计算机网络

1 概述

1. 时延带宽积： *[Math Processing Error]时延带宽积=传播时延*带宽* 表示的是这样的链路能够容纳多少比特
2. 往返时间RTT：在计算机网络中，往返时间RTT是一个重要的性能指标，因为在许多的情况下，网络上的信息不是单向传输而是交互的。 往返时间包括在链路上往返传播的时延，排队时延以及转发数据的时延。
3. 有效数据率： *[Math Processing Error]有效数据率=数据长度/(发送时间+RTT)* 单位是*[Math Processing Error]bit/s*
4. 利用率: 利用率分为信道利用率和网络利用率，信道利用率表示信道有百分之几的时间是被利用的。网络利用率是网络信道利用率的加权平均值。信道利用率并非是越高越好，当然，网络利用率也不是越高越好。**当信道利用率或者网络利用率过高时，会产生非常大的时延。**
5. 协议与服务：协议是两个对等的实体(或多个实体)进行通信的规则集合。在协议的控制下，两个对等实体之间的通信使得本层能够像上一层提供服务。要实现本层的协议，还需要下面一层所提供的服务。协议与服务的概念是不同的，协议是“水平的”，但是服务是“垂直的”。第n层向上面的第n+1层所提供的服务实际上已经包括了它下面各层所提供的服务(因为第n-1层也向第n层提供服务，第n-2层也向第n-1层提供服务)
6. 协议有意和很重要的特点，就是协议把所有的不利条件都估计到，而不能假定一切都是正常的和非常理想的。
7. 网络体系结构描述的是网络的层次，每层使用的协议，每层要完成什么样的功能，**没有描述协议内部的实现细节**

2 物理层

1. 可以将物理层的任务描述成为确定与传输媒介接口有关的一些特性，即：
   - 机械特性：接口的形状和尺寸，引脚的数目和排列，固定和锁定装置等
   - 电气特性：接口电缆的各条线上出现电压的范围
   - 功能特性：指明某一条线上出现的某一种电平的意义
   - 过程特性：指明对于不同功能的各种肯能事件出现的顺序
2. 模拟信号：代表消息的参数的取值是连续的
3. 数字信号：代表消息的参数的取值是离散的。不同离散数值的基本波形称为**码元**。在用用二进制编码时，只有两种不同的码元，一种代表0，一种代表1
4. 三种通信的基本方式：
   - 单向通信：又称为单工信道，即只能有一个方向的通信而没有反方向的交互。例子：广播。
   - 双向交替通信：又称为半双工通信。双方都可以发送信息，但是双方不能同时发送(当然也不能同时接收)。这种通信只能一边发送另一边接收，或者是反过来。例子：对讲机(对讲机一般都是半双工的，电影里面特种兵在用对讲机通话时一般都是你说一句我听一句，很少有两个人同时在一起对喷的.....)
   - 双向同时通信： 又称为全双工通信。双方都能同时发送信息同时接受信息。例子：手机
5. 常用的编码方式(21年选择题考了，我没答上来。。。)：
   - 不归零制：正电平代表1，负电平代表0 (最简单，一条线的那种)

- 归零制：正脉冲代表1， 负脉冲代表0 (正脉冲：凸，负脉冲:凹) 王道书上的波形和教材上的波形不一样....
- 曼彻斯特编码：位于周期中线向上跳代表0，向下跳代表1 曼彻斯特编码有自同步能力
- 差分曼彻斯特编码：在每一位的中心处都有跳动，位于开始边界有跳动的代表0，没有跳动的代表1 (如果这一位是0的话，波形与上一位相同，如果这一位是1的话，波形和上一位相反,当然也可以规定相反) 差分曼彻斯特编码也有自同步能力

6. 比特率与波特率
   - 比特率是指单位时间内传输的二进制码元个数
   - 波特率表示单位时间内传输的码元个数
   - 比如某一个系统有16种不同电平，假设现在用二进制编码来表示这16种电平，则二进制编码需要4位。这样的系统在传输时，一段时间内如果传输了一个码元，实际上传输了4位二进制编码，这样比特率就是波特率的4倍。再比如曼彻斯特编码，虽然只有两种电平(0和1),但是对于每一种电平需要两位二进制编码来表示(分别是01和10)。如果传播一个码元，实际需要传输2位二进制编码，这样比特率是波特率的两倍。
7. 奈氏准则与香农定理
   - 奈氏准则：*[Math Processing Error]理想低通信道下的极限数据传输速率=2Wlog2(V)* 其中W为信道的带宽。V表示每个码元的离散电平数目(实际做题发现貌似就是系统中有多少电平数目…)
   - 香农定理：*[Math Processing Error]信道的极限数据传输速率=W*log2(1+S/N)* 其中W为信道带宽，S为传输信号的功率，W为噪声的功率
   - 信噪比：*[Math Processing Error]信噪比=10*log10(S/N)*,单位为dB。实际问题一般都是给信噪比，然后导出S/N,再带入香农定理
8. 引导型传输媒体
   - 双绞线：可以传输模拟信号和数字信号
   - 同轴电缆
   - 光缆
9. 码分复用：
   - 每一个站都有唯一的m bit 的**码片序列** ，如果一个站要发送1，就发送自己的码片序列，如果要发送0，就发送自己的二进制反码。为了方便，将码片种的0写为-1 1写为+1。 比如一个站的码片是00001111，那他的码片序列就是(-1,-1,-1,-1,+1,+1,+1,+1)。每一个站的码片都不同，且相互正交。而且，与各个站的反码的码片也正交。自己的规格化内积为1。
   - 现在假如有有A,B,C,D四个站，假设他们的码片长度都是10，现在有A站收到了一个信号S(这个S可能包含着B,C,D的叠加信号)。如果A想知道B发的信号是什么，只需要用B的码片与S做内积即可。如果结果为10，那么B发的就是1，如果结果为-10，那么发的就是-1，如果结果是0，那就是B啥也没发。
10. 工作在物理层的设备：中继器，集线器，网卡，网线，调制解调器

3 数据链路层

1. 数据链路层的信道主要有两种：点对点信道，广播信道。数据链路层的三个基本问题：封装成帧，透明传输和差错检测。
2. 局域网虽然是一个网络，但是不需要路由器的转发，所以从整个互联网来看，局域网仍然是数据链路层的范围。
3. 封装成帧：就是在一段数据的前后分别加上首部和为尾部，就构成了一个帧。所有在互联网上传输的数据都是以**IP数据报**为单位传送。**网络层的IP数据报**传输到数据链路层就是帧的数据部分。每一种链路层协议都规定了传送帧的数据部分的长度上限。
4. 透明传输：帧的首位都有控制字符，所以数据种的任何8 bit的组合都不能和首位的控制字符相同。如果其中某个组合和收尾的控制字符相同，需要向其前面添加一个转义字符“ESC”。
5. 差错控制：数据链路层广泛使用了循环冗余检验。(这个貌似不怎么考....)
6. 为网络层提供服务：

- 无确认无连接服务
- 有确认无连接服务
- 有确认面向连接服务
- 注：有链接就有确认，不存在**有链接无确认**的服务。

7. 点对点协议PPP：用户通常需要连接到某个ISP才能链接路联网，PPP协议就是用户和ISP进行通信时所用的数据链路层协议。
8. PPP协议由三部分组成:
   - 将IP数据报封装到串行链路的方法。IP数据报在PPP帧种就是其信息部分。这个信息部分的长度收到最大传送单元MTU的限制
   - 一个用来建立，配置和测数数据链路链接的**链路控制协议LCP**
   - 一套**网络控制协议NCP**
9. PPP协议的零比特填充：为了防止数据中出现和标志字段**01111110**一样的的信息，采用零比特填充的方法。简单的说就是数据中每发现连续的5个1，就在后面填充一个0。不论这5个1后面跟的数字是什么。比如数据01111110010，填充后是011111**0**10010。在比如，数据是0111110，填充后是01111**0**0。
10. 以太网：以太网是一种计算机局域网技术。IEEE组织的IEEE 802.3标准制定了以太网的技术标准，它规定了包括物理层的连线、电子信号和介质访问层协议的内容。以太网是目前应用最普遍的局域网技术，取代了其他局域网技术如令牌环、FDDI和ARCNET。(百度百科) 为了通信的简便，以太网采取了下面的措施：
    - 采用较为灵活的**无连接**的工作方式，不必先建立链接就可以直接发送数据。适配器对发送的数据帧不进行编号，也不要求对方发回确认。(这样可以让以太网工作起来非常简单，而局域网的信道质量很好，产生差错的概率小)。因此，以太网提供的服务是尽最大努力交付，就是不可靠的交付。如果接收站收到有问题的数据帧，直接丢弃，什么也不用做。**对有差错的帧是否需要重传是由高层决定的。**比如高层用的是TCP协议，那么TCP就会发现丢失了一部分数据，于是一段时间后，TCP就把这些数据重新交给以太网进行重传。**但是以太网不知道这是重传帧，而是当成新的数据帧来发送。**
    - 总线上只要有一台计算机在发送数据，总线的传输资源就会被占用。因此，**因此，在同一时间只能允许一台计算机发送数据。**为了减小冲突的发生概率，以太网使用了**CSMA/CD协议，意思是载波监听多点接入/碰撞检测。**以太网是半双工的。
    - 以太网数据使用了曼彻斯特编码。
11. CSMA/CD协议：先听后发，边听边发，冲突停发，随机重发
    - 1适配器从网络层获得一个分组，封装成以太网帧，放入适配器的缓存，准备发送。
    - 2如果适配器侦听到信道空闲，那么就开始发送该帧。如果监听到信道忙，那么就持监听直到信道上没有信号能量。然后开始发送该帧。
    - 3在发送过程中，持续监听信道。 如果一直没有检测到碰撞，就顺利的法这个帧发送完毕。如果检测到碰撞，就终止数据发送，并且发送一个拥塞信号，让所有用户都知道。
    - 4在终止发送后，适配器就执行指数退避算法，等待一段时间后重新回到步骤2。
12. CSMA/CD协议的最小帧长：*[Math Processing Error]最小帧长=总线的传播时*延数据传输速率*2*
13. 纯ALOHA协议与时隙ALOHA协议(教材上没有涉及，但是王道考研中有)
    - 纯ALOHA协议：当网络中的任何一个站点需要发送数据时，可以不进行任何检测就发送数据，如果一段时间未收到确认，该站点就认为传输过程中发生了冲突，发送站点需要等待一段时间再发送数据。
    - 时隙ALOHA协议：把各个站上的时间同步起来，并将时间划分成为一段段等长的时隙，规定只能在每一个时隙的开始时才能发送一个帧
14. 三种CSMA协议(教材上没有涉及，但是王道考研中有)
    - 1-坚持 CSMA协议：信道忙时持续监听，信道空闲时立刻发送数据。
    - p-坚持 CSMA协议：信道忙时持续监听，信道空闲时以p的概率发送数据，以1-p的概率推迟到下一个时隙，下一个时隙如果信道空闲就再以p的概率发送数据，以1-p的概率推迟…如果信道忙就监听信道。
    - 非坚持 CSMA协议：信道空闲时立刻发送数据，信道忙时就放弃监听，等待一个随机时间后再监听。
    - 这三个协议都没有涉及到碰撞检测，CSMA/CD协议是CSMA协议的改进版。
15. 停止-等待流量控制(王道上面有，教材上貌似没有涉及，也可能是我没看到....)：

- 停止-等待流量控制的基本原理：发送方每发送一个帧，都要等待接收方对的应答信号，之后才能发送下一个帧。接收方每接受一个帧，都要反馈一个应答信号，表示可以接受下一个帧。

16. 滑动窗口的控制(教材上貌似没有涉及，但是明显是必考内容):在任意时刻，发送方都维持一组连续的允许发送的帧的序号，称为**发送窗口**，同时接收方也维持一组连续的允许接收帧的信号，称为**接收窗口。**发送窗口用来对发送方进行流量控制，发送窗口的大小**WT代表还未收到对方确认信息的情况下，最多还能发送多少个数据帧。**在接收方，**只有收到的数据帧的序号落入接收窗口内时，才能将该数据帧收下。**如果在就收窗口外，直接**丢弃。**(其实我感觉这个滑动窗口，真的有点像算法里面的滑动窗口.^_^.)
17. 滑动窗口的一些特性：
    - 只有接收窗口向前滑动时(同时接收方发送了确认帧)，发送窗口才**有可能**(只有收到了确认帧后才一定)向前滑动。
    - 停止等待协议：发送窗口大小=1 接收窗口大小=1
    - 后退N帧协议：发送窗口>1 接收窗口=1
    - 选择重传协议: 发送窗口>1 接受窗口>1
18. 停止等待协议：最简单的滑动窗口，接受和发送窗口都是1。其中出了出现数据帧丢失以外，还可能会出现两种问题：
    - 到达目的站的帧可能遭到破坏，接收站将其丢弃。解决：发送站设置一个计数器，如果时间到后还没有收到接受站的确认帧，就再发一次，直到收到确认帧。(接收站是不会发送过来错误信息的)
    - 数据帧正确，但是返回的确认帧被破坏。此时接收方已将收到了正确的数据帧，但是发送方收不到确认帧，因此发送方会重传已经被发送过的数据帧，接受方如果收到同样的数据帧，就会丢弃这个帧，同时再发送一个确认帧。
19. 后退N帧协议：发送方无需在收到上一个帧对的ACK后才开始发送下一个帧，而是可以连续的发送多个帧。当接受方检测出失序的信息帧后，要求发送方重新发送最后一个正确信息帧**之后的信息帧。**比如说发送方一次连着发送了 1 2 3 4 5 6 7， 接收方收到了1 2 3，结果4号帧丢失了，此时接收方即使真的收到了5 6 7也不会保存，而是直接丢弃。接收方会要求发送方从新发送 3 以后的数据帧。后退N帧协议因为没有保存后面的帧，所以如果信道质量很差时，数据帧频繁的丢失，此时后退N帧协议的效率可能不如停止等待协议。
20. 后退N帧的接收窗口大小为1。如果采用n比特帧进行编号(就是数据帧的编号一共有2^n种)，则发送方窗口要小于等于 (2^n)-1。为什么不是2^n呢? 考虑这种情况：一共有8种编号的数据帧，发送方一次性把8个帧全部发送出去，接收端全部正确接收，发送一个确认帧。但是这个确认帧丢失了，所以一段时间后发送端再次重新发送这8个帧。但是接收端并不知道确认帧已经丢失去，确认端会认为这8个帧是新的8个数据帧(因为编号是循环的，接收端收到1~8以后向后滑动到下一个1，分辨不出来这是上一组的帧还是下一组的帧)，所以继续接收，这样就多收了一组数据帧，造成了数据错误。 入过发送窗口为7，发送端发出1~7的数据帧，接收端正确收到，但是返回的确认帧又丢失了(这个确认帧也太不靠谱了 >_<) 此时一段时间后发送方重新发送这1-7这7个帧，此时接收方本来需要8，但是收到了1~7，就会重新一个确认帧来通知发送方。
21. 选择重传协议：选择重选协议会避免重复传送那些已经正确到达接收端的数据帧。接收端有缓冲区，缓冲区的大小等于接收窗口的大小。接收端一旦认为出现错误，就会发送一个NAK帧给发送方，发送方会重新发送NAK帧中指定的帧。
22. 选择重传协议中，接收窗口 <= 发送窗口，不过一般都是 接收窗口= 发送窗口。且 接收窗口+发送窗口<=(2^n)。当接收窗口取最大值时，接收窗口=发送窗口=2^(n-1)。
23. 以太网的mac层：在局域网中，mac地址又叫做物理地址或者硬件地址。mac地址是独一无二的48位二进制地址。
24. 冲突域与广播域：
    - 当一个网卡发送信息时，只要有可能和另一个网卡的信息发生冲突，那么这些发生冲突的网卡构成一个冲突域。
    - 当一个方法发送广播时，能收到这个广播的所有网卡的集合构成一个广播域。
25. 工作在数据链路层的设备：网桥，交换机。网桥和交换机都能隔离冲突域，但是不能隔离广播域。

(先这些，以后继续补充)

4 网络层

1. 网络层：**网络层向上只提供简单灵活的，无连接的，尽最大努力交付的数据报服务。**网络层**不能保证服务的质量**，这里的数据报，基本说的就是IP数据报。(其实我感觉网络层能用两个字概括：IP)
2. 网际协议IP：网际协议IP只TCP/IP体系中最重要的协议之一。它还有三个配套使用的协议：
   - 地址解析协议 ARP (Address Resolution Protocol)
   - 网际控制报文协议 ICMP (Internet Control Message Protocol)
   - 网际组管理协议 IGMP (Internet Group Management Protocol)
3. 虚拟网络链接(虽然这里王道书中没有，但是我总感觉很多题中都有这里的知识点)
   - 要把世界上的所有网络都连接起来，是非常复杂的，比如有很多问题：不同寻址方案，不同的最长分组，不统计网络接入机制....将这些网络连接起来，需要一些中间设备。
   - 物理层使用的设备叫**转发器。**
   - 数据链路层使用的中间设备叫**网桥**或者**桥接器**
   - 网络层使用的设备叫**路由器**
   - 在网络层以上的设备叫**网关。**网关链接两个不相容的系统需要在高层进行协议的转换。
   - 路由器是一台专用的计算机，在互联网中进行路由选择。(后来我听做过通信的群友的介绍，路由器是一种非常NB的设备)**由于历史原因，许多关于TCP/IP的文献把网络层使用的路由器叫网关(教材里面有时也会这样用)**
4. IP地址:
   - IP地址给互联网上的每一台计算机或者路由器的一个接口分配一个32位的标识符。
   - A类IP地址：8位网络号+24位主机号，其中网络号第一位必须是0
   - B类IP地址：16位网络号+16位主机号，其中网络号的前两位必须是10
   - C类IP地址：24位网络号+8位主机号，其中网络号的前三位必须是110
   - D类IP地址：前四位是1110，用于多播(一对多通信)
   - E类IP地址：前四位是1111，保留以后使用
5. 特殊的IP地址：
   - 网络号全0的IP地址：表示本网络
   - A类IP地址中网络号是127(01111111)：本地软件环回测试本主机的进程之间通信。例如127.0.0.1。因此，网络号位127的地址根本不是一个网络地址。
6. 一般不适用的特殊IP地址(教材121页的表格)：

网络号	主机号	源地址使用	目的地址使用	代表的意思
0	0	可以	不可	在本网络上的本主机
0	host-id	可以	不可	在本网络上的一台主机号位host-id的主机
全1	全1	不可	可以	在本网络上广播(路由器不转发)
net-id	全1	不可	可以	对网络号为net-id上的所有主机广播(我认为路由器应该要转发)
127	非全0或全1的任何数	可以	可以	本地软件环回测试

7. IP地址的一些特点：
   - IP地址的管理机构在分配IP是只分配网络号，主机号由得到网络号的单位自行划分。
   - 路由器仅根据目的主机所连接的网络号来转发分组，不考虑主机号。
   - IP地址不是某个计算机的”身份证”,而是一个主机和一条链路的接口。如果主机连接到两个网络，它必须也有两个相应的IP地址，其网络号必须是不同的。
   - 用转发器和网桥连接起来的若干个局域网仍然是一个网络。只能有一个网络号。
   - 路由器的每一个接口都有一个不同网络号的IP地址

8. IP地址与MAC地址的区别：
   - MAC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(IP地址是用软件实现的)
   - 使用IP地址的IP数据帧在数据链路层被封装成MAC帧，MAC帧的源地址和目的地址都是MAC地址，这两个MAC地址都写在了MAC帧的首部。
   - 在网络上传输的MAC帧，他的源MAC地址和目的MAC地址可能会因为路由器的转发而发生变化，但是他的源IP地址和目的IP地是不会变化的。
   - 在IP层抽象的互联网上只能看见IP数据报
   - 路由器只会根据目的IP地址的网络号进行路由选择
   - 在**局域网**的链路层，只能看见MAC帧。
9. 地址解析协议ARP(Address Resolution Protocol)：
   - 作用:已知一个机器的IP地址，需要找出其对应的MAC地址，地址解析协议就是用来解决这样的问题。
   - 每一台主机都有一个ARP高速缓存，里面有**本局域网**上面的各个主机与路由器的IP地址和MAC地址映射表。
10. 路由器有MAC地址，网桥没有MAC地址(这个没有找到资料，百度知道上查的，不清楚对不对.....)
11. IP 数据报的含义(数据报的格式一般都会给出，不然就太变态了.....):
    - IP数据报的首部有20字节的固定部分
    - 版本：4位，指的是IP协议的版本。通信双方的IP协议版本必须一致。
    - 首部长度：4位，表示首部长度是多少字节，注意其单位是4字节。
    - 区分服务：8位，只有使用区分服务时才用到这个字段，一般不涉及
    - 总长度：16位，首部和数据之和的长度，单位是字节。以太网规定在以太网上传输的IP数据报的最大**总长度**是**1500字节**。(这个貌似考过)
    - **标识(identification)**：16位，数据报在分片后，相同标识的数据报分片才能组装成原来的数据报
    - **标志(flag)**：占3位，但是只有两位有意义。标示字段中的最低为为MF(more fragment)，MF=1表示后面还有分片，MF=0表示这是数据报片中的最后一个(即后面没有分片)。中间的一位DF(dont fragment)，表示不能分片。当DF=1时**表示数据报不能分片**，DF=0时表示可以分片。
    - 片偏移：13位，表示在某一个分片在分片之前，相对于 **数据字段** 的起点，该片从何处开始。片偏移的单位是8字节。也就是说每一个分片的长度一定是8字节的整数倍。
    - 生存时间：8位，表示数据报在网络中的寿命，一旦生生存时间等于0，就丢掉这个数据报。
    - 协议：占8位，表示数据报的数据部分使用了那种协议(比如ICMP，TCP，UDP之类的)
    - 首部检验和：16位，这个字段只检验数据报的首部，但是不包括数据部分。
    - 源地址：32位
    - 目的地址 32位
    - 注意区分标识(identification)和标志(flag)，标识更类似于某种身份信息，标志类似于属性。标志占的位数很小(和算法题用到里面的flag一样，一般都不会用一个非常长的数字来表述某个属性吧....)
12. 路由器的路由表中，对一条路由最重要的两个信息:(目的网络地址，下一跳地址)
13. 划分子网：
    - 一个拥有许多物理网络的单位，可将所属的物理网络划分为若干个子网。划分子网属于一个单位内部的事情。本单位以外的网络看不见这个网络是由多少个子网组成的。因为这个单位对外表现为一个网络。
    - “三级IP地址” {网络号，子网号，主机号}
    - 凡是从其他网络发个本单位某一台主机的IP数据报，仍然是根据IP数据报的目的网络好链接在本单位上的路由器。**但是，路由器在收到IP数据报后，再根据目的网络号和子网号来找得到目的子网，把IP数据报交付给主机。**
    - 没有划分子网时，IP地址时两级结构，划分子网后，IP地址变成了三级结构，**划分子网只是把IP地址的主机号这部分再划分，而不改变IP地址原来的网络号**
14. 子网掩码：
    - 如果一个网络不划分子网，那么这个网络的子嘛就是默认子网掩码。
    - A类地址的默认子网掩码：255.0.0.0
    - B类地址的默认子网掩码：255.255.0.0
    - C类地址的默认子网掩码：255.255.255.0

- 划分子网增加了灵活性，但是减少了能链接在网络上的主机总数。(因为划分了子网后，每一个子网的主机号都不能是全0或者全1，没划分以前只要在网络号中去掉一个全0全1就可以了)

15. 划分子网后，路由表中必须由三个内容:目的网络地址，子网掩码，下一跳地址。
16. 无分类编制CIDR(构成超网)：
    - 取消了A，B，C类地址和划分子网的概念，把32位IP地址划分成两部分，前面是**网络前缀**，后面是网络号
    - 和可以使用斜线记法，如：128.14.35.7/20，前面20位是网络前缀，后面12位是主机号。
    - CIDR使用的是**地址掩码**，虽然CIDR不适用子网，但是由目前还有一些网络在只用子网划分和子网掩码，所以CDIR中的地址掩码还可以继续称为子网掩码。在使用斜线记法中，斜线后面的数就是1的个数。
    - 路由选择时使用最长前缀匹配。就是从匹配正确的结果可能有多个，在这些结果中选择前最长共前缀的网络进行跳转。
17. 网际控制报文协议ICMP(这个貌似涉及的不多)：
    - 为了有效地转发IP数据报和提高交付成功的机会，在网际层使用了国际报文协议（Internet Control Message Protocol）
18. 内部网关协议：IGP(Interior Gateway Protocol):在一个自治系统内部使用的路由选择协议，比如RIP和OSPF协议。
19. 外部网关协议：源主机和目的主机在不同的自制体系中，需要一种协议可以将路由选择信息从一个路由选择体系传递到另一个路由选择体系中。如BGP-4.(以上两个”网关”，指的好像就是路由器)。
20. RIP协议：中名字叫路由选择协议。涉及到：
    - 距离向量算法(来源于Bellman-Ford算法)
    - RIP协议在传输层使用的是**UDP数据报**进行传送。
21. OSPF协议：
    - 使用dijkstra算法
    - 不使用UDP，而是**直接使用IP数据报**传送。
22. 组播(多播):组播一定是使用UDP的，因为TCP是面向链接的。组播的工作形式非常像递归遍历一棵树。
23. 网络地址转换NAT:(按照我的理解，可能是错的)简单的说就是现在的计算机太多了，这些计算机都链接到一个互联网上IP地址不够用，或者是有的计算机专用网络为了数据安全要和外网进行隔离，所以引入私有IP地址。这些私有IP地址不能直接和外网链接，必须先通过NAT协议来转换成一个全球IP地址才可以。私有IP地址是可以复用的，比如数学院组成一个内网，物理学院也组成一个内网，数学有个计算机的IP是192.168.10.10，物理学院也有一台计算机的IP是192.168.10.10，但是他们通过NAT协议所映射的外网地址是不同的。
24. 当NAT路由器有N个全球IP地址时，专用网内**最多可以同时**有N台主机接入互联网。但是不代表专用网内最多有N台主机。如果有大于N台主机，它们可以轮流接入互联网。
25. 私有IP地址：
    - A类：10.0.0.0-10.255.255.255
    - B类：172.16.0.0-172.31.255.255
    - C类：192.168.0.0-192.168.255.255
26. 工作在网络层的设备：路由器,可以隔离广播域。

5 传输层

1. 传输层(教材中叫运输层，但是王道和考研中一般叫传输层)是整个网络体系结构中的关键层次之一。
2. 从IP层来说，通信的两端是两台主机之间的通信，IP数据报的首部也是明确的标志了两台主机的IP地址。但是”两台主机之间的通信”这种说法不够明确，因为真正的通信的实体时来台主机之间的进程。**是一台主机的进程与另一台主机的进程之间交换数据。**因此严格的说，两台主机之间的通信是指两台主机之间的进程通信。

3. 网络层为主机之间提供逻辑通信，而传输层为应用进程之间提供**端到端的逻辑通信。**
4. 传输层要为收到的报文提供**差错检测。**网络层的IP数据报里面的检验和字段，只是用来检测首部受否出现差错，而不检测数据部分。
5. 传输层向高层用户屏蔽了下面网络核心的细节(比如网络拓扑，所采用的路由选择协议等)。它使应用进程看见的好像是在连个传输实体之间有一条端到端的逻辑通信信道。
6. TCP使提供**面向连接**的服务。在传输数据传输之前必须建立可靠的链接，数据传输之后要释放链接。TCP不提供广播或者组播服务。
7. UDP在传输之前不需要建立链接。虽然UDP不提供可靠的交付，但是在某些情况下，UDP确实使一种最有效的工作方式。
8. 使用UDP或者TCP的一些应用：

应用	**应用层协议**	**传输层协议**
名字转换	DNS(域名系统)	UDP
文件传输	TFTP(简单文件传送协议)	UDP
路由选择协议	RIP(路由信息协议)	UDP
IP地址配置	DHCP(动态主机配置协议)	UDP
网络管理	SNMP(简单网络管理协议)	UDP
远程文件服务器	NFS(网络文件系统)	UDP
IP电话	专用协议	UDP
流式多媒体通信	专用协议	UDP
多播	IGMP(网际组管理协议)	UDP
电子邮件	SMTP(简单邮件传送协议)	TCP
远程终端接入	TELNET(远程终端协议)	TCP
万维网	HTTP(超文本传输协议)	TCP
文件传送	FTP(文件传送协议)	TCP

9. UDP(User Datagram Protocol用户数据报协议)的首部格式：
    - 源端口：源端口号，在需要对方回信时选用，不需要时可以使用全0.
    - 目的端口：目的端口号，在终点交付报文时使用。
    - 长度：UDP用户数据报的长的，单位是字节，其最小值为8字节(仅有首部长度)
    - 检验和：检测UDP用户数据报在传输中是否存在差错，有错就丢弃。
10. UDP用户数据报在计算检验和时，要在UDP用户数据报之前在添加12字节的伪首部，计算方法与IP数据报类似，但是IP数据报只是计算首部，**UDP数据报会把首部和数据部分一起计算。**(教材上是这么写的，不过看具体过程伪首部也要加上)
11. 传输控制协议TCP(Transmission Control Protocol):
    - TCP是一个面向链接的传输层协议。
    - 每一个TCP链接只能有两个端点。每一条TCP链接都只能是点对点的，
    - TCP提供的是全双工通信。
    - TCP面向字节流。
12. TCP报文的首部格式：
    - 源端口和目的端口：分别占2字节，分别写入源端口号和目的端口号。

- 序号：占4字节，TCP是面向字节流的，在TCP链接中的字节流每一个字节都按顺序编号。(有点类似IP数据报中分组后的编号)
- 确认号：占4字节，是**期望收到对方下一个报文段的第一个数据字节的序号。**
- 数据偏移：占4位，指出TCP报文段的数据距离TCP报文段的起始部分有多远。
- 保留：占6位，保留为今后使用，但目前应置为0。
- 紧急URG：当URG=1，表明紧急指针有效，告诉系统此报文中有紧急数据，要尽快传送。不需要按照原来的排队顺序来传送。
- 确认ACK：当ACK=1时，确认字号段才有效，当ACK=0时，确认号无效。TCP规定，当建立链接后所有传输的报文的ACK置为1。
- 推送(PSH)： 当两个应用程序进行交互式通信时，有时在一端的应用程序希望在键入一个命令后能立刻受到对方的响应。在这种情况下啊，发送方的PSH=1.
- 复位RST：当RST=1时，说明TCP链接中有严重的差错，必须释放连接。然后重新建立链接。
- 同步SYN：在建立链接时用来同步序号。当SYN=1而ACK=0时，表明这是一个**请求报文段**。如果对方同意建立链接，在响应的报文段中使用SYN=1和ACK=1。因此，SYN置为1是一个链接请求或者链接接收报文。
- 终止FIN：用来结束一个链接，当FIN=1时，表明此段报文的发送方的数据发送完毕，请求释放运输链接。
- 窗口：占2字节，窗口值时[0,2^16-1]之间对的整数。**窗口指的是发送本报文段的一方的接收窗口。**窗口值是为了告诉对方，**从本报问段首部中的确认信号算起，接收方目前允许对方发送的数据量(单位为字节)**。窗口值是接收方让发送方设置发送窗口的依据。
- 检验和：占2字节，包含首部和数据两部分。
- 紧急指针：只有URG=1时才有意义。指出本报问中紧急数据的字节数。因此，紧急指针时指出了紧急数据的末尾在报文中的位置。
- 选项：长度可变，最长可以40字节，没有选项时，TCP首部长度20字节。

13. TCP使用滑动窗口来实现流量控制。
14. TCP的拥塞控制：TCP进行拥塞控制的算法有四种，分别是：慢开始，拥塞避免，快重传，快恢复。其中快重传和快恢复是慢开始，拥塞避免算法的改进(这些算法需要记住具体怎么操作)
15. TCP运输链接管理：TCP是面向链接的协议。链接有三个阶段吗，即为：连接的建立，数据传送和链接释放。
16. **TCP链接的建立(三次握手)**：
    - 第一步，客户机的TCP首先向服务器的TCP发送连接请求报文，这个特殊的报文段首部中的同步位SYN=1，同时选择一个初始序号seq=x。TCP规定，SYN报文段不能携带数据，但是要消耗一个序号，这时，TCP客户端程序进入了SYN-SEBT(同步已发送)状态。
    - 第二步：服务器的TCP收到连接请求报文后，如果同意建立连接，则向客户机发出出确认。并为该TCP链接分配缓存和变量。在确认报文中，SYN=1，ACK=1,确认号是ASK=x+1。同时，自己也选择一个初始序号seq=y。这段确认报文也不能携带数据。但是也要消耗一个序号。此是，TCP服务器为SYN-RCVD(同步收到状态)。
    - 第三步：当客户机收到报文后，还要向客户端确认，并且为该TCP链接分配缓存和变量。确认报文的ACK置为1，确认号ack=y+1.序号seq=x+1。该报文已经可以携带数据。如果不携带数据，就不消耗序号。TCP客户端进入了ESTABLISHED(已确立链接状态)。
17. **TCP链接的释放(四次握手)**：
    - 第一步：客户端打算关闭连接，向其TCP发释放连接的报文，并停止发送数据。主动关闭TCP链接。该报文的终止位FIN=1，序号seq=u，它等于全面已经传送数据报的最后一个字节数+1。这条报文**即使不携带数据，也要消耗一个序号。**这是TCP客户端进入了FIN-WAIT-1(终止等待1)状态。**TCP是全双工的，可以想象成一条TCP链接上有两条数据通路，在发送FIN的一端不再发送数据(即为关闭其中一条数据通道)，但是对方还可以发送数据。**
    - 第二步：服务器收到连接释放报文后即发出确认，确认号为ack=u+1。序号seq=v，等于前面已经传送过的数据的最后一个字节数+1。然后服务器进入CLOSE-WAIT(关闭等待)此时，从客户端到服务器方向的链接就释放了，TCP链接处于半关闭状态。但是如果服务器要发送数据，客户端也要接收。也就是从服务器到客户端的链接还没有完全关闭。
    - 第三步：如果服务器已经没有要向客户端发送的数据，就通知TCP释放链接，此时发出FIN=1的连接释放报文段。设该报文段的序号为w(因为刚才服务器可能又发送了一些数据)，还要重复上次发送的确认报号ack=u+1。此时服务器进入了LAST-ACK(最后确认)状态。

- 第四步：客户机收到了连接释放报文后，必须发出确认。把报文确认段中的确认位ACK=1。确认号ack=w+1，序号seq=u+1。此时TCP链接还没有释放，必须经过时间按等待计时设置的时间2MSL后，客户机才进入CLOSE状态。

18. TCP链接总结：
    - 连接建立分三步(A为客户端，B为服务器)
    - 1 SYN=1,seq=x。 A->B
    - 2 SYN=1,ACK=1,seq=y,ack=x+1 B->A
    - 3 ACK=1,seq=x+1,ack=y+1(可携带数据) A->B
    - 链接释放分4步
    - 1 FIN=1，seq=u (可携带数据) A->B
    - 2 ACK=1，seq=v，ack=u+1 (可携带数据) B->A
    - 3 FIN=1，ACK=1,seq=w,ack=u+1 B->A
    - 4 ACK=1，seq=u+1，ack=w+1 A->B

6 应用层

1. 每一个应用层协议都是为了解决某一类问题，而这些问题又必须通过位于不同主机中的多个进程直之间的通信和协同工作来完成。
2. 应用层协议应当定义：
    - 应用进程交换报文的类型，入请求报文和响应报文。
    - 各种报文类型的语法，如报文中各个字段及其详细描述。
    - 字段的予以，即为包含在字段中的信息的含义。
    - 进程何时，如何发送报文，以及对报文进行的响应。
3. 应用层的许多协议都是基于客户端服务器方式。即为P2P对等通信方式。实质上也是一种特殊的客户服务器模式。客户是服务的请求方，服务器是服务的提供方。
4. 域名系统DNS(Domain Name System):是互联网使用的命名系统，用来把便于人们使用的及其名字转换为IP地址。计算机的用户只是间接而不是直接的使用域名系统，但是DNS却为互联网的各种网络应用提供了核心服务。
5. 域名的语法：比如wwww.baidu.com，从右到左，com为顶级域名，baidu为二级域名，www为三级域名。域名不区分大小写。由多个符号组成的完整域名总部不超过255个字符。
6. 域名服务器：
    - 根域名服务器
    - 顶级域名服务器
    - 权限域名服务器
    - 本地域名服务器
7. 主机向本地域名服务器查询一般采用的都是递归查询。查询中使用UDP报文。
8. 本地域名服务器向跟域名服务器查询通常采用迭代查询。查询中使用UDP报文。
9. 为了提高域名服务器的可靠性，DNS域名服务器都把数据复制到几个域名服务器来保存。其中一个是主域名服务器，其他的是辅助域名服务器。
10. 为了提高DNS查询效率，并且嘉庆根域名服务器的符合和减少互联网上DNS查询报文属灵，在根域名服务器中广泛地使用了高速缓存。
11. FTP协议(基于TCP)：FTP使用客户服务器模式，一个FTP服务器可以同时为多个客户提供服务。FTP的服务分为两部分组成，一个是**主进程**，负责接收新的请求。另外一个是**从属进程**。负责处理单个请求。
12. 主进程的工作步骤如下：
    - 打开熟知端口号，使客户进程能够连接上。
    - 等待客户进程发出链接请求。

- 启动从属进程处理客户进程发来的请求。从属进程对客户进程请求处理完毕后即终止，但是从属进程在运行期间根据需要还可以创建其他的一些子进程。
- 回到等待状态，继续接收其他客户发来的请求。主进程与从属进程的处理使并行的。

13. 服务器有两个从属进程：控制进程和数据传送进程。
14. 在进行文件传输时，FTP的客户端和服务器有两个并行的TCP链接，分别是**控制连接**和**数据连接**。控制连接在整个会话期间一直保持打开。
15. 简单文件传送协议TFTP(基于UDP)：
    - 每一次传送的数据报文中有512字节的数据，但是最后一次可以不足512字节。
    - 数据报问按照序列编号，从1开始。
    - 支持ASCII码或者二进制传送。
    - 可以对文件进行读或写。
    - 使用很简单的首部。
16. 超文本传输协议HTTP：从层次的角度看，HTTP是面向事务的应用层协议。HTTP使用了面向连接的TCP协议，保证了数据的可靠。HTTP协议虽然使用的TCP协议，但是**HTTP协议本身是无连接的**。这就是说通信的双方在交换HTTP报文前不需要先建立HTTP链接。
17. 电子邮件：电子邮件从发送到接收全程使用了TCP链接。电子邮件中最总要的两个标准：简单邮件传送协议SMTP和互联网文本报文格式[RFC 5322]。
18. 发送邮件用到的协议：SMTP，读取邮件用到的协议：POP3。

7 其余补充

1. 在教材的绪论部分的一张图片中，分组交换的速度是大于报文交换的速度，但是区别好像只是分组的对报文进行了切片，这是为什么?
    - 我认为这里可以类比于指令的流水线。在报文交换中，需要把整个报文全部传输完毕，才能在下一个节点进行传输。但是如果对报文进行切片，在数据的传输过程中，原先的数据其中一部分在一个节点传输完毕后，不用再等待其他数据的传输，直接到达下一个节点。这就会比报文交换更节省时间。假设对数据进行极为细小的切片，数据报退化比特流，那么每个比特在节点中传输的时间基本可以忽略不计，此时的传输过程更加接近电路交换。
2. 对于几个网络通信场景的个人理解，假设路由器链接不同网络，同一个网络中的设备只有网桥(**这个理解极有可能是错误的**)
    - 在局域网内两台主机通信：在局域网中，一台主机A要和另一台主机B通信，首先A先把要通信的数据封封装成IP数据报，然后再封装成mac帧。这里因为A和B同在一个局域网上，所以A知道B的mac地址。mac帧上的目的mac地址就是B的mac地址。这个mac帧被发出后，因为有网桥的转发，所以这个局域网上只有部分主机感受到了这个mac帧。如果没有网桥，所有的主机都会感受到这个mac帧。(注意经过网桥后，mac帧的源地址仍是A的mac地址，因为mac帧经过网桥时不会改变mac帧的首部，只有路由器才改变。)虽然感受到这个mac帧的主机有很多，但是只有主机B的mac地址与这个mac帧的目的地址相同，别的主机发现mac帧的目的地址不同后直接丢弃这个mac帧。此时，B收到mac帧，解封装后得到IP数据报。A与B的通信完成。(不过再局域网中每一个主机都知道另一个主机的mac地址，所以我认为在局域网主机可以直接通过mac就可以找到另一台主机，不涉及到IP协议。)最后，假设这个局域网中还有一个路由器与其他网络相连，这个路由器也会收到这个mac帧，路由器也会检查mac帧的首部，发现mac地址与自己不同后丢弃。
    - 在不同的网络中的两台主机间的通信：在不同的两个网络中，一台主机A与另一台主机B通信，首先A要知道B的IP地址，但是A并不知到B的mac地址(应该说知道了也用不上)。A把要通信的数据封装成IP数据报，此时IP数据报的目的地址时B的IP地址。之后被封装成mac帧，而A通过查询发现局域网中没有主机的IP地址和目的IP地址相同(这个过程是我猜的)，所以主机A推测出IP地址不属于这个网络，需要路由器的转发，所以A把mac帧的目的地址写为路由器的mac地址，并且发出该mac帧。这时路由器收到了这个mac帧，发现mac帧的目的地址与自己相同，开始解封装。路由器在提取出IP数据报中的目的IP地址后，重新封装成mac帧，再通过路由算法转发这个mac帧。此时mac帧的源地址变成了路由器的地址，mac帧的目的地址可能是下一个路由器的mac地址，或者是目的主机B的mac地址。总之，可能会经过多次转发，但是最后mac帧的目的地址一定是主机B的mac地

址，它的源地址是最后一个转发它的路由器的mac地址，其实也是主机B所在网络的路由器的mac地址。为什么主机B的mac地址最后又知道了呢?因为这个mac帧是主机B所在网络的路由器最后转发的。路由器是一台特殊的计算机，里面同样会存储所在局域网中的所有主机的mac地址，不然就没法再封装mac帧了。最后B收到这个mac帧，通信完成。

操作系统

1.绪论

1. 操作系统：操作系统是一组控制和管理计算机硬件和软件资源，合理地对各类作业进行调度，以及方便用户使用的程序的集合。操作系统的目标:
   - 有效性：提高系统资源的利用率 提高系统资源的吞吐量
   - 方便性
   - 可扩充性
   - 开放性
2. OS是用户与计算机硬件之间的接口，用户可以通过三种方式来使用计算机：
   - 命令方式，这是由OS提供了一组联机命令接口，用户用键盘输入相关命令来取得操作系统的服务。
   - 系统调用方式：OS提供了一组系统调用，用户可以在自己的应用程序种通过相应的系统调用来实现与操作系统的通信，并获取它的服务。
   - 图形，窗口方式：就是你现在正在用的方式。
3. OS实现了对计算机资源的抽象。
4. **无操作系统**的计算机系统：
   - 人工操作系统：用户独占全机，CPU等待人工操作。
   - 脱机输入/脱机输出：先把数据通过外围机输入到此带上，CPU需要这些数据时，直接从磁带上调入内存。输出时，CPU先间数据输出到磁带上，磁带在外围机的控制下输出到相关的设备。优点：减少了CPU的空闲时间，提高了IO速度
5. 单道批处理系统：系统对作业的处理都是成批的，且内存中始终都只有一道作业，故称为单道批处理系统。
6. 多道批处理系统：
   - 优点1：资源利用率高：由于在内存中驻留了多道程序，它们共享资源，可以保持资源处于忙碌状态，从而使个各种资源都得到充分利用。
   - 优点2：系统吞吐量大：因为CPU和其他资源充分保持着忙碌状态，仅当作业完成时或着运行不去时才进行切换，系统开销小。
   - 缺点1：平均周转时间长：在批处理系统中，由于作业要排队，依次处理，因而作业的周转时间长。
   - 缺点2：五交互能力。一旦把作业交给系统，直到作业完成，用户都不能和作业交互。
7. 分时系统：经常被用于查询系统中，满足许多查询用户的需求。主要为：人机交互， 共享主机，便于用户上机。分时系统的特征：
   - 多路性：允许在一台主机上同时链接多台联机终端，系统按分时的分时的原则为每一个用户服务。在微观上，每个用户作业轮流运行一个时间片。多路性即同时性，它提高了系统的资源利用率，降低了费用。
   - 独立性：每个用户独占一个终端，彼此独立操作，互不干扰。
   - 及时性：用户的请求能够在极短的时间内获得响应。
   - 交互性：用户可以通过终端与系统进行广泛的人机对话。
8. 实时操作系统：实时控制，实时信息处理。实时操作系统与分时操作系统的比较：
   - 多路性：实时信息处理系统也能按照分时的原则为多个终端服务。

- 独立性：对信息的采集与对对象的控制独立，互不干扰。
- 及时性：实时控制系统的及时性，时以控制对象所要求的开始时截止时间或者完成截止时间来确定的。
- 交互性：实时信息处理系统虽然也有交互性，但是仅限于访问系统中某些特定的专用服务程序。
- 可靠性：实时操作系统要求系统又高度的可靠性。

9. 操作系统的基本特性：
   - 并行与并发：并行性是指两个或则和多个事务在同一时刻发生，并发性是指两个或者多个事务在**同一个时间间隔内发生**。在多道程序环境下，并发性是指在一段时间内宏观上有多个程序同时运行，但是在单机处理系统中，每一时刻却仅能有一道程序，故微观上这些程序这能是分时地交替执行。
   - 引入进程：通常程序是**静态的实体**，在多道程序系统中，他们是不能够独立运行的，更不能和其他的程序并发执行。在操作系统中，引入进程的目的，就是为了使得多个程序能够并发执行。
   - 进程：为了使得多个程序能够并发的执行，系统必须分别为每个程序建立进进程。**简单的说，进程是指在操作系统中能独立运行并且作为资源分配的基本单位。**它是由一组**机器指令，数据和堆栈组成的。是一个能够独立运行的活动实体。**多个进程之间可以并发执行和交换信息。一个进程在运行时需要一定的资源，如CPU，存储空间及IO等设备。
   - 引入线程：由于进程拥有自己的资源，故使得调度进程的开销非常大。通常在一个进程中可以包含多个线程。他们可以利用进程所拥有的资源，在引入线程的OS中，通常都是把**进程作为分配资源的基本单位。**而把线程作为**独立运行和独立调度的基本单位。**线程比进程小，基本上不拥有系统资源。
10. 共享性：在操作系统的环境下，所谓的**共享**，是值系统中的资源可以供内存中多个并发执行的进程(线程)共同使用。
    - 互斥共享方式：系统中的某些资源，如打印机，磁带机，虽然他们可以提供给多个进程(线程)是以哦那个，但是为使所打印或者记录的结果不混淆，规定**在一段时间内只允许有一个进程(线程)访问该资源。**计算机系统中的大所示物理设备，以及某些软件中所使用的栈，变量，表格，都属于临界资源。他们求奥求被互斥的共享。
    - 同时访问方式：系统中有一类资源，允许在一段时间内多个进程“同时”访问，这里的同时，在单处理机的环境下往往是宏观上的，而在微观上，这些进程可能是交替读对该资源进行访问。**最典型的可以工多个进程同时访问的资源就是磁盘设备，一些用冲入代码编写的文件也可以被同时共享。**
11. 虚拟技术：虚拟技术是值通过某一种技术把一个物理实体转变为若干个逻辑上的对应物。
    - 虚拟处理及技术：在虚拟处理机技术中，利用多道程序设计技术，为每一道程序建立一个进程，让多道程序并发的执行，一次来分时的使用一台处理机。
    - 虚拟设备技术：我们通过虚拟设备技术，将一台物理IO设备虚拟为多台逻辑上的IO设备。并且允许每个用户占用一台逻辑上的IO设设备。
    - 虚拟磁盘技术：通常在一台机器上只配置一台硬盘，我们可以通过虚拟磁盘技术将一台硬盘虚拟为多台虚拟硬盘。这样使用起来方便安全。例如：1，2，3，4四个卷，在通过安装程序将他们分别安装在C，D，E，F四个逻辑驱动器上。这样，机器上便有了四个虚拟硬盘。
    - 虚拟存储器技术
12. 异步性：由于受制于资源等因素，使得进程的执行通常不是一气呵成的，而是以走走停停的方式运行。
13. 处理机管理功能：
    - 进程控制：进程控制的主要功能是为作业创建进程，撤销已经结束的进程，以及控制进程在运行过程中的状态转换。在现当代OS中，进程控制开应该具有一个进程创建若干个线程的功能和撤销已经完成的线程的功能。
    - 进程同步：进程同步有的主要任务是为多个进程(含线程)的运行进行协调。有两种协调方式：进程互斥方式，进程同步方式。实现进程同步的最常见的机制是信号量机制。
    - 进程通信：进程通信的任务就是用来实现相互合作的进程之间的信息交换。
    - 作业调度：作业调度的基本任务是从后被队列中按照一定的算法，选择出若干个作业，为他们分配运行所需要的资源。
    - 进程调度：进程调度的任务是从进程的就绪队列中，按照一定算法选出一个进程，把处理及分配给它，并且为它设置运行现场。
14. 存储器管理功能：

- 内存分配：内存分配的主要任务是为每道程序分配内存空间，提高存储器的利用率，允许正在运行的程序申请附加的内存空间。OS在实现内存分配时，可以采用静态和动态两种方式，在静态分配内存中，每个作业的空间时在作业装入时确定的，在作业装入后的整个运行期间，不允许该作业再申请新的内存空间，也不许作业在内存中移动。在动态分配方式中，每个作业所要求的基本内存空间**也时在装入时确定的**，但是允许作业在运行过程中继续申请新的附加内存空间，以适应程序和数据的动态增长。也允许作业在内存中移动。
- 内存保护：内存保护的主要任务时确保每道用户程序都旨在自己的内存空间内运行，比起互不干扰。**绝不允许用户程序访问操作系统的程序和数据，也不允许用户程序转移到非共享的其它用户程序中执行。**
- 地址映射：简单的说就是把程序中的逻辑地址映射到实际的物理地址。
- 请求调入功能：**允许在装入一部分用户程序合和数据的情况下**，就能启动该程序运行。在运行过程中，若发现要继续运行时所需要的程序和数据尚未装入内存，可以向OS发出请求，由OS从从盘中将所需要部分调入内存，以便继续运行。
- 置换功能：若发现内存中一节那个没有足够的空间来装入需要调入的程序和数据时，系统应能将内存中的一部分暂时不用的程序和数据调至盘上，以腾出内存空间，然后在将所需要调入的部分装入内存。

15. 设备管理功能：
    - 缓冲管理：缓冲区为了解决设备的速度不匹配，缓冲区由OS中的缓冲管理机制将他们管理起来。
    - 设备分配：设备分配的基本任务是根据用户进程的 I/O 请求、系统的现有资源情况以及按照某种设备的分配策略，为之分配其所需的设备。
    - 设备处理：设备处理程序又叫做**设备驱动程序**，其基本任务是用于实现CPU和设备驱动器之间的通信。
16. 文件管理功能：
    - 文件存储空间的管理：系统应该设置相应的数据结构，用于记录文件存储的空间的使用情况，以供分配存储空间时参考。系统还应具有对存储空间仅从分配胡回收的功能。为了提高存储空间的利用率，对存储空间的分配，通常是离散式的，为了减少外存零头，并且以盘块为基本单位分配。
    - 目录管理：为了使用户方便地在外存上找到自己所需要的文件，通常由系统为每个文件建立一个目录项。目录项包括：文件名，文件属性，文件在磁盘上的物理位置…**由若干个目录项又可以构成一个目录文件。**
17. OS与用用户之间的接口：
    - 用户接口：它是提供给用户使用的接口，用户可以通过该接口起的操作系统的服务。
    - 程序接口：它是提供给程序员在编程时使用的接口，是用户程序获取操作系统服务的唯一途径。
18. 程序接口：由一组系统调用组成，每一个系统调用都是一个能完成特定功能的子程序。每当应用程序要求OS提供某种服务功能时，便调用具有相应共功能的系统调用。在高级语言以及C语言中，往往提供了与各系统调用一一对应的库函数，这样，应用程序便可通过调用对应的函数库来实现系统调用。但是在近几年的操作系统中，如UNIX，其系统调用本身就是用C语言来编写的，并且以函数形式提供，故在用C语言编写的程序中，可以直接进行系统调用。

2 进程管理

1. 进程的特征和定义：在多道程序环境下，程序的执行属于并发执行，此时它们失去其封闭性，并且具有间断性以及不可再现性的特征。这决定了通常的程序是不能参与并发执行的，因为程序执行的结果不可再现，这样，程序的运行就失去了意义。为了使程序能并发执行，且对并发执行的程序加以描述和控制，人们引入了“进程”的概念。
    - 结构特征：通常的程序是不能并发执行的。为了使得程序(含数据)能够独立运行，应该为之配置一个进程控制块PCB(Process Control Block)。**而由程序段，相关的数据段和PCB三部分便构成了进程实体。**在许多情况下所说的进程，实际上是指进程实体。例如：所谓的创建进程，实质上是创建进程实体中的PCB，而撤销进程，实质上是撤销进程的PCB。
    - 动态性：进程的实质就是进程实体的一次执行过程。因此，动态性是进程的最基本的特征。动态性还表现在：它由创建而产生，由调度而执行，由撤销而消亡。可见，进程实体由一定的生命

期，而**程序则只是一组有序的指令集和，并且存放于某种介质上，其本身并不具有运动的含义，所以是静态的。**

- 并发性：这是指多个进程实体同存在于内存中，并且能在一段时间内运行。并发性是进程的重要特性，同时也是OS的重要特征。引入进程的目的正式为了使其进程实体能和其他进程实体并发运行，而程序(没有建立PCB)是不能并发执行的。
- 独立性：在传统的OS中，独立性是指的进程实体是一个能独立运行，独立分配资源和独立接受调度的基本单位。**凡是没有建立PCB的的程度都不能作为一个独立的单位参与运行**
- 异步性：进程都是按照各自独立的，不可预知的速度向前推进，或者说进程实体按照异步的方式运行。

2. 进程的三种基本状态：

- 就绪状态：当进程已经分配了**除CPU以外**的所有资源后，只要获得CPU，便可以立刻执行。这样的状态叫做就绪状态。在一个系统中**处于就绪状态的进程可以有多个。**它们通常排列成一个队列，称为就绪队列。
- 执行状态：进程已经获得了CPU，其程序正在执行，**在单处理机系统中，只有一个进程处于执行状态。在多处理机系统中，则有多个进程处于执行状态。**
- 阻塞状态：在执行的进程由发生某种事情而暂时无法执行时，便放弃处理机而处于暂停的状态。即进程的执行收到阻塞，把这种暂停状态称为阻塞状态。典型事件：请求IO，申请缓存空间等。注意，**正在运行的进程时间片用完后变为就绪状态，不是阻塞状态**(因为它不缺其他的资源，只是暂时不能用CPU了)。

3. 进程控制块的作用：为了描述和控制进程的运行，系统为每个进程都定义了一个数据结构：进程控制块(PCB)。它是进程实体的一部分，是操作系统中最重要的记录型数据结构。

- PCB中记录了操作系统所需要的，用于描述进程当前情况以及控制进程运行的全部信息。进程控制块的作用是使一个在多道程序环境下不能独立运行的程序(含数据)，成为一个能够独立运行的基本单位。一个与其他进程并发执行的进程。
- OS是根据PCB对并发进行的进程进行控制和管理的。例如：当OS需要调度某一个进程执行时，要从该进程的PCB中查出其现行状态以及优先级。
- 在调度某个进程后，要根据其PCB中所保存的处理机状态信息，设置该进程回复运行的现场。并根据其PCB中的程序和数据的内存地址，找到其程序和数据。
- 进程在执行过程中，当需要和与之合作的进程实现同时，通信或者访问文件时，也要访问PCB。
- 当进程由于某种原因需要暂停执行时，又要将断点的处理机环境保存在PCB中。

4. 在进程的整个生命周期中系统总是通过PCB来对进程进行控制的。系统是根据PCB来感知进程的存在的。所以说，PCB是进程存在的唯一标志。

5. 因为PCB经常被系统访问，尤其是被运行频率很高的进程及分派程序访问，故PCB应该常驻于内存。

6. 原语:是由若干条指令组成的，用于完成一定功能的一个过程。它与一般过程的区别在于：它们是原子操作，原子操作，指的是一个操作中的所有动作要么全都做，要么全都不做。他是一个不可分割的基本单位。因此，在执行过程中不允许被中断。**原子操作是在管态下执行的，常驻内存。**

7. 子进程可以继承父进程所拥有的资源，例如，继承父进程打开的文件，继承父进程所分配的缓冲区等。当子进程被撤销时，应将其从父进程那里获得的资源归还给父进程。此外，在撤销父进程时，也要撤销其所有的子进程。

8. 引起进程创建的事件：

- **用户登录**：在分时操作系统中，用户在终端键入登录命令后，如果合法用户，系统将为该终端建立一个进程，并把它插入道就绪队列中。
- **作业调度**：在批处理系统中，当作业调度程序按照一定的算法调度某作业时，便将该作业装入内存，为它分配必要的资源，并立刻为它创建进程，在插入到就绪队列中。
- **提供服务**：当运行中的用户程序提出某种请求后，系统将专门为其创建一个进程来提供用户所需要的服务。
- **应用请求**：基于应用进程的需求，由它自己创建的一个新进程，使得进程以并发运行方式完成特定的任务。

9. 引起进程终止的事件：

- 正常结束
- 异常结束：越界错误，保护错，非法指令，特权指令错，运行超时，等待超时，算术运送错误(比如除0)，IO故障

- 外界干预：操作员或者是操作系统请求，父进程请求，父进程终止

10. 引起进程阻塞和唤醒的事件：
    - 请求系统服务：当正在执行的进程请求操作系统提供服务时，由于某种原因，操作系统并并不立刻满足该进程的要求，该进程只能转变为阻塞状态来等待。例如：一个进程请求使用某种资源，如打印机，由于系统已经将打印机分配给其他进程而不能分配给请求进程，这时请求者进程只能被**阻塞**，仅在其他进程在释放打印机的同时，才能将进程唤醒。
    - 启动某种操作：当进程启动某种操作后，如果该进程必须在操作完成后才能执行，则必先使该进程阻塞，以等待该操作的完成。
    - 新数据尚未到达：对于相互合作的进程，如果其中一个进程需要获得另一个进程提供的数据后才能对数据进行处理，则只要其所在的数据尚未到达，该进程就只能被阻塞。
    - 无新工作可做：系统往往设置一些具有某特定功能的系统进程，每单那个这种进程完成任务后，便把自己阻塞起来以等待新任务的到来。(听起来有点像自闭....)
11. 进程的阻塞：正在执行的进程，当发现上述的某种事情时，由于无法继续执行，于是进程便通过调用阻塞原语block把自己阻塞住。可见，**进程的阻塞是进程自身的一种行为。**
12. 临界资源：许多的硬件资源，如打印机，磁带机等，都属于临界资源。进程之间应采取互斥的方式，实现对这种资源的共享。
13. 临界区：人们把在每个进程中访问临界资源的那段代码称为临界区。欣然，若能保证进程互斥地进入自己的临界区，便可以实现进程对临界资源的互斥访问。
14. 同步机制应该遵循的规则：
    - 空闲让进：当无进程处于临界区是，表明临界资源处于空闲状态。应该允许一个请求进入临界区的进程立即进入自己的临界区，以有效的利用临界资源。
    - 忙则等待：当自己有进程进入临界区时，表明临界资源正在被访问，因而其他试图进入临界区的进程必须等待，以保证对临界区资源互斥的访问。
    - 有限等待：对要求访问临界资源的进程，应保持在有限的时间内能进入自己的临界区，以避免陷入“死等”状态。
    - 让权等待：当进程不能进入自己的临界区时，应该立即释放梳理及，以免进程陷入“忙等”状态。
15. 管程：代表共享资源的数据结构，以及由对该数据结构实施操作的一组过程 所组成的资源管理程序，共同构成了一个操作系统的资源管理模块，我们称之为管程。管程被请求和释放资源的进程所调用。**一个管程定义了一个数据结构和能为并发进程所执行(在该数据上)的一组操作，**管程由四部分组成：
    - 管程的名称
    - 局部于管程内部的共享数据结构说明
    - 对该数据结构进行操作的一组数据
    - 对局部于管程内部的共享数据设置初始值的语句
16. 管程相当于围墙，它把共享变量和对它进行操作的若干过程围了起来，所有的进程要访问临界资源时，都必须经过管程(相当于通过未将的门)才能进入。而管程每次只允许一个进程进入管程，从而实现了进程互斥。
17. 管程时一种程序设计语言结构成分，它和信号量由同等的表达能力，从语言的角度来看，管程主要有以下的特征：
    - 模块化，管程是一个基本程序单位，可以单独编译。
    - 抽象数据类型，管程中不仅有数据，而且有对数据的操作。
    - 信息掩蔽，管程中的数据结构只能被管程中的过程访问，这些过程也是在管程内部定义的，供管程外的进程调用，而管程中的数据结构以及过程(函数)的具体实现外部不可见。
18. 管程和进程的区别：
    - 虽然二者都定义了数据结构，但是进程定义的是私有数据结构PCB，**管程定义的是公共数据结构，如消息队列。**
    - 二者都存在对各自数据结构上的操作，但进程是由顺序程序执行有关的操作，而管程主要是进行同步操作和初始化操作。
    - 设置进程的目的是在于实现系统的并发性，而管程的设置则是解决共享资源互斥使用的问题。
    - **进程通过调用管程中的过程对共享数据结构实行操作**，该过程就如通常的子程序一样被调用，因而管程为被动工作方式，进程为主动工作方式。

- 进程之间能并发对的执行，**而管程不能与其调用者并发。**
- 进程具有动态性，由创建而诞生，由撤销而消亡，而管程则是操作系统中的一个资源管理模块，供进程调用。

19. 进程通信：目前，高级通信机制可以分为三大类：共享存储器系统，消息传递系统，管道通信系统。
20. 共享存储器系统：在共享存储器系统中，相互通信的进程共享某些数据结构或者共享存储区，进程之间能够通过这些空间进行通信。据此，又可以把它们分成以下两种类型：
    - 基于共享数据结构的通信方式：在这种通信方式中，要求进程公用某些数据结构，借以实现进程之间的信息交换。
    - 基于共享存储区的通信方式：为了传输大量数据，在存储器中划出了一块共享存储区，进程可以通过对共享存储区中的数据读或者写来实现通信。
21. 消息传递系统：程序员直接利用操作系统提供的一组通信命令(原语)，不仅能实现大量数据的传递，而且还隐藏了通信的实现细节，使通信对过程对用户是透明的，从而大大简化了通信程序编制的复杂性，因而获得了广泛的使用。
22. 管道通信：**管道是一种共享文件**，是指的用于链接一个读进程和一个写进程以实现它们通信的一个共享文件，又名pipe文件。向管道文件提供输入的发送进程，以字符流形式将大量数据送入管道，而接收管道输出的接收进程，则从管道中就收数据，由于发送进程和接收进程是利用管道进行通信的，故称为管道通信。
23. 线程的引入：在操作系统中引入**进程**的目的，是为了使多个程序能并发的执行，以提高资源的利用率和吞吐量，**那么在操作系统中再引入线程，则是为了减少程序再并发执行时所付出的时空开销，使OS有更好的并发性。**
24. 由于进程是一个资源的拥有者，因而再创建，撤销和切换中，系统必须为之付出较大的时空开销。在引入线程的操作系统中，通常一个进程都会有若干个线程，至少也会有一个线程。
25. 线程与进程的比较：

	进程	**线程**
调度	资源拥有的基本单位	调度和分派的基本单位
并发性	进程之间可以并发执行	一个进程中的多个线程也可以并发的执行
拥有资源	拥有资源	不拥有资源(当然还是要有一点必不可少的资源)
系统开销	高	低

26. 线程的属性：在多线程操作系统中，通常一个进程中包括多个线程，每个线程都是作为利用CPU的基本单位，是花费最小开销实体。线程具有以下的属性：
    - 轻型实体：线程中的实体基本上不拥有系统资源，只是有一点必不可少，能保证其独立运行的资源。
    - 独立调度和分派的基本单位：在多线程OS中，线程是能独立运行的基本单位，因而也是独立调度和分派的基本单位。
    - 可并发执行：在一个进程中的多个线程之间可以并发的执行，甚至允许在同一个进程中的所有线程都并发的执行。同样的，不同进程中的线程也可以并发的执行。
27. 线程的终止：同进程一样，线程也是有生命周期的，终止线程的方式有两种：
    - 在线程完成了自己的工作后自愿退出。
    - 线程在运行中出现错误或者由于某种原因被其他线程强行终止。
28. 线程之间的同步：貌似和进程差不多
29. 内核级线程：**对于通常的进程，无论是系统进程还是用户进程，进程的创建，撤销以及要求由系统设备完成的IO操作，都是利用系统调用进入内核，在由内核中相应的处理程序予以完成的。**进程的切换同样是在内核的支持下实现的。因此我们说，无论什么进程，它们都是在操作系统的支持下运行的，是与内核紧密相关的。

这里所谓的**内核支持线程**，也是同样在内核的支持下运行的。即使无论是用户进程中的线程，还是系统进程中的线程，他们的创建，撤销和切换等也是依靠内核，在内核空间中实现的，此外，在内核空间还为每一个内核支持线程设置了一个**线程控制块**(听起来好像和PCB有点像)。内核时根据该控制块而感知某线程的存在，并对其加以控制。

30. 内核级线程实现方式的优点：
    - 在多处理器系统中，内核能够同时调度同一个进程中的多个线程并行执行。
    - 如果其中一个线程被阻塞，内核还可以调度该进程中的其他线程占有处理器运行，也可以运行其他进程中的线程。
    - 内核支持线程具有很小的数据结构和堆栈。线程的切换比较快，切换开销小。
    - 内核本身也是可以采用多线程技术。可以提高系统的执行速度和效率。
31. 用户级线程：
    - 用户级线程**仅存在与用户空间中。**对于这种线程的创建，撤销，线程之间的同步与通信功能，**都无须利用系统调用来实现。**对于用户级线程的却换，通常发生在一个应用的诸多线程之间，这时，**也同样无需内核的支持。**由于切换的规则远比进程调度和切换的规则简单，因而使用线程的切换速度特别快。由于这种线程是于内核无关的，用户级线程的任务控制块都在用户空间，而线程所执行的操作也无需内核的帮助，因而内核完全不知道用户级线程的存在。
32. 值得说明的是，**对于设置了用户级线程的系统，其调度仍是以进程为单位进行的**,假如系统中设置的是内核支持线程，则调度便是以线程为单位进行的。(操作系统教材中写的，不过王道考研中貌似没有涉及到这一点。我去，好他喵的纠结。如果真要是碰上这种问题就....好吧，因该就按着教材上写吧)
33. 用户级线程的优点：
    - 线程的切换不需要转换到内核空间，对一个进程而言，其所由的管理数据结构均在该进程的用户空间中，管理线程切换的线程库也在用户地址空间运行。
    - 调度算法可以是进程专用的。在不干扰操作系统调度的情况下，不同的进程可以根据自身需要，选择不同的调度算法对自己的线程进行管理和调度，而与操作系统的低级调度算法是无关的。
34. 用户级线程的缺点：
    - 系统调用的阻塞问题。在基于进程机制的操作系统中，大多数系统调用将阻塞进程，因此，当线程执行一个系统调用时，不仅该线程被阻塞，而其进程内的所有线程都会被阻塞。**而在内核支持级线程方式中，进程中的其他线程还可以运行。**
    - 在单纯的用户级线程实现中，多线程并不能利用处理机的多重处理的有优点，内核每次分配给一个进程的仅有一个CPU，因此进程中仅有一个线程执行。在该线程放弃CPU之前，其他线程只能等待。
35. 用户及线程以内核控制线程链接：
    - 一对一模型：该模型为每一个用户线程都设置一个内核控制线程与之链接，当一个线程被阻塞时，允许调度另一个线程运行。该模型的并行能力较强，但是每次创建一个用户级别线程就要创建一个内核级线程，因此开销较大。
    - 多对一模型，将多个用户级线程映射到一个内核控制线程上，为了管理方便，**这些用户线程一般(注意是一般)属于一个进程**，运行在该进程的用户空间，对这些线程的调度和管理也在该进程的用户空间中完成。当用户线程需要访问内核时，才将其映射到一个内核控制线程上，但每一次只允许一个线程进行映射。该模型的主要优点时管理的开销小，效率高，但是当一个线程在访问内核时发生阻塞，整个进程照样也会被阻塞。**而且，在多处理机的系统中，一个进程的多个线程无法实现并行。**
    - 多对多模型：将多个用户线程映射到多个内核控制线程，内核控制线程的数目可以根据应用进程和系统的不同而变化，可以比用户线程少，也可以与之相同，但是不能比用户线程多(我认为可能是因为多了没有意义)。

3.处理机调度

1. 高级调度，中级调度，初级调度(这里王道上貌似没有涉及，考研貌似也没有涉及)

- 高级调度：又称为作业调度，或者长程调度。其主要功能是根据某种算法，把外存上处于后被队列中的那些作业调入内存，也就是说，它的调度对象是作业。
- 中级调度：又称为中程调度，引入中级调度的主要目的是提高**内存**的利用率和吞吐量。决定哪些进程在内存中。中级调度实际上是存储器管理中的兑换功能。
- 低级调度：通常也把低级调度称为进程调度或者短程调度，它所调度的对象是进程(或者是内核级线程)。

2. 周转时间：**指的是从作业被提交给系统开始，到作业完成的这段时间间隔。**对于每个用户而言，都希望自己的作业周转时间最短，但是作为计算机系统的管理者，则总是希望能使得平均周转时间最短。这不仅会有效的提高系统资源的利用率，而且能让多数用户满意。平均周转时间就是N个周转时间的平均值。

3. 响应时间：响应时间是指从用户通过键盘提交第一个请求开始，直到系统首次产生响应为止的时间。这是分时操作系统中进程调度算法的重要准则之一。

4. 带权周转时间：就是周转时间T与为它提供服务时间Ts的比值，即为W=T/Ts。平均带权周转时间就是所有带权周转时间的平均值。

5. 先来先服务算法(FCFS)：顾名思义，先到先得。最简单的一种调度算法，该算法既可以用于作业调度，也可以用于进程调度。

   - 先来先服务算法**有利于长作业，不利于短作业。**
   - 先来先服务算法有利于CPU繁忙型的作业，而不利于IO繁忙型的作业。通常科学计算属于CPU繁忙型作业，目前大多数事务处理都属于IO繁忙型作业。

6. 短作业有限(SJF)算法：对于短作业或者短进程优先调度的算法。短作业有限算法能够有效的降低平均作业等待时间，提高系统的吞吐量。但是有以下几个缺点：

   - 该算法对于长作业不利，而且肯能会导致长进程长期不会被调度。
   - 该算法完全没有考虑作业的紧迫度，不能保证紧迫性的进程会被及时处理。
   - 由于作业的长短只能是根据用户所提供的执行时间而顶的，而用户有可能有意或者无疑的缩短其作业的估计运行时间，导致该算法不一定真正的做到短作业优先。

7. 非抢占式优先权算法和抢占式优先权算法：简单的说就是为每一个作业设置一个优先级，优先级高的作业先运行，非抢占式算法中，一个进程如果获得了处理机，就会一直运行到其结束，即使中途有更高优先级的进程需要调度也不会放弃处理机，抢占式算法就是如过新来的进程优先级更高，正在运行的进程就会放弃处理机，让给新来的进程，新来的的进程就要挂在就绪队列上，等待处理机。(这个算法王道考研中没有涉及，貌似真的考研中也没有涉及)

8. 高响应比优先算法：相应比：(等待时间+要求服务时间)/要求服务时间。相应比高的进程优先得到调度。高响应比优先算法的特点：

   - 等待时间相同时，短作业优先被调度。所以该算法对短作业有利
   - 当服务时间相同时，等待时间长的作业优先被调度，等待时间越长，优先级越高，以此来实现先来先服务。
   - 对于长作业，随着等待的时间增加，其优先级也逐渐提高，从而获得处理机。

9. 高响应比优先算法照顾了短作业，又考虑了作业到达的先后顺序，不会使得长作业一直得不到服务，因此该算法是一个比较好的折中，当然在利用此算法时，都要先计算响应比，这会增加系统开销。

10. 时间片轮转法：简单的说，就是把所有的作业都排成一个队列，每一次都为队头的作业服务一段时间。如果队头的作业被处理完成，就离开队列，处理下一个作业，否则就把这个作业从处理机上撤下，插进队尾，处理新的队头作业,这样周而复始....这个算法的时间片的大小对性能影响很大，如果时间片很小，有利于短作业，如果，但是会频繁的发生中断，切换进程，从而加大系统的开销，然是如果时间片很长，这个算法的退化成先来先服务算法....一个较为合理的时间片大小时，时间片略大于一次经典交互所需要的时间。

11. 多级反馈队列调度算法(我去，这个算法貌似有点复杂):

    - 首先设置多个就绪队列，编号为q1，q2，q3....qn，其中q1的优先级最高，q2的优先级第二…qn的优先级最低。其中系统分配q1队列中的任务时间片最短，q2的时间片比q1长一倍，q3的时间片比q2长一倍....(教材和王道考研中的时间片长度是按照*2的关系给的)
    - 当一个新的进程来到内存中时，先把它插入到q1的队尾，然后系统先为q1对头的作业服务，如果在q1的时间片内完成了，就调出队列，服务q1中下一个作业。如果没有完成，就调入q2的队尾，服务q1中的下一个作业....

- 假设q1中的作业全都服务过一遍，现在q1为空，系统便开始服务q2中的作业，和刚才一样，首先服务q2队头的作业,完成就出队，否则就挂在q3的队尾…
- 假设q2中的作业全都服务过一边，现在q2为空，q1也为空(注意，如果某时刻又来了新的作业，也要挂在q1上，这个进程会抢占处理机，然后再按照刚才的逻辑执行)，系统便开始服务q3中的作业....

12. 多级反馈队列调度算法的性能：
    - 终端型作业用户：由于终端型作业用户提交的作业大多数都属于交互性作业，作业通常比较小，系统只需要能使这些作业在第一队列所规定的时间内完成，变可以使终端作业用户都满意。
    - 短批处理作业用户：对于很短的批处理型作业，开始时像终端作业一样，如果仅仅在第一队列中执行一个时间片即可完成，便可获得与终端型作业一样的响应时间。对于稍长的作业，通常也只需在第二队列和第三队列各执行一个时间片即可完成，其周转时间仍然较短。
    - 长批处理作业用户，对于长作业，它将依次在1，2，3，4…n个队列中运行，然后再按轮转的方式运行，不用担心得不到处理。
13. 死锁： 产生死锁的原因可以归结为以下两点：
    - 当系统中提供多个进程共享的资源，如打印机，公用队列等，其数目不足以满足诸多进程的需要时，会引起进程对资源的竞争而产生死锁。
    - 进程间的推进顺序非法性。进程在运行过程中，请求和释放资源的顺序不当，也同样会导致产生进程死锁。
14. 产生死锁的必要条件：
    - 互斥条件：指进程对所分配的资源进行排他性使用，即在一段时间内某资源只能由一个进程占用，如果此时还有其他进程请求该资源，则请求者只能等待。直至占有该资源的进程用毕释放。
    - 请求和保持条件：指进程已经保持了至少一个资源，但是又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程阻塞，但是又对自己获得的其他资源保持不放。
    - 不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时自己释放。
    - 环路等待条件：后一个等待前一个，后一个等待前一个....第一个等待最后一个....
15. 处理死锁的基本方法：
    - 预防死锁：这是一种较为简单和只管的事先预防的方法，该方法是通过设置某些限制条件，去破坏产生死锁的四个必要条件中的一个或者几个，来预防死锁。预防死锁是一种比较易于实现的方法，已经被广泛使用。但是由于限制条件往往太严格，因而会导致系统资源利用率和系统吞吐量降低。
    - 死锁避免：该方法同样是属于事先预防的策略，单是并不是事先采取各种限制措施去破坏死锁的四个必要条件，**而是在资源的动态分配中，用某种方法去防止系统进入不安全状态，从而避免死锁。**这种方法只需事先施加较弱的限制条件，便可获得较高的资源利用率及系统吞吐量，但在实现上有一定的难度。目前在较完善的系统中常用此方法来避免发生死锁。
    - 检测死锁：**这种方法并不须事先采取任何限制性措施，也不必检查系统是否已经进入不安全区，而是允许系统在运行过程中发生死锁。**但可通过系统所设置的检测机构，及时地检测出死锁的发生，并精确地确定与死锁有关的进程和资源； 然后，采取适当措施，从系统中将已发生的死锁清除掉。
    - 解除死锁:**这是与检测死锁相配套的一种措施。当检测到系统中已发生死锁时，须将进程从死锁状态中解脱出来。**常用的实施方法是撤消或挂起一些进程，以便回收一些资源，再将这些资源分配给已处于阻塞状态的进程，使之转为就绪状态，以继续运行。死锁的检测和解除措施有可能使系统获得较好的资源利用率和吞吐量，但在实现上难度也最大
16. 银行家算法是**死锁避免算法**

(其实感觉这部分的知识点比较成体系，很连贯，需要总结的东西也不多)

4.存储器管理

1. 高速缓存与寄存器的引入：CPU与外围设备交换的信息一般也依托于主存储器的地址空间，由于主存储器的访问速度远低于CPU执行指令的速度，为了缓和这一矛盾，在计算机系统中引入了寄存器和高速缓存。

- 寄存器：寄存器的访问速度最快，完全可以和CPU协调工作，但是非常昂贵，容量也比较小。
- 高速缓存：访问速度大于主存小于寄存器，容量大于寄存器，小于主存。
- 磁盘缓存：目前的磁盘IO速度远低于对主存的访问速度，因此将频繁使用的一部分磁盘数据和新消息，暂时存放在磁盘缓存中，可以减少访问磁盘的次数。

2. 将一个用户源程序变为一个可以在内存中执行的程序：

- 首先要编译：又编译程序将用户源代码编译成若干个**目标模块。**
- 其次是链接：由链接程序将编译后形成的一组目标模块，以及它们所需要的库函数链接在一起，形成一个完整的**装入模块。**
- 最后是装入：由装入程序将装入模块装入内存。

3. 将一个装入模块装入内存中时，由绝对装入，可重定位装入，动态装入方式。

- 绝对装入：在编译时，如果知道程序将驻留在内存的什么位置，那么，编译程序将产生绝对地址的目标代码。绝对装入程序按照装入模块中的地址，将程序和数据装入内存，装入模块被装入内存后，由于程序中的逻辑地址与实际内存地址完全相同，故不须对程序和数据的地址进行修改。(只适用于单道程序环境)
- 可重定位装入：**在多道程序环境下，编译程序不可预知所编译的目标模块应该放在内存的何处，因此，绝对装入方式只适用于单道程序环境。**在多道程序环境下，所得到的目标模块的起始地址通常由0开始，程序中的其他地址也都是相对于起始地址计算的。此时应该采用可重定位装入方式，根据内存的当前情况，将装入模块装入到内存中的适当位置。在可重定位装入方式中，装入模块中的逻辑地址与实际地址可能时不同的。(适合多道程序，但是程序运行时不能再内存中移动)
- 动态运行时装入：**可重定位装入方式可将装入模块装入到内存中的任何位置，所以可以用于多道环境，但是这种方式不允许程序运行时再内存中移动。**但是，实际情况时，再运行过程中的程序可能再内存中的位置可能会经常改变，此时就应该采取动态运行时装入方式。**动态运行时装入程序再把装入模块装入内存后，并不立刻把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正要执行时才进行。**因此，装入内存后的所有地址仍是相对地址。为了使地址转换不影响指令的执行速度，这种装入方式需要一个重定位寄存器的支持。

4. 程序的链接：源程序经过编译后，可得到一组目标模块，再利用连接程序将这组目标模块链接，形成装入模块，根据链接时的不同，可分为三种：(这一部分教材就是放在装入后介绍的，不过我感觉按照顺序因该是先链接后装入啊.....)

- 静态链接：再程序装入之前，先将个目标模块按照它们所需要的库函数，链接称为一个完整的装入模块，以后不再拆开。我们把这种事先进行链接的方式称为静态链接。
- 装入时动态链接：这是指将用户源程序编译后多得到的一组目标模块，在装入内存时，采用边装入边链接的方式。
- 运行时动态链接：这是指对某些目标模块的链接，是在程序执行中需要该模块时，才对它进行的链接。

5. 动态分区分配中的算法：首次适应算法，循环首次适应算法，最佳适应算法，最坏适应算法，快速适应算法。这三种方法属于连续分配的管理方式。

- 首次适应算法：要求空闲分区连的地址按照地址的次序链接，在分配内存时，从链首开始顺序查找，直到找到一个大小能满足要求的空闲分区为止。然后再按照作业的大小，从该分区中划分出一块内存空间分配给请求者，余下的空闲分区仍留在空闲链中。**该算法倾向于优先利用内存中低地址部分的空闲分区，从而保留了高地址部分的大空闲分区，这为给提后到达的大作业分配大的内存空间创造了条件。**其缺点是低地址部分被不断地划分，会留下很多难以利用，很小地空闲分区，而每一次查找都是从低地址部分开始，这无疑会增加查找可用空闲分区时的开销。
- 循环首次适应算法：该算法从首次适应算法演变而来，再为进程分配内存空间时，不再是从链首部开始查找，**而是从上一次找到空闲分区的下一个空闲分区开始查找(注意，是下一个)。**直到能找到一个满足要求的空闲分区，从中画出一块请求大小相等的内存空间分配给作业。**该算法能使内存中的空闲分区分布的更均匀，从而减少查找空闲分区时的开销，但是这样会缺乏大的空闲分区**
- 最佳适应算法：指的是每一次为作业分配内存时，总能把满足要求，又是最小的空闲分区分配给作业，避免大材小用。**孤立的看，最佳适应算法似乎是最佳的，但是宏观上来看却不一定，因为每一次分配后，所切割下来的剩余部分总是最小的，这样再存储器中总会留下许多很难利用的小空间。**
- 最坏适应算法：和最佳适用算法相反，这个算法每一次都会找出一个最大的分区来分配给作业。**这样的优点是剩余的空闲区不至于太小，产生碎片的几率反而是最低的。**

- 快速适应算法(王道书中没有涉及)：该算法又称为分类搜索法，是将空闲分区根据其容量大小进行分类，对于每一类具有相同容量的所有空闲分区，单独设立一个空闲分区链表，这样，系统中存在多个空闲分区链表，同时在内存中设立一张管理索引表，该表的每一个表项对应了一种空闲分区类型，并记录了该类型空闲分区链表表头的指针。该算法的优点是查找效率高，仅需要根据进程的长度，寻找到能容纳它的最小空闲区链表，并取下第一块进行分配即可。

6. **内部碎片和外部碎片**：我去，感觉王道书上说的也比较模糊(当然更可能是因为我理解能力太差)。这里按照我的理解说一下：
   - 内部碎片：一般在固定分配内存中产生，比如说固定分配100kb的内存块，即使作业实际只用到了1kb的内存，但是也会给该作业分配100kb的内存块。这99kb剩下的空间虽然永远不用，但是不允许其他作业使用，这99kb的内存就是**内部碎片**。
   - 外部碎片：一般在动态分配中产生，比如作业需要用到10kb的内存，系统就会分配正好10kb的内存。这时就完全没有内部碎片了，但是考虑这样的一个情况，现在又三个作业(都是动态分配内存)，第一给分配1000kb，第二个分配10kb，第三个分配1000kb(假设内存空间是连续分配)。现在第二个作业运行很快，马上就释放了这10kb的内存，但是因为这一块内存实在是太小了，以后也可能很难利用到，这就形成了**外部碎片**。
7. 基本分页存储管理方式：连续分配方式会形成许多的“碎片”，虽然可以通过“紧凑”方法 将许多碎片拼接成可用的大块空间，但须为之付出很大开销。如果允许将一个进程直接分散地装入到许多不相邻接的分区中，则无须再进行“紧凑”。基于这一思想而产生了离散分配方式。如果离散分配的基本单位是页，则称为分页存储管理方式；如果离散分配的基本单位是段，则称为分段存储管理方式。(这里可以看出，分页管理与刚才的连续管理的”地位”是相同的，都是用来管理为作业分配内存)

   在分页存储管理方式中，如果不具备页面对换功能，则称为基本的分页存储管理方式，或称为纯分页存储管理方式，它不具有支持实现虚拟存储器的功能，它要求把每个作业全部装入内存后方能运行。
8. 页面和物理块：分页存储管理是将一个进程的逻辑地址空间分为若干个大小相等的片，称为页面或者页，相应的，也把内存空间分分成与页面相同的若干个存储块，称为(物理)块或者页框。由于进程的最后一页经常装不满而形成了不可利用的碎片，称之为“页面碎片”(其实就是刚才说的内部碎片)。
9. 分页地址中的地址结构：由两部分组成，分别是页号和偏移量
10. 页表：在分页系统中，允许将进程的各页离散的存储在内存中的不同物理块中，**但是系统能保证进程正确的运行，即能在内存中找到每个页面所对应的物理块。**为此，系统又为每个进程建立了一张页面映像表，简称页表。进程在地址空间中的所有页，依次在页表中有一个**页表项**。其中记录了相应页在内存中对应的物理块号。可见，**页表的作用是实现先从页号到物理块号的地址映射。**
11. 快表：由于页表使存放在内存中的(书上说通常都是在内存中)，这使得CPU在每一次额存取一个数据时，都需要访问两次内存。(第一次访问页表，拼接出来实际地址，第二次通过实际地址来读取数据)这使得计算机处理速度降低近1/2，是得不偿失的。为了提高地址变换速度，引入”快表”。快表是一个有并行查询能力的特殊高速缓冲寄存器。用来存放当前访问的那些**页表项(注意，是页表项)**。引入快表后，进行地址变换时，首先会访问快表，如果快表有该地址所对应的页表项，就直接拼接地址，没有的话就再访问页表，拼接地址(注意，也有的题目是同时访问页表和快表，这个看具体情况而定)，同时再把这个页表项存在页表中。(这里描述的貌似和Cache有点像，但是注意这两个本质上可是不同的，快表是为了加速拼接地址，避免在拼接地址时频繁访存，里面存放的是页号和物理地址的映射关系。Cache里面存放的是一整块数据，为了是在访问数据是避免频繁访存。)
12. 两级页表和多级页表：主要是为了解决单个页表过大的问题。现在来说明一下多级页表是怎么节省空间的。现在有逻辑为32位，其中页面偏移量是后12位。
    - 在单页表的地址结构中，前20位是页号，每一个进程的页表的页表项有2^20个，在运行该进程时，这个包含着2^20个页表项的超级大页表就会放在内存中，非常占用空间。
    - 现在使用二级页表地址结构，逻辑地址的前10位为一级页号，中间10位为二级页号，现在的页表变为：现在又一个一级页表，其中又2^10个“页表项”，记录着每一个一级地址引出的二级页表的地址。而每一个一级页表引出的而二级页表的页表项也有2^10个，其中就记录着拼接出来的页号对应的物理地址。现在运行该进程时，只有在**需要时**才会将部分页表调入内存，其他的页表还在外存上。(需要调入一个一级页表，在根据需要调入需要的二级页表，这样总共只有2^10+2^10个页表项，比2^20个页表项少了很多)
13. 基本分段存储管理方式：为了满足用户和程序员的一系列需求：

- 方便编程：通常，用户把自己的作业按照逻辑划分成若干个段，每一个段都是从0开始编址，并且有自己的名字和长度。因此，希望要访问的逻辑及地址是由段名和段内偏移量决定的。
- 信息共享：实现对程序和数据的共享时，式以信息的逻辑单位为基础的，比如某个例程和函数，分页系统中的也只是存放信息的物理块，没有完整的意义，不利于实现共享。而段是信息的**逻辑单位**。
- 信息保护：信息保护同样是对信息的逻辑单位进行保护，因此，分段管理方式能有效和方便的实现信息保护。
- 动态增长：在实际应用中，往往有些段，特别是数据段，在使用中会不断地增长，而事先不能直到数据段会增长到多大，前面的存储方式，很难应付这种动态增长的情况。
- 动态链接：动态链接是指在作业运行之前，并不把几个目标程序链接起来，要运行时，陷阱主程序所对应的目标程序装入内存并启动运行，只有当运行过程中需要调用到某些段时，才将该段调入内存并进行链接。可见，动态链接也要求以段作为管理的单位。

14. 分段地址的形式：段号+段内地址。 段表：和页表类似，里面记录了从段号到实际物理地址的映射关系，除此以外，还记录了段的长度(因为段的大小都是可能不同的)。

15. 段页式存储器：分页系统能有效的提高内存的利用率，分段系统能更好的满足用户的需求，为了结合两种管理方式的优点，引出段页式存储器。段页式系统是分段与分页的相结合，**先将用户程序分成若干个段，在把若干个段分成若干个页，并且为每一个段赋予一个段名。**简单的说就是先把程序进行分段，而每个段的内部再分页(个人理解：页的大小都是相同的，所以显然段的大小都是页的大小的整数倍,普通的分段存储堆对于段的大小没有特别的要求，所以可能会造成管理的困难，现在的段的大小有了一定的要求，但是)。

16. 段页式地址结构：段号+段内页号+页内地址。在段页式存储器中，为了获得一个数据需要访存三次，第一次访问段表，获得页表的地址，第二次访问对应的页表，拼接出来物理地址，第三次才能访问数据。

17. **虚拟**存储器：前面的各存储器的管理方式有一个共同的特点，就是它们都要求一个作业全部装入内存后才能运行，于是，就出现了下面两种情况：
    - 作业很大，已经超过了内存空间中的总容量。作业不能全部装进内存，使该作业无法运行。
    - 有大量的作业要运行，但是因为内存的容量不足以容纳所有的作业，只能将少数的作业装入内存让他们先运行，而将其他大量的作业留在外存上等待。
18. 常规存储器管理方式的特种：
    - 一次性：就是运行作业必须要一次性对的把作业的全部都装入内存。
    - 驻留行：作业装入后，便一直的驻留再内存中，直到作业结束。即使是某些进程会因为IO进行长时间的等待，即使是某些模块运行一次后就再也不会被用到。
19. 局部性原理：
    - 时间局部性：如果某条指令一旦被执行，则不久之后该指令还有可能再次被执行。如果某数据被访问过，则不久后该数据可能被再次访问过。产生时间局部性的典型原因是因为由于程序中有大量的循环操作。
    - 空间局部性：一旦程序的某一个存储单元被访问过，在不久后，其附近的存储单元也将再次被访问。在一段时间内所访问的地址，可能集中在一定范围内，其典型情况就是程序的顺序执行。
20. 虚拟存储器的实现方法：在虚拟存储器中，允许将一个作业**分多次**调入内存。

21. 分页请求系统：
    - 分页请求系统(我去，我记得怎么是请求分页系统? 难道是PDF错了?啊这…不过也都一样)：这时在**分页系统**的基础上，增加了请求调页面和页面置换功能所形成的页式虚拟存储系统，它允许只装入少数页面的程序，便启动运行。以后，再通过调页功能以及页面置换功能，陆续地将把即将要运行地页面调入内存，同时把暂时不运行地内存再调出内存到外存上，置换时以**页面**为单位。
      硬件支持：请求分页的页表机制，缺页中断机构，地址变换机构。
22. 分段请求系统：
    - 请求分段系统：这是再**分段系统**的基础上，增加了请求分段以及分段置换功能后所形成的段式虚拟存储系统。它允许只装入少数段的用户和程序，便可以启动。(听起来和分页貌似差不多)。以后在通过调段功能和段的置换功能将暂时不能运行的段调出，同时调入即将运行的段。段的置换是以**段**为单位进行的。
      硬件支持：请求分段的段表机制，缺段中断机构，地址变换结构(听起来貌似和上面也差不多)

23. 虚拟存储器的特征：

- 多次性：多次性是指一个作业被分成多次调入内存运行，亦即在作业运行时没有必要将其全部装入，只需将当前要运行的那部分程序和数据装入内存即可；以后每当要运行到尚未调入的那部分程序时，再将它调入。多次性是虚拟存储器最重要的特征，任何其它的存储管理方式都不具有这一特征。**因此，我们也可以认为虚拟存储器是具有多次性特征的存储器系统。**
- 对换性：对换性是指允许在作业的运行过程中进行换进、换出，亦即，在进程运行期间，允许将那些暂不使用的程序和数据，从内存调至外存的对换区(换出)，待以后需要时再将它们从外存调至内存(换进)；甚至还允许将暂时不运行的进程调至外存，待它们重又具备运行条件时再调入内存。**换进和换出能有效地提高内存利用率。可见，虚拟存储器具有对换性特征。**
- 虚拟性：虚拟性是指能够从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际内存容量。这是虚拟存储器所表现出来的最重要的特征，也是实现虚拟存储器的最重要的目标。值得说明的是，虚拟性是以多次性和对换性为基础的，或者说，仅当系统允许将作业分多次调入内存，并能将内存中暂时不运行的程序和数据换至盘上时，才有可能实现虚拟存储器；而多次性和对换性又必须建立在离散分配的基础上。

24. 请求分页中的页表机制：

- 在请求分页系统中所需要的主要数据结构是页表，其基本作用仍然是将用户空间中的逻辑地址变成内存中的物理地址。但是还要再加入若干项，供程序在换进，换出时参考。在请求分页系统中的每个页表项如下所示：
- 页号-物理块号-状态位P-访问字段A-修改位M-外存地址
- 状态位P：用于指示该页是否已经调入内存，供程序访问时参考。
- 访问字段A：用来记录本页在一段时间内被访问的次数，或者记录本页最近已经有多长时间未被访问，供选择换出页面时参考。
- 修改为M：表示该字段调入内存后有没有被修改过。
- 外存地址：用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

25. 缺页中断：在请求分页系统时，每当所要访问的页面不再内存时，便会产生缺页中断，请求OS将所缺的页调入内存。它和一般的中断相比，有明显的区别：

- 在指令执行期间产生和处理的中断信号，通常，CPU都是在一条指令执行完后，才检查是否有中断请求到达。若有，便去响应，否则就继续执行下一条指令。然而，缺页中断时在指令执行期间，发现索要访问的指令或数据不在内存时产生和处理的。
- 一条指令在执行期间，**可能会产生多次缺页中断**(比如，某一条指令需要处理的数据横跨好几个页面)。

26. 物理块的分配策略：

- 固定分配局部置换：**为整个进程分配一定数目的物理块，在整个运行期间不再改变。**在该策略中，如果进程在运行中发生了缺页，则只能从该进程的n个页面中选择一个页面换出，然后再调入一页，保证分配给该进程的内存空间不变。(按我的理解，这里的固定值得就是物理块的固定，局部置换指的是置换物理块上的数据，而不是置换物理块)
- 可变分配全局置换：**先为该进程分配一定数目的物理块，而OS自己也有一个空闲物理块的队列。**当某进程发现缺页时，系统会从空闲队列中调出一个空闲物理块给进程，然后把缺的页放入这个新的物理块。只要发生缺页就重复这个过程，直到空闲队列中的物理块用完，再从内存中选择一个页调出。(按照我的理解，这里的可变指的是物理块数目的可变，全局置换指的是一旦缺页，直接加物理块，不再是先把旧的页面调出，再把新的页面调入)
- 可变分配局部置换：首先为进程分配一定数目的物理块，当进程发生缺页时，只允许进程从内存中的页面中换出一页，这样不会影响其他的进程。只有当频繁发生缺页时，才为该进程分配新的物理块。如果这个进程的缺页率特别低，还会收回这个进程的部分物理块。(按照我的理解，这里的可变指的是物理块数目的可变，局部置换指的还是置换物理块上的数据。)

27. 页面置换策略：

- 先进先出算法(FIFO)：顾名思义，换出最早进入的页面。会产生贝拉迪异常。就是分配的物理块增加，反而缺页率变高了。
- 最近最久未被使用(LRU)算法：换出最久没有被使用过的页面。
- 最少使用置换算法：换出最近时期使用次数最少的页面。
- 简单clock置换算法
- 改进clock置换算法

28. 简单/改进clock置换算法：这个算法的描述在教材和王道上面都有，描述起来还是比较复杂的。这里按照我对书上的描述以及我的理解写出一个代码的描述：

```
//简单clock算法
struct node{
    Data//页面中的数据
    bool flag;//访问位。如果被访问过，就置为1，如果没有被访问过，就置为0
};

const int maxn=10;//指的是为某个进程固定分配某个数量的物理块
node page[maxn];

void change_algorithm_1(){//简单置换算法
    //这个算法一共需要循环的检测两次
    for(int i=0;i<maxn;i++){
        if(page[i].flag==0){
            change(page[i]);//将此页面换出，算法结束
            return;
        }else{
            page[i].flag=0;//否则，将该页的访问位置为0
        }
    }

    for(int i=0;i<maxn;i++){//这一次必有一个页面被换出
        if(change==0){
            changr(page[i]);
            return;
        }
    }
}
//改进clock算法
struct node{
    Data//数据部分
    bool A;//访问标志
    bool M;//修改标志
};

const int maxn=10;

node page[maxn];

void change_page_algorithm2(){

    while(true){//注意，这可不是死循环，这个循环最多执行两次
        for(int i=0;i<maxn;i++){
            if(page[i].A==0&&page[i].M==0){
                change(page[i]);
                return ;
            }
        }

        for(int i=0;i<maxn;i++){
            if(page[i].A==0&&page[i].M==1){
                change(page[i]);
                return ;
            }else{
                page[i].A=0;
            }
        }
    }
}
```

```
59 //还有一个小细节，整个算法的执行过程中，不会改变M(修改标志)的值
```

1. 抖动(教材中貌似没有涉及)：在页面置换过程中，一种最糟糕的情况是，刚刚换出的页面马上又要换入主存，刚刚换入的页面马上要放出主存。这种频发的页面调度称为抖动，或者叫颠簸。主要原因是进程频繁访问的页面数多于可用的物理块数。
2. 工作集与驻留集(教材上貌似也没有涉及)：工作集就是指在某一个时间间隔内，进程要访问的页面集合。驻留集就是系统分配给进程的物理块数。

5 文件管理(这一个章节主要参考了天勤操作系统中的描述)

1. 文件的**逻辑**结构：文件的逻辑结构是从用户观点来看所观察到的文件的组织形式。是用户可以直接处理的数据及其结构。
   - 顺序文件：顺序文件又称为连续结构，是最简单的文件结构，其将一个逻辑文件的信息连续存放，以顺序结构存放的文件称为顺序文件或者连续文件。顺序文件的主要优点是顺序存取时速度较快，**如果文件时定长记录文件，还可以根据文件的起始地址以及记录的长度随机访问。**但是因为文件存储要求连续的存储空间，所以会产生碎片，同时页不利于文件的动态扩充。
   - 索引文件：索引结构为一个逻辑文件的信息建立一个索引表，索引表中的表目存放文件记录的长度和所在逻辑文件的起始位置，因此逻辑文件中不再记录的长度信息，**索引表本身是一个定长文件**，每个逻辑块是可以变长的。索引表和逻辑文件两者构成了索引文件。**索引文件的优点是可以随机访问，也利于文件的增删。**但是索引表的使用增加了存储空间的开销，另外，索引表的查找策略对文件系统的效率影响很大。
   - 索引顺序文件：索引顺序文件是顺序文件和索引文件的结合，索引顺序文件将顺序文件中的所有记录分为若干个组，为顺序文件建立一张索引表，**并且为每一组中的第一个(注意，是第一个，不是所有)记录在索引表中建立一个索引项，其中含有该记录的关键字和指向该记录的指针。**索引表中包含关键字指针和指针两个数据项，索引表中的索引项按照关键字顺序排列，索引顺序文件的逻辑文件是一个顺序文件，每个分组内部的关键字不必有序排列，但是组与组之间的关键字是有序排列的。
   - 直接文件和散列(hash)文件：建立文件的关键字和对应的物理记录之间的关系，也就是说关键字决定其物理地址。这种结构称为直接文件，没有顺序特性。散列文件是一种典型的直接文件，通过散列函数对关键字进行转换，转换的结构直接决定记录的物理位置。散列文件有很高的存取速度，但是因为不同的关键字可能对应相同的散列函数值而引起冲突。
2. 文件控制块(FCB)：从文件管理的角度来看，文件由文件控制块和文件体两部分组成，文件体就是文件的本身，**文件控制块是保存文件属性信息的数据结构。**它包含的信息因为操作系统而差异，但是至少包含以下的信息：
   - 文件名：该信息用于表示一个文件的符号，每一个文件都具有一个唯一的名字，这样用户可以按照文件名对文件进行操作。
   - 文件的结构：该信息说明文件的逻辑结构是记录式文件还是流式文件，若为记录式文件还需要进一步说明记录是否定长，记录长度和个数，说明文件的物理结构是顺序文件还，索引文件，还是顺序索引文件。
   - 文件的物理位置：该信息用来指示文件在外存上的存储位置，包括存文件的设备名，文件在外存的存储地址以及文件的长度等
   - 存取控制信息：该信息用于只是文件的存取权限，包括文件的拥有者的存取权限以及文件主同组用户的权限和其他一般用户的权限。
   - 管理讯息：该信息包括文件的建立日期以及时间，上次存取文件的日期和当前文件的使用状态的信息。
3. 索引节点：在检索目录文件的过程中，**只用到了文件名，仅当找到了匹配的目录项时，才需要从该目录项中读取出该文件的物理地址。**也就是说，在检索目录时，文件的其他描述信息时不会被用到的，因而也不需要调入内存。因此，有些系统采用了文件名与描述信息分开的方法，将文件的描述信息单独形成一个索引节点，简称i节点。文件目录中的每一个目录项仅由文件名和指向给该文件i节点的指

针构成。存放在磁盘上的索引节点称为磁盘索引节点，每一个文件都由唯一一个磁盘索引节点，其中主要包括以下内容：

- 文件主标识符：拥有该文件的个人或小组的标识符。
- 文件类型：包括普通文件，目录文件或者特别文件
- 文件的存取权限：各类用户对该文件的存取权限
- 文件的物理地址：每个索引节点间接或者直接的方式给出数据文件所在的盘块的编号。
- 文件长度：以**字节**为单位的文件长度。
- 文件链接计数：表明在本文件系统中所有指向该文件的文件名的指针数。
- 文件存取时间：本文件醉最近被存取的时间，最近被修改的时间以及索引节最近被修改的时间。

4. 当文件比打开时，磁盘索引节点被复制到内存索引节点中，以便使用。存放在内存中的索引节点被称为**内存索引节点**，其**增加**了以下的内容

- 索引节点编号：用于标识内存索引节点。
- 状态：指示i是否上锁或者被修改。
- 访问技术：正在访问文件的进程数。
- 逻辑设备号：文件所属文件系统的逻辑设备号
- 链接指针：设置分贝指向空闲链表和散列队列的指针。

5. 基于索引节点的共享方式(硬链接)：索引节点把FCB中的文件描述信息单独构成一个数据结构，也就是说，物理块的信息在索引节点中。此时，目录项中只有文件名和指向文件索引节点的指针。两个不同的目录项只需要指向相同的索引节点就可以实现共享。在索引节点中再增加一个计数值来统计指向该节点的目录项的个数，这样一来就需要再删除该文件时可以先判断计数值，**只有在计数值为1时才能删除该节点。**这种方法可以实现文件的异名共享，但是当多个用户共享时，文件的拥有者不能删除该文件。

6. 利用符号链接实现的文件共享(软连接)：该方法是创建一个**称为链接**的新目录项，例如，为了使用户b能共享用户c的一个文件，可以由此系统为用户b建立一个指向该文件的新目录项，并放在用户b的目录下。新的目录下个中包含了被共享文件的路径名，可以使绝对路径也可以是相对路径。

- 在利用符号链接方式实现文件共享时，只有文件的拥有者才拥有指向其索引节点的指针。而文件对的共享者和其他用户只有该文的路径名，并不拥有指向其索引节点的指针。这样就不会发生文件拥有者删除共享文件后留下悬空指针的情况。当文件的拥有者把一个共享文件删除后，其他用户试图通过符号链接去访问一个已经被删除的文件时，系统就会因为找不到该文件而访问失败，于是再将符号删除。
- **符号链接方式有一个很大的优点，就是它能够用于链接(通过计算机网络)世界上任何计算机中的文件**，此时只需呀提供该文所在机器的网络地址以及文件在该机器中的路径即可。
- 该方法解决了基于索引节点共享方式中文件的拥有者不能删除共享文件的问题，但是其他用户在访问共享的文件式，要逐层查找目录，开销较大。

7. 文件保护：

- 访问类型：对文件的保护可以从限制文件的访问类型出发，可以加以控制的访问类型有读，写，执行，添加，列表清单等。此外，还可以对文件的重命名，复制，编辑等加以控制。
- 访问控制：访问控制就是对不同的用户访问同一个文件采取不同的访问类型。根据用户的权限不同，可以把用户分为拥有者，工作组用户，其他用户等。然后对不同的用户组采取不同的访问类型，以防止文件被非法访问。

8. 目录的实现：

- 线性表：最为简单的目录实现方法就是使用存储文件名和数据块的线性表。创建新文件时，必须首先搜索目录表以确定没有同名的文件的存在，接着在目录表后增加一个目录项，若是要删除文件，根据给定的文件名搜索目录表，接着释放分配给它的空间，采用链表结构可以减少删除文件的时间，其优点在于实现简单，但是线性表需要采用顺序查找的方法来查找特定的目录项，所以运行比较耗时。
- 散列表：散列表时根据文件名得到一个值，并返回一个指向线性表中元素的指针。这种方法可以大大缩短查找的时间，插入和删除也比较简单，但是要采取一些措施来防止冲突。

9. 文件的实现：文件的实现主要是指文件在存储器上的实现，即为文件的物理结构的实现，包括外存的分配方式和文件的存储空间。

10. 外存的分配方式-连续分配(注意，这里是物理结构，分配方法和上面说过的逻辑结构不同)：连续分配是最简单的磁盘空间分配策略。该方法要求为文件分配连续的磁盘区域，在这种分配算法中，用户必须在分配之前说明待创建文件所需要的存储空间的大小，然后系统查找空闲区的管理表格，查看是否有足够大的空心区供用户使用。如果有，就给文件分配所需要的空间，如果没有。该文件就不能建立，用户进程必须等待。
    - 采用连续分配方式时，可以把逻辑文件中的记录 顺序地存储到相邻地物理盘块中，这样所形成地文件结构称为顺序文件结构，此时地物理文件称为顺序文件，这种分配方式保证了逻辑文件中地记录顺序于存储器中地文件占用盘块地顺序一致。
    - **连续分配地优点时查找速度比其他方法块(只需要起始盘块号和文件大小)目录中关于文件的物理存储信息也比较简单**，但是缺点时容易产生碎片。这种分配方式不适用于文件随时间动态增长和减少的情况。也不适合适合实现不知道文件大小的情况。
11. 外存的分配方式-链式分配：对于文件的长度需要动态增长以及用户实现不知到文件的大小的情况，往往采用链式分配。
    - 隐式分配：该实现方式用于链接物理的指针隐式的放在每一个物理块中，目录项中有指向索引文件中的第一块和最后一块的指针。此外每个盘块中都有含有指向下一盘块的指针。若是要房屋内某一个盘块，需要从第一个盘块开始一个个盘块都读出指针来(书上就是这么写的，貌似有点不通顺，差不多就是和链表的查找类似)。所以存在随机访问效率低的问题。由于其中任何一个盘块的指针错位都会导致后面的盘块位置丢失，因此这种实现方法的可靠性不高。
    - 显示连接：该种方案用于链接物理块的指针显示的放在一张连接表中，每一个磁盘都设置一个连接表。(注意，这个时链接表，存储的是每一个指针指向的信息，可不是索引表)这个表称为**文件分配表(FAT)。**由于还是链接方式，因此在FAT中找到一个记录的物理块还是要一个一个找，不能随机查找。**但是和隐式链接相比，该方案是在内存中查找而不是在硬盘中查找，所以速度快了不少**
12. 索引分配方式：链式分配存在一些问题，首先就是是随机读取时太慢，而且指针也占用了一定的磁盘空间。
    - 在索引分配方式中，系统为每一个文件都分配了一个索引块，索引块中存放者索引表。索引表中的每一个表项对应着分配给文件的一个物理块。**索引表不仅支持直接访问，也不会产生外部碎片，文件长度的限制问题也能得到了解决。**缺点就是由于索引块的分配，增加了系统存储空间的开销。另外，**存取文件需要两次访问外存，第一次时读出索引块中的内容，第二次时访问具体的磁盘块。**为了更有效的使用索引表，避免两次访问外存，可以在访问文件时先将索引表调入内存中，这样就只用访问一次外存就可以了。
    - 单极索引分配：单极索引分配方法就是将每一个文件对应的盘块号集中放在一起，为每一个文件分配一个索引块，在把分配给该文件的所有盘块号都放在该索引块中。因此，该索引块就是一个包含着多个盘块号的数组。
    - 两级索引分配：当文件较大时，一个索引块不能放下所有的文件，可以对索引块在建立索引。这样成立二级索引。
    - 混合索引分配：就是把两种索引方式放在一起，索引表上既有直接地址，又有二级索引表的地址，甚至还有更多级索引表的地址。
13. 文件的存储空间管理：
    - 空闲文件表法：如图所示

序号	第一个空闲块号	空闲块数目	物理块号
1	5	3	(5,6,7)
2	13	5	(13,14,15,16,17)
3	20	6	(20,21,22,23,24,25)
4	…	…	…

    - 空闲块链表法：空闲块链表法是将文件存储设备上的所有空闲块都链接在一起，形成一条空闲块链，并设置一个头指针指向空闲链块的第一个物理块。当用户建立文件时，就需要从链收其次取下几个空闲块分配给文件，当撤销文件时，回收其存储空间，并将回收的空闲块一次链入空闲块链表中。

-位图法：位图法就是建立一张位示图(尽管式称其为图，但是实际上是一串二进制位)以反映整个存储空间的分配情况。其中，每一个二进制位都对应着一个物理块，若某位为1，表示物理块已被分配，若某一位为0，表示物理块空闲。

14. 组成链接法(UNIX的文件存储空间的管理方法)：
    - 成组链接方法适用于大型文件系统，该方法将一个文件所有的空闲块按照每组100块分成若干个分组，每一个组的盘块数目和改组的所有盘块号记入到**前一个组的第一个盘块中**，第一组的盘块数目和第一组的所有盘块号记录到**超级块**中。这样每一个组的第一个盘块就链接成了一个链表。而组内的多个盘块形成了堆栈，**每一组的第一块是存放下一组的块号的堆栈**,堆栈是临界资源，每一次只允许一个进程访问。所以系统设置了一把锁来对其互斥访问。
    - 分配空闲块的方法：当系统要为文件分配空闲块是，先查找第一组的盘块数，如果不止一块，就将超级快中的空闲磁盘块减1，将栈顶的盘块分配出去，若第一组只剩下一块(就是存放下一组的盘块数和盘块号的那个块，不是空闲块)且栈顶的盘块号不是结束标记0(说明这一组不是最后一组)，**则先将该块的内容读到超级快中(下一组成了第一组，所以下一组的盘块数和盘块号要读入超级块)**，然后再将该块分配出去(该块中的信息已经不再有用，这一块成了一个空闲块)。
    - 空闲块的回收方法，当系统回收空闲块时，若第一组不满100块，**则只要再超级块的空闲盘块的栈顶放入该空闲盘块号，并将其中的空闲盘块数加1即可。**若第一组已经有100块了，则先把第一组中的盘块数和空闲盘块号写入该空闲盘块中，然后将”盘块数=1”和”栈顶块还=该空闲盘块号”写入超级块中(该空闲盘块成了新的一组，原本的第一组成了第二组)。
15. 磁盘结构的中的信息(我去，这个还真的在大题中考过)：
    - 引导控制块：通常为分区的第一块，若该分区没有操作系统，则为空。
    - 分区控制块：其中包括分区的详情信息，如分区的块数，块的大小，空闲块的数目和指针等。
    - 目录结构：采用目录文件组织。
    - 文件控制块：其中包含文件的信息，如文件名，拥有者，文件大小和数据块位置等。
16. 磁盘的访问时间T：访问时间=寻道时间+旋转时间+传输时间
    - 寻道时间：指的是磁盘收到读取指令后，磁头从当前位置移动道目标磁道的位置所需要的时间
    - 旋转延迟：一般来说平均旋转时间就是旋转半周所用的时间。
    - 传输时间：取决于每次读写的字节数和磁盘的转速。
17. 磁盘的调度算法：磁盘是可以被多个进程共享的设备。当有多个进程都要访问磁盘时，用采用一种合适的调度算法。以使得各种进程对磁盘的平均访问时间最短。
    - 先来先服务算法FCFS：顾名思义，按照进程到来的先后顺序服务。
    - 最短寻道时间优先算法SSTF：优先为距离当前磁头最近的进程服务。该算法的寻道性能比FCFS好，但是肯能会导致某些进程饥饿。但是不能保证平均寻道时间最短。
    - 扫描算法SCAN或者电梯调度算法：优先为当前磁头移动方向上举例磁头最近的进程服务。(其实就是从左往右扫描，再从右向左扫描，碰见一个进程就服务一个进程)
    - 循环扫描C—SCAN：就是单向的扫描算法。

调度算法	为了解决什么问题引入	优点	缺点
FCFS		公平,简单	未对寻道进行优化，所以平均寻道时间较长，仅适合磁盘请求较少的场合
SSTF	为了解决FCFS的平均寻道时间太长的问题	比FCFS减少了平均寻道时间，有更好的寻道性能	并非最优，而且会导致“饥饿”
SCAN	为了解决SSTF算法饥饿现象	兼顾较好的寻道性能和防止饥饿现象广泛用于大中小型计算机和网络中	存在请求刚好被错过而需要等待很久的现象
C-SCAN	解决请求待时间过长的现象	兼顾较好的寻道性能和防止饥饿现象，同时解决了一个请求等待时间过长的问题	可能出现磁臂长期停留再某处而不动的现象

18. 磁盘管理(2021年408大题…好吧，当时我脑子里一片空白，果断二战('▽`〃))：
    - 磁盘的格式化：一个新的磁盘只是一个含有磁性记录材料的空白盘，再磁盘能存储数据之前，它必须要分成扇区以便磁盘控制器能够进行读写操作。这个过程称为低级格式化。
    - 低级格式化未磁盘的每个扇区采取了独特的数据结构，每个扇区通常由头部，数据区域和尾部组成。头部和尾部包含了一些磁盘控制器所使用的信息。
    - 为了使磁盘存储文件，操作系统还需要将自己的数据结构记录磁盘上。将磁盘分为由一个或者多个柱面组成分区(就是常见的C盘和D盘等分区)
    - 对物理分区进行逻辑格式化(创建文件系统)，操作系统将初始的文件系统数据结构存储道磁盘上，这些数据结构包括空闲和已分配的空间以及一个初始为空的目。
    - 引导块：计算机启动时需要一个初始化程序(自举系统)，它初始化CPU，寄存器，设备控制器和内存等，接着启动操作系统。为此，该自举程序应找到磁盘上的操作系统内核，装入内存，并转到初始地址，从而开始操作系统的运行。
    - **自举程序通常存储在ROM中**为了避免自举代码需要改变ROM硬件的问题，只在ROM中保留很小的**自举装入程序**，而将完整的自举程序保存在磁盘的**启动块**上。启动块位于磁盘的固定位置，拥有启动分区的磁盘称为启动磁盘或者系统磁盘。

6 IO管理

1. 按照信息交换的单位，可以将IO系统分为两类，第一类是块设备，这类设备用于存储信息，由于信息的存取总是以**数据块**为单位，因此得名。典型代表是磁盘。第二类为字符设备，用于数据的输入和输出，其基本单位是字符，故称为字符设备。代表由打印机，交互终端等。
2. 按照设备共享的属性分类：
    - 独占设备：指的是在**一段时间内只允许一个用户(进程)访问的设备**，即临界资源。因而，对于多个并发进程而言，应该互斥的访问这类设备。系统一旦把这类设备分配给某个进程后，便由该进程独占，直到用完释放。独占性设备有可能会引起进程的死锁。(比如打印机)
    - 共享设备：这是指**一段时间内允许多个进程同时访问的设备。**当然，对于每一时刻而言，该类设备仍只允许一个进程访问。显然，**共享设备必须是可寻址的和可随机访问的设备，典型代表是盘。**共享设备不仅可获得良好的设备利用率。而且也是实现文件系统和数据库系统的物质基础。
    - 虚拟设备，**这是通过虚拟技术将一台独占设备变为若干台逻辑设备，供若干个用户同时使用**
3. 三类总线：
    - 数据信号线：这类信号线用于在设备和设备控制器之间传信号。
    - 控制信号线：这是作为由设备控制器向IO设备发送控制信号时的通路
    - 状态信号线：这类信号线用于传送指示设备当前状态的信号。
4. 设备控制器是计算机中的一个实体(这个貌似考研没有涉及)，其主要职责是控制一个或者多个IO设备，以实现IO设备和计算机之间的数据交换，他是CPU和IO设备之间的接口，它接收从CPU发出来的命令，并且控制IO设备工作，以使处理机从繁杂的设备控制事务中解脱出来。设备控制器是一个可编址的设备，当它仅控制一个设备时，它只有一个唯一的设备地址；若控制器可连接多个设备时，则应含有多个设备地址，并使每一个设备地址对应一个设备。设备控制器的基本功能：接收和识别命令，数据交换，标识和报告设备状态，地址识别，数据缓冲，差错控制。
5. IO通道设备的引入(这个貌似考研也没有涉及)：虽然在CPU和IO设备之间加入了设备控制器后，已经大大的减少了CPU对IO的干预，但是当主机所配置的外设很多是，CPU的负担仍然很重，为此，在CPU和设备控制器之间又增设了通道。其主要目的是为了建立独立的IO操作，不仅使得数据的传送能够独立于CPU，而且也希望有关对IO操作的组织，管理异界结束处理尽量独立，以保证CPU有更多的时间来进行数据处理。或者说，其目的是使一些由CPU处理的IO任务由通道来承担，从而把CPU从繁杂的IO任务中解脱出来。在设置了通道后，CPU只要向通道发送一条IO指令，通道在收到IO指令后，便从内存中去除本次需要执行的通道程序，然后执行该通道程序，仅当通道完成了规定的IO任务后，才向CPU发送中断信号。

   实际上，**IO通道是一种特殊的处理机**，它具有执行IO指令的能力，并通过执行通道IO程序来控制IO操作，但是IO通道又于一般对的处理机不同，主要表现在以下两个仿麦呢：一是指令的类型单一，二是通道没有自己的内存，通道所执行的通道程序是放在主机的内存中的。

6. IO控制方式1-程序直接控制方式：在早期的计算机系统中，因为没有中断系统，所以CPU和IO设备进行通信，数据传输时，由于CPU的速度远远块于IO设备，因此CPU需要不断的测试IO设备。这种方式又称为轮询或者忙等。
   - 当数据输入时，由处理器向设备控制器发出一条IO指令启动设备进行输入，在设备的输入期间，处理器通过循环执行测试指令不断的检测设备状态寄存器的值，当状态寄存器的值显示设备输入完成时，处理器将数据寄存器中的数据取出并送入内存的指定单元，然后再启动设备去读下一条数据。当用户进程需要向设备输出数据时，也必须同样发出启动命令启动设备输出并等待输出完成操作。
   - 优点：程序直接控制方式的工作过程非常简单。
   - 缺点：CPU的利用率非常低。因为IO设备速度太慢，跟不上CPU，使得CPU在绝大部分时间都要测试IO设备是否已经完成数据的传输。
7. IO控制方式2-中断控制方式：为了减少程序直接控制方式中的CPU等待时间，提高CPU于设备的并行工作程度，现代计算机系统中广泛采用了中断的控制方式来对IO设备进行控制。
   - 当用户进程需要输入数据时，由CPU向设备控制器发出指令启动外设输入数据，在输入数据的同时，CPU可以做其他工作，当输入完成时，设备控制器向CPU发出一个中断信号，CPU收到信号以后，转去执行设备中断处理程序。设备中断处理程序向输入设备的寄存器中的数据传送道某一个特定的内存单元中，供要求输入的进程使用，然后再启动设备去读取下一个数据。
   - 优点：与程序直接控制相比，有了中断的硬件支持后，CPU和IO设备之间可以并行工作，CPU只需要再收到中断信号后处理即可，大大提高了CPU的利用率。
   - 缺点：这种控制方式仍然存在着一些问题，比如每一台设备输入，输出一个数据，都要中断CPU，中断CPU的次数太多，也会大量消耗CPU的时间。
8. 中断程序处理过程(仅指IO完成时发出的中断)：
   - 唤醒被阻塞的驱动(程序)进程：可用signal操作或者发送信号来唤醒被阻塞的驱动程序。
   - 保护被中断进程的CPU环境：将**处理器状态字PSW**和**程序计数器PC**压入栈中加以保护，其他需要被压入栈中保护的还有CPU的寄存器等，这些都由应将完成。
   - 转入想要的设备处理程序：测试中断源以确定引起中断的设备号
   - 中断处理：针对该设备调用相应的中断处理程序。
   - 恢复被中断进程的现场，把当时压入栈保护的寄存器等数据弹出，回复当时的CPU执行的上下文。

   注意，这里所说的中断仅指由输入，输出引起的中断。
9. IO控制方式3-DMA控制方式：DMA控制方式的基本思想时外设和内存之间开辟直接的数据交换通路。在DMA控制方式中，设备控制器有更强大的功能，在其控制下，设备和内存之间可以**成批的交换数据**，而不用CPU的干预，这样大大减轻了CPU的负担，这种交换方式一般用于**块设备**的数据传送。
   - 以数据的输入为列，当用户进程需要数据时，CPU将准备存放输入数据的内存的起始地址以及要传输的字节数分别送入DMA控制器中的内存地址寄存器和传送字节数寄存器。在传输数据的同时，CPU可以去做其他事情，输入设备不断地挪用CPU的工作周期，将数据寄存器中的数据源源不断的写入内存，直到要求传输的数据全部传输完毕。**DMA控制器在传输完毕时向CPU发出一个中断信号**，CPU收到中断信号后转中断处理程序执行，中断结束后返回被中断的程序。
   - DMA控制方式的基本特点：数据传输的基本单位是数据块，数据是单项传输，从设备直接送入内存(或者是从内存直接送到数据)。仅在传输一个或者多个数据块的开始和结束时，才需要CPU的干预，整块的数据的传送是在控制器的控制下完成的。
   - DMA控制方式和中断方式的主要区别是：中断控制方式在每个数据传输完成后中断CPU，而DMA是在所要求传送的一批数据全部传送结束时才中断CPU。中断控制方式的数据传送时在中断处理时由CPU控制完成的。而DMA控制方式是在DMA控制器控制下完成的。
   - DMA控制器中主要有4类寄存器，用于主机和控制器之间的成块数据交换。
   - 命令/状态字寄存器(CR)：用于接收从CPU发来的IO命令或者有关的控制信息，或者设备的状态。
   - 内存地址寄存器(MAR)；用于存放数据从设备传送到内存或者从内存到设备的内存地址。
   - 数据寄存器(DR):用于暂存从设备到内存或者从内存到设备的数据
   - 数据计数器(DC):存放本次要传送的字数。
   - 优点：DMA控制下，设备和CPU可以并行工作，同时设备与内存的数据交换速度加快，不需要CPU的干预。

- 缺点：DMA方式在数据传送的方向，存放数据的内存起始地址，和数据长度都需要CPU的控制，**并且每一台设备都要有一个DMA控制器**，当设备增加时，多个DMA控制器的使用不经济。

10. IO控制方式4-通道控制方式：通道控制方式与DMA控制方式类似，也是一种以内存为中心，实现设备与内存直接交换数据的控制方式。与DMA控制方式相比，通道所需要的CPU干预更少，而且可以做到**一个通道控制多台设备**，从而减少了CPU的复旦。通道的本质是一个简单的处理器，它独立于CPU，有运算和逻辑控制，有自己的指令系统，也在程序控制下工作，专门负责输入，输出控制，具有**执行IO指令的能力**，并且通过执行通道IO程序来控制IO操作。
    - 与CPU不同的是，通道的指令类型单一，由于通道硬件比较简单，其所能执行的命令主要局限于IO操作的有关指令。**通道没有自己的内存，而是与CPU共享内存。**
    - 优点：通道控制方式解决了IO操作的独立性和各部件工作的并行性，通道把中央处理器从繁琐的输入，输出操作中解放出来，采用通到技术后，不仅能实现CPU和通道的并行操作，而且**通道和通道之间也能实现并行操作。各种通道上的设备也能实现并行操作**，从而达到提高整个系统效率的根本目的。
    - 缺点：需要更多的硬件，成本较高。
    - 与DMA方式的区别：DMA方式中需要CPU来控制传输的数据块大小，传输的内存，而通道方式中这些信息都是由通道来控制和管理的。其次，一个DMA控制器对应一台设备与内存传递数据，一个通道可以控制多台设备与内存交换数据。
11. 假脱机技术：系统中独占设备的数量有限，往往不能满足多个进程的需要，从而成为系统的瓶颈，使得许多的进程被阻塞。另外，得到独占设备的进程，在整个运行期间往往占有但不经常使用的设备，使得设备的利用率低，为了克服这种缺点，人们通过共享共享设备来虚拟独占设备，将独占设备改造成虚拟设备。从而提高了系统的利用效率，这种技术成为假脱机SPOOLing技术。
12. SPOOLing的技术特点：
    - 提高了IO速度，从低速IO设备进行的操作变为了对输入井与输出井的操作。如同脱机一样，提高了IO速度，缓和了CPU与低速IO设备速度不匹配的矛盾。
    - 设备并没有分配给任何进程。在输入井与输出井中，分配给进程的是一个存储区和建立一张IO请求表。
    - 实现了虚拟设备功能。多个进程同时使用一个独占设备，对于每一个进程而言，都认为自己独占了这个设备，从而实现了设备的虚拟分配。不过，这个设备是逻辑上的设备。
    - SPOOLing除了是一种速度匹配技术外，也是一种虚拟设备技术。它用一种物理设备来模拟另一类物理设备。使得各个作业在执行期间只是用虚拟设备，而不是使用物理的独占设备。使得设备的使用率与系统的效率都得到提高。

计算机组成原理

1.计算机系统概述

1. 计算机的发展历程：
    - 第一代计算机(电子管时代)：主要特点：电子管作为开关器件，使用计算机语言，可以存储信息，输入/输出很慢
    - 第二代计算机(晶体管时代)：主要特点：晶体管代替电子管，采用磁心存储器，汇编语言取代机器语言
    - 第三代计算机(中小规模集成电路时代)：主要特点：中小规模集成电路代替晶体管，操作系统的问世。
    - 第四代计算机(超大规模集成电路时代)：主要特点：采用集成度很高的电路，微处理器问世。
2. 第一代到第四代的计算机，计算机体系机构都是相同的，即为由**控制器，存储器，运算器，输入设备和输出设备**组成，成为冯诺依曼结构。
3. 机器字长：值得是一次整数运算给能够处理多少二进制数据的位数
4. 有关于存储器的概念解释:

存储的概念类	说明
存储元	也成为存储元件和存储基元，用来存放一位二进制信息
存储单元	由若干个存储元组成，能存放多位二进制信息
存储体	许多的存储单元构成存储体
存储字	每一个存储单元中二进制代码的组合即为存储字，可以代表数值，指令和地址等
存储字长	每一个存储单元中二进制的代码的位数是存储字长

5. 运算器：运算器是对信息处理和运算的不见，主要功能是进行算术和逻辑运算，其核心是逻辑单元(ALU)。
6. 控制器：控制器是整个计算机的指挥中心，它使得计算机分各个部件自动协调工作，计算机有两种信息在流动，一种是控制信息，另一种是数据信息。控制器是由程序计数器PC，指令寄存器IR，和控制单元CU组曾。PC用来存放当前的预执行指令的地址，可以**自动+1**形成下一条指令的地址。他与MAR之间有一条直接通路。IR用来存放当前的指令，其内容来自MDR。指令中的操作码字段OP送至CU，用以分析指令发出的各种位操作命令序列。指令中的地址码字段Ad送至MAR获取操作数。
7. **计算机如何判断去除的是数据还是指令?因为数据和指令要送往不同的地方吗?**
   - 通常完成一条指令可以分为取址阶段和执行阶段，在取址阶段，通过访问存储器可以取出指令，在执行阶段，通过访问存储器可以取出操作数，这样，虽然指令和数据都是以二进制代码的形式放在存储器中，但是CPU可以判断在取址周期访问存储器取出的是指令，在执行阶段访问存储器取出的是数据。
   -
8. 编译程序，解释程序的区别：
   - 解释程序：是高级语言翻译程序的一种，它将源语言书写的源程序作为输入，解释一句就交给计算机执行一句，并不形成目标程序。
   - 编译程序：把高级语言的源程序作为输入，进行翻译转换，产生出机器的目标程序，然后在让计算机去执行这个目标程序，得到计算结果。
   - 可见，编译程序于解释程序的最大区别的就是：编译程序生成目标代码，解释程序不生成。此外，编译程序产生的目标代码执行速度比解释程序的执行速度快。
9. CPU周期：又称为**机器周期**，在计算机中，为了方便管理，把一条指令的执行分为若干个阶段。每一个阶段完成一个基本操作，完成一个基本操作所需要的时间成为机器周期，通常一个指令周期由若干个CPU周期组成。**CPU周期和CPU时钟周期不一样，后者是CPU主频的倒数。**

2.数据的表示和运算

我猜这一块可能真不考,不写了

3.存储器的层次结构

1. 存储器的分类：按照存取方式，可以分为随机存取存储器，只读存储器，顺序存储器和直接存储器。
   - 随机存取存储器：随机存储器RAM，在随机存储器中存取信息，存取时间于和存储位置没有关系。优点是读写方便，使用灵活，缺点是断电后信息丢失。分为静态RAM(SRAM)和动态RAM(DRAM)静态RAM通常作为高速缓冲存储器，动态RAM常作为主存。
   - 只读存储器：只读存储器(ROM)。顾名思义，只读存储器的内容只能随机读出但是不能写入。并且其内容断电后仍可以保留。所以把一些固定的，不变的程序放在这里，只读存储器ROM于随机存储器RAM一起构成了主存。只存储器主要包括了掩膜型只读存储器(MROM)，可编程只读存储

器(PROM)可擦除只读存储器(EPROM)。电可擦除只读存储器(EEPROM)和快擦除读写存储器(Flash Memory)。

- 串行访问存储器：串行访问存储器对存储单元进行读或者写操作时，需要按照物理位置的先后顺序一次访问，主要包括顺序存取存储器(磁带),和直接存取存储器(磁盘)。磁盘是属于半串行的，因为磁盘在寻找数据时，先要寻道，这个寻道是直接找磁道的，不需要按照顺序查找，所以属于随机访问。寻道之后要在磁道旋转，顺序寻找需要找的信息，因此有时串行访问。将这种前端直接访问，后端是穿行放问的存储器称为直接存取存储器。

2. ROM是采用随机存取的方式继续宁信息的**访问**，为什么这里将存储器分为只读存储器和随其存取存储器?随机存取是什么意思?难道ROM不是随机存取存储器吗?

   - 随机存取是表示可以随时的访问存储器的任意单元地址，就好型数组可以通过下标来访问目标元素一样，而链表就需要一个一个的去找，因此数组是随机存取，而链表不是随机存取。
   - ROM确实可以算是随机存储器，因为RON就是采用随机存取的方式**访问**信息，但是随机存储器一般定义是必须要满足可以随机进行存取，但是ROM只能进行取操作，不能进行存操作，所以一般将ROM与RAM分开

3. 注意，CDROM不是随机访问存储器，CDROM属于光盘，没有随机访问的特性。

4. 读周期：指的是芯片连续两次读操作的最小时间间隔，读时间表是进行一次存储器的读操作的时间。显然小于读周期。

5. 写周期：指的是芯片连续两次写操作的最小时间间隔，读时间表是进行一次存储器的读操作的时间。显然小于写周期。

6. DRAM存储器的刷新：DRAM的存储原理：采用电容的方式存储，电容中充满了电荷和电容中没有电荷正好对应了1和0的两种状态，根据电路的基础知识吗，电容中的电荷不能永久保留，而是会随着时间慢慢流逝，这是时间如果考研中没有做特殊说明，一般是2ms(即为每个基本存储单元在2ms内都要刷新一次，否则会使得电荷流逝，进而导致存储信息出错。)

7. 通常刷新方式有三种，集中刷新，分散刷新和异步刷新。

   - 集中刷新：一般来说，电容上的电荷基本只能维持2ms，在2ms内必须要刷新一次，将2ms看成一个刷新周期，假设存储周期为0.5us，那么在一个刷新收起里有4000个存储周期，假设该矩阵有32行，则对32行集中刷新徐娅16us，在刷新期间内是不能进行读操作或者写操作的，故程刷新时间为死时间。又称为访存的死区。这个死去的占用比例为32/4000=0.8%
   - 分散刷新：在分散刷新中，存储周期已经不再是传统的存储周期了，也就是说，此时存储周期不再等于读(写)周期，这里对操作的定义：**存储时间=读或写周期+刷新一行的时间。**此处的刷新一行的时间又可以看成等于存储周期的。分散刷新将存储器的存储周期认为的延长了，因此严重的降低了系统的速度。
   - 异步刷新：异步刷新时把存储矩阵的每一行分散到2ms内刷新，但是又不是集中刷新。而是平均分配。这能保证当刷新完第一行后，再经过2ms又可以完成对下一行的刷新。异步刷新的死时间为一个读/写周期。

8. 对于刷新的补充：

   - 对于集中刷新：死时间很好求，就是那一段集中刷新的时间。
   - 对于分散刷新，刷新占据了时间，而且刷新时间内不能进行读或写对的创造，那不就是”死时间”吗?但是为什么有些教材上说分散刷新没有死时间?

     死时间是存储器为的不能进行读或者写操作的刷新时间。而且这段死时间，要连续不能累积的。因此，对于分散刷新来说，由于刷新时间被存储器包含了，它在内而不在外，所以死时间是不存在的。
     - 对于异步刷新来说，由于死时间是不能累加的，因此与不刷新的死时间就是一个读或者写周期。或者是没又异变的存储周期。
     - 存贮芯片是按照一行一行来刷新的，不是一个一个刷新的。

9. 闪存Flash存储器：闪存虽然是内存的一种，但是又不同于内存众所周知，如果没有电流的供应，计算机内存中的内容会立刻消失，而闪存能在没有电流的条件下长久的保存数据。其存储特性相当于硬盘，这项特性是是闪存得以成为携带数字设备的存储介质的基础。一般来说，闪存是按照**块**来存取数据的，不是字节。

内存类型	非易失性	可写

内存类型	非易失性	可写
闪存	是	是
SRAM	不是	是
DRAM	不是	是
ROM	是	不是
PROM	是	不是
EPROM	是	是
EEPROM	是	是

10. 存储器的扩充：存储器的扩充，假设存储器的形式位a*b的形式，表示又a个长度位b的存储单元：
    - 位扩充：增加b，指的是**扩充存储字长**
    - 字扩充：增加a。
11. 多体并行存储器：多题并行存储器的就是采用多个模块组成的存储器，每一个模块都有相同的容量和存取速度，每个模块都有独立的地址寄存器，数据寄存器，地址寄存器和读写电路。每个模块都可以看作是一个独立的存储器。
    - 高位交叉编址的多体寄存器：每个模块被的地址顺序是连续的，因此被称为顺序存储。(就是低位地址是存储器内的地址，高位地址是存储体的地址)。优点：有利于存储器的扩充，只要将存储单元的编号往后加即可。缺点：每一个模块都是**串行工作**，因此存储器的带宽受到了很大的限制。
    - 低位交叉编址的多体存储器：低位地址表示位体号，高位地址表示位块内地址，这样连续的地址就放在不同的存储体中，而每一这存储体中的地址都是不连续的。对于连续字成块的传送，低位交叉编址的多体寄存器可以实现多模块流水线式的并行存取，可以大大的提高存储器的带宽。
12. 低位交叉编址存储器的分析：
    - 讨论的前提是模块的字长等于数据总线的宽度。假设模块存储周期位T，总线传输周期为t，且存储器是由m个模块组成，如果T=mt，那么要低位交叉存储器的模块数目必须大于或者等于m以保证当经过mt的时间后再次启动存储体，它的上一次的存取操作已经完成。
13. 高速缓存Cache：在多体并行存储器中，外部设备优先级最高，这样会导致CPU等待外部设备访存的现象，导致CPU空等一段时间，甚至是等待几个主存周期，从而降低了CPU的工作效率。为了不免CPU与IO设备争抢缓存，可以在CPU与IO之间加上一个Cache。这样，如果外部设备正在和主存交换信息，CPU就可以不同等待，直接从Cache中取得信息。
    - Cache地址：Cache地址由两部分组成，高c标识Cache块号，低b位为块内地址。Cache一共由2^c块。
    - 命中的概率：CPU访问信息在Cache中的比例。
    - 平均访问时间：假设命中概率为h，tc为命中时访问Cac和的时间，tm为未命中时访问主存的时间，则Cache-主存系统的平均访问时间ta为：**ta=h*tc+(1-h)*tm**
    - Cache-主存系统的效率：**e=tc/ta**
    - CPU与Cache之间的数据攒送的基本单位时字，而主存与Cache之间的数据传送单位为块。
    - CPU访问主存时，会同时的把地址给Cache和主存，Cache的逻辑依据地址判断此字是否在Cache中，若此字在Cache中，立刻传送给CPU，否则，用主存读周期把此字从主存读出并传从到CPU，与此同时，把含有这个字的一整块数据从主存读出，并传送到Cache中。
    - Cache放的时主存信息的副本，所以不会增加系统的存储容量。
14. Cache与主存之间的映射关系：
    - 直接映射：**i=j mod C** 其中，i为Cache的块号，j为主存的块号，C为Cache的块数。根据上面的公式就可以将主存中第j块的信息复制到第i个Cache块中。优点：简单，缺点：不够灵活，冲突概率高(抖动)
    - 全相联映射：全相连映允许主存中的每一个块映射到Cache中的任何一个位置。优点：Cache的命中率高，减小了Cache中的冲突率，提高了Cache中的命中率。缺点：tag的位数增加，访问Cache时

主存子块需要和Cache的全部标记进行比较，才才能判断所访问的内容是否在Cache中，这种比较通常采用**按内容寻址**的相连存储器来完成。

- 组相联映射：组相连映射时对直接映射和全相联映射的一种折中的方式，假设把Cache分成Q组，每一个组有R块，**i=j mod Q**其中，i为Cache中的组号，j为内存中的块号，Q为Cache中的组数。通俗的说，就是把第j块的内容复制到Cache中的第i组，至于是第i组中的那一块，就随意放即可。

15. 组相联映射的补充：

- 当相连中只有一组时，此组相联映射就相当于全相连映射，当每一个组只有一块时，该组相联映射就相当于直接映射。
- 在组相连映射中，主存从高位到低位一共划分3部分，标记tag，组号和内存地址。**块内地址=log2(块大小)，组号=log2(Cache组数)，标记tag=主存剩余的其他位。**
- 假设每一组中有N块，就称为N路组相联。

16. Cache中的替换算法：只有使用全相联映射与组相联映射时才会使用到替换算法。一般常用的有先进先出算法，最近最少使用算法，随机算法。其中，随机算法随机的选出一个被替换的块，没有用到局部性原理，所以不能提高Cache的命中率。

17. Cache的写策略：由于Cache的内容是主存中的副本，所以它因该与主存保持一致，而CPU对Cache的写入改变了Cache的内容，就会导致Cache的内容与主存的内容不一致，如何让Cache与主存的内容保持一致就是写策略需要完成的事。

- 写回法：写回法要求当CPU写Cache命中时，只要修改Cache中的内容，而不是立刻的写会主存。这种方式可以减少访存次数，但是实现这种方式需要设置一个修改位，当某一行被换出时，根据此行的修改位是0还是1，来决定将该行的内容写回主存还是简单的弃去。注意，上面考虑的是Cache命中时，那没有命中的时候呢?如果CPU要对Cache中的某一位进行就该，但是此时此字恰好没有在Cache中，**就需要从主存中找到包含该字的数据块，千万注意：CPU不会再主存中直接修改，而是找到之后直接复制到Cache中进行修改。**等从Cache中换出此块时，再复制到主存中。(修改只发生再Cache中)
- 全写法：当写Cache命中时，Cache与主存同时修改，因而较好的保持了Cache与主存的一致性。很明显，此时的Cache不需要在每一行中都设置修改位，当写Cache未命中时，直接在主存中修改。至于在主存中修改后需不需要再复制到Cache中，这个视情况而定。可以复制也可以不复制。
- 写一次法:写命中与没有写命中时的处理办法与写回法基本相同，**仅仅是第一次写命中时要同时写入主存。**

18. 虚拟存储器: 这个再操作系统中有比较详细的记录,和这里基本相同，这里只做一点补充：

- 虚拟存储器是一个逻辑模型，并不是一个实际的物理存储器。
- 虚拟存储器必须建立再主存-辅存的基础上，但是两者有差别，虚拟存储器允许使用比主存大得多的地址空间，虚拟存储器每一次访问时，都要进行虚实地址变换，而非虚拟存储器则不必。
- 虚拟存储器的作用是分割地址空间，解决主存容量问题以及实现程序的重定位。
- 虚拟存储器的容量由计算机地址总线的数量来决定。

19. 两个转换关系(天勤计算机组成原理中的)：

- 逻辑地址(虚拟地址)是由程序员给出的，经过查询快表(TLB)，页表(很多书中对于这两者的先查谁还是同时查没有给出标准，编者给出的解释是，现代的电路技术已经完全可以实现同时查询，如果TLB命中，就直接输出由TLB中的页表查询所得到的物理地址，如果TLB没有命中，就根据页表输出物理地址)得到物理地址。但是现在的物理地址并不一定是最终的地址。如果Cache命中，就需要转换为Cache地址，转换后的Cache地址才是最终的物理地址。但是如果Cache没有命中，还需要经过TLB或者是页表得到的地址来直接访存。因此，再不命中的情况下，两个转换关系就变成了一个，即逻辑地址->物理地址。

20. TLB命中，页表必然命中，但是Cache的命中与TLB或是页表的命中没有必然关系。

21. CPU执行一次存储访问操作最少需要访问几次主存?

- 在具有Cache并采用动重定位的存储系统中，一次访存的大致操作如下：
- 第一步：根据虚页号查找快表，如果块表中由对应的页表项，则取出页框号形成物理地址，转到第二步，如果快表中不存在对应虚页的页表项，则发生TLB确实，转换到第三步。
- 第二步：判断物理地址中的标记是否和Cache中的标记相等且有效位是否为1，若为1，则Cache命中，从Cache中读出数据或者写数据到Cache中(全写的方式下，同时也要写主存。)若不为1，则发生Cache缺失，转到第四步。

- 第三步：当TLB缺失，根据页表基址寄存器的值和虚页号找到主存中对用的页表项，判断有效位是否为1，若是，则说明虚页存在主存中，此时把页表项转入到TLB中，并取出页框号作为物理地址，转到第二步。如果不是，说明该虚页不在主存中，即为发生了缺页异常，此时调出系统中的缺页异常处理程序，实现从磁盘中读入一个页面的功能，缺页处理后，重新执行当前的指令，这一次一定能在主存中找到。
- 第四步：Cache缺失时，CPU根据物理地址到主存读一块信息到Cache，然后读入CPU或者CPU写信息到Cache中。
- 由此可见，当CPU进行一次访存操作时，最好的情况下不需要访问主存(TLB命中，Cache命中)，最坏的情况下不仅要多次访问主存，还要访问磁盘(Cache缺失，TLB缺失，缺页异常)

22. TLB，页表，Cache，主存之间的访问关系

TLB	页表	Cache	是否可能发生
命中	命中	命中	可能
命中	命中	不命中	可能
命中	不命中	命中	不可能(因为TLB是页表的子集)
命中	不命中	不命中	不可能(因为TLB是页表的子集)
不命中	命中	不命中	可能(数据在主存，不在Cache)
不命中	命中	命中	可能
不命中	不命中	命中	不可能(因为数据不在主存)
不命中	不命中	不命中	可能(因为数据不在主存)

4.指令系统

1. 一条指令首先要告诉机器，用户要干什么(操作码)，还有就是要直到对谁操做(地址码)。因此，一条指令由操作码和地址码组成。地址码需要做的事情：
   - 需要指出操作数的地址，即为用哪里的数来操作。
   - 需要指出操作后的结果存放在哪里，即为给出存放结果的地址。
   - 需要给出该条指令执行结束后因该怎么办，即为给出下一条指令的地址。
2. 零地址指令：零地址指令只给出操作码字段OP，没有地址码字段，主要包括两种情况：
   - 不需要操作数的指令：如空操作指令，停机指令，关中断指令
   - 堆栈计算机中的零地址指令：堆栈计算机中参与运算的两个操作数隐含的从栈顶和次栈顶弹出，送到计算机中进行运算，运算的结构在隐含的压入栈顶中。
3. 一地址指令：一地址指令中的指令字段只有一个，主要包括两种情况：
   - 只有目的操作数的单操作数指令，从A1中读取操作数，进行OP操作后，返回A1，操作码通常为+1，-1，求反码，求补码等操作。(书上没有说访存几次，貌似是只用访存一次即可，就是取指的一次)
   - 隐含约定目的地址的双操作数指令，按指令地址A1可读取源操作数，指令可以隐含约定另一个操作数由累加器ACC提供，运算结果也保存在ACC中。(书上没有说要访存几次，不过貌似是两次，取指，取操作数)
   - 二地址指令：OP， A1， A2，A1为源操作数的地址，A2为目的操作数的地址。常见于加减乘除运算中，运算的结果保存在A2所指定的存储单元中。(书上没有说要访存几次，不过貌似是四次，取指，去操作数*2，存放结果)
   - 三地址指令：指令格式为OP，A1，A2，A3(结果)。其中A1，A2为两个源操作数地址，A3为运算结果地址。此类指令如果地址字段都是主存地址，则需要四次访存。(取指，取操作数*2，存放结果)

- 四地址指令：指令格式：OP，A1，A2，A3(结果)，A4(下一指令地址)。和三地址指令一样，同样需要进行四次访存

4. 指令寻址：找到下一条要执行指令的地址，称为指令寻址，指令寻址基本上是按执行顺序存放在主存中的，执行过程中，指令总是从内存单元被渠道指令寄存器IR中。一般来说，指令寻址只有两种方式，顺序执行时，用指令计数器(PC) +1 来获取下一条指令的地址，跳转执行时，通过转移指令的寻址方式，计算出目标地址，送入PC即可。目标转移地址的形式有三种，立即寻址，相对寻址和间接寻址。

5. 数据寻址方式1-立即寻址：**op # A** 这个寻址方式是直接给出操作数，不需要给出地址去其他地方找操作数，其中的#为**立即**寻址特征，A不是操作数的地址，而是操作数本身。通常把#放在立即数前面，以标识寻址方式为立即寻址，如#20H，其他寻址方式则不用特殊符号来标识。

   - 优点：只要取出指令，便可以立即获得操作数，采用立即寻址特征的指令只要在取指令时访问存储器，而在执行阶段不必访存储器。
   - 缺点：因为A是立即数，所以A的位数限制了立即数的范围。例如，A占8为，则立即数表示范围为-128-127(立即数都是用补码来表示)。
   - 立即数寻址的用途1：例如，需要传送一个循环次数给某一个专用寄存器，就可以用立即数寻址将循环次数作为立即数送入。
   - 立即数寻址的用途2：例如，需要将某程序的首地址送入程序计数器中，而且程序的首地址可以看成是一个操作数，则可以使用通过立即寻址将该程序的首地址作为立即数送入。
   - 用途总结：立即寻址方式通常用于对某一个寄存器或者内存单元赋初值。

6. 数据寻址方式2-直接寻址：**LAD 寻址特征码 A**，其中A是操作数的有效地址，不不需要进行地址变换。**有效地址EA=A**

   - 优点:简单，取出操作数只需要一次访存。
   - 缺点：操作数的有效地址由A决定，但是A的位数一般比较小，因此寻址范围比较小。

7. 数据寻址方式3-隐含寻址(了解即可，天勤上就是这么写的…)隐含寻址指指令字中不明显的给出操作数的地址，其操作数隐含在操作码或者某个寄存器中，其中最典型的例子就是一地址指令的加法指令，操作码是ADD，说明其至少要由两操作数才能做加法，而地址码只给出了一个操作数的地址，其另外一个操作数在ACC中。也就是说一地址格式的算术运算指令的另一个操作数隐含在ACC中。

8. 数据寻址方式4-间接寻址：**OP 寻址特征码 A**，直接寻址的A比较小，所以寻址范围非常小。间接寻址给出的是操作数**有效地址的地址**。间接寻址又分为一次间接寻址和多次间接寻址。对于一次间接寻址来说，**EA=(A)。**

   - 优点，便于子程序返回和查表(不需要深究，书上就是这么写的)。
   - 缺点：一次间接寻址需要在**执行阶段**访问两次存储器，第一次是取操作数的有效地址，第二次是取操作数。而N次间接寻址需要访存N+1次。(注意，这里的访存说的是在执行阶段，如果把取指令也算上的话貌似还要再加上一次取指令时访存。)
   - 用途：在寻找终端服务程序的入口就是使用间接寻址。

9. 数据寻址方式5-寄存器寻址：和直接寻址相似，在直接寻址中，地址码给出的时主存地址，在寄存器寻址方式中，地址码给出寄存器的编号Ri，操作数的有效地址EA=Ri。

   - 优点：由于操作数在寄存器中，因此在执行阶段不需要访存，减少了执行时间。
   - 减少了指令字的长度(选择题可能会出)。

10. 数据寻址方式6-寄存器间接寻址：就是寄存器中的不是操作数，而是操作数所在**主存单元的地址号**。有效地址EA=(Ri)。

    - 优点：便于编制循环程序。

11. 数据寻址方式7-基址寻址。字面意思就是操作数的有效地址需要加上某一个基础地址来形成，这个基础地址放在一个基址寄存器(BR)中，其操作数的有效地址**EA=A+(BR)。**

    - 基址寄存器可以采用隐式和显式两种，所谓的隐式，就是指在计算机内专门设置一个基址寄存器(BR)，使用时用户不必明显的指出该基址寄存器，只需要指令的寻址特征位反应出基址寻址即可。显式是值在一组通用寄存器中，用户指明哪个寄存器作为基址寄存器，存放基地址。
    - 优点：扩大寻址范围。比如原来只能寻址300-304，但是在基址设置成200后，就能寻址500-504了。
    - 便于解决**多道程序问题。**(书上说知道即可)
    - 注意：基址寄存器内容由操作系统给出，在程序的执行过程中用户不能随意的改变。

- 虽然基址寄存器的内容不能由用户改变，但是采用通用寄存器组来作为基址寄存器时，用户有权知道到底使用了哪个通用寄存器来作为基址寄存器。

12. 数据寻址方式8-变址寻址：和基址寻址类似，有一个变址寄存器IX，有效地址**EA=A+(IX)**。
    - 注意，在变址寻址中，变址寄存器的内容是由用户设定的，在执行过程中其值是可以改变的，但是指令的形式地址A是不可以改变的。这一点恰好的基址寄存器相反。
    - 优点：扩大寻址范围。
    - 非常适合处理数组和循环的问题：由于编制寄存器的内容是可以由用户改变的，因此在处理数据问题时，只需要将指令字中的形式地址设置位数组首地址，然后用户只需要不断地改变变址寄存器IX中地内容即可。
    - 基址寄存器与变址寄存器地区别：两种方式的有效地址都是寄存器内容+偏移地址，但是在基址寻址中，程序源操作的时偏移地址，基址寄存器中的内容由操作系统控制，在执行过程中**动态调整(不是一直不变的，用户不能改不代表系统不能改)**。而变址寻址中，程序员操作的时变址寄存器，偏移地址是固定不变的。
13. 数据寻址方式9-相对寻址：相对寻址基于**局部性原理**，相对寻址的有效地址是将程序计数器(PC)中的内容与指令字的形式地址A相加而成的。**EA=(PC)+A**。
    - 相对寻址的用途1：用于转移类指令，转移后的目标地址与当前指令由一段距离，称为相对位移量，此位移量由指令字的形式地址给出，故A又称为位移量。位移量可正可负，通常用补码表示。
    - 相对寻址的用途2：用于编制浮动程序：即程序的正确运行不受程序所在的物理地址的限制。
    - **基址寻址，变址寻址，相对寻址都可以看成是偏移寻址。**
14. (可能会考到)在各种寻址方式中，指令的地址码字段可能出现的情况如下：
    - 寄存器的编号(寄存器间接寻址)
    - 设备的端口地址(IO指令)
    - 存储器的单元地址(直接寻址，间接寻址)
    - 数值(立即寻址)
15. 寻址方式的总结：

寻址方式	有效地址的计算方式	用途和特点
立即寻址		用于给寄存器赋初值
直接寻址	EA=A	
隐含寻址		缩短指令字长
一次间接寻址	EA=(A)	扩大寻址范围，易于完成子程序返回
寄存器寻址	EA=Ri	指令字长比较短，指令实行速度快
寄存器间接寻址	EA=(Ri)	扩大寻址范围
基址寻址	EA=A+(BR)	扩大操作数的寻址范围，适用于多道程序涉及，常用于为程序或者数据分配存储空间
变址寻址	EA=A+(IX)	主要用于处理数组问题
相对寻址	EA=A+(PC)	用于转移指令和程序浮动
先间址再变址	EA=(A)+(IX)	
先变址再间址	EA=(A+(IX))	

16. CISC和RISC的基本概念：CISC为复杂指令系统(complex)，RISC为精减指令系统(reduce)。CISC常常于多，大，不固定等词语一起出现。但是有一个特殊的抵挡，就是RISC的寄存器多。
17. RISC的主要特点总结：
    - 选取使用频率较高的一些简单指令以及一些很有用但是不复杂的指令，让复杂指令的功能能由使用频率较高的简单指令的组合来实现。
    - 指令的**长度固定**，指令的种类少，寻址方式种类少。
    - CPU中有多个通用寄存器(比CSIC的多)
    - **只有取数\存数指令访问存储器**，其他指令的操作都在寄存器中完成。
    - 采用流水线技术(**注意，RISC一定是采用流水线**)，大部分指令在一个始终周期内完成，采用超标量和超流水线技术，可以使得每一条指令的平均时间小于一个时钟周期。
    - **控制器采用组合逻辑控制，不用微程序控制。**
    - 采用优化的编译程序。
18. CISC的主要特点总结：
    - 指令系统负责庞大，指令的数目一般多大200-300条。
    - 指令的长度**不固定**，指令的种类多，寻址的方式多。
    - 可以访存的指令不受限制。(RISC中只有取数，存数指令访问寄存器)
    - 由于80%的程序使用其20%的指令，因此CISC各种指令的使用频率差距大。
    - 各种指令的执行时间相差很大，大多数指令需要多个时钟周期才能完成。
    - **控制器大多数采用微程序控制。**
    - 难以优化编译生成高效率的目标代码程序。
      (百度了一下，Intel和AMD的电脑cpu和服务器cpu貌似基本都是CISC的，手机和嵌入式用的CPU大多数是RISC的。)
19. RISC与CISC的比较：
    - RISC比CISC更能提高计算机的运算速度。例如，由于RISC寄存器多，因此就可以减少访存次数，其次，由于指令数和寻址方式少，因此指令译码比较快。
    - RISC比CISC更便于设计，可降低成本，提高可靠性。
    - RISC能有效的支持高级语言程序。
    - RISC设计者主要把经理放在那些常用的指令上，尽量是他们具有简单高效的特性。对于不常用的功能，通过组合简单的指令来完成。因此，RISC机器上实现特殊的功能时，效率可能比较低，但是可以采用流水线技术和超标量技术来弥补。而CISC的指令比较丰富。有专用的指令来完成特定的功能。因此处理特殊的任务效率高。

比较内容	**CISC**	**RISC**
指令系统	复杂，庞大	精简，简单
指令数目	一般大于200	一般少于100
寻址方式	一般大于4	一般小于4
指令字长	不固定	等长
可访存指令	不限制	只有Load与Store指令
各种指令使用频率	相差很大	相差不大
各种指令的执行时间	相差很大	绝大多数在一个周期内完成
优化编译实现	很难	较容易
寄存器个数	少	多
控制器实现的方式	绝大多数为微程控制	绝大多数为硬布线控制
软件系统开发时间	较短	较长

5.中央处理器

1. CPU的功能：
   - 控制器能自动的形成指令的地址，并能发出取指令的命令，将对应地址的指令取到控制器中，称为指令控制
   - 取到指令之后，应产生完成每一条指令所需要的控制命令，称为操作控制。
   - 控制命令产后，需要对各种控制命令加上时间上的控制，称为时间控制。
   - 在执行命令的过程中，可能需要进行算术运算和逻辑运算，称为数据加工。
   - 最后还要有处理中断的能力，称为中断处理。
2. CPU的基本结构：控制单元CU，数据加工ALU，中断处理系统，寄存器。
3. 运算器中的寄存器：
   - 暂存寄存器：暂存寄存器用于存放从主存读来的数据，这个数据**不能放在**通用寄存器中，否则会破坏其原有的内容。**暂存寄存器对程序员是透明的**
   - 累加寄存器(acc):它是一个通用寄存器，其功能是为ALU提供一个工作区，累加寄存器暂时存放ALU运算结果的信息。显然，运算器中至少要有一个累加寄存器。目前，CPU中的累加寄存器一般达到了16个到32个，甚至更多。当使用多个累加器时，就变成了通用寄存器堆的结构，其中任何一个可以存放一个源操作数，也可以存放结果操作数，在这种情况下，需要在指令格式中对寄存器编号加以编制。(书上没有说是否透明，我认为时不透明的，因为可以编制，那肯定是要感知到它的存在。)
   - 通用寄存器组：主要用于存放操作数(包括源操作数，目的操作数和中间结果)和各种地址信息等，常见的通用寄存器有AX，BX，CX，DX，以及堆栈指针SP等。通用寄存器对程序员不透明。
   - 状态条件寄存器(PSW)：保存由算术指令和逻辑指令运行或测试结果建立的各种条件码，如进位结果标志，运算溢出标志等..对程序员不透明(百度查的)。
4. 控制器中寄存器
   - 程序计数器PC：在程序开始执行时，必须将程序的起始地址，也就是程序第一条指令所在的地址送入程序计数器，当执行指令时，CPU将自动修改PC中的内容，以便于使其保存的总是下一条政令的地址，由于大多数指令都是顺序执行的，所以所谓的修改就是简单的将PC+1。当遇到转移指令时，后继指令的地址必须从指令的地址段获得，在这种情况下，下一条从内存中取出的指令将由转移指令来规定。因此，PC具有寄存信息与计数两种功能。PC对程序员不透明。
   - 指令寄存器：用来保存**当前正在执行**的指令。对程序员透明。
   - 存储器数据寄存器MDR：用来暂存从主存读出的一条指令或者是数据。对程序员透明。
   - 存储器地址寄存器MAR：用来存储当前CPU访问的内存单元的地址。对程序员透明。

寄存器	**对程序员是否透明**
暂存寄存器	透明
累加寄存器	(不透明)
通用寄存器	不透明
PSW	不透明
PC	不透明
IR	透明
MDR	透明
MAR	透明

(带括号的没有找到资料，我猜测的，不要当真)

5. 取址周期：

```
(PC)->MAR // 将要执行的指令的地址放到MAR。

1->R //发出读命令，这里可以不写

M(MAR)->MDR //将要执行的指令从存储器中读出到MDR，其中(MAR)表示MAR中的内容，M(MAR)就是表示从主存中此地址的内容，即为要执行的指令本身。

(MDR)->IR //将要执行的指令放入指令寄存器

OP(IR)->CU // (IR)表示IR中的内容，就是指令的本身，OP(IR)表示指令的操作码，AD(IR)表示指令的地址码(下面会看到)

(PC)+1->PC// 形成下一条指令的地址。
```

1. 间址周期：主要是为了取出操作数的有效地址，然后到对用的存储器中去取操作数。注意，不是所有的指令执行都对有间址周期，比如立即寻址的指令。

```
Ad(IR)->MAR //将指令字中的地址码送入MAR

1->R //发出读命令

M(MAR)->MDR //将有效地址从主存送入MDR
```

1. 执行周期
   - (以加法指令为例)：假设一个操作数在累加器，另一个操作数在主存单元A中，并且运算结果送入累加器。

```
Ad(ID)->MAR // 将指令的地址码送入主存地址寄存器中

1->R //启动存储器读命令

M(MAR)->MDR //将MAR所指向的主存单元中的内容(操作数)经过数据总线读到MDR

(ACC)+(MDR)->ACC //给ALU发出加命令，将ACC中的内容和MDR中的内容相加，结果存到ACC
 - 存数指令的例子：假设要将上面的ACC的结果放入主存中A地址单元
Ad(IR)->MAR // 将指令的地址码送入主存地址寄存器

1->W // 启动存储器写

(ACC)->MDR // 将累加器的内容送入MDR

(MDR)->M(MAR) //将MDR中的内容放进指定的贮存单元中
```

1. 中断周期：执行周期结束后，CPU要查询是否有请求中断的事情发生，如果有，就进入中断周期。假设程序的断点保存至主存的0号单元，且采用硬件向量法寻找入口地址。

```
0->MAR // 将主存0 号单元的地址送入主存寄存器中

1->W //启动存储器写

(PC)->MDR //将PC中的内容(程序断点) 送入主存数据寄存器

(MDR)->M(MAR) //将MDR中的内容送到对指定的地址单元

向量地址->PC //将向量地址形成部件的输入送至PC

0->EINT //关中断，将允许中断触发器清零
```

```
 - 如果断点存入的不是主存，而是存入堆栈，那么微程序命令0->MAR改为
(SP)-1->SP

(SP)->MAR
```

1. 组合逻辑控制(或称为硬布线逻辑控制)：由基本的门电路组合实现。这种方式实现的控制器的处理速度快，但是电路复杂，制造周期长，不灵活，可维护性差。
2. 微程序控制：仿照程序设计的方法编制每个机器指令对应的**微程序**，每个微程序有若干个**微指令**构成，各微指令包含若干条**微命令**。所有的指令对应的程序只放在只读存储器中，当执行到某一条指令时，取出微程序中各条微指令，译码产生对应的微命令，送到机器相应的地方，控制其动作。这个只读存储器称为控制存储器，微程序控制方式下，控制单元的设计简单，指令添加容易灵活，可维护性好，但是速度较慢。
3. 微程序控制的相关概念：
   - 微命令与为操作：一条机器指令可以分解为一个微操作序列，这些为创造u欧式计算机中最基本的，不可再分割的操作。在**微程序控制的计算机中**，将控制部件向执行部件发出的各种控制命令称为微命令，它是构成控制序列的最小单位。**微命令和微操作是一一对应的**，微命令是微操作的控制信号，微操作是微命令的执行过程。
   - 微指令与为周期：微指令是若干微命令的集合，存放微指令的控制存储器单元地址称为微地址。微指令包含两大部分的信息：操作控制字段,又称为操作码字段，用来产生某一步操作所需要的各种操作控制信号。顺序控制字段：又称为微地址字段，用来控制产生下一条要执行的微指令的的地址。
   - 主存储器与控制存储器：主存储器用于存放程序和数据，在CPU外部，用RAM实现，控制存储器用于存放微程序，在CPU的内部，用ROM来实现。
   - 程序与微程序：程序是指令的有序集合，用于完成某种特定的功能，微程序是微指令的有序集合，一条指令的功能是由一段微程序来实现的。
4. 微指令的基本格式：微指令分为两个字段，一个是操作控制字段，该字段发出各种控制信号，另一个是顺序控制字段，该字段可指出下地址，以控制微指令序列的执行。这个是类似于PC的。
5. 微指令的编码方式：微指令的编码方式又称为微指令的控制方式，他是指如何对位指令的控制字段进行编码，以形成控制信号。
   - 直接编码方式：在微指令的微命令字段中每一位都代表一个微命令，设计微指令时，**选用或者不选用某一个微命令，自选要将表示该微命令的对应位设置为1或者0即可**。因此微命令的产生不需要译码器。直接编码的优点就是简单，直观，执行速度块，操作的并行性好，其缺点时微指令字过长，造成控制存储器的容量极大。
   - 字段直接编码方式：将微指令的微命令段分成若干个小字段，把互斥性的微命令(在同一个微指令周期中不能同时出现的微命令称为互斥性微命令)组合在同一个字段中，把相容性的微命令组合在不同的字段中。每一个字段都独立编码，每一种编码代表一个微命令且字段编码的含义单独定义，与其他的字段无关。这种方式可以缩短微指令的字长，但是因为要用过译码电路后再发出微命令，所以比直接编码的方式慢。注意：一般每一个小段还要留出来一个状态，表示本段不发出任何现行指令。因此，当某段字段的长度为3位时，最多只能表示7个互斥的微命令，通常用000表示不操作。
   - 字段间接编码方式：一个字段的某些微命令需要另一个字段中的某些微命令来解释，由于不是字段直接编码的方式发出微命令，故称为字段间接编码，又称为隐式编码。这种方式可以进一步缩短微指令的字长，但是削弱了微指令的并行控制能力。因此通常做为字段直接编码的一种辅助手段。
   - 混合编码方式：混合编码方式由直接编码于字段编码混合使用。
6. 微指令序列地址的形成：
   - 第一种方式：后续的微指令的地址可以由微指令的下地址字段直接给出。这种方式称为断定方式。
   - 第二种方式：后续微指令的地址还可以根据机器指令的操作码形成，微地址形成部件实际是一个编码器，其输入为指令操作码。即为将机器指令的操作码送入地址译码器进行译码，输出的结果就是对机器指令微程序的首地址。然后拿着首地址去微命令寄存器找到相应的微命令。
7. 微指令的格式：

比较项目	水平型微指令	垂直型微指令
并行性	好	不好
执行指令需要的微指令数目	少	多
和机器指令的相似度	差别很大	相似
最后陈述	用较短的微程序结构换取较长的微指令结构	用较长的微程序结构换取较短的微指令结构

8. 指令流水线会发生的三种问题：资源相关(结构相关),数据相关(数据冒险)，控制相关(控制冒险).
9. 资源相关：所谓的资源相关就是多条指令进入流水线以后在同一个机器时钟周期使用了同一个功能部件所发生的冲突。
10. 数据相关：在一个程序中，如果必须等待前一条指令执行完毕后，才能执行后一条指令，那么这两条指令就是数据相关的。数据相关分成三类：
    - 写后读(RAW):本来因该先写入再读取，但是现在没有写入就读取了(读取了旧的数据)，出现了错误。
    - 读后写(WAR)：本来因该先读取在写入，但是现在是先写入再读取(本来因该是读取旧的数据，但是现在读取了新的数据)，出现错误。
    - 写后写(WAW)：本来因该前面一条指令先写数据，后面一条指令再写数据，但是现在相反，导致了错误。
11. 解决数据相关的技术：最简单的技术是操作延后，但是一般采用数据旁路技术来解决。数据旁路技术就是设置相关的专用通路，即不等前一条指令把计算结果写回寄存器组，而是之际把前面一条指令的计算结果作为输入交给下一条需要此结果的指令。使得本来需要暂停的操作可以继续执行。
12. 控制相关：控制相关是由于**转移指令**引起的，当执行转移指令时，依据转移指令的产生结果，可能顺序的执行下一条指令，也可能转移到新的目标地址取指令，从而使流水线发生断流。**解决方法**：采用猜测法技术，机器先选定转移分支中的一个，按它取指令并且处理，条件码生成后，如果猜测正确，那么流水线继续进行下去，否则之前预先取的指令失效。
13. 流水线的性能指标：
    - 吞吐率：单位时间内流水线所完成的指令或者输出结果的数量。
    - 加速比： 不使用流水线所用的时间 / 使用流水线所用的时间
    - 流水线的效率：时空图上有任务的面积 / 时空图的总面积
14. NB的流水线技术：
    - 超标量流水线：一个时钟周期能并发的执行多条指令。
    - 超级流水线：将机器指令划分为更多级操作，后面的指令只要等待前面的指令完成某一小步后就可以执行，而不是像以前一样要等待一个机器周期。
    - 超长指令字：发掘指令潜在的并行性。将多条并行操作的指令组成一条具有多个操作的代码字段的超长指令字。
    - 动态流水线：多种运算可以同时运行，而静态流水线只是一种运算完成再进行下一种运算。

6.总线

1. 总线的传输周期：指的是CPU通过总线对存储器IO端口进行一次访问所需要的时间。包括总线申请阶段，寻址阶段，传输阶段和结束阶段。
2. 按照链接部件不同可以分为片内总线，系统总线，通信总线等…
    - 片内总线：顾名思义指的是**芯片内部**的总线，如在CPU芯片内部，寄存器与寄存器之间，寄存器与算术逻辑单元之间都是片内总线链接的。

- 系统总线：链接五大部件(运算器，控制器，存储器，输入设备，输出设备)之间的信息传输线。按照系统传输信息的不同，可以分为数据总线，地址总线和控制总线。其中，数据总线是双向的，地址总线的单向的。**地址总线上的代码用来指出CPU欲访问的存储单元或者IO设备的地址，由CPU给出**
- 通信总线：用于计算机系统之间或者计算机系统与其他系统之间的通信。分为串行通信和并行通信。并行通信适用于近距离传输，并行通信适用于串行传输。握手(应答)信号必须在通信总线上传输。

3. 总线的性能指标：
   - 总线宽度：通常指的是数据总线的根数。用bit来表示
   - 总线带宽：单位时间内在总线上传输的数据的位数。
   - 总线服用：地址总线与数据总线共用一组。
   - 信号线数：地址总线，数据总线和控制总线3种总线数的和。
4. 单总线结构：单总线结构与就是将CPU，主存，IO设备都链接在一组总线上，允许IO设备之间，IO设备与CPU之间或者IO设备与主存之间直接交换信息。单总线的特点：由于IO设施与主存共用一组总线，因此主存与IO设备之间是统一编址的。CPU可以像访问内存设备一样的访问外部设备。
5. 双总线结构：双总线结构就是将速度较低的IO设备从单总线中分离出来，形成主存与IO总线分开的结构。
6. 三总线结构：与双总线结构相比，三总线结构增加了一条专门用于IO高速设备与主存直接交换信息的DMA总线。在三总线结构中，任意时刻只能使用一种总线，主存总线与DMA总线不能同时对主存进行存取。IO总线只有在CPU执行IO指令时才能用到。
7. 集中仲裁方式：总线上所链接的各类设备，按照其对总线有无控制功能可以分为主设备和从设备。主设备对总线有控制权，从设备只能相应从主设备发出的总线命令，对总线没有控制权。总线上的信息的传送是由主设备启动的，如果某个主设备欲与另一个设备进行通信，首先由主设备发出总线请求信号，若多个主设备要同时使用总线，就由总线控制器判断。仲裁逻辑按照一定的优先顺序确定哪一个主设备能使用总线。只有获得总线使用权的主设备才能开始传送数据。总线的判优控制有三种：连视查询，计数器查询，独立请求查询。
8. 链式查询：一共就3根控制线，BS总线忙，BR总线请求，BG总线同意。其中BG链式的链接所有IO接口。
   - 链式查询优先级判别方式：离总线控制器越近的部件，其优先级越高，离着总线控制器越远的部件，其优先级越低。
   - 链式查询的优点：只需要3根控制线就能按照一定的优先级实现总线的控制。结构简单，易于扩充。
   - 链式查询的缺点：1对电路故障敏感 2高优先级的设备如果频繁使用总线，那么低优先级的设备将长期不能使用总线。
9. 计数器查询方式：计数器定时查询方式采用一个计数器控制总线的使用权，相对于链式查询的方式多了一组设备地址线，少了一根总线同意线。各个设备共用一个请求线。
   - 计数器查询的优先级判别方式：当总线控制器接收到总线请求信号判断总线不忙时，计数器开始计数，计数值通过一组地址线发向各个部件。当地址线上的计数值于请求使用总线设备的地址一致时，该设备获得总线控制权，同时，终止计数器及查询工作。
   - 计数器有两种控制方式，每次从0开始计数与每次从上一个数字开始计数。当使用第二种方法时，所有设备使用总线的优先级相等。计数器的初值可以用程序设置，所以优先级顺序是可以改变的。
   - 计数器查询方式的优点：各个设备的优先级可以改变，且对电路故障不敏感。
   - 计数器查询方式的缺点：增加了控制线，多了log2N条地址线，控制器也比较复杂。
10. 独立请求方式：每一个设备都有一对总线请求信号BR和总线同意信号BG。
    - 独立请求优先级的判别方式：当总线上的部件都需要使用总线时，经各自的总线请求线发送总线请求的信号，在总线控制器种排队。当总线控制器按照一定的优先顺序决定批准某个部件的请求时，则给该部件发送总线响应信号。该部件接收到了此信号就获得了总线的使用权，开始传输数据。
    - 独立请求方式的优点：响应速度块，对优先级控制相当灵活。
    - 独立请求方式的缺点: 需要2n+1根控制线。总线控制更复杂。

11. 同步定时方式：同步定时方式是指采用一个统一的时钟信号来协调发送方和接收方的传送定时关系。时钟信号通常是由CPU的总线控制器发出，然后送到总线上的所有部件。
    - 同步通信方式的优点：传输速度快，具有较高的传输速率。
    - 同步通信方式的缺点：同步方式必须按照最慢的模块来设计公共时钟，当总线上的模块存取速度差别很大时，便会大大的损失总线的效率，**并且不知道被访问的外部设备是否真正的响应，所以可靠性较低。**(以前复习貌似遗漏了这个点，我一直以为可靠性因该是最高的…)
    - 同步通信的使用范围：总线长度较短，总线所链接的部件存取时间都比较接近。
12. 异步通信方式：
    - 不互锁方式：**主模块的请求信号和从模块的回答信号之间没有制约关系。**主模块发出请求信号后，不必等待从模块的应答信号。而是经过一段时间后，自动撤销。(默认从模块收到了该信号)
    - 半互锁模式：主模块发出请求信号后，必须接收到从模块发出的应答信号才会撤销请求信号。
    - 全互锁模式：主模块发出请求信号后，从模块收到该信号，然后一直向主模块发出应答信号，主模块收到应答信号后撤销请求信号，同时再发送一个回答信号，从模块收到该应答信号后才会撤销应答信号。
13. 异步通信方式可以用于串行或者并行传输。
14. 系统总线标准：
    - ISA：工业标准体系结构总线，最早出现的微型计算机的系统总线标准。
    - EISA：扩展的ISA总线，是配合32为CPU而设计的总线扩展标准。完全兼容ISA。
    - VESA：VESA局部总线标准是一个32位标准的计算机局部总线，是针对多媒体PC要求高速传输活动图像的大量数据应运而生的。
    - PCI：PCI局部总线式高性能的32位或64位总线，是为高速集成的外部部件，扩充查办和处理器/存储器系统而设计的互联机制。支持即插即用并且可以对数据和地址进行奇偶效验。
    - PCI-Express：最新的总线和接口标准，这个标准将全面取代现在的PCI和AGP，最总实现总线标准的统一。
15. 设备总线标准：
    - IDE：集成设备电路是一种IDE接口磁盘驱动器接口类型，用于处理器和磁盘驱动器之间。
    - AGP：加速图形接口是一种视频接口标准，用于链接主存和图形处理器。
    - USB接口：串行接口，可以并联，热拔插，即插即用。
    - SATA：一种硬盘接口规范。

7.IO

1. 操作系统最后一节的总结基本都包含这部分内容了,这里就不再重复了....