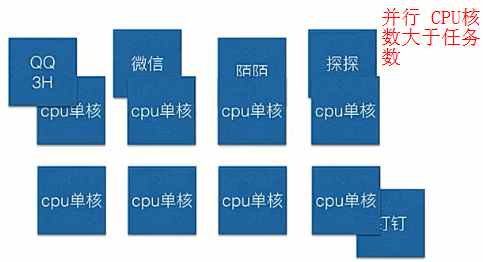
**单核CPU 时间片轮转 优先级调度**

**多核CPU 并发（任务数大于CPU核数，多任务交替执行，由于切换速度足够快，看起来同时执行，单核CPU同一时间只能处理一个任务） 并行 同一时间处理多任务**

****

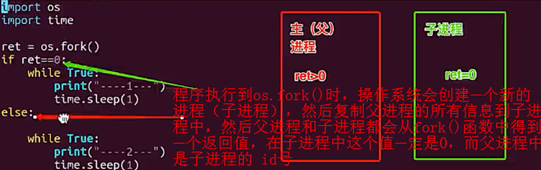
****

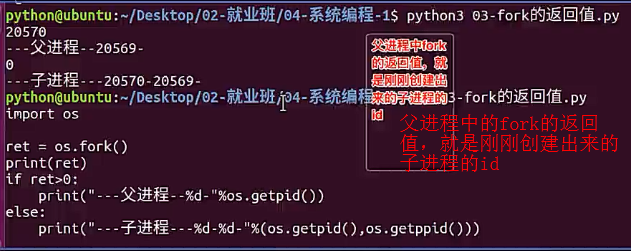
**进程 VS 程序 fork**

**编写完毕的代码，在没有运行的时候，称之为程序**

**正在运行着的代码，就成为进程**

**进程，除了包含代码以外，还有需要运行的环境等**

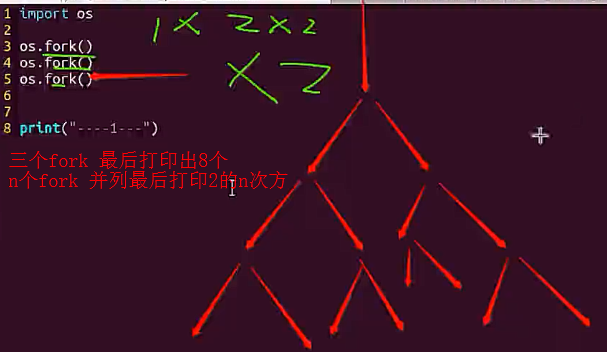
****

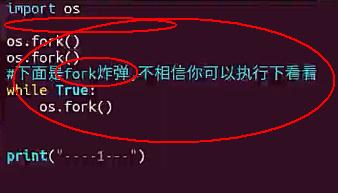
****

**多进程修改全局变量 多进程中，每个进程中所有数据（包括全局变量）都各有拥有⼀份，互不影响 进程好处完成多任务 缺点之间独立不能沟通**

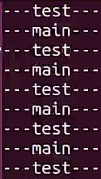
**多个fork的问题**

** **

****

****

**Multiprocessing** process类 proces创建的进程

** **

**Process创建的进程，等着所有子进程结束 主进程才结束 和fork的区别**

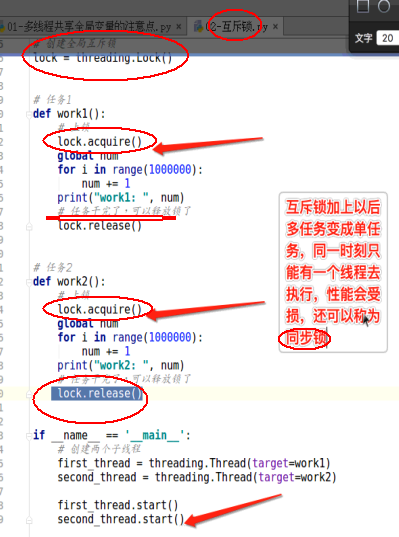
**join子进程**

**线程（程序中）**

**python的thread模块是比较底层的模块，python的threading模块是对thread做了一些包装**

****

**多线程共享全局变量**

****

**创建全局互斥锁 lock = threading.lock() lock.acquire() lock.release()**

**进程 一个运行起来的程序或者软件，每启动一次，进程都会向操作系统索要足够的运行资源，来保证程序能够正常运行起来，让线程执行相应的代码，也就是说进程是操作系统资源分配的基本单位。提示：每启动一次进程需要额外开辟运行资源，会有资源分配，进程只负责索要运行资源，真正干活的是在进程的线程中。可以把公司（提供电脑、桌椅等资源）理解成进程，把员工（负责用公司资源去干活）理解成线程**

**进程是如何完成多任务的 multiprocessing**

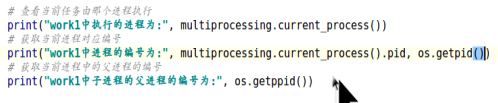
**Sub\_process = multiprocessing.Process(target=work1)**

**Sub\_process.start()**

**multiprocessing.current\_process()**

**multiprocessing.current\_process().pid**

****

****

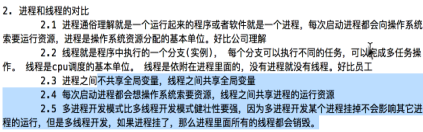
**进程之间不共享全局变量的**

**Read data read process target**

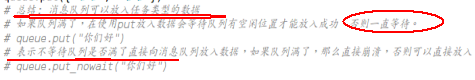
**2 进程和线程的区别**

**进程**

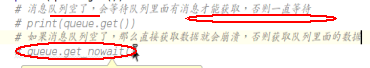
**线程是程序执行的一个分支（实例）**

****

**消息队列可以放任意类型的数据 如果队列满了 在使用put放入数据会等待队列有**

****

**如果消息队列**

****

**import threading  
import time  
# 定义全局变量  
num = 0  
# 任务1  
def work1():  
 global num  
 for i in range(1000000):  
 num += 1  
 print("work1: ", num)  
# 任务2  
def work2():  
 global num  
 for i in range(1000000):  
 num += 1  
 print("work2: ", num)  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 # 创建两个子线程  
 first\_thread = threading.Thread(target=work1)  
 second\_thread = threading.Thread(target=work2)  
 first\_thread.start()  
 # 1. time方式延时  
 # time.sleep(2)  
 # 总结： 为了数据不发生错了，可以让一个线程先执行完，让另外一个线程再去执行。  
 #按照一定顺序执行的任务就叫做同步  
 # 2. 线程等待 xxx. join(self,timeout) timeout线程等待时间 可以控制顺序  
 first\_thread.join()  
 second\_thread.start()  
 # 总结： 多个线程同时操作全局变量的时候注意资源竞争和数据错了的问题，可以使用同步方式解决问题**

****

**# 创建全局互斥锁 lock = threading.Lock()**

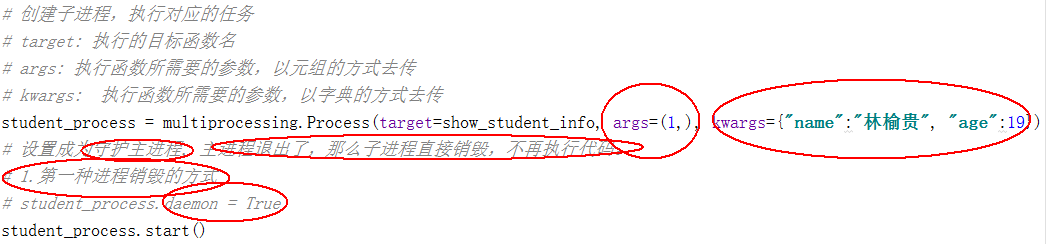
**# 上锁 lock.acquire()  
# 任务干完了，可以释放锁了 lock.release() at least use twice 无序**

****

****

**# 任务1  
def work1():  
 # 查看当前任务由那个进程执行  
 print("work1中执行的进程为:", multiprocessing.current\_process())  
 # 获取当前进程对应编号  
 print("work1中进程的编号为:", multiprocessing.current\_process().pid, os.getpid())  
 # 获取当前进程中的父进程的编号  
 print("work1中子进程的父进程的编号为:", os.getppid())  
 # os.kill() 根据指定的进程的编号杀死指定的进程**

**# 在主进程创建子进程  
# group： 进程组，一般不用，就是使用目前只能None  
# target： 进程执行的目标函数  
sub\_process = multiprocessing.Process(target=work1)  
# 启动进程  
sub\_process.start()**

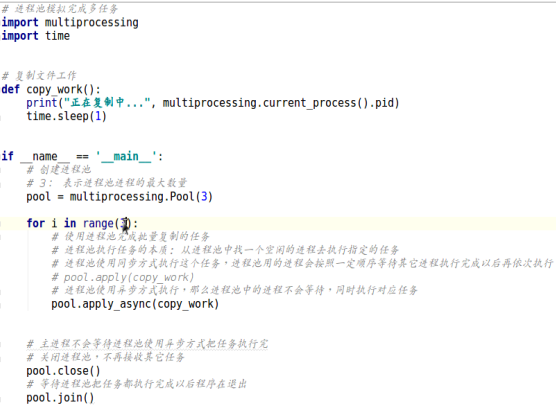
****

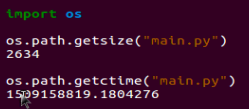
****

进程之间不共享全局变量

**进程池pool 可以完成大批量重复 可以合理利用资源**

****

****

****

**1 下载文件流程**

**1.1判断接收的消息是文件消息还是普通消息**

**1.2如果是文件消息，保存文件的基本信息，比如：消息包编号，文件序号（请求下载文件的时候需要），文件名（写入文件二进制数据的时候需要），对方ip（建立tcp连接的时候需要），文件大小（判断是否文件下载完成）**

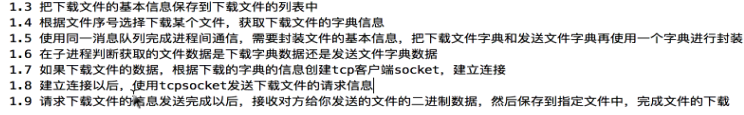
**1.3把下载文件的基本信息保存到下载文件的列表中**

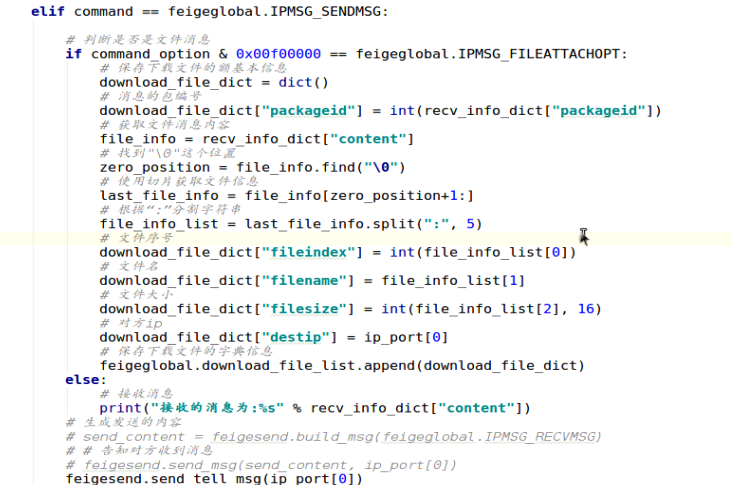
**1.4根据文件序号选择下载某个文件，获取下载文件的字典信息**

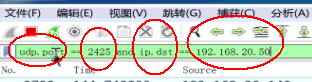
**1.5使用同一消息队列完成进程间通信，需要封装文件的基本信息，把下载文件字典和封装文件字典再使用一个字典进行封装**

**1.6子进程判断获取文件数据**

**1.3建立tcp连接发送请求下载文件的信息，获取对方发送文件的二进制数据，然后把接受的二进制数据写入指定文件，完成文件的下载**

****

**wireshark 抓包工具的使用**

****