## 多任务编程

并行运行这些相互独立的子任务,大幅度地提升整个任务的效率。就是多任务编程的目的。   
 多任务编程和多线程编程是实现多任务编程的重要手段

## 进程

计算机程序:是磁盘中可执行的，二进制（或其它类型）的数据。它们只有在被读取到内存中，被操作系统调用的时候才开始它们的生命期。

进程(重量级进程):是程序的一次执行。每个进程都有自己的地址空间，内存，数据栈以及其它记录其运行轨迹的辅助数据 （在linux系统下记录在PCB当中）。操作系统管理在其上运行的所有进程，并为这些进程公平地分配时间。在进程中存在父子进程的关系。init进程是系统的起始进程。

### 进程和程序的区别

`程序是静态的，它是一些保存在磁盘上的指令的有序集合，没有任何执行的概念.

进程是一个动态的概念，它是程序执行的过程，包括创建、调度和消亡进程是程序执行和资源管理的最小单位。

### 进程分类

交互进程：该类进程是由shell控制和运行的。交互进程既可以在前台运行，也可以在后台运行。

批处理进程：该类进程不属于某个终端，它被提交到一个队列中以便顺序执行。

守护进程：该类进程在后台运行。它一般在Linux启动时开始执行，系统关闭时才结束。

### 进程状态

运行态：此时进程或者正在运行，或者准备运行。

等待态：此时进程在等待一个事件的发生或某种系统资源。分为可中断和不可中断

停止态：此时进程被中止。

死亡态：这是一个已终止的进程，但还在进程向量数组中占有结构空间。

### 进程状体标识

D:不可终端的静止

R:正在执行中

S:阻塞状态

T:暂停执行

Z:僵尸状态

<:高优先级别的进程

N:低优先级的进程

L:有内存分页并所在内存中

PID:唯一的标示一个进程

### linux下常用进程命令

ps:查看系统中的进程

top:动态显示系统中的进程

nice:按用户指定的优先级运行进程

renice:改变正在运行进程的优先级

kill:向进程发送信号

## 线程

线程（轻量级进程）:跟进程有些相似，不同的是，所有的线程运行在同一个进程中，共享相同的运行环境。

由于进程的地址空间是私有的，因此在进程间上下文切换时，系统开销比较大。为了提高系统的性能，许多操作系统规范里引入了轻量级进程的概念，被称为线程。在同一个进程中创建的线程共享该进程的地址空间。在Linux中线程和进程都参与统一的调度。

一个进程中的多个线程既共享这个进程的资源，同时又有自己的私有资源：

## 创建进程

### 使用OS模块创建进程

因为Python的os模块包含普遍的操作系统功能。如果你希望你的程序能够与平台无关的话，这个模块是尤为重要的。因为这个模块生成的一些操作都是和底层系统相关。在这里以linux操作系统为例学习下如何使用OS模块来创建新的进程。

在os模块中很多操作都可以产生新的进程:

1. 新建进程用os.fork函数。但它只在POSIX系统上可用，在windows版的python中，os模块没有定义os.fork函数。相反，windows程序员用多线程编程技术来完成并发任务。

import time,os

pid = os.fork()

if pid < 0:

print "fork error"

elif pid == 0:

while True:

time.sleep(1)

print "Son Process...."

else:

while True :

time.sleep(1)

print "Parent Process..."

2. 用os.system 和 os.exec函数族来执行系统命令和其它程序。os.system使用shell来执行系统命令，然后在命令结束之后把控制权返回给原始进程；os.exec函数族在执行完命令后不将控制权返回给调用进程。它会接管python进程，pid不变。这两个函数支持unix和windows平台。

os.system(‘ls -l’)

这有点类似与exec（）函数，只是这里是shell语句，而exec函数中是Python语句。

3. os.popen()函数可执行命令，并获得命令的stdout流。函数要取得两个参数，一个是要执行的命令，另一个是调用函数所用的模式，如“r"只读模式。os.popen2()函数执行命令，并获得命令的stdout流和stdin流。

p = os.popen(‘ls -l’,’r’)

p = read()

### subprocess————创建附加进程

subprocess模块提供了一种一致的方法来创建和处理附加进程。与标准库中的其他模块相比，它提供了一个更高级的借口，用以替换os模块中的一系列函数。

import subprocess

subprocess.call([‘ls’,’-l’])

命令行参数作为一个字符串列表传入，这样就无需对引号或者其他可能由shell解释的特殊字符转义。

import subporcess.call([‘echo $HOME’],shell = True)

使用一个中间shell意味着在运行命令之前会优先处理命令串中的变量。

使用check\_call函数类似与call，只不过如果发生了错误会产生CalledProcessError异常。还可以使用check\_output来捕获输出结果。

### multiprocessing————更方便的管理进程

要创建第二个进程，最简单的方法是用一个目标函数实例化一个Precess对象，并调用start()让它开始工作。

## 僵尸进程

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：一个进程创建子进程，如果子进程退出，而父进程并没有获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。僵尸进程会浪费一定的系统资源。

如何避免僵尸的产生：

父亲先死，儿子活着，此时儿子寄养在init下，init是它的养父。

儿子先死，父亲活着，但是父亲对子进程退出信号做了相应处理，收尸，就不会产生僵尸。

### 处理僵尸进程方法

1. 使用os.wait()或者os.waitpid()函数。   
2. 创建二级子进程然后让一级子进程用sys.exit()退出，让二级子进程成为孤儿。   
3. 使用信号处理的方式忽略子进程退出信号。

## 守护进程

守护进程，也就是通常所说的Daemon进程，是系统中的后台服务进程。它是一个生存期较长的进程，通常独立于控制终端并且周期性的执行某种任务或等待处理某些发生的事件守护进程常常在系统启动时开始运行，在系统关闭时终止。

Linux系统有很多守护进程，大多数服务都是用守护进程实现的，在Linux中，每一个系统与用户进行交流的界面称为终端。从该终端开始运行的进程都会依附于这个终端，这个终端称为这些进程的控制终端。当控制终端被关闭时，相应的进程都会被自动关闭。

守护进程能够突破这种限制，它从开始运行，直到整个系统关闭才会退出。如果想让某个进程不会因为用户或终端的变化而受到影响，就必须把这个进程变成一个守护进程。

### 实现守护进程

实际应用中我们可能需要自己实现具有一定功能的守护进程。可以利用multiprocessing模块较容易的实现一个守护进程。

进程:

1.就绪状态:唤醒状态

2.运行态:占有cpu

3.休眠态(等待态):可中断休眠态和不可中断休眠

4.停止态:

ps -aux 命令:

user: 进程的创建者

PID: 进程ID号,每个进程ID号唯一

%CPU 占用CPU

$MEN 占用虚拟内存

TTY 类型

STAT 进程状态

START 创建时间

TIME 占用CPU时间

COMMAND 使用哪个命令创建的

ps -ajx 命令:

PPID 父进程的ID

PID 进程ID

PGID 进程组ID

SID 会话组ID

top 命令:

动态显示系统当前进程的信息

NI 优先级

19到-20 -20的优先级高 小于0的进程,需要root权限

STAT

状态补充:

s 会话组组长

l 有子进程

+ 前台进程

nice -n 程序名 以优先级n 运行程序

renice -n 程序ID 将程序优先级改为 n

kill -9 程序ID 杀死进程 kill为向进程传递一个信号

可以用kill -l 来查询kill的功能

fork()函数

功能:创建一个进程 无参数

返回值:

创建进程失败,返回一个负数

创建进程成功,子进程会原样复制父进程的代码,同时,在父进程中 os.fork的返回值为子进程的PID, 子进程的返回值为0

os.getpid():

参数一:-1,任何一个子进程退出,进行接收,0 进程组中任意一个进程结束,进行接收, 大于0的任何数, 接收特定PID的进程的退出

参数二:os.WNOHANG:判断子进程是否退出.如果退出,则进行处理,如果没有退出,则不进行阻塞,同时执行.

sys.exit()函数:

退出进程,括号中可以写一个值,表示退出状态