INF面试题汇总

**注意：**

**1、插入或删除题目时，请同时更新相应的试题编号;**

**2、修改完后，请更新目录;**

**目录**

[INF代码能力题目 4](#_Toc289939076)

[1 长方形个数计算(iis) 4](#_Toc289939077)

[2 IP字符串转换成为整数(iis) 5](#_Toc289939078)

[3 代码缺陷(iis) 5](#_Toc289939079)

[4 进制转换(sat) 6](#_Toc289939080)

[5 判断某些整数是否存在某个集合内(sat) 7](#_Toc289939081)

[6 lock-free实践(sat) 7](#_Toc289939082)

[7 vector/list/map区别和hash实现(dc) 8](#_Toc289939083)

[8 LRU算法与实现(dc) 9](#_Toc289939084)

[9 lock-free linked list(dc) 9](#_Toc289939085)

[10 判断梭哈大小(dc) 10](#_Toc289939086)

[11 手机英文输入法(dc) 10](#_Toc289939087)

[12 c++成员初始化列表(dc) 11](#_Toc289939088)

[13 C++方便锁操作(dc) 11](#_Toc289939089)

[14 字符串分析(ds) 12](#_Toc289939090)

[15 c++实现new[]/delete[](ds) 13](#_Toc289939091)

[16 C++中static的用法(ds) 14](#_Toc289939092)

[17 结构体对齐问题(ds) 14](#_Toc289939093)

[18 在线查找集合(hpc) 16](#_Toc289939094)

[19 memcpy和strcpy有什么区别？(hpc) 17](#_Toc289939095)

[20 代码缺陷(hpc) 17](#_Toc289939096)

[21 环形队列实现(hpc) 19](#_Toc289939097)

[22 处理器字节序(com) 19](#_Toc289939098)

[23 代码缺陷(com) 20](#_Toc289939099)

[24 C++虚函数安装(com) 21](#_Toc289939100)

[25 C++继承的引用和指针(com) 22](#_Toc289939101)

[26 将帅不见面(com) 23](#_Toc289939102)

[27 大无符号整数相加(com) 25](#_Toc289939103)

[28 判断处理器字节序(com) 27](#_Toc289939104)

[29 代码缺陷(com) 28](#_Toc289939105)

[30 strcpy实现(com) 29](#_Toc289939106)

[31 大数求解余数(com) 30](#_Toc289939107)

[32 C++拷贝构造为什么传递引用(com) 30](#_Toc289939108)

[33 代码缺陷(com) 31](#_Toc289939109)

[34 c类型转换问题(com) 32](#_Toc289939110)

[INF算法能力题目 33](#_Toc289939111)

[1 对字符串的分析 33](#_Toc289939112)

[2 完全二叉树和堆的定义？ 34](#_Toc289939113)

[3 最小全集串 34](#_Toc289939114)

[4 寻找缺少数 34](#_Toc289939115)

[5 随机数发生器 35](#_Toc289939116)

[6 二叉树共同祖先 35](#_Toc289939117)

[7 升序查找 36](#_Toc289939118)

[8 找出两个链表的第一个公共结点 37](#_Toc289939119)

[9 有序数组查找 38](#_Toc289939120)

[10 访问识别 38](#_Toc289939121)

[11 数列调整 39](#_Toc289939122)

[12 二分查找算法和变种 39](#_Toc289939123)

[13 max栈 40](#_Toc289939124)

[14 统计树的数组 40](#_Toc289939125)

[15 大数据量排序 41](#_Toc289939126)

[16 台阶走法 41](#_Toc289939127)

[17 线程安全hashmap 42](#_Toc289939128)

[18 链表复制 42](#_Toc289939129)

[19 消息传播 42](#_Toc289939130)

[20 Shuffle 43](#_Toc289939131)

[21 统计在线人数 44](#_Toc289939132)

[22 求第N个数 45](#_Toc289939133)

[23 下落高度 46](#_Toc289939134)

[24 实现命令 46](#_Toc289939135)

[INF操作系统部分考查题目 47](#_Toc289939136)

[1 逻辑地址与物理地址换算 47](#_Toc289939137)

[2 Tcp三次握手问题 47](#_Toc289939138)

[3 进程和线程的差别 48](#_Toc289939139)

[4 说明VFS的作用和Linux IO的简单层次结构 49](#_Toc289939140)

[5 实模式、保护模式 50](#_Toc289939141)

[6 说明静态库和动态库的区别和优缺点。 50](#_Toc289939142)

[7 操作系统内存问题 51](#_Toc289939143)

[8 非易失性内存之YY 52](#_Toc289939144)

[9 进程调度问题 53](#_Toc289939145)

[10 epoll&select 53](#_Toc289939146)

[11 iptables原理 54](#_Toc289939147)

[12 netstat结果解释 54](#_Toc289939148)

[INF网络部分考查题目 56](#_Toc289939149)

[1 DNS是什么 56](#_Toc289939150)

[2 简述DNS进行域名解析的过程 56](#_Toc289939151)

[3 FTP连接模式问题。 57](#_Toc289939152)

[4 端口扫描问题 57](#_Toc289939153)

[5 Traceroute问题 58](#_Toc289939154)

[6 网络嗅探 58](#_Toc289939155)

[7 请简要叙述ARP协议工作的具体过程 59](#_Toc289939156)

[8 未知协议分析问题 60](#_Toc289939157)

[9 NAT问题 61](#_Toc289939158)

[10 网络速度诊断 61](#_Toc289939159)

[11 静态路由与动态路由 62](#_Toc289939160)

[12 ARP欺骗 62](#_Toc289939161)

[13 IP可用地址计算问题 63](#_Toc289939162)

[14 子网计算问题 64](#_Toc289939163)

## INF代码能力题目

### 1 长方形个数计算(iis)

|  |
| --- |
| 问题内容 |
| 给定一个高度和宽度组成的网格, 返回该网格中总共的长方形(不包括正方形)的个数.  Eg: width=3, height=3. |
| 答案要点 |
| 对于该网格,里面包含 4个2\*3, 6个1\*3, 12个1\*2的长方形, 1\*1,2\*2,3\*3 是正方形,不计算在内. 返回 4+6+12=22 即可  class RectangularGrid  {  public:  long long countRectangles(int width,int height)  {  long long out=0;  for(into I=1;i<width’s++)  {  for(into j=1;j<height’s++)  {  out += I==j ? 0 : (width-i+1)\*(height-j+1);  }  }  return out;  }  }; |

### 

### 2 IP字符串转换成为整数(ibis)

|  |
| --- |
| 问题内容 |
| 实现函数 u\_long inet\_addr(const char \*cp) |
| 答案要点 |
| 描述：inet\_addr实现将字符串转换为代表ip的无符号long类型。 比如将"127.0.0.1"转换为16777343  实现算法：正常ip：a.b.c.d. 假设u\_long占4个byte，这样可以考虑让a，b，c，d分别放到u\_long的每个byte即可。 |

### 3 代码缺陷(ibis)

|  |
| --- |
| 问题内容 |
| void getmemory( char \*p )  {  　p = (char \*) malloc( 100 );  }  void test( void )  {  　char \*str = null;  　getmemory( str );  　strcpy( str, "hello world" );  　printf( str );  } |
| 答案要点 |
| 参数传递问题  判断分配是否成功  拷贝函数的风险 |

### 4 进制转换(sat)

|  |
| --- |
| 问题内容 |
| 输入一个十进制的数，将其转换成 N 进制数(0<n<=16) |
| 答案要点 |
| 考察点：此题比较基础，考察计算机基本数制知识和算法编程能力，编程语言不限，代码精简易懂为好  #include <stdio.h>  main()  {  into n,k,i=0;  into a[100];  printf("输入一个十进制的数:");  scanf("%d",&k);  printf("输入要转化的进制:");  scanf("%d",&n);  do{  a[++i] = k%n;  k = k/n;  }while(k!=0);  for(; I > 0; I--)  {  switch(a[I]){  case 15:printf("F");break;  case 14:printf("E");break;  case 13:printf("D");break;  case 12:printf("C");break;  case 11:printf("B");break;  case 10:printf("A");break;  default:printf("%d",a[I]);  }  }  printf("\n"); |

### 5 判断某些整数是否存在某个集合内(sat)

|  |
| --- |
| 问题内容 |
| 给40亿个不重复的unsigned into的整数，没排过序的，然后再给几个数，如何快速判断这几个数是否在那40亿个数当中? |
| 答案要点 |
| unsigned into 的取值范围是0到2^32-1。我们可以申请连续的2^32/8=512M的内存，用每一个bit对应一个unsigned into数字。首先将512M内存都初始化为0，然后每处理一个数字就将其对应的bit设置为1。当需要查询时，直接找到对应bit，看其值是0还是1即可。 |

### 6 lock-free实践(sat)

|  |
| --- |
| 问题内容 |
| 现代的处理器提供了compare-and-swap原子操作：  into compare\_and\_swap(into \* pv, const into cv, const into nv);  即比较\*pv与cv，如果相等，则把\*pv值替换为nv并返回\*pv原值，否则返回\*pv的值。  请利用上述原子操作实现如下操作：  into inc\_if\_gt\_zero(into \* pv);  即如果\*pv > 0，则把\*pv加1并返回修改后的\*pv，否则返回\*pv。  结果要线程安全且不使用锁、信号灯、互斥量、临界区或类似机制。 |
| 答案要点 |
| 这道题属于lock-free类型，一种实现方法是：  1. 获得\*pv的值cv； //获取\*pv的值cv这步， 要用volatile 修饰， 要不然在优化情况下很可能出问题。这个作为一个得分点.  2. 如果cv <=0，函数返回cv;  3. 否则(即cv > 0)，调用compare\_and\_swap(\*pv, cv, cv+1)，  如果返回值等于cv，则操作成功，函数返回cv；  4.否则(compare\_and\_swap()返回值不等于cv)，重新执行第1步； |

### 7 vector/list/map区别和hash实现(dc)

|  |
| --- |
| 问题内容 |
| 比较vector、list、map的区别（徐串） |
| 答案要点 |
| 考察对语言以及相关数据结构的掌握。  vector是动态数组，list是双向链表，map在C++中是红黑树。  进一步的考察点：  1、对于Java可以考察ArrayList和Vector、HashMap和Hashtable的区别。后者是前者的线程安全版本，操作是加锁的。  2、可以考察一下hash\_map相关知识。Hash表有哪些解决冲突的方式：开放地址法、二次hash法、链表法。可以紧接问题2。  3、vector的实现方式：预先分配固定大小的内存。然后每次增长到超过原有容量时，新申请一块内存并把数据复制过去。  hash实现  主要考察编程实现的技巧。可以单独问也可以紧接着问题1提出。 |

### 8 LRU算法与实现(dc)

|  |
| --- |
| 问题内容 |
| 可以先问一下cache相关替换算法有哪些：  然后考察一下如何实现一个快速的LRU算法。 |
| 答案要点 |
| 如常见的LRU、MRU等。一般是用hash\_map+单链表的方式。 |

### 9 lock-free linked list(dc)

|  |
| --- |
| 问题内容 |
| 对于一个单链表。在只有一个reader和一个writer的情况下如何做到不加锁。 |
| 答案要点 |
| 其主要思想是用一个空节点作为链表尾，利用指针读写操作的原子性来实现。每次写入的时候先把内容填入链表尾的空节点中，然后分配一个新的空节点添加到后面，最后把表尾指针指向新节点。读取的时候简单地返回链表头部内容即可。如果表头表尾指针一致就认为链表为空。 |

### 10 判断梭哈大小(dc)

|  |
| --- |
| 问题内容 |
| 考察编程实现能力。给两副梭哈牌，每副5张，比较其大小。梭哈规则是同花顺>四条>三带二>同花>顺子>三条>两对>1对>散牌。 |
| 答案要点 |
| 一张牌用2个字母表示。第一个字母表示点数：AKQJT98765432。第二个字母表示花色：SHCD，分别表示黑桃红桃梅花方片。  有很多种实现方式，可以导致程序极大的区别。在比较规则上可以根据面试者写程序的情况适当删减。例如可以去掉花色。  可以考察或者不考察有非法输入的情况。 |

### 11 手机英文输入法(dc)

|  |
| --- |
| 问题内容 |
| 给定一个英文词典，和一个数字字符串，找出最可能的英文单词。数字和英文字母的对应关系见手机键盘。可以要求完全匹配或者前缀匹配。 |
| 答案要点 |
| 完全匹配是单纯考察hash表。前缀匹配则演变成query suggestion。在单机上用trie解决。可以进一步考察大规模数据实现方案，这时候可以采用分布式hash表。 |

### 12 c++成员初始化列表(dc)

|  |
| --- |
| 问题内容 |
| c++成员初始化列表 |
| 答案要点 |
| 1. 可以更高效地初始化类成员； 2) 解决没有默认构造函数的类对象的生成； 3) 初始化类成员中的const对象或引用对象。 |

### 13 C++方便锁操作(dc)

|  |
| --- |
| 问题内容 |
| 在c++的方法中，经常会在方法进入前进行加锁操作，然后离开方法时要解锁，但是因为方法中经常有if/else语句，还有异常处理，造成书写时经常出现 bug，忘记解锁。在c++中，设计一种方法来解决这个问题。 |
| 答案要点 |
| 主要是利用c++的类的构造函数和析构函数来完成。  public LockObject {  private：  Lock& theLock;  public:  LockObject(Lock& lock): theLock(lock){  theLock.lock():  }    ~LockOject(){  theLock.unlock();  }    }  使用方式:  void myMethod() {  LockOject lo(lock);  doSomething;  ) |

### 14 字符串分析(ds)

|  |
| --- |
| 问题内容 |
| 对如下字符串  (234453)[234]{2324}  分析它的括号使用是否正确 |
| 答案要点 |
| 难度一层：  回答：使用栈，碰到左括号入栈，碰到右括号出栈，看最后栈是否空，是否还有未匹配完的右括号  建议：让面试者写一下代码，注意代码风格，异常检查是否完整  难度二层：  告诉面试者（和）匹配 [和]匹配  {和}匹配  回答：在一基础上要增加括号种类信息，匹配时要检查  难度三层：  告诉面试者要注意数学规则，括号有优先级  {[（）]} 是正确的  但是[{}] 不正确  回答：注意在二上要检查 压栈时的优先级检查  难度四层：  告诉面试者要注意数学规则，(()) 、 [[]] 、[] 这样的使用是错误的。但是{{[()]}} 是正确的  回答：压栈时的检查要更多，注意异常分支。 |

### 15 c++实现new[]/delete[](ds)

|  |
| --- |
| 问题内容 |
| 用c语言编写一个函数，实现c++中的new[]和delete[]操作符，写出代码30分钟，说出思路5分钟。 |
| 答案要点 |
| 答案：核心思想在于delete一个数组，不需要提供数组元素的个数。可以在malloc的时候，额外开辟一块空间保存元素个数，如：在返回的指针所指地址前一个into位置保存。delete时候，通过指针移动获取到该值，进行内存free。 |

### 16 C++中static的用法(ds)

|  |
| --- |
| 问题内容 |
| C++中static的用法 |
| 答案要点 |
| 答案：函数内、类成员变量和成员函数、文件内等 |

### 17 结构体对齐问题(ds)

|  |
| --- |
| 问题内容 |
| 32 位机器中：以下类的实例各占多少内存（sizeof结果）?  class A  {  };  class B  {  public:  void f();  void g();  };  class C  {  public:  void f();  void g();  private:  into a;  };  class D  {  public:  virtual f();  virtual g();  };  class E : public D  {  private:  into a;  }; |
| 答案要点 |
| sizeof(A) = 1, sizeof(B）=1，sizeof(C) = 4; sizeof(D) = 4; sizeof(E) = 8 |

### 18 在线查找集合(hpc)

|  |
| --- |
| 问题内容 |
| 现在有一组共计N个固定的集合，每个集合有个从0开始递增的集合ID，每个集合包含1-M个term，希望设计一个程序，能够持续对外服务，输入是一个term数组，输出其中任意一个集合ID（如果该term数组包含该集合的所有term），如果找不到，输出-1。要求：  时间复杂度最优，能够在短时间内对大量输入逐个输出。  实现具体的代码，其中常用的数据结构可以采用标准库  给出时间复杂度和空间复杂度 |
| 答案要点 |
| 对“一组共计N个固定的集合”进行预处理，采用合适的数据结构：可以建立term到集合ID之间的倒排索引；这部分得分2分；如果想不到合适的数据结构，只会用数组等不合适快速查找的数据结构，全题0分。这部分时间复杂度为O(N\*平均每个集合的term数量)  能够基于以上数据结构，给出具体的查找算法，其中一种较快的算法是，针对输入的term数组中的每个term，通过倒排查找到各集合ID，并建立辅助数据结构用于初始化每个集合的term数，然后计数，直到其中某个集合的term剩余数为0，说明找到一个集合。（需要对term进行去重处理）。时间复杂度为O(输入term数组的term数量\*平均每个term倒排对应的集合数量)以及初始化辅助数据结构的O(N)，所需空间O(N)。这部分得分为1~2分，如果查询过程采用倒排快速查找后，采用循环比较来确定是否有重合集合时，可以加分 |

### 19 memcpy和strcpy有什么区别？(hpc)

|  |
| --- |
| 问题内容 |
| memcpy和strcpy有什么区别？ |
| 答案要点 |
| 没有标准答案吧：  不过，总体来说，希望考察一下对基础知识是否了解得比较多。  1. 参数不同，结束条件不同  2. copy方式不同，效率不同，原因要讲清楚  3. 可以追问一下，效率一定不同么，（了解一下对cpu cache，是不是有直观的认识）  4. strlen->strcpy有变态的实现，是否知道（随便问问，不知道，也不能说明什么。）；这个算法有什么限制（对齐问题？） |

### 20 代码缺陷(hpc)

|  |
| --- |
| 问题内容 |
| 指出下面程序的问题所在，并给出原因。  1. #include <iostream>  2. using namespace std;  3.  4. into main()  5. {  6. char \*str = "this is a string";  7. str[0] = 'A';  8. cout << str << endl;  9.  10. char \*p = str;  11. p[0] = 'B';  12.  13. char arr[] = "this is a string, too";  14. arr++;  15. cout << arr << endl;  16.  17. char \*q = arr;  18. q++;  19. cout << q << endl;  20.  21. return 0;  22. } |
| 答案要点 |
| 以上有3个错误：  1. 第7行“str[0] = 'A';”有错误。因为char \*str = "this is a string"; 这句相当于申明const char \*str, 所以str指向的内容是常量，不能被修改。  2. 第11行“p[0] = 'B';”有错误。因为p和str指向同一个字符串，而那个字符串是常量，所以也是不能修改的。  3. 第14行“arr++;”有错误。因为char arr[] = "this is a string, too"; 这句相当于申明char\* const arr, arr是一个常量，不能自增。  第1，2个错误每个1分，第3个错误2分。  能指出错误，并给出原因的得满分，只指出错误，未给原因或给的原因不对，适当减一些分数。 |

### 21 环形队列实现(hpc)

|  |
| --- |
| 问题内容 |
| 在某实时操作系统内核中，要实现一个环形队列，该队列能够存放200个以上元素就可以。 这个结构会被频繁用到，因此对时间要求很高，如何实现比较高效？要求：   1. 回答环形队列的概念   2. 实现该环形队列 |
| 答案要点 |
| 评分标准：  1. 理解环形队列，1分，否则可以提醒，但只能得后面部分分数  2. 能够实现环形队列的读、写操作，加2分  3. 最好的设计即环形队列长度取256，各取一个unsigned char a作为环形队列的读写指针，当读写指针为255，再递增时，由于溢出，变为0，自动实现环的功能。 加2分 |

### 22 处理器字节序(com)

|  |
| --- |
| 问题内容 |
| unsigned into data = 0x19490738;  unsigned char \*str = (unsigned char \*)&data;  fprintf(stderr, "%x\n", \*str); |
| 答案要点 |
| 38 |

### 23 代码缺陷(com)

|  |
| --- |
| 问题内容 |
| #include <stdio.h>  #include <stdarg.h>  #define LOG\_BUF\_SIZE 1024  void log\_debug(const char \*fmt, ...){  char str[LOG\_BUF\_SIZE];  va\_list args;  va\_start(args, fmt);  vsnprintf(str, LOG\_BUF\_SIZE, fmt, args);  va\_end(args);  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s", str);  }  into main(){  unsigned into data = 0x19490738;  log\_debug("result[1-250] = %s, %d\n", data, "test");  return 0;  } |
| 答案要点 |
| 最后两个参数写反了 |

### 24 C++虚函数安装(com)

|  |
| --- |
| 问题内容 |
| #include <cstdio>  class A{  public:  A(){  f();  }  virtual void f(){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("A::f\n");  }  };    class B:public A{  public:  virtual void f(){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("B::f\n");  }  };    into main(){  B b;  return 0;  }  请问打印什么结果?为什么? |
| 答案要点 |
| A::f. 虽然程序中只有构造B,但是构造B的时候,需要先构造A. 它的基类A中的构造函数调动了虚函数f(). 但是这个时候vtable还没有安装完成,所以调用的是A::f()函数而非B::f函数。 |

### 25 C++继承的引用和指针(com)

|  |
| --- |
| 问题内容 |
| class A{  into \_a;  public:  virtual void foo(){  }  };    class B:public A{  into \_b;  public:  virtual void foo(){    }  };    void func(A\* a,int num){  for(into I=0;i<num;i++){  a[I].foo();  }  }    into main(){  B \*b=new B[10];  func(b,10);  return 0;  }  请问上面代码存在什么问题? |
| 答案要点 |
| func的接口传入的是类型B的数组指针,内部是作为类型A的数组来处理. 而B和A的长度不相等,可能会存在隐患. |

### 

### 26 将帅不见面(com)

|  |
| --- |
| 问题内容 |
| 中国象棋有个规则,"将帅不得见面".现在棋盘上只有一个将,一个帅,并且帅这边还有一个仕, 请写一个程序,输出棋盘上所有可能存在的合理残局情况.注:需要当场写程序并且调试通过. |
| 答案要点 |
| 定义将和帅的九宫格编号如下：  0 1 2  3 4 5  6 7 8    into main(){  into I = 0;  into j = 0;  into k = 0;  for (I = 0; I < 9; I++){  for (j = 0; j < 9; j++){  for (k = 0; k <9; k += 2){  if (j == k){  continue;  }  if ((I % 3 != j % 3)){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("i=%d,j=%d,k=%d\n", i,j,k);  }  else if ((j % 3 == k % 3) && (k < j)){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("i=%d,j=%d,k=%d\n", i,j,k);  }  }  }  }  return 0;  } |

### 27 大无符号整数相加(com)

|  |
| --- |
| 问题内容 |
| BigUnsignedInteger[?](http://com.baidu.com/twiki/bin/edit/Main/BigUnsignedInteger?topicparent=Main.Interview1Problem)实现的是一种无符号大数表示,你需要实现一个无符号大数叠加操作. 阅读下面代码,完成TODO部分.    class BigUnsignedInteger{  public:  explicit BigUnsignedInteger():  \_data(){  \_data.push\_back(0);  }  explicit BigUnsignedInteger(const BigUnsignedInteger & v):  \_data(v.\_data){  }  explicit BigUnsignedInteger(unsigned long long v):  \_data(){  while(v){  \_data.push\_back(v%10);  v/=10;  }  }  *//TODO:*  *//有待面试者编写完成的部分.*  BigUnsignedInteger &  operator +=(const BigUnsignedInteger & v);  const std::string ToString() const{  std::string res;  for(ssize\_t I=\_data.size()-1;i>=0;i--){  res+=(\_data[I])+'0';  }  return res;  }  private:  std::vector<char> \_data;  }; |
| 答案要点 |
| BigUnsignedInteger &  operator +=(const BigUnsignedInteger & v);  const std::string ToString() const{  std::string res;  for(ssize\_t I=\_data.size()-1;i>=0;i--){  res+=(\_data[I])+'0';  }  return res;  } |

### 28 判断处理器字节序(com)

|  |
| --- |
| 问题内容 |
| 如何判断你的CPU是小端还是大端的? |
| 答案要点 |
| #include <cstdio>  into main(){  union X{  into a;  char b[4];  }x;  x.a=0x1;  if(x.b[0]==0x1){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("little endian\n");  }else if(x.b[4]==0x1){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("big endian\n");  }else{  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("which cpu do you have?\n");  }  return 0;  } |

### 29 代码缺陷(com)

|  |
| --- |
| 问题内容 |
| 以下函数做了一个什么操作?存在哪些问题?  into calc(uint num){  uint sum;  for (; num >= 0; num--){  sum = sum + num\*num;  }  return sum;  } |
| 答案要点 |
| 该函数传入x,计算x^2+(x-1)^2+(x-2)^2...+2^2+1^2.   * 传入num是uint,永远大于等于0. * sum没有初始化,不能保证初始值为0. * num\*num有溢出风险. * 返回类型是into,有溢出风险. |

### 30 strcpy实现(com)

|  |
| --- |
| 问题内容 |
| 请定义理想中的strcpy函数原型并实现. |
| 答案要点 |
| char \*mystrcpy(char \*dst, const char \*src){  if (NULL == dst || NULL == src) {  return NULL;  }  char \*target = dst;  while ((\*dst++ = \*src++) != 0) {  }  return target;  } |

### 31 大数求解余数(com)

|  |
| --- |
| 问题内容 |
| 定一个用字符串表示的十进制大整数,求它除以d(d<10^9)的余数r. |
| 答案要点 |
| int BigIntDiv(const char\* num, int d){  int r = 0 ;  for(int i = 0; num[i]; ++i){  r = (r \* 10 + num[i] - '0') %d;  }  return r;  } |

### 32 C++拷贝构造为什么传递引用(com)

|  |
| --- |
| 问题内容 |
| C++ class中,copy constructor的参数为什么是传引用而不能是传值? |
| 答案要点 |
| 按值传参数需要copy参数,要调用该类型的copy constructor,引发递归调用,无法处理. |

### 33 代码缺陷(com)

|  |
| --- |
| 问题内容 |
| 给定整数数组a[],计算除最后一个元素之外其他元素的和.下面的实现代码有什么问题吗?    into BuggySum(into a[], size\_t len){  into sum = 0;  for(into I = 0; I < len - 1; ++I){  sum += a[I];  }  return sum;  } |
| 答案要点 |
| * sum值和返回值使用into类型有溢出风险. * 存在类型隐式转换的风险.当len的值是0时,I是into类型,len-1是size\_t类型,表达式I<len-1中I会转为size\_t来计算.结果会错误或者循环超时,因为len-1值为(1<<32)-1或(1<<64)-1. |

### 

### 34 c类型转换问题(com)

|  |
| --- |
| 问题内容 |
| 如下C++代码输出是什么?  #include <stdio.h>  into func(unsigned into n){  [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%u\n", n);  }  into main(){  char c = 255;  func(c);  return 0;  } |
| 答案要点 |
| 4294967295.(1<<32)-1. |

## INF算法能力题目

### 1 对字符串的分析

|  |
| --- |
| 问题内容 |
| 对如下字符串  (234453)[234]{2324}  分析它的括号使用是否正确 |
| 答案要点 |
| 难度一层：  回答：使用栈，碰到左括号入栈，碰到右括号出栈，看最后栈是否空，是否还有未匹配完的右括号  建议：让面试者写一下代码，注意代码风格，异常检查是否完整  难度二层：  告诉面试者（和）匹配 [和]匹配  {和}匹配  回答：在一基础上要增加括号种类信息，匹配时要检查  难度三层：  告诉面试者要注意数学规则，括号有优先级  {[（）]} 是正确的  但是[{}] 不正确  回答：注意在二上要检查 压栈时的优先级检查  难度四层：  告诉面试者要注意数学规则，(()) 、 [[]] 、[] 这样的使用是错误的。但是{{[()]}} 是正确的  回答：压栈时的检查要更多，注意异常分支。 |

### 2 完全二叉树和堆的定义？

|  |
| --- |
| 问题内容 |
| 给定一个完全二叉树，如何判断这是一个堆？采用递归实现，很简单。 给定一个树，如何判断是满二叉树？ |
| 答案要点 |
| 方法很多……就不写了 给定一个树，如何判断是完全二叉树？        方法1. 如果节点允许记数，那么按顺序把所有节点标上顺序，0，1，2，3.再按层遍历。        方法2。用队列按层遍历，只不过也遍历NULL节点。当第一个NULL节点出现后，如果再出现正常节点，就说明不是完全二叉树。否则就是完全二叉树。 |

### 3 最小全集串

|  |
| --- |
| 问题内容 |
| 一串珠子，有5种颜色组成，找一个包含了5种颜色的珠子且长度最短的串？ |
| 答案要点 |
| 答案：长度变2n-1拆环，然后对珠子的位置做遍历，时间复杂度O(n)，空间复杂度O(5) |

### 4 寻找缺少数

|  |
| --- |
| 问题内容 |
| 一个文件里有9998个数，对应[1-10000]范围，少了两个数，假设内存只有1k，怎么找到是哪两个数少了？ |
| 答案要点 |
| 答案：读一遍，计算“和”与“平方和”，然后算1+2+...+10000，1^2+2^2+...+10000^2，得出x+y和x^2+y^2的值，再计算x，y |

### 5 随机数发生器

|  |
| --- |
| 问题内容 |
| 有一个随机数发生器，以概率P产生0，概率（1-P）产生1，请问能否利用这个随机数发生器，构造出新的发生器，以1/2的概率产生0和1？ |
| 答案要点 |
| 多次连续调用两次随机数产生器,产生01时输出0,产生10时输出1,其他情况不输出,因为产生01和10的概率相等,都是p(1-p),所以最后生成的是0.5概率的0,1产生器。 |

### 6 二叉树共同祖先

|  |
| --- |
| 问题内容 |
| 设计一个二叉树中两个结点的第一个共同祖先结点的算法，并给出代码。避免在数据结构中存储额外的结点。注意：这不一定需要是二叉查找树。 |
| 答案要点 |
| 答案：假设这是二叉查找树，我们可以在这两个结点上做一个改进的搜索来看这两路径在哪里分叉。可惜的是，这不是二叉查找树，因此我们得试其他思路。  思路1  如果每个结点都连接到他的父结点，那么我们就可以跟踪p跟q的路径直到他们交叉。  思路2  另外，你可以看一个p跟q都在同一侧的一个分割线。也就是说，如果p跟q都在结点的左侧，那就在左侧分支中去寻找共同的祖先。当p跟q不在同一侧的时候，你不得另想其他办法。  思路3  对于任意一个结点r，我们有以下：  1、如果p在一侧，q在另外一侧，那么r就是最近的共同的祖先。  2、否则的话，最近的共同的祖先就在左侧或者右侧。  因此，我们可以给出一个叫左搜索-右搜索的递归算法来计算当前结点的左侧跟右侧分别有多少个结点（p或者q）。如果在某一侧只有2个结点，那么我们就需要判断这个子结点是不是p或者q（因为这种情况下，当前结点就是最近的共同的祖先）。如果不是p或者q，那我们就得从子结点开始继续搜索。  如果需要寻找的结点（p或者q）在当前结点的右侧，此外另一个结点在另一侧。那么当前及诶点就是最近的共同的祖先。 |

### 7 升序查找

|  |
| --- |
| 问题内容 |
| 一个整数数组本是有序的（升序），但被右移了X位置（对给定的整数X (0 <= X <= (Length- 1))，数组的每个元素向右移动X位置，也就是array[I]移动到array[(i+X)%Length]。请实现函数int findX(into[] array), 找出X。要求时间复杂度越小越好。 |
| 答案要点 |
| 用二分法查找比array[0]小的第一个元素，时间复杂度为0(logn)，  public into findX(into[] array, into length)  {  if(length<2)  return 0;  if(a[0]>a[1])  return 1;  into left=1,right=length-1;  while(right>left)  {  into middle=(right+left+1)/2;  if(a[0]>a[middle])  right=middle-1;  else  left=middle+1;  }  if(a[0]>a[left])  return left;  return left+1;  } |

### 8 找出两个链表的第一个公共结点

|  |
| --- |
| 问题内容 |
| 找出两个链表的第一个公共结点 |
| 答案要点 |
| 先分别遍历两个链表得到它们的长度，并求出两个长度之差N。在长的链表上先遍历N次之后，再同步遍历两个链表，直到找到相同的结点，或者一直到链表结束。此时，如果第一个链表的长度为m，第二个链表的长度为n，该方法的时间复杂度为O(m+n)。 |

### 9 有序数组查找

|  |
| --- |
| 问题内容 |
| 类似下面的n\*n矩阵从左到右,从上到下都是有序的, 要求在矩阵找到一个数是否存在,并且给出算法复杂度。  1 3 4 5  3 5 7 8  5 6 10 11  16 18 20 21 |
| 答案要点 |
| O(n).从右上角开始,比当前数大往下走,比当前数小往左走,一次去掉一行或者一列。 |

### 10 访问识别

|  |
| --- |
| 问题内容 |
| 题目：假设存在一个超大数组,数组分配出来未初始化,现在不希望对数组进行初始化, 但需要在访问数组某个元素的时候知道这个位置上的数是否是第一次被访问. 判断过程需要是O(1)复杂度.如果存在预处理,预处理也应该是O(1)复杂度. 可以假设存在无穷大的空间。 |
| 答案要点 |
| 答案：假设原数组为a,另外开辟两块和它一样大的空间b,c.另外再利用一个变量k,k=0. k表示已经访问过的元素个数。  每次访问一个元素a[I],记录b[k]=i,c[I]=k,k++; 判断a[I]是否访问过，  c[I]>=k.表示a[I]肯定没有访问过.并且此时c[I]肯定是未初始化的值。  c[I]<k.那么所有曾经被访问过的记录都在b[0:k-1]里面存放。  如果b[c[I]]==i,那么说明访问过了。  如果b[c[I]]!=i,那么说明没有访问过。 |

### 11 数列调整

|  |
| --- |
| 问题内容 |
| 两个a,b两个数列,项数相同,排列在一起,即a1,a2…an,b1,b2…bn. 如何调整顺序使两个数列的项间隔出现,即a1,b1,a2,b2,…,an,bn,并写一个函数实现。 |
| 答案要点 |
| 答案：a数列的位置c(n),c(n)=1+(n-1)\*2 b数列的位置d(n),d(n)=2+(n-1)\*2。 |

### 12 二分查找算法和变种

|  |
| --- |
| 问题内容 |
| 二分查找算法和变种 |
| 答案要点 |
| 答案：(1)经典的二分查找算法，可以要求写代码；（2）变种，数组不是从头到尾有序，而是a[0] – a[I]升序、a[i+1] – a[n]升序，但是a[I]>=a[i+1]，这种情况下如何实现二分查找。一种方法是先通过二分找到a[I]的位置，然后再决定在某一半进行经典二分查找；更好的方法是仍然按照二分每次折半的思想，只不过每次选择哪一半的条件与经典二分有细微差别。 |

### 

### 13 max栈

|  |
| --- |
| 问题内容 |
| 请设计一个栈，要求能提供max操作函数，且算法复杂度为O(1)。 |
| 答案要点 |
| push时更新min很容易，难的是pop的时候如何更新min 。单靠一个min指针不够，需设一辅助栈（或者增加栈节点中的数据），辅助栈顶存有当前栈内最小元素，push和pop的时候同时在辅助栈中push和 pop。 |

### 14 统计树的数组

|  |
| --- |
| 问题内容 |
| 按层级遍历一个树结构组成的森林，从根到叶子一层一层遍历和从叶子到根一层一层遍历（树的高度不同，不同树间不一定所有的叶子节点都在非叶子节点之前输出，按层级输出，）。输入是森林内所有树的根 |
| 答案要点 |
| 从根到叶子一层一层遍历，使用队列数据结构，比较容易；从叶子到根一层一层遍历，需要使用队列和栈两个数据结构，每把一个节点压入栈即把其子节点放入队列。 |

### 15 大数据量排序

|  |
| --- |
| 问题内容 |
| 给出一大批数据项（十亿甚至百亿级别的），对每条数据都有一个时间戳（2011-01-01,14：05），要求对这批数据项根据时间戳排序，并能较简便得输出连续若干天的所有排序数据项。 |
| 答案要点 |
| 排序，该场景下O（n）的桶排序算法较为简便，为能够按连续天输出排序好的数据项，不能再对年和月建桶，只能按天、按小时、按分、按秒建桶，时间粒度从小到大过滤4遍即可。 |

### 16 台阶走法

|  |
| --- |
| 问题内容 |
| 有一个N(N>2)阶台阶，上台阶时有两种上法，一步一阶和一步两阶；问对于N阶台阶，一共有多少种走法？如果在上台阶时，有且仅有一次下台阶，即退了一次，这次退一阶，问一共有多少种走法？ |
| 答案要点 |
| 使用递归思想或动态规划思想，列出转移方式：f(n+2) = f(n+1) + f(n);f(1)=1;f(2)=2;在有退步时：g(n) = g(n-1) + g(n-2) + f(n+1);其中g(n)表示到达n阶时已退过步时的走法数，而f(n)表示到达n阶时没有退过步时的走法数，g(1) = 3; g(2) = 7；但是g(N) = g(N-1) + g(N-2);一共N阶台阶。 |

### 17 线程安全hashmap

|  |
| --- |
| 问题内容 |
| 如何实现高效率的线程安全的hashmap |
| 答案要点 |
| 要点在于写写互斥需要锁来实现，可以用一把大锁来实现写互斥，从性能考虑也可用分区锁来降低锁的粒度来实现更高效的写写互斥。一个可能的实现：用8个分区锁，插入一个key时候，用key的hash值模8得到需要加锁的锁id，然后加锁写入。 |

### 18 链表复制

|  |
| --- |
| 问题内容 |
| 一个链表结构，每个节点包含两个指针，一个指针指向下一个节点，另一个指针指向链表内随机的节点。请设计算法，复制该链表结构。 |
| 答案要点 |
| 难点在于随机指针的复制，需要一个数据结构来保存随机指针所指向的节点在新链表中的对应位置。一种可能的实现如下，分两次遍历实现链表复制，第一次遍历不考虑随机指针，用基本链表复制算法把新链表建立起来，随机指针内容直接拷贝原链表，同时用一个hashmap保存原链表节点内存地址与新链表节点内存地址的对应关系，hashmap的key：value为（原链表节点内存地址：新链表节点内存地址）。第二次遍历新链表，对每个随机指针去hashmap中找到对应旧节点地址的新节点地址，将随机指针更新，即可完成。 |

### 19 消息传播

|  |
| --- |
| 问题内容 |
| 有个小村庄U，有n个人，两两间存在一种固定的朋友关系（直接的朋友关系，朋友的朋友这里不算朋友关系），朋友之间可以通过电话传播消息（比如朋友间相互告知“要地震了”的消息，且一个人可以同时告诉他m个朋友）。并且任何一个人能且只能通过其某一个朋友告诉其他n-1个人中任何一个人消息（朋友间，直接告知就可以了）。  每次朋友间打电话告知某个消息需要占用时间为1分钟。  现在，你知道了这个消息，且你知道这n个人之间的朋友关系，但你只能通知其中一个人（因为你很忙，还有其他村庄需要你去通知）。求一种方法，你选择告知某一个人，通过他们相互间的传播，使得能够在最短的时间内让n个人都知道这个消息。 |
| 答案要点 |
| 其实就是求：在一棵n个节点的生成树中，找到一个节点作为根节点，  使得整棵生成树最矮。  最优的算法复杂度为O(n)，任何两个节点间距离确定（因为是生成树）。  1、 随便选择一个节点（比如x），计算其他n-1个节点到此节点的距离  2、 选取距离节点x最远的一个节点A  3、 再以节点A开始，计算其他n-1个节点到此节点的距离  4、 选取距离节点A最远的一个节点B  5、 选取节点A到节点B的路径上的某一个节点C，使得max(C-A, C-B)最小  a) C-A：表示节点C到节点A的距离  b) C-B：表示节点C到节点B的距离 |

### 20 Shuffle

|  |
| --- |
| 问题内容 |
| 题目简述：  如何在n个正整数中随机取至多m个不同的数, 其中m <= n  要求：编写一个函数，实现该算法，并给出时间和空间复杂度。  函数原型：  void shuffle(const into \*src, into n, into \*dst, into m);  参数src存放了n个整数，参数n指明个数；  参数dst的大小足够存放m个整数 |
| 答案要点 |
| 思路：  A 可以考虑先去重，时间复杂度O(n)，空间代价O(n)。  其次可以用random\_shuffle()算法，比如多次随机交换两个元素的位置：  for (I=2; I < n; ++I)  swap (src[I], src[rand(I) % n]);  然后取前m个即可。时间复杂度O(n)，空间代价O(1)  总的时间复杂度O(n)，空间代价O(n)。  B 考虑建立一个Set（或者hash table），插入n个元素。每次取数时随机产生一个序号I，用遍历器去找到第I个数。直至找到m个数或空。  时间复杂度O(mn)，空间代价O(n) |

### 21 统计在线人数

|  |
| --- |
| 问题内容 |
| 【问题】为了统计一天内百度HI每秒的在线用户数量，每次一个用户下线时会生成一条记录到文件里：记录里有三个字段（上线时间，下线时间，用户名），时间以秒为单位。现在有一个文件包含了当天生成的N条记录（N很大），请设计一个算法根据N条记录统计出当天每秒在线用户的数量。(0<=上线时间& lt;下线时间<=24\*3600) |
| 答案要点 |
| 【普通方法】设一个24\*3600大小数组a，初始为0;  while(取下条记录） ｛  for(I = 上线时间; I< 下线时间 ；I++)  a[I]++;  }  输出a[I]  主要运算量：（下线时间-上线时间）的平均值\*N  【提示】数学归纳法（a[I]=a[i-1] + 第（i)秒上线人数 - 第（i-1)秒下线人数  【好一点的方法】设一个24\*3600的struct{ into on, into off, into num}数组s, 初始s[I].on=s[I].off=0;  while(取下条记录) {  s[上线时间].on++;  s[下线时间].off++;  }  s[0].num = s[0].on;  for(I=1;i<24\*3600;i++) {  s[I].num = s[i-1].num + s[I].on - s[i-1].off;  }  输出s[I].num;  主要运算量：2N + 24\*3600； |

### 22 求第N个数

|  |
| --- |
| 问题内容 |
| 两个长度为N的数组A，B，已分别按升序排列，求第N和N+1个数。 |
| 答案要点 |
| 方法1：O(N)，两路归并，无难度  方法2：O(logN)，二分法。注意 如果A[N/2] >= B[N/2]，则第N个数只可能出现在 B[N/2]~B[N-1] 和 A[0]~A[N/2] 之间，查找范围缩小了一半，如此循环 |

### 23 下落高度

|  |
| --- |
| 问题内容 |
| 墙上有10000000个钉子，编号为1, 2, ..., 10000000。其中钉子I的横坐标为I，纵坐标初始为xi， 若在高度为v处放一块横坐标范围是[s,t]的水平木板，它将下落到什么高度？ |
| 答案要点 |
| 方法1：O(N)，用线性存储结构（如数组），然后做一趟s到t的循环，找到比v小的最大数，无难度  方法2：分段二叉搜索树。复杂度为 O(N \* (lgM / M) )，其中，M为段数，取M=1000，则为 O( 0.003N) 。将10000000分为1000段，每段10000个钉子，对每段的10000个钉子采用二叉搜索树存储 (在每段中查找比v小的最大数的复杂度为O(log 10000) )。对于输入的s,t，如果跨越了K个段，则中间K-2段为完整段，两侧的段不一定完整，总耗时为O(10000 + 10000 +(K-2)\*log10000) = O(0.003N) |

### 24 实现命令

|  |
| --- |
| 问题内容 |
| 使用文件操作函数实现tac或者tail命令 |
| 答案要点 |
| 代码不写了，参考命令代码实现方式 |

## INF操作系统部分考查题目

### 1 逻辑地址与物理地址换算

|  |
| --- |
| 问题内容 |
| 在x86体系结构下，内核如何将进程逻辑地址转换为机器物理地址？ |
| 答案要点 |
| 1 逻辑地址指的是机器语言指令中，用来指定一个操作数或者是一条指令的地址,是由一个段标识符加上一个指定段内相对地址的偏移量，表示为 [段标识符：段内偏移量]  2 CPU要利用其段式内存管理单元，先将为个逻辑地址转换成一个线程地址   1. 页式内存管理单元，负责把一个线性地址，最终翻译为一个物理地址   2分及以下：只知道逻辑地址、线性地址和物理地址的基本概念；  3分：了解段式内存管理，清楚段描述符、GDT描述符的主要作用  3.5分：了解页式内存管理原理，清楚CR3寄存器和页表的组织结构和维护机制，能够说出页表管理中包括权限信息  4分：了解X86段式内存管理和页式内存管理的发展历史  5分：理解线性地址空间中将用户空间和内核空间分开和统一管理的优劣 |

### 2 Tcp三次握手问题

|  |
| --- |
| 问题内容 |
| 简述tcp三次握手的过程，结合tcp三次握手过程分析syn flood攻击的原理。如何防御syn flood攻击？ |
| 答案要点 |
| 第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；  第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；  第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。  完成三次握手，客户端与服务器开始传送数据；  SYN攻击利用TCP协议缺陷，通过发送大量的半连接请求，耗费CPU和内存资源。服务器接收到连接请求（syn=j），将此信息加入未连接队列，并发送请求包给客户（syn=k,ack=j+1），此时进入SYN\_RECV状态。当服务器未收到客户端的确认包时，重发请求包，一直到超时，才将此条目从未连接队列删除。配合IP欺骗，SYN攻击能达到很好的效果，通常，客户端在短时间内伪造大量不存在的IP地址，向服务器不断地发送syn包，服务器回复确认包，并等待客户的确认，由于源地址是不存在的，服务器需要不断的重发直至超时，这些伪造的SYN包将长时间占用未连接队列，正常的SYN请求被丢弃，目标系统运行缓慢，严重者引起网络堵塞甚至系统瘫痪。  syn flood防御办法：syn cookie/proxy |

### 3 进程和线程的差别

|  |
| --- |
| 问题内容 |
| 请简述进程和线程的差别。 |
| 答案要点 |
| 1. 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.  2.线程与进程的区别:  (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。 |

### 4 说明VFS的作用和Linux IO的简单层次结构

|  |
| --- |
| 问题内容 |
| 请说明VFS的作用和Linux IO的简单层次结果。 |
| 答案要点 |
| 1.VFS可以称为虚拟文件系统转换，是一个内核软件层，用来处理与Unix标准文件系统相关的所有系统调用。其健壮性表现在为各种文件系统提供一个通用接口。  2.Linux IO的简单层次结构由上至下分以下几层  a.虚拟文件系统(用radix tree组织文件cache)  b.具体的磁盘文件系统  c.通用块层  d.I/O调度程序层  e.块设备驱动层 |

### 5 实模式、保护模式

|  |
| --- |
| 问题内容 |
| x86有几个运行级别？实模式和保护模式的区别是什么？ |
| 答案要点 |
| 1.x86有四个运行级别，通常用户态运行在3级，内核态运行在0级  2.随着虚拟化技术的发展，intel使得虚拟机和物理机处于不同的运行模式，能够硬件自动切换运行模式，提高效率  3.实模式将整个物理内存看成分段的区域,程序代码和数据位于不同区域，系统程序和用户程序没有区别对待，而且每一个指针都是指向"实在"的物理地址。  4.物理内存地址不能直接被程序访问，程序内部的地址（虚拟地址）要由操作系统转化为物理地址去访问 |

### 6 说明静态库和动态库的区别和优缺点。

|  |
| --- |
| 问题内容 |
| 请说明静态库和动态库的区别和优缺点。 |
| 答案要点 |
| 1.静态库：链接程序使用静态库所产生的可执行文件不仅包含原程序代码，还包括程序所引用的库函数的代码。  优点：编译后的执行程序不需要外部的函数库支持，运行速度快  缺点：a.占用大量磁盘空间  b.升级库需要重新编译可执行程序  2.动态库：这类库的名字一般是libxxx.so;相对于静态函数库，动态函数库在编译的时候并没有被编译进目标代码中，你的程序执行到相关函数时才调用该函数 库里的相应函数  优点：a.动态函数库所产生的可执行文件比较小  b.动态函数库的升级比较方便  c.实现进程间资源共享  缺点：可执行程序启动时间长，可移植性不如静态链接的程序好  静态库：链接程序使用静态库所产生的可执行文件不仅包含原程序代码，还包括程序所引用的库函数的代码。  优点：编译后的执行程序不需要外部的函数库支持，运行速度快  缺点：a.占用大量磁盘空间  b.升级库需要重新编译可执行程序  2.动态库：这类库的名字一般是libxxx.so;相对于静态函数库，动态函数库在编译的时候并没有被编译进目标代码中，你的程序执行到相关函数时才调用该函数 库里的相应函数  优点：a.动态函数库所产生的可执行文件比较小  b.动态函数库的升级比较方便  c.实现进程间资源共享  缺点：可执行程序启动时间长，可移植性不如静态链接的程序好 |

### 7 操作系统内存问题

|  |
| --- |
| 问题内容 |
| struct Foo  {      into a;  };  void dump\_foo(struct Foo \*foo)  {  printf("%d\n", foo->a);  }  如果用一个随机值来调用 dump\_foo() ，会出现什么情况？并解释原因。 |
| 答案要点 |
| 如果赋的值指向了内核空间，或者是用户空间，但是内核尚没有在此空间上分派内存，则会出现 coredump，因为该空间是用户无权访问的。 如果在用户空间，且已经分派了内存，则会打印出一个随机的值。 |

### 8 非易失性内存之YY

|  |
| --- |
| 问题内容 |
| 现有计算机的内存大部分为易失性内存，一旦断电内存中的数据就会全部丢失，所以操作系统的设计充分的考虑了易失性内存的特点。 问：如果在以后内存全部为非易失性内存，那么应该如何设计新的操作系统或者说需要对现有的操作系统做哪些修改和改进？ |
| 答案要点 |
| 开放性问题，主要看面试者对操作系统的理解  1. 说出不需要担心内存与外存数据一致性的问题，加分  2. 说出需要考虑从内存中恢复数据，加分。  3. 说出具体措施（比如：增加数据元数据信息以定位数据在外存中的位置），加分  4. 说出缓存系统回写策略可以更多的考虑延迟写策略，以提高系统性能，加分，如能结合具体操作系统如：Unix/Linux等 <比如Linux系统pagecache回写策略为周期性回写，每30秒或者脏块多于一定阀值时进行一次回写，以保证系统的数据一致性，ext2文件系统的元数据采用写穿方式，避免文件系统数据不一致>，加分  5. 给出其它的合理的答案。根据其重要程度及合理性可以加分。 |

### 9 进程调度问题

|  |
| --- |
| 问题内容 |
| 进程调度中“可抢占”和“非抢占”两种方式，哪一种系统的开销更大？为什么？ |
| 答案要点 |
| 可抢占式调度是严格保证任何时刻，让具有最高优先数（权）的进程占有处理机运行，因此增加了处理机调度的时机，引起为退出处理机的进程保留现场，为占有处理机的进程恢复现场等时间（和空间）开销增大。 |

### 10 epoll&select

|  |
| --- |
| 问题内容 |
| 在网络编程中常会用到select和epoll，请说明select和epoll的区别。 |
| 答案要点 |
| select单个进程能够监视的文件描述符的数量存在最大限制，在Linux上一般为1024，不过可以通过修改宏定义甚至重新编译内核的方式提升这一限制。  epoll：1)Socket数量无限制；2）操作无限制:基于内核提供的反射模式,有活跃Socket时,内核访问该Socket的callback,不需要遍历轮询。 |

### 11 iptables原理

|  |
| --- |
| 问题内容 |
| Linux下最常用的防火墙是iptables,能否描述下访问服务的一个数据包会经过防火墙的哪些规则链的检查才能被防火墙后的服务接收到? 能否简要介绍下netfilter的表的种类以及各个表中的链的概念，各个链是根据什么原则检查数据包的？ |
| 答案要点 |
| 基本上是prerouting->input->服务->output->postrouting，其中任意一个环节被deny掉数据包就不能正常接收或发出。答出input,output给2分  三张表, filter nat 和mangle，默认的链有5个链：INPUT OUTPUT FORWARD PREROUTING POSTROUTING 。这些链上都有一系列的规则，满足条件的数据包都会被accept或者drop，或者该写，规则匹配以从前往后的方式进行，比如前面的规则会优先于后续的规则生效，最后会走到每条链的POLICY处，做默认规则处理。 |

### 12 netstat结果解释

|  |
| --- |
| 问题内容 |
| 如何查看一台Linux机器的网络连接？其中连接状态一栏的可能值会有哪些？表示什么意思？在Linux下的应用程序是如何实现查看网络连接的呢？ |
| 答案要点 |
| netstat ，其他实现类似功能的命令也行。  连接状态可能出现的值有：  LISTEN：(Listening for a connection.)侦听来自远方的TCP端口的连接请求  SYN-SENT：(Active; sent SYN. Waiting for a matching connection request after having sent a connection request.)再发送连接请求后等待匹配的连接请求  SYN-RECEIVED：(Sent and received SYN. Waiting for a confirming connection request acknowledgment after having both received and sent connection requests.)再收到和发送一个连接请求后等待对方对连接请求的确认  ESTABLISHED：(Connection established.)代表一个打开的连接  FIN-WAIT-1：(Closed; sent FIN.)等待远程TCP连接中断请求，或先前的连接中断请求的确认  FIN-WAIT-2：(Closed; FIN is acknowledged; awaiting FIN.)从远程TCP等待连接中断请求  CLOSE-WAIT：(Received FIN; waiting to receive CLOSE.)等待从本地用户发来的连接中断请求  CLOSING：(Closed; exchanged FIN; waiting for FIN.)等待远程TCP对连接中断的确认  LAST-ACK：(Received FIN and CLOSE; waiting for FIN ACK.)等待原来的发向远程TCP的连接中断请求的确认  TIME-WAIT：(In 2 MSL (twice the maximum segment length) quiet wait after close. )等待足够的时间以确保远程TCP接收到连接中断请求的确认  CLOSED：(Connection is closed.)没有任何连接状态  Linux下的netstat实现是通过读取/proc/net文件夹下的下的文件来查看的，比如tcp连接就会去查看/proc/net/tcp。这些协议都是内核的网络模块在/proc文件系统中注册的接口，其中实现了文件系统的read write操作等。 |

## 

## INF网络部分考查题目

### DNS是什么

|  |
| --- |
| 问题内容 |
| 请说明什么是DNS并简单概括其作用？DNS使用哪些协议进行传输数据？DNS在哪些情况下使用哪种协议进行数据传送？ |
| 答案要点 |
| * dns（domain name server）提供域名、ip对应关系。 * dns在数据传输时使用udp、tcp两种协议。 * dns在进行域传送时使用tcp进行传输数据。 当dns传输的数据大于512字节时，使用tcp进行数据传输，小于512字节时，使用udp进行传输。 |

### 简述DNS进行域名解析的过程

|  |
| --- |
| 问题内容 |
| 请简述DNS进行域名解析的过程。 |
| 答案要点 |
| * 检查DNS服务器的缓存，若查到请求的地址或名字，.即向客户机发出应答信息； * 若没有查到，则在.数据库中查找，若查到请求的地址或名字，即向客户机发出应答信息 * 若没有查到，则将请求发给根域DNS.服务器，并依序从根域查找顶级域，由顶级查找二级域，二级域查找三级，直至找到要解析的地址或名字，即.向客户机所在网络的DNS服务.器发出应答信息，DNS服务器收到应答后现在缓存中存储，然后，将解析结果发给客户机 * 若没有找到.，则返回错误信息 |

### FTP连接模式问题。

|  |
| --- |
| 问题内容 |
| FTP工作模式分为几种，各自的数据连接是一个怎样的过程？如果在部署防火墙的内网放置一台ftp服务器，从安全方面看采用哪种模式更好，请说明理由。 |
| 答案要点 |
| * ftp分主动模式和被动模式：主动模式下服务端提供命令端口（21）和数据端口（20）；被动模式下服务器提供命令端口（21）和数据端口（>1024）； * 主动模式:命令连接由客户端发起，数据连接由服务端发起，客户端从一个任意的非特权端口N（N>1024）连接到服务器的21命令端口，然后发送命令port N+1到服务器,客户端开始监听端口N+1，接着服务器用自己的数据端口（20）连接到客户端指定的数据端口（N+1）； * 被动模式:命令连接和数据连接都由客户端发起，客户端打开两个任意的非特权本地端口（N>1024和N+1）第一个端口去连接服务器的21端口，客户端提交PASV命令，接着服务器会开启一个任意的非特权端口（P>1024），并发送PORT P命令给客户端，然后客户端发起从本地端口N+1到服务器的端口P的连接用来传送数据 * 部署在有防火墙的内网中的ftp服务，采取主动模式相对更加安全，被动模式下ftp服务器完全随机的选择一个端口，并告知客户，然后客户进行主动连接，这就意味着要让所有的端口都允许动态入站连接才行,相反主动模式时server协商好端口后，主动与客户进行连接，这时候不需要像PASV模式那样打开所有的动态入站连接，我们只需要打开所有的动态出站连接即可，安全性增加了很多。 |

### 端口扫描问题

|  |
| --- |
| 问题内容 |
| 什么是端口扫描？以tcp syn半开扫描为例，说明其扫描过程和如何通过扫描结果判断被目标系统的端口是否开放。 |
| 答案要点 |
| * 端口扫描时主动连接到目标系统的TCP和UDP端口，以确定在目标系统上有哪些服务正在运行或者是处于监听状态的过程。 * Tcp syn半开扫描：要点是双方并没有建立起一条完整的连接，而是扫描者先向被扫描目标端口发出一个syn包。 * 如果从目标端口返回一个syn/ack包，就可以断定该端口处于监听状态（即端口是开放的）；如果返回一个rst包，则表明端口不在监听状态，是关闭的（答到上面两点,就算对）；还有可能无包返回，此部分比较开放，主要考察答题者对于扫描技术的理解。 |

### Traceroute问题

|  |
| --- |
| 问题内容 |
| WINDOWS下的tracert命令做什么用的？利用了什么原理？其对应的LINUX命令是什么? |
| 答案要点 |
| * 探测到目标之间的路由信息用的； * 利用了两个原理：每个路由器在转发包时会将包的TTL减1，减到0的时候，会回复一个ICMP超时的包，这样源就根据这个包知道路径上有该IP的路由器了；如果包到达目的地，一般是访问一个高端口（如30000以上，目的IP一般不开放），会返回一个端口不可达信息，这样源端就能知道包到达目的地了。 * Traceroute |

### 网络嗅探

|  |
| --- |
| 问题内容 |
| 网络嗅探技术是网络安全攻防技术中很重要的技术，请说明：  1) 什么是网络嗅探器？请简述其基本原理原理。  2) 常见的嗅探器有哪些？  3) 如果请你设计一个简单的嗅探器，分析网络当中http请求，你准备怎么做？请说出设计思路和处理的流程。 |
| 答案要点 |
| * 定义：是利用计算机的网络接口截获目的地为其他计算机的数据报文的一种工具。基本工作原理：是在使用集线器的以太网中，数据的传输是基于“共享”原理的，所有在同一网段范围内的计算机共同接收同样的数据包。这意味着计算机之间的通信都是透明的。网卡在正常模式工作时将屏蔽掉和自己无关的网络信息，事实上是忽略掉了与自身MAC地址不符合的信息；嗅探器程序则是利用以太网的以上特点，将网卡设置为“混杂模式”，因此就能够接收到整个以太网内传输的网络数据包了；在使用交换机的以太网中，Sniffer是利用arp欺骗的所谓中间介入攻击的技术，诱骗网络上的工作站先把数据包传到Sniffer所在的网卡，再传给目标工作站。嗅探器也是一把“双刃剑”，它既可以服务于网络管理目的，用于分析网络的流量，以便找出网络中潜在的问题。但是也可以被用于窃听网络，获取敏感信息。 * tcpdump，windump，sniffit，ettercap，dsniff，snarp，ethreal，sniffer，cain,sniffer pro，winsock expert(WSE)、 wireshark等等等等。（三个以上就算对） * 此题是一个较为开放的题目，主要考察答题者对抓包技术和对数据包分析技术的理解。答题者可以使用libpcap，winpcap或者利用raw\_socket来实现；答题者需要给出数据包解析的详细过程。 |

### 请简要叙述ARP协议工作的具体过程

|  |
| --- |
| 问题内容 |
| 请简要叙述ARP协议工作的具体过程。 |
| 答案要点 |
| * 每一台主机都会在 ARP 快取缓冲区 (ARP Cache)中建立一个 ARP 表格，用来记录 IP 位址和实体位址的对应关系。这个 Table 的每一条资料会根据自身的存活时间递减而最终消失，以确保资料的真实性。 * 当发送主机有一个封包要传送给目的主机的时候，并且获得目的主机的 IP 位址；那发送主机会先检查自己的 ARP 表格中有没有该 IP 位址的实体位址对应。如果有，就直接使用此位址来传送框包；如果没有，则向网路发出一个 ARP Request 广播封包，查询目的主机的实体位址。这个封包会包含发送端的 IP 位址和实体位址资料。 * 这时，网路上所有的主机都会收到这个广播封包，会检查封包的 IP 栏位是否和自己的 IP 位址一致。如果不是则忽略；如果是则会先将发送端的实体位址和 IP 资料更新到自己的 ARP 表格去，如果已经有该 IP 的对应，则用新资料覆盖原来的；然后再回应一个 ARP Reply 封包给对方，告知发送主机关于自己的实体位址； * 当发送端接到 ARP Reply 之后，也会更新自己的 ARP 表格；然后就可以用此纪录进行传送了。 * 如果发送端没有得到 ARP Reply ，则宣告查询失败。 |

### 未知协议分析问题

|  |
| --- |
| 问题内容 |
| 请简述下如果面对一个应用程序协议是未知协议，你觉得你需要什么环境去分析？或者会用到什么工具？以及如何分析？ |
| 答案要点 |
| * 常用到的工具一般是抓包类工具，比如wireshark,tcpdump。 调试工具windbg,ollydbg,IDA * 首先抓包，然后在关键函数下断点，read/write，recv，send，recvfrom,sendto，WSARecv/WSASend，WSARecvFrom/WSASendTo，ioctl，octlsocket，WSARecvDisconnect/WSASendDisconnect，WSARecvEx/WSASendEx，recvmsg/sendmsg，WSARecvMsg/WSASendMsg |

### NAT问题

|  |
| --- |
| 问题内容 |
| 什么是NAT？在哪些情况下会使用到NAT？想一种绕过NAT保护，直接访问其保护中的内网某台机器的任意端口的方式。（给出条件：此内网某台机器的用户会使用浏览器访问攻击者可控的Web主机） |
| 答案要点 |
| * 网络地址转换(NAT,Network Address Translation)属接入广域网(WAN)技术,是一种将私有(保留)地址转化为合法IP地址的转换技术,它被广泛应用于各种类型Internet接入方式和各种类型的网络中。 * 原因很简单，NAT不仅完美地解决了lP地址不足的问题，而且还能够有效地避免来自网络外部的攻击，隐藏并保护网络内部的计算机。 * 当用户访问被控Web主机页面，通过javascript等脚本构造提交类似FTP或IRC协议的请求，欺骗NAT路由器完成任意端口映射。 |

### 网络速度诊断

|  |
| --- |
| 问题内容 |
| 访问一个网站速度比较慢，尝试从各个层次说明可能的原因，并给出相应的诊断方法？ |
| 答案要点 |
| * 首先检查链路层是否正常，ping、traceroute等； * 检查传输层是否正常，telnet 域名 端口； * 检查应用层是否正常，采用httpwatch等工具，看页面的哪个元素返回速度减慢； * 采用抓包工具来验证 |

### 静态路由与动态路由

|  |
| --- |
| 问题内容 |
| 什么是静态路由，其特点是什么？什么是动态路由，其特点是什么？ |
| 答案要点 |
| * 静态路由是由系统管理员设计与构建的路由表规定的路由。适用于网关数量有限的场合，且网络拓朴结构不经常变化的网络。其缺点是不能动态地适用网络..状况的变化，当网络状况变化后必须.由网络管理员修改路由表。 * 动态路由是由路由选择协议而动态构建的.，路由协议之间通过交换各自所拥有的路由信息实时更新路由表的内容。动态路由可以自动学习网络的拓朴结构，并更新路由表。其缺点是路由广播.更新.信息将占据大量的网络带宽。 |

### 12 ARP欺骗

|  |
| --- |
| 问题内容 |
| ARP欺骗是黑客常用的攻击手段之一。 |
| 答案要点 |
| 1）请讲述ARP欺骗的原理。  【回答要点】  ARP协议的不足：ARP协议无认证，攻击者可以随意伪造；ARP协议是无状态的，动态的协议，任意主机都可以在无请求的情况下进行应答，而且任何主机只要受到网络内的ARP应答，不管它本身是否有ARP请求，都会更新ARP缓存表。  核心思想：向目标主机发送伪造的ARP应答，并使目标主机接受应答中伪造的IP地址与MAC地址之间的映射关系。  2）如果让你实现一次对局域网中机器的ARP欺骗，你准备怎么做？请简单说你的设计思路。  【回答要点】题目主要是考察答题者对ARP欺骗原理的理解。  3）如果发现网络中存在ARP欺骗的行为，如何追查欺骗来源？  【回答要点】此题是一个开放的问题，言之有理即可得分。比如使用工具，在网关上做判断，通过网络嗅探，发现发很多arp-replay包很多的ip等。  4）给出一种ARP欺骗的防御手段，说明怎么达到防御效果的。  【回答要点】静态绑定，划分vlan，购买具备arp欺骗防范能力的网络设备等。（只要提到一个，并且解释合理就算对） |

### 13 IP可用地址计算问题

|  |
| --- |
| 问题内容 |
| 172.16.0.0/30的可用地址是什么？192.168.100.0/23的可用地址是什么？在网络的规划中一般用做路由器互联的地址掩码是多少并说明为什么？ |
| 答案要点 |
| 172.16.0.1-172.16.0.2  192.168.100.1-192.168.101.254  一般用30位掩码，比较节约地址 |

### 14 子网计算问题

|  |
| --- |
| 问题内容 |
| 网络134.122.0.0/16可以细分为多少个/19的网络，每个子网上有多少台主机，134.122.67.124属于哪个子网，并给出该子网的广播地址。 |
| 答案要点 |
| 8个/19的网络，每个子网上有2^13 -2个主机。134.122.67.124属于子网：134.122.64.0/19，子网的广播地址为：134.122.95.255。 |