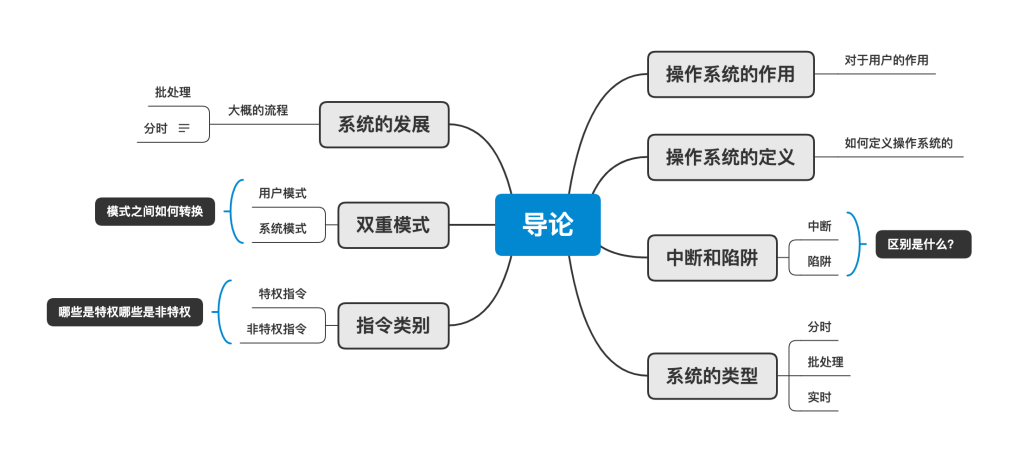
H1

操作系统

H2

导论



操作系统的作用

1. 对用户： 方便编程和执行用户程序
2. 对操作系统：统一管理资源 管理I/O操作设备 提高资源利用率

H3

操作系统的定义

1. 资源分配器： 操作系统管理计算机的资源
2. 控制程序：控制I/O设备
3. 一直运行的程序称之为操作系统的内核，外壳是操作系统和用户的接口程序

H3

计算机系统的存储结构

1. 高速缓存：匹配cpu的速度和内存的速度

H3

多道程序设计

操作系统最重要的特点是多道程序设计能力，他通过组织作业总能使一个进程运行。

1. 独立：同时存放多道程序。
2. 宏观上并行：同时运行几个程序，并且他们都没完成。
3. 微观上串行：程序轮流分时的占用cpu运行

H3

中断和陷阱

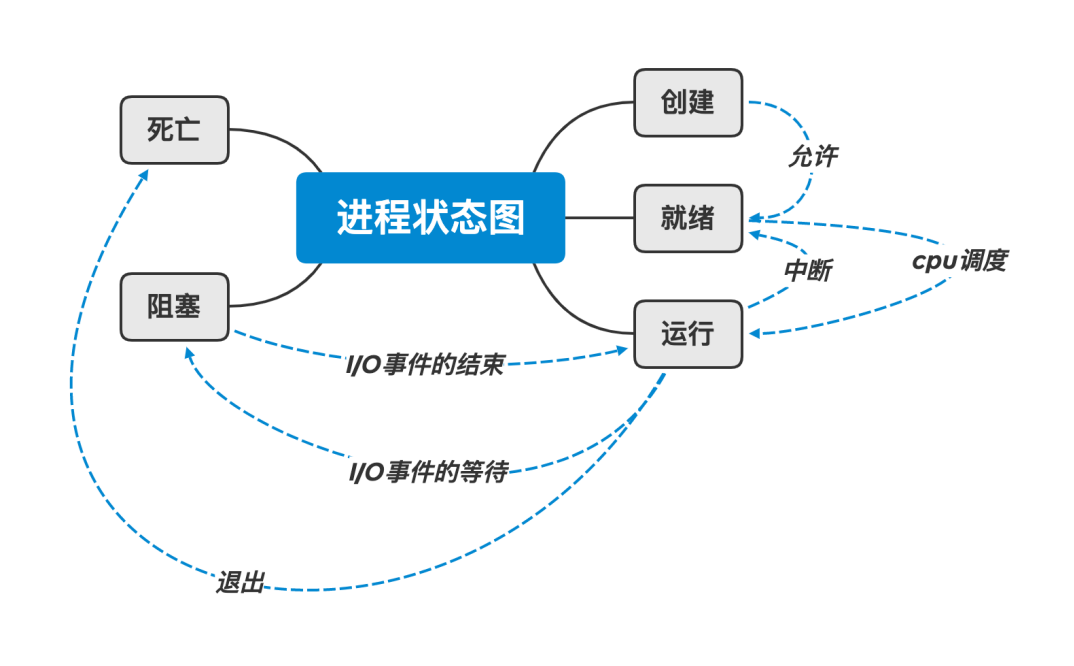
1. 系统造成的叫中断 系统就通过一系列的操作运行 中断会运行中断服务函数 异步、随时都有可能产生中断 I/O中断等等
2. 用户编写程序造成的 同步、在固定的点产生陷阱 段错误、指针越界等等

H3

中断

操作系统是中断驱动的。中断对于操作系统的重要不言而喻，没有了中断，操作系统就是顺序执行的。

1. 程序的执行进行以下的步骤：在内核模式进行程序的初始化-->初始化完成交由用户模式程序执行-->遇到中断、陷阱、系统调用程序又再次回到系统模式
    1. 中断：程序需要进行I/O等
    2. 陷阱：程序运行出错，交由系统进行中断服务程序（释放空间、保护内存等）
    3. 系统调用：用户状态自陷进内核状态，在内核状态下进行一些系统的指令
2. 操作系统如果不中断驱动的话会怎么样？那cpu在空闲的时候就要采取一种轮询的模式，在空闲的时候向所有的硬件设备发起询问，看看任务是否完成。这显然是不合理的。



双重模式及指令类别

即用户模式和管态模式，操作系统在这两种模式之间不断切换，用户在需要执行系统指令的时候要进入管态模式才能执行系统指令。在用户模式下需要执行访管指令才能进入管态模式，访管指令改变了PSW。

H3 **操作系统的访管指令只是改变了PSW。在用户模式下的命令不一定都是非特权指令。访管指令既不是非特权指令，也不是特权指令。**

> 特权指令都是一些底层的一些非常危险的容易导致操作系统崩溃、需要小心操作的指令。
>
> 以下为特权指令：
>
> 1. 资源的控制
> 2. 重要寄存器的读取
> 3. 中断的控制执行
> 4. 访问程序状态的指令
> 5. 内存管理
> 6. 时钟的管理
>
> 一下为非特权：
>
> 1. 访管指令
> 2. DMA操作
> 3. 改变磁盘空间分配
> 4. 计数器清0，写程序计数器
> 5. 取指令 取操作数 写内存

用户和造作系统的界面

每个操作界面都有用户界面，用户界面能够有多重形式。

1. 命令行界面：采用文本命令
2. 批处理界面：等于组合起来的文本
3. 图形用户界面

H3

解释程序被称为外壳。

现代操作系统的四大特性

1. 并发：多个程序同时运行，但是都没结束
2. 虚拟：将物理设备抽象成逻辑设备
3. 共享：进程共享操作系统的资源
4. 异步：理解为中断随时可能发生，每次的结果可能都不一样。

H3

操作系统的类型

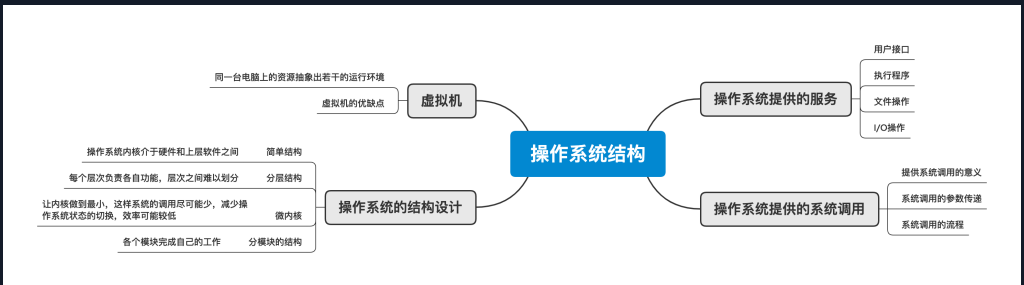
1. 批处理系统：将一批作业交给操作系统之后就不再干预，有单道批处理和多道批处理，提高了cpu的利用率。
2. 分布式操作系统：将物理上独立的计算机联合起来，可以并发的做一个事，执行一个任务。

H3

3. 分时操作系统：每个用户都可以向终端发送命令，完成作业的运行。
4. 实时操作系统：在指定的时间内对外部做出反应。

H2

操作系统结构



操作系统提供的系统调用

系统调用可以大致分为6类：

H3

1. 进程控制
2. 文件管理
3. 设备管理
4. 信息维护
5. 通信
6. 保护

为用户提供的系统调用接口：

1. 命令级调用接口：用户通过外壳进行调用
2. 程序级调用接口：用户编写代码进行调用

意义：

1. 方便用户编程
2. 隐藏细节

调用过程的参数传递：

1. 表传递：放到表里传递调用

2. 寄存器传递：参数较少时，速度估计比较快
3. 堆栈传递：调用压栈、使用出栈

操作系统结构设计

操作系统的设计要考虑策略和机制，策略即要干什么，机制即要怎么去干。

虚拟机的特点

H3

虚拟机的特点：

1. 每个机子就像独立存在，拥有独立的资源
2. 实现基于底层机子资源的共享
3. 提供与裸机共同的接口

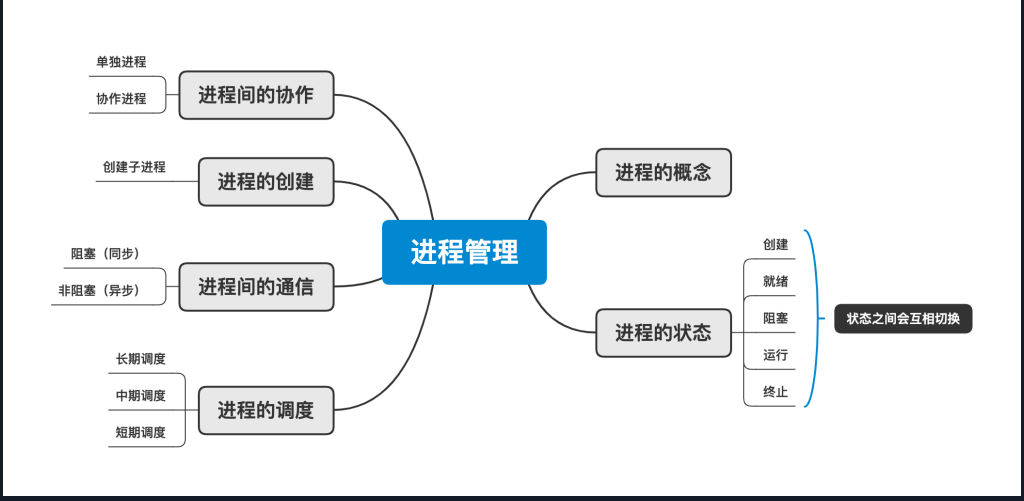
H3

虚拟机的优缺点：

1. 单独运行不必中断系统
2. 实现困难
3. 单独运行可以保护资源

进程管理

H2



进程的概念

进程是执行中的程序，是现代分时系统的工作单元。

进程不只是程序执行时的代码段，还包括当前指令、堆、栈、数据段。

H3

进程的状态

创建　就绪　阻塞 运行　终止

运行遇到I/O进入到阻塞等待状态，随后逐步恢复到就绪状态

H3 运行遇到抢占、时间片、定时器（中断）重新进入到就绪状态

理解状态之间的切换过程。进程上下文的切换需要保存PCB。

进程控制块

H3

1. 进程状态
2. 程序计数器
3. CPU寄存器
4. CPU调度信息
5. 内存管理信息
6. 记账信息
7. I/O状态信息

调度程序 是一个程序!

进程在声明周期中会在各种调度队列中迁移，操作系统为了调度必须按照一定方法来从这些队列中选择程序。进程的选择通过适当的调度程序执行。

H3

1. 长期调度程序：长期调度程序从缓冲池中调度程序到就绪队列。长期调度程序是认真的，他会合理搭配I/O密集进程和cpu密集进程。批处理系统中大量应用。
2. 中期调度程序：将进程从内存中移出，移动到阻塞队列（不在内存里），不就之后又会将其恢复，从中断的地方开始运行。降低多道程序设计难度。分时系统中使用较多。
3. 短期调度程序：从准备执行的进程中选择进程，分配给cpu调度。

进程之间的协作

独立进程：执行期间不会被其他进程影响。

协作进程：执行期间会受到其他进程的影响。

H3 优点：

1. 信息共享
2. 模块化
3. 方便

进程的创建

父进程创建fork创建子进程。

- 僵尸进程

  子进程结束时，父进程没有通过WAIT函数等获取子进程的状态，子进程一直处于最后的阶段，信息没有被读取清空，一直驻留在内存中，成为僵尸进程，父进程需要在必要的时候回收子进程的状态，避免产生僵尸进程。

H3

- 孤儿进程

  父进程已经结束，但是子进程还没有结束，子进程交由1号进程管理，最后会被回收，孤儿进程是无害的。

进程之间的通信

简单的理解通信无非直接通信与间接通信。

- 直接通信

  点对点的通信。

H3

- 间接通信

  中间增加缓存，发送方放入缓存中，接收方从缓存中取。

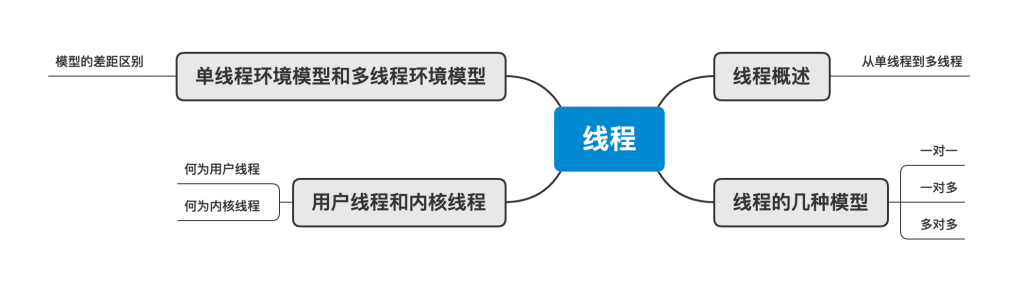
传递消息的方式可以是阻塞的也可以是非阻塞的，阻塞的称为同步，非阻塞的称为异步。

同步发送：程序不向下进行除非消息被接受或消息发送至邮箱。

同步接受：程序不向下进行除非接收到消息。

线程

H2



线程概述

概念：线程是CPU使用的基本单元，具有程序计数器、线程ID、栈、寄存器等。

线程具有独立的栈和寄存器：这个寄存器指的不是CPU的寄存器，是指进程每次在受到CPU调度的时候记录下来的CPU的寄存器信息和当次调度的栈信息。

H3

用户线程和内核线程

用户线程：用户态的线程，线程的运行可以由用户控制，线程库的所有代码和数据结构都位于用户空间中。不需要内核支持。

内核线程：内核支持的线程，由cpu调度管理。

H3

线程的几种模型

1. 一对一

   一个用户线程对用一个内核态进行管理，真正的可以实现放到多个核上进行多线程，等于一个个轻量级的进程，这些线程一个线程的阻塞并不影响其他的。

H3 2. 一对多

   多个用户线程对应一个内核态，多个线程对于内核来说还是只是一个进程，一个线程的阻塞会影响到其他线程，其他线程跟着阻塞，可以形象的认为这是一种“假的多线程”。

3. 多对多

   对个用户线程对应多个内个态，内核态的数目小于等于用户态的数目，每次内核态准备调度时，都会携带一个用户进程进行调度。

单线程环境和多线程环境的模型差异

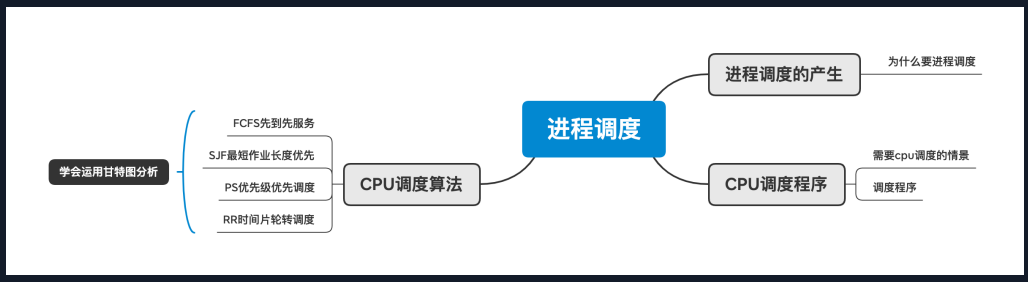
单线程环境的进程模型：进程控制块、进程数据块、用户栈、内核块

多线程环境进程的模型：进程控制快、进程数据块、{每个线程拥有：每个线程独立的栈、线程控制块、内核态}

H3

进程调度

H2



进程调度的产生

为什么要调度？CPU在执行程序的时候，不可避免的会遇到I/O操作，然后I/O的等待阻塞当前的线程，这个时候想必就没有必要把CPU的宝贵资源浪费在这个阻塞的进程上了吧。这时候就很有必要进行CPU的调度。

H3

CPU调度程序

1. 需要调度CPU的情景
    1. 进程由阻塞转为就绪，比如说I/O完成
    2. 进程由运行转为阻塞，比如说等待I/O
    3. 进程由运行转为就绪，比如说中断
    4. 进程由运行转为死亡，比如进程终止

    如果进程的调度只发生在2和4成为非抢占的调度，否则称之为抢占式的调度。1、3两个情况都都类似于中断的发生，而要知道中断是可以关闭的。
2. 调度程序
    1. 调度程序是一个小的模块
    2. 专门将CPU的控制交给短期调度
    3. 功能包括
        1. 切换上下文
        2. 切换到用户模式
        3. 定位到用户模式的位置，以便重新启动程序

CPU调度算法

要会画甘特图

1. 先来先服务：直接根据先到的时间决定谁先服务
2. 最短作业长度优先：根据最短的执行时间决定调度的顺序，分为抢占的和非抢占的，抢占的也成为最短剩余时间调度
3. 优先级调度：顾名思义，分为抢占和非抢占的
4. RR调度：根据先到达的时间，然后每个进程运行在时间片内运行调度，每个进程在运行完一个时间片之后一定会被别的进程抢占，因此RR调度是抢占的算法

H3

同步

H2

同步：解决进程并发执行的异步问题。进程推进问题。

互斥：解决进程并发执行的时候对临界资源的访问问题。

软件解决互斥问题

1. 单标志法：不满足空则进入原则（一定要按顺序来）、等则让权。
2. 双标志法：双标志将判断、上锁分为两个步骤，由于判断和上锁不是原子性的，在执行期间可能会有进程的调度，因此可能会出问题。

H3

    1. 先检查法：先进行检查，若在检查之后马上发生调度，那么两个进程会同时进入临界区。不满足忙则等待。
    2. 后检查法：后进行检查，先上锁。如果在上锁之后马上发生调度，那么两个进程将都无法进入临界区，产生饥饿现象。不满足空则进入。

3. 彼得森算法：类似双标志法的先上锁后检查法，后检查的时候采取孔融让梨的思想，如果对方想进入就让对方进入。不满足让权等待。

硬件解决

1. 中断屏蔽：直接把中断关掉，只能在内核里实现，因此只能在内核中的进程实现。
2. TS、TSL：即TestAndSetLock，不断为临界区上锁，并返回原先的锁值，如果原先的锁值变为false才能继续进入临界区。是由硬件实现的。不满足让权等待。
3. Swap：思想同上。不满足让权等待。

H3

死锁

H2

死锁产生的条件

1. 互斥
2. 占有并等待
3. 循环等待
4. 资源非抢占

H3

缺少其中的一个条件就不会产生死锁。

死锁问题的处理方案

1. 死锁预防与死锁避免

H3

求大神告知计算机基础中死锁的预防，避免，监测的含义和不同?

流沙，活跃在评论区

刚好在复习这方面的内容。顺便答一下。
死锁的预防是至少破坏死锁产生的四个必要条件之一来预防死锁的发生。通常通过调整对锁（资源）的请求和处理代码来实现。
死锁的避免是在动态考虑每个进程或线程的资源请求。如果当前请求不会造成死锁就允许。如果会造成死锁就不允许。调度器需要事先了解线程或进程的资源需求。
死锁的检测是检测系统中是否存在死锁。

你可能会迷惑死锁的预防和避免有什么区别。打个比方吧，要想马路不堵车，预防的方法是每辆车都遵守规则，避免的方法是让交警站在马路中间指挥每辆车，交警会根据路况判断那辆车该走，哪辆得等一会。
----感觉答的不好，以后再改

预防和避免都是为了不出现死锁，预防是制定协议规则，让进城按照这样的规则去调度。避免是采用算法对进城的资源调配进行管控，让进程的调度不出现死锁。

2. 死锁检测与死锁恢复
3. 由于发生死锁的概率很小，就不考虑这种情况

死锁预防

预防死锁即制定规则，去否定产生死锁的四个条件

1. 否定互斥：大多数资源要求都是互斥的，这个很难否定
2. 否定占有并等待：
    1. 在申请新的资源的时候，先释放自己拥有的所有资源
    2. 在申请资源的时候获得所有的资源

H3

这两种会导致资源利用率低，产生饥饿。

3. 否定资源非抢占：规定一个进程在申请新的资源的时候，允许他的现有资源可以被抢占，仅有少数的操作系统支持。
4. 否定循环等待：要求对资源类型进行排序，每个进程要以递增顺序申请资源，这样就不会产生循环。

死锁避免

安全状态一定不会出现死锁，非安全状态不一定会出现死锁。

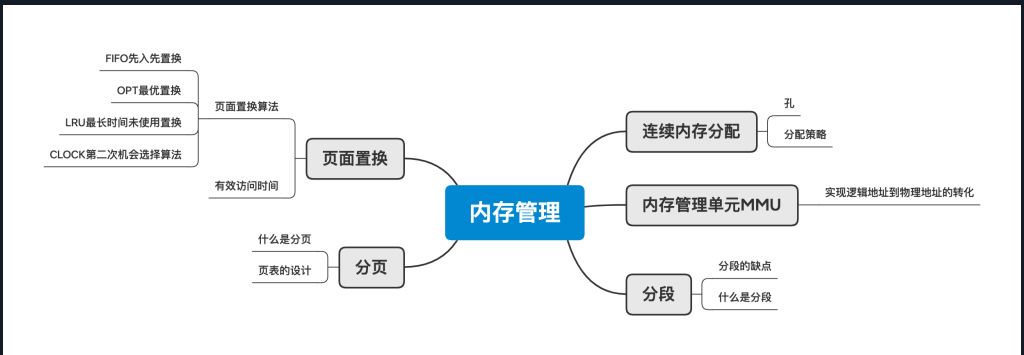
注意银行家算法和安装状态监测算法。

H3

死锁检测

一样用到资源图和银行家算法，具体步骤差不多，多写题目。

H2 内存管理



忽略虚拟内存地址是怎么产生的

存储管理应该具有的功能是：内存的分配与回收、地址映射、内存保护、内存扩充。

背景

CPU根据程序计数器的值从内存中提取到指令，这些指令可能引起对特定内存地址的额外加载与存储。

H3 内存管理单元MMU

实现逻辑地址到物理地址映射的物理单元。

H3 交换

交换有可能让所有进程的总的物理地址空间超过真实系统的物理地址空间，从而增加了多道程序的程度。

交换是从内存换到备份存储中，备份存储通常是快速磁盘。

H3 连续内存分配

保护：即保护内存不被越界访问，用过界限寄存器、基址寄存器、重定位寄存器。

只有操作系统可以通过特殊的特权指令，才能加载基址寄存器和界限地址寄存器。

H3 将程序分配进孔中。

分配到孔的常用方法：

1. 首次适应：遇到的第一个合适的就放进去。
2. 最佳适应：遇到最合适的放进去。
3. 最差适应：遇到可以把进程放进去的最大的孔放进去。

这种连续的内存分配容易产生碎片。

连续内存分配的进程没有共享代码的能力，因为它不能找到不连续的代码，没有标记什么的，他根本不知道要到哪里去找。

内部碎片与外部碎片

内部碎片的产生：内存中有一块50MB的空间，进程需要48MB，那么该怎么给进程分配呢？如果分出50MB中的48MB，那么剩下的2MB操作系统维护起来也非常困难。干脆就把这2MB也分给进程了，这样这2MB相当于在进程内部被浪费了，被称为内部碎片。

H3

外部碎片的产生：内存中有一块100MB的空间，进程需要70MB，操作系统分配了70MB给他，这30MB就空了出来，但是这30MB太小了，没有别的进程能进去，有好几个这样情况的出现就导致了外部碎片，他们剩下的小空间放不进进程，但是这些小空间合起来还能放好多个，这些进程外的小空间就被浪费了，因此成为外部碎片。

分段

分段是将程序分成代码段、变量段等各个段，然后将各个段放到内存中各自段的位置上，通过< 短号 , 偏移量 > 进行访问。

缺点：分段仍然不能解决外部碎片的问题。

H3

分页

几个容易搞混的点：

1. CPU生成的每个地址包含页码和页偏移，页偏移是逻辑地址提供的，不是页表提供的。
2. 逻辑地址空间可以比实际的物理地址空间大。
3. 页表中的项数=逻辑地址空间被分成的块数。
4. 页表中最多也就存逻辑页号对应的物理地址的映射和有效位。
5. 不要搞混逻辑地址和物理地址，不要搞混逻辑地址和页表项。
6. 操作系统也会维护帧表，表示物理内存中的哪些帧被使用，哪些帧未被使用。
7. 页表长度是由逻辑地址大小得出的。
8. 分页增加了上下文切换的时间。
9. 每个进程都有自己单独的页表，因此有上面的第8点，同时在进程切换的时候，内核会把新的页表地址写入CR3寄存器。

H3

硬件支持：将页表作为专用的寄存器实现 高速硬件缓冲TLB

制定页大小，将进程分成相应的页，将物理内存也分成页大小的帧，将进程的页放在内存中各个帧中。不必连续存放，操作系统维护页表保证对页的访问，通过页表的保护位对内存进行保护。页表中通过逻辑地址物理地址映射、偏移量实现对物理内存的访问。

缺点：分页的方式避免了外部的碎片，但是产生了内部的碎片。

共享页：几个进程有公共执行的代码部分，他们页表的几个页都指向物理内存的同一个地方。

> 页表的几种实现方式
>
> 起源：简单的列表就是连续的分布在内存上，页数过多会导致页表太长
>
> 1. 分层分页：将页表分级，比如二级页表、三级页表，这样在有大多数无效位的时候可以减少列表的长度。减少页表的大小，部分第二级页表没有用到的时候可以直接不分配空间，同时第二级页表可以不放在内存里，可以放在磁盘里，内存中只存第一个二级页表（因为用的比较多）。
> 2. 哈希页表：将页号进行hash，让后放在相应的位置，相同hash值的页用链表组织，hash的特点是访问比较迅速，同时页表项也不用集中放在一个地方。
> 3. 倒转页表：一反常态，根据物理地址的标号去找最终的物理地址。

页面置换

页面置换算法：

1. FIFO先来先滚蛋：在需要牺牲置换的时候，将最早进入内存的置换掉。
2. LRU最长时间未使用滚蛋：在每次访问的时候都更新内存中页面的访问时间，将最久时间没有访问的给置换掉。
3. OPT最优置换算法：该算法难以设计，要先知道未来确切的访问序列，每次置换的时候都选择将未来不再使用的界面置换，如果没有可以置换的，那就把先进来的给置换了。
4. CLOCK第二次机会选择置换算法：每个页面进来的之后都将引用位置为1，每次访问页面的时候也都将页面的引用位置为1，每次产生缺页置换的时候，都去找引用位为0的置换，任何页面在第一次的时候都不会被置换，因此成为第二次机会算法，在替换之后，引用位搜索的指针要指向下一位置。

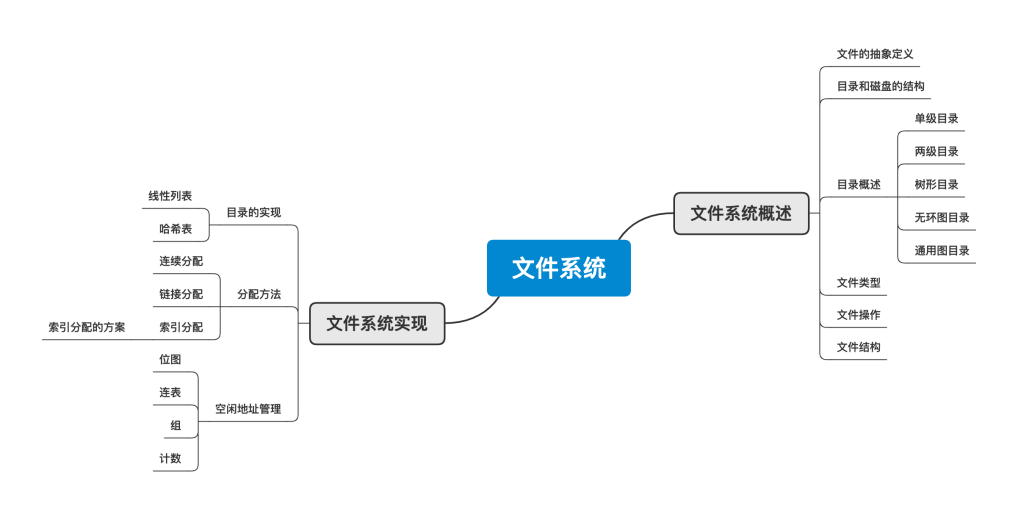
H3

有效页面访问时间：

有效页面访问时间 = （1 - 页面出错概率）* 内存访问时间 + 页面出错概率 * 错误处理时间。

H2

文件系统



文件的抽象定义

文件是磁盘资源的抽象，是信息存储的统一逻辑接口。

H3

目录：应当包含文件的各种信息。

文件类型：拓展名，操作系统需要知道拓展名才能进行相关的操作，同时也能节省一部分空间。

文件操作：读取，写入，重定位，创建文件，删除文件，截断文件

文件结构：简单结构、简单记录结构、记录树结构、复杂结构，插入控制字符实现后两种结构

目录概述

H3

1. 单级目录：简单，会有目录过大，不能重名的问题。
2. 两级目录：根据用户先创建子目录，在在子目录下面创建文件目录，解决重名问题，一定程度解决保护的问题。
3. 树形目录：将二级目录拓展到任意高度，每个文件具有唯一的路径名。相对路径和绝对路径。
4. 无环图目录：资源共享的问题。
5. 有环图目录：面对同样的问题

目录的实现

1. 线性列表：单个属性存放文件的相关信息，用列表的方式存储属性。结构简单，找起来有点困难。
2. 哈希表：每个文件对应一个hash值，根据hash值来查找文件的相关属性。类似map结构，给出key就能快速找到value。需要解决文件值的hash冲突问题。

H3

文件存储空间分配

1. 连续分配：支持顺序访问，随机访问，但是文件不可拓展，寻道时间最少，会有空间的浪费。
2. 链接分配：每个块的最后指向下一个块，支持顺序访问，不支持随机访问，文件可以拓展，存在块损毁数据丢失问题，分配的指针会消耗一定的块空间。

H3

    1. FAT：FAT即文件空间分配表，增强版的链接分配。每个卷开头部分的磁盘用来存储FAT。目录记录起始块，表中记录起始块的开始位置和下一个块的位置。支持随机访问，分配新块只需要找到FAT为0的地方。如果不引入缓存，每次开始都要到卷开头读取FAT，然后在做访问，会消耗大量的时间。
3. 索引分配：用索引块记录文件每个块的分配位置，支持直接访问，并且不会产生外部碎片的问题，但是索引块有索引块的开销，每个文件不管大小都要有一个索引块。采取一些优化的机制：
    1. 链接方案：支持大文件，将索引块链接起来，最后一点地方链接至下一个索引块。
    2. 多级索引：类似于多级页表，节省空间，同时也支持大文件。
    3. 组合方案：Unix采取的方案。前12个直接指向文件块，后3个分别1级、2级、3级索引指向，能存放超大的文件。

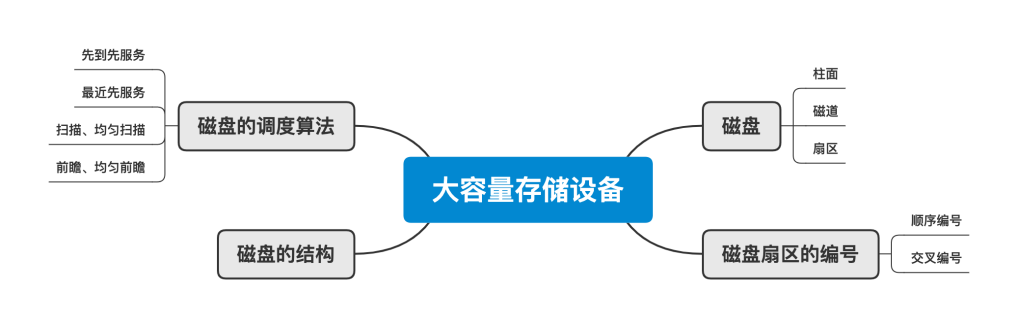
空闲空间管理

1. 位图：位图中每个元素的1和0代表每个块是否被分配出去。
2. 链表：用链表的方式链接空闲空间。
3. 组：几个连续的空闲块为1组，在最后有指向下一组的指针。
4. 记录：记录空闲块的开头和后面连续空闲块的数量n。

H3

大容量存储设备

H2



磁盘

1. 磁道：盘面逻辑的分成一个个圆形的磁道
2. 扇区：磁道在划分成一个个角度相同的扇形就称作扇区
3. 柱面：不同盘面上的同一磁臂位置上的所有磁道构成了柱面

H3

磁盘扇区的编号

编号的顺序都是（柱面、磁道、扇区），即先编完同一柱面同一磁道上的所有扇区，再编完同一柱面上的所有磁道，再编完所有的柱面。

1. 顺序编号：按上面的规则，扇区顺时针，磁道从上到下，柱面从外到里进行编号。

H3

2. 交叉编号：不是顺序的编号，从磁盘读取数据到加载到内存采用的是DMA，假设加载到内存需要时间t，在这t个时间里，由于磁盘一直都在转动，如果连续编号的话，需要读取的属于同一个文件的数据块可能会在t个时间内被错过。尽可能在t个时间内不让下一块出现，最好等t个时间过后刚刚好出现，这样能够使效率达到最大。

磁盘的结果

当做逻辑块的一维数组即可。

H3

磁盘调度算法

访问时间主要包括两部分：

1. 寻道时间：找到目标扇区柱面的时间
2. 旋转延迟：旋转目标扇区到磁头下的时间

H3

调度算法：

1. FCFS：先到先服务，逻辑简单，效率不高。
2. SSTF：最短寻道时间优先，总是选取最近的柱面先处理。
3. SCAN、C-SCAN：扫描，均匀扫描，扫描就是先向一个地方扫描，处理扫描路径上的所有柱面请求，到头之后更换扫描的方向进行同样的操作，被称为电梯算法，可能产生饿死的现象。均匀扫描是先向一个地方扫描，处理了路径上的所有请求，到头之后回到起点，再向同一个方向扫描，一定程度上均匀了等待时间。
4. LOOK、C-LOOK算法：前瞻、均匀前瞻算法，和扫描算法类似，唯一区别就是LOOK算法不是扫描到头返回，而是前面没人之后就返回，因此每次处理完请求之后还要判断前面有没有人。

算法的优劣：

1. 更多的采用SSTF

2. 在有大量的密集请求的时候不宜采用LOOK算法，大量的判断浪费时间。
3. 某块区域集中请求也不宜采用SCAN算法，容易造成饿死。

更应该依具体情况制定算法

H2 I/O系统

轮询 中断 DMA

假脱机：

假脱机是保存设备输出的缓冲区，这些设备，如打印机，不能接受交叉的数据流。虽然打印机一次只能执行一个任务，但是多个应用程序希望并发打印出。假脱机使得应用程序先将所有东西输入到假脱机，在输入完成后再由假脱机转到打印机打印。假脱机等于是一个缓存，缓存里面的文件排好序，等待被调入打印机打印。