100，上校为90，少校为80，上尉为70，中尉为60，以此类推。或者，在一个商业计算中心，高优先级作业每小时费用为100美元，中优先级每小时75美元，低优先级每小时50美元。UNIX系统中有一条命令nice，它允许用户为了照顾别人而自愿降低自己进程的优先级。但从未有人用过它。

为达到某种目的，优先级也可以由系统动态确定。例如，有些进程为I/O密集型，其多数时间用来等待I/O结束。当这样的进程需要CPU时，应立即分配给它CPU，以便启动下一个I/O请求，这样就可以在另一个进程计算的同时执行I/O操作。使这类I/O密集型进程长时间等待CPU只会造成它无谓地长时间占用内存。使I/O密集型进程获得较好服务的一种简单算法是，将其优先级设为$1/f$，f为该进程在上一时间片中所占的部分。一个在其50ms的时间片中只使用1ms的进程将获得优先级50，而在阻塞之前用掉25ms的进程将具有优先级2，而使用掉全部时间片的进程将得到优先级1。

可以很方便地将一组进程按优先级分成若干类，并且在各类之间采用优先级调度，而在各类进程的内部采用轮转调度。图2-42给出了一个有4类优先级的系统，其调度算法如下：只要存在优先级为第4类的可运行进程，就按照轮转法为每个进程运行一个时间片，此时不理会较低优先级的进程。若第4类进程为空，则按照轮转法运行第3类进程。若第4类和第3类均为空，则按轮转法运行第2类进程。如果不偶尔对优先级进行调整，则低优先级进程很可能会产生饥饿现象。

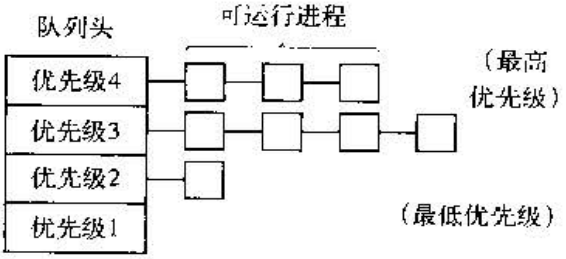

图2-42 有4个优先级类的调度算法

3. 多级队列

CTSS（Compatible TimeSharing System），M.I.T.在IBM 7094上开发的兼容分时系统（Corbató等人，1962），是最早使用优先级调度的系统之一。但是在CTSS中存在进程切换速度太慢的问题，其原因是IBM 7094内存中只能放进一个进程，每次切换都需要将当前进程换出到磁盘，并从磁盘上读入一个新进程。CTSS的设计者很快便认识到，为CPU密集型进程设置较长的时间片比频繁地分给它们很短的时间片要更为高效（减少交换次数）。另一方面，如前所述，长时间片的进程又会影响到响应时间，其解决办法是设立优先级类。属于最高优先级类的进程运行一个时间片，属于次高优先级类的进程运行2个时间片，再次一级运行4个时间片，以此类推。当一个进程用完分配的时间片后，它被移到下一类。

作为一个例子，考虑有一个进程需要连续计算100个时间片。它最初被分配1个时间片，然后被换出。下次它将获得2个时间片，接下来分别是4、8、16、32和64。当然最后一次它只使用64个时间片中的37个便可以结束工作。该进程需要7次交换（包括最初的装入），而如果采用纯粹的轮转算法则需要100次交换。而且，随着进程优先级的不断降低，它的运行频度逐渐放慢，从而为短的交互进程让出CPU。

对于那些刚开始运行一段长时间，而后来又需要交互的进程，为了防止其永远处于被惩罚状态，可以采取下面的策略。只要终端上有回车键（Enter键）按下，则属于该终端的所有进程就都被移到最高优先级，这样做的原因是假设此时进程即将需要交互。但可能有一天，一台CPU密集的重载机器上有几个用户偶然发现，只需坐在那里随机地每隔几秒钟敲一下回车键就可以大大提高响应时间。于是他又告诉所有的朋友……这个故事的寓意是：在实践上可行比理论上可行要困难得多。

已经有许多其他算法可用来对进程划分优先级类。例如，在伯克利制造的著名的XDS 940系统中（Lampson，1968），有4个优先级类，分别是终端、I/O、短时间片和长时间片。当一个一直等待终端输入的进程最终被唤醒时，它被转到最高优先级类（终端）。当等待磁盘块数据的一个进程就绪时，将它转到第2类。当进程在时间片用完时仍为就绪时，它一般被放入第3类。但如果一个进程已经多次用完时间片而从未因终端或其他I/O原因阻塞，那么它将被转入最低优先级类。许多其他系统也使用类似的算法，用以讨好交互用户和进程，而不惜牺牲后台进程。

4. 最短进程优先

对于批处理系统而言，由于最短作业优先常常伴随着最短响应时间，所以如果能够把它用于交互进程，那将是非常好的。在某种程度上，的确可以做到这一点。交互进程通常遵循下列模式：等待命令、执行命令、等待命令、执行命令，如此不断反复。如果我们将每一条命令的执行看作是一个独立的“作