- Python中同时进行两个任务
  - 基本做法
    - 进程分支
    - 线程派生
  - 对比
    - 功能上
      - 均依赖于操作系统的底层服务来并行地执行Python代码
    - 操作步骤上
      - 接口、跨平台移植性和通信上有大差异

多进程

要让Python程序实现多进程（multiprocessing），我们先了解**操作系统**的相关知识。

Unix/Linux操作系统提供了一个`fork()`系统调用，它非常特殊。普通的函数调用，调用一次，返回一次，但是`fork()`调用一次，返回两次，因为操作系统自动把当前进程（称为父进程）复制了一份（称为子进程），然后，分别在父进程和子进程内返回。

子进程永远返回`0`，而**父进程返回子进程的ID**。这样做的理由是，一个父进程可以fork出很多子进程，所以，父进程要记下每个子进程的ID，而子进程只需要调用`getppid()`就可以拿到父进程的ID。

Python的`os`模块封装了常见的系统调用，其中就包括`fork`，可以在Python程序中轻松创建子进程：

```
def getppid()
"""
Return the parent's process id.
"""


def getpid()
'''
Return the current process id.
'''


def fork()
"""
Fork a child process.
Return 0 to child process and PID of child to parent process.
"""
```

实例说明①

```
# 这个东东win跑不起来，去bash跑吧！

import os

def child():
    print('这是子进程哦，pid：', os.getpid())
    os._exit(0)

def parent():
    while True:
        newpid = os.fork() # 从这里开始分出来两个返回（分出两条路，同时走），父进程：pidof(子进程），
子进程：0
        if newpid == 0:
            child()
        else:
            print('这是父进程，pid：', os.getpid(), newpid)
        if input() == 'q' : break

parent()
```

运行结果：

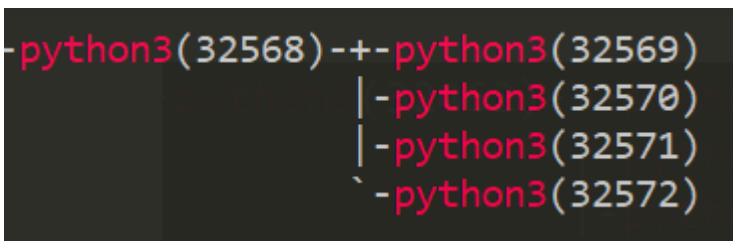
```
root@iZwz9jdpiao7167rzypua0Z:~/demo_projects/PyStudy/fork# python3 fork01.py
这是父进程，pid：  32568 32569
这是子进程哦，pid：  32569

这是父进程，pid：  32568 32570
这是子进程哦，pid：  32570

这是父进程，pid：  32568 32571
这是子进程哦，pid：  32571

这是父进程，pid：  32568 32572
这是子进程哦，pid：  32572
```

为了清晰显示逻辑，在另外一个console【即打开一个新的控制台】上运行Linux命令`pstree -p|grep python3`查看进程树：



分析一下，根据运行结果与`pstree`命令可以知道，父进程的`pid`是`32568`，第一轮循环的时候，经过`fork()`产生了一个子进程，它的`pid`是`32569`。然后后面又走了几次循环，共创建了四个子进程。通过`pstree`看到，他们同属于一个父进程`32568`。