ϒ什么是并发和多线程

ϒ为什么要在应用程序中使用并发和多线程

ϒC++中支持并发的一些历史

ϒ简单的多线程C++程序是什么样子的

在最简单和最基本的层面上，并发是指两个或多个单独的活动同时发生。

无论它们具有多个处理器还是一个处理器内有多个内核（或两者兼而有之），这些计算机都能够真正并行运行多个任务。 我们称之为硬件并发

在应用程序中利用并发性的第一种方法是将应用程序划分为同时运行的多个独立的单线程进程，就像同时运行 Web 浏览器和文字处理器一样。

 一个缺点是进程之间的这种通信通常要么设置复杂，要么速度慢，或者两者兼而有之，因为操作系统通常在进程之间提供大量保护，以避免一个进程意外修改属于另一进程的数据。

另一个缺点是运行多个进程存在固有的开销：启动进程需要时间，操作系统必须投入内部资源来管理进程，等等。

并发的另一种方法是在单个进程中运行多个线程。 线程很像轻量级进程：每个线程独立于其他线程运行，并且每个线程可以运行不同的指令序列。 但是进程中的所有线程共享相同的地址空间，并且大多数数据可以直接从所有线程访问 - 全局变量保持全局，并且对象或数据的指针或引用可以在线程之间传递。 尽管通常可以在进程之间共享内存，但这设置起来很复杂并且通常难以管理，因为相同数据的内存地址在不同进程中不一定相同。

本书专门关注使用多线程实现并发，并且以后对并发的引用都假设这是通过使用多个线程来实现的。

在应用程序中使用并发有两个主要原因：关注点分离和性能。 事实上，我什至可以说它们几乎是使用并发的唯一原因； 当你足够努力地观察时，其他任何事情都可以归结为其中之一

通过使用多线程来分离这些问题，用户界面代码和 DVD 播放代码不再需要如此紧密地交织在一起； 一个线程可以处理用户界面，另一个线程可以处理 DVD 播放。 它们之间必须有交互，例如当用户单击“暂停”时，但现在这些交互与手头的任务直接相关。

* **吞吐量** 是指在单位时间内系统能够完成的任务数量或处理的数据量。
* 在多线程并发编程中，吞吐量通常与线程数量、CPU核心数量和任务类型相关。
* 如果线程的划分是基于概念设计而不是为了提高吞吐量，那么线程数量可能与CPU核心数量无关。
* 通过增加线程数量、减少锁竞争、任务并行化和负载均衡等策略，可以优化系统的吞吐量。

 第一个也是最明显的是将单个任务划分为多个部分并并行运行每个部分，从而减少总运行时间。 这就是任务并行。 虽然这听起来很简单，但它可能是一个相当复杂的过程，因为各个部分之间可能存在许多依赖关系。 这些划分可以是在处理方面（一个线程执行算法的一部分，而另一个线程执行不同部分）或在数据方面（每个线程对数据的不同部分执行相同的操作）。 后一种方法称为数据并行。对于算法中并行程度不高的部分，您可以将算法划分为固定数量（因此不可扩展）的并行任务。

使用并发来提高性能的第二种方法是使用可用的并行性来解决更大的问题； 不是一次处理一个文件，而是处理 2 个文件，或者10 或 20，视情况而定。 虽然这是数据并行的应用，但通过同时对多组数据执行相同的操作，有不同的侧重点。 处理一大块数据仍然需要相同的时间，但现在可以在相同的时间内处理更多的数据。 显然，这种方法存在局限性，并且并非在所有情况下都有好处，但是这种方法带来的吞吐量增加可以使新事物成为可能——例如，如果视频处理的不同区域提高了分辨率， 图片可以并行处理。

使用并发来提高性能就像任何其他优化策略一样：它有可能大大提高应用程序的性能，但它也会使代码复杂化，使其更难以理解并且更容易出现错误。

**C++标准库高级设施** 的性能问题，并指出在大多数情况下，标准库提供的额外功能不会影响性能（因为“你不需要为你不需要的东西付费”）。然而，在极少数情况下，这些额外的功能可能会影响性能。如果性能成为瓶颈，可以考虑使用底层API手动实现所需功能，但在大多数情况下，额外的复杂性和潜在的错误风险远远超过了微小的性能提升。