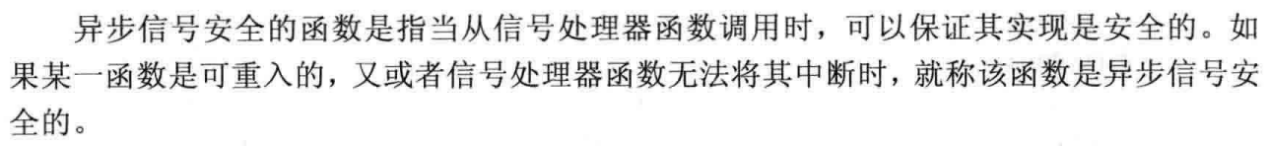
第三十一章

1. 回顾了一下信号那一章的可重入和异步信号安全函数。简单说一下。

什么是可重入？如果同一个进程的多个线程可以同时安全的调用某个函数，那么该函数就可以重入的。那么什么样的类型是不可重入的呢？

1. 更新全局变量或静态数据结构的函数是不可重入的。如malloc和free函数，里面维护着全局链表。
2. 使用了静态分配的内存返回信息。如crypt函数。
3. 将静态数据结构用于内部函数记账的也不可重入。如printf函数，它们维护缓冲区内部结构。



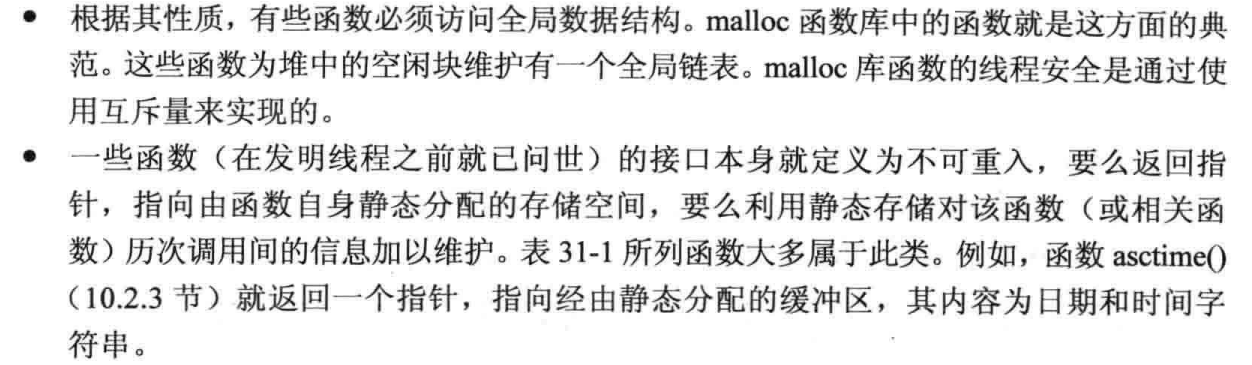
1. 线程安全：

函数可同时供多个线程安全调用，则可称为线程安全函数。反之，如果不是线程安全的，则不能并发调用。导致线程不安全的典型原因：使用了所有线程之间共享的全局变量或静态变量。对此，方法是：针对函数使用互斥量和针对临界区使用互斥量。

介绍了许多非线程安全的函数。

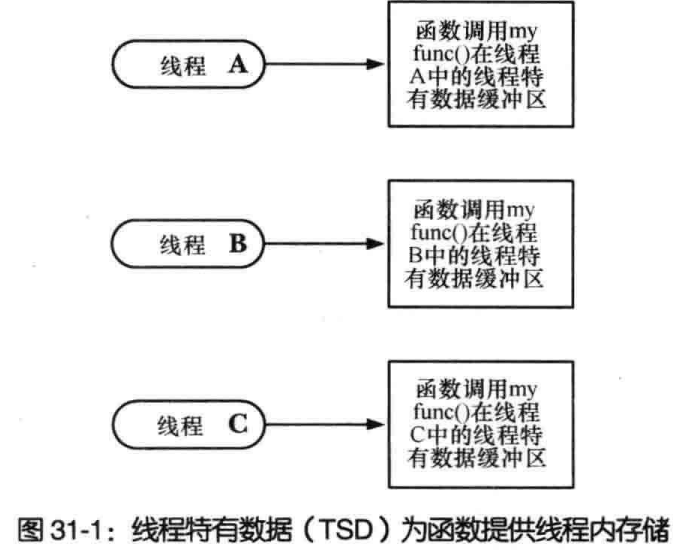
可重入函数无需使用互斥量即可实现线程安全，要诀在于避免对全局变量和静态变量的使用。需要返回给用户的信息和需要维护的信息都存储于调用者分配的缓冲区当中。对于一些不可重入的函数，SUSv3定义了\_r结尾的可重入“替身”。

不过仍然有许多函数由于某些原因不可重入：



1. 线程特有数据：

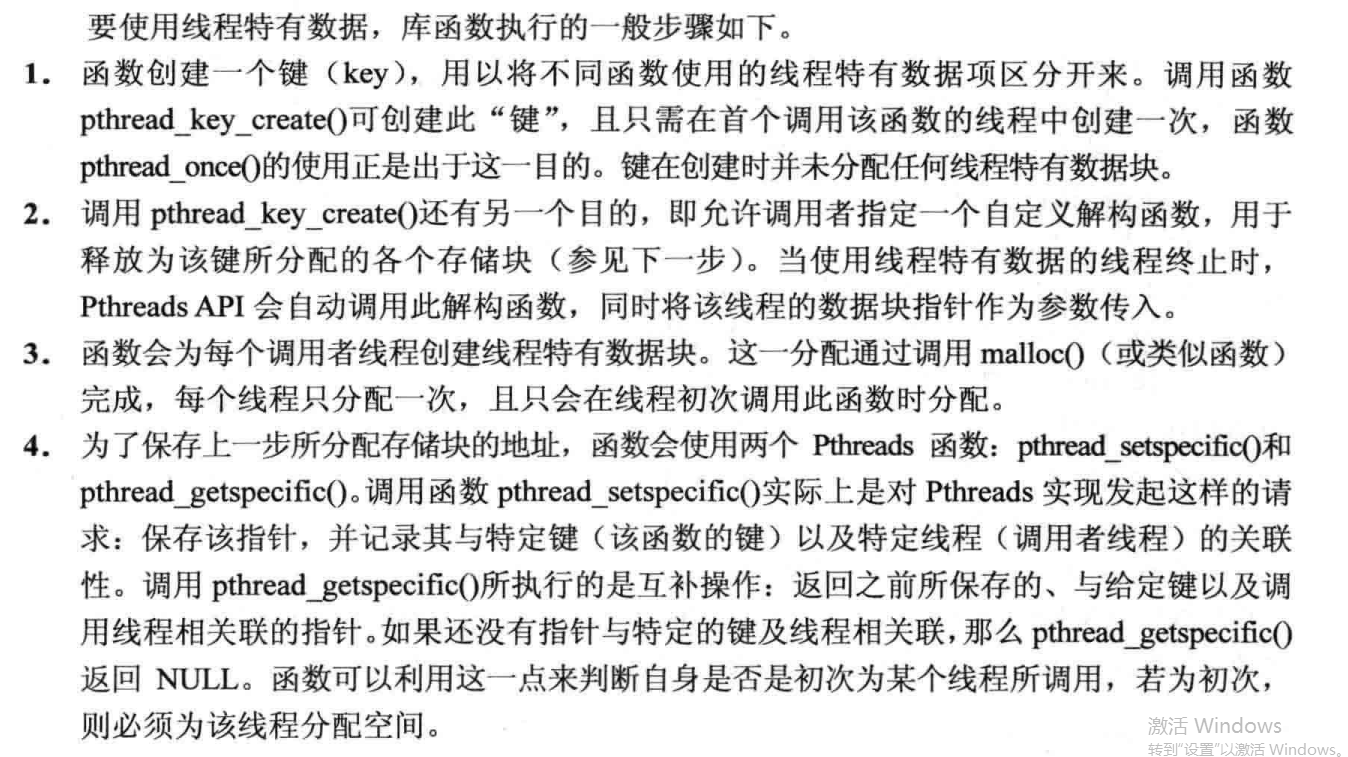
来源：实现函数线程安全最为有效的方法是使其可重入。但是这需要修改函数接口。使用线程特有数据，无需修改接口，即可实现可重入函数，让使用了这些函数的程序不需改变。简单来说，使用线程特有数据改写的库函数，本质上要实现的东西是每个线程为需要返回给用户程序的变量申请一个属于每个线程的存储。基本原理如下图：



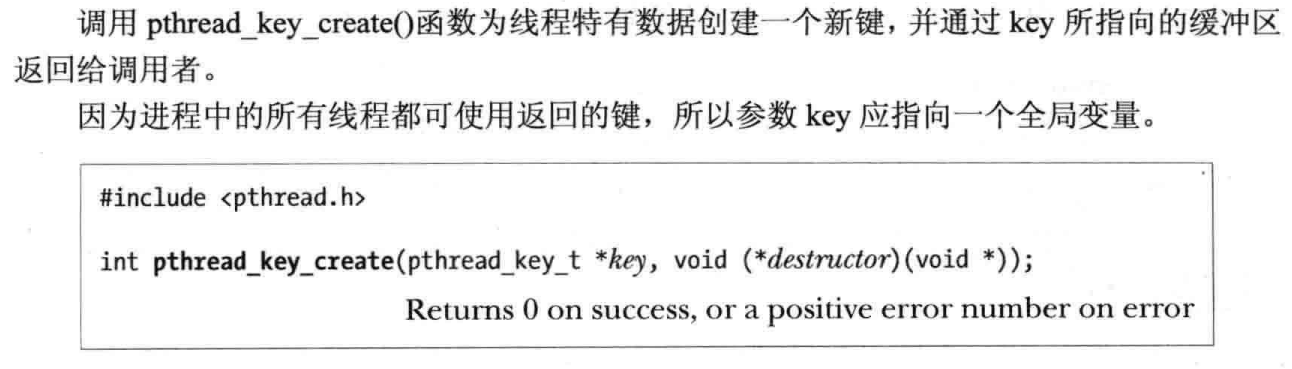
1. 库函数视角下的线程特有数据：

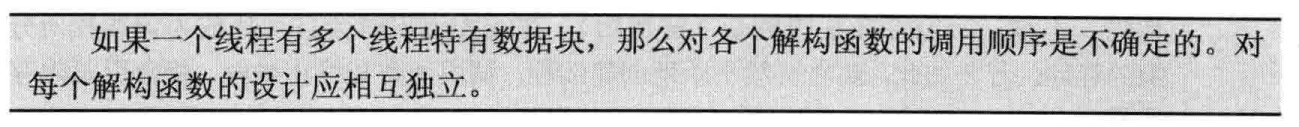
要在一个库函数下面使用线程特有数据这一技术，从这个库函数来看，需要做些什么呢？

1. 必须为每个线程分配单独的存储，且只需要初次调用的时候分配即可。
2. 此库函数需要获取这个初次分配的存储。由线程特有数据相关API提供。
3. 为此，需要特有的键获取及区别不同的使用这一技术的库函数。
4. 必须要有某些机制在线程结束时释放由线程分配的这些存储。
5. 线程特有数据API概述：

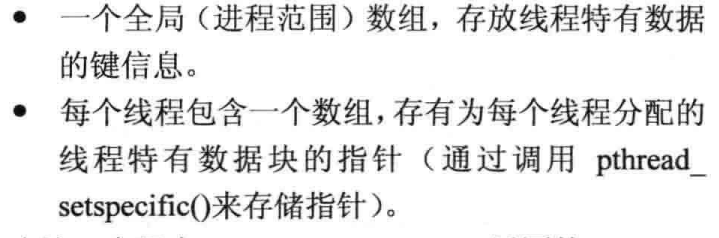


1. 线程特有数据API详述：

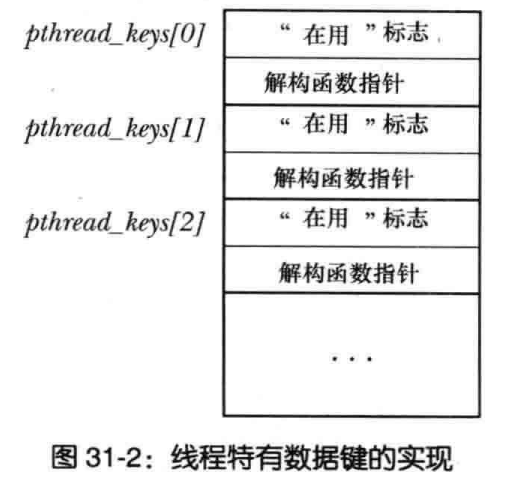


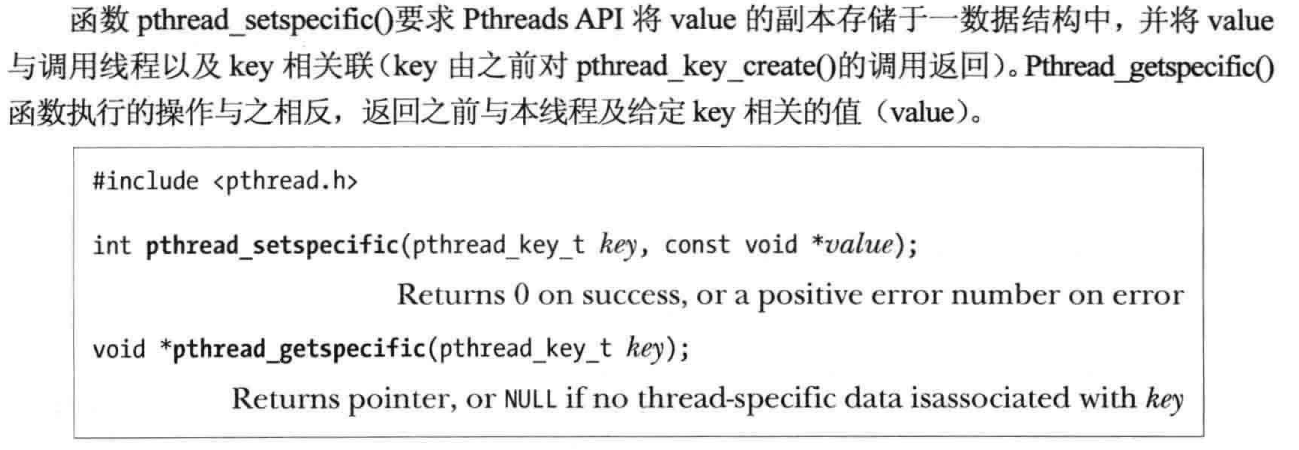


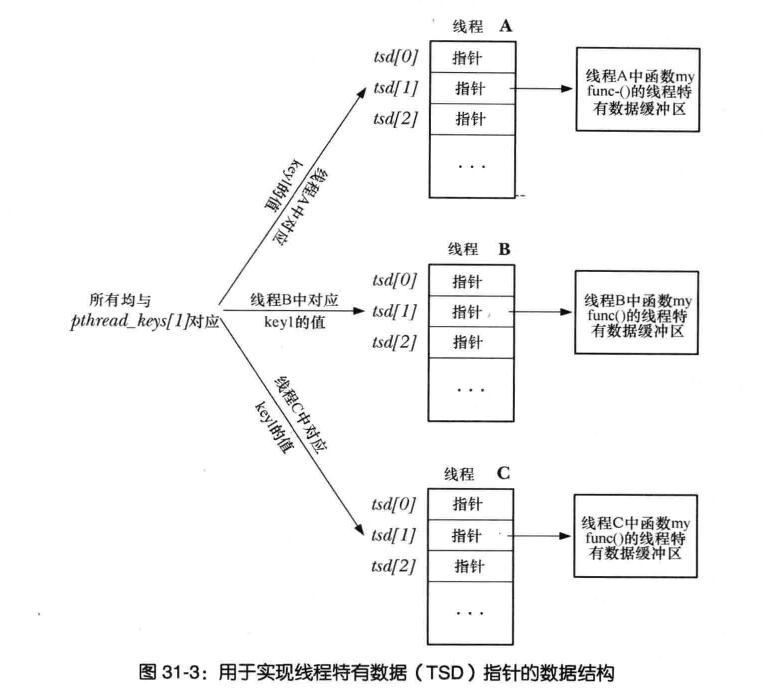
线程特有数据的典型实现会包含如下数据：



由pthread\_key\_create返回的key是作为全局数组的索引。







也就说，每个库函数要申请一个key，且只能申请一次，可以使用pthread\_once函数。根据这个key可以找到全局数组，里面记录了destructor。每一个全局数组对应于各个线程里的局部指针数组，指向各个函数申请的空间。这些空间也是只能申请一次，使用pthread\_getspecific返回值判断。我猜测，每当一个线程退出的时候，各个局部指针数组就会通过key找到相对应的destructor，销毁申请的空间。

1. 使用线程特有数据API：

介绍了使用线程特有数据的使用。很好的例子，可以看看。

1. 线程特有数据实现的限制：

对数据键的数量有所限制。不过已经绰绰有余了。

1. 线程局部存储：

类似于线程特有数据，线程局部存储提供了持久的每线程存储。是非标准实现。

