在图7-14中，最初所有三个进程都就绪要运行，优先级最高的进程A被选中，并准许它运行直到它在10ms时完成，如图7-14中的RMS一行所示。在进程A完成之后，进程B和C以先后次序运行。合起来，这些进程花费了30ms的时间运行，所以当C完成的时候，正是该A再次运行的时候。这一轮换持续进行直到t = 70时系统变为空闲。

在t = 80时，进程B就绪并开始运行。然而，在t = 90时，优先级更高的进程A变为就绪，所以它抢占B并运行，直到在t = 100时完成。在这一时刻，系统可以在结束进程B或者开始进程C之间进行选择，所以它选择优先级最高的进程B。

7.5.4 最早最终时限优先调度

另一个流行的实时调度算法是最早最终时限优先（Earliest Deadline First，EDF）算法。EDF是一个动态算法，它不像速率单调算法那样要求进程是周期性的。它也不像RMS那样要求每个CPU突发有相同的运行时间。只要一个进程需要CPU时间，它就宣布它的到来和最终时限。调度程序维持一个可运行进程的列表，该列表按最终时限排序。EDF算法运行列表中的第一个进程，也就是具有最近最终时限的进程。当一个新的进程就绪时，系统进行检查以了解其最终时限是否发生在当前运行的进程结束之前。如果是这样，新的进程就抢占当前正在运行的进程。

图7-14给出了EDF的一个例子。最初所有三个进程都是就绪的，它们按其最终时限的次序运行。进程A必须在t = 30之前结束，B必须在t = 40之前结束，C必须在t = 50之前结束，所以A具有最早的最终时限并因此而先运行。直到t = 90，选择都与RMS相同。在t = 90时，A再次就绪，并且其最终时限为t = 120，与B的最终时限相同。调度程序可以合理地选择其中任何一个运行，但是由于抢占B具有某些非零的代价与之相联系，所以最好是让B继续运行，而不去承担切换的代价。

为了消除RMS和EDF总是给出相同结果的想法，现在让我们看一看另外一个例子，如图7-15所示。在这个例子中，进程A、B和C的周期与前面的例子相同，但是现在A每次突发需要15ms的CPU时间，而不是只有10ms。可调度性测试计算CPU的利用率为0.500 + 0.375 + 0.100 = 0.975。CPU只留下了2.5%，但是在理论上CPU并没有被超额预定，找到一个合理的调度应该是可能的。

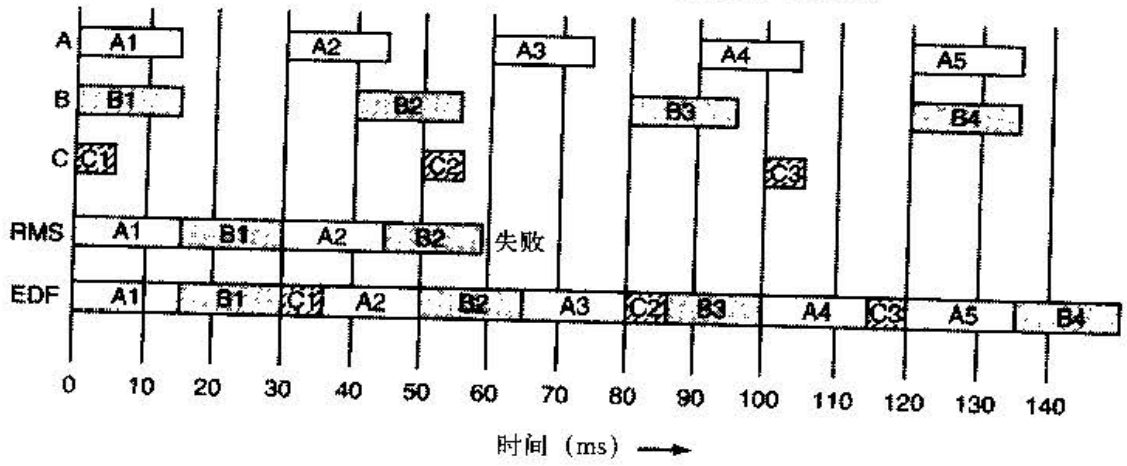

图7-15 以RMS和EDF进行实时调度的另一个例子

对于RMS，三个进程的优先级仍为33、25和20，因为优先级只与周期有关系，而与运行时间没有关系。这一次，进程B直到t = 30才结束，在这一时刻，进程A再次就绪要运行。等到A结束时，t = 45，此时B再次就绪，由于它的优先级高于C，所以B运行而C则错过了其最终时限。RMS失败。

现在看一看EDF如何处理这种情况。当t = 30时，在A2和C1之间存在竞争。因为C1的最终时限是50，而A2的最终时限是60，所以C被调度。这就不同于RMS，在RMS中A由于较高的优先级而成为赢家。

当t = 90时，A第四次就绪。A的最终时限与当前进程相同（同为120），所以调度程序面临抢占与否的选择。如前所述，如果不是必要最好不要抢占，所以B3被允许完成。

在图7-15所示的例子中，直到t = 150，CPU都是100%被占用的。然而，因为CPU只有97.5%被利用，所以最终将会出现间隙。由于所有开始和结束时间都是5ms的倍数，所以间隙将是5ms。为了获得要求