# 并发编程记录

1. 什么时候使用并发：关注点分离(SOC)和性能
2. 分类关注点的好处：通过将相关的代码与无关的代码分离，可以使程序更容易理解和测试，从而减少出错的可能性。
3. 什么时候需要一个任务切换框架：不显式地使用并发
4. 利用并发提高性能的两种方式：
   1. 将一个单个任务分成几部分，且各自并行运行，从而降低总运行时间。这就是任务并行（task parallelism）
   2. 每个线程在不同的数据部分上执行相同的操作，被称为数据并行（data parallelism）
5. 数据并行的应用：与其同时处理一个文件，不如酌情处理2个、10个或20个
6. 什么时候不适用并发：收益比不上成本
7. 原子操作库：可用于直接控制单个位、字节、内部线程间同步，以及所有变化的可见性
8. 并发设计应避免的问题：过多的线程竞争一个互斥单元，这将很明显影响性能
9. 如何访问平台提供的工具的同时，又使用标准c++线程库：在C++线程库中提供一个 native\_handle() 成员函数，允许通过使用平台相关API直接操作底层实现。就其本质而言，任何使用 native\_handle() 执行的操作是完全依赖于平台的
10. 并发程序与普通程序的区别：区别在于某些函数可以并发运行，所以需要确保共享数据在并发访问时是安全的。当然，为了并发地运行函数，必须使用特定的函数以及对象来管理各个线程
11. 线程退出的时机：线程执行完入口函数后，线程就会退出
12. 线程的创建和启动：构造std::thread对象，构造对象完成，意味着开始启动该线程
13. 守护线程【daemon threads】：通常称分离线程为守护线程，UNIX中守护线程是指，没有任何用户接口，并在后台运行的线程。这种线程的特点就是长时间运行；线程的生命周期可能会从某一个应用起始到结束，可能会在后台监视文件系统，还有可能对缓存进行清理，亦或对数据结构进行优化。另一方面，分离线程的另一方面只能确定线程什么时候结束，"发后即忘"(fire and forget)的任务就使用到线程的这种方式。