12.7　其他并发问题

你可能已经注意到了，一旦我们要求同步对共享数据的访问，那么事情就变得复杂得多了。迄今为止，我们已经看到了用于互斥和生产者-消费者同步的技术，但这仅仅是冰山一角。同步从根本上说是很难的问题，它引出了在普通的顺序程序中不会出现的问题。这一小节是关于你在写并发程序时需要注意的一些问题的(非常不完整的)综述。为了让事情具体化，我们将以线程为例描述讨论。不过要记住，这些典型问题是任何类型的并发流操作共享资源时都会出现的。

12.7.1　线程安全

当用线程编写程序时，必须小心地编写那些具有称为*线程安全性*(thread safety)属性的函数。一个函数被称为*线程安全的*(thread-safe)，当且仅当被多个并发线程反复地调用时，它会一直产生正确的结果。如果一个函数不是线程安全的，我们就说它是*线程不安全的*(thread-unsafe)。

我们能够定义出四个(不相交的)线程不安全函数类：

*第 1 类：不保护共享变量的函数。*我们在图 12-16 的 thread 函数中就已经遇到了这样的问题，该函数对一个未受保护的全局计数器变量加 1。将这类线程不安全函数变成线程安全的，相对而言比较容易：利用像 P 和 V 操作这样的同步操作来保护共享的变量。这个方法的优点是在调用程序中不需要做任何修改。缺点是同步操作将减慢程序的执行时间。

*第 2 类：保持跨越多个调用的状态的函数。*一个伪随机数生成器是这类线程不安全函数的简单例子。请参考图 12-37 中的伪随机数生成器程序包。rand 函数是线程不安全的，因为当前调用的结果依赖于前次调用的中间结果。当调用 srand 为 rand 设置了一个种子后，我们从一个单线程中反复地调用 rand，能够预期得到一个可重复的随机数字序列。然而，如果多线程调用 rand 函数，这种假设就不再成立了。

code/conc/rand.c

```
1   unsigned next_seed = 1;
2
3   /* rand - return pseudorandom integer in the range 0..32767 */
4   unsigned rand(void)
5   {
6       next_seed = next_seed*1103515245 + 12543;
7       return (unsigned)(next_seed>>16) % 32768;
8   }
9
10  /* srand - set the initial seed for rand() */
11  void srand(unsigned new_seed)
12  {
13      next_seed = new_seed;
14  }
```

code/conc/rand.c

图 12-37　一个线程不安全的伪随机数生成器(基于[61])

使得像 rand 这样的函数线程安全的唯一方式是重写它，使得它不再使用任何 static 数据，而是依靠调用者在参数中传递状态信息。这样做的缺点是，程序员现在还要被迫修