**1：tcp和udp的区别**

TCP：是面向连接的流传输控制协议，具有高可靠性，确保传输数据的正确性，有验证重发机制，因此不会出现丢失或乱序。

UDP：是无连接的数据报服务，不对数据报进行检查与修改，无须等待对方的应答，会出现分组丢失、重复、乱序，但具有较好的实时性，UDP段结构比TCP的段结构简单，因此网络开销也小。

**2：流量控制和拥塞控制**

拥塞控制  
网络拥塞现象是指到达通信子网中某一部分的分组数量过多,使得该部分网络来不及处理,以致引起这部分乃至整个网络性能下降的现象,严重时甚至会导致网络通信业务陷入停顿,即出现死锁现象。拥塞控制是处理网络拥塞现象的一种机制。  
流量控制  
数据的传送与接收过程当中很可能出现收方来不及接收的情况,这时就需要对发方进行控制,以免数据丢失。

**3：多线程如何同步**

windows

线程同步有四种方式：[临界区、内核对象、互斥量、信号量](http://blog.csdn.net/windows_nt/article/details/8953088)。

Linux

线程同步有最常用的是：[互斥锁、条件变量和信号量。](http://blog.csdn.net/windows_nt/article/details/28676157)

**4：进程间通讯的方式有哪些，各有什么优缺点**

[进程间通信](http://blog.csdn.net/windows_nt/article/details/28682333)

Linux 进程间通信（IPC）以下以几部分发展而来：  
早期UNIX进程间通信、基于System V进程间通信、基于Socket进程间通信和POSIX进程间通信。  
UNIX进程间通信方式包括：管道、FIFO、信号。  
System V进程间通信方式包括：System V消息队列、System V信号灯、System V共享内存、  
POSIX进程间通信包括：posix消息队列、posix信号灯、posix共享内存。  
现在linux使用的进程间通信方式：  
（1）管道（pipe）和有名管道（FIFO）  
（2）信号（signal）  
（3）消息队列  
（4）共享内存  
（5）信号量  
（6）套接字（socket)

**5：tcp连接建立的时候3次握手，**断开连接的4次握手**的具体过程**

建立连接采用的3次握手协议，具体是指：  
第一次握手是客户端connect连接到server，server accept client的请求之后，向client端发送一个消息，相当于说我都准备好了，你连接上我了，这是第二次握手，第3次握手就是client向server发送的，就是对第二次握手消息的确认。之后client和server就开始通讯了。  
断开连接的4次握手,具体如下：  
断开连接的一端发送close请求是第一次握手，另外一端接收到断开连接的请求之后需要对close进行确认，发送一个消息，这是第二次握手，发送了确认消息之后还要向对端发送close消息，要关闭对对端的连接，这是第3次握手，而在最初发送断开连接的一端接收到消息之后，进入到一个很重要的状态time\_wait状态，这个状态也是面试官经常问道的问题，最后一次握手是最初发送断开连接的一端接收到消息之后。对消息的确认。

**6：epoll与select的区别**

select在一个进程中打开的最大fd是有限制的，由FD\_SETSIZE设置，默认值是2048。不过 epoll则没有这个限制，它所支持的fd上限是最大可以打开文件的数目，这个数字一般远大于2048，一般来说内存越大，fd上限越大，1G内存都能达到大约10w左右。  
  
select的轮询机制是系统会去查找每个fd是否数据已准备好，当fd很多的时候，效率当然就直线下降了，epoll采用基于事件的通知方式，一旦某个fd数据就绪时，内核会采用类似callback的回调机制，迅速激活这个文件描述符，而不需要不断的去轮询查找就绪的描述符，这就是epoll高效最本质的原因。  
  
无论是select还是epoll都需要内核把FD消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll是通过内核于用户空间mmap同一块内存实现的，而select则做了不必要的拷贝

**7：epoll中et和lt的区别与实现原理**

LT：水平触发，效率会低于ET触发，尤其在大并发，大流量的情况下。但是LT对代码编写要求比较低，不容易出现问题。LT模式服务编写上的表现是：只要有数据没有被获取，内核就不断通知你，因此不用担心事件丢失的情况。  
ET：边缘触发，效率非常高，在并发，大流量的情况下，会比LT少很多epoll的系统调用，因此效率高。但是对编程要求高，需要细致的处理每个请求，否则容易发生丢失事件的情况。

=======================================================================

1、connect方法会阻塞，请问有什么方法可以避免其长时间阻塞？  
答:最通常的方法最有效的是加定时器；也可以采用非阻塞模式。  
  
2、网络中，如果客户端突然掉线或者重启，服务器端怎么样才能立刻知道？  
答:若客户端掉线或者重新启动，服务器端会收到复位信号，每一种tcp/ip得实现不一样，控制机制也不一样。  
  
3.在子网210.27.48.21/30种有多少个可用地址？分别是什么？  
答:  
  
简:  
30表示的是网络号(network number)是30位，剩下2位中11是广播(broadcast)地址，00是multicast地址，只有01和10可以作为host address。  
  
详:  
210.27.48.21/30代表的子网的网络号是30位，即网络号是210.27.48.21 & 255.255.255.251=210.27.48.20，此子网的地址空间是2位，即可以有4个地址：210.27.48.20, 210.27.48.21, 210.27.48.22, 210.27.48.23。第一个地址的主机号(host number/id)是0，而主机号0代表的是multicast地址。最后一个地址的最后两位是11，主机号每一位都为1代表的是广播(broadcast)地址。所以只有中间两个地址可以给host使用。其实那个问题本身不准确，广播或multicast地止也是可以使用的地址，所以回答4也应该正确，当然问的人也可能是想要你回答2。我个人觉得最好的回答是一个广播地址，一个multicast地址，2个unicast地址。  
  
4.TTL是什么？有什么用处，通常那些工具会用到它？（ping? traceroute? ifconfig? netstat?）  
答:  
简:TTL是Time To Live，一般是hup count，每经过一个路由就会被减去一，如果它变成0，包会被丢掉。它的主要目的是防止包在有回路的网络上死转，浪费网络资源。ping和traceroute用到它。  
  
详:TTL是Time To Live，目前是hup count，当包每经过一个路由器它就会被减去一，如果它变成0，路由器就会把包丢掉。IP网络往往带有环(loop)，比如子网A和子网B有两个路由器相连，它就是一个loop。TTL的主要目的是防止包在有回路的网络上死转，因为包的TTL最终后变成0而使得此包从网上消失(此时往往路由器会送一个ICMP包回来，traceroute就是根据这个做的)。ping会送包出去，所以里面有它，但是ping不一定非要不可它。traceroute则是完全因为有它才能成的。ifconfig是用来配置网卡的，netstat -rn 是用来列路由表的，所以都用不着它  
  
5.路由表示做什么用的？在linux环境中怎么来配置一条默认路由？  
答:  
简:路由表是用来决定如何将包从一个子网传送到另一个子网的，换局话说就是用来决定从一个网卡接收到的包应该送的哪一张网卡上的。在Linux上可以用“route add default gw <默认路由器IP>”来配置一条默认路由。  
  
详:路由表是用来决定如何将包从一个子网传送到另一个子网的，换局话说就是用来决定从一个网卡接收到的包应该送的哪一张网卡上的。路由表的每一行至少有目标网络号、netmask、到这个子网应该使用的网卡。当路由器从一个网卡接收到一个包时，它扫描路由表的每一行，用里面的netmask和包里的目标IP地址做并逻辑运算(&)找出目标网络号，如果此网络号和这一行里的网络号相同就将这条路由保留下来做为备用路由，如果已经有备用路由了就在这两条路由里将网络号最长的留下来，另一条丢掉，如此接着扫描下一行直到结束。如果扫描结束任没有找到任何路由，就用默认路由。确定路由后，直接将包送到对应的网卡上去。在具体的实现中，路由表可能包含更多的信息为选路由算法的细节所用。题外话：路由算法其实效率很差，而且不scalable，解决办法是使用IP交换机，比如MPLS。  
在Linux上可以用“route add default gw <默认路由器IP>”来配置一条默认路由。  
  
6.在网络中有两台主机A和B，并通过路由器和其他交换设备连接起来，已经确认物理连接正确无误，怎么来测试这两台机器是否连通？如果不通，怎么来判断故障点？怎么排除故障？  
答:测试这两台机器是否连通：从一台机器ping另一台机器  
     如果ping不通，用traceroute可以确定是哪个路由器不能连通，然后再找问题是在交换设备/hup/cable等。  
  
7.网络编程中设计并发服务器，使用多进程 与 多线程 ，请问有什么区别？   
答案一:  
1，进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。  
2，线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。  
两者都可以提高程序的并发度，提高程序运行效率和响应时间。  
线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。  
  
答案二:  
根本区别就一点：用多进程每个进程有自己的地址空间(address space)，线程则共享地址空间。所有其它区别都是由此而来的：  
1。速度：线程产生的速度快，线程间的通讯快、切换快等，因为他们在同一个地址空间内。  
2。资源利用率：线程的资源利用率比较好也是因为他们在同一个地址空间内。  
3。同步问题：线程使用公共变量/内存时需要使用同步机制还是因为他们在同一个地址空间内。  
等等

**3. 网络编程的一般步骤**

**对于TCP连接：**

1.服务器端1）创建套接字create；2）绑定端口号bind；3）监听连接listen；4）接受连接请求accept，并返回新的套接字；5）用新返回的套接字recv/send；6）关闭套接字。

2.客户端1）创建套接字create; 2）发起建立连接请求connect; 3）发送/接收数据send/recv；4）关闭套接字。

TCP总结：

Server端：create -- bind -- listen--  accept--  recv/send-- close

Client端：create------- conncet------send/recv------close.

**对于UDP连接：**

1.服务器端:1）创建套接字create；2）绑定端口号bind；3）接收/发送消息recvfrom/sendto；4）关闭套接字。

2.客户端:1）创建套接字create；2）发送/接收消息sendto/recvfrom；3）关闭套接字.

UDP总结:

Server端：create----bind ----recvfrom/sendto----close

Client端：create----  sendto/recvfrom----close.

**5. TCP的重发机制是怎么实现的？**

       1.滑动窗口机制，确立收发的边界，能让发送方知道已经发送了多少（已确认）、尚未确认的字节数、尚待发送的字节数；让接收方知道（已经确认收到的字节数）。

       2.选择重传，用于对传输出错的序列进行重传。

**6. TCP和UDP的区别？**

       1）TCP面向连接（三次握手机制），通信前需要先建立连接；UDP面向无连接，通信前不需要建立连接；

       2）TCP保障可靠传输（按序、**无差错、不丢失、不重复**）；UDP不保障可靠传输，使用最大努力交付；

       3）TCP面向字节流的传输，UDP面向数据报的传输。

   TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和[服务器](http://www.haodaima.net/tag/%E6%9C%8D%E5%8A%A1%E5%99%A8)彼此交换[数据](http://www.haodaima.net/tag/%E6%95%B0%E6%8D%AE)前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。  
UDP---[用户](http://www.haodaima.net/tag/%E7%94%A8%E6%88%B7)数据报协议，是一个简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把[应用程序](http://www.haodaima.net/tag/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)传给[IP](http://www.haodaima.net/tag/IP)层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快

**8.TCP为什么不是两次连接？而是三次握手？**

如果A与B两个进程通信，如果仅是两次连接。**可能出现的一种情况**就是：A发送完请报文以后，由于网络情况不好，出现了网络拥塞，即B延时很长时间后收到报文，即此时A将此报文认定为失效的报文。B收到报文后，会向A发起连接。此时两次握手完毕，B会认为已经建立了连接可以通信，B会一直等到A发送的连接请求，而A对失效的报文回复自然不会处理。依次会陷入B忙等的僵局，造成资源的浪费。

9. **connect方法会阻塞，请问有什么方法可以避免其长时间阻塞？**

可以考虑采用异步传输机制，同步传输与异步传输的主要区别在于同步传输中，如果调用recvfrom后会一致阻塞运行，从而导致调用线程暂停运行；异步传输机制则不然，会立即返回。

**8.网络编程中设计并发服务器，使用多进程与多线程，请问有什么区别？**

答案一:

1，进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。

2，线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。两者都可以提高程序的并发度，提高程序运行效率和响应时间。

线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。

答案二:

根本区别就一点：用多进程每个进程有自己的地址空间(address space)，线程则共享地址空间。所有其它区别都是由此而来的：

1。速度：线程产生的速度快，线程间的通讯快、切换快等，因为他们在同一个地址空间内。

2。资源利用率：线程的资源利用率比较好也是因为他们在同一个地址空间内。

3。同步问题：线程使用公共变量/内存时需要使用同步机制还是因为他们在同一个地址空间内。

等等

**17.流量控制和拥塞控制的实现机制**

|  |  |  |  |
| --- | --- | --- | --- |
| 拥塞控制 网络拥塞现象是指到达通信子网中某一部分的分组数量过多,使得该部分网络来不及处理,以致引起这部分乃至整个网络性能下降的现象,严重时甚至会导致网络通信业务陷入停顿,即出现死锁现象。拥塞控制是处理网络拥塞现象的一种机制。 流量控制 数据的传送与接收过程当中很可能出现收方来不及接收的情况,这时就需要对发方进行控制,以免数据丢失。  **流量控制机制：**  　　流量控制用于防止在端口阻塞的情况下丢帧，这种方法是当发送或接收缓冲区开始溢出时通过将阻塞信号发送回源地址实现的。流量控制可以有效的防止由于网络中瞬间的大量数据对网络带来的冲击，保证用户网络高效而稳定的运行。    **18.多线程如何同步：**   在这里简单说一下linux多线程同步的方法吧（win上有一定的差别，也有一定的累似）  **1：**线程数据，每个线程数据创建一个键，它和这个键相关联，在各个线程里，都使用这个键来指代线程数据，但在不同的线程里，这个键代表的数据是不同的，在同一个线程里，它代表同样的数据内容。以此来达到线程安全的目的。 **2：**互斥锁，就是在各个线程要使用的一些公共数据之前加锁，使用之后释放锁，这个是非常常用的线程安全控制的方法，而频繁的加解锁也对效率有一定的影响。 **3：**条件变量，而条件变量通过允许线程阻塞和等待另一个线程发送信号的方法弥补了互斥锁的不足，它常和互斥锁一起使用。使用时，条件变量被用来阻塞一个线程，当条件不满足时，线程往往解开相应的互斥锁并等待条件发生变化。一旦其它的某个线程改变了条件变量，它将通知相应的条件变量唤醒一个或多个正被此条件变量阻塞的线程。这些线程将重新锁定互斥锁并重新测试条件是否满足。一般说来，条件变量被用来进行线程间的同步。 **4：**信号量，信号量本质上是一个非负的整数计数器，它被用来控制对公共资源的访问。当公共资源增加时，调用函数sem\_post（）增加信号量。只有当信号量值大于０时，才能使用公共资源，使用后，函数sem\_wait（）减少信号量。函数sem\_trywait（）和函数pthread\_ mutex\_trylock（）起同样的作用，它是函数sem\_wait（）的非阻塞版本 另外pthread\_join也可以等待一个线程的终止。  **19.进程间通讯的方式有哪些，各有什么优缺点**  进程间通信主要包括管道, 系统IPC(包括消息队列,信号量,共享存储), socket.  管道包括三种:1)普通管道PIPE, 通常有种限制,一是半双工,只能单向传输;二是只能在父子进程间使用. 2)流管道s\_pipe: 去除了第一种限制,可以双向传输. 3)命名管道:name\_pipe, 去除了第二种限制,可以在许多并不相关的进程之间进行通讯.  系统IPC的三种方式类同,都是使用了内核里的标识符来识别  管道: 优点是所有的[UNIX](http://www.haodaima.net/tag/Unix)实现都支持, 并且在最后一个访问管道的进程终止后,管道就被完全删除;缺陷是管道只允许单向传输或者用于父子进程之间  系统IPC: 优点是功能强大,能在毫不相关进程之间进行通讯; 缺陷是[关键字](http://www.haodaima.net/tag/%E5%85%B3%E9%94%AE%E5%AD%97)KEY\_T使用了内核标识,占用了内核资源,而且只能被显式删除,而且不能使用SOCKET的一些机制,例如select,epoll等.  socket可以跨网络通讯，其他进程间通讯的方式都不可以，只能是本机进程通讯。    **20.tcp连接建立的时候3次握手的具体过程，以及其中的每一步是为什么**  建立连接采用的3次握手协议，具体是指：  第一次握手是[客户端](http://www.haodaima.net/tag/%E5%AE%A2%E6%88%B7%E7%AB%AF)connect连接到server，server accept client的请求之后，向client端发送一个消息，相当于说我都准备好了，你连接上我了，这是第二次握手，第3次握手就是client向server发送的，就是对第二次握手消息的确认。之后client和server就开始通讯了。  **21.tcp断开连接的具体过程，其中每一步是为什么那么做**  断开连接的4次握手,具体如下：  断开连接的一端发送close请求是第一次握手，另外一端接收到断开连接的请求之后需要对close进行确认，发送一个消息，这是第二次握手，发送了确认消息之后还要向对端发送close消息，要关闭对对端的连接，这是第3次握手，而在最初发送断开连接的一端接收到消息之后，进入到一个很重要的状态time\_wait状态，这个状态也是面试官经常问道的问题，最后一次握手是最初发送断开连接的一端接收到消息之后。对消息的确认。  =======================  1.C++模板的作用。  将算法与具体对象分离，与类型无关，通用，节省精力    2.socket编程，如果client断电了，服务器如何快速知道？？？  有以下几个技术：  使用定时器（适合有数据流动的情况）； 使用socket选项SO\_KEEPALIVE（适合没有数据流动的情况）;      3.fork()一子进程程后 父进程癿全局变量能不能使用？？？  fork后子进程将会拥有父进程的几乎一切资源，父子进程的都各自有自己的全局变量。不能通用，不同于线程。对于线程，各个线程共享全局变量。    4.4G的long型整数中找到一个最大的，如何做？？？？  我的想法是要找到最大的肯定要遍历所有的数的，而且不能将数据全部读入内存，可能不足。算法的时间复杂度肯定是O（n）  感觉就是遍历，比较。。。。还能怎么改进呢？？？？  可以改进的地方，就是读入内存的时候，一次多读些。。。。  需 要注意的就是每次从磁盘上尽量多读一些数到内存区，然后处理完之后再读入一批。减少IO次数，自然能够提高效率。而对于类快速排序方法，稍微要麻烦一些： 分批读入，假设是M个数，然后从这M个数中选出n个最大的数缓存起来，直到所有的N个数都分批处理完之后，再将各批次缓存的n个数合并起来再进行一次类快 速排序得到最终的n个最大的数就可以了。在运行过程中，如果缓存数太多，可以不断地将多个缓存合并，保留这些缓存中最大的n个数即可。由于类快速排序的时 间复杂度是O（N），这样分批处理再合并的办法，依然有极大的可能会比堆和败者树更优。当然，在空间上会占用较多的内存。    此题还有个变种，就是寻找K个最大或者最小的数。有以下几种算法：  容量为K的最大堆/最小堆，假设K可以装入内存；  如果N个数可以装入内存，且都小于MAX，那么可以开辟一个MAX大的数组，类似计数排序。。。从数组尾部扫描K个最大的数，头部扫描K个最小的数。      5.有千万个string在内存怎么高速查找，插入和删除？？？  对千万个string做hash，可以实现高速查找，找到了，插入和删除就很方便了。  关键是如何做hash，对string做hash，要减少碰撞频率。  In the String class, for example, the hash code h of a string s of length n is calculated as  or, in code,  int h = 0; for (int i = 0; i < n; i++) { h = 31\*h + s.charAt(i); }  In general the arithmetic operations in such expressions will use 32-bit modular arithmetic ignoring overflow  在实际中，BKDRhash函数比较好  // BKDR Hash unsigned int BKDRHash(char \*str) { unsigned int seed = 131; // 31 131 1313 13131 131313 etc.. unsigned inthash = 0;   while (\*str) { hash = hash \* seed + (\*str++); }   return (hash & 0x7FFFFFFF); }  6.tcp三次握手的过程，accept发生在三次握手哪个阶段？  三次握手：C----->SYN K                S------>ACK K+1 SYN J                C------->ACK J+1                DONE!  client 的 connect  引起3次握手  server 在socket， bind， listen后，阻塞在accept，三次握手完成后，accept返回一个fd，  因此accept发生在三次握手之后。。。。。。  7.Tcp流， udp的数据报，之间有什么区别，为什么TCP要叫做数据流？  TCP本身是面向连接的协议，S和C之间要使用TCP，必须先建立连接，数据就在该连接上流动，可以是双向的，没有边界。所以叫数据流 ，占系统资源多  UDP不是面向连接的，不存在建立连接，释放连接，每个数据包都是独立的包，有边界，一般不会合并。  TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证  8.  const的含义及实现机制，比如：const int i,是怎么做到i只可读的？  const指示对象为常量，只读。  实现机制：这些在编译期间完成，对于内置类型，如int， 编译器可能使用常数直接替换掉对此变量的引用。而对于结构体不一定。  看下面的例子：   |  | | --- | | const int j=100;    int \*p=const\_cast<int\*>(&j);    \*p=200;    cout<<j<<endl;    输出为什么是100呢？  cout<<\*p<<endl; //输出是改过的200  编译器在优化代码时把cout<<j直接优化成cout<<100了，所以虽然p和&j的值一样，但cout<<j不再通过访问j的地址输出。（反汇编时也有看到直接把数字压栈push 100 ）  这是因为，const型在压栈时，是使用的直接的数，就有点像C的#define a 100 |   对于非系统缺省类型，系统不知道怎么去直接替换，因此必须占据内存。   |  | | --- | | #include <iostream> using namespace std; struct A {    int i;    char ch;    A()    {        i = 100;        ch = 'S';    } }; int main() {    const A a;    const int i = 200;    int \*p1 = (int\*)&a.i;    int \*p2 = (int\*)&i;    \*p1 = 1;    \*p2 = 2; //   a.i = 200; //报错，左值不能为const    cout << a.i << " " << a.ch << endl;    cout << i << endl;    return 0; } |   运行结果：   |  | | --- | | 1 S 200 |   9.volatile的含义。  变量可能在编译器的控制或监控之外改变，告诉编译器不要优化该变量，如被系统时钟更新的变量。  10.OFFSETOF(s, m)的宏定义，s是结构类型，m是s的成员，求m在s中的偏移量。  #define OFFSETOF（s, m） size\_t（&((s\*)0)->m）  11.100亿个数，求最大的1万个数，并说出算法的时间复杂度。  用小根堆来实现。注意是小根堆，  读入1万个数，然后做  时间复杂度是O（NlogK）  12.设计一个洗牌的算法，并说出算法的时间复杂度。  第一种： for i:=1 to n do swap(a[i], a[random(1,n)]);  // 凑合，但不是真正随机  第二种： for i:=1 to n do swap(a[i], a[random(i,n)]);   // 真正的随机算法 其中，random(a,b)函数用于返回一个从a到b（包括a和b）的随机整数。  至于怎么证明上两个算法，没想好。  算法复杂度是O（n。。。），要研究下random的实现。    13.socket在什么情况下可读?  1. 接收缓冲区有数据，一定可读 2. 对方正常关闭socket，也是可读 3. 对于侦听socket，有新连接到达也可读  4.socket有错误发生，且pending～～～  引用unp的一段话 第六章 6.3节    A socket is ready for reading if any of the following four conditions is true: a. The number of bytes of data in the socket receive buffer is greater than or       equal to the current size of the low-water mark for the socket receive buffer.      A read operation on the socket will not block and will return a value greater than 0 b.  The read half of the connections is closed (i.e., A TCP connection that has received a FIN).      A read operation on the socket will not block and will return 0 (i.e., EOF) c. The socket is a listening socket and the number of completed connection is nonzero.      An accept on the listening socket will normally not block, although we will describe a    d. A socket error is pending. A read operation on the socket will not block and will return     an error (-1) with errno set to the specific error condition  14.流量控制与拥塞控制的区别，节点计算机怎样感知网络拥塞了？？？  拥塞控制是把整体看成一个处理对象的，流量控制是对单个的节点。  感知的手段应该不少，比如在TCP协议里，TCP报文的重传本身就可以作为拥塞的依据。依据这样的原理， 应该可以设计出很多手段。    15.C++虚函数是如何实现的？？？  使用虚函数表。 C++对象使用虚表， 如果是基类的实例，对应位置存放的是基类的函数指针；如果是继承类，对应位置存放的是继承类的函数指针（如果在继承类有实现）。所以 ，当使用基类指针调用对象方法时，也会根据具体的实例，调用到继承类的方法。    16.C++的虚函数有什么作用？ ？？  虚函数作用是实现多态，  更重要的，虚函数其实是实现封装，使得使用者不需要关心实现的细节。  在很多设计模式中都是这样用法，例如Factory、Bridge、Strategy模式。  17. 非阻塞connect（）如何实现？ ？？  将socket设置成non-blocking，操作方法同非阻塞read()、write();  18. 以下代码输出结果：  #include <stdio.h> #include <unistd.h> int main(void) { printf("call execl");  sleep(1);  execl("/bin/sh", "", NULL);  printf("error!\n"); }  本题考标准IO缓冲，标准出错是不带缓缓冲的。  如若是涉及终端设备的其他流，则他们是行缓冲的；否则是全缓冲的。    printf是标准IO的一个，格式化打印到标准输出，在这里是行缓冲，那么没有遇到换行符也就是‘\n’或者没有强制flush, 则不会输出。  execl是创建新的可执行程序映像，一旦成功就不会返回了，只有在出错的情况会返回1.  所以以上的程序没有打印printf的内容，直接执行/bin/sh，输出为  $    若是代码改为以下：  #include <stdio.h>  #include <unistd.h>    int main(int argc, char \*argv[])  {      printf("call execl\n");      /\*fprintf(stderr, "%s", "call execl");\*/      sleep(1);      execl("/bin/sh", "", NULL);      printf("error!\n");        return 0;  }  则输出为：  call execl  $  若改为：  #include <stdio.h>  #include <unistd.h>    int main(int argc, char \*argv[])  {      /\*printf("call execl\n");\*/      fprintf(stderr, "%s", "call execl");       // 标准错误，不缓冲      sleep(1);      execl("/bin/sh", "", NULL);      printf("error!\n");        return 0;  }  则输出为：  call execl$  若改为：  #include <stdio.h>  #include <unistd.h>    int main(int argc, char \*argv[])  {      /\*printf("call execl\n");\*/      fprintf(stdout, "%s", "call execl");     // stdout行缓冲      sleep(1);      execl("/bin/sh", "", NULL);      printf("error!\n");        return 0;  }  则输出为  $    19. TCP通讯中，select到读事件，但是读到的数据量是0，为什么，如何解决????  select 返回0代表超时。select出错返回-1。    select到读事件，但是读到的数据量为0，说明对方已经关闭了socket的读端。本端关闭读即可。    当select出错时，会将接口置为可读又可写。这时就要通过判断select的返回值为-1来区分。  20. 给出float与“零值”比较的 if 语句（假设变量名为var）？？？  const float EPSINON = 0.00001;   if ((x >= - EPSINON) && (x <= EPSINON)    浮点数在内存中的存贮机制和整型数不同，有舍入误差，在计算机中用以近似表示任意某个实数。具体的说，这个实数由一个整数或定点数（即尾数）乘以某个基数（计算机中通常是2）的整数次幂得到，这种表示方法类似于基数为10的科学记数法。   所以浮点数在运算过成功运算通常伴随着因为无法精确表示而进行的近似或舍入。但是这种设计的好处是可以在固定的长度上存储更大范围的数。   例如，一个指数范围为±4的4位十进制浮点数可以用来表示43210，4.321或0.0004321，但是没有足够的精度来表示432.123和43212.3（必须近似为432.1和43210）。当然，实际使用的位数通常远大于4。     所以浮点数不能够判断相等，像 if(x==0)的这样的编码是不总是正确的，我们在判断浮点数相等时，推荐用范围来确定，若x在某一范围内，我们就认为相等，至于范围怎么定义，要看实际情况而已了，float,和double 各有不同   所以const float EPSINON = 0.00001;     if ((x >= - EPSINON) && (x <= EPSINON) 这样判断是可取的   至于为什么取0.00001，可以自己按实际情况定义 |

**1. 简述 OSI 七层协议。**  
  
应用层  
与其它计算机进行通讯的一个应用，它是对应应用程序的通信服务的。例如，一个没有通信功能的字处理程序就不能执行通信的代码，  
从事字处理工作的程序员也不关心OSI的第7层。但是，如果添加了一个传输文件的选项，那么字处理器的程序员就需要实现OSI的第7层。示例：TELNET，HTTP，FTP，NFS，SMTP等。  
表示层  
这一层的主要功能是定义数据格式及加密。例如，FTP允许你选择以二进制或ASCII格式传输。如果选择二进制，那么发送方和接收方不改变文件的内容。如果选择ASCII格式，  
发送方将把文本从发送方的字符集转换成标准的ASCII后发送数据。在接收方将标准的ASCII转换成接收方计算机的字符集。示例：加密，ASCII等。  
会话层  
它定义了如何开始、控制和结束一个会话，包括对多个双向消息的控制和管理，以便在只完成连续消息的一部分时可以通知应用，从而使表示层看到的数据是连续的，  
在某些情况下，如果表示层收到了所有的数据，则用数据代表表示层。示例：RPC，SQL等。  
传输层  
这层的功能包括是否选择差错恢复协议还是无差错恢复协议，及在同一主机上对不同应用的数据流的输入进行复用，还包括对收到的顺序不对的数据包的重新排序功能。示例：TCP，UDP，SPX。  
网络层  
这层对端到端的包传输进行定义，它定义了能够标识所有结点的逻辑地址，还定义了路由实现的方式和学习的方式。为了适应最大传输单元长度小于包长度的传输介质，  
网络层还定义了如何将一个包分解成更小的包的分段方法。示例：IP，IPX等。  
数据链路层  
它定义了在单个链路上如何传输数据。这些协议与被讨论的各种介质有关。示例：ATM，FDDI等。  
物理层  
OSI的物理层规范是有关传输介质的特这些规范通常也参考了其他组织制定的标准。连接头、帧、帧的使用、电流、编码及光调制等都属于各种物理层规范中的内容。  
物理层常用多个规范完成对所有细节的定义。示例：Rj45，802.3等。  
  
  
**2. 什么是C/S和B/S架构?**  
C/S架构软件（即客户机/服务器模式）分为客户机和服务器两层：第一层是在客户机系统上结合了表示与业务逻辑，第二层是通过网络结合了数据库服务器。  
简单的说就是第一层是用户表示层，第二层是数据库层。客户端和服务器直接相连，这两个组成部分都承担着重要的角色。  
  
B/S架构第一层是浏览器（即客户端）只有简单的输入输出功能，处理极少部分的事务逻辑。由于客户不需要安装客户端，只要有浏览器就能上网浏览，  
所以它面向的是大范围的用户，所以界面设计得比较简单，通用。第二层是WEB服务器，扮演着信息传送的角色。当用户想要访问数据库时，就会首先向WEB服务器发送请求，  
WEB服务器统一请求后会向数据库服务器发送访问数据库的请求，这个请求是以SQL语句实现的。第三层是数据库服务器，它存放着大量的数据。当数据库服务器收到了WEB服务器的请求后，  
会对SQL语句进行处理，并将返回的结果发送给WEB服务器，接下来，WEB服务器将收到的数据结果转换为HTML文本形式发送给浏览器。  
  
  
**3. 简述 三次握手、四次挥手的流程**  
所谓三次握手（Three - WayHandshake）即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。  
在socket编程中，这一过程由客户端执行connect来触发，整个流程如下图所示：  
  
  
图2  
TCP三次握手  
  
（1）第一次握手：Client将标志位SYN置为1，随机产生一个值seq = J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。  
（2）第二次握手：Server收到数据包后由标志位SYN = 1  
知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack = J + 1，随机产生一个值seq = K，并将该数据包发送给Client以确认连接请求  
，Server进入SYN\_RCVD状态。  
（3）第三次握手：Client收到确认后，检查ack是否为J + 1，ACK是否为1，如果正确则将标志位ACK置为1，ack = K + 1，并将该数据包发送给Server，  
Server检查ack是否为K + 1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，  
随后Client与Server之间可以开始传输数据了。  
  
  
  
所谓四次挥手（Four - Way  
Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。在socket编程中，  
这一过程由客户端或服务端任一方执行close来触发，整个流程如下图所示：  
  
  
图3  
TCP四次挥手  
  
由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，  
收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。  
首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。  
（1）第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。  
（2）第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号 + 1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。  
（3）第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。  
（4）第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号 + 1，Server进入CLOSED状态，完成四次挥手。  
  
  
  
**4. 什么是arp协议?**  
  
答：ARP(AddressResolutionProtocol) 是个地址解析协议。最直白的说法是：在IP以太网中，当一个上层协议要发包时，有了该节点的IP地址，ARP就能提供该节点的MAC地址。  
  
**5.TCP和UDP的区别?**  
  
1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接  
  
2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;  
UDP尽最大努力交付，即不保证可靠交付  
  
Tcp通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。  
  
3、UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。  
  
4.每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信  
  
5、TCP对系统资源要求较多，UDP对系统资源要求较少。  
  
**6.什么是局域网和广域网?**  
答：什么是广域网（WAN、公网、外网），什么是局域网（LAN、私网、内网）？  
广域网（WAN），就是我们通常所说的Internet，它是一个遍及全世界的网络。 局域网（LAN），相对于广域网（WAN）而言，  
主要是指在小范围内的计算机互联网络。这个“小范围”可以是一个家庭，一所学校，一家公司，或者是一个政府部门。 BT中常常提到的公网、  
外网，即广域网（WAN）；BT中常常提到私网、内网，即局域网（LAN）。  
  
广域网上的每一台电脑（或其他网络设备）都有一个或多个广域网IP地址（或者说公网、外网IP地址），广域网IP地址一般要到ISP处交费之后才能申请到，  
广域网IP地址不能重复；局域网（LAN）上的每一台电脑（或其他网络设备）都有一个或多个局域网IP地址（或者说私网、内网IP地址），  
局域网IP地址是局域网内部分配的，不同局域网的IP地址可以重复，不会相互影响。  
  
**7.为何基于tcp协议的通信比基于udp协议的通信更可靠?**  
  
TCP的可靠保证，是它的三次握手双向机制，这一机制保证校验了数据，保证了他的可靠性。而UDP就没有了，udp信息发出后, 不验证是否到达对方, 所以不可靠。  
  
  
**8.什么是socket?简述基于tcp协议的套接字通信流程。**  
答：Socket又称”套接字”, 是系统提供的用于网络通信的方法.它的是指并不是一种协议, 没有规定计算机应当怎么传递信息, 只是给程序员提供了一个发送消息的接口,\  
 程序员能使用这个接口同的方法, 发送与接收消息.Socket描述了一个IP, 端口对.它简化了程序员的操作, 知道对方的IP以及port  
就可以给对方发送消息, 再由服务器处理发送的这些消息, Socket包含了通信的双发, 即客户端与服务端.  
  
TCP编程的服务器端一般步骤是：  
　　1、创建一个socket，用函数socket()； SOCKET  
SocketListen = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  
　　2、设置socket属性，用函数setsockopt();  
\*可选  
　　3、绑定IP地址、端口等信息到socket上，用函数bind();  
SOCKET\_ERROR = bind(SocketListen, (const  
sockaddr \*) & addr, sizeof(addr))  
　　4、开启监听，用函数listen()； SOCKET\_ERROR == listen(SocketListen, 2)  
　　5、接收客户端上来的连接，用函数accept()； SOCKET  
SocketWaiter = accept(SocketListen,  
  
 \_Out\_  
struct  
sockaddr \* addr  
  
\_Inout\_  
int \* addrlen);  
6、收发数据，用函数send()  
和recv()，或者read()  
和write();  
7、关闭网络连接； closesocket(SocketListen);  
closesocket(SocketWaiter);  
8、关闭监听；  
SOCK\_STREAM这种的特点是面向连接的，即每次收发数据之前必须通过connect建立连接，而SOCK\_DGRAM这种是User  
Datagram  
Protocol协议的网络通讯，它是无连接的，不可靠的。  
TCP编程的客户端一般步骤是：  
　　1、创建一个socket，用函数socket()；  
　　2、设置socket属性，用函数setsockopt();  
\*可选  
　　3、绑定IP地址、端口等信息到socket上，用函数bind();  
\*可选  
　　4、设置要连接的对方的IP地址和端口等属性；  
　　5、连接服务器，用函数connect()；  
　　6、收发数据，用函数send()  
和recv()，或者read()  
和write();  
　　7、关闭网络连接；  
  
intsend(  
  
\_In\_SOCKETs, // 向哪个socket发送，accept返回的socket。  
  
\_In\_const char \* buf,  
  
\_In\_ int len,  
  
\_In\_ int flags  
  
);  
  
**9.什么是粘包? socket 中造成粘包的原因是什么? 哪些情况会发生粘包现象?**  
粘包：多个数据包被连续存储于连续的缓存中，在对数据包进行读取时由于无法确定发生方的发送边界，而采用某一估测值大小来进行数据读出，  
若双方的size不一致时就会使指发送方发送的若干包数据到接收方接收时粘成一包，从接收缓冲区看，后一包数据的头紧接着前一包数据的尾。  
  
出现粘包现象的原因是多方面的，它既可能由发送方造成，也可能由接收方造成。发送方引起的粘包是由TCP协议本身造成的，TCP为提高传输效率，  
发送方往往要收集到足够多的数据后才发送一包数据。若连续几次发送的数据都很少，通常TCP会根据优化算法把这些数据合成一包后一次发送出去，  
这样接收方就收到了粘包数据。接收方引起的粘包是由于接收方用户进程不及时接收数据，从而导致粘包现象。这是因为接收方先把收到的数据放在系统接收缓冲区，  
用户进程从该缓冲区取数据，若下一包数据到达时前一包数据尚未被用户进程取走，则下一包数据放到系统接收缓冲区时就接到前一包数据之后，  
而用户进程根据预先设定的缓冲区大小从系统接收缓冲区取数据，这样就一次取到了多包数据。分包是指在出现粘包的时候我们的接收方要进行分包处理。  
（在长连接中都会出现） 数据包的边界发生错位，导致读出错误的数据分包，进而曲解原始数据含义。  
  
当时短连接的情况下，不用考虑粘包的情况  
2.  
如果发送数据无结构，如文件传输，这样发送方只管发送，接收方只管接收存储就ok，也不用考虑粘包  
3.  
如果双方建立连接，需要在连接后一段时间内发送不同结构数据  
处理方式：  
接收方创建一预处理线程，对接收到的数据包进行预处理，将粘连的包分开  
注：粘包情况有两种，一种是粘在一起的包都是完整的数据包，另一种情况是粘在一起的包有不完整的包  
  
**10.IO多路复用的作用?**  
  
　　IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程。IO多路复用适用如下场合：  
  
　　（1）当客户处理多个描述字时（一般是交互式输入和网络套接口），必须使用I / O复用。  
  
　　（2）当一个客户同时处理多个套接口时，而这种情况是可能的，但很少出现。  
  
　　（3）如果一个TCP服务器既要处理监听套接口，又要处理已连接套接口，一般也要用到I / O复用。  
  
　　（4）如果一个服务器即要处理TCP，又要处理UDP，一般要使用I / O复用。  
  
　　（5）如果一个服务器要处理多个服务或多个协议，一般要使用I / O复用。  
  
　　与多进程和多线程技术相比，I / O多路复用技术的最大优势是系统开销小，系统不必创建进程 / 线程，也不必维护这些进程 / 线程，  
从而大大减小了系统的开销。  
  
  
  
**11.什么是防火墙以及作用?**  
  
什么是防火墙  
防火墙是指设置在不同网络（如可信任的企业内部网和不可信的公共网）或网络安全域之间的一系列部件的组合。 它可通过监测、限制、  
更改跨越防火墙的数据流，尽可能地对外部屏蔽网络内部的信息、结构和运行状况， 以此来实现网络的安全保护。  
在逻辑上，防火墙是一个分离器，一个限制器，也是一个分析器，有效地监控了内部网和Internet之间的任何活动， 保证了内部网络的安全。  
2.  
使用Firewall的益处  
保护脆弱的服务  
通过过滤不安全的服务，Firewall可以极大地提高网络安全和减少子网中主机的风险。例如， Firewall可以禁止NIS、NFS服务通过，  
Firewall同时可以拒绝源路由和ICMP重定向封包。  
控制对系统的访问  
Firewall可以提供对系统的访问控制。如允许从外部访问某些主机，同时禁止访问另外的主机。例如， Firewall允许外部访问特定的Mail  
Server和Web  
Server。  
集中的安全管理  
Firewall对企业内部网实现集中的安全管理，在Firewall定义的安全规则可以运行于整个内部网络系统， 而无须在内部网每台机器上分别设立安全策略。  
Firewall可以定义不同的认证方法， 而不需要在每台机器上分别安装特定的认证软件。外部用户也只需要经过一次认证即可访问内部网。  
增强的保密性  
使用Firewall可以阻止攻击者获取攻击网络系统的有用信息，如Figer和DNS。  
记录和统计网络利用数据以及非法使用数据  
Firewall可以记录和统计通过Firewall的网络通讯，提供关于网络使用的统计数据，并且，Firewall可以提供统计数据， 来判断可能的攻击和探测。  
策略执行  
Firewall提供了制定和执行网络安全策略的手段。未设置Firewall时，网络安全取决于每台主机的用户。  
 **12.select、poll、epoll 模型的区别?**  
  
select本质上是通过设置或者检查存放fd标志位的数据结构来进行下一步处理。  
这样所带来的缺点是：  
  
1  
单个进程可监视的fd数量被限制  
  
2  
需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大  
  
3  
对socket进行扫描时是线性扫描  
  
poll本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态，如果设备就绪则在设备等待队列中加入一项并继续遍历，  
  
如果遍历完所有fd后没有发现就绪设备，则挂起当前进程，直到设备就绪或者主动超时，被唤醒后它又要再次遍历fd。这个过程经历了多次无谓的遍历。  
  
它没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点：大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。  
  
poll还有一个特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。  
  
epoll支持水平触发和边缘触发，最大的特点在于边缘触发，它只告诉进程哪些fd刚刚变为就需态，并且只会通知一次。  
  
在前面说到的复制问题上，epoll使用mmap减少复制开销。  
  
还有一个特点是，epoll使用“事件”的就绪通知方式，通过epoll\_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，  
epoll\_wait便可以收到通知  
  
**13．简述 进程、线程、协程的区别 以及应用场景?**  
1.  
线程和进程：  
线程是属于进程的，线程运行在进程空间内，同一进程所产生的线程共享同一内存空间，当进程退出时该进程所产生的线程都会被强制退出并清除。  
线程可与属于同一进程的其它线程共享进程所拥有的全部资源，但是其本身基本上不拥有系统资源，只拥有一点在运行中必不可少的信息(  
 如程序计数器、一组寄存器和栈)。  
2.  
线程、进程与协程：  
线程和进程的操作是由程序触发系统接口，最后的执行者是系统；协程的操作则是程序员  
协程存在的意义：对于多线程应用，CPU通过切片的方式来切换线程间的执行，线程切换时需要耗时（保持状态，下次继续）。协程，则只使用一个线程，  
在一个线程中规定某个代码块执行顺序。  
协程的适用场景： 当程序中存在大量不需要CPU的操作时（IO），适用于协程；  
  
  
  
**14.GIL锁是什么鬼?**  
  
GIL即全局解释器锁，全局解释器锁的存在，在同一时间内，python解释器只能运行一个线程的代码，这大大影响了python多线程的性能。而这个  
解释器锁由于历史原因，现在几乎无法消除。  
  
python GIL 之所以会影响多线程等性能，是因为在多线程的情况下，只有当线程获得了一个全局锁的时候，那么该线程的代码才能运行，而全局锁只有一个，  
所以使用python多线程，在同一时刻也只有一个线程在运行，因此在即使在多核的情况下也只能发挥出单核的性能。  
  
  
  
**15. Python中如何使用线程池和进程池?**  
答：Threadpool和ProcessPool模块。  
  
**16.threading.local的作用?**  
答：Threading.local可以创建一个对象, 每个线程都可以对他读写属性, 但不会互相影响  
  
**17.进程之间如何进行通信?**  
答：进程间通信（IPC，Interprocess  
communication），方法包括管道（PIPE）、消息排队、旗语、共用内存以及套接字（Socket）。  
  
**18.什么是并发和并行?**  
答：并发是指一个处理器同时处理多个任务。  
并行是指多个处理器或者是多核的处理器同时处理多个不同的任务。  
并发是逻辑上的同时发生（simultaneous），而并行是物理上的同时发生。  
  
  
**19.进程锁和线程锁的作用?**  
答：　线程锁: 大家都不陌生，主要用来给方法、代码块加锁。当某个方法或者代码块使用锁时，那么在同一时刻至多仅有有一个线程在执行该段代码。  
当有多个线程访问同一对象的加锁方法 / 代码块时，同一时间只有一个线程在执行，其余线程必须要等待当前线程执行完之后才能执行该代码段。  
但是，其余线程是可以访问该对象中的非加锁代码块的。  
  
　　进程锁: 也是为了控制同一操作系统中多个进程访问一个共享资源，只是因为程序的独立性，各个进程是无法控制其他进程对资源的访问的，  
但是可以使用本地系统的信号量控制（操作系统基本知识）。  
  
  
**20．解释什么是异步非阻塞?**  
答：Asynchronous I / O, or non - blocking I / O, is a form of input / output processing that permits other processing  
to continue before the transmission has finished  
  
**21.路由器和交换机的区别?**  
  
（1）工作层次不同 　　  
  
最初的的交换机是工作在OSI / RM开放体系结构的数据链路层，也就是第二层，而路由器一开始就设计工作在OSI模型的网络层。  
由于交换机工作在OSI的第二层（数据链路层），所以它的工作原理比较简单，而路由器工作在OSI的第三层（网络层），可以得到更多的协议信息，  
路由器可以做出更加智能的转发决策。 　　  
（2）数据转发所依据的对象不同 　　  
  
交换机是利用物理地址或者说MAC地址来确定转发数据的目的地址。而路由器则是利用不同网络的ID号（即IP地址）来确定数据转发的地址。  
IP地址是在软件中实现的，描述的是设备所在的网络，有时这些第三层的地址也称为协议地址或者网络地址。MAC地址通常是硬件自带的，由网卡生产商来分配的，  
而且已经固化到了网卡中去，一般来说是不可更改的。而IP地址则通常由网络管理员或系统自动分配。 　　  
（3）传统的交换机只能分割冲突域，不能分割广播域；而路由器可以分割广播域由交换机连接的网段仍属于同一个广播域，广播数据包会在交换机连接的所有网段上传播，  
在某些情况下会导致通信拥挤和安全漏洞。连接到路由器上的网段会被分配成不同的广播域，广播数据不会穿过路由器。虽然第三层以上交换机具有VLAN功能，  
也可以分割广播域，但是各子广播域之间是不能通信交流的，它们之间的交流仍然需要路由器。 　　  
（4）路由器提供了防火墙的服务 　  
  
路由器仅仅转发特定地址的数据包，不传送不支持路由协议的数据包传送和未知目标网络数据包的传送，从而可以防止广播风暴。 　　　  
  
**22.什么是域名解析?**  
  
域名解析是把域名指向网站空间IP，让人们通过注册的域名可以方便地访问到网站的一种服务。IP地址是网络上标识站点的数字地址，为了方便记忆，  
采用域名来代替IP地址标识站点地址。域名解析就是域名到IP地址的转换过程。域名的解析工作由DNS服务器完成。  
  
  
**23.如何修改本地hosts文件?**  
  
hosts  
文件所在位置  
  
c: / windows / system32 / drivers / etc / hosts  
  
左下角 搜索框 搜索 cmd  
  
弹出命令框  
输入  
ipconfig / displaydns  
  
显示所有 dns内容  
  
ipconfig / flushdns  
  
刷新所有 dns内容  
  
  
**24.生产者消费者模型应用场景及优势?**  
生产者与消费者模式是通过一个容器来解决生产者与消费者的强耦合关系，生产者与消费者之间不直接进行通讯，而是利用阻塞队列来进行通讯，  
生产者生成数据后直接丢给阻塞队列，消费者需要数据则从阻塞队列获取，实际应用中，生产者与消费者模式则主要解决生产者与消费者生产与消费的速率不一致的问题，  
达到平衡生产者与消费者的处理能力，而阻塞队列则相当于缓冲区。  
  
还有一个比较典型的例子便是日志的记录，多线程产生日志，但写日志由于文件独占，不能多线程来写，于是我们就可以把线程压入队列，由日志线程来读取队列数据，完成写日志的操作。  
  
  
  
**25.什么是cdn?**  
CDN的全称是Content  
Delivery  
Network，即内容分发网络。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。  
通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络，CDN系统能够实时地根据网络流量和各节点的连接、  
负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决  
Internet网络拥挤的状况，提高用户访问网站的响应速度。  
  
  
**26.LVS是什么及作用?**  
  
LVS是Linux Virtual Server ，Linux虚拟服务器；是一个虚拟的服务器集群【多台机器LBIP】。LVS  
集群为三层结构:  
  
负载调度器(loadbalancer)：它是整个LVS集群对外的前端机器，负责将client请求发送到一组服务器[多台LBIP]上执行，而client端认为是返回来一个同一个IP【通常把这个IP称为虚拟IP / VIP】  
  
服务器池(serverpool)：一组真正执行client请求的服务器，一般是我们的web服务器；除了web，还有FTP，MAIL，DNS  
  
共享存储(sharedstored)：它为server pool提供了一个共享的存储区，很容易让服务器池拥有相同的内容，提供相同的服务  
  
**27.Nginx是什么及作用?**  
  
1、静态HTTP服务器  
  
首先，Nginx是一个HTTP服务器，可以将服务器上的静态文件（如HTML、图片）通过HTTP协议展现给客户端。  
  
2、反向代理服务器  
  
什么是反向代理？  
  
客户端本来可以直接通过HTTP协议访问某网站应用服务器，网站管理员可以在中间加上一个Nginx，客户端请求Nginx，Nginx请求应用服务器，  
然后将结果返回给客户端，此时Nginx就是反向代理服务器。  
  
3、负载均衡  
  
当网站访问量非常大，网站站长开心赚钱的同时，也摊上事儿了。因为网站越来越慢，一台服务器已经不够用了。于是将同一个应用部署在多台服务器上，  
将大量用户的请求分配给多台机器处理。同时带来的好处是，其中一台服务器万一挂了，只要还有其他服务器正常运行，就不会影响用户使用。  
  
4、虚拟主机  
  
有的网站访问量大，需要负载均衡。然而并不是所有网站都如此出色，有的网站，由于访问量太小，需要节省成本，将多个网站部署在同一台服务器上。  
  
例如将www.aaa.com和www.bbb.com两个网站部署在同一台服务器上，两个域名解析到同一个IP地址，但是用户通过两个域名却可以打开两个完全不同的网站，  
互相不影响，就像访问两个服务器一样，所以叫两个虚拟主机。  
  
5、FastCGI  
  
Nginx本身不支持PHP等语言，但是它可以通过FastCGI来将请求扔给某些语言或框架处理（例如PHP、Python、Perl）。  
  
  
**28.keepalived是什么及作用?**  
  
eepalived是一个类似于Layer2, 4, 7  
交换机制的软件。是Linux集群管理中保证集群高可用的一个服务软件，其功能是用来防止单点故障。  
  
  
  
keepalived的工作原理：  
keepalived是基于VRRP协议实现的保证集群高可用的一个服务软件，主要功能是实现真机的故障隔离和负载均衡器间的失败切换，防止单点故障。在了解keepalived原理之前先了解一下VRRP协议。  
  
VRRP协议：  
Virtual Route Redundancy Protocol虚拟路由冗余协议。是一种容错协议，保证当主机的下一跳路由出现故障时，由另一台路由器来代替出现故障的路由器进行工作，  
从而保持网络通信的连续性和可靠性。

**29.haproxy是什么以及作用?**  
  
HAProxy是一个使用C语言编写的自由及开放源代码软件，其提供高可用性、负载均衡，以及基于TCP和HTTP的应用程序代理。  
  
HAProxy特别适用于那些负载特大的web站点，这些站点通常又需要会话保持或七层处理。HAProxy运行在当前的硬件上  
，完全可以支持数以万计的并发连接。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中， 同时可以保护你的web服务器不被暴露到网络上。  
  
HAProxy实现了一种事件驱动, 单一进程模型，此模型支持非常大的并发连接数。多进程或多线程模型受内存限制 、系统调度器限制以及无处不在的锁限制，  
很少能处理数千并发连接。事件驱动模型因为在有更好的资源和时间管理的用户空间(  
 User - Space)  
实现所有这些任务，所以没有这些问题。此模型的弊端是，在多核系统上，这些程序通常扩展性较差。这就是为什么他们必须进行优化以  
使每个CPU时间片(Cycle)  
做更多的工作。  
  
　  
**30. 什么是负载均衡?**  
  
负载平衡也称负载共享，是指对系统中的负载情况进行动态调整，以尽量消除或减少系统中各节点负载不均衡的现象。  
具体实现方法是将过载节点上的任务转移到其他轻载节点上，尽可能实现系统各节点的负载平衡，从而提高系统的吞吐量。  
负载共享有利于统筹管理分布式系统中的各种资源，便于利用共享信息及其服务机制扩大系统的处理能力。  
动态负载共享策略是指把系统中各节点上已有的负载作为参考信息，在运行过程中，根据系统中各节点的负载状况，  
随时调整负载的分配，使各节点尽可能保持负载的平衡。  
  
  
  
**31.什么是rpc及应用场景?**  
  
调用本地服务一样调用远程服务，而让调用者对网络通信这些细节透明，那么将大大提高生产力，比如服务消费方在执行helloWorldService.sayHello(“test”)时，  
实质上调用的是远端的服务。这种方式其实就是RPC（Remote Procedure Call Protocol）RPC调用的流程：  
  
1）服务消费方（client）调用以本地调用方式调用服务；  
  
2）client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；  
  
3）client stub找到服务地址，并将消息发送到服务端；  
  
4）server stub收到消息后进行解码；  
  
5）server stub根据解码结果调用本地的服务；  
  
6）本地服务执行并将结果返回给server stub；  
  
7）server stub将返回结果打包成消息并发送至消费方；  
  
8）client stub接收到消息，并进行解码；  
  
9）服务消费方得到最终结果。  
  
  
**32.简述 asynio模块的作用和应用场景。**  
  
asyncio模块提供了一种关于事件循环的框架。事件循环就是等待一些任务发生，然后执行相应的事件。它也会处理例如IO操作或者系统事件。  
asyncio实际中有好几种循环实现方式。模块默认使用的方式是其所运行的操作系统上最有效的方式。如果你愿意，你也可以显式地选择其它事件循环方式。  
一个事件循环就是当事件A发生时，函数B共同起作用。  
  
设想这样一个场景，服务器等待用户访问并请求一些资源，例如网页。如果这个网站不是非常知名的网站，这个服务器将会在很长的时间内处于空闲状态。  
但是，一旦某个时间用户点击了这个网站，服务器就需要作出响应。这个响应就是事件处理。当一个用户下载网页，服务器将会去检查并调用一个或者多个事件句柄。  
一旦这些事件句柄完成相应的处理，它们需要将控制交回给事件循环。为了在Python中完成这个任务，asyncio使用协程。  
  
协程是一个特殊的函数，可以将控制交回给它的调用函数，但是并不丢失它的状态。协程是一个消费者函数，并且是生成器的扩展。  
协程相比线程最大的优势就是执行协程时不需要占用太多内存。你需要注意的是，当你调用一个协程函数，它并没有真正执行。  
相反，它将会返回一个协程对象，你可以将这个协程对象传递给事件循环，然后可以立即或者稍后执行它。  
  
当你在使用asyncio模块时，另一个你可能会执行的是future。future就是一个可以表示还没有结束的任务结果的对象。  
你的事件循环可以观察future对象并等待它们结束。当一个future结束时，它被设置为已完成。asyncio模块也支持锁和信号。  
  
**33.简述 gevent模块的作用和应用场景。**  
  
gevent是第三方库，通过greenlet实现协程，其基本思想是：当一个greenlet遇到IO操作时，比如访问网络，就自动切换到其他的greenlet，等到IO操作完成，  
再在适当的时候切换回来继续执行。由于IO操作非常耗时，经常使程序处于等待状态，有了gevent为我们自动切换协程，就保证总有greenlet在运行，而不是等待IO。  
  
由于切换是在IO操作时自动完成，所以gevent需要修改Python自带的一些标准库，这一过程在启动时通过monkey  
patch完成  
  
使用gevent，可以获得极高的并发性能，但gevent只能在Unix / Linux下运行，在Windows下不保证正常安装和运行。  
  
由于gevent是基于IO切换的协程，所以最神奇的是，我们编写的Web  
App代码，不需要引入gevent的包，也不需要改任何代码，仅仅在部署的时候，用一个支持gevent的WSGI服务器，立刻就获得了数倍的性能提升。  
  
  
**34.twisted框架的使用和应用?**  
Twisted应用的基本问题，可说是“一个中心，两个基本点”，即：  
  
以“事件”event为中心，以  
"建立连接"  
connect和“定义反馈“callback为基本点。  
  
Twisted  
对event  
的管理机制，可划分为后台和前台两种形式。  
  
后台的管理，是Twisted  
框架的内在机制，自动运行，对程序员透明无须干预，在程序文本中不见其踪迹。  
  
前台的管理，是Twisted  
授权程序员，在程序文本中显式写码来实现。程序员的工作，主要是按照既定的方式，实现  
event。我们所关心、所用到的，是这部分东西（API）。

1、TCP和UDP的区别：

1）TCP提供面向连接的传输，通信前要先建立连接（三次握手机制）；UDP提供无连接的传输，通信前不需要建立连接。

2）TCP提供可靠的传输（有序，无差错，不丢失，不重复）；UDP提供不可靠的传输。

3）TCP面向字节流的传输，因此它能将信息分割成组，并在接收端将其重组；UDP是面向数据报的传输，没有分组开销。

4）TCP提供拥塞控制和流量控制机制；UDP不提供拥塞控制和流量控制机制。

2、流量控制和拥塞控制的实现机制：

1）TCP采用大小可变的滑动窗口机制实现流量控制功能。窗口的大小是字节。在TCP报文段首部的窗口字段写入的数值就是当前给对方设置发送窗口的数据的上限。  
在数据传输过程中，TCP提供了一种基于滑动窗口协议的流量控制机制，用接收端接收能力（缓冲区的容量）的大小来控制发送端发送的数据量。

2）采用滑动窗口机制还可对网络进行拥塞控制，将网络中的分组（TCP报文段作为其数据部分）数量维持在一定的数量之下，当超过该数值时，网络的性能会急剧恶化。传输层的拥塞控制有慢开始（Slow-Start）、拥塞避免（Congestion Avoidance）、快重传（Fast Retransmit）和快恢复（Fast Recovery）四种算法。  
拥塞：　大量数据报涌入同一交换节点（如路由器），导致该节点资源耗尽而必须丢弃后面到达的数据报时，就是拥塞。

3、重传机制：

TCP每发送一个报文段，就设置一次定时器。只要定时器设置的重发时间到而还没有收到确认，就要重发这一报文段。   
TCP环境

报文往返时间不定、有很大差别  
A、B在一个局域网络，往返时延很小  
A、C在一个互联网内，往返时延很大  
因此，A很难确定一个固定的、与B、C通信都适用的定时器时间

TCP采用了一种自适应算法。这种算法记录每一个报文段发出的时间，以及收到相应的确认报文段的时间。这两个时间之差就是报文段的往返时延。将各个报文段的往返时延样本加权平均，就得出报文段的平均往返时延T。  
4、滑动窗口机制：  
TCP 采用大小可变的滑动窗口进行流量控制。窗口大小的单位是字节。  
在 TCP 报文段首部的窗口字段写入的数值就是当前给对方设置的发送窗口数值的上限。发送窗口在连接建立时由双方商定。但在通信的过程中，接收端可根据自己的资源情况，随时动态地调整对方的发送窗口上限值(可增大或减小)。

5、多线程如何同步：

临界区、互斥区、事件、信号量四种方式  
临界区（Critical Section）、互斥量（Mutex）、信号量（Semaphore）、事件（Event）的区别  
1）、临界区：通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。在任意时刻只允许一个线程对共享资源进行访问，如果有多个线程试图访问公共资源，那么在有一个线程进入后，其他试图访问公共资源的线程将被挂起，并一直等到进入临界区的线程离开，临界区在被释放后，其他线程才可以抢占。  
2）、互斥量：采用互斥对象机制。 只有拥有互斥对象的线程才有访问公共资源的权限，因为互斥对象只有一个，所以能保证公共资源不会同时被多个线程访问。互斥不仅能实现同一应用程序的公共资源安全共享，还能实现不同应用程序的公共资源安全共享 .互斥量比临界区复杂。因为使用互斥不仅仅能够在同一应用程序不同线程中实现资源的安全共享，而且可以在不同应用程序的线程之间实现对资源的安全共享。  
3）、信号量：它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目 .信号量对象对线程的同步方式与前面几种方法不同，信号允许多个线程同时使用共享资源，这与操作系统中的PV操作相同。它指出了同时访问共享资源的线程最大数目。它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目。

PV操作及信号量的概念都是由荷兰科学家E.W.Dijkstra提出的。信号量S是一个整数，S大于等于零时代表可供并发进程使用的资源实体数，但S小于零时则表示正在等待使用共享资源的进程数。  
　　 P操作申请资源：  
　　（1）S减1；  
　　（2）若S减1后仍大于等于零，则进程继续执行；  
　　（3）若S减1后小于零，则该进程被阻塞后进入与该信号相对应的队列中，然后转入进程调度。  
　　  
　　V操作 释放资源：  
　　（1）S加1；  
　　（2）若相加结果大于零，则进程继续执行；  
　　（3）若相加结果小于等于零，则从该信号的等待队列中唤醒一个等待进程，然后再返回原进程继续执行或转入进程调度。  
4）、事 件： 通过通知操作的方式来保持线程的同步，还可以方便实现对多个线程的优先级比较的操作 .

总结：  
　　1） 互斥量与临界区的作用非常相似，但互斥量是可以命名的，也就是说它可以跨越进程使用。所以创建互斥量需要的资源更多，所以如果只为了在进程内部是用的话使用临界区会带来速度上的优势并能够减少资源占用量。因为互斥量是跨进程的互斥量一旦被创建，就可以通过名字打开它。  
　　2） 互斥量（Mutex），信号灯（Semaphore），事件（Event）都可以被跨越进程使用来进行同步数据操作，而其他的对象与数据同步操作无关，但对于进程和线程来讲，如果进程和线程在运行状态则为无信号状态，在退出后为有信号状态。所以可以使用WaitForSingleObject来等待进程和线程退出。  
　　3） 通过互斥量可以指定资源被独占的方式使用，但如果有下面一种情况通过互斥量就无法处理，比如现在一位用户购买了一份三个并发访问许可的数据库系统，可以根据用户购买的访问许可数量来决定有多少个线程/进程能同时进行数据库操作，这时候如果利用互斥量就没有办法完成这个要求，信号灯对象可以说是一种资源计数器。

6、进程和线程的区别：

答：线程是指进程内的一个执行单元,也是进程内的可调度实体。与进程的区别:

(1)调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位。

(2)并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行。

(3)拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.

(4)系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

7、进程间通讯的方式有哪些，各有什么优缺点：

1）管道：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程之间使用。进程的亲缘关系通常是指父子进程关系。

2）有名管道（FIFO）：有名管道也是半双工的通信方式，但是允许在没有亲缘关系的进程之间使用，管道是先进先出的通信方式。

3）信号量：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

4）消息队列：消息队列是有消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

5）信号 ( sinal ) ：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

6）共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。  
7）套接字( socket ) ：套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

8、tcp连接建立的时候3次握手的具体过程，以及每一步原因：

（1）    第一步：源主机A的TCP向主机B发出连接请求报文段，其首部中的SYN(同步)标志位应置为1，表示想与目标主机B进行通信，并发送一个同步序列号X(例：SEQ=100)进行同步，表明在后面传送数据时的第一个数据字节的序号是X＋1（即101）。SYN同步报文会指明客户端使用的端口以及TCP连接的初始序号。  
（2）　　第二步：目标主机B的TCP收到连接请求报文段后，如同意，则发回确认。在确认报中应将ACK位和SYN位置1，表示客户端的请求被接受。确认号应为X＋1(图中为101)，同时也为自己选择一个序号Y。  
（3）　　第三步：源主机A的TCP收到目标主机B的确认后要向目标主机B给出确认，其ACK置1，确认号为Y＋1，而自己的序号为X＋1。TCP的标准规定，SYN置1的报文段要消耗掉一个序号。  
　　运行客户进程的源主机A的TCP通知上层应用进程，连接已经建立。当源主机A向目标主机B发送第一个数据报文段时，其序号仍为X＋1，因为前一个确认报文段并不消耗序号。  
　　当运行服务进程的目标主机B的TCP收到源主机A的确认后，也通知其上层应用进程，连接已经建立。至此建立了一个全双工的连接。

9、tcp断开连接的具体过程，其中每一步是为什么那么做：

1)第一步：源主机A的应用进程先向其TCP发出连接释放请求，并且不再发送数据。TCP通知对方要释放从A到B这个方向的连接，将发往主机B的TCP报文段首部的终止比特FIN置1，其序号X等于前面已传送过的数据的最后一个字节的序号加1。

2)第二步：目标主机B的TCP收到释放连接通知后即发出确认，其序号为Y，确认号为X＋1，同时通知高层应用进程，这样，从A到B的连接就释放了，连接处于半关闭状态，相当于主机A向主机B说：“我已经没有数据要发送了。但如果还发送数据，我仍接收。”此后，主机B不再接收主机A发来的数据。但若主机B还有一些数据要发送主机A，则可以继续发送。主机A只要正确收到数据，仍应向主机B发送确认。

3)第三步：若主机B不再向主机A发送数据，其应用进程就通知TCP释放连接。主机B发出的连接释放报文段必须将终止比特FIN和确认比特ACK置1，并使其序号仍为Y，但还必须重复上次已发送过的ACK＝X＋1。

4) 第四步：主机A必须对此发出确认，将ACK置1，ACK＝Y＋1，而自己的序号是X＋1。这样才把从B到A的反方向的连接释放掉。主机A的TCP再向其应用进程报告，整个连接已经全部释放。

10、tcp建立连接和断开连接的各种过程中的状态转换细节：

客户端：主动打开SYN\_SENT--->ESTABLISHED--->主动关闭FIN\_WAIT\_1--->FIN\_WAIT\_2--->TIME\_WAIT

服务器端：LISTEN（被动打开）--->SYN\_RCVD--->ESTABLISHED--->CLOSE\_WAIT(被动关闭)--->LAST\_ACK--->CLOSED

11、epool与select的区别：

问题的引出，当需要读两个以上的I/O的时候，如果使用阻塞式的I/O，那么可能长时间的阻塞在一个描述符上面，另外的描述符虽然有数据但是不能读出来，这样实时性不能满足要求，大概的解决方案有以下几种：

1.使用多进程或者多线程，但是这种方法会造成程序的复杂，而且对与进程与线程的创建维护也需要很多的开销。（Apache服务器是用的子进程的方式，优点可以隔离用户）

2.用一个进程，但是使用非阻塞的I/O读取数据，当一个I/O不可读的时候立刻返回，检查下一个是否可读，这种形式的循环为轮询（polling），这种方法比较浪费CPU时间，因为大多数时间是不可读，但是仍花费时间不断反复执行read系统调用。

3.异步I/O（asynchronous I/O），当一个描述符准备好的时候用一个信号告诉进程，但是由于信号个数有限，多个描述符时不适用。

4.一种较好的方式为I/O多路转接（I/O multiplexing）（貌似也翻译多路复用），先构造一张有关描述符的列表（epoll中为队列），然后调用一个函数，直到这些描述符中的一个准备好时才返回，返回时告诉进程哪些I/O就绪。select和epoll这两个机制都是多路I/O机制的解决方案，select为POSIX标准中的，而epoll为Linux所特有的。

区别（epoll相对select优点）主要有三：

1.select的句柄数目受限，在linux/posix\_types.h头文件有这样的声明：#define \_\_FD\_SETSIZE    1024  表示select最多同时监听1024个fd。而epoll没有，它的限制是最大的打开文件句柄数目。

2.epoll的最大好处是不会随着FD的数目增长而降低效率，在selec中采用轮询处理，其中的数据结构类似一个数组的数据结构，而epoll是维护一个队列，直接看队列是不是空就可以了。epoll只会对"活跃"的socket进行操作---这是因为在内核实现中epoll是根据每个fd上面的callback函数实现的。那么，只有"活跃"的socket才会主动的去调用 callback函数（把这个句柄加入队列），其他idle状态句柄则不会，在这点上，epoll实现了一个"伪"AIO。但是如果绝大部分的I/O都是“活跃的”，每个I/O端口使用率很高的话，epoll效率不一定比select高（可能是要维护队列复杂）。

3.使用mmap加速内核与用户空间的消息传递。无论是select,poll还是epoll都需要内核把FD消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll是通过内核于用户空间mmap同一块内存实现的。

12、epool中et和lt的区别与实现原理：

epoll有2种工作方式:LT和ET。  
LT(level triggered)是缺省的工作方式，并且同时支持block和no-block socket.在这种做法中，内核告诉你一个文件描述符是否就绪了，然后你可以对这个就绪的fd进行IO操作。如果你不作任何操作，内核还是会继续通知你 的，所以，这种模式编程出错误可能性要小一点。传统的select/poll都是这种模型的代表．  
ET (edge-triggered)是高速工作方式，只支持no-block socket。在这种模式下，当描述符从未就绪变为就绪时，内核通过epoll告诉你。然后它会假设你知道文件描述符已经就绪，并且不会再为那个文件描述 符发送更多的就绪通知，直到你做了某些操作导致那个文件描述符不再为就绪状态了(比如，你在发送，接收或者接收请求，或者发送接收的数据少于一定量时导致 了一个EWOULDBLOCK 错误）。但是请注意，如果一直不对这个fd作IO操作(从而导致它再次变成未就绪)，内核不会发送更多的通知(only once),不过在TCP协议中，ET模式的加速效用仍需要更多的benchmark确认。  
    epoll只有epoll\_create,epoll\_ctl,epoll\_wait 3个系统调用。

13、写一个server程序需要注意哪些问题：

14、ThreadLocal与其它同步机制的比较：

Threadlocal和其他所有的同步机制都是为了解决多线程中的对同一变量的访问冲突，在普通的同步机制中，是通过对对象加锁来实现多个线程对同一变量的安全访问的。这时该变量是多个线程共享的，使用这种同步机制需要很细致的分析在什么时候对变量进行读写，什么时候需要锁定某个对象，什么时候释放该对象的索等等。所有这些都是因为多个线程共享了该资源造成的。Threadlocal就从另一个角度来解决多线程的并发访问，Threadlocal会为每一个线程维护一个和该线程绑定的变量副本，从而隔离了多个线程的数据共享，每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal。

总结：当然ThreadLocal并不能替代同步机制，两者面向的问题领域不同。同步机制是为了同步多个线程对相同资源的并发访问，是为了多个线程之间进行通信的有效方式；而ThreadLocal是隔离多个线程的数据共享，从根本上就不在多个线程之间共享资源（变量），这样当然不需要对多个线程进行同步了。所以，如果你需要进行多个线程之间进行通信，则使用同步机制；如果需要隔离多个线程之间的共享冲突，可以使用ThreadLocal，这将极大地简化你的程序，使程序更加易读、简洁。

15、内存池、进程池、线程池：

自定义内存池的思想通过这个"池"字表露无疑，应用程序可以通过系统的内存分配调用预先一次性申请适当大小的内存作为一个内存池，之后应用程序自己对内存的分配和释放则可以通过这个内存池来完成。只有当内存池大小需要动态扩展时，才需要再调用系统的内存分配函数，其他时间对内存的一切操作都在应用程序的掌控之中。 应用程序自定义的内存池根据不同的适用场景又有不同的类型。 从线程安全的角度来分，内存池可以分为单线程内存池和多线程内存池。单线程内存池整个生命周期只被一个线程使用，因而不需要考虑互斥访问的问题；多线程内存池有可能被多个线程共享，因此则需要在每次分配和释放内存时加锁。相对而言，单线程内存池性能更高，而多线程内存池适用范围更广。  
从内存池可分配内存单元大小来分，可以分为固定内存池和可变内存池。所谓固定内存池是指应用程序每次从内存池中分配出来的内存单元大小事先已经确定，是固定不变的；而可变内存池则每次分配的内存单元大小可以按需变化，应用范围更广，而性能比固定内存池要低。