# Linux内核简介

## 单内核和微内核

操作系统可以分为两大阵营，单内核和微内核

- 单内核
  - 单内核的性能更高，设计简单，速度更快
  - 单内核可以直接调用函数
- 微内核
  - 微内核功能被划分为多个过程，每个过程被称作一个服务器。只有强烈请求特权服务的服务器才运行在特权模式下，其他服务器都运行在用户空间
  - 单内核不能直接调用函数，而是通过消息传递处理微内核通信，系统采用了进程间通信IPC机制。
  - 这样的多个服务器的模块化系统可以有效的避免由于一个服务器失效导致的连锁反应，也可以方便的更换不需要的服务器

## Linux的内核

Linux是一个单内核，但是他汲取了微内核的精华。他与传统的unix系统存在差异

- Linux支持动态加载内核模块，尽管他是单内核
- Linux支持对称多处理（SMP）机制。
- Linux内核可以抢占。他具有允许在内核运行的任务优先执行的能力
- Linux内核并不区分线程和一般进程。对于Linux内核来说都一样，只不过是共享了一些资源而已
- Linux提供具有设备类面向对象的设备模型，热插拔事件，以及用户空间设备文件系统

# 内核开发的特点

## 无libc库也没有标准头文件

- 与用户空间不同，内核不能链接使用标准的C函数库以及其他的函数库，因为他们都太大且低效
- 常见的数据结构函数组也有实现，在类似<linux/string.h>的头文件中。

## GNU C

内核开发者使用C语言涵盖了 ISO C99 和GNU C的扩展特性。包括下面几个比较关键的

- 内联函数：当你使用的时候函数会在他所调用的位置上展开，可以消除函数调用和返回带来的开销。优先使用内联函数而不是复杂的宏

- 内联汇编：gcc编译器支持在C函数嵌入汇编指令，当然在内核编程的时候，必须知道对应的体系结构才能用
- 分支声明：我们可以使用likely/unlikely来干扰当前分支选择的优化

## 没有内存保护机制

- 如果一个用户程序试图进行一次非法访问，内核就会发出SIGSEGV信号，但是如果是内核自己访问了内存那就无法发现
- 内核中的内存不分页。每用掉一个字节，物理内存就会减少一个字节

## 不要轻易的在内核中使用浮点数

- 与用户进程空间不同，内核并不能完美的支持浮点操作，因为他本身不能陷入

## 容积小而固定的栈

- 与用户栈不同，内核栈的大小随着体系结构改变，在64位为16kb，32位为8kb

## 同步和并发

- 内核很容易产生竞争条件，和单线程的用户空间程序不同，内核许多特性都要求能并发的访问和共享数据

# 进程管理

## 进程概述

进程就是处于执行期的程序，通常进程还包括其他资源，比如打开的文件，挂起的信号，内部内核数据，处理器状态等等

- 对于Linux来说，线程是一种特殊的进程，他享有一个独立的计数器，进程栈和一个进程寄存器内核调度的对象是线程
- 程序本身不是进程，进程是出于执行期的程序以及其相关资源的总称，实际上，完全可能存在两个或者多个不同的进程执行同一个程序

进程从他被创建的时候开始存活

- 在Linux系统中，通过fork()这组函数，该系统通过复制一个现有的进程来创建要给全新的进程，调用fork()的进程被叫做父进程，新产生的被叫做子进程
- 在调用结束的时候，在返回点这个相同的位置上，父进程恢复执行，子进程开始执行，fork()从系统调用从内核返回两次，一次是回到父进程，一次是回到新产生的子进程
- 然后创建完新的进程，就会接着调用exec()这组函数就可以创建新的地址空间，将新的程序载入其中（在Linux系统中，这组过程主要是由clone()这套系统调用实现，具体的在后面的笔记）

- 最终，程序通过exit()系统调用退出执行，父进程可以通过wait4()系统调用查询子进程是否终结。进程退出后设置为僵死状态，直到他的父进程调用wait()或waitpid()

## 进程描述符以及任务结构

进程描述符

- 内核把进程列表存放在任务队列的双向循环链表中，该结构定义在<Linux/sched.h>文件中，对于链表中每一项，我们把它叫做taskstruct，也叫做进程描述符。
- 进程描述符中包含数据能完整描述一个正在执行的程序，包括：打开的文件，进程地址的空间，挂起的信号，进程的状态等

分配进程描述符

- Linux通过slab分配器分配task_struct的结构，这样能达到对象复用和缓存着色的目的。
- 分配器一般会在栈底/栈顶生成一个 struct thread info。他的成员task域中存放的就是主席昂该任务实际的指针

## 进程描述符的存放

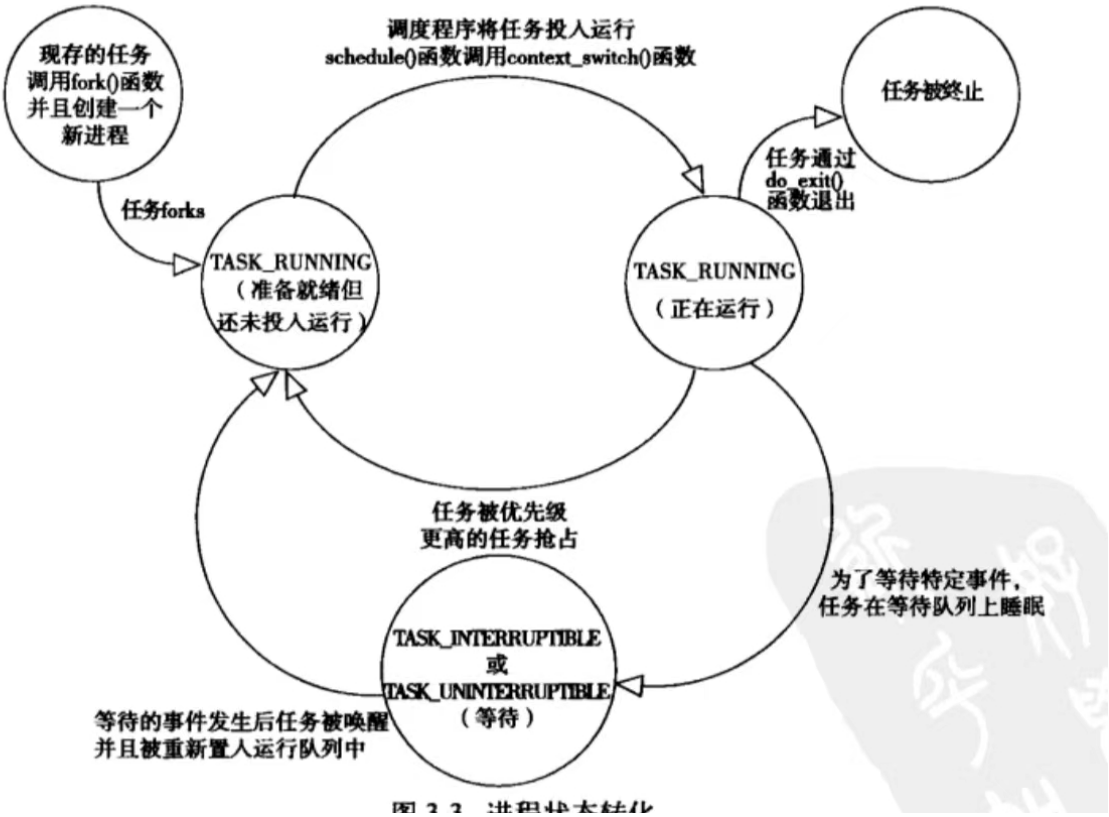
内核通过唯一的进程pid来标明每一个进程，他实际上是一个int类型

- pid默认值为32748（short int）的最大值，可以改只不过回合其他的unix系统不兼容了。
- 内核中大部分的代码都是通过task_struct进行运行的。
- 在访问任务的时候，通常需要获得只想其task_struct的指针，我们通过current宏去查找当前正在运行的进程。速度很快

## 进程状态

进程描述符中的state域描述了进程的当前状态，为下面五种标志之一

- TASK-RUNNING——进程是可执行/正在执行/等待执行的
- TASK-INTERRUPTIBLE——进程正在睡眠/被阻塞，等待某种条件达成后进行运行
- TASK-UNINTERRUPTIBLE——除了就算是接收到信号也不会唤醒或准备投入运行外，这个状态与可打断状态相同
- TASK-TRACED——被其他进程跟踪的进程，例如通过ptrace对调试程序进行跟踪
- TASK-STOPPED——进程停止执行，在接收到SIGSTOP等信号的时候停止运行




如果我们要设置对应的进程状态，我们就使用set_task_state(task,state)函数

## 进程上下文

- 一般的程序在用户空间运行
- 当一个程序执行了系统调用或者触发了某个异常，他就陷入了内核空间。此时我们称内核正在代表进程执行，并且处在进程上下文中（current宏指向当前所在的进程）
- 当内核退出的时候，程序恢复在用户空间继续执行

## 进程家族树

- 在Linux系统中，所有的进程都是PID为1的init进程的后代
- 系统中每一个进程都必有一个父进程，相应的每个进程可以拥有零个或者多个子进程。拥有同一个父进程的被称做兄弟。
- 每个task struct中都有一个指向父进程的parent指针，其指向上面一个task struct，以及一个称为children 的子进程链表。可以利用这种方式获取对应的进程，但是current宏会更快
- 对于init进程，他的进程描述符是作为init task静态分配的

# 进程创建

## clone()系统

Linux通过clone()系统完成一系列的进程创建。这其中包括几个关键的函数组：fork，exec，整个创建进程的过程就通过他们执行

- 首先fork()通过拷贝当前进程创建一个子进程，子进程和父进程的区别在于PID（每个进程唯一），PPID（父进程的进程号，子进程将其设置为被拷贝进程的PID）和某些资源的统计量
- exec()函数负责读取可执行文件并且将其载入地址空间开始运行

## fork()

#### fork与写时拷贝

Linux的fork()使用写时拷贝页实现。

- 在这里，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一个拷贝
- 只有在需要写入的时候，数据才会被复制，从而使得各个进程拥有各自的拷贝。资源的复制只有在需要写入的时候才进行，在此之前只会以只读的方式共享
- 在页根本不会被写入的情况下（fork()之后立刻调用exec())他们就无需复制了
- fork()的实际开销就是复制父进程的页表以及给与子进程创建唯一进程描述符

#### fork的具体实现

Linux通过clone()系统调用实现fork()。fork()的系列函数：fork(),vfork(),__clone()等函数都需要根据参数去调用clone()

clone一般主要去调用定义在<kernel/fork.c>的函数do_fork()，这个函数接着调用copy_process()完成复制工作

- 调用dup_task_struct()为新的进程创建一个内核栈，thread_info,task_struct。这些值的分配与当前进程相同。此时子进程和父进程的描述符完全相同的
- 检查并确保新创建这个子进程后，当前用户所拥有的进程数目没有超出给他分配的资源限制
- 子进程和父进程开始区别开。进程描述符的许多成员都要被清0或者设置初始值。他们不参与继承。task_struct中大多数数据还没有修改
- 随后把子进程的状态设置为TASK_UNINTERRUPTIBLE，保证他不会投入运行
- copy_process()调用copy_flags()以更新task_struct的flag成员。表明进程是否拥有超级用户权限的PF-SUPERPRIV标志被清0，表明进程还没有调用exec函数()的PF-FORKNOEXEC标志被设置
- 调用alloc_pid()为新的进程分配一个有效的PID
- 根据传递给clone()的参数标志，copy_process()拷贝或共享打开的文件，文件系统信息，命名空间等。
- 最后copy_process()做一个扫尾的工作并且返回一个指向子进程的指针

回到do_fork函数，如果copy_process()函数被成功设置，那么新创建的子进程会先运行，然后父进程再运行

- 这么做的好处是：一般情况下子进程都会立刻调用exec()函数，这样的话就可以避免写时拷贝的额外开销，如果父进程首先执行的话，有可能会开始向地址空间写入

注：对于vfork的话，和fork的区别就是他不拷贝写时页表项了，写时拷贝的消耗也在这

# 线程对于Linux的实现

## 简述

从Linux内核的角度来说，Linux没有线程这个概念，他把所有的线程都当作进程实现，内核也没有特别的调度算法或者特别的数据结构来组织线程，内核只是被视为和其他进程共享某些资源的进程，拥有独立的task_struct

## 创建线程

- 线程的创建和普通进程类似，但是在调用clone()需要传递一些特殊的参数标志指明需要共享的资源。
- 传递给clone的参数决定了新创建的进程的行为方式和父子进程的共享资源种类

## 内核线程

- 内核经常需要在后台执行一些操作，这种任务可以通过内核线程完成。
- 内核线程和普通进程间的区别在于内核进程没有独立的地址空间
- Linux会把一些任务交给内核线程来做，比如flush，ksofieqd

# 进程终结

当一个进程终结的时候，他必须释放他所占有的所有资源。它发生在进程调用exit()系统调用的时候，可以显示的调用也可以隐式的调用（比如C语言在代码后面会加上）这部分任务需要靠do_exit()来进行。具体如下

- 将task_struct中成员标志设置为 PF-EXITING
- 调用del_timer_sync()删除任一内核定时器，根据返回的结构，他确保没有定时器在排队，也没有定时器处理程序运行
- 然后调用exit_mm()函数释放进程占用的mm_struct()，如果没有别的进程使用，就彻底释放他们
- 接下来调用sem_exit()函数，如果进程排队等候IPC信号，他就离开队列
- 调用exit_files(),exit_fs()分别递减文件描述符，文件系统数据的引用计数，如果引用计数将为0，那就释放
- 调用exit_notify()向父进程发送信号，并且给子进程重新找养父，这个函数调用forget_original_parent(),find_new_reaper()来寻找新的进程，养父为线程组的其他进程或者是init进程，并且把进程状态设置为exit_zomble
    - 如果是init进程，那就是我们所说的孤儿进程。init进程会例行调用wait来检查，去释放这些资源
- do_exit()调用schedule()切换到新的进程，处于zomble状态的进程不会被调度，所以这是进程执行的最后一段代码，do_exit()永不返回

至此，他的唯一目的就是向父进程提供信息。并且系统保存了对应子进程的进程描述符，这么做就可以向父进程提供信息。在父进程获得已经终结的子进程的信息后或者通知内核它并不关注那些信息后，子进程对应的task_struct结构才被释放

当最终释放进程描述符的时候，release_task()会被调用，会完成下面的工作

- 他调用_*exit_signal()，该函数调用*unhash_process(),后者又调用detach_pid()从pidhash上删除对应的进程。也从任务列表中删除该进程
- _exit_signal()释放目前僵死进程所使用的剩余资源，并且做记录
- 如果这个进程是线程组最后一个线程，并且领头的线程以及死掉，那么release_task()就要体哦那个之僵死领头进程的父进程
- release_task()调用put_task_struct()释放进程和内核栈和thread-info结构所占的页，并且释放task-struct所占的slab缓存

到此所有的信息被释放掉了。