1. **进程与线程的区别**

1.1进程是os资源分配的基本单位，而线程是处理器任务调度和执行的基本单位

1.2每个进程有独立的代码和数据空间，切换会有较大的开销；线程可以看作轻量级进程，同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器，线程之间切换的开销小

1.3 一个进程有多个线程，线程是进程的一部分

1.4 一个进程崩溃，在保护模式下不会对其他的进程产生影响，但是一个线程崩溃，整个进程都会死掉。多进程要比多线程健壮

1.5 每个独立的进程有程序运行的入口、顺序执行序列和程序出口。但是线程不能独立执行，必须依存在应用程序中。

1. **用户态和内核态，为什么要划分？内存上怎么划分的？**

防止应用程序越权访问内存时使用了虚拟空间映射技术，在用户态下，应用程序访问的内存地址是虚拟内存地址，会映射到os的物理地址上，虚拟内存地址空间就是用户空间。

1. **怎么由用户态转到内核态？**

1、系统调用(处于用户态进程的主动请求切换到内核态的。使用了软中断)2、产生异常3、外设产生中断

**5. 用户程序使用系统调用是怎么做的，能否直接进行系统调用？**

软中断，Linux上一般是通过syscall指令进行系统调用。Go直接封装到了包里，fmt.Println调用了syscall.Write

1. **ISO七层模型** **(应用层(http/ftp)、表示层(telnet)、会话层(Session)DNS、传输层tcp udp、网络层ip arp icmp、数据链路层、物理层)**
2. **https中的ssl层属于什么层，说一说ssl握手的流程**

SSL协议位于TCP/IP协议与各种应用层协议之间，在运输层对网络连接进行加密。

Ssl目的1、c/s需要就一组用于保护数据的算法达成一致2、需要确立一组由那些算法所使用的加密密钥3、握手可以对client进行验证

1、c将所支持的算法列表和一个用作产生密钥的**随机数**发给服务器

2、s从算法列表中选择一种加密算法，将**它**和一份**包含服务器公用密钥的证书**发给客户端，证书还包含了用于认证目的的服务器标识，s还提供一个用于产生密钥的**随机数**

3、客户端对服务器的证书进行验证，抽取服务器的**公有密钥**；然后再产生一个pre\_master\_secret的**随机密码串**，并使用服务器的**公有密钥对随机密码串加密**，将加密后的信息发给s

4、客户端和服务器端根据**pre\_master\_secret随机密码串**以及客户端与服务器的**随机数值**独立计算出**加密和MAC密钥**

5、c将所有握手信息的MAC值发给s

6、s将所有握手信息的MAC值发给c

**56防止握手本身被篡改，c的mac由原始消息产生，s可能被攻击者修改，检查匹配**

1. **数据库的事务acid**

Atomicity（事务是一个不可分割的工作单位，事务的操作要么全部成功or失败）、consistency（事务执行前后都必须处于一致性状态）、isolation（多个并发事务相互隔离）、durabiliry（事务一旦提交，对数据的改变是永久性的）

1. **设计模式**

1、单例模式（一个类只有一个实例，并提供全局访问点）

class S:

instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if S.instance is None:

S.instance = super().\_\_new\_\_(cls)

def \_\_init\_\_(self):

pass

1. 工厂模式

1.不知道用户想要创建什么样的对象

2.当你想要创建一个可扩展的关联在**创建类**与**支持创建对象的类**之间。

class Factory:

def getPerson(self, name, gender):

if gender == ‘M':

return Male(name)

if gender == 'F':

return Female(name)

1. 抽象工厂模式

提供一个接口, **用于创建相关或依赖对象的家族**, 而**不需要指定具体类**。“ 工厂产生类 ”去让我们决定到底是要产生哪个类的实例。围绕一个超级工厂创建其他工厂。

class FactoryProducer:

def get\_factory(self,name):

if name=='Shape':

return ShapeFactory()

elif name=='Color':

return ColorFactory()

else:

return None

1. 原型模式、多例模式

14. Mysql语句检查：通过explain检查

15. 进程间的通信

管道（包括无名管道（有关进程）和命名管道（无关））、消息队列、信号量（os的pv信号量用于实现进程间的互斥与同步）、共享存储、Socket(于不同机器间的进程通信)、Streams

1. 线程间的通信

信号量、互斥量、条件变量

**17. Redis数据过多应该怎么办** lru算法、存热点数据

**18. 找出一堆树里面最大的Top k，说说几种思路，复杂度各是多少**

O(N·lgK)   【N为海量数据，K为要取的最大前K个数。】

算法题：平衡二叉树

Not equal return -1

堆的时间复杂度：初始化建堆是n，调整堆是nlogn

Session是KV存储，HashMap

**状态码403 无权限访问**

**链表**

**链表有顺序之分，插入删除快，但是查找慢**

1. **Redis单线程他会快**

Redis基于内存的操作（内存读写快）、CPU不是Redis的瓶颈，机器内存大小或者网络带宽才是、单线程省去了上下文切换线程的时间、redis使用IO多路复用技术，可以处理并发的连接。非堵塞IO 内部实现采用epoll+自己实现的简单的事件框架。epoll的读、写、关闭、连接都转化成了事件，利用epoll的多路复用特性，不在io上浪费时间。不需要各种锁（很细粒度的操作）的性能消耗、无法发挥多核CPU性能，不过可以通过在单机开多个Redis实例来完善

20. **为什么多路复用会快？**

redis 采用网络IO多路复用技术来保证在多连接的时候，**系统的高吞吐量。多路：多个socket网络连接，复用一个线程。**同时管理多个I/O流。系统不需要建立新的进程或者线程，也不必维护这些线程和进程，并且只有在真正有socket读写事件进行时，才会使用IO资源，所以它大大减少了资源占用。

1. **poll和epoll的区别？**

epoll 现在是线程安全的。epoll 现在不仅告诉你sock组里面数据，还会告诉你具体哪个sock有数据，你不用自己去找了。

Poll不是线程安全的，只能在一个线程里面处理一组I/O流。poll返回后，需要轮询pollfd来获取就绪的描述符。

select 如果任何一个sock(I/O stream)出现了数据，select 仅仅会返回，但是并不会告诉你是那个sock上有数据，于是你只能自己一个一个的遍历，且max\_size = 1024。会修改传入的参数数组。不是线程安全

1. **说一下五种IO模型，以及如何区别它们？**

阻塞I/O模型（所有文件操作的阻塞的）、非阻塞I/O模型（）、.多路复用IO模型（会有一个线程不断去轮询多个socket的状态，只有当socket真正有读写事件时，才真正调用实际的IO读写操作）、信号驱动IO模型（当用户线程发起IO请求操作，会给对应的socket注册一个信号函数，然后用户线程继续执行，当内核数据就绪时就发一个信息给用户线程，用户线程接收到信号时，在信号函数中调用IO读写操作进行实际的读写）、异步IO模型（IO操作的两个阶段都不会阻塞用户线程，这两个阶段都是由内核自动完成，然后发送一个信号告知用户线程操作已完成。）

23. redis的缓存淘汰机制？lru的实现

Lru，python 中是用ordereddict 底层是双向链表和hashmap。HashMap 的 Value 指向双向链表实现的 LRU 的 Node 节点。如果存储满了，可以通过 O(1) 的时间淘汰掉双向链表的尾部，每次新增和访问数据，都可以通过 O(1)的效率把新的节点增加到对头，或者把已经存在的节点移动到队头

24. 面向连接和无连接的区别？

无连接协议，每个分组的处理都独立于其他的分组。面向连接的协议，协议实现维护了当前分组和后继分组有关的状态信息。应用场景不同，可靠性要求较高TCP（HTTP、FTP）、无连接的套接字(UDP服务)，效率和实时为要求，DNS

1. 堆初始化后，堆顶与最后一个交换，然后向下筛选，将元素移动到正确的位置

Def heapify(arr,n,i):

Largest = i

L = 2i+1

R = 2i+2

if l<n and arr[l]>arr[i]:

largest = l

if r<n and arr[largest]<arr[r]:

largest = r

if largest != i:

swap(arr[largest],arr[i])

heapify(arr,n,largest)

def heapsort(arr):

n = len(arr)

for i in range(n>>1,-1,-1):

heapify(arr,n,i)

for i in range(n,0,-1):

swap(arr[i],arr[0])

heapify(arr,i,0)

1. **mysql联合索引，最左匹配，abcd，如果要用索引，不能跳过a，单独用b。c>4后,abc字段能用索引，d不行，遇到了范围查询。多键值的b+树。区分度高的字段放在前面，区分度低的字段放后面。像性别、状态这种字段区分度就很低，我们一般放后面。**
2. **查看打印日志cat -n test.log |grep "debug" >debug.txt**
3. **golang 死锁展示（wg := sync.WaitGroup;wg.Add(100)计数器为100，wg.Done()会减一，wg.Wait()会阻塞，直到计数器为0。sync.Mutex当一个goroutine获得这个锁，其他的只能等待释放 ）**

**type value stuct{**

**mu sync.Mutex**

**value int**

**}**

**func main(){**

**var wg sync.WaitGroup**

**PrintSum:=func(v1,v2 \*value){**

**defer wg.Done()**

**v1.mu.Lock()**

**defer v1.mu.Unlock()**

**v2.mu.Lock()**

**defer v2.mu.Lock()**

**}**

**var a,b value**

**wg.Add(2)**

**go PrintSum(&a,&b)**

**go PrintSum(&b,&a)**

**wg.Wait()**

**}**

**29.**