优先级为100的任务可以得到800ms的时间片，而优先级为139的任务只能得到5ms的时间片。

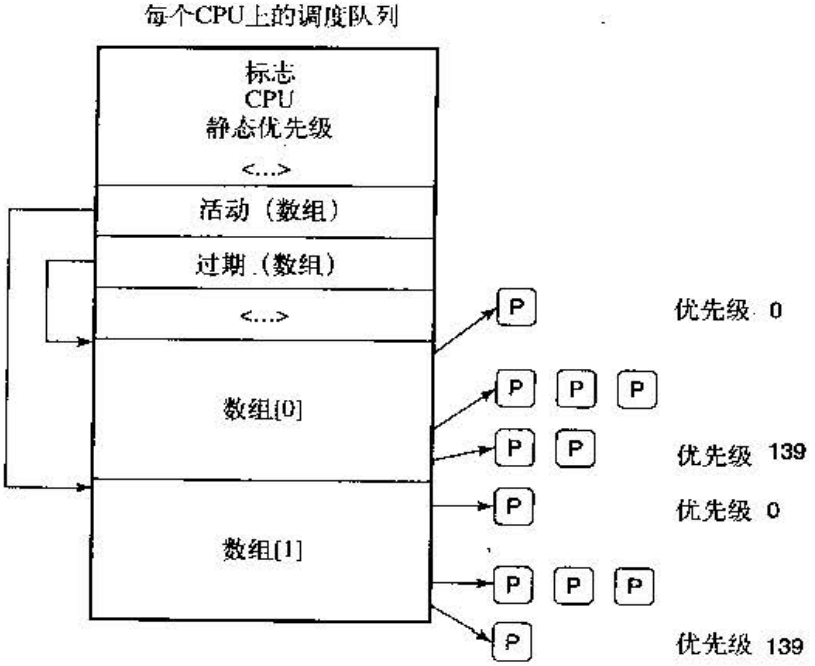

图10-10 Linux调度队列和优先级数组

这种调度模式的思想是为了使进程更快地出入内核。如果一个进程试图读取一个磁盘文件，在调用read函数之间等待一秒钟的时间显然会极大地降低进程的效率。每个请求完成之后让进程立即运行的做法会好得多，同时这样做也可以使下一个请求更快的完成。相似地，如果一个进程因为等待键盘输入而阻塞，那么它明显是一个交互进程，这样的进程只要准备好运行后就应当被赋予较高的优先级，从而保证交互进程可以提供较好的服务。在这种情况下，当I/O密集进程和交互进程被阻塞之后，CPU密集进程基本上可以得到所有被留下的服务。

由于Linux系统（或其他任何操作系统）事先不知道一个任务究竟是I/O密集的，还是CPU密集的，它只是依赖于连续保持的互动启发模式。通过这种方式，Linux系统区分静态优先级和动态优先级。线程的动态优先级不断地被重新计算，其目的在于：(1) 奖励互动进程，(2) 惩罚占用CPU的进程。最高的优先级奖励是−5，是从调度器接收的与更高优先级相对应的较低优先级的值。最高的优先级惩罚是+5。

说得更详细些，调度器给每一个任务维护一个名为sleep_avg的变量。每当任务被唤醒时，这个变量会增加；当任务被抢占或时间量过期时，这个变量会相应地减少。减少的值用来动态生成优先级奖励，奖励的范围从−5到+5。当一个线程从正在活动数组移动到过期失效数组中时，Linux系统的调度器会重新计算它的优先级。

这里讲述的调度算法指的是2.6版本内核，最初引入这个调度算法的是不稳定的2.5版本内核。早期的调度算法在多处理器环境中所表现的性能十分低下，并且当任务的数量大量增长时，不能很好地进行调度。由于上面描述的内容说明了通过访问正在活动数组就可以做出调度决定，那么调度可以在一个固定的时间O（1）内完成，而与系统中进程的数量无关。

另外，调度器包含了对于多处理器和多核平台而言非常有益的特性。首先，在多处理器平台上，运行队列数据结构与某一个处理器相对应，调度器尽量进行亲和调度，即将之前在某个处理器上运行过的任务再次调入该处理器。第二，为了更好地描述或修改一个选定的线程对亲和性的要求，有一组系统调用可供调用。最后，在满足特定性能和亲和要求的前提下，调度器实现在不同处理器上阶段性地加载平衡，从而保证整个系统的加载是平衡的。

调度器只考虑可以运行的任务，这些任务被放在适当的调度队列当中。不可运行的任务和正在等待各种I/O操作或内核事件的任务被放入另一个数据结构当中，即等待队列。每一种任务可能需要等待的事件对应了一个等待队列。等待队列的头包含一个指向任务链表的指针及一枚自旋锁。为了保证等待队