**进程的概念 ：**

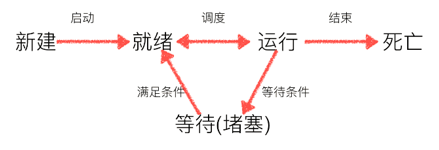
**程序：例如xxx.py这是程序，是一个静态的**

**进程：一个程序运行起来后，代码+用到的资源称之为进程，他是操作系统分配资源的基本单元**

**不仅可以通过线程完成多任务，进程也是可以的**

**进程的状态：**

工作中，任务数往往大于cpu的核数，即一定有一些任务正在执行，而另一些任务在等待cup进行执行，因此导致了有了不同的状态



就绪态：运行的条件都已经慢去，正在等在cpu执行

执行态：cpu正在执行其功能

等待态：等待某些条件满足，例如一个程序sleep了，此时就处于等待态

进程、线程对比

（1、一个程序至少有一个进程，一个进程至少有一个线程

（2、线程的划分尺度小于进程（资源比进程少），使得多线程程序的并发行高

（3、进程在执行过程中拥有独立的内存单元，而**多个线程共享进（进程所分配的资源）内存，从而极大地提高了程序的运行效率**



**创建进程-multiprocessing模块**

1. **由于python是跨平台的，自然应该提供一个跨平台的多进程支持。Multiprocessing模块就是跨平台版本的多进程模块**
2. **Multiprocessing模块提供了一个Process类来表示一个进程对象，创建子进程时，只需要传入一个执行函数和函数的参数，创建一个Process实例，用start()方法启动，join()方法可以等待子进程结束后在继续往下运行，通常用于进程间的同步**

**Process语法结构**

**Process**([group[,target[,name[,args[,kwargs]]]]])

**target**:如果传递了函数的引用，可以任务这个子进程就执行这里的代码

**args**:给target指定的函数传递的参数，以数组的方式传递

**kwargs**:给target指定的函数传递命名函数

**name**:给进程设定一个名字，可以不设定

**group**:指定进程组，大多数情况下用不到

**Process创建的实例对象的常用方法：**

**start**():启动子进程实例（创建子进程）

**is\_alive**():判断主进程子进程是否还在活着

**join([timeout])**:是否等待子进程执行结束，或等待多少秒

**terminat**e():不管任务是否完成，立即终止子进程

**Process创建的实例对象的常用属性：**

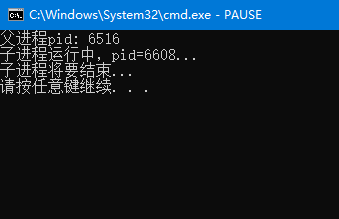
**name**:当前进程的别名，默认为Process-N,N为从1开始递增的整数

**pid**:当前进程的pid(进程号)

**实例2- 打印进程pid**

**import** multiprocessing

**import** os



**import** time

**def** run\_proc**():**

"""子进程要执行的代码"""

# os.getpid获取当前进程的进程号

**print(**'子进程运行中，pid=%d...' **%** os**.**getpid**())**

**print(**'子进程将要结束...'**)**

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

# os.getpid获取当前进程的进程号

**print(**'父进程pid: %d' **%** os**.**getpid**())**

p **=** multiprocessing**.**Process**(**target**=**run\_proc**)**

p**.**start**()**

**进程间不共享全局变量**

**多进程中，同一个变量，各自有一份拷贝存在于每个进程中，互不影响**

**from** multiprocessing **import** Process

**import** os

**import** time

nums **=** **[**11**,** 22**]**

**def** work1**():**

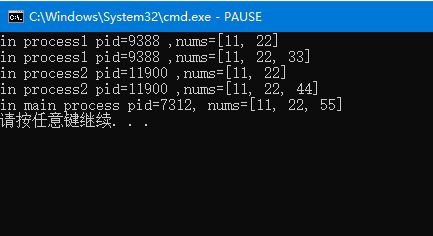
"""子进程要执行的代码"""

**print(**"in process1 pid=%d ,nums=%s" **%** **(**os**.**getpid**(),** nums**))**

nums**.**append**(**33**)**

time**.**sleep**(**1**)**

**print(**"in process1 pid=%d ,nums=%s" **%** **(**os**.**getpid**(),** nums**))**



**def** work2**():**

"""子进程要执行的代码"""

**print(**"in process2 pid=%d ,nums=%s" **%** **(**os**.**getpid**(),** nums**))**

nums**.**append**(**44**)**

time**.**sleep**(**1**)**

**print(**"in process2 pid=%d ,nums=%s" **%** **(**os**.**getpid**(),** nums**))**

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

p1 **=** Process**(**target**=**work1**)**

p1**.**start**()**

p1**.**join**()**

p2 **=** Process**(**target**=**work2**)**

p2**.**start**()**

p2**.**join**()**

nums**.**append**(**55**)**

**print(**"in main process pid=%d, nums=%s" **%** **(**os**.**getpid**(),** nums**))**

进程间通信：

Process之间有时需要通信，可以使用multiprocessing模块的Queue实现多进程之间的数据传递，Queue本身是一个消息队列程序

初始化Queue()对象时（例如：q=Queue()）,若括号中没有指定最大可接收的消息数量，或数量为负值，那么就代表可接受的消息数量没有上限（直到内存的尽头）；

* **Process之间有时需要通信，可以使用multiprocessing模块的Queue实现多进程之间的数据传递，Queue本身是一个消息队列程序**
* **初始化Queue()对象时（例如：q=Queue()），若括号中没有指定最大可接收的消息数量，或数量为负值，那么就代表可接受的消息数量没有上限（直到内存的尽头）；**
* **Queue.qsize()：返回当前队列包含的消息数量；**
* **Queue.empty()：如果队列为空，返回True，反之False ；**
* **Queue.full()：如果队列满了，返回True,反之False；**
* **Queue.get([block[, timeout]])：获取队列中的一条消息，然后将其从列队中移除，block默认值为True；**
* **Queue.put(item,[block[, timeout]])：将item消息写入队列，block默认值为True；**

**from** multiprocessing **import** Process**,** Queue

**import** time

# 写数据进程执行的代码:

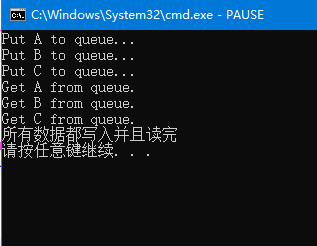
**def** write**(**q**):**

**for** value **in** **[**'A'**,** 'B'**,** 'C'**]:**

**print(**'Put %s to queue...' **%** value**)**

q**.**put**(**value**)**

time**.**sleep**(**1**)**



# 读数据进程执行的代码:

**def** read**(**q**):**

**while** **True:**

**if** **not** q**.**empty**():**

value **=** q**.**get**(True)**

**print(**'Get %s from queue.' **%** value**)**

time**.**sleep**(**1**)**

**else:**

**break**

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

q **=** Queue**()** # 父进程创建Queue，并传给各个子进程：

pw **=** Process**(**target**=**write**,** args**=(**q**,))**

pr **=** Process**(**target**=**read**,** args**=(**q**,))**

pw**.**start**()** # 启动子进程pw，写入:

pw**.**join**()**  # 等待pw结束:

pr**.**start**()** # 启动子进程pr，读取:

pr**.**join**()**

**print(**'所有数据都写入并且读完'**)**

**进程池**

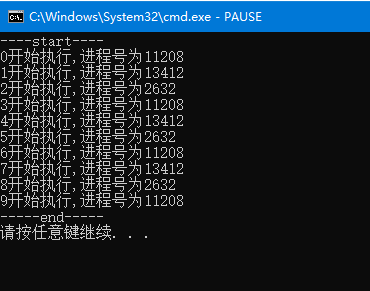
* **当需要创建的子进程数量不多时，可以直接利用multiprocessing中的Process动态生成多个进程，但如果是上百甚至上千个目标，手动的去创建进程的工作量巨大，此时就可以用到multiprocessing模块提供的Pool方法。**
* **初始化Pool时，可以指定一个最大进程数，当有新的请求提交到Pool中时，如果池还没有满，那么就会创建一个新的进程用来执行该请求；但如果池中的进程数已经达到指定的最大值，那么该请求就会等待，直到池中有进程结束，才会用之前的进程来执行新的任务，**

**multiprocessing.Pool常用函数解析**

* **apply\_async(func[, args[, kwds]]) ：使用非阻塞方式调用func（并行执行，堵塞方式必须等待上一个进程退出才能执行下一个进程），args为传递给func的参数列表，kwds为传递给func的关键字参数列表；**
* **close()：关闭Pool，使其不再接受新的任务；**
* **terminate()：不管任务是否完成，立即终止；**
* **join()：主进程阻塞，等待子进程的退出， 必须在close或terminate之后使用；**

**from** multiprocessing **import** Pool

**import** os**,** time



**def** worker**(**msg**):**

**print(**"%s开始执行,进程号为%d" **%** **(**msg**,**os**.**getpid**()))**

time**.**sleep**(**1**)**

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

po **=** Pool**(3)** # 定义一个进程池，最大进程数3

**for** i **in** range**(**10**):**

# Pool().apply\_async(要调用的目标,(传递给目标的参数元祖,))

# 每次循环将会用空闲出来的子进程去调用目标

po**.**apply\_async**(**worker**,(**i**,))**

**print(**"----start----"**)**

po**.**close**()** # 关闭进程池

，关闭后po不再接收新的请求

po**.**join**()** # 等待po中所有子进程执行完成，必须放在close语句之后

**print(**"-----end-----"**)**