**一、数据结构**

**1、数组和链表的区别。**

**从逻辑结构上来看**，数组必须实现定于固定的长度，不能适应数据动态增减的情况，即数组的大小一旦定义就不能改变。当数据增加是，可能超过原先定义的元素的个数；当数据减少时，造成内存浪费；链表动态进行存储分配，可以适应数据动态地增减的情况，且可以方便地插入、删除数据项。  
**从内存存储的角度看**；数组从栈中分配空间（用new则在堆上创建），对程序员方便快速，但是自由度小；链表从堆中分配空间，自由度大但是申请管理比较麻烦。  
**从访问方式类看**，数组在内存中是连续的存储，因此可以利用下标索引进行访问；链表是链式存储结构，在访问元素时候只能够通过线性方式由前到后顺序的访问，所以访问效率比数组要低。

**2、简述快速排序过程**

1）选择一个基准元素,通常选择第一个元素或者最后一个元素,  
2）通过一趟排序将待排序的记录分割成独立的两部分，其中一部分记录的元素值均比基准元素值小。另一部分记录的元素值比基准值大。  
3）此时基准元素在其排好序后的正确位置  
4）然后分别对这两部分记录用同样的方法继续进行排序，直到整个序列有序。

**3、快速排序的改进**

只对长度大于k的子序列递归调用快速排序,让原序列基本有序，然后再对整个基本有序序列用插入排序算法排序。实践证明，改进后的算法时间复杂度有所降低，且当k取值为 8 左右时,改进算法的性能最佳。

**选择基准元的方式**

对于分治算法，当每次划分时，算法若都能分成两个等长的子序列时，那么分治算法效率会达到最大。也就是说，基准的选择是很重要的。选择基准的方式决定了两个分割后两个子序列的长度，进而对整个算法的效率产生决定性影响。最理想的方法是，选择的基准恰好能把待排序序列分成两个等长的子序列。

方法1 固定基准元

如果输入序列是随机的，处理时间是可以接受的。如果数组已经有序时，此时的分割就是一个非常不好的分割。

方法2 随机基准元

这是一种相对安全的策略。由于基准元的位置是随机的，那么产生的分割也不会总是会出现劣质的分割。在整个数组数字全相等时，仍然是最坏情况，时间复杂度是O(n^2）。实际上，随机化快速排序得到理论最坏情况的可能性仅为1/(2^n）。所以随机化快速排序可以对于绝大多数输入数据达到O(nlogn）的期望时间复杂度。

方法3 三数取中

引入的原因：虽然随机选取基准时，减少出现不好分割的几率，但是还是最坏情况下还是O(n^2），要缓解这种情况，就引入了三数取中选取基准。

分析：最佳的划分是将待排序的序列分成等长的子序列，最佳的状态我们可以使用序列的中间的值，也就是第N/2个数。可是，这很难算出来，并且会明显减慢快速排序的速度。这样的中值的估计可以通过随机选取三个元素并用它们的中值作为基准元而得到。事实上，随机性并没有多大的帮助，因此一般的做法是使用左端、右端和中心位置上的三个元素的中值作为基准元。

**4、各类排序算法对比**

**时间复杂度来说：**

(1)平方阶(O(n2))排序 各类简单排序:直接插入、直接选择和冒泡排序； (2)线性对数阶(O(nlog2n))排序 快速排序、堆排序和归并排序； (3)O(n1+§))排序,§是介于0和1之间的常数。

希尔排序 (4)线性阶(O(n))排序 基数排序，此外还有桶、箱排序。

说明：

当原表有序或基本有序时，直接插入排序和冒泡排序将大大减少比较次数和移动记录的次数，时间复杂度可降至O（n）；

而快速排序则相反，当原表基本有序时，将蜕化为冒泡排序，时间复杂度提高为O（n2）；

原表是否有序，对简单选择排序、堆排序、归并排序和基数排序的时间复杂度影响不大。

**稳定性：**

排序算法的稳定性:若待排序的序列中，存在多个具有相同关键字的记录，经过排序，这些记录的相对次序保持不变，则称该算法是稳定的；若经排序后，记录的相对次序发生了改变，则称该算法是不稳定的。

稳定的排序算法：冒泡排序、插入排序、归并排序和基数排序

不是稳定的排序算法：选择排序、快速排序、希尔排序、堆排序

**选择排序算法准则：**

一般而言，需要考虑的因素有以下四点：

设待排序元素的个数为n.

**1）**当n较大，则应采用时间复杂度为O(nlog2n)的排序方法：快速排序、堆排序或归并排序序。

**2）**当n较大，内存空间允许，且要求稳定性：归并排序

**3）**当n较小，可采用直接插入或直接选择排序。

直接插入排序：当元素分布有序，直接插入排序将大大减少比较次数和移动记录的次数。

直接选择排序 ：元素分布有序，如果不要求稳定性，选择直接选择排序

**5）**一般不使用或不直接使用传统的冒泡排序。

**6）**基数排序 它是一种稳定的排序算法，但有一定的局限性： 1、关键字可分解。 2、记录的关键字位数较少，如果密集更好 3、如果是数字时，最好是无符号的

**5、冒泡排序算法的改进**

1．设置一标志性变量pos,用于记录每趟排序中最后一次进行交换的位置。由于pos位置之后的记录均已交换到位,故在进行下一趟排序时只要扫描到pos位置即可。

2．传统冒泡排序中每一趟排序操作只能找到一个最大值或最小值,我们考虑利用在每趟排序中进行正向和反向两遍冒泡的方法一次可以得到两个最终值(最大者和最小者) , 从而使排序趟数几乎减少了一半。

**6、邻接矩阵与邻接表**

邻接矩阵表示法：在一个一维数组中存储所有的点，在一个二维数组中存储顶点之间的边的权值

邻接表表示法：图中顶点用一个一维数组存储，图中每个顶点vi的所有邻接点构成单链表

**对比**

1）在邻接矩阵表示中，无向图的邻接矩阵是对称的。矩阵中第 i 行或 第 i 列有效元素个数之和就是顶点的度。

在有向图中 第 i 行有效元素个数之和是顶点的出度，第 i 列有效元素个数之和是顶点的入度。

2）在邻接表的表示中，无向图的同一条边在邻接表中存储的两次。如果想要知道顶点的度，只需要求出所对应链表的结点个数即可。

有向图中每条边在邻接表中只出现一次，求顶点的出度只需要遍历所对应链表即可。求入度则需要遍历其他顶点的链表。

3）邻接矩阵与邻接表优缺点：

邻接矩阵的优点是可以快速判断两个顶点之间是否存在边，可以快速添加边或者删除边。而其缺点是如果顶点之间的边比较少，会比较浪费空间。因为是一个 n∗n 的矩阵。

而邻接表的优点是节省空间，只存储实际存在的边。其缺点是关注顶点的度时，就可能需要遍历一个链表。

**7. 用循环比递归效率高吗？**

递归和循环两者完全可以互换。不能完全决定性地说循环地效率比递归的效率高。

2.1递归算法：

优点：代码简洁、清晰，并且容易验证正确性。

缺点：它的运行需要较多次数的函数调用，如果调用层数比较深，需要增加额外的堆栈处理（还有可能出现**堆栈溢出**的情况），比如参数传递需要压栈等操作，会对执行效率有一定影响。但是，对于某些问题，如果不使用递归，那将是极端难看的代码。在编译器优化后，对于多次调用的函数处理会有非常好的效率优化，效率未必低于循环。

2.2循环算法：

优点：速度快，结构简单。

缺点：并不能解决所有的问题。有的问题适合使用递归而不是循环。如果使用循环并不困难的话，最好使用循环。

**8、解决哈希冲突的方法**

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。

1） 线性探测法

2） 平方探测法

3） 伪随机序列法

4） 拉链法

**9、KMP算法**

在一个字符串中查找是否包含目标的匹配字符串。其主要思想是每趟比较过程让子串先后滑动一个合适的位置。当发生不匹配的情况时，不是右移一位，而是移动（当前匹配的长度– 当前匹配子串的部分匹配值）位。

**10、B树**

根据B类树的特点，构造一个多阶的B类树，然后在尽量多的在结点上存储相关的信息，保证层数尽量的少，以便后面我们可以更快的找到信息，磁盘的I/O操作也少一些，而且B类树是平衡树，每个结点到叶子结点的高度都是相同，这也保证了每个查询是稳定的。

B树和B+树的区别，以一个m阶树为例。

1. 关键字的数量不同；B+树中分支结点有m个关键字，其叶子结点也有m个，其关键字只是起到了一个索引的作用，但是B树虽然也有m个子结点，但是其只拥有m-1个关键字。
2. 存储的位置不同；B+树中的数据都存储在叶子结点上，也就是其所有叶子结点的数据组合起来就是完整的数据，但是B树的数据存储在每一个结点中，并不仅仅存储在叶子结点上。
3. 分支结点的构造不同；B+树的分支结点仅仅存储着关键字信息和儿子的指针（这里的指针指的是磁盘块的偏移量），也就是说内部结点仅仅包含着索引信息。
4. 查询不同；B树在找到具体的数值以后，则结束，而B+树则需要通过索引找到叶子结点中的数据才结束，也就是说B+树的搜索过程中走了一条从根结点到叶子结点的路径。

**二、组成原理**

**1、四门计算机专业课之间的关系**

数据结构研究的是数据和操作这些数据的算法，属于软件层面。

组成原理研究的是如何实现计算机体系结构中的属性，属于硬件层面。

操作系统是软件和硬件的交界面，是计算机最底层的软件。

计算机网络研究的是计算机与计算机之间是怎么进行互联的。

**2、组成原理和体系结构的区别**

计算机体系结构：指那些能够被程序员看到的计算机的属性。

计算机组成原理：指如何实现计算机体系结构所体现出来的属性。

以乘法指令为例，计算机是否有乘法指令，属于体系结构的问题。乘法指令是采用专用的乘法器，还是使用加法器和移位器构成，属于计算机组成的问题。

**3、冯诺依曼机的特点**

指令和数据以同等的地位存储，可按照地址访问

指令和数据都以二进制表示

存储程序思想：将指令以二进制代码的形式事先存入存储器中

以运算器为核心（现代计算机中以存储器为中心）

**4、计算机的层次结构【高汇操传微】**

高级语言机器：执行高级语言

汇编语言机器：执行汇编语言

操作系统机器：向上提供广义指令

传统机器：执行机器语言指令

微程序机器：执行微指令

**5、翻译程序**

汇编程序：汇编程序翻译成机器语言

编译程序：高级语言翻译成机器语言（一次性翻译，生成可执行代码）

解释程序：高级语言翻译成机器语言（一句一句翻译）

**6、计算机性能指标**

主频：1s内有多少个时钟周期

IPS：1s内执行多少条指令

CPI：1条指令消耗多少之中周期

FLOPS：1s内执行多少次浮点运算（最能反映计算机的计算速度）

**7、纠错与检错编码**

1.奇偶校验码

奇校验：整个校验码的1的个数为奇数

偶校验：整个校验码的1的个数为偶数

检测出奇数位错误，没有纠错能力

2.循环冗余校验码CRC

利用多项式生成校验码

检测出奇数位错误和双比特错误，选择合适的生成多项式才有纠错能力

3.海明校验码

本质是分组偶校验，多个校验位可以反应出错位置

海明码有1位纠错能力，2位检错能力

**8、原码反码补码移码**

补码：可以将减法操作变成等价的加法操作

反码：只是原码转换为补码的一个中间过程

移码：移码=真值+偏置值，可以直观地比较出数据之间的大小

范围：

原码和反码的真值0有两种形式

补码和移码的真值0只要一种形式

补码和移码相差一个符号位

转换

正数的原反补码相同

[x]补-->全部取反+1-->[-x]补

**9、定点数的运算**

移位运算

算数移位：不带符号位的移位，相当于乘2或者除2

逻辑移位：全部移动

循环移位：移出的位不丢弃

加减运算

原码加减：符号单独判断

补码加减：符号带着算[A-B]补=[A]补+[-B]补

乘法运算

原码一位乘法

补码一位乘法：booth算法

除法运算

原码除法：恢复余数法

补码除法：加减交替法

**10、溢出判断**

一位符号位：正+正=负；负+负=正时溢出

双符号位：01上溢；10：下溢

符号位进位+最好数值位进位：01上溢；10：下溢

**11、边界对齐**

以空间换时间的思路，起始地址要能够被自身长度整除

采用边界对齐后，访问一个字只需要一次访存

**12、存储方式**

大端方式：高字节存储在低地址（顺着存）

小端方式：高字节存储在高地址（逆着存）

**13、浮点数加减运算**

对阶：小阶向大阶看齐

尾数加减

规格化

左规：尾数的最高位为非有效位时进行左规

右规：尾数溢出时右规

舍入：在右规和对阶的过程中都有可能存在舍入的问题

溢出判断：

阶码上溢：抛出异常

阶码下溢：按机器0处理

**14、IEEE 754 标准**

阶码用移码表示，尾数用原码表示，隐藏最高位的1

**15、并行进位加法器**

通过进位产生函数和进位传递函数，让加法器的多个进位同时产生

**16、介绍一下计算机的多级存储系统**

主存-缓存：用来缓解速度压力，完全由硬件实现，缓存是主存的副本

主存-辅存：用来缓解容量压力，有操作系统和硬件配合实现，主存是缓存的副本

**17、半导体存储器**

半导体存储器包括RAM和ROM，其中RAM又分为SRAM和DRAM

是否易失：取决于断电之后还有存储的效果

是否破坏：取决于读出之后是否还能继续存储

SRAM：存储原理是双稳态触发器，因此是易失性的，非破坏性性读出

DRAM：存储原理是栅极电容，因此是易失性的，破坏性读出

ROM：存储原理是熔丝，因此是非易失性的，破坏性读出

其中包括EPROM,PROM等

**18、SRAM VS DRAM**

DRAM只需要两个电容结构简单，集成度高，行列地址分两次传送，因此存储成本低，发热量低，但是运行速度比较慢，常用主存

SRAM需要双稳态触发器结构复杂，集成度低，行列地址一次性传送，因此存储成本高，发热量高，但是运行速度比较快，常用cache

另外虽然两者都是易失性的，但是由于电容中存储的电量即使不断电也维持不了太久，因此对DRAM要刷新，刷新方式包括集中刷新，分散刷新，异步刷新。

**19、双端口RAM与多模块存储器**

双端口与多模块存储器都是用来增加存储器读写带宽的

双端口RAM

就是通过一个busy信号对同一个RAM 进行读写控制

多模块存储器

分为单体多字存储器

在单体多字存储器中，一次性可以读出多个字，因此会要求指令和数据连续存放，并且当只需要对单个字进行读写操作时，会产生不必要的读写时间

多体并行存储器

多体并行存储器指的是多个存储器有自己的读写控制，可以并行运行，他包括高位交叉编址，和低位交叉编址

低位交叉编址是多个存储器轮流启动，采用流水线的方式进行读取，能有效提高存储器的带宽

高位交叉编址实际上相当于扩充了存储器的容量，各个存储器之间会有忙闲不均的情况，对增强存储器的带宽没有实际作用

**20、Cache与主存映射的原理**

将主存和cache划分为等大的块，将主存中常用的数据存放在cache中，两者通过地址形成映射关系。映射方式包括全相联映射，直接相连映射，和组相联映射。

全相联映射：主存中的块可以存放在任意一个cache块中，地址划分为标记和块内地址

直接相联映射：主存中的块只能被映射到唯一一块cache中，地址划分为标记，行号，块内地址

组相联映射：主存张总的块只能被映射到某一组cache块中，但是在组内可以随意放置，地址划分为标记，组号，和块内地址

**21、指令寻址方式**

顺序寻址：PC+1

跳跃寻址方式：修改当前PC值，目标地址=（pc）+指令字长+k\*OFFSET

**22、数据寻址方式（10种）**

隐含寻址：有一个操作数隐含在ACC中

立即寻址：地址字段为操作数的补码，执行阶段不访存

直接寻址：地址字段操作数的地址，执行阶段1次访问

间接寻址：地址字段为操作数的地址的地址，执行阶段多次访存（可能1次间址也可能多次间址）

寄存器寻址：地址字段为寄存器号，寄存器中是操作数，执行阶段不访存

寄存器间接寻址：地址字段为寄存器号，寄存器中是操作数地址，执行阶段不访存，可以扩大寻址范围

相对寻址：地址码字段为相对地址，有效地址=pc+（指令长度/1B）+A

基址寻址：有效地址=基址寻址器+地址码字段

变栈寻址：有效地址=变址寻址器+地址码字段

堆栈寻址：读写地址由堆栈寄存器给出

**23、变址寻址 VS 基址寻址**

变址寻址：变址寄存器内容用户可以改变，面向程序员，适合处理数组，编制循环程序

基址寻址：基址寄存器内容用户不能改变，面向操作系统，适合处理程序浮动，利于多道程序设计

**24、CISC VS RISC**

RISC是精简指令集，CISC是复杂指令集

在CISC中，指令系统非常庞大，指令数目很多，指令字长不固定，访存指令不加限制，各个指令的执行时间也相差很大，使用频率也相差很大，有较少的寄存器，一般适用于微程序控制，可以通过一定的方式实现流水线

在RISC中，指令系统比较简单，指令数目比较少，指令字长固定，并规定只有LOAD和STORE指令可以访存，各个指令的执行时间相差不大，使用频率也相差不大，有很多的寄存器，一般适用于组合逻辑电路控制，必须实现流水线控制方式

**25、CPU的结构和功能**

功能：【只做中间数】

指令控制

操作控制

中断处理

时间控制

数据加工

结构：

cpu由运算器和控制器构成。运算器中有累加器ACC，乘商寄存器MQ，算数逻辑单元ALU，以及暂存寄存器X。控制器中有CU和指令寄存器IR以及PC。

MAR，MDR在逻辑上属于主存，但实际上是放在CPU中的控制器中的

**26、指令周期 VS 机器周期 VS 节拍**

指令周期：从主存取出一条指令并执行的全部时间

机器周期：一般把总线事务访问一次主存的时间定为机器周期

指令周期中有若干机器周期，机器周期中有若干节拍

**27、机器字长 VS 存储字长 VS 指令字长**

机器字长：计算机能直接处理的二进制数据位数，为内部寄存器字长等长

指令字长：一条指令的二进制位数

存储字长：一个存储单元所能存储的二进制的位数

三者都必须是字节的整数倍，彼此之间没有直接的制约关系

**28、一条指令的执行过程**

取指周期--FE--fetch：访问主存取指令分析指令

间址周期--IND--indirect：访问主存取操作数的有效地址

执行周期--EX--execute：访问主存取操作数

中断周期--INT-interrupt：访问主存保存程序断点

**29、主存储器 VS 控制存储器**

主存储器在cpu内部，ram实现，存放数据和程序

控制存储器在cpu内部，rom实现，存放微程序

**30、微程序 VS 程序**

程序是指令的集合，对程序员不透明

微程序是微指令的集合，对程序员透明

一条指令对应一个微程序，一个微程序由多个微指令构成，一条微指令对应多个微命令

**31、微程序控制器的基本结构**

指令送到指令寄存器后，取出操作码op送入微地址形成部件，产生初始微地址和后继微地址.

CMAR是微地址寄存器，接收微地址，为在CM中去读微指令做准备

CM中存储了各个指令对应的微程序

CMDR用于存储从CM中取出的微指令，操作码字段送到CPU内部，下地址字段送到顺序逻辑。

**32、硬布线控制器 VS 微程序控制器**

1.硬布线控制器是用组合逻辑电路来实现控制器中的CU，微程序控制器是用存储程序的思想来实现CU

2.硬布线的执行速度块，微程序控制器执行速度慢

3.硬布线繁琐，不规整，微程序控制器较规整

4.硬布线应用于RISC，微程序控制器应用于CISC

5.硬布线不易扩充，微程序控制器易于扩充

**33、影响指令流水线的因素**

结构冲突：多条指令在同一时刻争用统一资源（比如取址与取数据争用内存）

数据冲突：按序发射，按序输出的流水线中，只可能有RAW（写后读冲突）

控制冲突：遇到转移指令，或者其他改变PC的指令（指令流水线处理的必须是连续的任务）

**34、流水线改进**

超标量流水线：每个时钟周期并发多条独立的指令，配置多个功能部件

超流水线：一个时钟周期内在分段，允许一个部件在一个时钟周期内多次使用

超长指令字：多个操作码字段，多个部件并行执行

**35、增强总线效能的办法**

猝发传输：读写一组连续的字时，只需要给出首地址即可

总线复用：信号线在不同的时间传输不同的信息

**36、系统总线结构**

单总线：

cpu和外设之间可以之间进行信息交换，无需中间设备干预；cpu与外设之间速度不匹配

三总线：

主存总线+io总线+DMA总线

DMA总线在内存和高速外设之间传输数据，系统吞吐量提高，但是系统工作效率低

双总线：

主存总线+io总线

将低速设备从单总线上分离出来，但是增加通道等硬件设备

**37、总线仲裁**

解决多个主设备同时竞争总线控制权的问题

集中仲裁方式：将所有的总线请求集中起来，利用特定的算法进行总裁

分步仲裁方式：不需要总仲裁器，每个潜在的主模块都有自己的仲裁号和仲裁器

**38、总线定时**

总线双方在交换数据过程中需要时间上的配合，这一控制叫做总线定时

同步定时方式：使用同一的时钟

异步定时方式：双方通过握手信号来实现控制

**39、输入输出设备**

磁盘冗余阵列RAID

将多个磁盘组成一个独立的逻辑盘，数据在多个物理盘分割交叉存储，并行访问。可以使得数据存储更加安全，更加可靠

RAID0：类似低位交叉编址，没有容错能力

RAID1：镜像存储，粗暴地存储两份数据

RAID2：采用纠错的海明码磁盘阵列

等级越高，可靠度越高

**三、操作系统**

**1.操作系统的组成**

1、驱动程序是最底层的、直接控制和监视各类硬件的部分，它们的职责是隐藏硬件的具体细节，并向其他部分提供一个抽象的、通用的接口。

2、内核是操作系统之最内核部分，通常运行在最高特权级，负责提供基础性、结构性的功能。

3、支承库是一系列特殊的程序库，它们职责在于把系统所提供的基本服务包装成应用程序所能够使用的编程接口（API），是最靠近应用程序的部分。例如，GNU C运行期库就属于此类，它把各种操作系统的内部编程接口包装成ANSI C和POSIX编程接口的形式。

4、外围是指操作系统中除以上三类以外的所有其他部分，通常是用于提供特定高级服务的部件。例如，在微内核结构中，大部分系统服务，以及UNIX/Linux中各种守护进程都通常被划归此列。

**2.操作系统中的缓存**

缓存（cache），原始意义是指访问速度比一般随机存取存储器（RAM）快的一种高速存储器，可以进行高速数据交换的存储器，它先于内存与CPU交换数据。

**3.进程和线程**

定义：进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

（1）进程是对运行时程序的封装,是系统进行资源调度和分配的基本单位,实现操作系统的并发。

（2）线程是进程的子任务,是CPU调度和分派的基本单位,用于保证程序的实时性,实现进程内部的并发。

（3）一个程序至少有一个进程,一个进程至少有一个线程,线程依赖进程的存在。

（4）进程执行过程中拥有独立的内存单元,而多个线程共享进程的内存。

**4.进程间的通信的几种方式**

管道（pipe）及命名管道（named pipe）： 管道可用于具有亲缘关系的父子进程间的通信，有名管道除了具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

信号（signal）： 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；

消息队列： 消息队列是消息的链接表，它克服了上两种通信方式中信号量有限的缺点，具有写权限得进程可以按照一定得规则向消息队列中添加新信息；对消息队列有读权限得进程则可以从消息队列中读取信息；

共享内存： 可以说这是最有用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据得更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等；

信号量： 主要作为进程之间及同一种进程的不同线程之间得同步和互斥手段；

套接字： 这是一种更为一般得进程间通信机制，它可用于网络中不同机器之间的进程间通信，应用非常广泛。

几种方式的比较：

管道：速度慢、容量有限

消息队列：容量收到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题。

信号量：不能传递复杂信息，只能用来同步。

共享内存：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全。

**5.线程间通信**

(1)同步

多个线程通过synchronized通讯，类似于共享内存

(2)while轮询

线程A不断改变条件，线程B不断查看条件是否满足需求（比方说=5），从而实现通讯。

效率不高，因为B一直在查看，没做别的

(3)wait/notify

进入阻塞，而不是像轮询一样一直占用CPU资源

(4)管道通信

通过管道，将一个线程的消息发送个另一个线程

**6.什么是死锁？死锁产生的条件？**

1). 死锁的概念

在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲，就是两个或多个进程无限期的阻塞、相互等待的一种状态。

2). 死锁产生的四个必要条件

互斥：至少有一个资源必须属于非共享模式，即一次只能被一个进程使用；若其他申请使用该资源，那么申请进程必须等到该资源被释放为止；

占有并等待：一个进程必须占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有；

非抢占：进程不能被抢占，即资源只能被进程在完成任务后自愿释放

循环等待：若干进程之间形成一种头尾相接的环形等待资源关系

3). 死锁的处理基本策略和常用方法

解决死锁的基本方法主要有 预防死锁、避免死锁、检测死锁、解除死锁 、鸵鸟策略 等。

(1). 死锁预防

死锁预防的基本思想是 只要确保死锁发生的四个必要条件中至少有一个不成立，就能预防死锁的发生，具体方法包括：

打破互斥条件：允许进程同时访问某些资源。但是，有些资源是不能被多个进程所共享的，这是由资源本身属性所决定的，因此，这种办法通常并无实用价值。

打破占有并等待条件：可以实行资源预先分配策略(进程在运行前一次性向系统申请它所需要的全部资源，若所需全部资源得不到满足，则不分配任何资源，此进程暂不运行；只有当系统能满足当前进程所需的全部资源时，才一次性将所申请资源全部分配给该线程)或者只允许进程在没有占用资源时才可以申请资源（一个进程可申请一些资源并使用它们，但是在当前进程申请更多资源之前，它必须全部释放当前所占有的资源）。但是这种策略也存在一些缺点：在很多情况下，无法预知一个进程执行前所需的全部资源，因为进程是动态执行的，不可预知的；同时，会降低资源利用率，导致降低了进程的并发性。

打破非抢占条件：允许进程强行从占有者哪里夺取某些资源。也就是说，但一个进程占有了一部分资源，在其申请新的资源且得不到满足时，它必须释放所有占有的资源以便让其它线程使用。这种预防死锁的方式实现起来困难，会降低系统性能。

打破循环等待条件：实行资源有序分配策略。对所有资源排序编号，所有进程对资源的请求必须严格按资源序号递增的顺序提出，即只有占用了小号资源才能申请大号资源，这样就不回产生环路，预防死锁的发生。

(2). 死锁避免的基本思想

死锁避免的基本思想是动态地检测资源分配状态，以确保循环等待条件不成立，从而确保系统处于安全状态。所谓安全状态是指：如果系统能按某个顺序为每个进程分配资源（不超过其最大值），那么系统状态是安全的，换句话说就是，如果存在一个安全序列，那么系统处于安全状态。资源分配图算法和银行家算法是两种经典的死锁避免的算法，其可以确保系统始终处于安全状态。其中，资源分配图算法应用场景为每种资源类型只有一个实例(申请边，分配边，需求边，不形成环才允许分配)，而银行家算法应用于每种资源类型可以有多个实例的场景。

(3). 死锁解除

死锁解除的常用两种方法为进程终止和资源抢占。所谓进程终止是指简单地终止一个或多个进程以打破循环等待，包括两种方式：终止所有死锁进程和一次只终止一个进程直到取消死锁循环为止；所谓资源抢占是指从一个或多个死锁进程那里抢占一个或多个资源，此时必须考虑三个问题：

(I). 选择一个牺牲品

(II). 回滚：回滚到安全状态

(III). 饥饿（在代价因素中加上回滚次数，回滚的越多则越不可能继续被作为牺牲品，避免一个进程总是被回滚）

**7.进程有哪几种状态？**

就绪状态：进程已获得除处理机以外的所需资源，等待分配处理机资源；

运行状态：占用处理机资源运行，处于此状态的进程数小于等于CPU数；

阻塞状态： 进程等待某种条件，在条件满足之前无法执行；

**8.线程有几种状态？**

1. 新建(NEW)：新创建了一个线程对象。

2. 可运行(RUNNABLE)：线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取cpu 的使用权 。

3. 运行(RUNNING)：可运行状态(runnable)的线程获得了cpu 时间片（timeslice） ，执行程序代码。

4. 阻塞(BLOCKED)：阻塞状态是指线程因为某种原因放弃了cpu 使用权，也即让出了cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有机会再次获得cpu timeslice 转到运行(running)状态。阻塞的情况分三种：

(一). 等待阻塞：运行(running)的线程执行o.wait()方法，JVM会把该线程放入等待队列(waitting queue)中。

(二). 同步阻塞：运行(running)的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池(lock pool)中。

(三). 其他阻塞：运行(running)的线程执行Thread.sleep(long ms)或t.join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入可运行(runnable)状态。

5. 死亡(DEAD)：线程run()、main() 方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期。死亡的线程不可再次复生。

在给定的时间点上，一个线程只能处于一种状态。

**9.分页和分段有什么区别（内存管理）？**

段式存储管理是一种符合用户视角的内存分配管理方案。在段式存储管理中，将程序的地址空间划分为若干段（segment），如代码段，数据段，堆栈段；这样每个进程有一个二维地址空间，相互独立，互不干扰。段式管理的优点是：没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）

页式存储管理方案是一种用户视角内存与物理内存相分离的内存分配管理方案。在页式存储管理中，将程序的逻辑地址划分为固定大小的页（page），而物理内存划分为同样大小的帧，程序加载时，可以将任意一页放入内存中任意一个帧，这些帧不必连续，从而实现了离散分离。页式存储管理的优点是：没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）。

两者的不同点：

目的不同：分页是由于系统管理的需要而不是用户的需要，它是信息的物理单位；分段的目的是为了能更好地满足用户的需要，它是信息的逻辑单位，它含有一组其意义相对完整的信息；

大小不同：页的大小固定且由系统决定，而段的长度却不固定，由其所完成的功能决定；

地址空间不同： 段向用户提供二维地址空间；页向用户提供的是一维地址空间；

信息共享：段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享受到限制；

内存碎片：页式存储管理的优点是没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）；而段式管理的优点是没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）。

**10.操作系统中进程调度策略有哪几种？**

FCFS(先来先服务，队列实现，非抢占的)：先请求CPU的进程先分配到CPU

SJF(最短作业优先调度算法)：平均等待时间最短，但难以知道下一个CPU区间长度

优先级调度算法(可以是抢占的，也可以是非抢占的)：优先级越高越先分配到CPU，相同优先级先到先服务，存在的主要问题是：低优先级进程无穷等待CPU，会导致无穷阻塞或饥饿；解决方案：老化

时间片轮转调度算法(可抢占的)：队列中没有进程被分配超过一个时间片的CPU时间，除非它是唯一可运行的进程。如果进程的CPU区间超过了一个时间片，那么该进程就被抢占并放回就绪队列。

多级队列调度算法：将就绪队列分成多个独立的队列，每个队列都有自己的调度算法，队列之间采用固定优先级抢占调度。其中，一个进程根据自身属性被永久地分配到一个队列中。

多级反馈队列调度算法：与多级队列调度算法相比，其允许进程在队列之间移动：若进程使用过多CPU时间，那么它会被转移到更低的优先级队列；在较低优先级队列等待时间过长的进程会被转移到更高优先级队列，以防止饥饿发生。

**11.进程同步与互斥**

互斥：指某一个资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的

同步：是指在互斥的基础上（大多数情况下），通过其它机制实现访问者对资源的有序访问。大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。

同步：体现的是一种协作性。互斥：体现的是排它性。

**12.进程同步有哪几种机制:**

1.信号量机制

一个信号量只能置一次初值，以后只能对之进行p操作或v操作。 由此也可以看到，信号量机制必须有公共内存，不能用于分布式操作系统，这是它最大的弱点。

2.自旋锁

旋锁是为了保护共享资源提出的一种锁机制。 调用者申请的资源如果被占用，即自旋锁被已经被别的执行单元保持，则调用者一直循环在那里看是否该自旋锁的保持着已经释放了锁，自旋锁是一种比较低级的保护数据结构和代码片段的原始方式，可能会引起以下两个问题;

（1）死锁

（2）过多地占用CPU资源

3.管程

信号量机制功能强大，但使用时对信号量的操作分散，而且难以控制，读写和维护都很困难。因此后来又提出了一种集中式同步进程——管程。其基本思想是将共享变量和对它们的操作集中在一个模块中，操作系统或并发程序就由这样的模块构成。这样模块之间联系清晰，便于维护和修改，易于保证正确性。

4.会合

进程直接进行相互作用

5.分布式系统

由于在分布式操作系统中没有公共内存，因此参数全为值参，而且不可为指针。

**13.线程同步的方式**

互斥量 Synchronized/Lock：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，所以可以保证公共资源不会被多个线程同时访问

信号量 Semphare：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量

事件(信号)，Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作

**14.什么是虚拟内存？**

1).内存的发展历程

没有内存抽象(单进程，除去操作系统所用的内存之外，全部给用户程序使用) —> 有内存抽象（多进程，进程独立的地址空间，交换技术(内存大小不可能容纳下所有并发执行的进程)

）—> 连续内存分配(固定大小分区(多道程序的程度受限)，可变分区(首次适应，最佳适应，最差适应)，碎片) —> 不连续内存分配（分段，分页，段页式，虚拟内存）

2).虚拟内存

虚拟内存允许执行进程不必完全在内存中。虚拟内存的基本思想是：每个进程拥有独立的地址空间，这个空间被分为大小相等的多个块，称为页(Page)，每个页都是一段连续的地址。这些页被映射到物理内存，但并不是所有的页都必须在内存中才能运行程序。当程序引用到一部分在物理内存中的地址空间时，由硬件立刻进行必要的映射；当程序引用到一部分不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的命令。这样，对于进程而言，逻辑上似乎有很大的内存空间，实际上其中一部分对应物理内存上的一块(称为帧，通常页和帧大小相等)，还有一些没加载在内存中的对应在硬盘上，如图5所示。

注意，请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的，通过请求实现内存与外存的信息置换。

由图可以看出，虚拟内存实际上可以比物理内存大。当访问虚拟内存时，会访问MMU（内存管理单元）去匹配对应的物理地址（比如图5的0，1，2）。如果虚拟内存的页并不存在于物理内存中（如图5的3,4），会产生缺页中断，从磁盘中取得缺的页放入内存，如果内存已满，还会根据某种算法将磁盘中的页换出。

3). 页面置换算法

FIFO先进先出算法：在操作系统中经常被用到，比如作业调度（主要实现简单，很容易想到）；

LRU（Least recently use）最近最少使用算法：根据使用时间到现在的长短来判断；

LFU（Least frequently use）最少使用次数算法：根据使用次数来判断；

OPT（Optimal replacement）最优置换算法：理论的最优，理论；就是要保证置换出去的是不再被使用的页，或者是在实际内存中最晚使用的算法。

4). 虚拟内存的应用与优点

虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待它的一部分读入内存时，可以把CPU交给另一个进程使用。虚拟内存的使用可以带来以下好处：

在内存中可以保留多个进程，系统并发度提高

解除了用户与内存之间的紧密约束，进程可以比内存的全部空间还大

**15.颠簸**

颠簸本质上是指频繁的页调度行为，具体来讲，进程发生缺页中断，这时，必须置换某一页。然而，其他所有的页都在使用，它置换一个页，但又立刻再次需要这个页。因此，会不断产生缺页中断，导致整个系统的效率急剧下降，这种现象称为颠簸（抖动）。

内存颠簸的解决策略包括：

如果是因为页面替换策略失误，可以修改替换算法来解决这个问题；

如果是因为运行的程序太多，造成程序无法同时将所有频繁访问的页面调入内存，则要降低多道程序的数量；

否则，还剩下两个办法：终止该进程或增加物理内存容量。

**16.局部性原理**

(1). 时间上的局部性：最近被访问的页在不久的将来还会被访问；

(2). 空间上的局部性：内存中被访问的页周围的页也很可能被访问。

**17.中断和轮询**

中断指在计算机执行期间，系统内发生任何非寻常的或非预期的急需处理事件，使得CPU暂时中断当前正在执行的程序而转去执行相应的事件处理程序，待处理完毕后又返回原来被中断处继续执行或调度新的进程执行的过程

轮询指定时对各种设备轮流询问一遍有无处理要求

**18.临界区和冲突解决**

临界资源的定义:

一次仅允许一个进程使用的资源

临界区的定义:

每个进程中访问临界资源的那段程序

解决冲突:

如果有若干进程要求进入空闲的临界区，一次仅允许一个进程进入，任何时候，处于临界区内的进程不可多于一个。进入临界区的进程要在有限时间内退出，以便其它进程能及时进入自己的临界区。如果进程不能进入自己的临界区，则应让出CPU，避免进程出现“忙等”现象。

**19.缓冲区溢出**

缓冲区溢出的定义:指当计算机向缓冲区内填充数据时超过了缓冲区本身的容量，溢出的数据覆盖在合法数据上

缓冲区溢出的危害:程序崩溃导致拒绝服务、跳转并且执行一段恶意代码

缓冲区溢出的原因:程序中没有仔细检查用户输入的参数

**20.什么是操作系统？它有哪些功能和特征？**

答：操作系统（Operating System，简称OS）是管理和控制计算机硬件与软件资源的计算机程序，是直接运行在“裸机”上的最基本的系统软件，任何其他软件都必须在操作系统的支持下才能运行。

（1）操作系统应是计算机系统资源的管理者：①处理机管理②存储器管理③设备管理④文件管理

（2）操作系统是用户与计算机硬件系统之间的接口，同时也是计算机硬件和其他软件的接口：①命令接口②程序接口

总的来说操作系统的功能，包括管理计算机系统的硬件、软件及数据资源，控制程序运行，改善人机界面，为其它应用软件提供支持，让计算机系统所有资源最大限度地发挥作用，提供各种形式的用户界面，使用户有一个好的工作环境，为其它软件的开发提供必要的服务和相应的接口等。实际上，用户是不用接触操作系统的，操作系统管理着计算机硬件资源，同时按照应用程序的资源请求，分配资源，如划分CPU时间，内存空间的开辟，调用打印机等。

操作系统的特征：①并发：两个或者多个事件在同一时间间隔内发生。

②共享：系统中的资源可供内存中多个并发执行的进程共同使用。

③虚拟：把一个物理上的实体变为若干个逻辑上的对应物。

**四、计算机网络**

**1、Http和Https的区别**

Http协议运行在TCP之上，明文传输，客户端与服务器端都无法验证对方的身份；Https是身披SSL(Secure Socket Layer)外壳的Http，运行于SSL上，SSL运行于TCP之上，是添加了加密和认证机制的HTTP。二者之间存在如下不同：

端口不同：Http与Http使用不同的连接方式，用的端口也不一样，前者是80，后者是443；

资源消耗：和HTTP通信相比，Https通信会由于加减密处理消耗更多的CPU和内存资源；

开销：Https通信需要证书，而证书一般需要向认证机构购买；

Https的加密机制是一种共享密钥加密和公开密钥加密并用的混合加密机制。

**2、对称加密与非对称加密**

对称密钥加密是指加密和解密使用同一个密钥的方式，这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方；而非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送密文的一方使用对方的公钥进行加密处理，对方接收到加密信息后，使用自己的私钥进行解密。

由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性；但是和对称加密比起来，它非常的慢，所以我们还是要用对称加密来传送消息，但对称加密所使用的密钥我们可以通过非对称加密的方式发送出去。

**3、三次握手与四次挥手**

(1). 三次握手(我要和你建立链接，你真的要和我建立链接么，我真的要和你建立链接，成功)：

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。

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。

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。

(2). 四次挥手(我要和你断开链接；好的，断吧。我也要和你断开链接；好的，断吧)：

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。

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。此时TCP链接处于半关闭状态，即客户端已经没有要发送的数据了，但服务端若发送数据，则客户端仍要接收。

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。

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、为什么TCP链接需要三次握手，两次不可以么，为什么？**

为了防止 已失效的链接请求报文突然又传送到了服务端，因而产生错误。

客户端发出的连接请求报文并未丢失，而是在某个网络节点长时间滞留了，以致延误到链接释放以后的某个时间才到达Server。这是，Server误以为这是Client发出的一个新的链接请求，于是就向客户端发送确认数据包，同意建立链接。若不采用“三次握手”，那么只要Server发出确认数据包，新的链接就建立了。由于client此时并未发出建立链接的请求，所以其不会理睬Server的确认，也不与Server通信；而这时Server一直在等待Client的请求，这样Server就白白浪费了一定的资源。若采用“三次握手”，在这种情况下，由于Server端没有收到来自客户端的确认，则就会知道Client并没有要求建立请求，就不会建立链接。

**5、TCP协议如何来保证传输的可靠性**

TCP提供一种面向连接的、可靠的字节流服务。其中，面向连接意味着两个使用TCP的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个TCP连接。在一个TCP连接中，仅有两方进行彼此通信；而字节流服务意味着两个应用程序通过TCP链接交换8bit字节构成的字节流，TCP不在字节流中插入记录标识符。

对于可靠性，TCP通过以下方式进行保证：

数据包校验：目的是检测数据在传输过程中的任何变化，若校验出包有错，则丢弃报文段并且不给出响应，这时TCP发送数据端超时后会重发数据；

对失序数据包重排序：既然TCP报文段作为IP数据报来传输，而IP数据报的到达可能会失序，因此TCP报文段的到达也可能会失序。TCP将对失序数据进行重新排序，然后才交给应用层；

丢弃重复数据：对于重复数据，能够丢弃重复数据；

应答机制：当TCP收到发自TCP连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒；

超时重发：当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段；

流量控制：TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据，这可以防止较快主机致使较慢主机的缓冲区溢出，这就是流量控制。TCP使用的流量控制协议是可变大小的滑动窗口协议。

**6、客户端不断进行请求链接会怎样？DDos(Distributed Denial of Service)攻击？**

服务器端会为每个请求创建一个链接，并向其发送确认报文，然后等待客户端进行确认

1)、DDos 攻击

客户端向服务端发送请求链接数据包

服务端向客户端发送确认数据包

客户端不向服务端发送确认数据包，服务器一直等待来自客户端的确认

2)、DDos 预防 ( 没有彻底根治的办法，除非不使用TCP )

限制同时打开SYN半链接的数目

缩短SYN半链接的Time out 时间

关闭不必要的服务

**7、Get与POST的区别**

GET与POST是我们常用的两种HTTP Method，二者之间的区别主要包括如下五个方面：

(1). 从功能上讲，GET一般用来从服务器上获取资源，POST一般用来更新服务器上的资源；

(2). 从REST服务角度上说，GET是幂等的，即读取同一个资源，总是得到相同的数据，而POST不是幂等的，因为每次请求对资源的改变并不是相同的；进一步地，GET不会改变服务器上的资源，而POST会对服务器资源进行改变；

(3). 从请求参数形式上看，GET请求的数据会附在URL之后，即将请求数据放置在HTTP报文的 请求头 中，以?分割URL和传输数据，参数之间以&相连。特别地，如果数据是英文字母/数字，原样发送；否则，会将其编码为 application/x-www-form-urlencoded MIME 字符串(如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如：%E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII)；而POST请求会把提交的数据则放置在是HTTP请求报文的 请求体 中。

(4). 就安全性而言，POST的安全性要比GET的安全性高，因为GET请求提交的数据将明文出现在URL上，而且POST请求参数则被包装到请求体中，相对更安全。

(5). 从请求的大小看，GET请求的长度受限于浏览器或服务器对URL长度的限制，允许发送的数据量比较小，而POST请求则是没有大小限制的。

**8、TCP与UDP的区别**

TCP (Transmission Control Protocol)和UDP(User Datagram Protocol)协议属于传输层协议，它们之间的区别包括：

TCP是面向连接的，UDP是无连接的；

TCP是可靠的，UDP是不可靠的；

TCP只支持点对点通信，UDP支持一对一、一对多、多对一、多对多的通信模式；

TCP是面向字节流的，UDP是面向报文的；

TCP有拥塞控制机制;UDP没有拥塞控制，适合媒体通信；

TCP首部开销(20个字节)比UDP的首部开销(8个字节)要大；

**9、TCP的拥塞处理**

计算机网络中的带宽、交换结点中的缓存及处理机等都是网络的资源。在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就会变坏，这种情况就叫做拥塞。拥塞控制就是 防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。注意，拥塞控制和流量控制不同，前者是一个全局性的过程，而后者指点对点通信量的控制。拥塞控制的方法主要有以下四种：

1). 慢启动：不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小;

2). 拥塞避免：拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍，这样拥塞窗口按线性规律缓慢增长。

3). 快重传：快重传要求接收方在收到一个 失序的报文段 后就立即发出 重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。

4). 快恢复：快重传配合使用的还有快恢复算法，当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半，但是接下去并不执行慢开始算法：因为如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。

**10、从输入网址到获得页面的过程**

(1). 浏览器查询 DNS，获取域名对应的IP地址:具体过程包括浏览器搜索自身的DNS缓存、搜索操作系统的DNS缓存、读取本地的Host文件和向本地DNS服务器进行查询等。对于向本地DNS服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析(此解析具有权威性)；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个IP地址映射，完成域名解析（此解析不具有权威性）。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置发起递归查询或者迭代查询；

(2). 浏览器获得域名对应的IP地址以后，浏览器向服务器请求建立链接，发起三次握手；

(3). TCP/IP链接建立起来后，浏览器向服务器发送HTTP请求；

(4). 服务器接收到这个请求，并根据路径参数映射到特定的请求处理器进行处理，并将处理结果及相应的视图返回给浏览器；

(5). 浏览器解析并渲染视图，若遇到对js文件、css文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；

(6). 浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

**11、Session、Cookie 与 Application**

Cookie和Session都是客户端与服务器之间保持状态的解决方案，具体来说，cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

(1). Cookie及其相关API

Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie，而客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器，服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

(2). Session及其相关API

同样地，会话状态也可以保存在服务器端。客户端请求服务器，如果服务器记录该用户状态，就获取Session来保存状态，这时，如果服务器已经为此客户端创建过session，服务器就按照sessionid把这个session检索出来使用；如果客户端请求不包含sessionid，则为此客户端创建一个session并且生成一个与此session相关联的sessionid，并将这个sessionid在本次响应中返回给客户端保存。保存这个sessionid的方式可以采用 cookie机制 ，这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器；若浏览器禁用Cookie的话，可以通过 URL重写机制 将sessionid传回服务器。

(3). Session 与 Cookie 的对比

实现机制：Session的实现常常依赖于Cookie机制，通过Cookie机制回传SessionID；

大小限制：Cookie有大小限制并且浏览器对每个站点也有cookie的个数限制，Session没有大小限制，理论上只与服务器的内存大小有关；

安全性：Cookie存在安全隐患，通过拦截或本地文件找得到cookie后可以进行攻击，而Session由于保存在服务器端，相对更加安全；

服务器资源消耗：Session是保存在服务器端上会存在一段时间才会消失，如果session过多会增加服务器的压力。

Application（ServletContext）：与一个Web应用程序相对应，为应用程序提供了一个全局的状态，所有客户都可以使用该状态。

(4). Application

Application（Java Web中的ServletContext）：与一个Web应用程序相对应，为应用程序提供了一个全局的状态，所有客户都可以使用该状态。

**12、SQL 注入**

SQL注入就是通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

1). SQL注入攻击的总体思路

(1). 寻找到SQL注入的位置

(2). 判断服务器类型和后台数据库类型

(3). 针对不通的服务器和数据库特点进行SQL注入攻击

2). SQL注入攻击实例

比如，在一个登录界面，要求输入用户名和密码，可以这样输入实现免帐号登录：

3). 应对方法

(1). 参数绑定

使用预编译手段，绑定参数是最好的防SQL注入的方法。目前许多的ORM框架及JDBC等都实现了SQL预编译和参数绑定功能，攻击者的恶意SQL会被当做SQL的参数而不是SQL命令被执行。在mybatis的mapper文件中，对于传递的参数我们一般是使用#和$来获取参数值。当使用#时，变量是占位符，就是一般我们使用javajdbc的PrepareStatement时的占位符，所有可以防止sql注入；当使用$时，变量就是直接追加在sql中，一般会有sql注入问题。

(2). 使用正则表达式过滤传入的参数

**13、 XSS 攻击**

XSS是一种经常出现在web应用中的计算机安全漏洞，与SQL注入一起成为web中最主流的攻击方式。XSS是指恶意攻击者利用网站没有对用户提交数据进行转义处理或者过滤不足的缺点，进而添加一些脚本代码嵌入到web页面中去，使别的用户访问都会执行相应的嵌入代码，从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。

1). XSS攻击的危害

盗取各类用户帐号，如机器登录帐号、用户网银帐号、各类管理员帐号

控制企业数据，包括读取、篡改、添加、删除企业敏感数据的能力

盗窃企业重要的具有商业价值的资料

非法转账

强制发送电子邮件

网站挂马

控制受害者机器向其它网站发起攻击

2). 原因解析

主要原因：过于信任客户端提交的数据！

解决办法：不信任任何客户端提交的数据，只要是客户端提交的数据就应该先进行相应的过滤处理然后方可进行下一步的操作。

进一步分析细节：客户端提交的数据本来就是应用所需要的，但是恶意攻击者利用网站对客户端提交数据的信任，在数据中插入一些符号以及javascript代码，那么这些数据将会成为应用代码中的一部分了，那么攻击者就可以肆无忌惮地展开攻击啦，因此我们绝不可以信任任何客户端提交的数据！！！

3). XSS 攻击分类

(1). 反射性XSS攻击 (非持久性XSS攻击)

漏洞产生的原因是攻击者注入的数据反映在响应中。一个典型的非持久性XSS攻击包含一个带XSS攻击向量的链接(即每次攻击需要用户的点击)，例如，正常发送消息：

[http://www.test.com/message.php?send=Hello,World](https://link.zhihu.com/?target=http%3A//www.test.com/message.php%3Fsend%3DHello%2CWorld)！

接收者将会接收信息并显示Hello,World；但是，非正常发送消息：

[http://www.test.com/message.php?send=](https://link.zhihu.com/?target=http%3A//www.test.com/message.php%3Fsend%3D)<script>alert(‘foolish!’)</script>！

接收者接收消息显示的时候将会弹出警告窗口！

(2). 持久性XSS攻击 (留言板场景)

XSS攻击向量(一般指XSS攻击代码)存储在网站数据库，当一个页面被用户打开的时候执行。也就是说，每当用户使用浏览器打开指定页面时，脚本便执行。与非持久性XSS攻击相比，持久性XSS攻击危害性更大。从名字就可以了解到，持久性XSS攻击就是将攻击代码存入数据库中，然后客户端打开时就执行这些攻击代码。

例如，留言板表单中的表单域：

<input type=“text” name=“content” value=“这里是用户填写的数据”>

正常操作流程是：用户是提交相应留言信息 —— 将数据存储到数据库 —— 其他用户访问留言板，应用去数据并显示；而非正常操作流程是攻击者在value填写:

<script>alert(‘foolish!’)；</script> <!--或者html其他标签（破坏样式。。。）、一段攻击型代码-->

并将数据提交、存储到数据库中；当其他用户取出数据显示的时候，将会执行这些攻击性代码。

4). 修复漏洞方针

漏洞产生的根本原因是 太相信用户提交的数据，对用户所提交的数据过滤不足所导致的，因此解决方案也应该从这个方面入手，具体方案包括：

将重要的cookie标记为http only, 这样的话Javascript 中的document.cookie语句就不能

获取到cookie了（如果在cookie中设置了HttpOnly属性，那么通过js脚本将无法读取到cookie信息，这样能有效的防止XSS攻击）；

表单数据规定值的类型，例如：年龄应为只能为int、name只能为字母数字组合。。。。

对数据进行Html Encode 处理

过滤或移除特殊的Html标签，例如: <script>, <iframe> , < for <, > for>, " for

过滤JavaScript 事件的标签，例如 “οnclick=”, “onfocus” 等等。

需要注意的是，在有些应用中是允许html标签出现的，甚至是javascript代码出现。因此，我们在过滤数据的时候需要仔细分析哪些数据是有特殊要求（例如输出需要html代码、javascript代码拼接、或者此表单直接允许使用等等），然后区别处理！

**14、OSI网络体系结构与TCP/IP协议模型**

我们对OSI七层模型的各层进行简要的介绍：

1). 物理层

参考模型的最低层，也是OSI模型的第一层，实现了相邻计算机节点之间比特流的透明传送，并尽可能地屏蔽掉具体传输介质和物理设备的差异，使其上层(数据链路层)不必关心网络的具体传输介质。

2). 数据链路层（data link layer）

接收来自物理层的位流形式的数据，并封装成帧，传送到上一层；同样，也将来自上层的数据帧，拆装为位流形式的数据转发到物理层。这一层在物理层提供的比特流的基础上，通过差错控制、流量控制方法，使有差错的物理线路变为无差错的数据链路，即提供可靠的通过物理介质传输数据的方法。

3). 网络层

将网络地址翻译成对应的物理地址，并通过路由选择算法为分组通过通信子网选择最适当的路径。

4). 传输层（transport layer）

在源端与目的端之间提供可靠的透明数据传输，使上层服务用户不必关系通信子网的实现细节。在协议栈中，传输层位于网络层之上，传输层协议为不同主机上运行的进程提供逻辑通信，而网络层协议为不同主机提供逻辑通信。实际上，网络层可以看作是传输层的一部分，其为传输层提供服务。但对于终端系统而言，网络层对它们而言是透明的，它们知道传输层的存在，也就是说，在逻辑上它们认为是传输层为它们提供了端对端的通信，这也是分层思想的妙处。

5). 会话层（Session Layer）

会话层是OSI模型的第五层，是用户应用程序和网络之间的接口，负责在网络中的两节点之间建立、维持和终止通信。

6). 表示层（Presentation Layer）：数据的编码，压缩和解压缩，数据的加密和解密

表示层是OSI模型的第六层，它对来自应用层的命令和数据进行解释，以确保一个系统的应用层所发送的信息可以被另一个系统的应用层读取。

7). 应用层（Application layer）：为用户的应用进程提供网络通信服务

**15、TCP和UDP分别对应的常见应用层协议**

1). TCP对应的应用层协议

FTP：定义了文件传输协议，使用21端口。常说某某计算机开了FTP服务便是启动了文件传输服务。下载文件，上传主页，都要用到FTP服务。

Telnet：它是一种用于远程登陆的端口，用户可以以自己的身份远程连接到计算机上，通过这种端口可以提供一种基于DOS模式下的通信服务。如以前的BBS是-纯字符界面的，支持BBS的服务器将23端口打开，对外提供服务。

SMTP：定义了简单邮件传送协议，现在很多邮件服务器都用的是这个协议，用于发送邮件。如常见的免费邮件服务中用的就是这个邮件服务端口，所以在电子邮件设置-中常看到有这么SMTP端口设置这个栏，服务器开放的是25号端口。

POP3：它是和SMTP对应，POP3用于接收邮件。通常情况下，POP3协议所用的是110端口。也是说，只要你有相应的使用POP3协议的程序（例如Fo-xmail或Outlook），就可以不以Web方式登陆进邮箱界面，直接用邮件程序就可以收到邮件（如是163邮箱就没有必要先进入网易网站，再进入自己的邮-箱来收信）。

HTTP：从Web服务器传输超文本到本地浏览器的传送协议。

2). UDP对应的应用层协议

DNS：用于域名解析服务，将域名地址转换为IP地址。DNS用的是53号端口。

SNMP：简单网络管理协议，使用161号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。

TFTP(Trival File Transfer Protocal)：简单文件传输协议，该协议在熟知端口69上使用UDP服务。

**16、网络层的ARP协议工作原理**

网络层的ARP协议完成了IP地址与物理地址的映射。首先，每台主机都会在自己的ARP缓冲区中建立一个ARP列表，以表示IP地址和MAC地址的对应关系。当源主机需要将一个数据包要发送到目的主机时，会首先检查自己ARP列表中是否存在该IP地址对应的MAC地址：如果有，就直接将数据包发送到这个MAC地址；如果没有，就向本地网段发起一个ARP请求的广播包，查询此目的主机对应的MAC地址。此ARP请求数据包里包括源主机的IP地址、硬件地址、以及目的主机的IP地址。网络中所有的主机收到这个ARP请求后，会检查数据包中的目的IP是否和自己的IP地址一致。如果不相同就忽略此数据包；如果相同，该主机首先将发送端的MAC地址和IP地址添加到自己的ARP列表中，如果ARP表中已经存在该IP的信息，则将其覆盖，然后给源主机发送一个ARP响应数据包，告诉对方自己是它需要查找的MAC地址；源主机收到这个ARP响应数据包后，将得到的目的主机的IP地址和MAC地址添加到自己的ARP列表中，并利用此信息开始数据的传输。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

**17、IP地址的分类**

IP地址是指互联网协议地址，是IP协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。IP地址编址方案将IP地址空间划分为A、B、C、D、E五类，其中A、B、C是基本类，D、E类作为多播和保留使用，为特殊地址。

每个IP地址包括两个标识码（ID），即网络ID和主机ID。同一个物理网络上的所有主机都使用同一个网络ID，网络上的一个主机（包括网络上工作站，服务器和路由器等）有一个主机ID与其对应。A~E类地址的特点如下：

A类地址：以0开头，第一个字节范围：0~127；

B类地址：以10开头，第一个字节范围：128~191；

C类地址：以110开头，第一个字节范围：192~223；

D类地址：以1110开头，第一个字节范围为224~239；

E类地址：以1111开头，保留地址

1). A类地址：1字节的网络地址 + 3字节主机地址，网络地址的最高位必须是“0”

一个A类IP地址是指， 在IP地址的四段号码中，第一段号码为网络号码，剩下的三段号码为本地计算机的号码。如果用二进制表示IP地址的话，A类IP地址就由1字节的网络地址和3字节主机地址组成，网络地址的最高位必须是“0”。A类IP地址中网络的标识长度为8位，主机标识的长度为24位，A类网络地址数量较少，有126个网络，每个网络可以容纳主机数达1600多万台。

A类IP地址的地址范围1.0.0.0到127.255.255.255（二进制表示为：00000001 00000000 00000000 00000000 - 01111110 11111111 11111111 11111111），最后一个是广播地址。A类IP地址的子网掩码为255.0.0.0，每个网络支持的最大主机数为256的3次方-2=16777214台。

2). B类地址: 2字节的网络地址 + 2字节主机地址，网络地址的最高位必须是“10”

一个B类IP地址是指，在IP地址的四段号码中，前两段号码为网络号码。如果用二进制表示IP地址的话，B类IP地址就由2字节的网络地址和2字节主机地址组成，网络地址的最高位必须是“10”。B类IP地址中网络的标识长度为16位，主机标识的长度为16位，B类网络地址适用于中等规模的网络，有16384个网络，每个网络所能容纳的计算机数为6万多台。

B类IP地址地址范围128.0.0.0-191.255.255.255（二进制表示为：10000000 00000000 00000000 00000000—-10111111 11111111 11111111 11111111），最后一个是广播地址。B类IP地址的子网掩码为255.255.0.0，每个网络支持的最大主机数为256的2次方-2=65534台。

3). C类地址: 3字节的网络地址 + 1字节主机地址，网络地址的最高位必须是“110”

一个C类IP地址是指，在IP地址的四段号码中，前三段号码为网络号码，剩下的一段号码为本地计算机的号码。如果用二进制表示IP地址的话，C类IP地址就由3字节的网络地址和1字节主机地址组成，网络地址的最高位必须是“110”。C类IP地址中网络的标识长度为24位，主机标识的长度为8位，C类网络地址数量较多，有209万余个网络。适用于小规模的局域网络，每个网络最多只能包含254台计算机。

C类IP地址范围192.0.0.0-223.255.255.255（二进制表示为: 11000000 00000000 00000000 00000000 - 11011111 11111111 11111111 11111111）。C类IP地址的子网掩码为255.255.255.0，每个网络支持的最大主机数为256-2=254台。

4). D类地址:多播地址，用于1对多通信，最高位必须是“1110”

D类IP地址在历史上被叫做多播地址(multicast address)，即组播地址。在以太网中，多播地址命名了一组应该在这个网络中应用接收到一个分组的站点。多播地址的最高位必须是“1110”，范围从224.0.0.0到239.255.255.255。

5). E类地址:为保留地址，最高位必须是“1111”

**18、IP地址与物理地址**

物理地址是数据链路层和物理层使用的地址，IP地址是网络层和以上各层使用的地址，是一种逻辑地址，其中ARP协议用于IP地址与物理地址的对应。

**19、 常见状态码及原因短语**

HTTP请求结构： 请求方式 + 请求URI + 协议及其版本

HTTP响应结构： 状态码 + 原因短语 + 协议及其版本

1×× : 请求处理中，请求已被接受，正在处理

2×× : 请求成功，请求被成功处理

200 OK

3×× : 重定向，要完成请求必须进行进一步处理

301 : 永久性转移

302 ：暂时性转移

304 ： 已缓存

4×× : 客户端错误，请求不合法

400：Bad Request,请求有语法问题

403：拒绝请求

404：客户端所访问的页面不存在

5×× : 服务器端错误，服务器不能处理合法请求

500 ：服务器内部错误

503 ： 服务不可用，稍等