一. 网络操作系统、并行操作系统、分布式操作系统的特点和主要区别；

网络操作系统：

1）具有网络功能的操作系统，无严格定义。MS-DOS:1)网络通信能力;2)提供网络服务 .

2) 网络上各节点的主机运行自身的操作系统，它不仅要保证本机的系统进程或用户进程能简便、有效地使用网络中各种资源；同时，也为网中其它用户使用本机资源提供服务。

3) OS+网络协议.

分布式操作系统:

1) 分布式操作系统是为分布式计算机系统配置的一种操作系统。

2) 分布式OS在这种多机系统环境下，负责控制和管理以协同方式工作的各类系统资源；负责分布式进程的同步与执行，处理机间的通信、调度与分配等控制事务，自动实行全系统范围内的任务分配和负载平衡；具有高度并行性以及故障检测和重构能力。

3) 与网络操作系统比较 :

每台计算机没有各自独立的OS，用户不了解其文件存储在什么地方，也不了解其程序是由远程处理机执行的，分布式OS自动管理文件的放置；

网络OS每台计算机均有自己的OS；网络OS的用户要访问资源，用户必须了解资源的位置，用“文件传输”命令在计算机之间移动文件。

并行操作系统:

1) 并行计算是指同时对多个任务或多条指令、或对多个数据项进行处理。完成此项处理的计算机系统称为并行计算机系统，它将多个处理器（从几个到几万个不等）通过网络连接以一定的方式有序地组织起来。

2) 并行机 —> 并行操作系统 .

并行DBMS —> 并行算法 —> 并行程序设计语言及其开发环境（并行编译）.

实时操作系统：

1）支持实时系统工作的操作系统，响应时间有明确的规定：

执行效率高、快速、实时性强；

系统小，可剪裁，核心部分更小；

主要应用于实时控制领域。

不同类型操作系统之间的比较 ：（图1）

一. 网络操作系统、并行操作系统、分布式操作系统的特点和主要区别；

网络操作系统：

特征：硬件独立性；支持多用户；支持网络实用程序及其管理功能；支持多种客户端；提供目录服务；支持多种增值服务；可操作性。

分布式操作系统：

网络操作系统是运行在松散耦合硬件上的松散耦合软件；分布式操作系统是在松散耦合（即多计算机）硬件上运行紧耦合软件。

特征：1）有一个统一的全局进程间通信机制来保证每一个进程可以与任意一个其它进程进行联系。不允许在不同机器上采用不同的通信机制或者对本地通信与远程通信采用不同的通信机制。2）有一个全局保护方案。仅把访问控制列表和UNIX的保护位以及各种能力堆砌在一起是形成不了单一的系统映像的。3）所有机器上的进程管理必须相同即进程的创建、撤消、运行和停止不能因机器而异。4）所有机器上有统一的一组系统调用，并且这些调用必须适应分布式环境。5）所有机器上的文件系统也必须一样。6）除了受保护及安全性限制以外，每个文件应在任何一个地方都是可访问的。7）系统中所有的CPU必须运行相同内核。

8）需要一个全局的文件系统。9）每个内核对自己的局部资源应有较大的控制权。

二. 组播通信的特点及在分布式操作系统中的应用；

组播通信的特点

(1)以重复服务获取系统的容错性

(2)在系统中寻找一个对象如文件

(3)用于保持多副本数据的一致性

(4)多结点数据的刷新原子性和定序

分布式操作系统共享资源、加强通信、通过负载平衡提高系统的效率，扩充了系统能力。优点主要有以下几点：

  ①更经济—分布式操作系统有较高的性能价格比。

  ②速度更快—分布式操作系统平均响应时间比大型机系统短。

  ③分布式操作系统对固有分布性问题求解的适应性。

  ④可扩充性—分布式操作系统比较松散的构成，使得节点的增减很容易。

  ⑤更可靠—分布式操作系统自动降级运行保障，故障时不停机，安全更加具有保障性

⑥宽适应性—分布式操作系统增加了对分散用户要求协同的支持，满足了用户的需求

三. RPC通讯模型及主要通讯过程；

通信模型：

是基于client／server进程间相互通信模型的一种同步通信形式，它对client提供了 远程服务的过程抽象，其底层消息传递操作对client是透明的．在RPC中，client是请求服务的调用者(caller)，server是执行client的请求而被调用的程序。

Client：

（1）发送“请求服务”的消息给远程的server；

（2）等待Server回复；

（3）得到Server的回复消息后恢复继续执行

• Server方：

（1）处于监听状态，等待客户的请求服务

（2）提取调用参数，执行指定程序；

（3）将执行结果以消息形式返回给Client。

（图2）

实现过程主要有以下几步：

1.用户以本地方式调用系统产生的 Client 方的一个 Stub 过程；

2. 该 Stub 将用户提供的参数按照网络通信的要求封装成消息包，并将它们发给 Client 方的 RPC runtime；

3. RPC runtime 将它们发送到指定结点；

4. 远程 RPC runtime 接收此消息，并将此消息发送给相应Server 的 Stub；

5. Server 的 Stub 拆封消息，形成被调过程 p 要求的形式，并调用 p；

6. p 按照所获参数进行执行，并将结果返回给 Server 的Stub；

7. Server 的 Stub 将此结果封装成消息，发送给 Server 的RPC runtime，然后逐级地传送给用户。

RPC的实现要考虑两个方面的问题。第一，当进行远程过程调用时，调用场点必须能定位出被调用的过程实际上运行在哪个场点上；第二，相关的两个场点必须能协同合作交换信息。

四. NTP时间同步协议；

NTP协议被用来保证 Internet中时间服务器的时钟同步。时间服务器被链入一个逻辑分层结构的网络中。主服务器直接连接时间资源（如：UTC接收器），以主服务器为根，实现分层同步。当某个服务器发生故障时，可以进行适当替换。

NTP协议属于应用层协议，是用于在分布式时间服务器和客户端之间进行时间同步的，它定义了协议实现过程中所使用的结构、算法、实体和协议。NTP协议是基于IP和UDP的，也可以被其它协议组使用。NTP是从时间协议（TIME PROTOCOL）和ICMP 时间戳报文（ICMP TIMESTAMP MESSAGE）演变而来，主要是从准确性和强壮性方面进行了特殊的设计。

NTP的优点：

采用分层的方法来定义时钟的准确性，可以迅速同步网络中各 台设备的时间。

支持访问控制和MD5验证。

可以选择采用单播、广播或组播发送协议报文。

NTP被用作Internet上时钟同步的标准，其主要设计目标和重要特征有：

1）提供Internet上的客户精确地与UTC同步的 服务；

2）提供可靠的服务以保证连接的最小损失；

3）能使客户频繁的再次同步；

4）对时间服务提供保护。

NTP应用：

1）在备份服务器和客户机之间进行增量备份时，要求这两个系统之间的时钟必须同步。

2）确保系统之间的RPC（远程系统调用）能够正常进行。

3）有的应用程序需要知道一个用户是什么时候登录到系统的；以及一个文件的修改时间。

4）在一个网络中，系统之间的时钟相差一分钟或者更少的情况很多。如果网络很大，不可能完全依靠系统管理员手工输入date（时间设置命令）命令来调节各个系统的时钟。

5）调试与事件时间戳（timestamps）：从不同路由器采集的调试与事件时间戳是没有什么意义的，除非这些路由器是以同一公共时间为参考。

6）事件：事物处理需要精确的时间戳（timestamps）。

7）仿真：复杂的事物往往需细分，由多个系统来处理，为保证事件的正确顺序，多个系统必须参考同一时钟。

8）系统维护：完成某些功能如同时重装（reload）网络内的所有路由器，整个网络必须拥有公共时钟。

五. LAMPORT算法；

基本思想：

资源是大家的，因此，要想获得某一资源的使用权，必须征得大家的同意；面对冲突，则按“定序”进行处理。

算法：

1、当进程 p 要进入临界区时，向系统中的所有进程发送Request(Tp, p)，并将此请求放入自己的请求队列；

2、当进程q接到此消息，将其放入自己的请求队列，并返回一个带有时间戳的Reply(Tq)；[Tq>=Tp+1]；

3、当下面的条件成立时，p 进入临界区，否则 p等待：

a) Request(Tp, p)在p的队列的队首；

b) p收到全部其它进程的时间戳迟于Tp的Reply。

4、当p退出临界区时，从自己的队列中去掉 Request(Tp, p)，并向其它进程发出Release；

5、当q接到Release后，将相应的Request(Tp, p)从自己的队列中删除。然后看自己是否正在等待此临界区：

a) 如果是，则转3；

b) 如果不是，不做任何进一步的工作。

讨论：

① 每完成一次临界区操作，要发Request、Reply、Release消息各 n-1 个；

② 时间戳及全排序保证了不会死锁。

六. 资源共享和管理的主要方法；

实现资源共享的方法：

1. 数据迁移：整个文件 部分文件 通过文件或数据库的 水平分割、垂直分割 但分割较麻烦。

2. 计算迁移：传递计算比传递数据更有效。

3. 作业迁移：隐式：作业迁移最终由系统实现；

显式：用户指明作业如何迁移。

资源管理的的方式：

局部集中管理；

分散式管理；

分级式管理。

资源管理的原则 ：

方便、高效、公平 。

资源管理内容：

配置管理；故障管理；安全管理；性能管理；帐户管理。

资源管理的任务：

1）接受来自客户方（用户、进程）申请资源的请求，并从资源中选择适当的资源进行分配 。

2）接受系统提供的资源，并能组成资源池（资源库）。具有一定的监控，最终可以收回资源。

七. 招标算法

招标算法是分布式资源管理最常用的算法之一。

算法描述（6个步骤）：

1）当资源管理者向其他站点申请资源时，先将招标消息广播出去；

2）当其他资源管理者接收到招标消息后;

①若该站点有所需资源，则它根据一定方法计算出“标数”；

②然后给申请者发一条投标消息，否则回复一条拒绝投标的消息。

3) 当申请者接收到所有的投标消息后，根据策略选择一个投标者，并向其发送一条申请资源的消息;

4）对接收到请求资源消息的资源管理者，将申请者的名字排入其等待队列，并在可以分配所指资源时再发消息通知申请者。

5）申请者在使用完所需资源后，通知分配资源者回收资源。

投标与选标策略可视具体情况而定：

1). 使用等待队列中排队等待的申请者的个数作为标数来投标，选标时则选择标数最小的投标者中标；

2). 不仅考虑有多个资源申请者，还考虑到投标者与招标者之间的距离。可规定标数为： x=c1a+c2b（a为等待的申请者个数，b为投标者与招标者之间的距离，c1 ， c2为常数），选取最小的x中标，这种策略考虑到资源使用的均衡性和有效性。

若考虑站点故障而仍使该算法有效，可增加下述措施：

6）若申请消息超时未得到响应，则向中标者发一询问消息，

①若中标者未故障就立即予以回复；

②若发出询问消息后仍无回复，则申请者重新广播招标消息；

其中第6步主要是考虑站点故障时而增加的措施，同时第3步还需要考虑超时问题。

招标算法的三个特点：

①不会出现饥饿现象。因为只要系统中有所申请的资源就必有一个中标者；只要每个资源占有者在有限长时间内归还所占资源，申请者总能从中标者处获得所需资源。

②在无站点故障情况下，从广播招标消息到接到获得资源的通知，一共交换了2(n一1)+2=2n条消息。

③该算法特别适用于共享通路结构(如全互连、总线型)的分布式系统。因为只要广播消息一发出，就可向系统中的各资源管理者招标。

环形结构招标算法的描述：

1）申请资源者向其邻近站点发一招标消息。

2）接收到招标消息后，若本站点上无所指资源，则它将招标消息沿环传递给下一邻近站点，否则:

①若此消息中未附投标消息，则它将本站点的投标信息附上，并将这一新形成的消息传递给下一邻近站点；

②若此消息中已附有投标消息，则将本站点的投标消息同此消息进行比较；优选一个附上传递给下一邻近站点。

3）某站点接收到自己发出的招标消息后，从其中所附的投标信息可知中标者是谁，直接向中标的资源管理者发一申请资源的信息。

4）中标者接收到申请消息后，将申请者的名字排入其等待队列，并在可以分配所需资源时向申请者发通知。

5）当资源使用完，申请者通知分配资源者回收资源。

八. 进程等待图与集中式死锁检测方法；

资源分配图：

G＝(V，E)表示资源分配图。 其中V是顶点的集合。

包含两部分：

①所有进程的集合P＝{p1，p2，…，pn}，

②所有资源类的集合R={r1，r2，…，rm}。

E是所有有序对(pi，rj)或(rj，pi)的集合，其中pi∈P rj∈R

V是顶点的集合。

若(pi，rj)∈E，则它表示从进程pi到资源类rj有一条有向边，意指，进程pi请求分配资源类rj的一个实例，若(rj，pi)∈E，则它表示从资源类rj到进程pi有一条有向边，意指，资源类rj的一个实例已被分配给进程pi。

几个图标的解释：

进程：用圆圈表示；

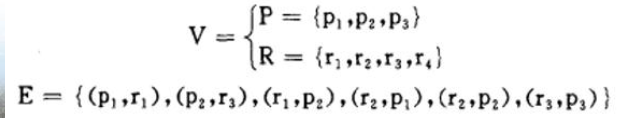
资源类：用椭圆表示；

椭圆中的“○”：表示该资源类的实例；

请求边：指向椭圆；

分配边：对应其中的“○”；

当进程pi请求分配资源类rj的一个实例时，则向资源分配图中插入一请求边(pi，rj)。从动态的观点看，当请求边被满足后，该请求边马上就变成一条分配边(rj，pi)。考虑如右图所示的资源分配图G，其中V是顶点的集合（包含进程集合P,资源集合R），E是所有有序对(pi,rj)或(rj,pi)的集合。

 （图3）

进程p1占有r2的一个实例并正申请r1的一个实例；进程p2占有r1和r2中的各一个实例，并正申请r3的一个实例；进程p3占有r3的一个实例。

给定的资源分配图G，若G中含环路，则表明系统未发生死锁；反之，若G中含有环路，则表明系统可能存在死锁。例如，若在图中插入边(p3，r2)(如虚线所示)，则在这种情形，系统可能出现死锁。（图4）

如给定的资源分配图G，若G中不含环路，则表明系统未发生死锁；反之，若G中含有环路，则表明系统可能存在死锁。例如，在图中插入边(p3，r2)(虚线)，则在这种情形，系统可能出现死锁，构成如下环路：。

若系统中每类资源仅有一个实例，则G中的环路表明已经发生死锁，此时，G中的环路是产生死锁的充要条件，如右图所示（图5）

若资源中有多个实例，在资源分配图中的环路不一定就发生死锁，即这种情况下，图中的环路死锁存在的必要而非充分条件，如：

原因：进程p3可以释放r2中的实例从而使得p2得到满足，可以断开这个环路。

进程等待图:

从资源分配图中去掉表示资源类的椭圆并合并相关的有向边便可得到对应的进程等待图(process waiting graph)。例如，与图4.3中资源分配图对应的进程等待图如图4.5所示。

进程等待图中从pi到pj的有向边(pi，pj)意指：pi等待pj释放它所需要的资源，而有向边(pi，pj)在进程等待图中存在，当且仅当对应的资源分配图中，对某个资源类rk存在两条边：(pi，rk)和(rk，pj)。

与进程等待图有关的一个结论：若系统中的每一资源类仅含一个实例，则系统发生死锁的充要条件是进程等待图中存在环路。

集中式死锁检测方法：

集中式死锁检测方法的执行过程如下：

①协调者向系统中每一站点发送一条初始消息；

②当接收到这一消息后，各站点将它的局部等待图发送给协调者。注意， 这每一个等待图包含该站点的所有局部信息。这种等待图反映了相应站点瞬时的状态，但它并不与反映任何其他站点的等待图同步。

九. 分布式进程的状态与切换；

所谓分布式进程(distributed processes)是能够真正在多个处理机同时运行的诸进程。一般的并发进程利用的是多个虚拟处理机的概念，而分布式进程利用的是多个真正的物理处理机。

分布式进程的状态与切换:

进程状态:是指进程的活动情况，而进程状态的切换就是进行进程调度。

在分布式环境下，进程的状态也有运行态、等待态、挂起态和就绪态四种。

1）运行态（running）：当进程占有处理机或其他资源并正在执行指令的状态时称为运行态。但进程在整个运行周期不一定都是在同一处理机上运行。

2）等待态（waiting）：在进程运行过程中，因等待某种事件的发生所处的一种状态称为等待态。此状态是为了在进程被挂起之前对其占用的系统资源情况进行检查，并为其可能正要进行中的访问内存操作保留一定时间而设置的。

3）挂起态（suspended）：进程进入等待态只不过是一个暂存的过渡，此后必须进入挂起态，暂时停止执行。此时，进程必须释放它所占的资源。为了防止死锁，有些系统还要求挂起的进程释放占用的内存和设备。

4）就绪态（ready）：当进程已符合运行要求，只是因为系统中当前的进程个数已超过了处理机的个数而不能进入运行时，就进入就绪态；或者是当进程运行的时间超过了系统预分的时间时，系统就强制其进入就绪态，就进入等待下次运行机会。

进程状态的切换 ：通过操作原语的控制来实现，操作原语是进程调度程序的主要组成部分 。

3个特点：

1）分布式系统是以任务级并行为特征的。因此分布式操作系统的基本调度单位不再是单机上的进程，而是在各处理机上运行着的并行进程所组成的任务队列。

2）同一任务队列的并发进程可分配到不同处理机上并行执行。同一处理机也可执行多个不同任务队列中的进程，这就使得在单机系统中许多行之有效的调度算法，如优先数法、时间片法等，都不完全适用于分布式系统。

3）寻求合理、高效的进程调度算法仍是目前分布式系统的重要研究课题之一。

十. 遗传算法

概念：

遗传算法（Genetic Algorithm, GA）起源于对生物系统所进行的计算机模拟研究。它是模仿自然界生物进化机制发展起来的随机全局搜索和优化方法，它借鉴了达尔文的进化论和孟德尔的遗传学说。

其本质是一种高效、并行、全局搜索的方法，它能在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最佳解。

优点：

1）对可行解表示的广泛性。

2）群体搜索特性。

3）不需要辅助信息。

4）内在启发式随机搜索特性。

5）遗传算法在搜索过程中不容易陷入局部最优，即使在所定义的适应度函数是不连续的、不规则的或有噪声的情况下，也能以很大的概率找到全局最优解。

6）遗传算法采用自然进化机制来表现复杂的现象，能够快速可靠地解决求解非常困难的问题。

7）遗传算法具有固有的并行性和并行计算的能力。

8）遗传算法具有可扩展性，易于同别的技术混合。

缺点：

（1）编码不规范及编码存在表示的不准确性。

（2）单一的遗传算法编码不能全面地将优化问题的约束表示出来。考虑约束的一个方法就是对不可行解采用阈值，这样，计算的时间必然增加。

（3）遗传算法通常的效率比其他传统的优化方法低。

（4）遗传算法容易出现过早收敛。

（5）遗传算法对算法的精度、可信度、计算复杂性等方面，还没有有效的定量分析方法。

基本思想：

遗传算法是依据生物进化中的“适者生存”规律的基本思想设计的，它把问题的求解过程模拟为群体的适者生存过程，通过群体的一代代的不断进化（包括竞争、繁殖和变异等）出现新群体，相当于找出问题的新解，最终收敛到“最适应环境”的个体（解），从而求得问题的最优解或满意解。

遗传算法一般由编码与解码、适应度函数、遗传算子和控制参数等四个部分组成。

编码：由设计空间向遗传算法编码空间的映射称为编码；

适应度函数：是用以描述个体适应环境的程度，也是生物进化中决定哪些染色体可以产生优良后代（适者生存）的依据；

遗传算子：包括复制（或选择）算子、交叉算子和变异算子；

控制参数：包括种群的规模M、交叉率Pc和变异率Pm。

计算步骤：

1） 确定寻优参数，进行编码。编码时先要设置编码长度；

2） 随机产生一组初始解（即个体）组成初始种群。初始种群中个体的数目称作初始种群的规模；

3） 计算种群中各个个体的目标函数值及其相应的适应度函数值；

4） 形成匹配集。根据种群中各个染色体的适应度函数值，采取一定的选择方法，从种群中选出适应值较大的个染色体（其中有些染色体是重复的），称这个染色体的集合即为匹配集。这一过程即为选择操作。

5） 按某种复制规则进行繁殖。由匹配集中的个染色体繁殖产生个新的染色体，得到一个新的种群。繁殖方法主要有两种：交叉和变异。

6）若遗传代数（迭代次数）达到给定的允许值或其它收敛条件已满足时停止遗传，否则返回步骤3）。

简单遗传算法的运行过程为一个典型的迭代过程，其必须完成的工作内容和基本步骤如下：

1）选择编码策略，把参数集合X和域转换为位串结构空间S。

2）定义适应度函数 。

3）确定遗传策略，包括选择群体大小n，选择、交叉、变异方法，以及确定交叉概率 、变异概率等遗传参数。

4）随机初始化生成种群P。

5）计算群体中个体位串解码后的适应度值 。

6）按照遗传策略，运用选择、交叉和变异算子作用与群体，形成下一代群体。

7）判断群体性能是否满足某一目标，或者已完成预定迭代次数，不满足则返回步骤6），或者修改遗传策略再返回步骤6）。

遗传算法的终止条件的主要判据有以下几种：

1） 判别遗传算法进化代数是否达到预定的最大代数；

2） 判别遗传搜索是否已找到某个较优的染色体；

3） 判别各染色体的适应度函数值是否已趋于稳定、再上升否等。

遗传算法遗传操作包括复制（选择）、交叉和变异等三个基本遗传算子：

遗传算法中的复制操作就是用来确定如何从父代群体中按某种方法选取哪些个体遗传到下一代群体中的一种遗传运算。为了进行选择操作，需要对种群中的每个个体设定一个复制概率，使该个体被复制的可能性与其在种群中的适应性成比例，即适应性强的个体被选择复制的可能性越大。这里介绍两种设定个体复制概率的常用方法。

常见的选择操作主要有以下几种：1）轮盘赌选择；2）排序选择；3）最优个体保存；4）随机联赛选择；

交叉算子：

交叉是指对两个相互交叉的染色体按某种方式相互交换其部分基因，从而形成两个新的个体。它是产生新个体的主要方法，决定了遗传算法的全局搜索能力，在遗传算法中起关键作用。

几种常用的适用于二进制编码或实数编码方式的交叉算子如下：

1）单点交叉；2）两点交叉；3）均匀交叉；4）算术交叉。

变异算子;

指将个体染色体编码串中的某些基因座上的基因值用该基因座的其他等位基因来替换，从而形成一个新的个体。

常见变异操作：

1）基本位变异；2）均匀变异；3）二元变异；4）高斯变异。

例：利用遗传算法，求解区间[0，31]上的二次函数f (x) =x2的最大值。

分析：原问题可转化为在区间［0，31］中搜索能使y取最小值的点x的问题。那么，［0，31］中的点x就是个体，函数值f(x)恰好就可以作为x的适应度，区间［0，31］就是一个（解）空间。这样，只要能给出个体x的适当染色体编码，该问题就可以用遗传算法来解决。二次函数的图像如图所示。

（1）设定种群规模，编码染色体，产生初始种群。

将种群规模设定为4；用5位二进制数编码染色体；取下列个体组成初始种群 ：

s1 = 13（01101）， s2= 24（11000）

s3= 8 （01000）， s4= 19（10011）

（2）定义适应度函数 ：取适应度函数：f (x) =x2。

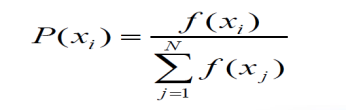
（3）计算各代种群中的各个体的适应度，并对其染色体进行遗传操作，直到适应度最高的个体（即31（11111））出现为止。

首先计算种群S1中各个体的适应度 。容易求得，

f (s1) = f(13) = 132 = 169 f (s2) = f(24) = 242 = 576

f (s3) = f(8) = 82 = 64 f (s4) = f(19) = 192 = 361

再计算种群 中各个体的选择概率。选择概率的计算公式为：



由此求得， P(s1) = P(13) = 0.14 P(s2) = P(24) = 0.49

P(s3) = P(8) = 0.06 P(s4) = P(19) = 0.31

于是，经复制得群体：

s1’ =11000（24）， s2’ =01101（13）

s3’ =11000（24）， s4’ =10011（19）

设交叉率pc=100%，即S1中的全体染色体都参加交叉运算。设s1’与s2’配对，s3’与s4’配对。分别交换后两位基因，得新染色体：

s1’’=11001（25）， s2’’=01100（12）

s3’’=11011（27）， s4’’=10000（16）

设变异率pm=0.001。这样，群体S1中共有5×4×0.001=0.02位基因可以变异。0.02位显然不足1位，所以本轮遗传操作不做变异。于是，得到第二代种群S2：

s1=11001（25）， s2=01100（12）

s3=11011（27）， s4=10000（16）

假设这一轮选择-复制操作中，种群 中的4个染色体都被选中，则得到群体：

s1’=11001（25）， s2’= 01100（12）

s3’=11011（27）， s4’= 10000（16）

做交叉运算，让s1’与s2’，s3’与s4’ 分别交换后三位基因，得

s1’’ =11100（28）， s2’’ = 01001（9）

s3’’ =11000（24）， s4’’ = 10011（19）

这一轮仍然不会发生变异。于是，得第三代种群S3：

s1=11100（28），s2=01001（9）

s3=11000（24）， s4=10011（19）

设这一轮的选择-复制结果为：

s1’=11100（28）， s2’=11100（28）

s3’=11000（24）， s4’=10011（19）

做交叉运算，让s1’与s4’，s2’与s3’ 分别交换后两位基因，得

s1’’=11111（31）， s2’’=11100（28）

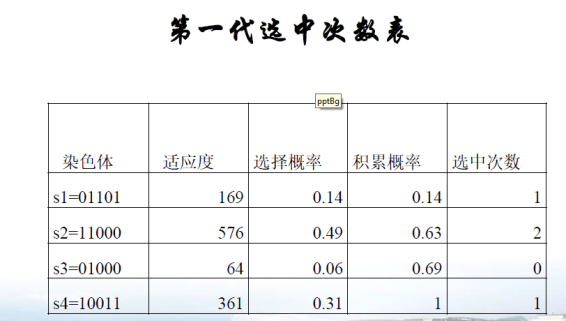
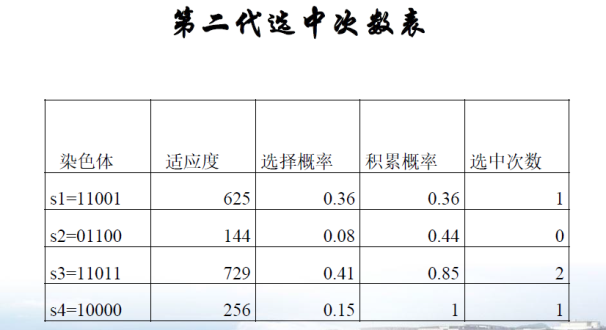
s3’’=11000（24）， s4’’=10000（16）

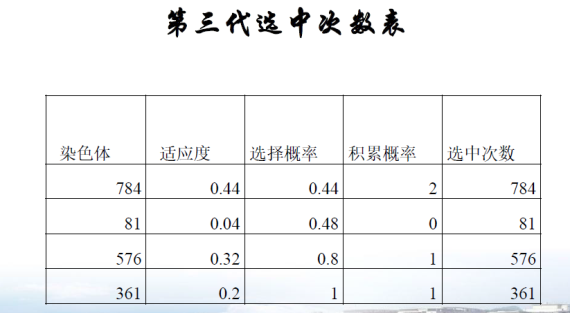
这一轮仍然不会发生变异。于是，得第四代种群S4：

s1=11111（31）， s2=11100（28）

s3=11000（24）， s4=10000（16）

显然，在这一代种群中已经出现了适应度最高的染色体s1=11111。于是，遗传操作终止，将染色体“11111”作为最终结果输出。然后，将染色体“11111”解码为表现型，即得所求的最优解：31。将31代入函数y=x2中，即得原问题的解，即函数y=x2的最大值为961。



十一. 动态负载平衡策略的四个组成部分；

负载平衡也称负载共享，是指对系统中的负载情况进行动态调整，以尽量消除或减少系统中各场点负载不均匀的现象．具体实现方法是将重载场点上的任务转移到其他轻载场点上，尽可能实现系统中各场点的负载平衡，从而提高系统的吞吐量． 负载共享有利于统筹管理分布式系统中的各种资源，便于利用共享信息及其服务机制扩大局部分布式系统的处理能力。

动态负载平衡策略包含四个部分：

转移策略，选择策略，定位策略和信息策略。

1）转移策略：决定一个节点是否处于合适的状态来参与任务转移。参与转移时，要么是发送者，要么是接收者。一个普遍认可的转移策略是门槛(threshold)策略：

①当某个节点产生一个新任务时，当节点负载超过T1时，转移策略将该节点确定为发送者；

②当节点负载小于T2时，转移策略将它确定为远程任务的接收者，这里T1是发送门槛值，T2是接收门槛值，T1>T2；

③另一个转移策略是相关转移策略，例如，·若一个节点的负载在全系统中最低，则被确定为接收者．

2）选择策略：

一旦转移策略决定了一个节点是发送者．选择策略就要从本地选择一个任务来转移，如果选择策略不能发现一个合适的任务来转移，这个节点就不再被认为是发送者。最简单的办法是选一个导致节点成为发送者的新近产生的任务，这种任务转移开销较小。

选择策略要考虑以下几个因素：

①转移的额外开销尽量小；

②被选择的任务应该足够大，以值得花去额外开销。

3）定位策略：

定位策略的作用是为准备转移的负载选择合适的“转移伙伴”。

在非集中式算法中一个广泛采用的方法是轮转搜寻，即由一个节点询问其他每个节点以确定是否可以共享负载．轮转搜寻可以采取串行和并行两种方式(并行即广播方式)。

在集中式算法中，一个指定的节点作为协调者来定位、收集系统信息，发送者从协调者处获得系统信息来选择接收者。

4）信息策略：

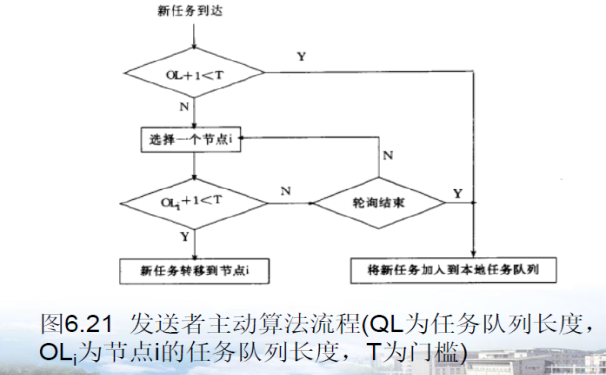
信息策略决定系统中其他节点的状态信息如何收集、从哪里收集、收集哪些信息等等．有三种类型的信息策略：

①要求驱动策略：在非集中式策略中，一个节点只有在成为发送者或接收者时才能收集其他节点的信息；

②周期策略：定期地收集信息。可用于集中式算法，也可用于非集中式算法之中；

③状态改变驱动策略：在自身状态改变到一定程度时，节点向外发布它的状态信息。

十二. 发送者主动算法；

在发送者主动算法中，当一个节点超载时，它就尝试将任务发送给一个轻载节点(接收者)．图6.21给出了算法流程．我们从算法的四个组成策略及稳定性加以说明． 

算法的组成策略：

1. 转移策略：这里采用的是门槛策略，当一个节点产生一个新任务时，若任务队列长度超过门槛T，则该节点被确定为发送者．

2. 选择策略：本算法选择新到达的任务作为要发送的任务．

3. 定位策略：根据被轮询节点的任务队列长度，判断是否超过门槛值，若没有超过就转移到该节点，这是门槛定位策略．

4. 信息策略：在本算法中采用的信息策略是要求驱动策略，一个节点只有在成为发送者时才能收集其他节点的信息．

十三. X86平台虚拟化的三种方式；

1）全虚拟化：

客户操作系统运行在Ring 1级，VMM运行在Ring 0级，对于不能虚拟化的特权指令， 通过二进制转换方式转换为同等效果的指令序列运行，而用户级指令可直接运行。

不需要修改操作系统，虚拟机具有较好的隔离性和安全性。

2）半虚拟化：

需要修改操作系统内核，将不能虚拟化的指令替换为hypercall，hypercall直接与虚拟层通信；

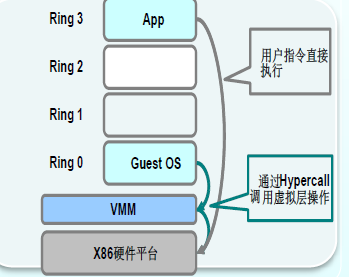
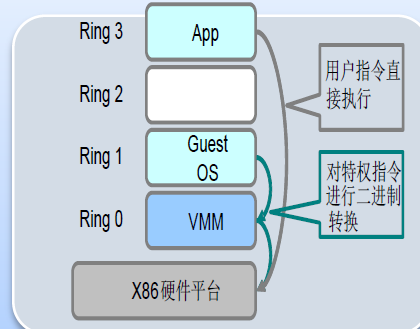
显著减少了虚拟化开销，性能较高，但是由于需要修改操作系统内核，对于非开放的操作系统，如windows 2000/xp，则无法支持。

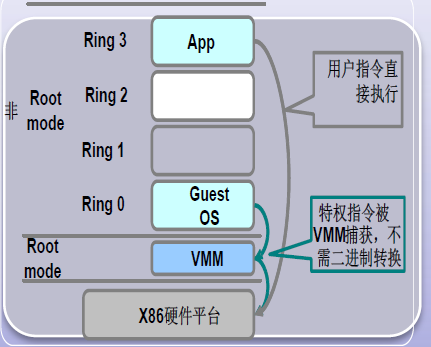
3）硬件辅助虚拟化：

CPU在Ring 0级之下还提供了一个Root Mode，VMM运行在Root Mode下。特权指令自动被VMM捕获，不需要进行二进制转换或调用Hypercall；

效率较高，无需修改操作系统。

ZTE最新的刀片支持硬件辅助虚拟化。



全虚拟化：通常是指其上运行的客户OS不需要作任何修改，但客户OS所要求的CPU指令集必须为物理CPU的指令集所兼容。这种模型使用一个虚拟机在客户操作系统和原始硬件之间进行协调。VMM 在客户操作系统和裸硬件之间提供协作，特权指令必须被捕获下来并在 hypervisor 中进行处理，这是因为底层硬件并不由操作系统拥有，而是由操作系统通过 hypervisor 共享的。

半虚拟化通常是指其上运行的客户OS需要作少量的修改，与VM共同协作完成虚拟机的工作。半虚拟化与全虚拟化有一些类似。它同样使用了一个 hypervisor 来实现对底层硬件的共享访问，不同之处在于它将与虚拟化有关的代码集成到了操作系统本身中。

十四. CPU、内存、I/O设备虚拟化的主要任务；、

CPU虚拟化技术的目标：

1）能够尽可能快的处理一般指令 ；

2）特权指令转交给模拟例程 。

是一种硬件方案，支持虚拟技术的CPU带有特别优化过的指令集来控制虚拟过程，通过这些指令集，VMM会很容易提高性能，相比软件的虚拟实现方式会很大程度上提高性能。虚拟化技术可提供基于芯片的功能，借助兼容VMM软件能够改进纯软件解决方案。由于虚拟化硬件可提供全新的架构，支持操作系统直接在上面运行，从而无需进行二进制转换，减少了相关的性能开销，极大简化了VMM设计，进而使VMM能够按通用标准进行编写，性能更加强大。CPU的虚拟化技术除支持广泛的传统操作系统之外，还支持64位客户操作系统。

内存虚拟化主要任务：

VMM维护一个虚拟机内存管理数据结构——镜像页表(shadow page table)。VMM通过镜像页表给不同的虚拟机分配机器的内存页，如操作系统虚拟内存一样，VMM能将虚拟机内存换页到磁盘，因此，虚拟机申请的内存可以超过机器的物理内存。VMM也可以根据每个虚拟机的要求,动态地分配相应的内存。

3）I/O设备虚拟化：

1. 宿主型I/O虚拟化：用宿主型的体系结构，使用宿主操作系统的L／0设备驱动程序。这种结构的缺点之一是大大增加了虚拟化的性能开销；另一个缺点是现代操作系统如Windows和linux并没有资源管理的支持为虚拟机提供性能隔离和服务保证，而这是很多服务器环境的基本要求。

2. 硬件I/O虚拟化：I/O子系统的产业趋势是朝着硬件支持的方向发展。拥有足够的硬件支持，直接传送I/O设备到虚拟机中的软件是完全可能的，这将有效地消除所有I/O虚拟开销。要做到这一点， I/O装置需要了解虚拟机的情况和能够支持多个虚拟接口，以便VMM能安全地映射接口到虚拟机。

十五. 什么是IaaS,PaaS,SaaS？

云计算模式：

1）软件即服务（SaaS）；

2）平台即服务（PaaS）；

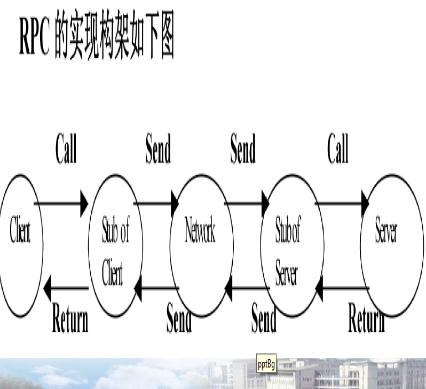
3）基础设施即服务（LaaS）。

SaaS: 提供给客户的服务是服务商运行在云计算基础设施上的应用程序，可以在各种客户端设备上通过瘦客户端界面访问，比如浏览器。客户不需要管理或控制的底层的云计算基础设施，包括网络、服务器、操作系统、存储，甚至单个应用程序的功能。

SaaS是Software-as-a-service（软件即服务）的简称, 它是一种通过Internet提供软件的模式，通常的定义为“软件部署为托管，通过互联网存取”，相对于传统软件模式，SaaS用户不用再购买软件，而向SaaS服务提供商租用基于Web的软件。

SaaS提供商为企业搭建信息化所需要的所有网络基础设施及软件、硬件运作平台，并负责所有前期的实施、后期的维护等一系列服务，企业无需购买软硬件、建设机房、招聘IT人员，即可通过互联网使用信息系统。

SaaS是一种软件布局模型，其应用专为网络交付而设计，便于用户通过互联网托管、部署及接入。 SaaS 应用软件的价格通常为“全包”费用，囊括了通常的应用软件许可证费、软件维护费以及技术支持费，将其统一为每个用户的月度租用费。saas对于广大中小型企业来说，SaaS是采用先进技术实施信息化的最好途径。但SaaS绝不仅仅适用于中小型企业，所有规模的企业都可以从SaaS中获利。

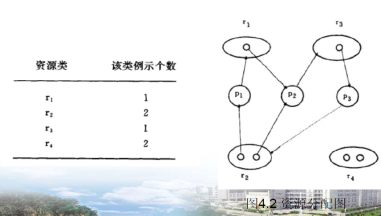
 PaaS: 是指将软件研发的平台作为一种服务提供给客户的， PaaS能将现有各种业务能力进行整合，具体可以归类为应用服务器、业务能力接入、业务引擎、业务开放平台，向下根据业务能力需要测算基础服务能力，通过IaaS提供的API调用硬件资源，向上提供业务调度中心服务，实时监控平台的各种资源，并将这些资源通过API开放给SaaS用户。

PaaS主要具备以下三个特点:

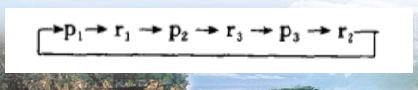
(1)平台即服务:PaaS所提供的服务与其他的服务最根本的区别是PaaS提供的是一个基础平台，而不是某种应用。在传统的观念中，平台是向外提供服务的基础。

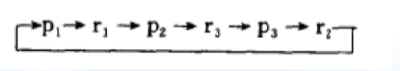
（2）平台及服务:PaaS运营商所需提供的服务，不仅仅是单纯的基础平台，而且包括针对该平台的技术支持服务，甚至针对该平台而进行的应用系统开发、优化等服务。

（3）平台级服务:PaaS运营商对外提供的服务不同于其他的服务，这种服务的背后是强大而稳定的基础运营平台，以及专业的技术支持队伍。

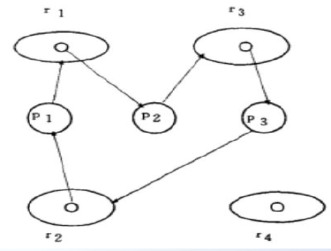
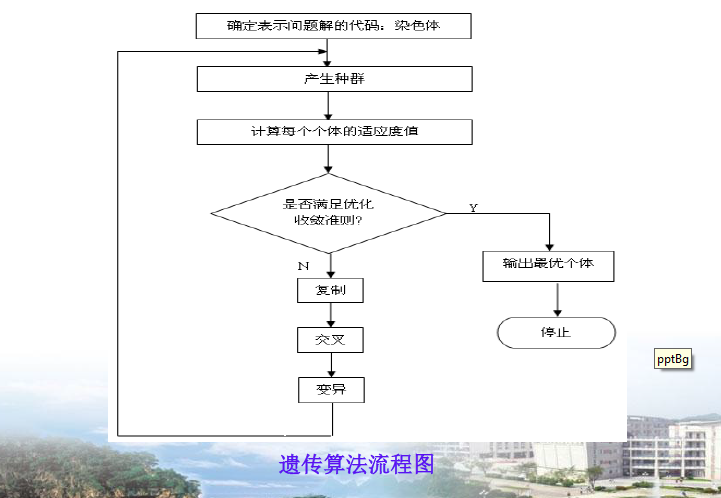
 LaaS: 提供给客户的是出租处理能力、存储、网络和其它基本的计算资源，用户能够部署和运行任意软件，包括操作系统和应用程序。客户不管理或控制的底层的云计算基础设施，但能控制操作系统、储存、部署的应用，也有可能选择网络组件（例如，防火墙，负载均衡器）。

十六. OpenStack的核心组件及主要功能。

概念：

 OpenStack最早由美国国家航天局（NASA）研发的Nova和Rackspace研发的Swift组成；后来以Apache许可证授权，是一个旨在为公共及私有云的建设，面

向云平台管理的自由软件和开放源代码项目。

核心项目组成：

Nova：计算 ； Keystone：认证 ；Glance：镜像 ；Cinder：块存储

Neutron：网络 ；Swift：对象存储 ；Heat：编排 ；Ceilometer：计量

Horizon：界限； Trovel

