状态，叫做禁止区(forbidden region)，其中 $s<0$。因为信号量的不变性，没有实际可行的轨迹线能够包含禁止区中的状态。而且，因为禁止区完全包括了不安全区，所以没有实际可行的轨迹线能够接触不安全区的任何部分。因此，每条实际可行的轨迹线都是安全的，而且不管运行时指令顺序是怎样的，程序都会正确地增加计数器值。

从可操作的意义上来说，由 P 和 V 操作创建的禁止区使得在任何时间点上，在被包围的临界区中，不可能有多个线程在执行指令。换句话说，信号量操作确保了对临界区的互斥访问。

总的来说，为了用信号量正确同步图 12-16 中的计数器程序示例，我们首先声明一个信号量 mutex：

```
volatile long cnt = 0; /* Counter */
sem_t mutex;           /* Semaphore that protects counter */
```

然后在主例程中将 mutex 初始化为 1：

```
Sem_init(&mutex, 0, 1);  /* mutex = 1 */
```

最后，我们通过把在线程例程中对共享变量 cnt 的更新包围 P 和 V 操作，从而保护它们：

```
for (i = 0; i < niters; i++) {
    P(&mutex);
    cnt++;
    V(&mutex);
}
```

当我们运行这个正确同步的程序时，现在它每次都能产生正确的结果了。

```
linux>  ./goodcnt 1000000
OK cnt=2000000

linux>  ./goodcnt 1000000
OK cnt=2000000
```

旁注　进度图的局限性

进度图给了我们一种较好的方法，将在单处理器上的并发程序执行可视化，也帮助我们理解为什么需要同步。然而，它们确实也有局限性，特别是对于在多处理器上的并发执行，在多处理器上一组 CPU/高速缓存对共享同一个主存。多处理器的工作方式是进度图不能解释的。特别是，一个多处理器内存系统可以处于一种状态，不对应于进度图中任何轨迹线。不管如何，结论总是一样的：无论是在单处理器还是多处理器上运行程序，都要同步你对共享变量的访问。

12.5.4　利用信号量来调度共享资源

除了提供互斥之外，信号量的另一个重要作用是调度对共享资源的访问。在这种场景中，一个线程用信号量操作来通知另一个线程，程序状态中的某个条件已经为真了。两个经典而有用的例子是生产者-消费者和读者-写者问题。

1. 生产者-消费者问题

图 12-23 给出了生产者-消费者问题。生产者和消费者线程共享一个有 n 个槽的有限缓冲区。生产者线程反复地生成新的项目(item)，并把它们插入到缓冲区中。消费者线程不断地