School of Economics and Management, Beihang University

现代程序设计技术

赵吉昌

本周内容

- 面向对象编程
  - 多进程
  - 多线程
  - 信号（SI）

多进程

- 内存共享
  - 通过Value，Array实现内存共享
    - 返回一个从共享内存上创建的 ctypes 对象
    - 从共享内存中申请并返回一个具有ctypes类型的数组对象
    - Demo: mpva.py
  - 通过Manager实现内存共享
    - Manager返回的管理器对象控制一个服务进程，且由该进程保存Python对象并允许其他进程通过代理操作对象
    - 返回的管理器支持类型支持list、dict等{可嵌套}
    - 注意同步：可能需要加锁，尤其是碰到+=更新时
  - Demo: mpm.py

多进程

- 进程池
  - 进程开启过多导致效率下降（同步、切换成本）
  - 应固定工作进程的数目
    - 由这些进程执行所有任务，而非开启更多的进程
    - 与CPU的核数相关

多进程

- 创建进程池
  - Pool([numprocess [,initializer [, initargs]]])
    - numprocess
      - 要创建的进程数，默认使用os.cpu_count()的值
    - initializer
      - 每个工作进程启动时要执行的可调用对象，默认为None
    - initargs
      - initializer的参数
  - p.apply()
    - 同步调用
    - 只有一个进程执行（不并行）
    - 但可以直接得到返回结果（阻塞至返回结果）

多进程

- 进程池
  - p.apply_async()
    - 异步调用
    - 并行执行，结果不一定马上返回（AsyncResult）
    - 可以有回调函数：进程池中任意进程完成任务后会立即通知主进程，主进程将调用另一个函数去处理该结果，该函数即回调函数，其参数为返回结果
  - p.close()
    - 关闭进程池
  - p.join()
    - 等待所有工作进程退出，只能在close()或teminate()之后调用

多进程

- Demo
  - mppo.py
- 补充：
  - 回调函数的参数只有一个，即结果
  - 回调函数是由主进程调用的
  - 回调函数应该迅速结束
  - 回调的顺序跟子进程启动的顺序无关
  - p.map()
    - 并行，主进程会等待所有子进程结束
  - p.map_async()
    - 并行，有回调函数

多进程

- ProcessPoolExecutor
  - 对multiprocessing进一步抽象
  - 提供更简单、统一的接口
  - submit(fn, *args, **kwargs)
    - returns a Future object representing the execution of the callable
    - 补充：马上调用Future的result()会阻塞
  - map(func, *iterables, timeout=None)
    - func is executed asynchronously, i.e., several calls to func may be made concurrently and returns an iterator of results

多进程

- ProcessPoolExecutor
  - shutdown(wait=True)
    - signal the executor that it should free any resources that it is using when the currently pending futures are done executing
    - regardless of the value of wait, the entire Python program will not exit until all pending futures are done executing
    - can avoid having to call this method explicitly if you use the with statement
  - Demo: mpoe.py

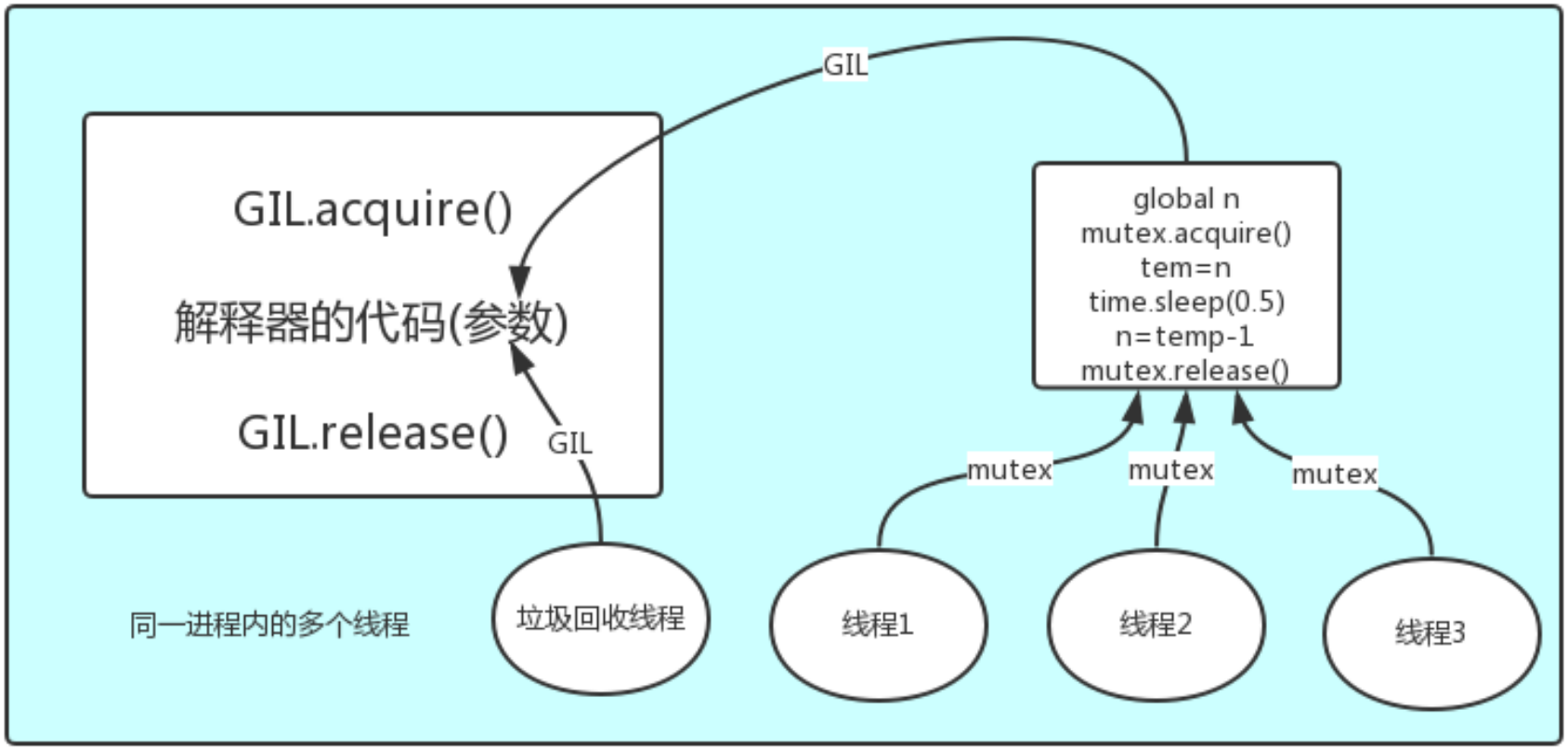
补充：多进程

- 分布式多进程
  - 多机环境
  - 跨设备数据交换
  - 如master-worker模型
  - 通过manager暴露Queue
  - Demo: mpd_server.py, mpd_worker.py mp_qm.py

多线程

- GIL(Global Interpreter Lock)
  - GIL非Python特性，而是实现Python解释器（Cpython）时引入的概念
  - GIL本质上是互斥锁，控制同一时间共享数据只能被一个任务修改，以保证数据安全
  - GIL在解释器级保护共享数据，在用户编程层面保护数据则需要自行加锁处理
  - Cpython解释器中，同一个进程下开启的多线程，同一时刻只能有一个线程执行，无法利用多核优势
    - 可能需要先获取GIL

多进程


GIL
GIL.acquire()
解释器的代码(参数)
GIL.release()
GIL
global n
mutex.acquire()
tem=n
time.sleep(0.5)
n=temp-1
mutex.release()
mutex
mutex
mutex
同一进程内的多个线程
垃圾回收线程
线程1
线程2
线程3

多线程

- GIL
  - 进程可以利用多核，但是开销大，而多线程开销小，但却无法利用多核
    - If you want your application to make better use of the computational resources of multi-core machines, you are advised to use multiprocessing or concurrent.futures.ProcessPoolExecutor
    - 多进程用于计算密集型，如金融分析
    - However, threading is still an appropriate model if you want to run multiple I/O-bound tasks simultaneously.
    - 多线程用于IO密集型，如socket，爬虫，web

多线程

- 多线程编程
  - threading模块
  - multiprocessing模块和threading模块在使用层面十分相似
  - threading.currentThread()
    - 返回当前的线程实例
  - threading.enumerate()
    - 返回所有正在运行线程的list
  - threading.activeCount()
    - 返回正在运行的线程数量，与len(threading.enumerate())结果相同

多线程

- 创建多线程
  - 通过指定target参数
  - 通过继承Thread类
  - 设置守护线程
    - setDaemon(True)
    - 应在start()之前
  - Demo: mt.py

多线程

- 线程同步
  - 锁(threading.Lock, threading.RLock,可重入锁)
    - 一旦线程获得重入锁，再次获取时将不阻塞
    - 线程必须在每次获取后释放一次
    - 区别：递归调用
  - 信号量(threading.Semaphore)
  - 事件(threading.Event)
  - 条件(threading.Condition)
    - Demo: mtc.py
  - 定时器(threading.Timer)
    - Demo: mtt.py

多线程

- 线程同步
  - Barrier
    - This class provides a simple synchronization primitive for use by a fixed number of threads that need to wait for each other.
    - Each of the threads tries to pass the barrier by calling the wait() method and will block until all of the threads have made their wait() calls. At this point, the threads are released simultaneously.
    - `Demo: mtb.py`

多线程

- 线程局部变量
  - 通过threding.local()创建
  - 线程在使用该类变量时会创建独立的拷贝
  - 看似全局变量（可以被各个线程调用），但各线程拥有各自己独立的副本
  - 避免数据的同步
  - Demo: mtl.py

多线程

- 队列
  - queue.Queue
  - queue.LifoQueue
  - queue.PriorityQueue （元素是元组，第一个元素为优先级）
  - Demo: mtq.py

多线程

- 线程池
  - ThreadPoolExecutor
    - assuming that ThreadPoolExecutor is often used to overlap I/O instead of CPU work and the number of workers should be higher than the number of workers for ProcessPoolExecutor
    - `as_completed(fs, timeout=None)`
      - Returns an iterator over the Future instances given by `fs` that yields futures as they complete (finished or cancelled futures)
    - 补充：注意参数与`future`的对应
    - `Demo: mtp.py mtqdm.py`

多进程和多线程的比较

- Demo: ptc.py

本周作业

- 以爬取网易云歌单为例，练习多线程的使用。
  - 1. 获取一个分类下的所有歌单的id。观察url可以发现其页码规律：https://music.163.com/#/discover/playlist/?order=hot&cat=%E8%AF%B4%E5%94%B1&limit=35&offset=35。offset是本页开始的数据位置，第1页是0，第2页是35。分类参数是utf-8编码后的汉字"说唱"，使用str.encode可以处理。
  - 2. 对每个id，获取歌单的详细信息，至少包括：歌单的封面图片（需把图片保存到本地）、歌单标题、创建者id、创建者昵称、介绍、歌曲数量、播放量、添加到播放列表次数、分享次数、评论数。可以自行实现其他信息的获取。基本信息汇总到同一张表中，以csv文件保存。https://music.163.com/playlist?id=3037221581
  - 3. 要求使用生产者-消费者模式实现，要求1作为生产者，每次请求后产生新的任务交给消费者，消费者执行要求2。
  - 4.（附加）爬虫程序往往需要稳定运行较长的时间，因此如果你的程序突然中断或异常（比如网络或被封），如何能够快速从断点重启？
  - 5.（附加）爬虫程序往往需要比较友好的状态输出，因此可否专门有一个线程 动态地进行输出更新，来显示当前的状态，比如程序连续运行的时长，要完成的 总页面数，其中有多少已被爬取，已收集的文件占用了多少空间，大概还需要多 少时间才能完成，预计需要耗费多少硬盘空间等。

SI：信号

- signal模块（有操作系统的依赖）
  - 信号：操作系统中进程间通讯的一种有限制的方式
  - 一种异步的通知机制，提醒进程一个事件已经发生
  - 当信号发送至进程时，操作系统将中断其执行
    - 任何非原子操作都将被中断
    - 如果进程定义了该信号的处理函数，将执行该函数，否则执行默认处理函数
    - `signal.signal(signal.SIGTSTP, handler)`
  - 可用于进程的中止
  - Python信号处理只在主线程中执行
    - 即使信号在另一个线程中接收
    - 信号不能被用作线程间通信的手段
    - 只有主线程才被允许设置新的信号处理程序
  - `Demo: sg.py`