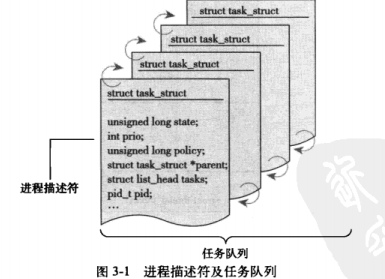
**主要内容：**

* 进程和线程
* 进程的生命周期
* 进程的创建
* 进程的终止

**进程描述符及任务结构**

内核把进程的列表存放在叫做任务队列(task list©)的双向循环链表中。链表中的每一 项都是类型为task\_struct、称为进程描述符(process descriptor)的结构，该结构定义在＜linux/ sched.h＞文件中。进程描述符中包含一个具体进程的所有信息。



### ****1. 进程和线程****

Linux不严格区分进程和线程，线程不过是一种特殊的进程。

进程提供2种虚拟机制：虚拟处理器和虚拟内存

每个进程有独立的虚拟处理器和虚拟内存，

每个线程有独立的虚拟处理器，同一个进程内的线程有可能会共享虚拟内存。

内核中进程的信息主要保存在task\_struct中(include/linux/sched.h)

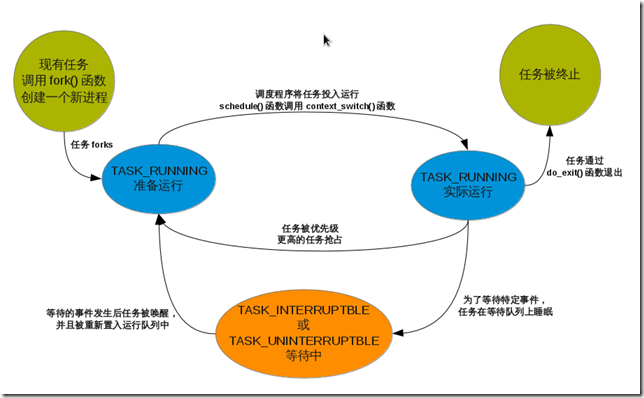
Linux中可以用ps命令查看所有进程的信息：

ps -eo pid,tid,ppid,comm

Linux把所 有的线程都当做进程来实现。线程仅仅被视为一个与其他进程共享某些资源的进程。每个线程都拥有唯一隶属于自己的task\_struct,所以在内核中，它看起来就像是一个普通的进程

### ****2. 进程的生命周期****

进程的各个状态之间的转化构成了进程的整个生命周期。

[](http://images.cnblogs.com/cnblogs_com/wang_yb/201208/201208201741296349.png)

内核经常需要调整某个进程的状态。这时最好使用set\_task\_state(task, state)函数:

set\_task\_state (task, state) ； /\* 将任务 task 的状态设置为 state \*/

set\_current\_state(stete)和 set\_task\_state(current, state)含义是等同的。

每个task\_struct都包含一个指向其父进程tast\_struct、叫做parent的指针，还包含一个称为children的子进程链表。

### ****3. 进程的创建****

fork: 通过拷贝当前进程创建一个子进程

exec: 读取可执行文件，将其载入到内存中运行

创建的流程：

1. 调用dup\_task\_struct()为新进程分配内核栈，task\_struct等，其中的内容与父进程相同。
2. check新进程(进程数目是否超出上限等)
3. 清理新进程的信息(比如PID置0等)，使之与父进程区别开。
4. 新进程状态置为 TASK\_UNINTERRUPTIBLE
5. 更新task\_struct的flags成员。
6. 调用alloc\_pid()为新进程分配一个有效的PID
7. 根据clone()的参数标志，拷贝或共享相应的信息
8. 做一些扫尾工作并返回新进程指针

### ****4. 进程的终止****

和创建进程一样，终结一个进程同样有很多步骤：

子进程上的操作(do\_exit)

1. 设置task\_struct中的标识成员设置为PF\_EXITING
2. 调用del\_timer\_sync()删除内核定时器, 确保没有定时器在排队和运行
3. 调用exit\_mm()释放进程占用的mm\_struct
4. 调用sem\_\_exit()，使进程离开等待IPC信号的队列
5. 调用exit\_files()和exit\_fs()，释放进程占用的文件描述符和文件系统资源
6. 把task\_struct的exit\_code设置为进程的返回值
7. 调用exit\_notify()向父进程发送信号，并把自己的状态设为EXIT\_ZOMBIE
8. 切换到新进程继续执行

子进程进入EXIT\_ZOMBIE之后，虽然永远不会被调度，关联的资源也释放掉了，但是它本身占用的内存还没有释放，  
比如创建时分配的内核栈，task\_struct结构等。这些由父进程来释放。

如果子进程的父进程已经退出了，那么子进程在退出时，exit\_notify()函数会先调用forget\_original\_parent()，然后再调用find\_new\_reaper()来寻找新的父进程。

find\_new\_reaper()函数先在当前线程组中找一个线程作为父亲，如果找不到，就让init做父进程。(init进程是在linux启动时就一直存在的)