# 进程调度

软件部------邵望权

# 进程的产生

**进程**

- 进程描述符task_strust
- 状态

**产生**

- Linux系统的启动
- 0号进行的由来

**创建**

- **fork**

# 进程

- 进程：是处于执行期的程序及其相关资源的总称。

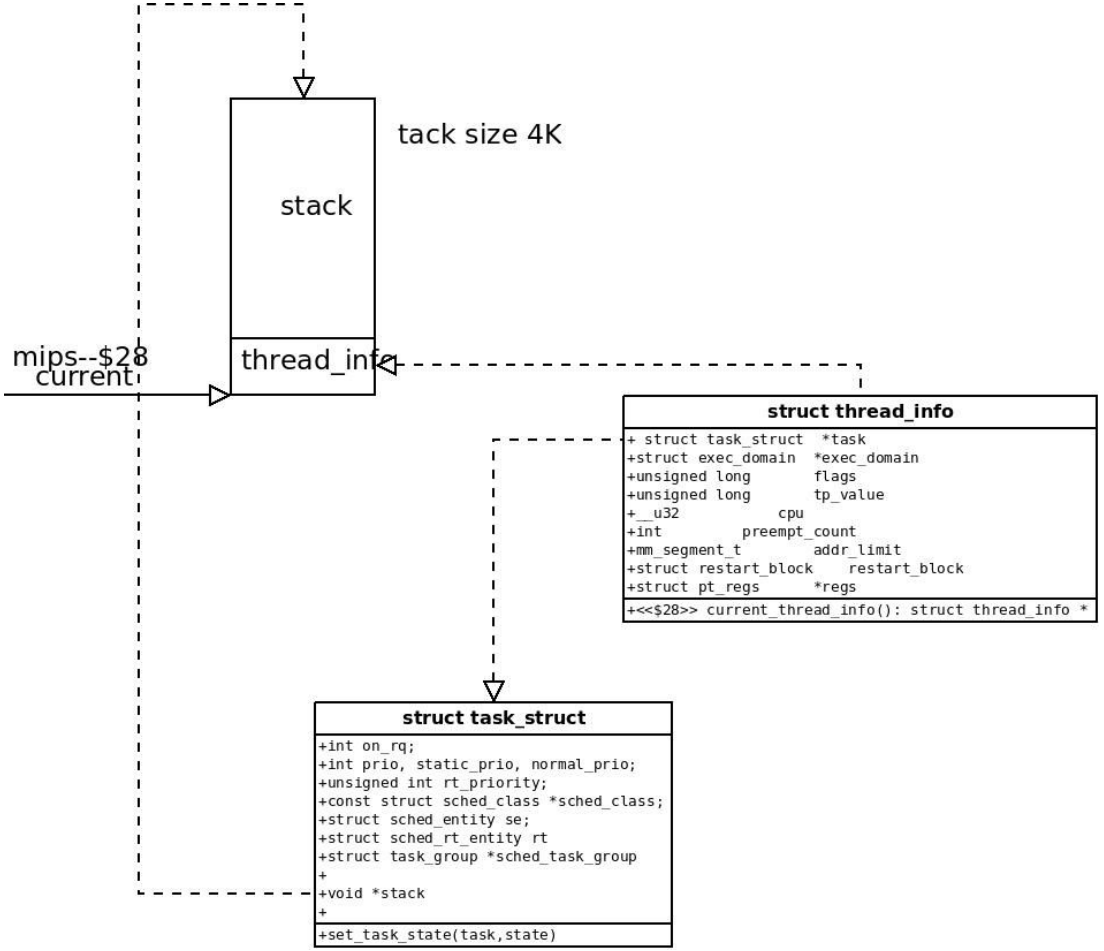
- 在内核中对进程的描述是通过tast_struct，进程描述符记录。

# Task_struct

- 内核数据结构task_struct

- Init_task

# 内核栈和进程描述符之间的关系



# 进程生命周期

- 进程是一个动态的实体。

- **进程因创建而产生，因调度而执行，因撤销而消亡**
- **do_fork → schedule → exit**
- 三种状态：

  **就绪态：** 进程已经获得了除cpu以外的所有其它资源，在就绪队列中等待cpu调度。

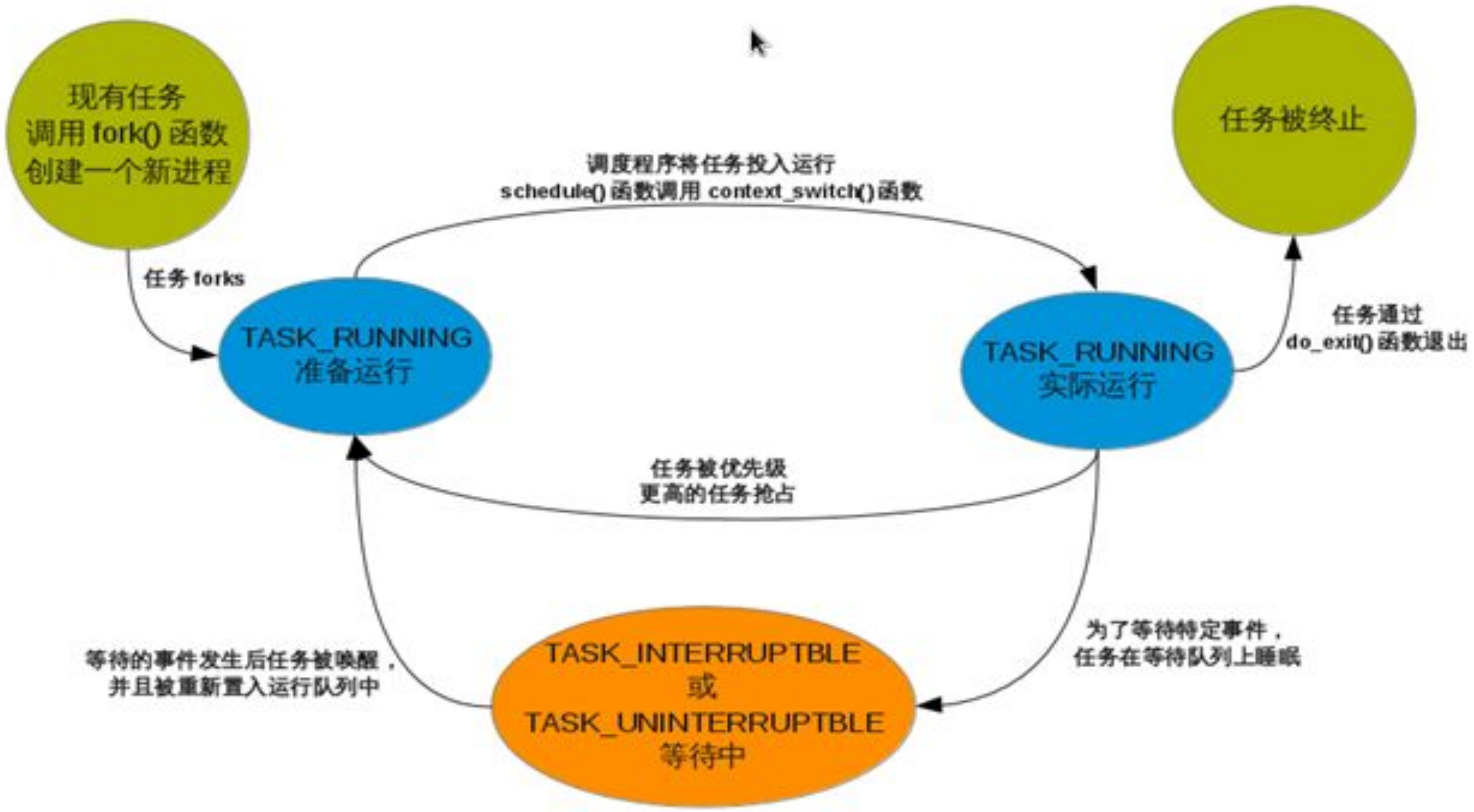
  **执行状态：** 已经获得cpu以及所有需要的资源正在运行。

  **阻塞状态(等待状态)：** 进程因等待所需要的资源而放弃处理器，或者进程本来就不拥有处理器，且其它资源也没有满足。

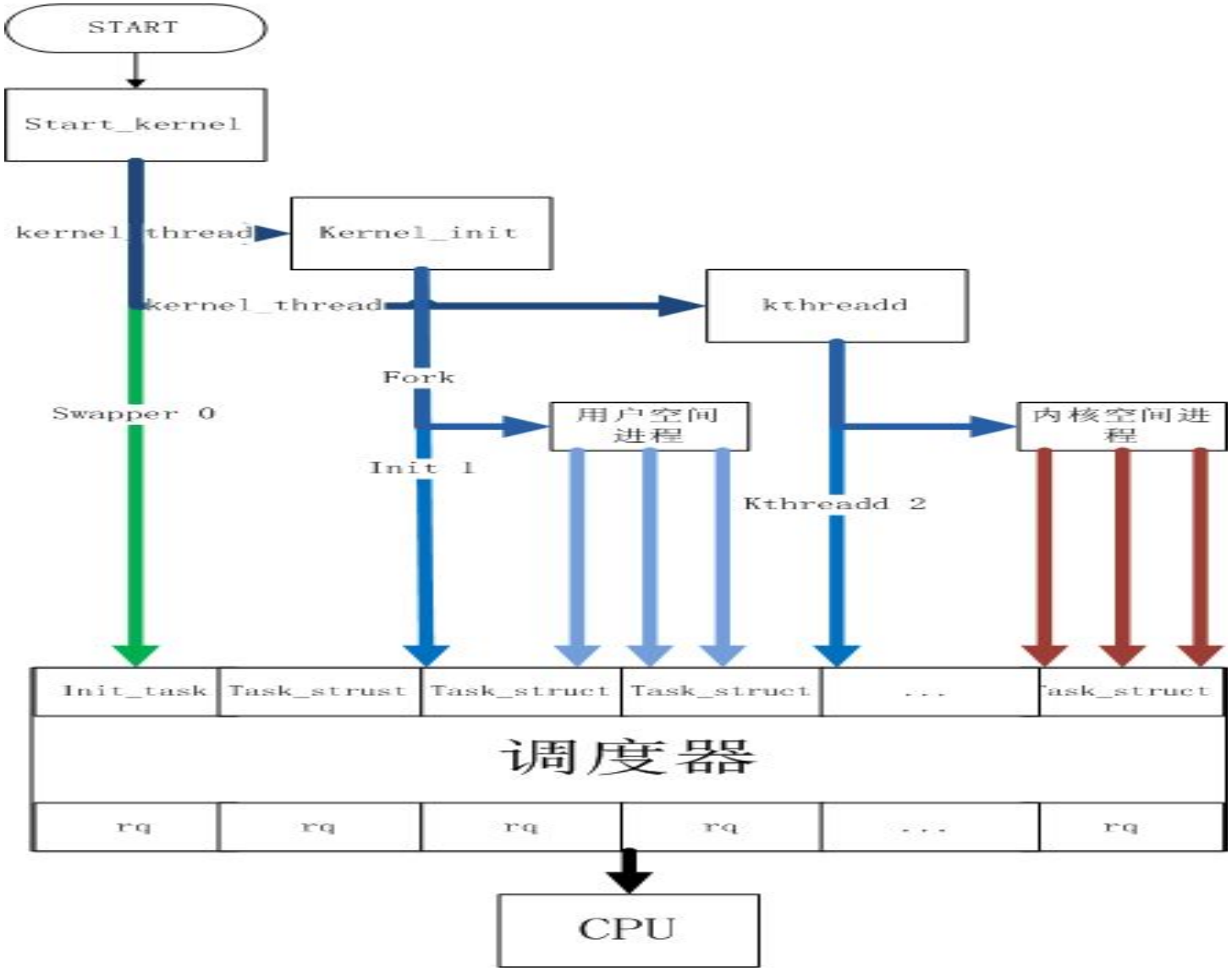
# 进程状态描述

- TASK_RUNNING
- TASK_INTERRUPTIBLE
- TASK_UNINTERRUPTIBLE
- __TASK_TRACED
- __TASK_STOPPED

# 转化关系


现有任务
调用 fork() 函数
创建一个新进程
任务 forks
TASK_RUNNING
准备运行
调度程序将任务投入运行
schedule() 函数调用 context_switch() 函数
TASK_RUNNING
实际运行
任务被终止
任务通过
do_exit() 函数退出
任务被优先级
更高的任务抢占
为了等待特定事件，
任务在等待队列上睡眠
TASK_INTERRUPTBLE
或
TASK_UNINTERRUPTBLE
等待中
等待的事件发生后任务被唤醒，
并且被重新置入运行队列中

# Linux内核进程关系




单核处理器，多核有所不同

道生一，一生二，

二生三，三生万物

-----出自老子的《道德经》第二十四章

# 进程创建

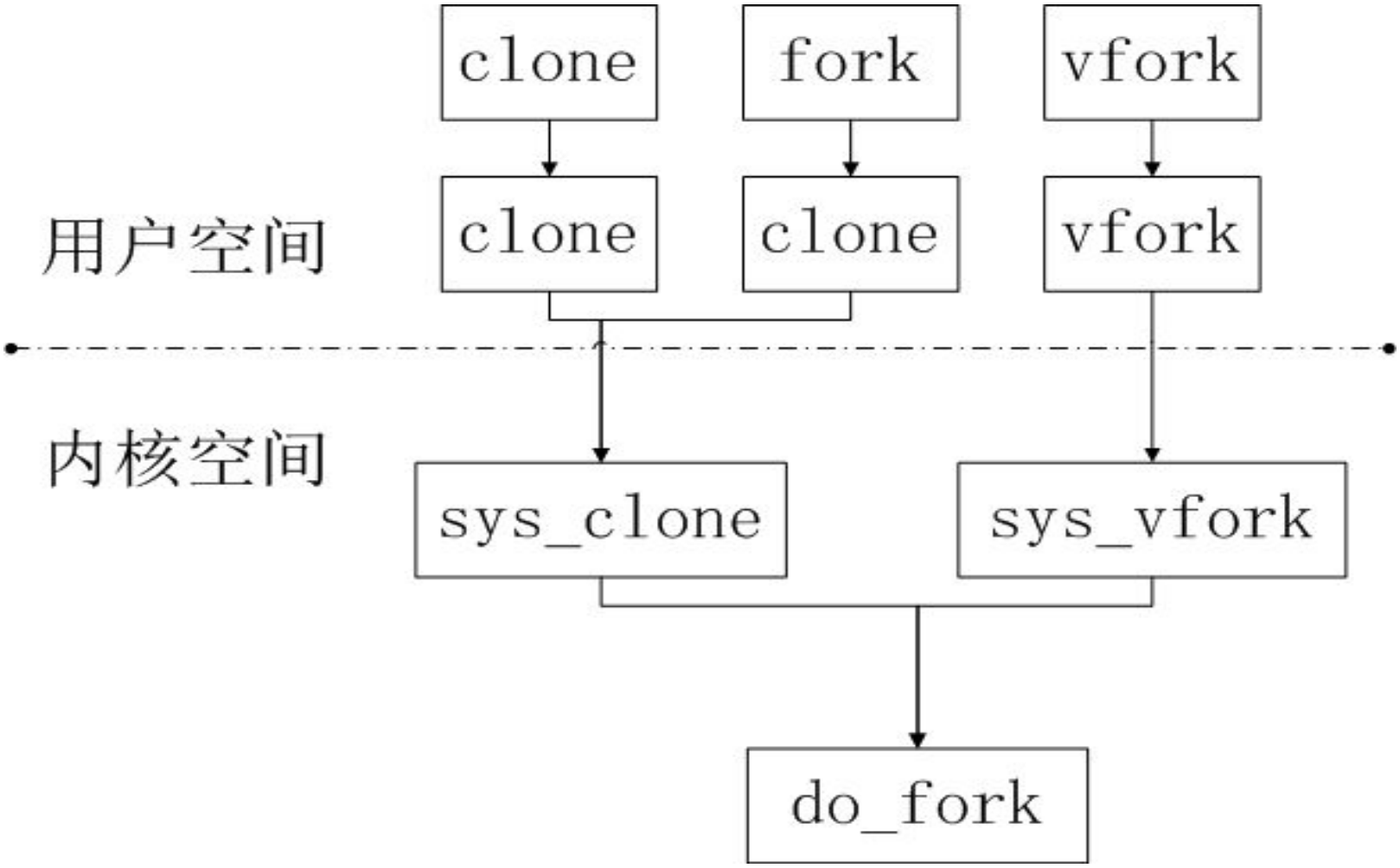
- 用户空间
  - fork
    - fork创造的子进程是父进程的完整副本，复制了父进程的资源
  - vfork
    - vfork创建的子进程与父进程共享数据段,
  - clone
    - 将父进程资源有选择地复制给子进程(带有参数)
- 内核空间
  - kernel_thread

# clone

- 函数：
- int clone(int (*fn)(void *), void *child_stack, int flags, void *arg)
  - fn为函数指针，此指针指向一个函数体，即想要创建进程的静态程序；
  - child_stack为给子进程分配系统堆栈的指针；
  - arg就是传给子进程的参数；
  - flags为要复制资源的标志：

- CLONE_PARENT 创建的子进程的父进程是调用者的父进程，新进程与创建它的进程成了“兄弟”而不是“父子”
- CLONE_FS 子进程与父进程共享相同的文件系统
- CLONE_FILES 子进程与父进程共享相同的文件描述符表
- CLONE_NEWNS 在新的namespace启动子进程
- CLONE_SIGHAND 子进程与父进程共享相同的信号处理（signal handler）表
- CLONE_PTRACE 若父进程被trace，子进程也被trace
- CLONE_VFORK 父进程被挂起，直至子进程释放虚拟内存资源
- CLONE_VM 子进程与父进程运行于相同的内存空间
- CLONE_PID 子进程在创建时PID与父进程一致
- CLONE_THREAD Linux 2.4中增加以支持POSIX线程标准，子进程与父进程共享相同的线程群

# 三者关系


clone
fork
vfork
clone
clone
vfork
用户空间
内核空间
sys_clone
sys_vfork
do_fork

# 示例

- fork
- vfork
- clone

# 区别联系

- 区别：
  - fork() 子进程拷贝父进程的数据段，代码段。
  - vfork() 子进程与父进程共享数据段。
  - fork和vfork，拷贝完成后都是子进程先执行。
  - vfork在子进程执行时，父进程将被阻塞。

- 联系：
  - 子进程都是通过父进程拷贝得到。

# 系统调用

- sys_fork
  - do_fork(SIGCHLD, 0, 0, NULL, NULL)
- sys_vfork
  - do_fork(CLONE_VFORK | CLONE_VM | SIGCHLD, 0, 0, NULL, NULL);
- sys_clone
  - do_fork(clone_flags, newsp, 0, parent_tidptr, child_tidptr)

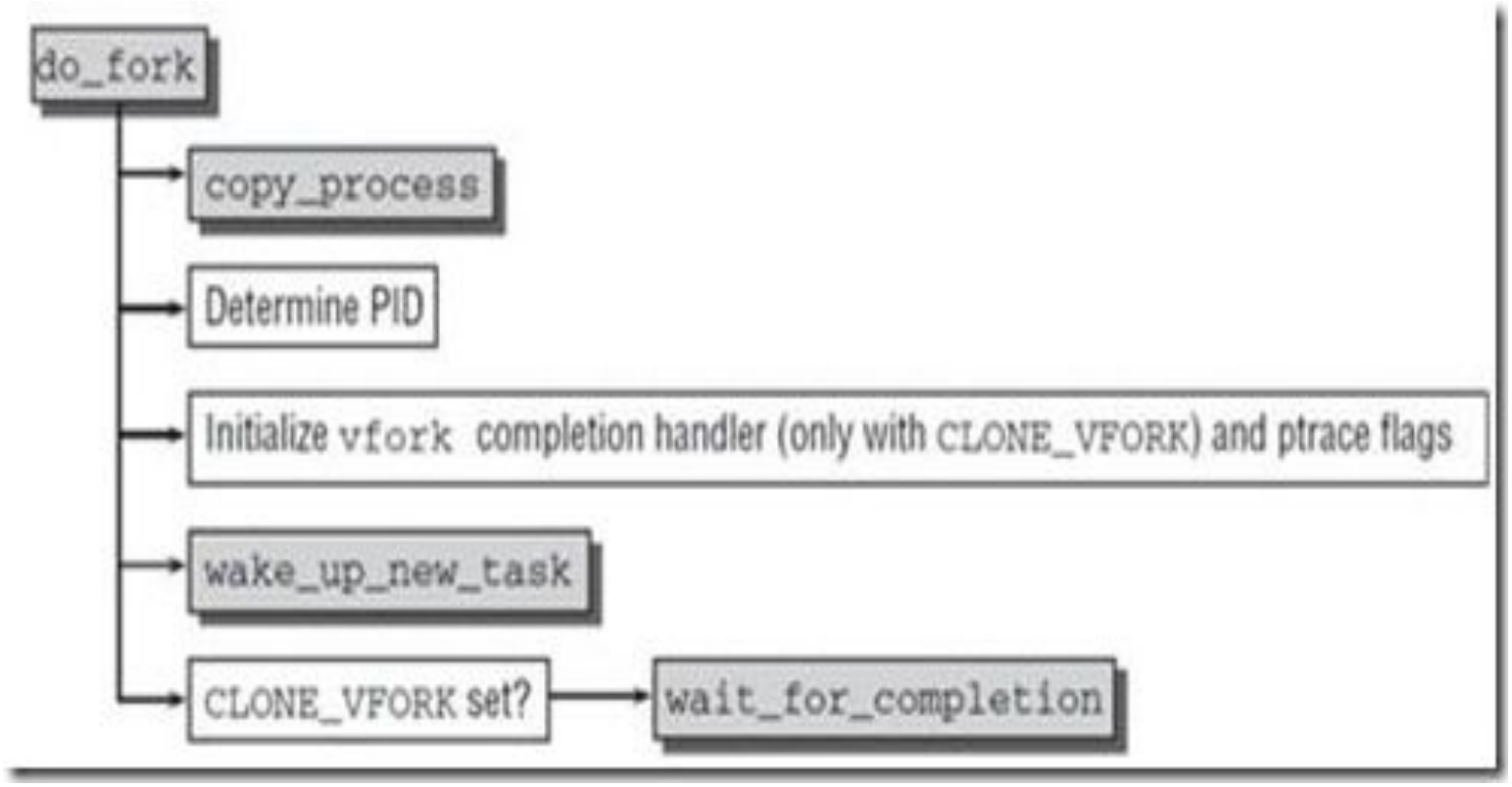
# do_fork

- 函数：
  - long do_fork(unsigned long clone_flags,　unsigned long stack_start, unsigned long stack_size, int __user *parent_tidptr, int __user *child_tidptr)
- 参数：
  - clone_flags与clone()参数flags相同, 用来控制进程复制过的一些属性信息, 描述你需要从父进程继承那些资源。
  - stack_start与clone()参数stack_start相同, 子进程用户态堆栈的地址regs是一个指向了寄存器集合的指针, 其中以原始形式, 保存了调用的参数, 该参数使用的数据类型是特定体系结构的struct pt_regs，其中按照系统调用执行时寄存器在内核栈上的存储顺序, 保存了所有的寄存器, 即指向内核态堆栈通用寄存器值的指针，通用寄存器的值是在从用户态切换到内核态时被保存到内核态堆栈中的(指向pt_regs结构体的指针。当系统发生系统调用，即用户进程从用户态切换到内核态时，该结构体保存通用寄存器中的值，并被存放于内核态的堆栈中)
  - stack_size用户状态下栈的大小, 该参数通常是不必要的, 总被设置为0
  - parent_tidptr与clone的ptid参数相同, 父进程在用户态下pid的地址，该参数在CLONE_PARENT_SETTID标志被设定时有意义child_
  - tidptr与clone的ctid参数相同, 子进程在用户态下pid的地址，该参数在CLONE_CHILD_SETTID标志被设定时有意义

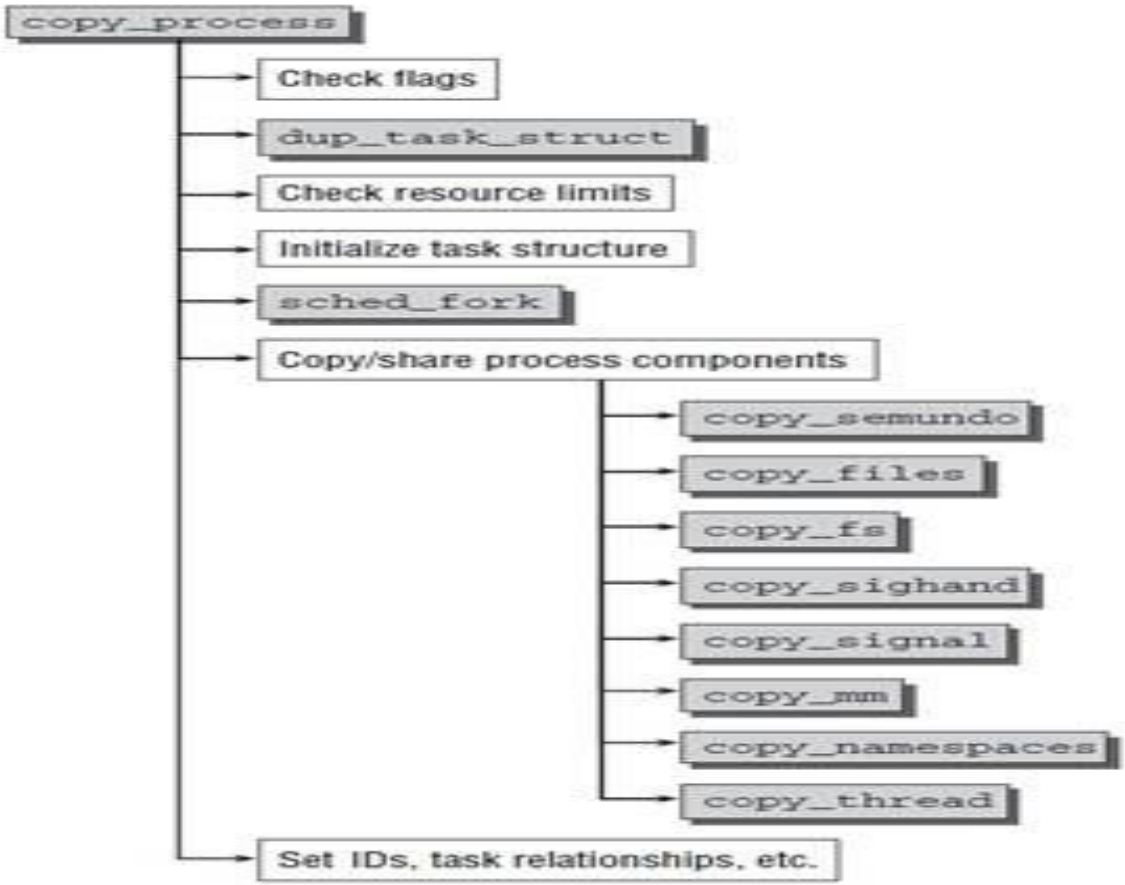
# 进程创建的相关属性

| 标志名称 | 说明 |
| --- | --- |
| CLONE_VM | 共享内存描述符和所有的页表； |
| CLONE_FS | 共享根目录和当前工作目录所在的表，以及用于屏蔽文件初始许可权的位掩码值，即文件的 umask； |
| CLONE_PTRACE | 如果父进程被跟踪，那么子进程也被跟踪； |
| CLONE_VFORK | 在发出 vfork 系统调用时设置； |
| CLONE_STOPPED | 强迫子进程开始于 TASK_STOPPED 状态； |
| CLONE_SIGHAND | 共享父进程信号处理表，阻塞信号表和挂起信号表。如果设置了此标志，就必须同时设置 CLONE_VM 标志； |
| CLONE_THREAD | 把子进程插入到父进程所在的线程组中，并迫使子进程共享父进程的信号描述符。因此该标志被设置时，就必须设置 CLONE_SIGHAND 标志； |

# do_fork现实


do_fork
copy_process
Determine PID
Initialize vfork completion handler (only with CLONE_VFORK) and ptrace flags
wake_up_new_task
CLONE_VFORK set?
wait_for_completion

# copy_process



# 问题

- 使用fork创建进程时，为什么父进程返回子进程ID，而子进程返回0？

# COW---copy on write

- 写时拷贝是一种可以推迟甚至免除拷贝数据的技术。内核此时并不复制整个进程地址空间，而是让父进程和子进程共享同一个拷贝。只有在需要写入的时候，数据才会被复制，从而使各个进程拥有各自的拷贝。也就是说，资源的复制只有在需要写入的时候才进行，在此之前，只是以只读方式共享。

# 消亡

- 进程的消亡：
  - 一个进程既有父进程又有子进程，因此进程消亡时，既要通知父进程，也要安排好子进程（找养父）。

- 实现：
  - sys_exit()
  - sys_wait4()

# 特殊情况

- 孤儿进程：
  - 一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程
- 僵尸进程：
  - 一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

# 调度

为什么调度
- 调度原因
- 调度过程

调度
- 调度策略
- 调度算法

应用
- 抢占
- **schdule**

# 目标

- 公平：保证每个进程得到合理的CPU时间
- 高效：使CPU保持忙碌状态
- 响应时间：交互用户的响应时间尽可能短
- 周转时间：使批处理用户等待输出的时间尽可能短
- 吞吐量：使单位时间内处理的进程数量尽可能多
- 负载均衡：在多核多处理器系统中提供更高的性能

# 调度原因

- 正在执行的进程执行完毕
- 执行中进程将自己阻塞进入睡眠等状态
- 执行中进程因资源不足而被阻塞；或调用了v原语操作激活了等待资源的进程队列
- 执行中进程提出I/O请求后被阻塞
- 在分时系统中时间片已经用完
- 在执行完系统调用等系统程序后返回用户进程时，这时可看作系统进程执行完毕
- 就绪队列中的某进程的优先级变得高于当前执行进程的优先级，从而也将引发进程调度

# 依据

- IO消耗型和处理器消耗型进程
- 进程优先级
  - Nice值
    - 范围：-20～+19
    - Nice越高优先级越低
  - 实时优先级
    - 范围：0～99
    - 越大优先级越高
- 时间片

# 策略

- SCHED_NORMAL
  - 应用：普通进程
- SCHED_FIFO
  - 应用：实时进程
- SCHED_RR
  - 应用：实时进程
- SCHED_BATCH
  - 应用：普通进程（不能抢占）
- SCHED_IDLE

# 调度类

- idle_sched_class
  - pid=0, 调度类属于：idel_sched_class，所以在ps里面是看不到的。一般运行在开机过程和cpu异常的时候做dump。
- stop_sched_class
  - 进程迁移和CPU热插拔
- rt_sched_class
  - 实时线程
- fair_sched_class
  - 普通进程（CFS调度算法）

# 调度

- 调度是对运行队列（rq）中的进程进行处理
  - rq
    - rt_rq
    - cfs_rq

- 调度顺序：
  - stop_sched_class → rt_sched_class → fair_sched_class → idle_sched_class

# 相关数据结构

- struct task_group
- struct sched_entity
- struct sched_class

# 调度算法

- 普通调度
  - 公平调度（CFS）
    - deal_time = sum_runtime * se.weight/cfs_rq.weight
    - 红黑树实现
- 实时调度
  - 优先级+时间片

# schedule

- 作用：选择哪个进程运行，何时投入运行

- 选择：pick_next_task(rq)
  - 进程的选择是调度类的选择
  - 调度类 = 运行队列 +调度算法
- 切换：
  - context_switch(rq, prev, next)
  - switch_to(prev, next, prev)

# 睡眠和唤醒

- 休眠
  - 等待队列
  - 状态：TASK_INTERRUPTIBLE

- 唤醒
  - wake_up -> try_to_wake_up
  - 状态：TASK_RUNNING

# 抢占

- 用户抢占
  - 从系统调用返回用户空间时
  - 从中断处理程序返回用户空间时
- 内核抢占
  - 中断处理程序正在执行，且返回内核空间之前
  - 内核代码在一次具有可抢占性的时候
  - 内核中的任务显示的调用schedule
  - 内核中的任务阻塞

谢谢