.什么是 fork
　fork 分叉的意思，为什么叫分叉，因为一个进程在调用 fork()时，就产生了另一个进程，于是进程就“分叉”了
.什么是进程？
　进程是一个实体，每一个进程有自己的地址空间，包括文本区域，数据区域，堆栈等~
　进程是一个执行中的程序。程序是一个没有生命的实体，只有赋予程序生命的时候，程序才能成为一个活动的实体，我们称之为进程
　简单的说 进程就是正在运行的程序的实例
.什么是 PID？
　PID 又叫进程号，是操作系统内核用于唯一标识进程的一个数值
.为什么要用 fork
　进程为什么要分叉出其他进程？一个进程不好么？
　假如现在需要盖房子，一个人，先打地基，然后再搬砖活水泥砌墙...这样好，还是有多个人，有人去打地基，有人去搬砖，有人活水泥这样好？
　进程也是这样，fork()的作用就是创建出新的进程，就像招到新的工人来。

如何对程序，进程和线程进行一种抽象类似的近似；
OS 操作系统：　　公司
CPU 计算资源，I/O 输入输出：　　公司的某些资源
进程（进程就是程序的一次执行的抽象）　公司的部门
线程：　　部门的人，干活的实体

***进程的标识符 pID, 进程名 pName;
在 Windows 下面：使用任务管理器来查看进程信息；
在 Linux 下面：快照 ps aux;

几类特殊的进程：
孤儿进程：父进程终止了，子进程还在；
僵尸进程：子进程终止了，父进程还在，父进程没有为子进程善后；

.fork 生命周期
1.父进程 fork 出子进程并挂起
　一个进程一旦使用 fork(),首先，系统先让父进程和子进程使用同一文本区域(代码块)，因为它们的程序还是相同的，对于数据区域和堆栈，系统则复制一份给新的进程，这样父进程的所有数据都可以留给子进程
2.子进程运行
　一旦子进程运行时，虽然它继承了父进程的一切数据，但实际上数据却已经分开，相互之间不再有影响了，也就是说，它们之间不再共享任何数据了
3.子进程运行结束
　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
4.父进程知悉子进程结束，子进程所有资源释放

forking 编程
　　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 PID;对于子进程返回 PID0

```
#!/usr/bin/env python
#coding: utf8

import os

print("starting....")
os.fork()
print("hello world!")
```

#会打印两行 helloworld，因为 fork 创建子进程，该子进程具有与父进程相同的运行环境
#因为 print“hello world”，在 fork 下面，所以 父进程会运行一次，子进程也运行一次

```
#!/usr/bin/env python
#coding: utf8

import os

print "starting...."
import os
pid = os.fork()
print(pid)
if pid:
    print("hello from parent.")
else:
    print("hello from child")
```

fork 返回两个值，针对父进程返回子进程 pid，针对子进程返回 0，
#当第一次返回时 pid 为非 0 值，则打印 hello from parent
#当第二次返回时 pid 为 0,则打印 hello from child

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

练习

```
 import os
 import time

 pid = os.fork()
 if pid:
       print( ‘In parent’ )
       time.sleep(10)
       print(“parent exit”)
 else:
       for i in range(0,5):
            print(time.strftime( “%m/%d/%Y %H:%M:%S”))
            time.sleep(1)
       print( “child exit”)
```

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

```
#!/usr/bin/env python
#coding:utf8

import os
import time

pid = os. fork()
if pid:
    print( ‘in parent.sleeping.... ‘)
    time.sleep(30)
    print( ‘parent done. ‘)
```

```
else:
    print( ‘in child.sleeping... ‘)
    time.sleep(10)
    print( ‘child.exit. ‘)
```

再开启一个终端
watch -n 1 ps a 观测效果。查看 stat
工作过程：
父进程，子进程分别运行，父进程 sleep 30 s，子进程 sleep 10s，父进程没有处理子进程的代码，子进程进入 zombie 状态，父进程 sleep 后，init 进程回收父进程资源，父进程退出，子进程仍在，init 接管子进程，并回收子进程资源
Init 进程是内核启动的第一个用户级*进程*，是 Linux 运行最基本的程序之一
使用轮询解决 zombie 问题
        python 可以使用 waitpid()来处理子进程
        waitid()接受两个参数，第一个参数设置为-1，表示与 wait()函数相同;第二个参数如果设置为 0 表示挂起父进程，直到子进程退出，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
1、挂起的情况

```
#!/usr/bin/env python
#coding:utf8

import os
import time

pid = os. fork()

if pid:
    print( ‘in parent.sleepin.... ‘)
    print(os.waitpid(-1,0))
    time.sleep(5)
    print( ‘parent done. ‘)
else:
    print( ‘in child.sleeping... ‘)
    time.sleep(10)
    print( ‘child.done. ‘)
```

2、不挂起的情况

```
#!/usr/bin/env python
#coding:utf8

import os
import time

pid = os. fork()

if pid:
    print( ‘in parent.sleepin.... ‘)
    print(os.waitpid(-1,1))
   # print(os.waitpid(-1,os.WNOHANG))   WNOHANG 是 os 常量，值为 1，
    time.sleep(5)
    print( ‘parent done. ‘)
else:
    print( ‘in child.sleeping... ‘)
    time.sleep(10)
    print( ‘child.done. ‘)
```

forking 服务器
        在网络服务中，forking 被广泛使用(apache 的工作方式)

如果服务器需要同时响应多个客户端，那么 forking 是解决问题最常用的方法之一
父进程负责接受客户端的连接请求
子进程负责处理客户端的请求

利用 forking 创建 tcp 时间戳服务器

编写 tcp 服务器

1、服务器监听在 0.0.0.0 的端口上
2、收到客户端数据后，将其加上时间戳后回送给客户端
3、如果客户端发过来的字符全是空白字符，则终止与客户端的连接
4、服务器能够同时处理多个客户端的请求
5、程序通过 forking 来实现

```
#!/usr/bin/env python
#coding: utf-8

import os
import time
import socket

host =  ' '
port = 21345
addr = (host,port)

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
s.bind(addr)
s.listen(1)

while True:
    try:
        while True:
            result = os.waitpid(-1,os.WNOHANG)
            if result[0] == 0:
                break
    except OSError:
        pass
    cli_sock,cli_addr = s.accept()
    pid = os.fork()
    if pid:
        cli_sock.close()
    else:
        s.close()
        while True:
            data = cli_sock.recv(4096)
            if not data.strip():
                cli_sock.close()
                sys.exit()
            cli_sock.send("[%s] %s" % (time.ctime(),data))
        cli_sock.close()
s.close()
```

多线程

多线程工作原理

多线程的动机

在多线程(MT)编程出现之前，电脑程序的运行由一个执行序列组成，执行序列按顺序在主机的中央处理器(CPU)中运行

无论是任务本身要求顺序执行还是整个程序是由多个子任务组成，程序都是按这种方式执行的

即使子任务相互独立，互相无关(即，一个子任务的结果不影响其他子任务的结果)时也是这样

如果并行运行这些相互独立的子任务可以大幅度地提升整个任务的效率

多线程任务的工作特点

它们本质上就是异步的，需要有多个并发事务

各个事务的运行顺序可以是不确定的，随机的，不可预测的

这样的编程任务可以被分成多个执行流，每个流都有一个要完成的目标

根据应用的不同，这些子任务可能都要计算出一个中间结果，用于合并得到最后的结果

什么是进程

计算机程序只不过是磁盘中可执行的、二进制(或其它类型)的数据

进程(有时被称为重量级进程)是程序的一次执行

每个进程都有自己的地址空间，内存以及其它记录其运行轨迹的辅助数据

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空间

什么是线程

线程(有时被称为轻量级进程)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

线程有开始，顺序执行和结束三部分

线程的运行可能被抢占(中断)，或暂时的被挂起(也叫睡眠)，让其它的线程运行，这叫做让步

一个进程中的各个线程之间共享同一片数据空间，所以线程之间可以比进程之间更方便的共享数据以及相互通讯

线程一般都是并发执行的，正是由于这种并行和数据共享机制使得多个任务的合作变为可能

需要注意的是，在单 CPU 的系统中，真正的并发十不可能的，每个线程会被安排成每次只运行一小会，然后就把 CPU 让出来，让其他的线程去运行

多线程编程

多线程相关模块

thread 和 threading 模块允许程序员创建和管理线程

thread 模块提供了基本的线程和锁的支持，而 threading 提供了更高级别/功能更强的线程管理功能

推荐使用更高级别的 threading 模块

只建议那些有经验的专家在想访问线程的底层结构的时候才使用 thread 模块

传递函数给 Thread 类

多线程编程有多种方法，传递函数给 threading 模块的 Thread 类是加烧得第一种方法

Thread 对象使用 start()方法开始线程的执行，使用 join()方法挂起程序，直到线程结束