**1、tcp和udp的区别**

TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，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，它只是把应用程序传给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

**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。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。

**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。

Linux --- 线程同步有最常用的是：互斥锁、条件变量和信号量。

**4、进程间通讯的方式有哪些，各有什么优缺点**

Linux 进程间通信（IPC）以下以几部分发展而来：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[https://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](https://www.cnblogs.com/jeakeven/p/5311575.html#16923962)](https://www.cnblogs.com/jeakeven/p/5311575.html#16923962)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)

管道:

优点是所有的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。

**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**

三次握手 --- 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。

四次握手 --- 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**

1）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。

2）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。

3）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。

**8、connect方法会阻塞，请问有什么方法可以避免其长时间阻塞？**

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。

或者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。

**9、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。

**10、在子网210.27.48.21/30种有多少个可用地址？分别是什么？**

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。

**11、TTL是什么？有什么用处，通常那些工具会用到它？（ping? traceroute? ifconfig? netstat?）**

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

**12、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？**

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。

**13、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。

**14、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。

**15、 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： socket -- bind -- listen--  accept--  recv/send-- close

Client端：socket------- 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：socket ----bind ----recvfrom/sendto----close

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

**16、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。

**17、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。

**18、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

**19、socket编程，如果client断电了，服务器如何快速知道？？？**

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）;

**20、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。Fork返回0为子进程，父进程返回编号。

**21、4G的long型整数中找到一个最大的，如何做？？？？**

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。

**22、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。

**23、tcp三次握手的过程，accept发生在三次握手哪个阶段？**

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。。。。。。

**24、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

**25、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。

**26、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（TYPE, MEMBER） size\_t（&((TYPE\*)0)->MEMBER）

**27、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

**28、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。

**29、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。

**30、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。

**31、 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。

**32、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)

33、初始化序列和构造函数谁先执行

34、timewait如何强制结束

35、receive返回值为0表示什么

36、gdb调试进程

37、显示当前进程信息的命令

1.具体说明STL如何实现vector

vector的内部是使用动态数组的方式来实现的，如果动态数组的内部实现不够用，就要动态的重新分配内存。然后把原数组的内容拷贝过去。

2.vector和list的区别

vector和数组类似，拥有连续的内存空间，支持随机的存取，在中间进行元素的插入和删除的操作时间复杂度是O(n)

list是由双向链表实现的，只能通过数组指针来进行数据访问，遍历中间的元素，时间的复杂度是O(n).

3.如何选择使用vector或dequeue

一般情况下使用vector,在需要从首尾两端进行插入或删除操作的时候需要选择dequeue.

map内部的实现

map的底层是一棵红黑树，在对节点的插入或是删除操作中，通过旋转来保持平衡性，最坏情况下的插入、删除、查找时间是O(logn)

map和hashmap的区别

底层数据结构不同，map是红黑树，hashmap是哈希表

map元素可以自动按照键值排序，hashmap的各项操作平均时间复杂度接近常数

map是C++标准的一部而hashmap并不是

vector中erase方法与algorithn中的remove方法区别

vector中erase方法真正删除了元素，迭代器不能访问了

remove只是简单地将元素移到了容器的最后面，迭代器还是可以访问到。因为algorithm通过迭代器进行操作，不知道容器的内部结构，所以无法进行真正的删除。

什么是auto\_ptr以及如何使用

auto\_ptr可以代替指针进行类似指针的操作，并不用关心内存的释放，auto\_ptr的析构函数自动释放它绑定的动态分配对象。

auto\_ptr的实现

每次创建类的新对象的时候，初始化指针并将引用计数设置为1；

当对象作为另一个对象的副本而创建的时候，拷贝构造函数，拷贝指针并增加与之相对应的引用计数

对一个对象进行赋值的时候，赋值操作符减少左操作数所指对象的引用计数，并增加右操作数所指对象的引用计数

调用析构函数的时候减少引用计数，如果引用计数减少至0，则删除基础对象

重载\*操作符和->操作符，使auto\_ptr有类似于普通指针的操作

使用auto\_ptr的限制

auto\_ptr所所指向的对象要求只能拥有一个拥有者

如下用法是错误的

classA \*pA = new classA;

auto\_ptr<classA> ptr1(pA);

auto\_ptr<classB> ptr2(pA);

auto\_ptr不能以传值的方式进行传递b

不支持数组

1.C++ STL 之所以得到广泛的赞誉，也被很多人使用，不只是提供了像vector, string, list等方便的容器，更重要的是STL封装了许多复杂的数据结构算法和大量常用数据结构操作。vector封装数组，list封装了链表，map和set封装了二叉树等。

2.标准关联容器set, multiset, map, multimap内部采用的就是一种非常高效的平衡检索二叉树：红黑树。RB树的统计性能要好于一般的[平衡二叉树](http://www.nist.gov/dads/HTML/avltree.html" \t "_blank)。

3.STL map和set的使用虽不复杂，但也有一些不易理解的地方，如：

（1）为何map和set的插入删除效率比用其他序列容器高？

答：因为对于关联容器来说，不需要做内存拷贝和内存移动。说对了，确实如此。map和set容器内所有元素都是以节点的方式来存储，其节点结构和链表差不多，指向父节点和子节点

（2）为何每次insert之后，以前保存的iterator不会失效？

答：iterator这里就相当于指向节点的指针，内存没有变，指向内存的指针怎么会失效呢(当然被删除的那个元素本身已经失效了)。

  相对于vector来说，每一次删除和插入，指针都有可能失效，调用push\_back在尾部插入也是如此。因为为了保证内部数据的连续存放，iterator指向的那块内存在删除和插入过程中可能已经被其他内存覆盖或者内存已经被释放了。即使时push\_back的时候，容器内部空间可能不够，需要一块新的更大的内存，只有把以前的内存释放，申请新的更大的内存，复制已有的数据元素到新的内存，最后把需要插入的元素放到最后，那么以前的内存指针自然就不可用了。特别时在和find等算法在一起使用的时候，牢记这个原则：不要使用过期的iterator。

（3）为何map和set不能像vector一样有个reserve函数来预分配数据？

答：我以前也这么问，究其原理来说时，引起它的原因在于在map和set内部存储的已经不是元素本身了，而是包含元素的节点。也就是说map内部使用的Alloc并不是map<Key,Data, Compare, Alloc>声明的时候从参数中传入的Alloc。

4.set, multiset  
set和multiset会根据特定的排序准则自动将元素排序，set中元素不允许重复，multiset可以重复。

因为是排序的，所以set中的元素不能被修改，只能删除后再添加。  
向set中添加的元素类型必须重载<操作符用来排序。排序满足以下准则：  
1、非对称，若A<B为真，则B<A为假。  
2、可传递，若A<B,B<C，则A<C。  
3、A<A永远为假。

set中判断元素是否相等：if(!(A<B || B<A))，当A<B和B<A都为假时，它们相等。

5.map，multimap  
map和multimap将key和value组成的pair作为元素，根据key的排序准则自动将元素排序，map中元素的key不允许重复，multimap可以重复。

因为是排序的，所以map中元素的key不能被修改，只能删除后再添加。key对应的value可以修改。

6. List的功能方法   
实际上有两种List: 一种是基本的ArrayList,其优点在于随机访问元素，另一种是更强大的LinkedList,它并不是为快速随机访问设计的，而是具有一套更通用的方法。   
List : 次序是List最重要的特点：它保证维护元素特定的顺序。List为Collection添加了许多方法，使得能够向List中间插入与移除元素(这只推荐LinkedList使用。)一个List可以生成ListIterator,使用它可以从两个方向遍历List,也可以从List中间插入和移除元素。

ArrayList : 由数组实现的List。允许对元素进行快速随机访问，但是向List中间插入与移除元素的速度很慢。ListIterator只应该用来由后向前遍历ArrayList,而不是用来插入和移除元素。因为那比LinkedList开销要大很多。   
LinkedList : 对顺序访问进行了优化，向List中间插入与删除的开销并不大。随机访问则相对较慢。(使用ArrayList代替。)还具有下列方法：addFirst(), addLast(), getFirst(),getLast(), removeFirst() 和 removeLast(), 这些方法 (没有在任何接口或基类中定义过)使得LinkedList可以当作堆栈、队列和双向队列使用。

7.1 hash\_map和map的区别在哪里？

构造函数。hash\_map需要hash函数，等于函数；map只需要比较函数(小于函数).

存储结构。hash\_map采用hash表存储，map一般采用[红黑树(RB Tree)](http://stl.winterxy.com/html/000039.html" \t "_blank)实现。因此其memory数据结构是不一样的。

7.2 什么时候需要用hash\_map，什么时候需要用map?

总体来说，hash\_map 查找速度会比map快，而且查找速度基本和数据数据量大小，属于常数级别;而map的查找速度是log(n)级别。

并不一定常数就比log(n)小，hash还有hash函数的耗时，明白了吧，如果你考虑效率，特别是在元素达到一定数量级时，考虑考虑hash\_map。但若你对内存使用特别严格，希望程序尽可能少消耗内存，那么一定要小心，hash\_map可能会让你陷入尴尬，特别是当你的hash\_map对象特别多时，你就更无法控制了。而且hash\_map的构造速度较慢。

现在知道如何选择了吗？权衡三个因素:   查找速度,   数据量,   内存使用   。

8.一些使用上的建议：  
Level 1 - 仅仅作为Map使用：采用静态数组  
Level 2 - 保存定长数据，使用时也是全部遍历：采用动态数组（长度一开始就固定的话静态数组也行）  
Level 3 - 保存不定长数组，需要动态增加的能力，侧重于寻找数据的速度：采用vector  
Level 3 - 保存不定长数组，需要动态增加的能力，侧重于增加删除数据的速度：采用list  
Level 4 - 对数据有复杂操作，即需要前后增删数据的能力，又要良好的数据访问速度：采用deque  
Level 5 - 对数据中间的增删操作比较多：采用list，建议在排序的基础上，批量进行增删可以对运行效率提供最大的保证  
Level 6 - 上述中找不到适合的：组合STL容器或者自己建立特殊的数据结构来实现

9.(1).vector - 会自动增长的数组

vector<int>vec(10) //一个有10个int元素的容器  
push\_back和pop\_back。前端和中间对数据进行操作必然会引起数据块的移动，这对性能影响是非常大的。最大优势就是随机访问的能力。

vector<T1>::iterator相关的方法有：  
begin()：用来获得一个指向vector第一个元素的指针  
end()：用来获得一个指向vector最后一个元素之后的那个位置的指针，注意不是指向最后一个元素。  
erase(vector<T1>::iterator)：用来删除作为参数所传入的那个iterator所指向的那个元素。

(2).list - 擅长插入删除的链表

list<string>Milkshakes;list<int> Scores;

push\_back, pop\_back，push\_front. pop\_front

list是一个双向链表的实现。  
为了提供双向遍历的能力，list要比一般的数据单元多出两个指向前后的指针

一个使用iterator来删除元素的例子  
list<string> stringList;  
list<string>::iterator iter;  
advance(iter, 5); //将iterator指向stringList的第五个元素  
stringList.erase(iterator);//删除  
那么删除操作进行以后，传入erase()方法的iterator指向哪里了呢？它指向了删除操作前所指向的那个元素的后一个元素。

(3).deque - 拥有vector和list两者优点的双端队列

(4)．这三个模板的总结 比较和一般使用准则  
这三个模板都属于序列类模板，可以看到他们有一些通用方法  
size():得到容器大小  
begin():得到指向容器内第一个元素的指针（这个指针的类型依容器的不同而不同）  
end():得到指向容器内最后一个元素之后一个位置的指针  
erase():删除传入指针指向的那个元素  
clear():清除所有的元素  
＝＝运算符：判断两个容器是否相等  
＝运算符：用来给容器赋值  
上面的三个模板有各自的特点  
     vector模板的数据在内存中连续的排列，所以随机存取元素（即通过[]运算符存取）的速度最快，这一点和数组是一致的。同样由于它的连续排列，所以它在除尾部以外的位置删除或添加元素的速度很慢，在使用vector时，要避免这种操作。  
     list模板的数据是链式存储，所以不能随机存取元素。它的优势在于任意位置添加 删除元素的速度。  
     deque模板是通过链接若干片连续的数据实现的，所以均衡了以上两个容器的特点

|  |
| --- |
| STL容器的实现原理 |

STL来管理数据十分方便,省去了我们自己构建数据结构的时间.其实,STL的实现也是基于我们常见的数据结构.

序列式容器：

vector-数组，元素不够时再重新分配内存，拷贝原来数组的元素到新分配的数组中。

list－单链表。

deque-分配中央控制器map(并非map容器)，map记录着一系列的固定长度的数组的地址.记住这个map仅仅保存的是数组的地址,真正的数据在数组中存放着.deque先从map中央的位置(因为双向队列，前后都可以插入元素)找到一个数组地址，向该数组中放入数据，数组不够时继续在map中找空闲的数组来存数据。当map也不够时重新分配内存当作新的map,把原来map中的内容copy的新map中。所以使用deque的复杂度要大于vector，尽量使用vector。

stack-基于deque。

queue-基于deque。

heap-完全二叉树，使用最大堆排序，以数组(vector)的形式存放。

priority\_queue-基于heap。

slist-双向链表。

关联式容器：

set,map,multiset,multimap-基于红黑树(RB-tree)，一种加上了额外平衡条件的二叉搜索树。

hash table-散列表。将待存数据的key经过映射函数变成一个数组(一般是vector)的索引，例如：数据的key%数组的大小＝数组的索引(一般文本通过算法也可以转换为数字)，然后将数据当作此索引的数组元素。有些数据的key经过算法的转换可能是同一个数组的索引值(碰撞问题，可以用线性探测，二次探测来解决)，STL是用开链的方法来解决的，每一个数组的元素维护一个list，他把相同索引值的数据存入一个list，这样当list比较短时执行删除，插入，搜索等算法比较快。

hash\_map,hash\_set,hash\_multiset,hash\_multimap-基于hash table。

1. fork和for循环相遇时，需要考虑printf缓冲区的问题
2. 两个单链表求公共子节点：不可能是X型只可能是Y型

**算法与数据结构** **（手写代码实现）**   
1 大整数加、减、乘、除、求模运算实现  
2 很多整数，找其中出现次数最多的那个数  
3 单链表翻转（两个指针如何实现）、查找、删除、插入以及双向链表、有序链表合并  
4 判断一个整数是否是2的整数次幂.(n&(n-1))  
5 二分查找（注意边界条件）  
6 常见排序算法的实现以及稳定性（快排跟归并考的很多）  
7 字符串翻转（O(n)）、匹配（KMP算法）  
8 最长递增子序列（nlogn的算法）  
9 链表判断是否有环，环的入口，两个链表是否相交（快慢指针）。  
10 指定一个数组，求2个数的和等于指定的和（某一个数），如果是3,4,5，n个等于个的和（某一个数）呢？（可以看作背包问题）

11 跳台阶问题  
  
**其他**   
1 红黑树的性质以及插入和删除  
2 解析XML文件  
3 千万级的用户，提供一个服务，该服务有很多模块，现在有一个底层模块需要优化，问怎么实现，在不影响其他服务模块以及用户体验的情况下。（面IEG）  
4 卡特兰数以及公式推导（应多很多）  
5 未知大小的文件，翻转整个文件  
6 如果内存中有个cache存储qq号和最近登录时间问怎么样做hit和淘汰  
7 检测短信敏感词  
8 大数据问题  
9 C++、java和PHP有什么本质区别