LinuxKernel2.6进程调度的分析(1)Linux认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022_LinuxKerne_c103_645049.htm 第一章 Kernel 2.4存在的不足 根据对2.4进程调度的分析，我们总结出看出2.4内核总的特点就是： 内核调度简单有效 内核不可抢占 但是经过对2.4内核的分析，我们也明显看到了它的缺点： 1.调度算法复杂度是O(n)，与系统负荷关系较大。而且调度算法在设计上也有缺陷，比如： (1) 2.4进程调度只设置了一个进程就绪队列，这样有的进程用完了自己时间片以后还要呆在就绪进程队列里面。这样这个进程虽然在这一轮调度循环里面已经无法取得CPU的使用权，但是还要参与goodness()值的计算，这样就白白浪费了时间。(2) 就绪进程队列是一个全局数据结构，多个CPU只有一个就绪队列runqueue，因而调度器对它的所有操作都会因全局自旋锁而导致系统各个处理机之间的等待，使得就绪队列成为一个明显的瓶颈。 2.调度算法在内核态不可抢占。如果某个进程一旦进了内核态那么再高优先级的进程都无法剥夺，只有等进程返回内核态的时候才可以进行调度。缺乏对实时进程的支持。 第二章Kernel 2.6进程调度分析 一、基本思想 Kernel2.6调度算法仍然是基于优先级的调度，它的算法复杂度为O(1)，也就是说是调度器的开销是恒定的，与系统当前的负载没有关系。 1. 就绪队列的改进 每个CPU有两个按优先级排序的数组：一个是active array；一个是expired array。Active array是当前CPU可能选择执行的运行进程队列，队列中的每个进程都有时间片剩下。Expired array是那些用户时间片的就绪进程队列。一旦active array里面 某个普通进程的时间

片用完了，调度器将重新计算进程的时间片、优先级，将它从active array中删除，插入到expired array中相应得优先级队列中。Active array和expired array是通过两个指向每个CPU运行队列的指针来访问的。所以当active array中所有的进程都用完时间片，只需将两个指针切换一下就可以了，这比Kernel 2.4的切换要改进了很多。 2. 快速查找应该执行的进程 系统中往往有很多的就绪进程，如何快速找到CPU即将运行的进程就成了关系到系统性能的一个重要因素。针对2.4的缺点，Kernel 2.6进行了重新设计：引进了一个64bit的bitmap作为进程队列的索引，用bitmap来记载某个优先级的进程队列上有无进程，如果有则为1。这样使得寻找优先级最高的任务只需要两个BSFL命令。 3. 引进"load estimator" 在一个负载很重的系统上有一个很好的交互感是一件很困难的事情，设计者经过研究发现一味的激励（boost）交互任务并不够，还需惩罚（punish）那些需求大于可获得CPU时间的进程。调度器通过对用户睡眠时间和运行时间的纪录来判断进程是否是交互进程，一旦被认为是交互进程，调度器会给进程很多"奖励"（bonus）。 4. 内核可抢占 内核可抢占可以说是2.6内核调度器优于2.4内核的一个很重要的原因。当内核进程没有访问内核的关键数据，也就是内核没有被加锁，此时内核代码是可重入的，因此更高优先级的进程可以在此时中断正在执行的进程，从而达到抢占的目的。 5. 调度器相关的负载均衡 负载均衡有两种策略，一种是从别的CPU上将进程迁移过来，称为"pull"；一种是将本CPU上的进程迁移出去，称为"push"。 二、数据结构 1. 进程优先级的划分 Kernel 2.6将进程优先级作了以下规定：进程优先级范围是从0 ~ MAX_PRIO-1，其中实

时进程的优先级的范围是0 ~ MAX_RT_PRIO-1，普通进程的优先级是MAX_RT_PRIO ~ MAX_PRIO-1。数值越小优先级越高。 2. 就绪队列runqueue（kernel/sched.c） struct runqueue是2.6调度器中一个非常重要的数据结构，它主要用于存放每个CPU的就绪队列信息。限于篇幅，这里只介绍其中相对重要的部分： (1) prio_array_t *active, *expired, arrays[2] 这是runqueue中最重要的部分。每个CPU的就绪队列都是一个数组，按照时间片是否用完将就绪队列分为两个部分，分别用指针active和expired来指向数组的两个下标。prio_array_t的结构如下： struct prio_array { int nr_active. /*本进程组中进程个数*/ struct list_head queue[MAX_PRIO]. /*每个优先级的进程队列*/ unsigned long bitmap[BITMAP_SIZE]. /*上述进程队列的索引位图*/ }. 数组queue[MAX_PRIO]里面存放的是优先级为i（MAX_PRIOgt.=0）的进程队列的链表头，即task_struct::runlist（通过runnlist即可找到task_struct）。