开始自我介绍：

一般根据项目提问

---------------封装、多态、继承---------------------

封装：所谓封装就是将某些东西包装盒隐藏起来，让外界无法直接使用，只能通过某些特定的方式才能访问。封装的目的是增强**安全性**和**简化编程**，使用者**不必了解具体的实现细节**，而只是通过外部接口以及特定的访问权限来使用类的成员。

继承：借助继承，可以**扩展**原有的代码，应用到其他程序中，而**不必重新编写**这些代码。

多态:可以使我们以**相同的方式**处理**不同类型**的对象，我们可以使用一段代码处理不同类型的对象，**只要他们继承/实现了相同的类型**。这样我们没有必要为每一种类型的对象编写相同的逻辑，极大的提高了代码的重用。

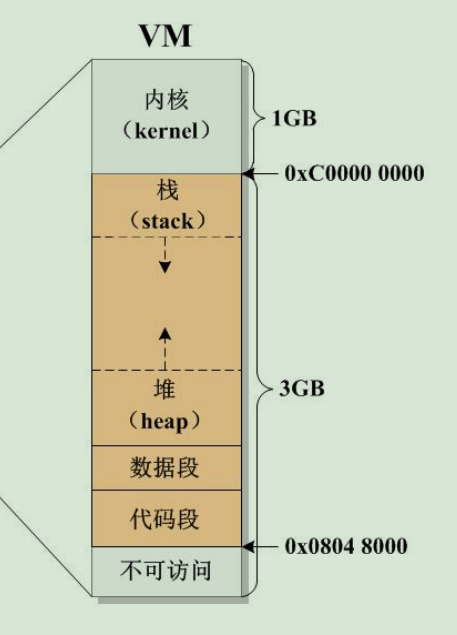
面向对象多态的实现：

1.c++的多态性就是通过动态绑定技术来实现的。

2.在基类的函数前加上virtual关键字，在派生类中重写该函数，运行时将会根据对象的实际类型来调用相应的函数，如果对象类型是派生类，就调用派生类的函数，如果对象类型是基类，就调用基类的函数。

3.虚函数，抽象类，覆盖，模板

**---------C/C++内存分配---------**



栈（Stack）：

局部变量，函数参数等存储在该区，由编译器自动分配和释放.栈属于计算机系统的数据结构，进栈出栈有相应的计算机指令支持，而且分配专门的寄存器存储栈的地址，效率分高，内存空间是连续的，但栈的内存空间有限。

堆(Heap)：

需要程序员手动分配和释放（new,delete），属于动态分配方式。内存空间几乎没有限制，内存空间不连续，因此会产生内存碎片。操作系统有一个记录空间内存的链表，当收到内存申请时遍历链表，找到第一个空间大于申请空间的堆节点，将该节点分配给程序，并将该节点从链表中删除。一般，系统会在该内存空间的首地址处记录本次分配的内存大小，用于delete释放该内存空间。

全局/静态存储区:

全局变量，静态变量分配到该区，到程序结束时自动释放，包括DATA段（全局初始化区）与BBS段（全局未初始化段）。其中，初始化的全局变量和静态变量存放在DATA段，未初始化的全局变量和静态变量存放在BBS段。BBS段特点：在程序执行前BBS段自动清零，所以未初始化的全局变量和静态变量在程序执行前已经成为0.

文字常量区：存放常量，而且不允许修改。程序结束后由系统释放。

程序代码区：存放程序的二进制代码

**---------内存管理---------**

堆和栈的区别：

1）空间大小：栈的内存空间是连续的，空间大小通常是系统预先规定好的，即栈顶地址和最大空间是确定的；而堆得内存空间是不连续的，由一个记录空间的链表负责管理，因此内存空间几乎没有限制，在32位系统下，内存空间大小可达到4G

2）管理方式：栈由编译器自动分配和释放，而堆需要程序员来手动分配和释放，若忘记delete，容易产生内存泄漏。

3）增长方向不同：对于栈：向着内存地址减小的方向生长的，这也是为什么栈的内存空间是有限的；而堆是向着内存地址增大的方向生长的

4）碎片问题：由于栈的内存空间是连续的，先进后出的方式保证不会产生零碎的空间；而堆分配方式是每次在空闲链表中遍历到第一个大于申请空间的节点，每次分配的空间大小一般不会正好等于申请的内存大小，频繁的new操作势必会产生大量的空间碎片

5）分配效率：栈属于机器系统提供的数据结构，计算机会在底层对栈提供支持，出栈进栈由专门的指令执行，因此效率较高。而堆是c/c++函数库提供的，当申请空间时需要按照一定的算法搜索足够大小的内存空间，当没有足够的空间时，还需要额外的处理，因此效率较低。

--------------------------------------------------------------------

基类的析构函数不是虚函数，会带来什么问题？

基类的的析构函数不是虚函数的话，删除指针时，只有其类的内存被释放，派生类的没有。这样就内存泄漏了。

--------------------------------------------------------------------

**什么是引用**

1)引用仅是变量的别名，而不是实实在在地定义了一个变量，因此引用本身并不占用内存，而是和目标变量共同指向目标变量的内存地址.

2）表达式中的取地址符&不再是取变量的地址，而是用来表示该变量是引用类型的变量。

3) 定义一个引用时，必须对其初始化。

4)用引用返回一个函数值的最大好处是，在内存中不产生被返回值的副本。

--------------------------------------------------------------------

**指针和引用的区别**

(1)指针：指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已。

(3)**可以有const指针，但是没有const引用；即指针存在指针常量的int \* const p和常量指针const int \*p和int const \*p，而引用只存在常量引用int const &a，不存在引用常量int& const a。**

(4)指针可以有多级，但是引用只能是一级（int \*\*p；合法 而 int &&a是不合法的）

(5)指针的值随时可以改变，引用不能改变，且定义的时候必须初始化。

(7)”sizeof引用”得到的是所指向的变量(对象)的大小，而”sizeof指针”得到的是指针本身的大小；

(8)指针和引用的自增(++)运算意义不一样；

(9)如果返回动态内存分配的对象或者内存，必须使用指针，引用可能引起内存泄漏；

--------------------------------------------------------------------

**数组和指针的区别**

(1)数组就是存放 一定数量变量的 地址连续的 内存空间

(2)在作为函数传递参数的时候，数组是等同于指针的

(3)在使用下标的时候，二者的用法相同。都是原地址加上下标的值，不过数组的原地址就是数组首元素的地址是固定的，指针的原地址就不是固定的，指针指向那个元素，该元素的地址就是要加上下标之前的原地址，因为这样，这样指针的灵活性很大。

--------------------------------------------------------------------

**C++容器**

c++中有两种类型的容器：**顺序容器**和**关联容器**，

顺序容器主要有：vector、list、deque等。其中vector表示一段连续的内存地址，基于数组的实现，list表示非连续的内存，基于链表实现。deque与vector类似，但是对于首元素提供删除和插入的双向支持。

关联容器主要有map和set。map是key-value形式的，set是单值。map和set只能存放唯一的key值，multimap和multiset可以存放多个相同的key值。（红黑树）

容器类自动申请和释放内存，我们无需new和delete操作。

--------------------------------------------------------------------

**struct 和 class的区别**

默认的继承访问权限struct是public的，class是private的。

struct里的变量叫数据，class内的变量叫成员，虽然它们并无区别。

--------------------------------------------------------------------

**c++设计模式**

（01）——工厂模式:https://blog.csdn.net/wuzhekai1985/article/details/6660462

1.简单工厂:

1)主要特点:需要在工厂类中做判断，从而创造相应的产品。当增加新的产品时

，就需要修改工厂类。

2.工厂方法:

1)定义一个用于创建对象的接口，让子类决定实例化哪一个类。Factory Method使一个类的实例化延迟到其子类。

2.抽象工厂:

1)提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

（02）——策略模式:https://blog.csdn.net/wuzhekai1985/article/details/6665197

1.定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。

2.本模式使得算法可独立于使用它的客户而变化。也就是说这些算法所完成的功能一样，对外的接口一样，只是各自实现上存在差异。

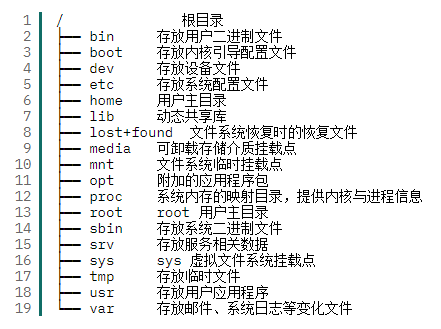
3.用策略模式来封装算法，效果比较好。

（03）——适配器模式:https://blog.csdn.net/wuzhekai1985/article/details/6665542

（04）——单例模式:https://blog.csdn.net/wuzhekai1985/article/details/6665869

单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例单例模式(不能让一个程序打开两次 如：不能同时打开2个迅雷 迅雷用的单例模式)

（05）——原型模式、模板方法模式（06）——建造者模式（07）——外观模式、组合模式（08）——代理模式（09）——享元模式（10）——桥接模式（11）——装饰模式（12）——备忘录模式（13）——中介者模式（14）——职责链模式（15）——观察者模式（16）——状态模式



**线程进程：**https://www.cnblogs.com/Yogurshine/p/3640206.html

1.进程:

进程是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。

进程是一个具有独立功能的程序关于某个数据集合的一次运行活动。它可以申请和拥有系统资源，

是一个动态的概念，是一个活动的实体。

2.线程:

轻量进程

线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。

3多线程:https://www.cnblogs.com/qtiger/p/5812483.html

在单个程序中同时运行多个线程完成不同的工作，称为多线程。

为了解决负载均衡问题,充分利用CPU资源.为了提高CPU的使用率,采用多线程的方式去同时完成几件事情而不互相干扰.为了处理大量的IO操作时或处理的情况需要花费大量的时间等等,比如:读写文件,视频图像的采集,处理,显示,保存等。

优点：

1.多线程技术使程序的响应速度更快 ,因为用户界面可以在进行其它工作的同时一直处于活动状态;

2.当前没有进行处理的任务时可以将处理器时间让给其它任务;

3.占用大量处理时间的任务可以定期将处理器时间让给其它任务;

4.可以随时停止任务;

5.可以分别设置各个任务的优先级以优化性；

6.程序的运行效率可能会提高

缺点：

1.如果有大量的线程,会影响性能,因为操作系统需要在它们之间切换.

2.更多的线程需要更多的内存空间

3.线程中止需要考虑对程序运行的影响.

4.通常块模型数据是在多个线程间共享的,需要防止线程死锁情况的发生

--------------------------------------------------------------------

**进程线程的通信**

进程的通信:

1.进程间通信主要包括管道, 系统IPC（包括消息队列,信号量,共享内存), 套接字(SOCKET).

管道:

1)普通管道PIPE, 通常有两种限制,一是单工,只能单向传输;二是只能在父子或者兄弟进程间使用.

2)流管道s\_pipe: 去除了第一种限制,为半双工，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.

信号:

（Signals ）是Unix系统中使用的最古老的进程间通信的方法之一。操作系统通过信号来通知进程系统中发生了某种预先规定好的事件（一组事件中的一个），它也是用户进程之间通信和同步的一种原始机制。

系统V IPC:

1)共享内存:

不需要内核这个“代理人”，一般需要配合互斥锁和信号量来使用。

共享内存是所有进程间通信方式中效率最高(原因是采用指针操作，直接访问地址)

2)消息队列:（多条并存的管道）

消息队列就是消息的一个链表，它允许一个或多个进程向它写消息，一个或多个进程从中读消息。Linux维护了一个消息队列向量表：msgque，来表示系统中所有的消息队列。

消息是按照队列一样排队的，先进先出FIFO当你接收的消息类型没有的时候，会阻塞当前进程

3)信号量（旗语）:

通过对进程拥有信号量资源的pv操作去控制多个进程对于临界区资源的访问保证(通过阻塞其他进程)两个或多个关键代码段(临界区资源)不被并发调用

--------------------------------------------------------------------

**等待队列**

设置一个等待的条件，条件满足，进程就继续向下执行；条件不满足，进程就**阻塞**在等待队列上。当条件满足后，中断会唤醒等待队列中的进程，进程再继续向下执行。

--------------------------------------------------------------------

**linux中断服务程序是一个原子过程**

原子过程，这个过程是一个连续的过程，该过程不能产生阻塞，不能被打断。

--------------------------------------------------------------------

**ARM中断处理**

1)当中断产生后，除了复位中断立即中止当前指令外，其余情况都是处理器完成当前指令后，才去执行异常处理程序。

2)CPSR（程序状态寄存器）--> SPSR（程序状态保存寄存器）

3)设置当前状态寄存器CPSR的相应位。设置CPSR中的M4~M0的5位，进入相应工作模式，设置I=1禁止IRQ中断，如果进入复位模式或FIQ模式，还要设置F=1以禁止FIQ中断。

4)将引起异常指令的下一条地址（断点地址）保存到新异常工作模式的LR（R14）中，使异常处理程序执行完后正确返回原来程序处继续向下执行。

5)首先恢复原来被保护的用户寄存器。

6)将SPSR寄存器复制到CPSR中，使得原来CPSR状态从相应的SOSR中恢复，一恢复被中断的程序状态。

7)根据异常类型将PC值恢复成断点地址，以继续执行用户原来运行着的程序。

注意：

（1）程序状态寄存器及断点地址的恢复必须同时进行。

（2）由于异常随机发生，所以要对异常向量进行初始化，即在异常向量的地址处放置一条跳转指令，跳转到异常处理程序

ARM微处理器共有37个32位寄存器，其中31个为通用寄存器，6个为状态寄存器。

ARM Cortex共有40个寄存器，32个通用寄存器，7个状态寄存器，1个程序计数器PC

ARM9采用哈佛体系结构，指令和数据分属不同的总线，可以并行处理。在流水线上，ARM7是三级流水线，ARM9是五级流水线。

ARM处理器7种工作模式：

用户模式（USR）：正常程序执行模式，不能直接切换到其他模式

系统模式（SYS）：运行操作系统的特权任务，与用户模式类似，但具有可以直接切换到其他模式等特权

快中断模式（FIQ）：支持高速数据传输及通道处理，FIQ异常响应时进入此模式

中断模式（IRQ）：用于通用中断处理，IRQ异常响应时进入此模式

管理模式（SVC）：操作系统保护模式，系统复位和软件中断响应时进入此模式（由系统调用执行软中断SWI命令触发）

中止模式（ABT）：用于支持虚拟内存和/或存储器保护，在ARM7TDMI没有大用处

未定义模式（UND）：支持硬件协处理器的软件仿真，未定义指令异常响应时进入此模式

除用户模式外，其余6种工作模式都属于特权模式

特权模式中除了系统模式以外的其余5种模式称为异常模式（特权异常模式）

大多数程序运行于用户模式

进入特权模式是为了处理中断、异常、或者访问被保护的系统资源

硬件权限级别：系统模式 > 异常模式 > 用户模式

--------------------------------------------------------------------

**Linux有什么看法？**

(1)基本思想：第一，一切都是文件；第二，每个软件都有确定的用途。

(2)完全免费,吸收了无数程序员的精华，不断壮大。

(3)完全兼容POSIX1.0标准

(4)多用户、多任务

(5)具有字符界面和图形界面。

(6)支持多种平台

(7)LFS

(8)

(9)

--------------------------------------------------------------------

**字符串拷贝函数**

\_strcpy(char \* dest, const char \* src)

{

char \*p=dest;

while(\*dest++ = \*src++);

dest=p;

}

--------------------------------------------------------------------

**快速排序或者堆排序**

void qSort(int \*a,int p,int q){

if(p >= q) return;

int i = p;

for(int j = p; j < q ; j++)

{

if(a[j] < a[q])

{

if(i != j)

swap(a[i],a[j]);

i++;

}

}

swap(a[i],a[q]);

qSort(a,p,i-1);

qSort(a,i+1,q);

}

--------------------------------------------------------------------

**你认为嵌入式软件工程师应该做什么？流程？**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。它一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户的应用程序等四个部分组成，用于实现对其他设备的控制、监视或管理等功能。

嵌入式软件工程师主要从事嵌入式软件开发工作。涉及应用层以及底层软件开发和设计工作 。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--------------------------------------------------------------------

**嵌入式系统的特点?**

(1)与应用紧密结合的，它具有很强的专用性

(2)系统内核小,系统精简。

(3)高实时性的系统软件

(4)具有较长的生命周期。

(5)程序固化

嵌入式系统通常执行的是带有特定要求的预先定义的任务。

国内普遍认同的嵌入式系统定义为：以应用为中心，以计算机技术为基础，软硬件可裁剪，适应应用系统对功能、可靠性、成本、体积、功耗等严格要求的专用计算机系统。

--------------------------------------------------------------------

**五年内有什么规划？**

1、成为一名顶尖的嵌入式工程师，实现自己的中国梦。

2、了解IT行业，互联网行业是如何运作的。比如，一个APP的背后隐含了多少的东西。

3、人不可能一直工作：做一些没做过的事情。

--------------------------------------------------------------------

14.最后怎么解决项目遇到的问题。

大体框架：

模块化：

内存泄漏；

打印调试；

--------------------------------------------------------------------

网络层：路由器、防火墙

数据链路层：网卡、网桥、交换机

物理层：中继器、集线器

**详细解释TCP(第四层传输层)**

应用层向TCP层发送用于网间传输的、用8位字节表示的数据流，然后TCP把数据流分区成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传输单元（MTU）的限制）。之后TCP把结果包传给IP层，由它来通过网络将12:13 2018/10/12包传送给接收端实体的TCP层。

TCP为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；

如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。

在数据正确性与合法性上，TCP用一个校验和函数来检验数据是否有错误，在发送和接收时都要计算校验和；同时可以使用md5认证对数据进行加密。

在保证可靠性上，采用超时重传和捎带确认机制。

在流量控制上，采用滑动窗口协议，协议中规定，对于窗口内未经确认的分组需要重传。

连接建立（三次握手）

连接终止（四次握手）

--------------------------------------------------------------------

**有哪几种方法创建一个文件**

普通文件：

$ touch regular

目录：

$ mkdir directory

符号链接：

$ ln -s ./regular symbolic

$ mknod fifo p

# mknod character c 1 1

# mknod block b 1 2

socket文件：

$ mksock sock

如果没有mksock命令，使用nc命令进行创建：

$ nc -UI ./sock

--------------------------------------------------------------------

**内核空间和用户空间的区别，内核空间多大(1 GB)**

1)每个进程有各自的私有用户空间（0～3G），这个空间对系统中的其他进程是不可见的,而内核空间则为所有进程以及内核所共享。

2)Linux使用两级保护机制：0级供内核使用，3级供用户程序使用。

3)用户态切换到内核态的3种方式:

a. 系统调用

这是用户态进程主动要求切换到内核态的一种方式，用户态进程通过系统调用申请使用操作系统提供的服务程序完成工作，比如fork()实际上就是执行了一个创建新进程的系统调用。而系统调用的机制其核心还是使用了操作系统为用户特别开放的一个中断来实现，例如Linux的int 80h中断。

b. 异常

当CPU在执行运行在用户态下的程序时，发生了某些事先不可知的异常，这时会触发由当前运行进程切换到处理此异常的内核相关程序中，也就转到了内核态，比如 缺页异常。

c. 外围设备的中断

当外围设备完成用户请求的操作后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条即将要执行的指令转而去执行与中断信号对应的处理程序，如果先前执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了由用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

这3种方式是系统在运行时由用户态转到内核态的最主要方式，其中系统调用可以认为是用户进程主动发起的，异常和外围设备中断则是被动的。

--------------------------------------------------------------------

17.觉得自己有哪些方面没表现出来

18.有什么问题想问

1、嵌入式工程师很多时候是在编程（跟软件打交道）吗？

2、嵌入式工程师需要了解哪些硬件知识

3、顶尖的嵌入式工程师需要必备什么技能

--------------------------------------------------------------------

2、R15 、R14、R13寄存器是什么寄存器

R13：堆栈指针寄存器 SP

R14：称为子程序链接寄存器LR

R15：是程序计数寄存器PC，指向当前的程序地址。

--------------------------------------------------------------------

4、主设备号和次设备号的区别

主设备号用来区分不同种类的设备，而次设备号用来区分同一类型的多个设备。对于常用设备，Linux有约定俗成的编号，如硬盘的主设备号是3，混杂设备10。

--------------------------------------------------------------------

5、驱动设备有哪几类

大致分为三类，字符驱动，块设备驱动，网络设备驱动。

字符设备可以看成是用字节流存取的文件

块设备则可以看成是可以任意存取字节数的字符设备，在应用上只是内核管理数据方式不同

网络设备可以是一个硬件设备，或者是软件设备，他没有相应的read write，它是面向流的一种特殊设备。

--------------------------------------------------------------------

**平台驱动是什么**

platform\_driver

platform总线

Linux platform driver机制和传统的device driver 机制(通过driver\_register函数进行注册)相比，一个十分明显的优势在于platform机制将设备本身的资源注册进内核，由内核统一管理，在驱动程序中使用这些资源时通过platform device提供的标准接口进行申请并使用。这样提高了驱动和资源管理的独立性，并且拥有较好的可移植性和安全性(这些标准接口是安全的)。

--------------------------------------------------------------------

**probe函数作用**

probe函数在设备驱动注册最后收尾工作，当设备的device 和其对应的driver 在总线上完成配对之后，系统就调用platform设备的probe函数完成驱动注册最后工作。

--------------------------------------------------------------------

**字符串匹配的函数**

递归 遍历

--------------------------------------------------------------------

**端口的定义**

电脑运行的系统程序 就像一个闭合的圆圈，系统程序设计者把这个 圆圈截成 很多段，这些线段接口就叫端口（通俗讲就是断口，就是中断），系统运行到这些端口时，根据端口是否打开或关闭，如果关闭，就是绳子接通了，系统往下运行，如果端口是打开的，系统就得到命令，有外部数据输入，接受外部数据并执行。

端口在报头中占两个字节，也就是16位。端口号用来表示和区别网络中的不同应用程序。

（1）公认端口（Well Known Ports）：0-1023之间的端口号。这些端口由 IANA 分配管理。IANA 把这些端口分配给最重要的一些应用程序，让所有的用户都知道，当一种新的应用程序出现后，IANA必须为它指派一个公认端口。

常用的公认端口有：

FTP : 21

TELNET : 23

SMTP : 25

DNS : 53

TFTP : 69

HTTP : 80

SNMP : 161

（2）注册端口（Registered Ports）：从1024-49151。是公司和其他用户向互联网名称与数字地址分配机构（ICANN）登记的端口号，利用因特网的传输控制协议（TCP）和用户数据报协议（UDP）进行通信的应用软件需要使用这些端口。在大多数情况下，这些应用软件和普通程序一样可以被非特权用户打开。

（3）客户端使用的端口号：49152~65535.这类端口号仅在客户进程运行时才动态选择，因此又叫做短暂端口号。被保留给客户端进程选择暂时使用的。也可以理解为，客户端启动的时候操作系统随机分配一个端口用来和服务器通信，客户端进程关闭下次打开时，又重新分配一个新的端口。

--------------------------------------------------------------------

**\*[P + 1] 换成 \*[1 + P]是否可以**

--------------------------------------------------------------------

**UDP实现广播时会不会出现对方接受不到**

怎样划分子网 ：

一个B类网络的默认子网掩码为255.255.0.0，你如果想把它划分为2个子网，网络位数就成立17位（2^1)，也就是说子网掩码就变成了255.255.128.0；想划分为16个子网，因为16是2的4次方，所以网络位数加4变成了20位，也就是说子网掩码加长，成了20位，就是255.255.240.0。C类和A类网络子网换分的时候，子网掩码也是这么求的，所以还是很简单的。我们在做这种子网划分的问题的时候，只需要抓住两个重点，一个网络属于哪一类还有就是子网数，这种问题就迎刃而解了

--------------------------------------------------------------------

**gcc和g++区别**

误区一：gcc只能编译c代码，g++只能编译c++代码

两者都可以，但是请注意：

1.后缀为。c的，gcc把它当作是C程序，而g++当作是c++程序；后缀为。cpp的，两者都会认为是c++程序，注意，虽然c++是c的超集，但是两者对语法的要求是有区别的。C++的语法规则更加严谨一些。

2.编译阶段，g++会调用gcc，对于c++代码，两者是等价的，但是因为gcc命令不能自动和C＋＋程序使用的库联接，所以通常用g++来完成链接，为了统一起见，干脆编译/链接统统用g++了，这就给人一种错觉，好像cpp程序只能用g++似的。

误区二：gcc不会定义\_\_cplusplus宏，而g++会

实际上，这个宏只是标志着编译器将会把代码按C还是C++语法来解释，如上所述，如果后缀为。c，并且采用gcc编译器，则该宏就是未定义的，否则，就是已定义。

误区三：编译只能用gcc，链接只能用g++

严格来说，这句话不算错误，但是它混淆了概念，应该这样说：编译可以用gcc/g++，而链接可以用g++或者gcc -lstdc++.因为gcc命令不能自动和C＋＋程序使用的库联接，所以通常使用g++来完成联接。但在编译阶段，g++会自动调用gcc，二者等价。

--------------------------------------------------------------------

CVTE HR面：

1、对前面两轮的表现打个分，对自己打个分

2、说说家庭，和家里谁的关系最好

3、未来5-10年会做什么，怎么规划

4、有没有别的offer

5、如果给offer可能什么原因没来

6、什么原因可能会离职

7、对工作的地点、时间、薪资怎么看

--------------------------------------------------------------------

文件系统是对一个存储设备上的数据和元数据进行组织的机制。由于定义如此宽泛，支持它的代码会很有意思。正如前面提到的，有许多种文件系统和媒体。由于存在这么多类型，可以预料到 Linux 文件系统接口实现为分层的体系结构，从而将用户接口层、文件系统实现和操作存储设备的驱动程序分隔开。

--------------------------------------------------------------------

**volatile**

volatile是一个类型修饰符（type specifier）.volatile的作用是作为指令关键字，确保本条指令不会因编译器的优化而省略，且要求每次直接读值。

精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。

--------------------------------------------------------------------

**linux组成**

内核，shell，文件系统，应用程序

--------------------------------------------------------------------

**单向链表交叉问题**

--------------------------------------------------------------------