1. linux当在一个目录下创建文件时，无法创建，可能是什么问题？

①磁盘空间满了

②inode节点不足

③系统ulimit限制打开文件数目

1. ext2、ext3、ext4文件系统的区别？

ext2和ext3的区别就是一个磁盘日志功能。

ext3与ext4:的区别：

①支持更大容量的文件系统和文件，ext3支持16TB和最大2T把文件，ext4支持1EB和最大16TB文件

②Extends，操作大文件效率要高

③多块一次性分配ext3每次比如分配4KB的page来写，ext4可以一次多个page写

④延时分配

⑤快速fsck

⑥日志校验

⑦无日志模式

⑧在线碎片整理

⑨默认启用barrier，实际上就是缓存和日志的关系，写入斗鱼缓存，只有当日志记录之后才进行写磁盘操作。

1. linux一些用于监控的命令？

①w

查看系统负载，当前有哪些用户登录到系统中以及一些行为

登录用户 终端号 fromip 登录时间 1 5 15分钟平均负载

②vmstat

比较好用，可以看正在使用cpu的进程数、阻塞进程数、内存空间、磁盘IO、cpu空闲使用情况

③top

主要是cpu和内存的使用情况

④sar

一般用来查看各个网卡收发数据包情况

⑤nload

也是查看网卡流量，比sar更直观

⑥iostat

查看cpu和磁盘io情况

⑦iotop

查看那些进程占用io

1. linux的缓存与缓冲机制？

①cache（缓存）：是指读取出来的数据保存在内存中，当再次读取时，不用读取硬盘

②buffer（缓冲）：是指在写入数据时，先把分散的写入操作保存在内存中，当达到一定程度再集中写入磁盘，减少磁盘碎片和硬盘的反复寻道，加速数据的写入过程

清除缓冲：sync

清除缓冲：To free pagecache:

# echo 1 > /proc/sys/vm/drop\_caches

To free dentries and inodes:

# echo 2 > /proc/sys/vm/drop\_caches

To free pagecache, dentries and inodes:

# echo 3 > /proc/sys/vm/drop\_caches

缓存回收策略：

①基于空间，当缓存空间不足时

②基于容量，当缓存条数达到一定量时

③基于时间，一种是存活期，当存活一定时间自动清除，还有一种是空闲期，当有一段时间没有被访问过就清除。

缓存回收算法：

①FIFO：先进先出，用队列实现，在mysql面试题中有整理

②LRU：最近最少使用，用hash表+双向链表实现，更多用的是LRU-2而不是单纯的LRU在mysql面试题中有整理

③LFU：最不常使用，用队列+计数实现，在mysql面试题中有整理

关于redis和mysql的LRU算法实现在mysql面试题中整理了，下面来看看linux是如何应用什么策略清除缓存的——双LRU

即用两个LRU链表，一个activeLRU链表，一个inactiveLRU链表，数据页在两个链表之间可以移动，每次清除是清除inactive链表的尾部。

其实关于linux释放数据页也分回收和不回收（释放），回收又分回写磁盘和回收到swap分区等等知识，慢慢来。

1. 你说说swap分区是怎么回事吧？

定义：当物理内存不够用的时候，把硬盘空间的一部分释放出来以供当前运行的程序使用，意思就是有些很长时间没什么操作的程序把他放进swap分区（磁盘），进而达到增大内存的效果，如果被放进swap分区的程序再次运行，就恢复到内存中，比如电脑打开很多进程，突然回到很久没打开的程序中就会出现卡顿的现象！！

注意：并不是所有从物理内存中交换出来的数据都会被放到swap分区中，不然swap得死，有相当一部分数据被直接交换到文件系统或者直接释放，例如读文件时，因为是只读的，回收缓存时不存在脏数据，所以直接释放即可，而写文件时就需要回写到文件系统了（有文件背景的是“文件页”），然而那些new出来的对象、栈中的变量数据、代码段他们在文件系统中没有映射，所以是“匿名页”，这些匿名页的缓存回收时需要swap分区保存。

这就是刚才所说了linux在回收缓存的时候哪些回收哪些释放，回收的又分为回收到swap分区还是磁盘。

现在应该串起来了吧。

1. 你知道五种IO模型吗？

先了解几个概念：

①同步：调用一个功能，在该功能没有结束前，我死等结果

②异步：调用一个功能，不需要知道该结果，有结果后回调通知

③阻塞：就是调用函数，该函数没有接受完数据或者没有得到结果之前不会反回

④非阻塞，就是调用函数，该函数立即返回，通过select通知调用者

那么有人会问，同步和阻塞到底什么区别啊，这不看上去一样嘛？

注意：他们修饰的对象不同，同步与异步指的是调用函数或者功能这个调用的手段，而阻塞与非阻塞指的是被调用的那个函数

下面看一下linux下的五种IO模型：

①阻塞IO

进程会一直阻塞，直到数据拷贝完成，比如网络io，当数据通过网卡写入内存后系统会通知cpu执行中断程序cpu将数据从内核拷贝到用户进程空间，这个过程中就有recv()、recvfrom()、send等函数的调用，比如调用recv()之后会调用内核操作，等待数据和复制数据，这就是阻塞进程了，该线程会被挂起。

虽然这是我们一开始学网络编程时写的，但是满足不了生产者消费者的需要。如果每个连接都开辟一个通道，开销太大

②非阻塞IO

用户进程不断的主动询问内核数据是都准备好而不会发生阻塞挂起，如果没准备好就立即返回一个error信息，不断的询问直到数据准备好。

非常不推荐，非常占用cpu

③IO复用模型（事件驱动模型）

主要是依赖select和epoll，对于一个IO端口，两次调用、两次返回，比阻塞IO并没有什么优越性，关键是能够通过对多个IO端口进行监听！

这种方式，虽然也会阻塞进程，但是强就强在epoll可以同时管理多个socket连接并且将维护等待的socket队列和阻塞进程完全分离（也是比select高明的地方）进程调用完recv之后阻塞，而socket交给epoll来监听，当内核收到数据之后会通知socket，而epoll会通知进程进入就绪状态，获得cpu之后就执行，那么比阻塞IO高明之处在于不用进程自己管理socket连接，epoll来同时管理多个socket连接！！，不然你进城就得采用多线程、多进程的模式，刚才讲了，开销太大。

④信号驱动IO

⑤异步IO

简单来说就是数据拷贝的时候进程无需阻塞。可以干其他的。

注意：经过五种IO模型的介绍，现在应该对异步和非阻塞有了很好的理解了吧，异步就是我把一切托付给你，完成后通知我，但是非阻塞则是你来干但是我会监督你。同步就是不管阻塞非阻塞我就等着你完成。

1. select和epoll你了解吗？

上面既然说了IO多路复用，那么能说一下linux下的select、poll、epoll的联系和区别吗？

首先本质上select和poll没有大的区别，只不过poll没对针对单个进程打开最大连接数的限制，select在64位系统中限制是2048个连接。

要知道网络编程中io多路复用用的很多，比如nginx、redis都有epoll的支持。

首先我们先要理清请求和服务之间是怎么做的，也就是说数据进入服务器到真正被处理的过程：

①网卡收到网线传来的数据

②通过硬件的传输写入内存的某个地址上，此时操作系统（内核）才可以读取

③网卡想cpu发出一个中断信号，操作系统便能得知有新数据进来，中断信号是硬件发给cpu的，优先级很高，比如断电之后电容有一定电量，此时cpu会中断当前进程去内存中把数据回写到磁盘，为了保证数据不丢

④内核使用执行中断程度的cpu把内存中根据数据来判断这个数据是谁需要的将数据拷贝到用户进程，同时唤醒需要这个数据的阻塞的进程。

下面说一下为什么需要select和epoll？

比如编程时创建了一个socket对象，这个对象包含发送缓冲区、接收缓冲区、等待队列成员（各个进程），当执行到recv()时，操作系统把该进程移动到该socket的等待队列中，所以该进程就被阻塞了。

那么现在问题来了，一个recv()就是一个socket连接，但是要知道一台服务器要接受几千个连接呢，难道要用几千个线程？肯定不合适，开销不行，所以就需要一个东西可以管理多个socket连接！此时select和epoll就派上用场了。

select核心思想：

有一个socket列表，如果所有socket都没有数据，那么就挂起进程，一旦socket中有一个或者多个有数据进来了那么就唤醒进程，并且遍历socket，到底哪个socket有数据我除了那个。

实现：

比如现在又三个socket连接到了A进程，那么把A进程用引用分别加入到三个socket的等待队列中，当某个socket有数据，那么cpu唤醒进程A来处理，也就是把A进程从等待队列中移除方导工作队列中。

缺陷：

三次遍历，第一次：没有数据时要把进程A放入全部的sockets的等待队列中，第二次：当有数据时还要把A从所有sockets等待队列中移除，第三次：当A被唤醒后需要遍历socket才知道谁的数据来了。

epoll核心思想：

①功能分离：维护等待队列和阻塞进程分离。意义在于不必像select似的只要阻塞就添加到所有sockets等待队列。

②就绪列表：就绪列表rdlist保存进来数据的socket，就不必像select一样遍历所有sockets了

实现：

创建epoll对象，该对象该对象内部维护就绪列表，区别来了：epoll对象会把自己的引用放入所有sockets的等待队列而不是进程A，这样cpu执行中断程序的时候会唤醒epoll对象而不是操作进程A，同时epoll的就绪列表中有了有数据的socket引用，而关键点就在这里，进程A的阻塞和唤醒跟这个就绪列表rdlist有关，如果就绪列表为空就阻塞否则你懂的就处理呗，此时处理就不用遍历了因为只有有数据的socket才会在就绪列表中。

问个小问题，请问epoll用什么数据结构存储就绪列表和用什么数据结构管理epoll对socket的添加和删除。

①双向链表作为就绪列表的数据结构可以快速插入和删除

②那么监控socket用什么，快速插入和删除是必须的，还要便于搜索，防止重复插入，那么这个数据结构就是二叉搜索树，但是为了避免高度极端采用平衡二叉树，而红黑树就是一个缓冲，既是平衡二叉树又没平衡二叉树那么苛刻。