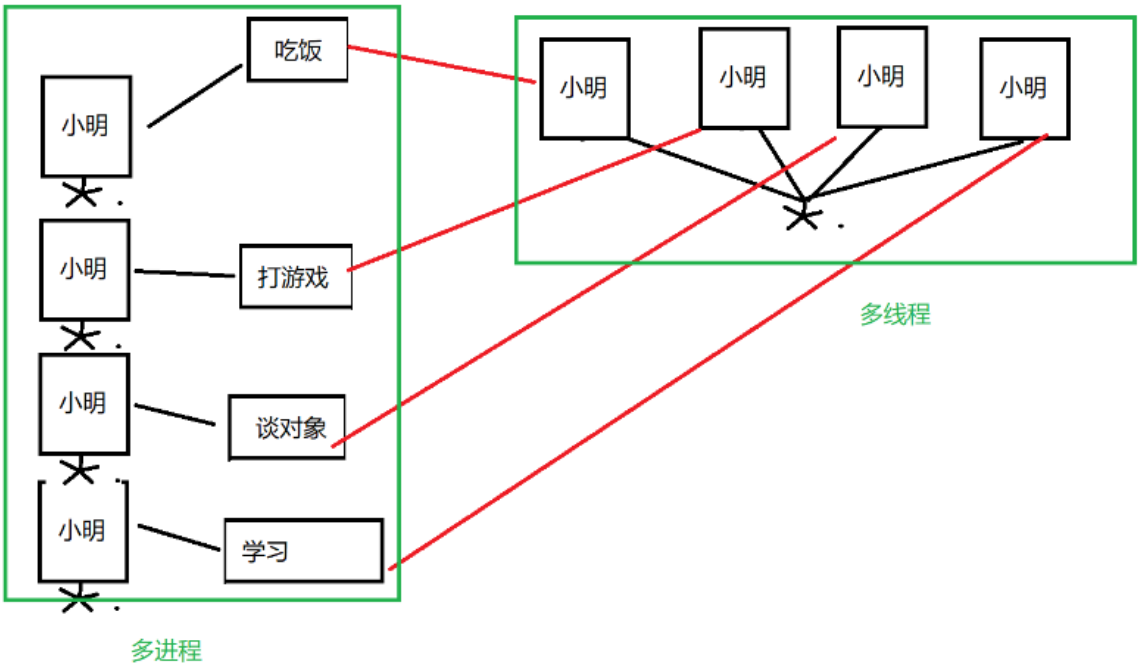
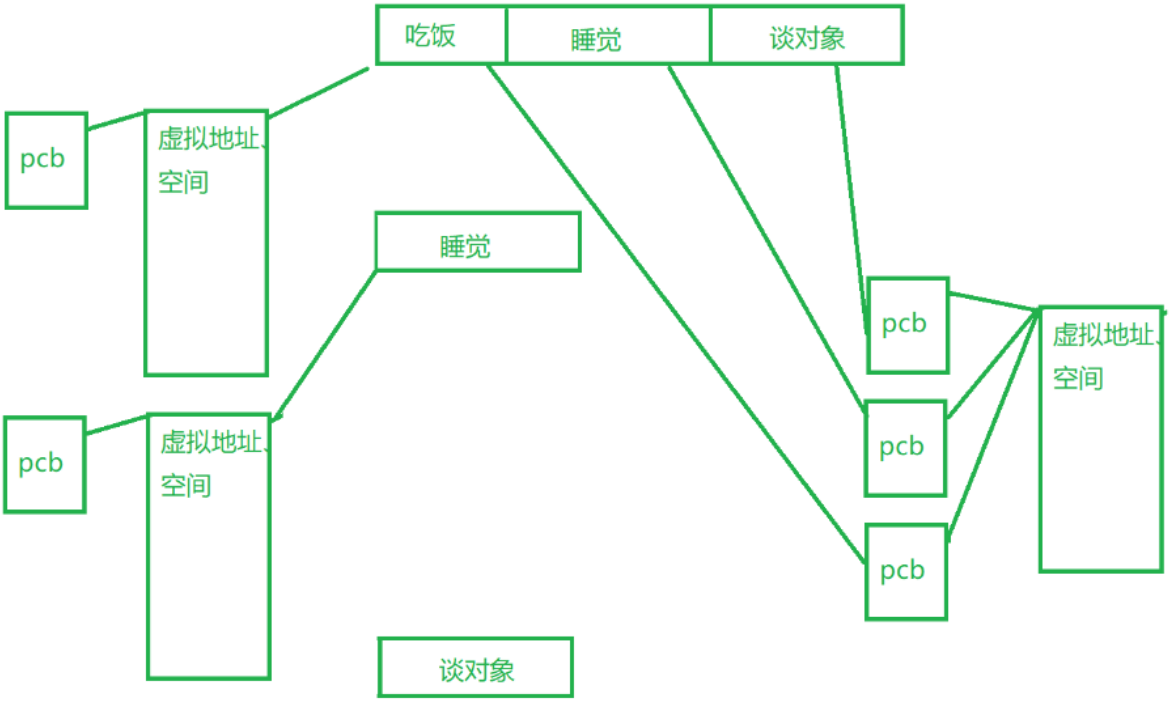
多线程：线程概念/线程控制/线程安全/知识的整合应用

## 线程概念：

多线程其实是实现多任务并发/并行处理的 / 多进程也是完成多任务的

通过多线程/多进程进行多任务处理的不同之处来理解什么是线程




多线程的处理思路：一个运行中的程序中，具有多个执行流，各自完成一个功能模块的实现

linux的操作系统中：认为一个pcd就是一个执行流(pcd是操作系统调用一段程序运行的实体--描述了程序的运行过程)

linux下的线程就是一个pcd

pcd：是一个进程

现在，多个pcd可以共用同一个虚拟地址空间，这些pcd共用了一个运行中程序的资源

linux下的线程就是一个pcd是一个轻量级进程，因为一个运行中程序的多个pcd共用同一份资源

一个运行中的程序就是一个进程，以前我们所说的进程，这时候再来解释，就是具有一个线程的进程，linux下的线程是一个pcb(轻量级进程)，是一个进程中的一条执行流；
这时候再来理解进程的话：进程就是一个线程组；

若以后面试官问到进程/线程
1、进程跟线程是要一起说的(否则单一说会有一种冲突感)
2、进程就是一个运行中的程序，操作系统会创建一个pcb(运行中程序的描述)，并且分配资源，通过pcb来调度运行整个程序
3、线程是一个进程中的执行流，但是linux下实现进程中的执行流的时候，使用了pcb实现
4、因此就说linux下的线程是一个pcb,称作轻量级进程,因为同一个进程中的线程共用进程分配的资源
5、而进程就是所有线程的统称，就是一个线程组，系统在运行程序，分配资源的时候是分配给线程组，分配给整个进程的
进程是资源分配的基本单位，线程是cpu调度的基本单位
进程就像是一个工厂，线程就是工厂里干活的工人

vfork--创建出来的子进程与父进程共用同一个虚拟地址空间，但是父进程会阻塞，因为同时运行会出现栈混乱
线程是一个pcb; 一个进程中若有多个线程，也就意味着有多个pcb,这些pcb共用同一个虚拟地址空间;
它是如何做到同时运行而不会出现栈混乱的情况的呢?
线程之间的独有与共享
**独有**:栈(每个线程一个，就可以避免出现栈混乱了)，寄存器(每一个pcb都是一个执行流)，信号屏蔽字(阻塞自己想阻塞的信号)，errno (每调用一次系统调用都会重置errno)
**共享**:代码段和数据段(虚拟地址空间)，信号的处理方式，I0信息， 工作路径，用户id,组id

多线程与多进程都能进行多任务处理，我们以后写代码的时候就要根据各自的优缺点以及应用场景来选择技术

多线程任务处理相较于多进程的**优点**:

1.线程间的通信更加灵活方便(出了进程间通信方式以外，还可以通过全局变量/函数传参实现通信)

2.线程的创建和销毁成本更低(线程间共享进程等的大部分资源)

3.同一个进程中的线程间调度成本更低(切换页表,数据)

4.线程的执行粒度更加细致

**缺点:**

线程间缺乏访问控制，有些系统调用(exit)或者异常是针对整个进程产生效果

多线程任务处理没有多进程任务处理稳定性高

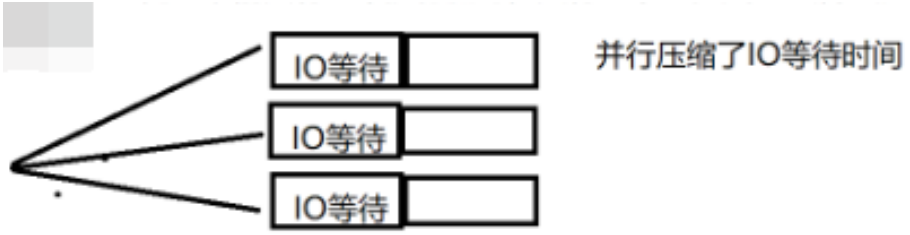
应用场景:多进程应用场景，对主程序的稳定性安全性要求更高的场景，比如shell/网络服务器;

多进程/多线程进行多任务的并发处理的共同优势:

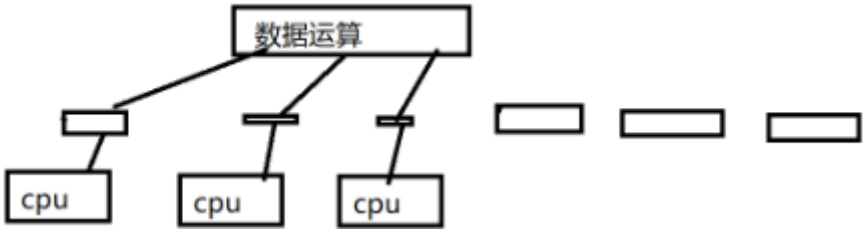
**IO密集型程序:**程序中大部分的工作都是进行IO　　占用cpu极少

可以在一个执行流中发起一个IO;避免了以前只有一个执行流时,只有一个IO完成了之后才能进

行下一个的情况，提高了IO效率




**CPU密集型程序**：程序中的大部分工作都是进行数据运算




在cpu密集型程序中，执行流的创建并不是越多越好，多了反而会提高cpu调度的成本

cpu密集型程序中，线程的创建最好是cpu核心数+1

# 线程控制:线程创建/线程终止/线程等待/线程分离

讲的是对线程所能进行操作接口的学习

实际上linux操作系统并没有给用户提供创建一个线程的系统调用接口，用户无法直接创建线程;

但是大佬们封装了一套线程库，通过库函数可以实现线程控制的各种操作

我们创建线程就是在用户态创建线程，可以实现在内核态创建一个轻量级进程实现调度

**线程创建:**

int pthread_create(pthread_ t *tid, pthread_attr _t  *attr, void* (*thread_ routine) (void *arg),void *arg)

tid:输出型参数，用于向用户返回线程id,是后续线程操作的句柄

attr:用于设置线程属性，通常置NULL

thread_ routine: 线程的入口函数

arg:最终会通过线程入口函数的参数传递给线程的数据

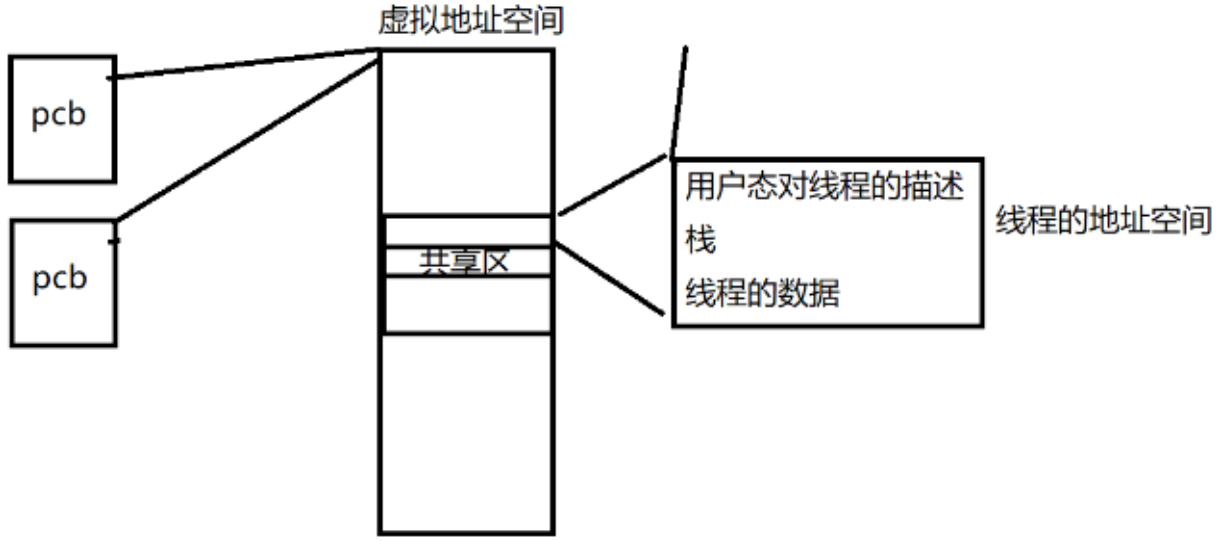
返回值:若成功则返回0， 失败返回一个非0值

使用ps  -L选项可以查看轻量级进程信息

LWP这一项是轻量级进程id,实际上就是pcb中的pid

在外边查看到的进程id实际是pcb中tgid-线程组id　　线程组id等于进程中主线程的轻量级进程id的值

tid是什么，如何成为线程的操作句柄?




创建线程，会在进程的虚拟地址空间中开辟一块相对独立的空间作为线程的地址空间，这个空

间内包含用户态对线程的描述，实现对线程的操作，线程栈...；创建线程成功之后，会将这

块空间的首地址返回给用户;

用户对线程进行操作，就可以通过首地址找到线程的数据，进行进行一系列的操作，

因此返回的线程地址空间的首地址就是线程的操作句柄，也就是pthread_create函数返回的tid

线程的地址空间也是在进程的地址空间之内的一部分

tid:线程空间在虚拟地址空间中的首地址
PID:实际上是线程组id　tgid = mainthread-> pcb-> pid
LWP:实际上是每个线程pcb中的pid　pcb-> pid

pthread_t pthread_ self() :返回调用线程的线程id
pthread_ create创建一个线程的接口应用
pthread_ self()获取线程id的接口应用
线程中的id (tid　pid　lwp)

**线程终止**:退出-一个线程
主动退出:线程入口函数中的return (main函数中的return是 用于退出进程的而不是主线程) ;

void pthread_exit(void *retval); 退出调用线程;　retval :返回一个退出返回值;
被动退出: int pthread_cancel(pthread_t thread);取消- 个线程; thread: 要取消的线程id

在主线程中调用pthread_ exit可以退出主线程，但是不会退出进程
只有所有的线程都退出了，进程才会退出