

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。所有函数的集合被划分成不相交的线程安全和线程不安全函数集合。可重入函数集合是线程安全函数的一个真子集。

可重入函数通常要比不可重入的线程安全的函数高效一些，因为它们不需要同步操作。更进一步来说，将第 2 类线程不安全函数转化为线程安全函数的唯一方法就是重写它，使之变为可重入的。例如，图 12-40 展示了图 12-37 中 `rand` 函数的一个可重入的版本。关键思想是我们用一个调用者传递进来的指针取代了静态的 `next` 变量。

```
code/conc/rand-r.c
```

```

1  /* rand_r - return a pseudorandom integer on 0..32767 */
2  int rand_r(unsigned int *nextp)
3  {
4      *nextp = *nextp * 1103515245 + 12345;
5      return (unsigned int)(*nextp / 65536) % 32768;
6  }

```


```
code/conc/rand-r.c
```

图 12-40 `rand_r`: 图 12-37 中的 `rand` 函数的可重入版本

检查某个函数的代码并先验地断定它是可重入的，这可能吗？不幸的是，不一定能这样。如果所有的函数参数都是传值传递的（即没有指针），并且所有的数据引用都是本地的自动栈变量（即没有引用静态或全局变量），那么函数就是显式可重入的（explicitly reentrant），也就是说，无论它是被如何调用的，都可以断言它是可重入的。

然而，如果把假设放宽松一点，允许显式可重入函数中一些参数是引用传递的（即允许它们传递指针），那么我们就得到了一个隐式可重入的（implicitly reentrant）函数，也就是说，如果调用线程小心地传递指向非共享数据的指针，那么它是可重入的。例如，图 12-40 中的 `rand_r` 函数就是隐式可重入的。

我们总是使用术语可重入的（reentrant）既包括显式可重入函数也包括隐式可重入函数。然而，认识到可重入性有时既是调用者也是被调用者的属性，并不只是被调用者单独的属性是非常重要的。

 **练习题 12.12** 图 12-38 中的 `ctime_ts` 函数是线程安全的，但不是可重入的。请解释说明。

12.7.3 在线程化的程序中使用已存在的库函数

大多数 Linux 函数，包括定义在标准 C 库中的函数（例如 `malloc`、`free`、`realloc`、`printf` 和 `scanf`）都是线程安全的，只有一小部分是例外。图 12-41 列出了常见的例外。（参考[110]可以得到一个完整的列表。）`strtok` 函数是一个已弃用的（不推荐使用）函数。`asctime`、`ctime` 和 `localtime` 函数是在不同时间和数据格式间相互来回转换时经常使用的函数。`gethostbyname`、`gethostbyaddr` 和 `inet_ntoa` 函数是已弃用的网络编程函数，已经分别被可重入的 `getaddrinfo`、`getnameinfo` 和 `inet_ntop` 函数取代（见第 11 章）。除了 `rand` 和 `strtok` 以外，所有这些线程不安全函数都是第 3 类的，它们返回一个指向静态变量的指针。如果我们需要在一个线程化的程序中调用这些函数中的某一个，对调用者来说最不惹麻烦的方法是加锁-复制。然而，加锁-复制方法有许多缺点。首先，额外的同步降低了程序的速度。第二，像 `gethostbyname` 这样的函数返回指向复杂结构的结构的指针，要复制整个结构层次，需要深层复制（deep copy）结构。第三，加锁-复制方法对像 `rand` 这样依赖跨越调用的静态状态的第 2 类函数并不有效。