多线程:
信号量:
对比互斥锁:信号量更多的用于实现同步，主要是对资源进行计数，实现资源获取的合理性
对比条件变量:信号虽通过自身的计数进行条件判断，不需要搭配互斥锁一-起使用
线程池:
对有大量业务请求进行多任务并发/并行处理的场景
针对每个任务创建线程进行处理存在的问题:
1.若是任务过多,有可能创建线程过多，导致资源耗尽
2.每个线程都处理一一个任务后退出， 会在任务处理整个过程中带来额外的线程创建/销毁成本
线程池:在服务器初始化时候，创建大量线程，并且创建一个任务队列， 线程池中的线程不断的从任务队列中获取任务进行处理
如何实现:
C++ STL容器的线程安全的讨论:都不是线程安全的智能指针的线程安全: 都是线程安全的
单例模式:是设计模式中的一种，是大佬们针对典型场景设计的解决方案
典型场景: 一个对象/一个资源只能被初始化加载一次 (比如游戏中的图片资源，就不希望被加载多份)
实现方式:饿汉/懒汉
饿汉:所有资源在程序初始化阶段一次性完成初始化(资源在初始化的时候一次性全部加载， 后续只需要使用就可以)

```
template <class T>
class single {
    static T. _data; //static --所有实例化的对象共用同一份资源
    T* get_ instance() { return & data;}
} ///程序初始化的时候会慢一点， 但是运行起来之后，会很流畅;
```

懒汉:资源在使用的时候进行初始化(游戏中有很多的图片资源， 用到的时候再去加载，当然也要保证只加载一次)

```
#include <mutex>
class single {
    volatile static T *_data; //防止编译器过度优化后出现逻辑问题
    T *get_ instance() {
        if( _data == NULL){ //外部进行二次判断， 尽可能降低锁冲突概率
            mutex.lock();
```

```
        if( data == NULL) _data = new T(); //实例化资源的过程是-一个非原子操作，需要保护起来
        mutex.unlock();
    }
    return_ data;
}
```

资源没有加载的时候，加锁进行判断,然后申请资源解锁
资源被加载成功之后，获取资源的时候平白多了两步加锁解锁操作，并且在加锁成功后，其它线程不能加锁，阻塞直到解锁(又平白多了一步等待加锁的时间) ;二次判断就可以减少这种情况，不为空就直接获取资源进行操作

1.什么是单例模式:这种设计模式针对的典型场景----一个类只能实例化一个对象/一个资源只能被加载一次
2.如何实现:
饿汉:在程序初始化时加载一次资源，运行过程中就不再重新加载了
懒汉:在使用的时候加载资源，会涉及到线程安全问题(volatile / static / mutex /二次判断)
使用static保证多个对象使用同一份空间资源；保证资源只被加载一次实现单例模式
加锁保护资源申请过程，实现线程安全； 加锁保护，防止竞态条件下，资源被加载多份
在加锁之外进行二次判断，减少所冲突概率，提高效率
使用volatile关键，防止编译器过度优化，每次判断都为NULL的情况

下去之后调研一下锁的种类:乐观锁(CAS) /悲观锁(互斥锁)
读写锁/自旋锁
读写锁:多人读，少量写的场景---写互斥，读共享的场景
写的时候，有人读，会出现读的不完整的情况，有人写，则会写冲突的场景--- 写的时候其它线程不能读也不能写-写互斥读的时候，大家可以一起读，只是读的时候别人不能写
针对这种场景，使用一般的互斥锁已经不能满足; 使用读写锁实现这种场景的数据操作保护
加读锁:当前没有人写，就能加读锁　加写锁:当前既没有人读，也没有人写　解锁
读写锁的原理实现: 一个读者计数器 + 一个写着计数器就可以实现
加读锁，就是读者计数+ 1;加读锁的前提就是写 者计数为0
加写锁，就是写者计数+1;加写锁的前提就是读者和写者计数都为0

读写锁中若不能加锁，就需要等待，但是这里的等待与互斥锁的等待是不一样的

互斥锁的等待:挂起等待(将pcb状态置位阻塞)　　读写锁的等待: 自旋等待(通过自旋锁实现)

自旋锁:一直占用cpu进行条件判断直到条件满足---为什么自旋锁要-直占用cpu, 如果切换调度了会出现什么情况课后调研

自旋锁:响应速度比较快，但是比较占用cpu资源(一直循环判断) ---适用的场景就是确定等待时间比较短的场景

多线程:

线程概念/数据的独有与共享/多任务处理的优势

线程的创建/销毁/等待/分离

线程安全概念/实现方式/互斥锁/死锁/条件变量/信号量/生产者与消费者模型

线程池/线程安全的单例模式/ stl容器的线程安全/智能指针的线程安全/锁的种类/读写锁/自旋锁

条件变量:

1.条件变量如何实现同步的---两个接口---等待/唤醒

pthread_ cond_ t pthread_ cond_ init

1.判断访问条件,是否能够访问或者获取资源

2.若不能访问， 则调用pthread_ cond_ wait进行等待

3.其它线程若是促使条件满足了 ,通过条件变量唤醒等待的线程pthread_cond_signal

注意:

1.判断访问条件这个过程是-一个对临界资源访问的过程---需要搭配互斥锁加锁保护因此pthread_ cond_ wait 解锁/休眠/加锁

2.有可能多个吃面的被唤醒，但是只有哦一个吃面的能抢到锁吃饭，其它的卡在锁出;吃面的吃碗面解锁，有可能抢到锁的不是厨师,有可能会出现卡在锁处等待加锁吃面的线程，抢到锁在没有面的情况下吃面--因此条件的判断需要是循环

3.反之厨师唤醒的不是顾客而还是厨师，因此多种角色使用多个条件变量，不同的角色要加入不同条件变量的等待队列上