**[Python Day 32 并发编程 (守护进程, 进程同步(multiprocessing.Lock、multiprocessing.Semaphore、multiprocessing.Event) 进程间通信 multiprocessing.Queue)](https://www.cnblogs.com/eailoo/p/9174257.html)**

**守护进程**

注意：进程之间是互相独立的，主进程代码运行结束，守护进程随即终止(主进程和子进程是异步的),当主进程停止,该守护进程不在继续执行.守护进程也是一种子进程.

主进程创建守护进程

　　其一：守护进程会在主进程代码执行结束后就终止.(但本质上是在主进程结束之前结束的,主进程需要负责回收资源)

　　其二：守护进程内无法再开启子进程,否则抛出异常：AssertionError: daemonic processes are not allowed to have children

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 创建守护进程

**多进程中的方法**

**p = Process(target=func,args=(1,)) #创建一个进程对象**

       p.start()　　启动一个进程

　　p.daemon = True　　设置进程为守护进程,随主进程结束而结束.

　　p.is\_alive()　　判断进程是否存活,返回bool值

　　p.terminate()　　发送给操作系统指令,关闭进程

　　p.pid() 　　查看进程pid

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Process

import time

import os

def func(num):

print(f'{num},pid:{os.getpid()},ppid:{os.getppid()}')

while True:

print('is alive')

time.sleep(0.5)

def wahaha():

i = 0

while i < 10:

i += 1

print(f'第{i}秒')

time.sleep(1)

if \_\_name\_\_ == '\_\_main\_\_':

p2 = Process(target=wahaha)

p2.start()

#

p = Process(target=func,args=(1,))

p.daemon = True #主进程结束,该子进程结束

p.start()

time.sleep(3)

print(p.is\_alive())

print(p2.is\_alive())

p2.terminate()

time.sleep(0.1)

print(p.is\_alive())

print(p2.is\_alive())

print(f'pid:{os.getpid()},ppid:{os.getppid()}')

print('主进程结束')

p2.join()

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import socket

from multiprocessing import Process

def talk(conn,addr):

while True:

msg\_r = conn.recv(1024).decode('utf-8')

print(addr,msg\_r)

msg\_s = 'client{}登陆'.format(addr)

conn.send(msg\_s.encode('utf-8'))

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

sk = socket.socket()

sk.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1)

sk.bind(('127.0.0.1',8091))

sk.listen(5)

try:

while True:

conn,addr = sk.accept()

Process(target=talk,args=(conn,addr)).start()

finally:

sk.close()

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import socket

sk = socket.socket()

sk.connect(('127.0.0.1',8091))

while True:

msg\_s = input('请输入内容:')

sk.send(msg\_s.encode('utf-8'))

msg\_r = sk.recv(1024).decode('utf-8')

print(msg\_r)

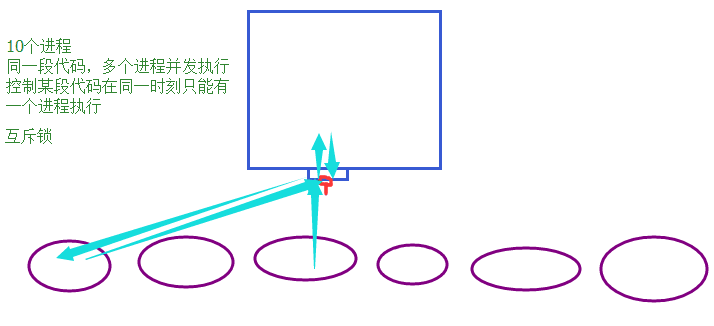
sk.close()

[复制代码](javascript:void(0);)

**进程同步(multiprocessing.Lock、multiprocessing.Semaphore、multiprocessing.Event)**

**锁 multiprocessing.Lock (\*\*\*\*\*)**

　　避免同一段代码被多个进程同时执行



　　lock = Lock()　　创建锁对象

　　lock.acquire()　　查询钥匙,如果有就拿走,如果没有就等待

　　lock.release()　　归还钥匙

　　lock可以使用with上下文进行管理(类似于文件读取)

　　with lock:

　　　　print('hello' )

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

维护数据的安全

降低了程序的效率

所有的效率都是建立在数据安全的角度上的

但凡涉及到并发编程都要考虑数据的安全性

我们需要在并发部分对数据修改的操作格外小心，如果会涉及到数据的不安全，就需要进行加锁控制

lock acquire release的另外一种用法

lock 内部实现了进程之间的通信，使得谁acquire了谁release了能够在多个拥有lock参数的子进程中透明

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Lock

lock = Lock() #创建一个锁对象

lock.acquire() #想拿钥匙,如果有就拿,没有就一直等

print('拿到要钥匙了1')

lock.release() #还钥匙

lock.acquire() #想拿钥匙

print('拿到要钥匙了2')

lock.release() #还钥匙

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

#db文件内容 {"count": 0}

import json

import time

from multiprocessing import Process,Lock

def search(i):

f =open('db')

ticket\_dic =json.load(f)

f.close()

print(f"{i} 正在查票,剩余票数{ticket\_dic['count']}")

def buy(i):

with open('db') as f: ticket\_dic = json.load(f)

time.sleep(0.2)

if ticket\_dic['count'] > 0:

ticket\_dic['count'] -= 1

print(f'{i} 买到票了')

time.sleep(0.2)

with open('db','w') as f :json.dump(ticket\_dic,f)

else:

print(f"{i} 太火爆被抢购一空了,剩余票数{ticket\_dic['count']}")

# def get\_ticket(i,lock):

# search(i)

# lock.acquire()

# buy(i)

# lock.release()

def get\_ticket(i,lock):

search(i)

with lock:

buy(i)

if \_\_name\_\_ == '\_\_main\_\_':

lock = Lock()

for i in range(10):

p = Process(target=get\_ticket,args=(i,lock))

p.start()

[复制代码](javascript:void(0);)

**信号量(标志True False) multiprocessing.Semaphore(\*\*\*)  (锁+计数器)**

　　有多个钥匙的锁

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

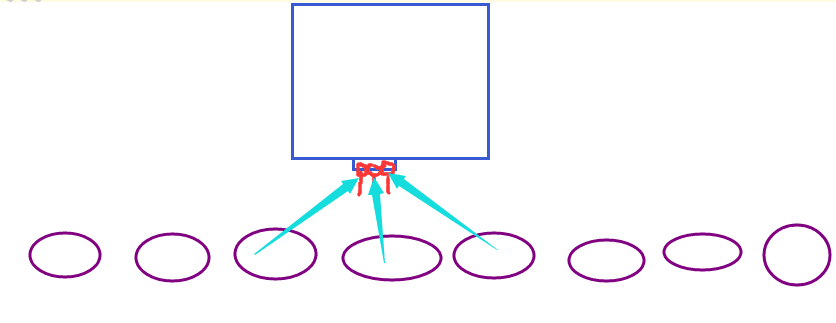
互斥锁同时只允许一个线程更改数据，而信号量Semaphore是同时允许一定数量的线程更改数据 。

假设商场里有4个迷你唱吧，所以同时可以进去4个人，如果来了第五个人就要在外面等待，等到有人出来才能再进去玩。

实现：

信号量同步基于内部计数器，每调用一次acquire()，计数器减1；每调用一次release()，计数器加1.当计数器为0时，acquire()调用被阻塞。这是迪科斯彻（Dijkstra）信号量概念P()和V()的Python实现。信号量同步机制适用于访问像服务器这样的有限资源。

信号量与进程池的概念很像，但是要区分开，信号量涉及到加锁的概念



　　sem = Semaphore(4)　　创建锁对象,4把钥匙,可以被连续acquire4次

　　sem.acquire()　　查询钥匙,如果有就拿走,如果没有就等待

　　sem.release()　　归还钥匙

　　sem 可以使用with上下文进行管理(类似于文件读取)

　　with sem:

　　　　print('hello' )

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Semaphore

sem = Semaphore(4) #4把钥匙

sem.acquire()

print(1)

sem.acquire()

print(2)

sem.release()

sem.acquire()

print(3)

sem.acquire()

print(4)

sem.acquire()

print(5)

sem.acquire()

print(6)

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Semaphore,Process

import time

import random

# def ktv(sem,i):

# sem.acquire()

# print(f'{i}走进ktv')

# time.sleep(random.randint(1,3))

# print(f'{i}走出ktv')

# sem.release()

def ktv(sem,i):

with sem:

print(f'{i}走进ktv')

time.sleep(random.randint(1,3))

print(f'{i}走出ktv')

if \_\_name\_\_ == '\_\_main\_\_':

sem = Semaphore(4)

for i in range(10):

p = Process(target=ktv,args=(sem,i))

p.start()

[复制代码](javascript:void(0);)

**事件 multiprocessing.Event(\*\*)**

　　控制子进程执行还是阻塞的一个机制

　　e = Event()　　创建一个事件对象

Event方法  在事件中有一个信号(标志)

　　wait()   如果这个标志是True wait的执行效果就是pass ,如果是False,wait方法的效果就是阻塞,直到这个标志变成True

*控制标志方法*

*is\_set()　　判断标志的状态,返回bool值*

*set()　　将标志设置为True*

*clear()　　将标志设置为False*

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

from multiprocessing import Event

e = Event() #阻塞,事件的创建之初标志的状态是False

print(e.is\_set())

e.set() #将标志改为True

print(e.is\_set())

e.wait() #当标志为True是pass,不阻塞

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Event,Process

import time

def func1(e):

print('start func1')

print(e.is\_set()) #事件创建之初是False

e.wait(1) #不修改状态(网络测试,发送短信,发送邮件)，超时后继续执行，不继续阻塞

print(e.is\_set())

e.wait() #持续阻塞

print(e.is\_set()) #主进程3(异步)s后修改信号标志为True ，继续执行

print('end func1')

if \_\_name\_\_ == '\_\_main\_\_':

e = Event()

Process(target=func1,args=(e,)).start()

time.sleep(3)

e.set()

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Event,Process

import time

import random

def tarffic\_light(e):

while True:

while e.is\_set():

print('\033[1;32m绿灯亮\033[0m')

time.sleep(2)

e.clear()

else:

print('\033[1;31m红灯亮\033[0m')

time.sleep(2)

e.set()

def car(i,e):

while not e.is\_set():

print(f'{i}正在等待通过...')

e.wait()

else:

print(f'{i}通过.')

if \_\_name\_\_ == '\_\_main\_\_':

e = Event()

light = Process(target=tarffic\_light,args=(e,))

light.daemon =True

light.start()

car\_list = []

for i in range(1,21):

p = Process(target=car,args=(i,e))

car\_list.append(p)

p.start()

time.sleep(random.randint(0,3))

for i2 in car\_list:i2.join() #控制子进程先执行完毕

print('执行完啦')

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import time

import random

from multiprocessing import Process,Event

def traffic\_light(e):

print('\033[1;31m红灯亮\033[0m')

while True:

time.sleep(2)

if e.is\_set():

print('\033[1;31m红灯亮\033[0m')

e.clear()

else:

print('\033[1;32m绿灯亮\033[0m')

e.set()

def car(i,e):

if not e.is\_set():

print('car%s正在等在通过'%i)

e.wait()

print('car%s通过'%i)

if \_\_name\_\_ == '\_\_main\_\_':

e = Event()

light = Process(target=traffic\_light,args=(e,))

light.daemon = True

light.start()

car\_lst = []

for i in range(20):

p = Process(target=car,args=(i,e))

p.start()

time.sleep(random.randint(0,3))

car\_lst.append(p)

for car in car\_lst:car.join()

[复制代码](javascript:void(0);)

说明:红绿灯的的变化和汽车的通行是两个独立的进程,汽车通过对红绿灯的事件信号的查询判断等待和放行,每一个汽车都是独立的进程

**进程间通信(进程之间数据共享)**

**进程间通信 IPC(Inter-Process Communication)**

**队列 multiprocessing.Queue  (先进先出)  队列是基于（管道+锁）实现的**

*创建共享的进程队列，Queue是多进程安全的队列，可以使用Queue实现多进程之间的数据传递。*

Queue([maxsize])

创建共享的进程队列。

参数 ：maxsize是队列中允许的最大项数。如果省略此参数，则无大小限制。

底层队列使用管道和锁定实现。

　　q =Queue()　　创建一个队列   q = Queue(5)　队列长度为5

　　q.put(1)　　向队列中放一个数据,可以是int list dict ... 当队列满时会阻塞

　　q.get()　　从队列中获取一个数据  没有值会一直阻塞

　　q.empty()　　判断队列是否为空 返回bool值 　　多进程时不准 ,如果其他进程或线程正在往队列中添加项目，结果是不可靠的。也就是说，在返回和使用结果之间，队列中可能已经加入新的项目。

　　q.full()　　判断队列是否已满 返回bool值　　多进程时不准 由于线程的存在，结果也可能是不可靠的

　　q.qsize() 　　返回队列中目前项目的正确数量。此函数的结果并不可靠，因为在返回结果和在稍后程序中使用结果之间，队列中可能添加或删除了项目。在某些系统上，此方法可能引发NotImplementedError异常。

　　q.close() 　　关闭队列，防止队列中加入更多数据。调用此方法时，后台线程将继续写入那些已入队列但尚未写入的数据，但将在此方法完成时马上关闭。如果q被垃圾收集，将自动调用此方法。关闭队列不会在队列使用者中生成任何类型的数据结束信号或异常。例如，如果某个使用者正被阻塞在get（）操作上，关闭生产者中的队列不会导致get（）方法返回错误。

　　q.cancel\_join\_thread() 　　不会再进程退出时自动连接后台线程。这可以防止join\_thread()方法阻塞。

q.join\_thread() 　　连接队列的后台线程。此方法用于在调用q.close()方法后，等待所有队列项被消耗。默认情况下，此方法由不是q的原始创建者的所有进程调用。调用q.cancel\_join\_thread()方法可以禁止这种行为。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

q = Queue(3)

try:

q.get\_nowait()

except:

print('队列中没有值')

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

q = Queue(3)

q.put(1)

q.put('aaa')

q.put([1,2,3])

# q.put('alex') #队列满会阻塞

try:

q.put\_nowait('alex')

except:

print('丢失了一个数据')

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif创建一个队列

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from multiprocessing import Process,Queue

def func(num,q):

q.put({num:num\*\*num})

if \_\_name\_\_ == '\_\_main\_\_':

q = Queue()

# p = Process(target=func, args=(10,q))

# p.start()

# print(q.get())

for i in range(10):

p = Process(target=func,args=(i,q))

p.start()

for i in range(10):

print(q.get())

=============

{0: 1}

{1: 1}

{3: 27}

{4: 256}

{2: 4}

{5: 3125}

{9: 387420489}

{8: 16777216}

{6: 46656}

{7: 823543}

[复制代码](javascript:void(0);)