**一、线程间私有和共享的资源**

进程和线程的区别是什么？共享资源意味着什么？共享资源这种机制是如何实现的？线程之间共享了哪些进程资源？

进程是操作系统分配资源的基本单位，现成是调度的基本单位，现成之间共享进程资源。

线程运行的本质其实就是函数的执行，CPU从入口函数开始执行从而形成一个执行流，只不过我们人为的给执行流起一个名字，这个名字就叫线程。

**（1）线程私有数据区：线程上下文**

函数运行时的信息保存在栈帧中，栈帧中保存了函数的返回值、调用其它函数的参数、该函数使用的局部变量以及该函数使用的寄存器信息。此外，CPU执行指令的信息保存在一个叫做程序计数器的寄存器中，通过这个寄存器我们就知道接下来要执行哪一条指令。由于操作系统随时可以暂停线程的运行，因此我们保存以及恢复程序计数器中的值就能知道线程是从哪里暂停的以及该从哪里继续运行了。

由于线程运行的本质就是函数运行，函数运行时信息是保存在栈帧中的，因此每个线程都有自己独立的、私有的栈区。同时函数运行时需要额外的寄存器来保存一些信息，像部分局部变量之类，这些寄存器也是线程私有的，一个线程不可能访问到另一个线程的这类寄存器信息。

从上面的讨论中我们知道，到目前为止，所属线程的栈区、程序计数器、栈指针以及函数运行使用的寄存器是线程私有的。以上这些信息有一个统一的名字，就是线程上下文，thread context。我们也说过操作系统调度线程需要随时中断线程的运行并且需要线程被暂停后可以继续运行，操作系统之所以能实现这一点，依靠的就是线程上下文信息。

**补充线程私有数据技术**：线程局部存储，Thread Local Storage，TLS。

所谓线程局部存储，是指存放在该区域中的变量有两个含义：

存放在该区域中的变量是全局变量，所有线程都可以访问；

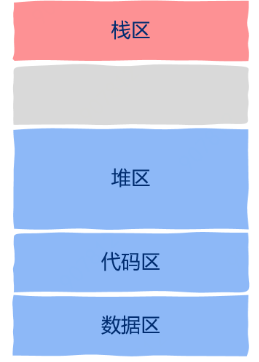
虽然看上去所有线程访问的都是同一个变量，但该全局变量独属于一个线程，一个线程对此变量的修改对其他线程不可见。

\_\_thread int a = 1; // 线程局部存储

全局变量a前面加了一个\_\_thread关键词用来修饰，也就是说我们告诉编译器把变量a放在线程局部存储中。

线程局部存储可以让你使用一个独属于线程的全局变量。也就是说，虽然该变量可以被所有线程访问，但该变量在每个线程中都有一个副本，一个线程对改变量的修改不会影响到其它线程。

**（2）线程共享资源**



这其实就是进程地址空间的样子，也就是说线程共享进程地址空间中除线程上下文信息中的所有内容，意思就是说线程可以直接读取这些内容。

1. **代码区**

进程地址空间中的代码区，保存的就是我们写的代码，更准确的是编译后的可执行机器指令。那么这些机器指令又是从哪里来的呢？答案是从可执行文件中加载到内存的，可执行程序中的代码区就是用来初始化进程地址空间中的代码区的。线程之间共享代码区，这就意味着程序中的任何一个函数都可以放到线程中去执行，不存在某个函数只能被特定线程执行的情况。

1. **数据区**

进程地址空间中的数据区，这里存放的就是所谓的全局变量。所谓全局变量就是那些你定义在函数之外的变量。

在程序员运行期间，也就是run time，数据区中的全局变量有且仅有一个实例，所有的线程都可以访问到该全局变量。

注意：值得注意的是，在C语言中还有一类特殊的“全局变量”，那就是用static关键词修饰过的变量。static修饰的全局变量的作用域被限制在定义它的文件内部，只能在该文件中访问。

1. **堆区**

在C/C++中用malloc或者new出来的数据就存放在这个区域，很显然，只要知道变量的地址，也就是指针，任何一个线程都可以访问指针指向的数据，因此堆区也是线程共享的属于进程的资源。

1. **栈区**

从线程这个抽象的概念上来说，栈区是线程私有的，然而从实际的实现上看，栈区属于线程私有这一规则并没有严格遵守。不像进程地址空间之间的严格隔离，线程的栈区没有严格的隔离机制来保护，因此如果一个线程能拿到来自另一个线程栈帧上的指针，那么该线程就可以改变另一个线程的栈区，也就是说这些线程可以任意修改本属于另一个线程栈区中的变量。

1. **动态链接库**

编译器在将可执行程序翻译成机器指令后，接下来还有一个重要的步骤，这就是链接，链接完成后生成的才是可执行程序。完成链接这一过程的就是链接器。

其中链接器可以有两种链接方式，这就是静态链接和动态链接。静态链接的意思是说把所有的机器指令一股脑全部打包到可执行程序中，动态链接的意思是我们不把动态链接的部分打包到可执行程序，而是在可执行程序运行起来后去内存中找动态链接的那部分代码，这就是所谓的静态链接和动态链接。动态链接一个显而易见的好处就是可执行程序的大小会很小。而动态链接的部分生成的库就是我们熟悉的动态链接库，在Windows下是以DLL结尾的文件，在Linux下是以so结尾的文件。

如果一个程序是动态链接生成的，那么其地址空间中有一部分包含的就是动态链接库，否则程序就运行不起来了，这一部分的地址空间也是被所有线程所共享的。

1. **打开的文件**

如果程序在运行过程中打开了一些文件，那么进程地址空间中还保存有打开的文件信息，进程打开的文件也可以被所有的线程使用，这也属于线程间的共享资源。

**二、线程安全代码编写方法**

我们说一段代码是线程安全的，当且仅当我们在多个线程中同时且多次调用的这段代码都能给出正确的结果，这样的代码我们才说是线程安全代码，Thread Safety，否则就不是线程安全代码，thread-unsafe.。

线程使用这些共享资源必须要遵守秩序，这个秩序的核心就是对共享资源的使用不能妨碍到其它线程，无论你使用各种锁也好、信号量也罢，其目的都是在维护公共场所的秩序。

线程私有的栈区多个线程使用是没有问题的；代码区和动态链接库这两个区域是不能被修改的，也就是说这两个区域是只读的，因此多个线程使用是没有问题的。

关于线程安全的一切问题全部围绕着线程私有数据与线程共享数据来处理，抓住了线程私有资源和共享资源这个主要矛盾也就抓住了解决线程安全问题的核心。

1. **只使用线程私有资源：**

不依赖任何全局变量，不依赖任何函数参数，且使用的局部变量都是线程私有资源，这样的代码也被称为无状态函数，stateless，很显然这样的代码是线程安全的。

1. **线程私有资源+函数参数：**

**①按值传参**：

按值传入的这些参数是线程私有资源。安全。

**②按引用传参：**

如果调用该函数的线程传入的参数是线程私有资源，那么该函数依然是线程安全的，但如果传入的参数是全局变量，那此时func函数将不再是线程安全代码，因为传入的参数指向了全局变量，这个全局变量是所有线程可共享资源，这种情况下如果不改变全局变量的使用方式，那么对该全局变量的加1操作必须施加某种秩序，比如加锁。

如果我传入的不是全局变量的指针(引用)是不是就不会有问题了？答案依然是it depends，要看情况。即便我们传入的参数是在堆上(heap)用malloc或new出来的，依然可能会有问题，为什么？答案很简单，因为堆上的资源也是所有线程可共享的。假如有两个线程调用func函数时传入的指针(引用)指向了同一个堆上的变量，那么该变量就变成了这两个线程的共享资源，在这种情况下func函数依然不是线程安全的。

改进也很简单，那就是每个线程调用func函数传入一个独属于该线程的资源地址，这样各个线程就不会妨碍到对方了，因此，写出线程安全代码的一大原则就是能用线程私有的资源就用私有资源，线程之间尽最大可能不去使用共享资源。

1. **使用全局资源：**

如果使用的全局资源只在程序运行时初始化一次，此后所有代码对其使用都是只读的，那么没有问题。否则func函数就不再是线程安全的了，对全局变量的修改必须加锁保护。

1. **线程局部存储：**

我们看到全局变量global\_num前加了关键词\_\_thread修饰，这时，func代码就是又是线程安全的了。意思是说这个变量是线程私有的全局变量：

global\_num是全局变量

global\_num是线程私有的

各个线程对global\_num的修改不会影响到其它线程，因为是线程私有资源，因此func函数是线程安全的。

1. **函数返回值：**

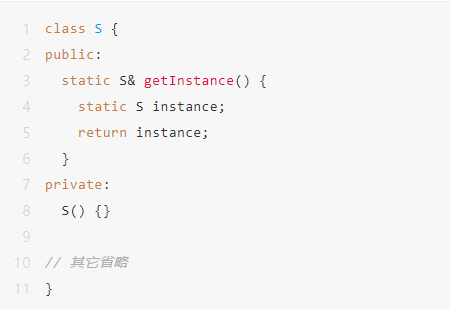
**①返回的是值**：

是线程安全的

**②返回的是引用：**

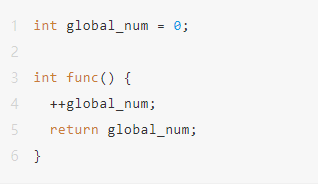
在使用该变量前其值可能已经被其它线程修改了。例如，因为该函数使用了一个静态全局变量，只要能拿到该变量的地址那么所有线程都可以修改该变量的值，因为这是线程间的共享资源，

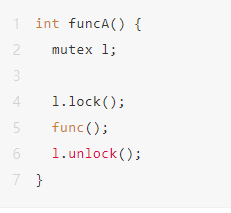
但是，请注意，有一个特例，这种使用方法可以用来实现设计模式中的单例模式，就像这样：



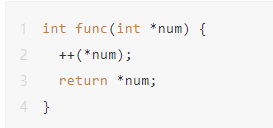
为什么呢？因为无论我们调用多少次func函数，static局部变量都只会被初始化一次，这种特性可以很方便的让我们实现单例模式

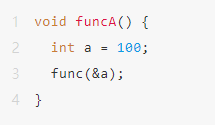
1. **调用非线程安全代码**





虽然func函数是非线程安全的，但是我们在调用该函数前加了一把锁进行保护，那么这时funcA函数就是线程安全的了，其本质就是我们用一把锁间接的保护了全局变量。





因为传入的参数是线程私有的局部变量，无论多少线程调用funcA都不会干扰到其它线程。

1. **如何实现线程安全**

①不使用任何全局资源，只使用线程私有资源，这种通常被称为无状态代码

②线程局部存储，如果要使用全局资源，是否可以声明为线程局部存储，因为这种变量虽然是全局的，但每个线程都有一个属于自己的副本，对其修改不会影响到其它线程

③只读，如果必须使用全局资源，那么全局资源是否可以是只读的，多线程使用只读的全局资源不会有线程安全问题。

④原子操作，原子操作是说其在执行过程中是不可能被其它线程打断的，像C++中的std::atomic修饰过的变量，对这类变量的操作无需传统的加锁保护，因为C++会确保在变量的修改过程中不会被打断。我们常说的各种无锁数据结构通常是在这类原子操作的基础上构建的 。

⑤同步互斥，到这里也就确定了你必须要以某种形式使用全局资源，那么在这种情况下公共场所的秩序必须得到维护，那么怎么维护呢？通过同步或者互斥的方式，这是一大类问题，我们将在《[深入理解操作系统](https://mp.weixin.qq.com/mp/appmsgalbum?__biz=MzU2NTYyOTQ4OQ==&action=getalbum&album_id=1433368223499796481" \l "wechat_redirect&__biz=MzU2NTYyOTQ4OQ==#wechat_redirect" \t "https://mp.weixin.qq.com/_blank)》系列文章中详细阐述这一问题。