**Python 3-1**

**forking工作原理**

**• fork(分岔)在Linux系统中使用非常广泛**

**• 当某一命令执行时,父进程(当前进程)fork出一个子进程**

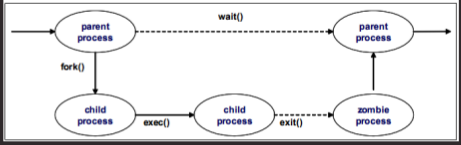
**•父进程将自身资源拷贝一份,命令在子进程中运行时,就具有和父进程完全一样的运行环境**

**进程的生命周期**

**• 父进程fork出子进程并挂起**

**• 子进程运行完毕后,释放大部分资源并通知父进程,这个时候,子进程被称作僵尸进程**

**• 父进程获知子进程结束,子进程所有资源释放**

****

**僵尸进程**

**• 僵尸进程没有任何可执行代码,也不能被调度**

**• 如果系统中存在过多的僵尸进程,将因为没有可用的进程号而导致系统不能产生新的进程**

**• 对于系统管理员来说,可以试图杀死其父进程或重启系统来消除僵尸进程**

**forking编程基本思路**

**• 需要使用os模块**

**• os.fork()函数实现forking功能**

**• python中,绝大多数的函数只返回一次,os.fork将返回两次**

**• 对fork()的调用,针对父进程返回子进程的PID;对于子进程,返回PID0**

**• 因为所有的父子进程拥有相同的资源,所以在编写程序时要避免资源冲突**

**网络编程思路如下:**

**pid = os.fork() #实现forking**

**if pid: #在父进程中关闭子进程连接**

**close\_child\_conn #接着处理其他的连接请求**

**handle\_more\_conn**

**else: #子进程关闭父进程连接,响应当**

**close\_parent\_conn #前的用户连接**

**process\_this\_conn**

**使用轮询解决zombie问题**

**• 父进程通过os.wait()来得到子进程是否终止的信息**

**• 在子进程终止和父进程调用wait()之间的这段时间,子进程被称为zombie(僵尸)进程**

**• 如果子进程还没有终止,父进程先退出了,那么子进程会持续工作。系统自动将子进程的父进程设置为init进程,init将来负责清理僵尸进程**

**• python可以使用waitpid()来处理子进程**

**• waitpid()接受两个参数,第一个参数设置为-1,表示与wait()函数相同;第二参数如果设置为0表示挂起父进程,直到子程序退出,设置为1表示不挂起父进程**

**• waitpid()的返回值:如果子进程尚未结束则返回0,否则返回子进程的PID**

**import** time  
**import** os  
**import** tarfile  
**import** hashlib  
**import** pickle  
  
  
**def check\_md5(**fname**):** m **=** hashlib.md5**()** # 创建md5对象  
 **with** open**(**fname, **'rb') as** fobj**:  
 while True:** data **=** fobj.read**(**4096**)  
 if not** data**:  
 break** m.update**(**data**)** # 更新md5对象  
 **return** m.hexdigest**()** # 返回md5对象  
  
  
**def full\_backup(**src\_dir, dst\_dir, md5file**):** # 完全备份  
 fname **=** os.path.basename**(**src\_dir.rstrip**('/'))** # 去除目录最后的根号(rstrip())，取最后的一个单个文件(basename())  
 fname **= '%s\_full\_%s.tar.gz' % (**fname, time.strftime**('%Y%m%d'))** # 备份后包的名字:文件名\_full\_日期.tar.gz  
 fname **=** os.path.join**(**dst\_dir, fname**)** # 合并路径，形成绝对路径  
 md5dict **= {}** # 定义一个字典，用于放文件名和对应的md5值  
  
 tar **=** tarfile.open**(**fname, **'w:gz')** ###对文件进行打包  
 tar.add**(**src\_dir**)** # 添加要打包的目录  
 tar.close**()  
  
 for** path, folders, files **in** os.walk**(**src\_dir**):** # walk(),遍历目录树，自顶向下或自底向上生成目录树下的文件名  
 **for** each\_file **in** files**:** key **=** os.path.join**(**path, each\_file**)** # 合并路径  
 md5dict**[**key**] =** check\_md5**(**key**)** # 添加到字典  
  
 **with** open**(**md5file, **'wb') as** fobj**:** pickle.dump**(**md5dict, fobj**)** # 将字典的内容添加到文件中  
  
  
**def incr\_backup(**src\_dir, dst\_dir, md5file**):** # 增量备份  
 fname **=** os.path.basename**(**src\_dir.rstrip**('/'))** fname **= '%s\_incr\_%s.tar.gz' % (**fname, time.strftime**('%Y%m%d'))** fname **=** os.path.join**(**dst\_dir, fname**)** md5dict **= {}  
  
 with** open**(**md5file, **'rb') as** fobj**:** oldmd5 **=** pickle.load**(**fobj**)** # 将旧的md5值从文件内取出来  
  
 **for** path, folders, files **in** os.walk**(**src\_dir**):  
 for** each\_file **in** files**:** key **=** os.path.join**(**path, each\_file**)** md5dict**[**key**] =** check\_md5**(**key**)** # 向字典内添加新的md5值  
  
 **with** open**(**md5file, **'wb') as** fobj**:** pickle.dump**(**md5dict, fobj**)** # 将新的那个md5字典内容写入文件内  
  
 tar **=** tarfile.open**(**fname, **'w:gz')  
 for** key **in** md5dict**:** # 循环遍历，将旧的字典内容与新的字典内容进行比对，然后将有改变的文件进行打包  
 **if** oldmd5.get**(**key**) !=** md5dict**[**key**]:** tar.add**(**key**)** tar.close**()  
  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** # mkdir /tmp/backup;cp -r /etc/security /tmp/  
 src\_dir **= '/tmp/security'** dst\_dir **= '/tmp/backup'** md5file **= '/tmp/backup/md5.data'  
 if** time.strftime**('%a') == 'Mon':** # 星期一,完全备份，否则增量备份  
 full\_backup**(**src\_dir, dst\_dir, md5file**)  
 else:** incr\_backup**(**src\_dir, dst\_dir, md5file**)**