# 线程和进程

## 对比

* 进程是对运行时程序的封装，是系统资源调度和分配的基本单位
* 线程是进程的子任务，CPU调度和分配的基本单位，实现进程内并发
* 一个进程可以包含多个线程，线程依赖进程存在，并共享进程内存

## 线程安全

* 一个操作可以在多线程环境中安全使用，获取正确的结果
* 线程安全的操作好比线程是顺序执行而不是并发执行的（i+=1）
* 一般如果涉及到写操作需要考虑如何让多个线程安全访问数据

## 进程间通信的方式

共享内存，信号量，套接字

## Python中如何使用多线程

threading, start, join

## Python中如何使用多进程

multiprocessing

## 几个概念

### 进程 vs 线程

* 进程 进程是表示资源分配的基本单位，又是调度运行的基本单位。程序并不能单独运行，只有将程序装载到内存中，系统为它分配资源才能运行，而这种执行的程序就称之为进程。程序和进程的区别就在于：程序是指令的集合，它是进程运行的静态描述文本；进程是程序的一次执行活动，属于动态概念。
* 线程 线程是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。

### 同步 vs 异步

* 同步 不同程序单元为了完成某个任务，在执行过程中需靠某种通信方式以协调一致，称这些程序单元是同步执行的。同步意味着有序，按顺序执行。
* 异步 为完成某个任务，不同程序单元之间过程中无需通信协调，即多个任务之间没有先后顺序.

### 阻塞 vs 非阻塞

* 阻塞 阻塞是当请求不能满足的时候就将进程挂起，使调用者不能继续往下执行。
* 非阻塞 非阻塞则不会阻塞当前进程，可以继续执行，就是说非阻塞的。

### 并行 vs 并发

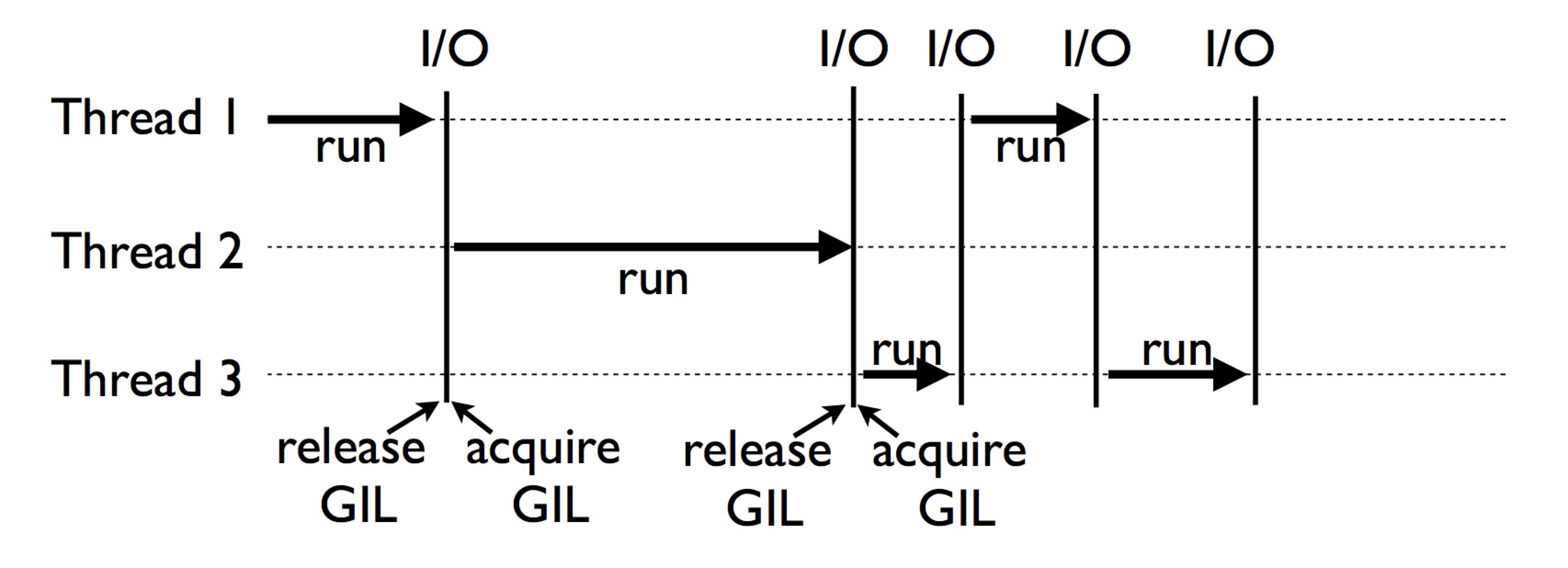
* 并发 并发描述的是程序的组织结构。指程序要被设计成多个可独立执行的子任务。以利用有限的计算机资源使多个任务可以被实时或近实时执行为目的。
* 并行 并行描述的是程序的执行状态。指多个任务同时被执行。以利用富余计算资源（多核 CPU）加速完成多个任务为目的。

## 全局解释器锁（GIL）

全局解释器锁（GIL）表示在同一时刻只有一个线程对共享资源进行存取。

GIL 并不是 Python 的特性，Python 完全可以不依赖于 GIL。

先明确的一点是 GIL 并不是 Python 的特性，它是在实现 Python 解析器(CPython)时所引入的一个概念。就好比 C++ 是一套语言（语法）标准，但是可以用不同的编译器来编译成可执行代码。Python 也一样，同样一段代码可以通过 CPython、PyPy、Psyco 等不同的 Python 执行环境来执行。像其中的 JPython 就没有 GIL。然而因为 CPython 是大部分环境下默认的 Python 执行环境。所以在很多人的概念里 CPython 就是 Python，也就想当然的把 GIL 归结为 Python 语言的缺陷。



GIL

线程何时切换？一个线程无论何时开始睡眠或等待网络 I/O，其他线程总有机会获取 GIL 执行 Python 代码。这是协同式多任务处理。CPython 也还有抢占式多任务处理。如果一个线程不间断地在 Python 2 中运行 1000 字节码指令，或者不间断地在 Python 3 运行15 毫秒，那么它便会放弃 GIL，而其他线程可以运行。

Python 的多线程在多核 CPU 上，只对于 IO 密集型计算产生正面效果；而当有至少有一个 CPU 密集型线程存在，那么多线程效率会由于 GIL 而大幅下降。

如何避免 GIL 对性能的影响：

* 多进程
* 使用别的解析器，比如 Jpython

## 多线程

多线程在 Python3 中 2 种实现方式：

* Threading 模块
* concurrent.futures

# 直接调用  
import threading  
import time  
  
def sayhi(num): #定义每个线程要运行的函数   
 print("running on number:%s" %num)   
 time.sleep(3)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 t1 = threading.Thread(target=sayhi,args=(1,)) #生成一个线程实例  
 t2 = threading.Thread(target=sayhi,args=(2,)) #生成另一个线程实例  
 t1.start() #启动线程  
 t2.start() #启动另一个线程  
  
 print(t1.getName()) #获取线程名  
 print(t2.getName())  
# 继承调用  
import threading  
import time  
  
class MyThread(threading.Thread):  
 def \_\_init\_\_(self,num):  
 threading.Thread.\_\_init\_\_(self)  
 self.num = num  
  
 def run(self):#定义每个线程要运行的函数  
 print("running on number:%s" %self.num)  
 time.sleep(3)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 t1 = MyThread(1)  
 t2 = MyThread(2)  
 t1.start()  
 t2.start()  
# 线程池  
from concurrent.futures import ThreadPoolExecutor  
import time  
  
def return\_future\_result(message):  
 time.sleep(0.5)  
 return message  
  
pool = ThreadPoolExecutor(max\_workers=2) # 创建一个最大可容纳2个task的线程池  
future1 = pool.submit(return\_future\_result, ("hello")) # 往线程池里面加入一个task  
future2 = pool.submit(return\_future\_result, ("world")) # 往线程池里面加入一个task  
print(future1.done()) # 判断task1是否结束  
time.sleep(1)  
print(future2.done()) # 判断task2是否结束  
print(future1.result()) # 查看task1返回的结果  
print(future2.result()) # 查看task2返回的结果k

## 多进程

多进程 在 Python3 中 2 种实现方式：

* multiprocessing
* concurrent.futures ProcessPoolExecutor

from multiprocessing import Process  
import time  
def f(name):  
 time.sleep(2)  
 print('hello', name)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 p = Process(target=f, args=('bob',))  
 p.start()  
 p.join()  
  
from multiprocessing import Process  
import os  
  
def info(title):  
 print(title)  
 print('module name:', \_\_name\_\_)  
 print('parent process:', os.getppid())  
 print('process id:', os.getpid())  
 print("\n\n")  
  
def f(name):  
 info('\033[31;1mfunction f\033[0m')  
 print('hello', name)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 info('\033[32;1mmain process line\033[0m')  
 p = Process(target=f, args=('bob',))  
 p.start()  
 p.join()  
  
from concurrent.futures import ProcessPoolExecutor  
import os,time,random  
def task(n):  
 print('%s is running' %os.getpid())  
 time.sleep(0.2)  
 return n\*\*2  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 p=ProcessPoolExecutor() #不填则默认为cpu的个数  
 l=[]  
 start=time.time()  
 start = time.time()  
 with ProcessPoolExecutor() as p:  
 future\_tasks = [p.submit(task, i) for i in range(10)]  
 print('=' \* 30)  
 print([obj.result() for obj in future\_tasks])  
 print(time.time() - start)

## 参考资料

《全面剖析 Python 面试知识点》

https://gitbook.cn/books/5ca40fd11763103ff10b0e43/index.html