总结进程线程

1. 两者都是多任务编程方式，都能够使用计算机的多核资源
2. 进程的创建删除消耗的计算机资源比线程多
3. 进程空间独立，数据互不干扰，有专门的IPC，线程使用全局变量进行通信
4. 一个进程可以创建多个线程分支，两者之间存在包含关系
5. 多个线程公用进程的资源，在资源操作时往往需要同步互斥
6. 进程线程在系统中都有自己特有的属性，ID，代码段，栈区等资源

使用场景

* 需要创建较多并发，同时任务关联性强，一般用多线程
* 不同的任务模块可能更多使用进程
* 使用进程线程需要考虑数据的处理复杂度，比如进程间通信是否方便，同步互斥是否过于复杂

**要求：**

**进程线程的区别和联系**

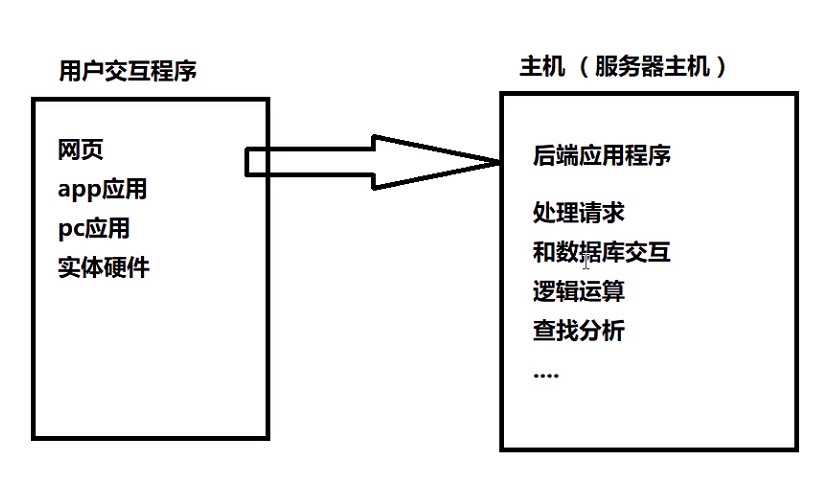
**进程间通信方式知道有哪些，有什么特点**

**同步互斥意义是什么，什么情况下用**

**给一个情形，分析下用进程还是用线程，理由**

**僵尸进程（概念，处理方法），进程状态，GIL（执行效率等）**

服务器模型



硬件服务器：主机 集群

厂商 ： IBM HP 联想 浪潮

软件服务器：编写的服务器端应用程序，在硬件服务器上运行，一般依托于操作系统，给用户提供一套完整的服务。

HTTPserver---》处理HTTP请求

webserver🡪网站的后端应用服务器程序

邮箱服务器—》邮件处理

ftp文件服务器—》文件的上传下载

功能：网络连接 逻辑处理 数据交互 数据传输 协议的实现

结构： c/s 客户端服务器模型

b/s 浏览器服务器模型

服务器目标： 处理速度更快，并发量更高，安全性更强

硬件 ： 更高得多配置，更好的集成分布技术 ，更好的网络优化和网络安全技术。

软件：占用资源更少，运行更稳定，算法更优良，安全性更好，并发性更高，更容易扩展

基础服务端模型

循环模型：循环接收客户端的请求，处理请求。同一时刻只能处理一个请求，处理完毕后在处理下一个。

优点：实现简单，占用资源少

缺点：无法同时处理多个客户端任务

适用情况：处理任务可以短时间完成，不需要建立并发，更适合udp使用

并发模型：能够同时处理多个客户端请求

IO并发：IO多路复用

优点：资源消耗少，IO处理速度快

缺点：不能使用CPU密集型程序

多进程，多线程并发：为每一个客户端创建单独的进程线程，执行请求

优点：每个客户端可以长期占有服务端运行程序，能够使用多核资源，可以处理IO或者CPU运算

缺点：消耗系统资源高

多进程并发模型

使用fork实现多进程并发

1. 创建套接字，绑定，监听
2. 等待接收客户端请求
3. 创建新的进程处理客户端请求
4. 原来进程继续等待接收新的客户端连接
5. 如果客户端退出，则关闭子进程

cookie：

#在父进程中忽略子进程状态改变，子进程退出自动由系统处理

signal.signal(signal.SIGCHLD,signal.SIG\_IGN)

ftp文件服务器

项目功能：

\*服务端和客户端两部分，要求启动那个一个服务端，可以同时处理多个客户端请求

\*功能：1.可以查看服务端文件库中所有的普通文件

2.从客户端可以下载文件库的文件到本地

3.可以将本地文件上传的服务端文件库

4.退出

\*客户端使用print在终端打印简单的命令提示，通过命令提示发起请求

思路：1.技术分析（fork TCP并发）还有细节

2.每一个功能要单独封装，整体功能写在一个类中

3.如何搭建整体架构，完成网络通讯

定义一个类

需要find查找

提供一个下载的方法，

功能分析：

1. 获取文件列表

* 客户端：\* 发送请求

\* 得到回复判断能否获取列表

\* 接收文件名列表打印

\* 服务端：\* 接收请求

\* 判断请求类型

\* 判断能否满足请求，回复信息确认

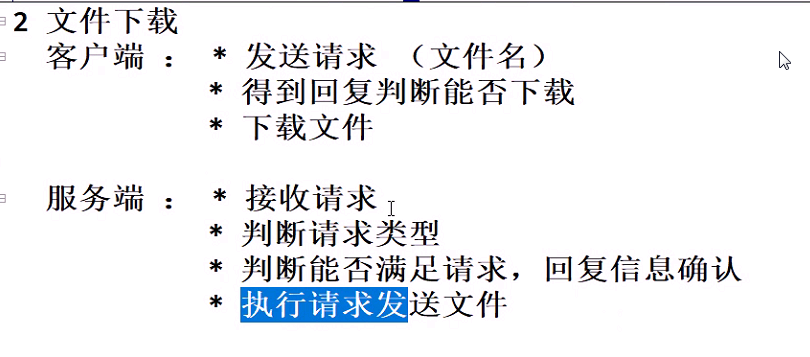
\* 执行请求发送文件列表

cookie：：os.listdir(path)查看文件夹下所有内容

os.path.isfile() 判断是否为普通文件

os.path.isdir() 判断是否为目录

1. 文件下载

* 

3.文件上传

客户端：发送请求，需要上传，（内容有 名字）

接到请求，是否能上传，打开文件 读取，关闭文件、

服务端：接收请求

判断请求类型，

判断文件库有没有这个文件，没有则回复确认

打开新文件名，写入。

否则发送原因