POSIX多线程程序设计：概述

术语定义

线程：是指机器中连续的、顺序的属性集合，一个线程是能够被一个调度器单独管理的最小的程序片段。一个线程包含执行一系列指令所必须的机器状态，包括当前指令位置、地址和数据寄存器等。

进程：是正在执行的计算机程序的实例。它包含程序代码及其活动。进程可以由多个执行线程组成，且这些执行线程可以同时执行。

线程 VS 进程

- 进程通常是独立的，而线程是作为进程的子集的存在;
- 进程能够比线程携带更多的状态信息，而在进程中的多个线程共享着内存和其他资源等进程状态;
- 进程拥有分开的地址空间，而线程共享着地址空间;
- 进程只能通过系统提供的进程间通信(IPC)机制进行数据交流;
- 在一个进程中，线程间的上下文切换通常比进程间的上下文切换要快;

异步（asynchronous）：表明事情相互独立地发生，除非有强加的依赖性。

并发（concurrency）：让实际上可能串行发生的事情好像同时发生一样。并发操作之间可能任意交错，导致程序相互独立地运行（一个程序不必等到另一个结束后才开始运行），但是并发并不代表操作同时进行。

并行（parallelism）：指并发序列同时进行。并行的补充含义是指事情在相同的方向上独立运行（没有交错）。

> 真正的并行只能在多处理器系统中存在，但是并发可以在单处理器系统和多处理器系统中都存在。并发实际上是并行的假象。并行要求程序能够同时执行多个操作，而并发只要求程序能够假装同时执行多个操作。

线程安全和可重入：线程安全是指代码能够被多个线程调用而不会产生灾难性的后果（如，数据被破坏）。它不要求代码可以在多个线程中高效运行，只要求能够安全运行。

并发控制功能：并发系统必须提供基本的核心功能，包括创建并发执行的环境，并在你的库或应用程序中对其操作进行控制。任何并发系统都应该具备以下基本功能：

- 执行环境：是并发实体的状态。并发系统必须提供建立、删除执行环境和独立维护他们状态的方式。必须能够时不时保存好一个执行环境，而去执行另外一个环境。
- 调度：决定在某个给定时刻该执行哪个环境，并在不同的环境中切换。
- 同步：为并发执行的环境提供协调访问共享资源的一种机制。

  表 1.1 执行环境、调度和同步

环境	执行环境	调度	同步
UNIX（无线程）	进程	优先级	等待和管道
Pthreads	线程	调度策略、优先级	条件变量和互斥量

线程的好处

1. 在多处理器系统中开发程序的并行性。在多处理器系统中，线程模式可以让一个进程同时执行多个独立运算。
2. 在等待慢速外设I/O操作结束的同时，程序可以执行其他运算，为程序的并发提供更有效、更自然的开发方式。线程编程模式允许程序在等待像I/O之类的阻塞操作时继续其他计算。
3. 是一种模块化编程模型，能清晰表达程序中独立事件间的相互关系。
4. *响应性*：多线程可以允许应用程序保持对输入的响应。在单线程程序中，如果主执行线程阻塞长时间运行的任务，整个应用程序可能会冻结。通过将这些长时间运行的任务移动到与主执行线程同时运行的*工作线程*，应用程序可以在后台执行任务时保持对用户输入的响应。另一方面，在大多数情况下，多线程并不是保持程序响应的唯一方法，非阻塞I / O和/或Unix信号可用于获得类似的结果。[6]
5. *更快的执行*：多线程程序的这一优势使其能够在具有多个中央处理单元（CPU）或一个或多个多核处理器的计算机系统上运行，或者在一组机器上运行得更快，因为程序的线程自然会借出它们本身是并行执行，假设有足够的独立性（它们不需要彼此等待）。
6. *降低资源消耗*：使用线程，应用程序可以使用比使用自身的多个进程副本时所需的资源更少的资源同时为多个客户端提供服务。例如，Apache HTTP服务器使用线程池：用于侦听传入请求的侦听器线程池，以及用于处理这些请求的服务器线程池。
7. *更好的系统利用率*：例如，使用多个线程的文件系统可以实现更高的吞吐量和更低的延迟，因为更快的介质（例如高速缓存存储器）中的数据可以被一个线程检索而另一个线程从较慢的介质中检索数据（例如作为外部存储器，两个线程都没有等待另一个完成。
8. *简化共享和通信*：与需要消息传递或共享内存机制来执行进程间通信（IPC）的进程不同，线程可以通过它们已共享的数据，代码和文件进行通信。
9. *并行化*：希望使用多核或多CPU系统的应用程序可以使用多线程将数据和任务拆分为并行子任务，并让底层架构管理线程的运行方式，既可以在一个内核上同时运行，也可以在多个内核上并行运行。像CUDA和OpenCL这样的GPU计算环境使用多线程模型，其中数十到数百个线程并行运行在大量内核上的数据上。

线程的代价

1. 计算负荷：线程代码中的负荷代价包括由于线程间同步所导致的直接影响。在几乎任何线程代码中你都需要使用某种同步机制，使用太多的同步很容易损失性能。
2. 编程规则：编写能够在多个线程中良好工作的代码需要认真的思考和计划，需要明白同步协议和程序中的不变量（invariant），不得不避免死锁、竞争和优先级倒置。一个进程中的所有线程共享地址空间，线程间没有保护界限。如果一个线程使用未初始化的指针写内存，则可能破坏其他线程的堆栈或堆空间。
3. 更难以调试：调试不可避免地要改变事件的时序，这在调试串行代码时不会有什么大问题，但是在调试异步代码时却是致命的。如果一个线程因调试陷阱而运行的稍微变慢了，则你要跟踪的问题就可能不会再出现。
4. *同步*：由于线程共享相同的地址空间，程序员必须小心避免竞争条件和其他非直观行为。为了正确操作数据，线程通常需要及时会合以便以正确的顺序处理数据。线程还可能需要互斥操作（通常使用互斥锁实现），以防止在修改过程中同时修改或读取公共数据。不小心使用这些原语可能会导致死锁，活锁或种族上的资源。

5. *线程崩溃进程*：线程执行的非法操作会导致整个进程崩溃; 因此，一个行为不当的线程可能会破坏应用程序中所有其他线程的处理。

Pthreads简介

类型和接口

类型	描述
pthread_t	线程标识符
pthread_mutex_t	互斥量
pthread_cond_t	条件变量
pthread_key_t	线程私有权访问键
pthread_attr_t	线程属性对象
pthread_mutexattr_t	h互斥量属性对象
pthread_condattr_t	条件变量属性对象
pthread_once_t	“一次性初始化”控制变量

> 以上所有数据类型都是不透明的，可移植的代码不应该对以上数据类型的实现做任何假设。例如，线程标识符ID的具体实现可能是整型，或者是浮点型，或者是结构体类型。

错误检查

传统上的函数成功返回时，会返回一个有效值或者指示调用成功的0值。在错误发生时，会返回特定的-1值，并对全局变量errno赋值以指示错误类型。

pthreads函数有错时不会设置errno变量（而大部分其他POSIX函数会这样），而是通过返回值来表示错误状态。pthreads函数返回0，并包含一个额外的输出参数来指向存有“有用结果”的地址，当发生错误时，函数返回一个包含在<errno.h>头文件中的错误代码。pthreads同样提供了一个线程内的errno变量以支持其他使用errno的代码，即，当线程调用使用errno报错函数时，该变量值不会被其他线程重写或读取。

> pthreads报错机制中的一个例外是pthread_getspecific函数，它返回线程私有数据-共享key值。pthread_getspecific函数根本不报错，如果pthread_key_t值无效或者线程为对它赋值，则pthread_getspecific函数返回NULL。

参考资料

> https://en.wikipedia.org/wiki/Thread_(computing)