其中m是进程数，在本例中，$m = 3$。注意，C_i / P_i只是CPU被进程i使用的部分。就图7-13所示的例子而言，进程A用掉CPU的10/30，进程B用掉CPU的15/40，进程C用掉CPU的5/50。将这些分数加在一起为CPU的0.808，所以该系统是可调度的。

到目前为止我们假设每个影片流有一个进程，实际上，每个影片流可能有两个（或更多个）进程，例如，一个用于音频，一个用于视频。它们可能以不同的速率运行并且每一脉冲可能消耗不同数量的CPU时间。然而，将音频进程加入到系统中并没有改变一般模型，因为我们的全部假设是存在m个进程，每个进程以一个固定的频率运行，对每一CPU突发有固定的工作量要求。

在某些实时系统中，进程是可抢占的，在其他的系统中，进程是不可抢占的。在多媒体系统中，进程通常是可抢占的，这意味着允许有危险错过其最终时限的进程在正在运行的进程完成工作以前将其中断，然后当它完成工作之后，被中断的前一个进程再继续运行。这一行为只不过是多道程序设计，正如我们在前面已经看过的。我们要研究的是可抢占的实时调度算法，因为在多媒体系统中没有拒绝它们的理由并且它们比不可抢占的调度算法具有更好的性能。惟一要关心的是如果传输缓冲区在很少的几个突发中被填充，那么在最终时限到来之前该缓冲区应该是完全满的，这样它就可以在一次操作中传递给用户，否则就会引起颤动。

实时算法可以是静态的也可以是动态的。静态算法预先分配给每个进程一个固定的优先级，然后使用这些优先级做基于优先级的抢占调度。动态算法没有固定的优先级。下面我们将研究每种类型的一个例子。

7.5.3 速率单调调度

适用于可抢占的周期性进程的经典静态实时调度算法是*速率单调调度*（Rate Monotonic Scheduling，RMS）（Liu和Layland，1973）。它可以用于满足下列条件的进程：

1) 每个周期性进程必须在其周期内完成。
2) 没有进程依赖于任何其他进程。
3) 每一进程在一次突发中需要相同的CPU时间量。
4) 任何非周期性进程都没有最终时限。
5) 进程抢占即刻发生而没有系统开销。

前四个条件是合理的。当然，最后一个不是，但是该条件使系统建模更加容易。RMS分配给每个进程一个固定的优先级，优先级等于进程触发事件发生的频率。例如，必须每30ms运行一次（每秒33次）的进程获得的优先级为33，必须每40ms运行一次（每秒25次）的进程获得的优先级为25，必须每50ms运行一次（每秒20次）的进程获得的优先级为20。所以，优先级与进程的速率（每秒运行进程的次数）成线性关系，这正是为什么将其称为速率单调的原因。在运行时，调度程序总是运行优先级最高的就绪进程，如果需要则抢占正在运行的进程。Liu和Layland证明了在静态调度算法种类中RMS是最优的。

图7-14演示了在图7-13所示的例子中速率单调调度是如何工作的。进程A、B和C分别具有静态优先级33、25和20，这意味着只要A需要运行，它就可以运行，抢占任何当前正在使用CPU的其他进程。进程B可以抢占C，但不能抢占A。进程C必须等待直到CPU空闲才能运行。

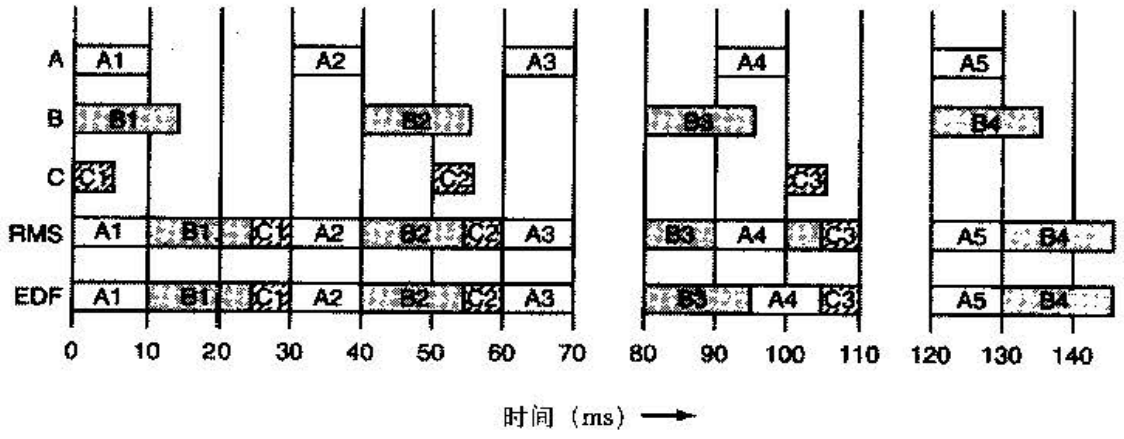

图7-14 RMS和EDF实时调度的一个例子