1. 多核、多CPU、超线程、多线程

1.1 为什么要多核

一个现代的CPU包含：处理器、寄存器、存储设备(L1L2缓存)、浮点运算单元、整数运算单元等一些辅助运算设备以及内部总线等。
为什么要用多核：

- 如果要跑一个多线程程序，因为是一个进程里的线程，所以需要一些共享一些存储变量，如果这台计算机是单核单线程CPU，那么不同的线程要经常要CPU之间的外部总线上通信，同时还要处理不同CPU之间不同缓存导致数据不一致的问题，所以多核单CPU架构就能发挥很大的优势，通信在内部总线，共用同一个缓存。

1.2 为什么要多CPU

如果要运行多个进程的话，假如只有一个CPU的话，就意味着要经常进行进程上下文切换，因为单CPU即便是多核的，也只是多个处理器核心，其他设备都是共用的，所以 多个进程就必然要经常进行进程上下文切换，这个代价是很高的。

1.3 为什么要超线程

- 超线程是Intel提出的，简单来说是在一个CPU上真正的并发两个线程。
- 一个CPU除了处理器核心还有其他设备，一段代码执行过程也不光是只有处理器核心工作，如果两个线程A和B，A正在使用处理器核心，B正在使用缓存或者其他设备，那AB两个线程就可以并发执行，但是如果AB都在访问同一个设备，那就只能等前一个线程执行完后一个线程才能执行。
- 实现这种并发的原理是 在CPU里加了一个协调辅助核心。

1.4 为什么要多线程

- 一个进程里多线程之间可以共享变量，线程间通信开销也较小，可以更好的利用多核CPU的性能，多核CPU上跑多线程程序往往会比单线程更快。
- 虽然多线程会有上下文切换和线程创建销毁开销，但是单线程程序会被IO阻塞无法充分利用CPU资源，加上线程的上下文开销较低以及线程池的大量应用，多线程在很多场景下都会有更高的效率。

1.5 线程与进程

- 进程是系统分配资源的基本单位，线程是调度CPU的基本单位。
- 进程是操作系统的管理单位，而线程则是进程的管理单位。
- 每个线程都有一个程序计数器（记录要执行的下一条指令），一组寄存器（保存当前线程的工作变量），堆栈（记录执行历史，其中每一帧保存了一个已经调用但未返回的过程）。
- 每个线程共享堆空间，拥有自己独立的栈空间。

2. 上下文切换

简单来说，在CPU处理多任务时，CPU从一个进程或线程切换到另一个进程或线程，这段过程就叫做上下文切换。
也就是说：CPU在利用**时间片轮转**的方式为每一个任务分配一个时间片段，当时间片段结束后，需要把当前任务的状态保存下来，然后再继续执行下一个任务，任务的状态保存及再加载, 这段过程就叫做上下文切换。

> 上下文：是指某一时间点 CPU 寄存器和程序计数器的内容。
> 寄存器：是 CPU 内部的数量较少但是速度很快的内存（与之对应的是 CPU 外部相对较慢的 RAM主内存）。寄存器通过对常用值（通常是运算的中间值）的快速访问来提高计算机程序运行的速度。
> 程序计数器是一个专用的寄存器，存的值为正在执行的指令的位置或者下一个将要被执行的指令的位置。

2.1 切换的步骤

上下文切换过程中的任务信息被保存在进程控制块中(PCB，process control block)。这些信息会一直保存在CPU内存中，直到他们被再次使用。
PCB通常是系统内存占用区中的一个连续存区，它存放着操作系统用于描述进程情况及控制进程运行所需的全部信息。其过程如下：

> 1. 保存进程A的状态（寄存器和操作系统数据）；
> 2. 更新PCB中的信息，对进程A的“运行态”做出相应更改；
> 3. 将进程A的PCB放入相关状态的队列；
> 4. 将进程B的PCB信息改为“运行态”，并执行进程B；
> 5. B执行完后，从队列中取出进程A的PCB，恢复进程A被切换时的上下文，继续执行A；

线程与进程的上下文切换也不同，进程上下文切换分为两步：

> 1. 切换页目录以使用新的地址空间；
> 2. 切换内核栈和硬件上下文；

对于linux来说，线程和进程最大的区别就在于地址空间。**对于线程切换，第一步是不用的，第2步是进程和线程切换都需要做的。** 所以明显是进程切换代价大。 线程上下文切换和进程上下文切换一个最主要的区别是**线程的切换虚拟内存空间依然是相同的，但是进程切换是不同的。**这两种上下文切换的处理都是 通过操作系统内核来完成的。内核的这种切换过程伴随的 最显著的性能损耗是将寄存器中的内容切换出。

其具体的过程如下：

对于一个正在执行的进程包括 程序计数器、寄存器、变量的当前值等 ，而这些数据都是 保存在CPU的寄存器中的，且这些寄存器只能是正在使用CPU的进程才能享用，在进程切换时，首先得保存上一个进程的这些数据（便于下次获得CPU的使用权时从上次的中断处开始继续顺序执行，而不是返回到进程开始，否则每次进程重新获得CPU时所处理的任务都是上一次的重复，可能永远也到不了进程的结束出，因为一个进程几乎不可能执行完所有任务后才释放CPU），然后将本次获得CPU的进程的这些数据装入CPU的寄存器从上次断点处继续执行剩下的任务。

3. 切换原因

引起上下文切换的原因主要有：

> 1. **中断处理**：在中断处理中，其他程序”打断”了当前正在运行的程序。当CPU接收到中断请求时，会在正在运行的程序和发起中断请求的程序之间进行一次上下文切换。中断分为硬件中断和软件中断，软件中断包括因为IO阻塞、未抢到资源或者用户代码等原因，线程被挂起。
> 2. **多任务处理**：在多任务处理中，CPU会在不同程序之间来回切换，每个程序都有相应的处理时间片，CPU在两个时间片的间隔中进行上下文切换。
> 3. **用户态切换**：对于一些操作系统，当进行用户态切换时也会进行一次上下文切换，虽然这不是必须的。

对于我们经常 使用的抢占式操作系统 而言，引起线程上下文切换的原因大概有以下几种：

> 1. 当前执行任务的时间片用完之后，系统CPU正常调度下一个任务；
> 2. 当前执行任务碰到IO阻塞，调度器将此任务挂起，继续下一任务；
> 3. 多个任务抢占锁资源，当前任务没有抢到锁资源，被调度器挂起，继续下一任务；
> 4. 用户代码挂起当前任务，让出CPU时间；
> 5. 硬件中断；

4. 切换损耗

上下文切换会带来 直接和间接 两种因素影响程序性能的消耗。

> 1. **直接消耗**：指的是CPU寄存器需要保存和加载, 系统调度器的代码需要执行, TLB实例需要重新加载, CPU 的pipeline需要刷掉；
> 2. **间接消耗**：指的是多核的cache之间得共享数据, 间接消耗对于程序的影响要看线程工作区操作数据的大小；

5. 减少切换

既然上下文切换会导致额外的开销，因此减少上下文切换次数便可以提高多线程程序的运行效率。但上下文切换又分为2种：

> 1. 让步式上下文切换：指执行线程主动释放CPU，与锁竞争严重程度成正比，可通过减少锁竞争来避免。
> 2. 抢占式上下文切换：指线程因分配的时间片用尽而被迫放弃CPU或者被其他优先级更高的线程所抢占，一般由于线程数大于CPU可用核心数引起，可通过调整线程数，适当减少线程数来避免。

所以，减少上下文切换的方法有无锁并发编程、CAS算法、使用最少线程和使用协程：

> 1. 无锁并发：多线程竞争时，会引起上下文切换，所以多线程处理数据时，可以用一些办法来避免使用锁，如将数据的ID按照Hash取模分段，不同的线程处理不同段的数据；
> 2. CAS算法：Java的Atomic包使用CAS算法来更新数据，而不需要加锁；
> 3. 最少线程：避免创建不需要的线程，比如任务很少，但是创建了很多线程来处理，这样会造成大量线程都处于等待状态；
> 4. 使用协程：在单线程里实现多任务的调度，并在单线程里维持多个任务间的切换；

6. 线程数量

1. 高并发，低耗时的情况：建议少线程，只要满足并发即可，因为上下文切换本来就多，并且高并发就意味着CPU是处于繁忙状态的， 增加更多地线程也不会让线程得到执行时间片，反而会增加线程切换的开销；例如并发100，线程池可能设置为10就可以；

2. 低并发，高耗时的情况：建议多线程，保证有空闲线程，接受新的任务；例如并发10，线程池可能就要设置为20；

3. 高并发高耗时：1. 要分析任务类型；2. 增加排队；3. 加大线程数；