Cpu某一时刻只能执行一个线程，为什么使用多线程，是为了通过并发提高处理任务的速度。（现在是一个逻辑cpu某一时刻只能执行一个线程了,cpu都是多核，核是完整的计算单元，不考虑超线程，一个逻辑cpu就是一个核）

## 1.首先理解一下什么叫并发什么并行。

并发的关键是你有处理多个任务的能力，不一定要同时。  
并行的关键是你有同时处理多个任务的能力。

并发是两个队列**交替**使用一台咖啡机，并行是两个队列**同时**使用两台咖啡机

（可以把咖啡机理解成cpu,队列就是线程）

2.首先，要实现多任务，通常我们会设计Master-Worker模式，Master负责分配任务，Worker负责执行任务，因此，多任务环境下，通常是一个Master，多个Worker。

如果用多进程实现Master-Worker，主进程就是Master，其他进程就是Worker。

如果用多线程实现Master-Worker，主线程就是Master，其他线程就是Worker。

读到这段话，有必要去理解一下，线程与进程，多线程与多进程.，以及他们的

优缺点。

2.1线程与进程，多线程与多进程.

进程是分配资源的最小单位，线程是调度和执行的最小单位。（一个进程可以含有多个线程）

以多进程形式处理多个任务，

以多线程方式处理单个任务的不同部分（这样理解感觉不太对，难道一个线程就不能处理一个任务。）  
—— 如果一个线程就完整处理一个任务，线程间不需要共享什么数据的话，在做在线服务server(不含一些离线数据处理的离线程序)，一般用多进程，多部署几个进程，这个在程序里没有线程同步等各种复杂的问题，编程简单

也是看情况开桌子吃菜的问题（客人不太需要互相敬酒，只顾吃自己的）且每个桌子人的菜都一样，那吃起来比较开心。

2.2多进程与多线程的区别：

多进程对于每个变量都有自己的拷贝，内存是独立的。

多线程是共享变量（资源），又因为线程的执行顺序是不确定的，对于需要读写的资源，会造成不能预知的结果，所以我们引入锁机制以及信号量机制解决这些问题。

补充：多线程环境下吗，每个线程都有自己的数据，一个线程使用自己的局部变量要比使用全局变量好，应为不会影响其他进程，使用全局变量一定要加锁。

thread\_local， 它用来指定一个变量是线程作用域级别的，各个线程对thread\_local的变量，持有自己的值，互不干扰。 但多线程程序，大部分是因为要处理共享进程级别的全局数据

2.3多线程与多进程的优缺点。

多进程模式最大的优点就是稳定性高，因为一个子进程崩溃了，不会影响主进程和其他子进程。（当然主进程挂了所有进程就全挂了，但是Master进程只负责分配任务，挂掉的概率低）著名的Apache最早就是采用多进程模式。

多进程模式的缺点是创建进程的代价大，在Unix/Linux系统下，用fork调用还行，在Windows下创建进程开销巨大。另外，操作系统能同时运行的进程数也是有限的，在内存和CPU的限制下，如果有几千个进程同时运行，操作系统连调度都会成问题。

多线程则因为共享资源的问题不稳定。

不过进程切换的代价远远大于线程切换的代价。（都是有代价的，过多的切换会造成系统崩溃）

不过一般来说，几个机器上不会跑很多程序（进程），因为cpu数量还是有限的，很少跑到上千个进程。

一般来说多开几个桌子吃菜（多弄几个进程），不用考虑线程安全，简单，而且linux鼓励我们多开桌子（花销并不大）。

### 3.线程同步的方式

锁有互斥锁，读写锁，条件变量这些

信号量的用来解决有限个数的资源使用问题，

信号量就像一个钥匙（钥匙的个数有限），有钥匙的人才能进入房间使用资源。。用完了还钥匙 mutex+condition\_variable来实现。

3.1避免死锁。

如果某个线程试图对同一个互斥量加锁两次，那么它将进入死锁状态。

还有一些更加不明显的方式也能产生死锁。比如：程序使用多个互斥量时，如果允许 线程甲 一直占有第一个互斥量 A 并且在试图锁住第二个互斥量 B 时处于阻塞状态，但是拥有第二个互斥量 B 的线程 乙 也在试图锁住第一个互斥量 A 。这时就会发生死锁。

解决方法：可用过小心的控制互斥量加锁顺序来避免死锁。

—— 把加锁和解锁的操作要有效的封装， 不要自己手动mutex.lock() mutex.unlock(),。

用std::lock\_guard （有condition\_variable时用unique\_guard）把mutex封装好

这种思想叫RAII

RAII[1]  （Resource Acquisition Is Initialization）,也称为“资源获取就是初始化”，是C++语言的一种管理资源、避免泄漏的惯用法，在其构造时获取资源，在对象生命期控制对资源的访问使之始终保持有效，对象销毁自动调用对象的析构函数来释放资源

3.2读写锁

读写锁和互斥量类似，区别在于：互斥量只有两个状态，即锁住状态和不加锁状态，而且一次只有一个线程可以对其加锁。而读写锁可以有三个状态，读模式下加锁状态，写模式下加锁状态，不加锁状态。一次只能有一个线程占有写模式的读写锁，但可以由多个线程同时占有读模式的读写锁。

读写锁非常适用于对数据结构读次数远大于写的情况。 

3.3 条件变量

线程同步时，某一线程阻塞等待符合某个条件在运行，而在另外一个进程运行中，满足了这个条件，条件变量就是使某一个线程阻塞等待直到满足条件，或者另外一个进程使条件成立以唤醒等待的线程。

条件变量使我们可以睡眠等待某种条件出现。为了防止竞争，条件变量的使用总是和一个[互斥锁](http://baike.baidu.com/view/1461738.htm)结合在一起。

<http://zh.cppreference.com/w/cpp/thread/condition_variable>

std::unique\_lock相对std::lock\_guard更灵活的地方在于在等待中的线程如果在等待期间需要解锁mutex，并在之后重新将其锁定。而std::lock\_guard却不具备这样的功能。

3.4原子操作

有点像数据库的事务，一个操作开始执行就一直要到执行完成，不会中途被线程调度中断。

Push

pop## 逻辑CPU

逻辑CPU的概念比较抽象，可简单理解为一个处理单元，通常来说，总的逻辑CPU数对应总的CPU核数，但借助超线程技术，一个核用起来像两个核，这时逻辑CPU数就是核心数的两倍了。t可通过如下命令来查看逻辑CPU数：

多核cpu与单核多cpu，单核多cpu进行多线程，只能每个线程跑1个cpu,.线程间的共享数据存于多个cpu的cache，不仅有空间浪费，而且线程的 所有协作都要走总线。

一般还是多核cpu，除非要跑大程序，要的内存多，且线程多到一个cpu的cache分分中满，交换的时候就很慢。

RAII

RAII[1]  （Resource Acquisition Is Initialization）,也称为“资源获取就是初始化”，是C++语言的一种管理资源、避免泄漏的惯用法