**1.内核中进程的信息**

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

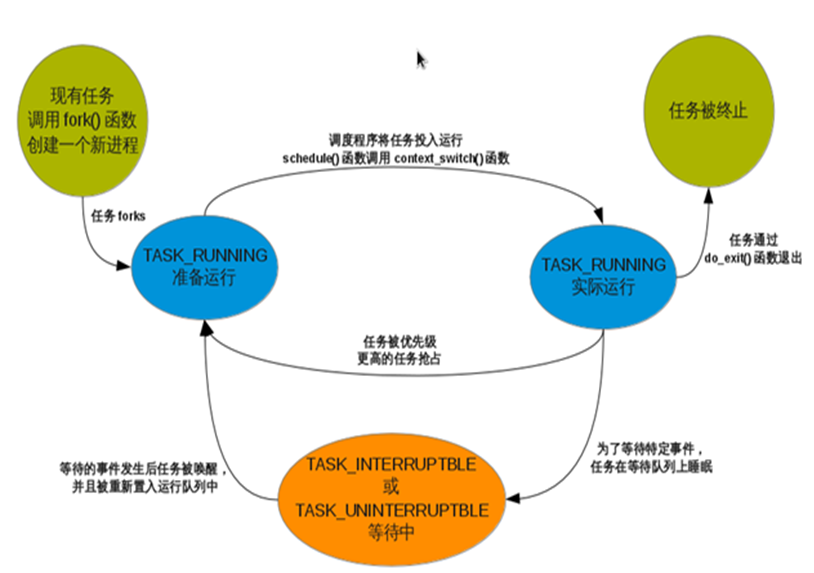
进程标识PID和线程标识TID对于同一个进程或线程来说都是相等的。

Linux中可以用ps命令查看所有进程的信息：

ps -eo pid,tid,ppid,comm

**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。



**3. 进程的创建**

Linux中创建进程与其他系统有个主要区别，Linux中创建进程分2步：fork()和exec()。

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：

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。

check新进程(进程数目是否超出上限等)

清理新进程的信息(比如PID置0等)，使之与父进程区别开。

新进程状态置为 TASK\_UNINTERRUPTIBLE

更新task\_struct的flags成员。

调用alloc\_pid()为新进程分配一个有效的PID

根据clone()的参数标志，拷贝或共享相应的信息

做一些扫尾工作并返回新进程指针

创建进程的fork()函数实际上最终是调用clone()函数。创建线程和进程的步骤一样，只是最终传给clone()函数的参数不同。比如，通过一个普通的fork来创建进程，相当于：clone(SIGCHLD, 0)创建一个和父进程共享地址空间，文件系统资源，文件描述符和信号处理程序的进程，即一个线程：clone(CLONE\_VM | CLONE\_FS | CLONE\_FILES | CLONE\_SIGHAND, 0)在内核中创建的内核线程与普通的进程之间还有个主要区别在于：内核线程没有独立的地址空间，它们只能在内核空间运行。这与Linux内核是个单内核有关。

**4. 进程的终止**

和创建进程一样，终结一个进程有很多步骤：

 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。

父进程上的操作(release\_task)

父进程受到子进程发送的exit\_notify()信号后，将该子进程的进程描述符和所有进程独享的资源全部删除。

从上面的步骤可以看出，必须要确保每个子进程都有父进程，如果父进程在子进程结束之前就已经结束了会怎么样呢？子进程在调用exit\_notify()时已经考虑到了这点。

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)