

- 死锁是一个相当困难的问题，因为它不总是可预测的。一些幸运的执行轨迹线将绕开死锁区域，而其他的将会陷入这个区域。图 12-44 展示了每种情况的一个示例。对于程序员来说，这其中隐含的着实令人惊慌。你可以运行一个程序 1000 次不出任何问题，但是下一次它就死锁了。或者程序在一台机器上可能运行得很好，但是在另外的机器上就会死锁。最糟糕的是，错误常常是不可重复的，因为不同的执行有不同的轨迹线。

程序死锁有很多原因，要避免死锁一般而言是很困难的。然而，当使用二元信号量来实现互斥时，如图 12-44 所示，你可以应用下面的简单而有效的规则来避免死锁：

**互斥锁加锁顺序规则：**给定所有互斥操作的一个全序，如果每个线程都是以一种顺序获得互斥锁并以相反的顺序释放，那么这个程序就是无死锁的。

例如，我们可以通过这样的方法来解决图 12-44 中的死锁问题：在每个线程中先对  $s$  加锁，然后再对  $t$  加锁。图 12-45 展示了得到的进度图。

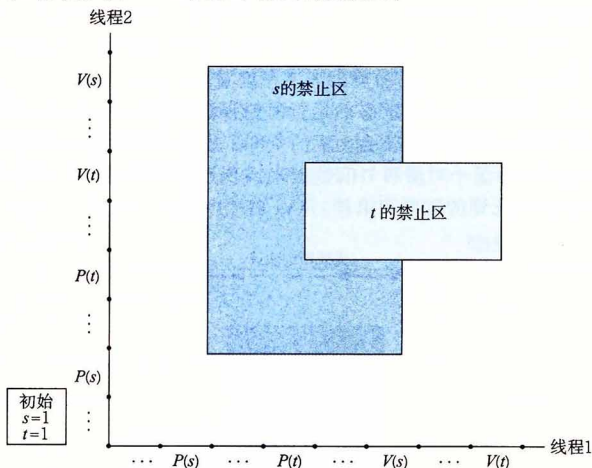



图 12-45 一个无死锁程序的进度图

 **练习题 12.15** 思考下面的程序，它试图使用一对信号量来实现互斥。

初始时:  $s = 1, t = 0$ .

线程1:	线程2:
$P(s);$	$P(s);$
$V(s);$	$V(s);$
$P(t);$	$P(t);$
$V(t);$	$V(t);$

- 画出这个程序的进度图。
- 它总是会死锁吗？
- 如果是，那么对初始信号量的值做哪些简单的改变就能消除这种潜在的死锁呢？
- 画出得到的无死锁程序的进度图。

## 12.8 小结

一个并发程序是由在时间上重叠的一组逻辑流组成的。在这一章中，我们学习了三种不同的构建并发程序的机制：进程、I/O 多路复用和线程。我们以一个并发网络服务器作为贯穿全章的应用程序。