业”，则我们可以通过首先运行最短的作业来使响应时间最短。这里惟一的问题是如何从当前可运行进程中找出最短的那一个进程。

一种办法是根据进程过去的行为进行推测，并执行估计运行时间最短的那一个。假设某个终端上每条命令的估计运行时间为T_0。现在假设测量到其下一次运行时间为T_1。可以用这两个值的加权和来改进估计时间，即$aT_0 + (1-a)T_1$。通过选择a的值，可以决定是尽快忘掉老的运行时间，还是在一段长时间内始终记住它们。当$a = 1/2$时，可以得到如下序列：

$$T_0,\ T_0/2 + T_1/2,\ T_0/4 + T_1/4 + T_2/2,\ T_0/8 + T_1/8 + T_2/4 + T_3/2$$

可以看到，在三轮过后，T_0在新的估计值中所占的比重下降到1/8。

有时把这种通过当前测量值和先前估计值进行加权平均而得到下一个估计值的技术称作老化(aging)。它适用于许多预测值必须基于先前值的情况。老化算法在$a = 1/2$时特别容易实现，只需将新值加到当前估计值上，然后除以2（即右移一位）。

5. 保证调度

一种完全不同的调度算法是向用户作出明确的性能保证，然后去实现它。一种很实际并很容易实现的保证是：若用户工作时有n个用户登录，则用户将获得CPU处理能力的$1/n$。类似地，在一个有n个进程运行的单用户系统中，若所有的进程都等价，则每个进程将获得$1/n$的CPU时间。看上去足够公平了。

为了实现所做的保证，系统必须跟踪各个进程自创建以来已使用了多少CPU时间。然后它计算各个进程应获得的CPU时间，即自创建以来的时间除以n。由于各个进程实际获得的CPU时间是已知的，所以很容易计算出真正获得的CPU时间和应获得的CPU时间之比。比率为0.5说明一个进程只获得了应得时间的一半，而比率为2.0则说明它获得了应得时间的2倍。于是该算法随后转向比率最低的进程，直到该进程的比率超过它的最接近竞争者为止。

6. 彩票调度

给用户一个保证，然后兑现之，这是个好想法，不过很难实现。但是，有一个既可给出类似预测结果而又有非常简单的实现方法的算法，这个算法称为*彩票调度*（lottery scheduling）（Waldspurger和Weihl，1994）。

其基本思想是向进程提供各种系统资源（如CPU时间）的彩票。一旦需要做出一项调度决策时，就随机抽出一张彩票，拥有该彩票的进程获得该资源。在应用到CPU调度时，系统可以掌握每秒钟50次的一种彩票，作为奖励每个获奖者可以得到20ms的CPU时间。

为了说明George Orwell关于“所有进程是平等的，但是某些进程更平等一些”的含义，可以给更重要的进程额外的彩票，以便增加它们获胜的机会。如果出售了100张彩票，而有一个进程持有其中的20张，那么在每一次抽奖中该进程就有20%的取胜机会。在较长的运行中，该进程会得到20%的CPU。相反，对于优先级调度程序，很难说明拥有优先级40究竟是什么意思，而这里的规则很清楚：拥有彩票f份额的进程大约得到系统资源的f份额。

彩票调度具有若干有趣的性质。例如，如果有一个新的进程出现并得到一些彩票，那么在下一次的抽奖中，该进程会有同它持有彩票数量成比例的机会赢得奖励。换句话说，彩票调度是反应迅速的。

如果希望协作进程可以交换它们的彩票。例如，有一个客户进程向服务器进程发送消息后就被阻塞，该客户进程可以把它所有的彩票交给服务器，以便增加该服务器下次运行的机会。在服务器运行完成之后，该服务器再把彩票还给客户机，这样客户机又可以运行了。事实上，如果没有客户机，服务器根本就不需要彩票。

彩票调度可以用来解决用其他方法很难解决的问题。一个例子是，有一个视频服务器，在该视频服务器上若干进程正在向其客户提供视频流，每个视频流的帧速率都不相同。假设这些进程需要的帧速率分别是10、20和25帧/秒。如果给这些进程分别分配10、20和25张彩票，那么它们会自动地按照大致正确的比例（即10：20：25）划分CPU的使用。

7. 公平分享调度

到现在为止，我们假设被调度的都是各个进程自身，并不关注其所有者是谁。这样做的结果是，如果用户1启动9个进程而用户2启动1个进程，使用轮转或相同优先级调度算法，那么用户1将得到90%的