284。

当我们提升线程数时，时间一下子被压缩到了0.48705，但当我们成倍提升线程数时，虽然平均时间也在减少，但是并没有质的飞跃了。

当我们提升进程数时，在cpu核数之内的进程时，效果也是比较明显，但是当核数超过了最大值时，便没有多大进步了。

以下几个问题，大家回答一下

1，为什么多线程多进程都有了一定效果?

2，为什么多线程在worker=4之后提升的效果就很差了?

3，为什么多进程在worker=4之后完全没有提升，反而可能下降呢?

4，为什么同样是4个进程和4个线程，4个进程要比4个线程慢?

解答

1，因为多进程与多线程都增加了worker的数量，所以执行时间都得到了比较大的减少。

2，这个与资源返回的速度和代码执行速度有关。

3，因为到了cpu核数之后，多进程便无法真*并行了。

4，因为进程之内是单线程的，在一个进程阻塞时不会有其他线程被调度上来。

最后一个问题，为什么对于CPU密集型Python多线程基本没有用呢?

答案显而易见，因为Python多线程只有在进行IO时或Sleep时才会被激活，而CPU密集型的任务很少有进行IO或者Sleep的操作。