信号的使用：

import signal

signal.alarm(sec)

功能; 向自身发送时钟信号---》SIGALRM

参数 ： sec时钟时间

* 进程中只能有一个时钟，第二个会覆盖第一个时钟

同步执行 ：按照顺序逐句执行，一步完成在做下一步

异步指行 ：在执行过程中利用内核记录延迟发生或者准备处理的事件。这样不影响应用层的持续执行。当事件发生时在由内核告知应用层处理

* 信号是唯一的异步通信的方法

signal.pause()

功能：阻塞等待接收一个信号

signal.signal(signum，handler)

功能： 处理信号

参数： signal要处理的信号

handler信号的处理方法

SIG\_DFL 表示使用默认的方法处理

SIG\_IGN 表示忽略这个信号

func传入一个函数表示用指定函数处理

def func(sig,frame)

sig : 捕获到的信号

frame ： 信号对象

信号量（信号灯）

原理：给定一个数量，对多个进程课件，且多个进程都可以操作，进程通过对数量多少的判断执行各自的行为。

multiprocessing—》Semaphore()

sem == Semaphore(num)

功能： 创建信号量

参数：信号量初始值

返回：信号量对象

sem.get\_value() 获取信号量值

sem.acpuire() 将信号量减1当信号量为0会阻塞

sem.release() 将信号量加1

进程的同步和互斥

临界资源 ： 多个进程或者线程都能够操作的共享资源

临界区： 操作临界资源的代码段

同步：同步是一种合作关系，为完成某个任务，多进程或者多线程之间形成一种协调，按照约定或者条件执行操作临界资源、

互斥：互斥是一种制约关系，当一个进程或者线程使用临界资源时进行上锁处理，当拎一个进程使用时会阻塞等待，直到解锁后才能继续使用。

Event事件

multiprocessing –》Event

创建事件对象

e =Event()

设置事件阻塞

e.wait([timeout])

事件设置 当事件被设置后e.wait()不在阻塞

e.set()

清除设置 当事件设置被clear后，e.wait又会阻塞

e.clear()

事件状态判断

e.is\_set()

Lock锁

创建对象

lock = Lock()

lock.acquire()上锁 如果锁已经是上锁状态调用此函数会阻塞

lock.release()解锁

with lock ： 上锁

。。。

。。。

解锁

线程：

什么是线程

线程也是一种多任务编程方法，可以利用计算机多核资源完成程序的并发执行，线程又被称为轻量级的进程。

线程特征：

* 线程计算机多核分配的最小单位
* 一个进程可以包含多个线程
* 线程也是一个运行的过程，消耗计算机资源，多个线程共享进程的资源和空间
* 线程的创建删除消耗的资源都要远远小于进程
* 多个线程之间执行互不干扰
* 线程也有自己的特有属性，比如指令集，ID

threading 模块创建线程

threading.Thread()

功能 ： 创建线程对象

参数 ： name线程名字 默认为Thread-1

target 线程函数

args 元组 给线程函数位置传参

kwargs 字典 给线程函数键值传参

t.start() 启动线程 自动运行线程函数

t.join([timeout]) 回收线程

线程对象属性

t.is\_alive() 查看线程状态

t.name线程名称

t.setName() 设置线程名称

t.getName() 获取线程名称

threadining.currentThread()获取当前线程对象

t．daemon（）属性

默认情况主线程退出不会影响分支线程执行如果设置为true则分支线程随主线程退出

设置方法：

t.daemon=true

t.setDaemon()

\* 要在start前设置，不会和join同用

创建自己的线程类

步骤：

1. 继承Thread
2. 加载Thread中的init
3. 重写run方法

传参，类，属性，继承，init

线程通信：

通信方法：多个线程共享进程的空间，所以线程间通信使用全局变量完成。

注意事项：线程间使用全局变量往往要同步互斥机制保证通信安全。

线程同步互斥方法;

线程的event

e = theading.Event() 创建事件对象

e.wait([timeout])如果e为设置状态则不阻塞否则阻塞

e.set()将e变为设置状态

e.clear()清除设置

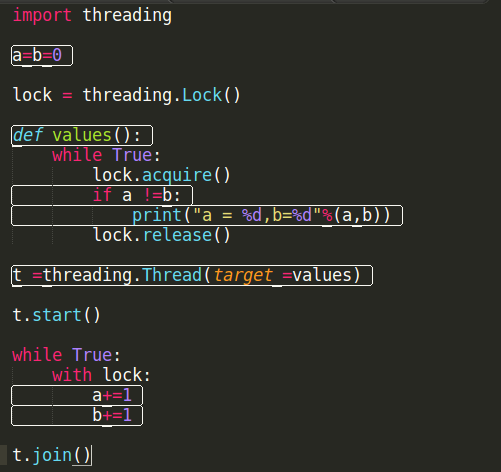
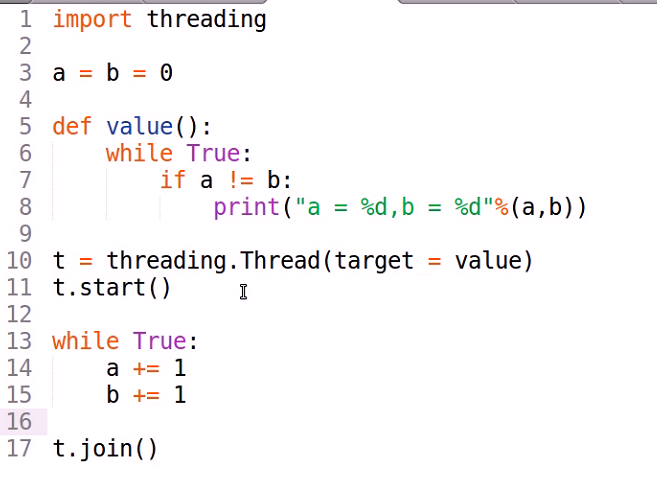
线程锁

lock = threading.lock()创建线程锁对象

lock.acquire() 上锁

lock .release() 解锁

* 也可以通过with上锁，上锁 状态调用acquire会阻塞



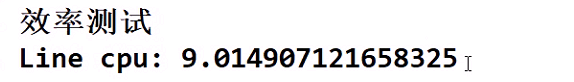
Python线程的GIL问题（全局解释器锁）

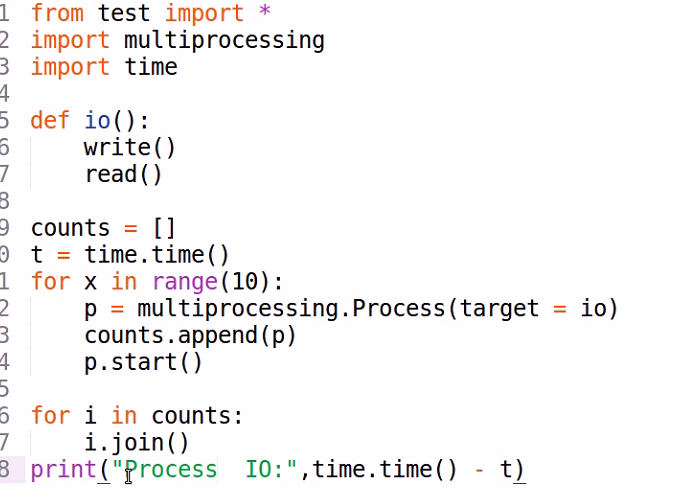
Python---》支持多线程---》同步互斥---》加锁---》超级锁--给解释器加锁----》解释器统一时刻只能解释一个线程

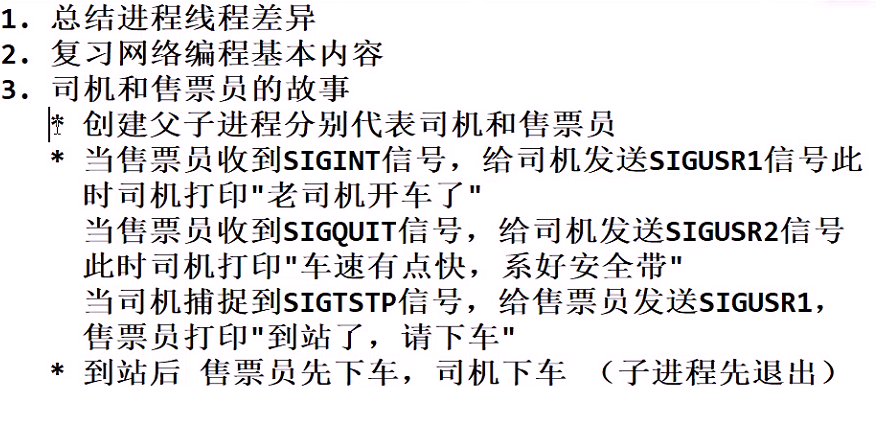
后果：一个解释器同一个时刻只能解释执行一个线程，所以导致Python线程效率低下。但是当遇到IO阻塞时线程会主动让出解释器，因此Python线程更加适合高延迟的IO程序并发。

解决方法：

* 尽量用进程完成并发
* 不适用c解释器
* 尽量使用多种方案组合得到方式进行并发操作，线程用作高延迟IO



sigusr1



1. 创建父为司机子为售票员进程，利用multiprocessing
2. 信号通信：子进程判断是否收到信号，收到信号发送给父进程