


当我们在系统上运行这个程序时，现在得到了正确的结果：

```
linux> ./norace
Hello from thread 0
Hello from thread 1
Hello from thread 2
Hello from thread 3
```

 **练习题 12.13** 在图 12-43 中，我们可能想要在主线程中的第 14 行后立即释放已分配的内存块，而不是在对等线程中释放它。但是这会是个坏注意。为什么？

 **练习题 12.14**

- 在图 12-43 中，我们通过为每个整数 ID 分配一个独立的块来消除竞争。给出一个不调用 malloc 或者 free 函数的不同的方法。
- 这种方法的利弊是什么？

### 12.7.5 死锁

信号量引入了一种潜在的令人厌恶的运行时错误，叫做死锁(deadlock)，它指的是一组线程被阻塞了，等待一个永远也不会为真的条件。进度图对于理解死锁是一个无价的工具。例如，图 12-44 展示了一对用两个信号量来实现互斥的线程的进程图。从这幅图中，我们能够得到一些关于死锁的重要知识：

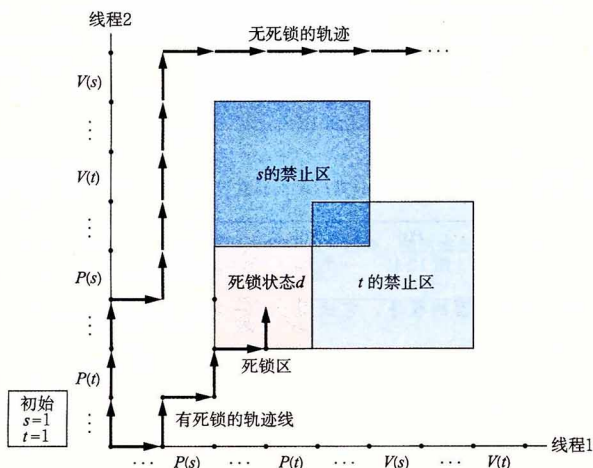


图 12-44 一个会死锁的程序的进度图

- 程序员使用  $P$  和  $V$  操作顺序不当，以至于两个信号量的禁止区域重叠。如果某个执行轨迹线碰巧到达了死锁状态  $d$ ，那么就不可能有进一步的进展了，因为重叠的禁止区域阻塞了每个合法方向上的进展。换句话说，程序死锁是因为每个线程都在等待其他线程执行一个根本不可能发生的  $V$  操作。
- 重叠的禁止区域引起了一组称为死锁区域(deadlock region)的状态。如果一个轨迹线碰巧到达了一个死锁区域中的状态，那么死锁就是不可避免的了。轨迹线可以进入死锁区域，但是它们不可能离开。