CPU时间，而用户2只得到10%的CPU时间。

为了避免这种情形，某些系统在调度处理之前考虑谁拥有进程这一因素。在这种模式中，每个用户分配到CPU时间的一部分，而调度程序以一种强制的方式选择进程。这样，如果两个用户都得到获得50% CPU时间的保证，那么无论一个用户有多少进程存在，每个用户都会得到应有的CPU份额。

作为一个例子，考虑有两个用户的一个系统，每个用户都保证获得50% CPU时间。用户1有4个进程A、B、C和D，而用户2只有1个进程E。如果采用轮转调度，一个满足所有限制条件的可能序列是：

A E B E C E D E A E B E C E D E …

另一方面，如果用户1得到比用户2两倍的CPU时间，我们会有

A B E C D E A B E C D E …

当然，大量其他的可能也存在，可以进一步探讨，这取决于如何定义公平的含义。

2.4.4 实时系统中的调度

实时系统是一种时间起着主导作用的系统。典型地，外部的一种或多种物理设备给了计算机一个刺激，而计算机必须在一个确定的时间范围内恰当地做出反应。例如，在CD播放器中的计算机获得从驱动器而来的位流，然后必须在非常短的时间间隔内将位流转换为音乐。如果计算时间过长，那么音乐就会听起来有异常。其他的实时系统例子还有，医院特别护理部门的病人监护装置、飞机中的自动驾驶系统以及自动化工厂中的机器人控制等。在所有这些例子中，正确的但是迟到的应答往往比没有还要糟糕。

实时系统通常可以分为硬实时（hard real time）和软实时（soft real time），前者的含义是必须满足绝对的截止时间，后者的含义是虽然不希望偶尔错失截止时间，但是可以容忍。在这两种情形中，实时性能都是通过把程序划分为一组进程而实现的，其中每个进程的行为是可预测和提前掌握的。这些进程一般寿命较短，并且极快地就运行完成。在检测到一个外部信号时，调度程序的任务就是按照满足所有截止时间的要求调度进程。

实时系统中的事件可以按照响应方式进一步分类为周期性（以规则的时间间隔发生）事件或非周期性（发生时间不可预知）事件。一个系统可能要响应多个周期性事件流。根据每个事件需要处理时间的长短，系统甚至有可能无法处理完所有的事件。例如，如果有m个周期事件，事件i以周期P_i发生，并需要C_i秒CPU时间处理一个事件，那么可以处理负载的条件是

$$\sum_{i=1}^{m}\frac{Ci}{Pi}\leqslant 1$$

满足这个条件的实时系统称为是可调度的。

作为一个例子，考虑一个有三个周期性事件的软实时系统，其周期分别是100ms、200ms和500ms。如果这些事件分别需要50ms、30ms和100 ms的CPU时间，那么该系统是可调度的，因为$0.5+0.15+0.2<1$。如果有第四个事件加入，其周期为1秒，那么只要这个事件是不超过每事件150ms的CPU时间，那么该系统就仍然是可调度的。在这个计算中隐含了一个假设，即上下文切换的开销很小，可以忽略不计。

实时系统的调度算法可以是静态或动态的。前者在系统开始运行之前作出调度决策；后者在运行过程中进行调度决策。只有在可以提前掌握所完成的工作以及必须满足的截止时间等全部信息时，静态调度才能工作。而动态调度算法不需要这些限制。这里我们只涉及一些特定的算法，而把实时多媒体系统留到第7章去讨论。

2.4.5 策略和机制

到目前为止，我们隐含地假设系统中所有进程分属不同的用户，并且，进程间相互竞争CPU。通常情况下确实如此，但有时也有这样的情况：一个进程有许多子进程并在其控制下运行。例如，一个数据库管理系统可能有许多子进程，每一个子进程可能处理不同的请求，或每一个子进程实现不同的功能（如请求分析，磁盘访问等）。主进程完全可能掌握哪一个子进程最重要（或最紧迫）而哪一个最不重要。但是，以上讨论的调度算法中没有一个算法从用户进程接收有关的调度决策信息，这就导致了调度程序很少能够做出最优的选择。

解决问题的方法是将调度机制（scheduling mechanism）与调度策略（scheduling policy）分离（著名的原则，Levin等人，1975），也就是将调度算法以某种形式参数化，而参数可以由用户进程填写。我