改调用程序中的代码。在一个大的程序中，可能有成百上千个不同的调用位置，做这样的修改将是非常麻烦的，而且容易出错。

第 3 类：返回指向静态变量的指针的函数。某些函数，例如 ctime 和 gethostbyname，将计算结果放在一个 static 变量中，然后返回一个指向这个变量的指针。如果我们从并发线程中调用这些函数，那么将可能发生灾难，因为正在被一个线程使用的结果会被另一个线程悄悄地覆盖了。

有两种方法来处理这类线程不安全函数。一种选择是重写函数，使得调用者传递存放结果的变量的地址。这就消除了所有共享数据，但是它要求程序员能够修改函数的源代码。

如果线程不安全函数是难以修改或不可能修改的(例如，代码非常复杂或是没有源代码可用)，那么另外一种选择就是使用加锁-复制(lock-and-copy)技术。基本思想是将线程不安全函数与互斥锁联系起来。在每一个调用位置，对互斥锁加锁，调用线程不安全函数，将函数返回的结果复制到一个私有的内存位置，然后对互斥锁解锁。为了尽可能地减少对调用者的修改，你应该定义一个线程安全的包装函数，它执行加锁-复制，然后通过调用这个包装函数来取代所有对线程不安全函数的调用。例如，图 12-38 给出了 ctime 的一个线程安全的版本，利用的就是加锁-复制技术。

code/conc/ctime-ts.c

```
1  char *ctime_ts(const time_t *timep, char *privatep)
2  {
3      char *sharedp;
4
5      P(&mutex);
6      sharedp = ctime(timep);
7      strcpy(privatep, sharedp); /* Copy string from shared to private */
8      V(&mutex);
9      return privatep;
10 }
```

code/conc/ctime-ts.c

图 12-38 C 标准库函数 ctime 的线程安全的包装函数。使用加锁-复制技术调用一个第 3 类线程不安全函数

第 4 类：调用线程不安全函数的函数。如果函数 f 调用线程不安全函数 g，那么 f 就是线程不安全的吗？不一定。如果 g 是第 2 类函数，即依赖于跨越多次调用的状态，那么 f 也是线程不安全的，而且除了重写 g 以外，没有什么办法。然而，如果 g 是第 1 类或者第 3 类函数，那么只要你用一个互斥锁保护调用位置和任何得到的共享数据，f 仍然可能是线程安全的。在图 12-38 中我们看到了一个这种情况很好的示例，其中我们使用加锁-复制编写了一个线程安全函数，它调用了一个线程不安全的函数。

12.7.2 可重入性

有一类重要的线程安全函数，叫做可重入函数(reentrant function)，其特点在于它们具有这样一种属性：当它们被多个线程调用时，不会引用任何共享数据。尽管线程安全和可重入有时会(不正确地)被用做同义词，但是它们之间还是有清晰的技术差别，值得留意。图 12-39 展示了可

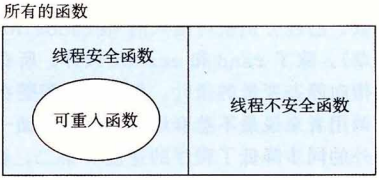

图 12-39 可重入函数、线程安全函数和线程不安全函数之间的集合关系