操作系统定义

控制和管理整个计算机系统的硬件与软件资源，合理地组调度计算机的工作与资源分配，进而为用户和其它软件提供方便接口与环境的程序集合

1. 计算机系统资源的管理者（处理机管理，存储器管理，文件管理，设备管理）

2. 用户与硬件系统之间的接口（命令接口，程序接口如GUI）

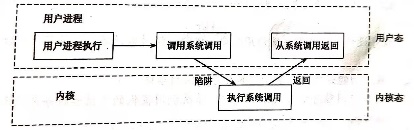
3. 操作系统用作扩充机器

系统调用定义

用户可以执行陷入指令（trap）来发起系统调用，请求操作系统提供服务，把CPU的使用权主动交给操作系统内核程序

用户态->核心态

如发生中断，执行系统调用，user产生错误或企图执行特权指令



操作系统的分层结构

Kernel Model和User Model

宏内核与微内核

Monolithic/micro kernel

宏内核：将操作系统的主要功能模块都作为一个紧密联系的整体运行在核心态，从而为应用提供高性能的系统服务。

【优】各模块之间共享信息，利用相互之间的有效特性【缺】层次之间的交互关系复杂

微内核：将内核中最基本的功能保留在内核，而将那些不需要在核心态执行的功能移到用户态执行

【优】分离了内核与服务、服务与服务，接口清晰，降低维护代价，保证可靠性【缺】性能问题，频繁在用户态与核心态切换

基于模块的结构

程序或系统按照功能或其他原则划分为若干个具有一定独立性和大小的模块

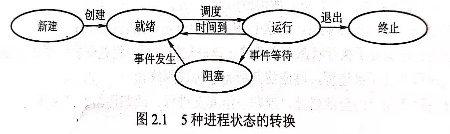
进程的定义

进程（动态）是进程映像（静态）的运行过程，是系统进行资源分配和调度的一个独立单位

PCB进程控制块描述进程的基本情况和运行状态，是进程存在的唯一标志

进程映像=程序段+相关数据段+PCB

进程状态转换



运行态：一次只能有一个进程处于CPU中

就绪态：进程获得了除CPU外的所有资源

阻塞态/等待态：进程正在等待某件事的发生，如某资源可用或IO完成

创建态：进程正在被创建。申请一块空白PCB并填入控制和管理进程的信息->系统分配资源->转入就绪态

结束态：进程正在系统中消失，之后再释放/回收资源

就绪态->运行态：进程获得CPU资源

运行态->就绪态：时间片用完，进程被迫让出CPU资源

运行态->阻塞态：进程等待某资源/事件

阻塞态->就绪态：进程请求的某一资源被分配或等待事件发生时

运行态到阻塞态是主动行为，反之则为被动行为

Linux系统中进程的表示

每个进程对应一个task\_struct结构体

struct task\_struct{

    //这个是进程的运行状态，-1代表不可运行，0代表可运行，>0代表已经停止。

    volatile long state;

    /\*  flags是进程当前的状态标志,具体如下:

        0x2表示进程正在被创建，0x4表示进程正准备退出，0x40表示此进程被fork出，但是并没有执行exec，0x400表示此进程由于其他进程发送相关信号而被杀死  \*/

    unsigned int flags;

    unsigned int rt\_priority;//进程的运行优先级

    struct mm\_struct \*mm; //进程内存使用的相关情况

    pid\_t pid;//进程号,是进程的唯一标识

    pid\_t tgid;//进程组号

    struct task\_struct \*real\_parent; //该进程的"亲生父亲"

    struct task\_struct \*parent;//该进程现在的父进程

    struct list\_head children;//该进程孩子的链表

    struct list\_head sibling;//其父进程的所有孩子的链表

    struct task\_struct \*group\_leader;//主线程的进程描述符

    struct   list\_head thread\_group;//该进程所有线程的链表

    cputime\_t utime,stime;//使用cpu时间的信息，utime是在用户态下执行的时间，stime 是在内核态下执行的时间

    char comm[TASK\_COMM\_LEN];//保存该进程名字的字符数组

    struct files\_struct \*files;//打开的文件相关信息结构体

    struct signal\_struct \*signal;//信号相关信息的句柄

    struct sigband\_struct \*sighand;

 };

进程间通信机制

共享存储：进程间有一块可直接访问的共享空间。操作系统提供可共享使用的存储空间和同步互斥工具，数据交换由用户自己安排W/R指令完成

IPC消息传递：以Message为单位传递

直接通信方式：进程->对方的消息缓冲队列->目标队列

间接通信方式：进程->中间实体->进程

管道机制：互斥、同步、确定对方存在

线程的定义

基本的CPU执行单元，也是程序执行流的最小单位，由线程ID、程序计数器、寄存器集合和堆栈组成。是进程中的一个实体，自己不拥有系统资源，可以创建、撤销其他线程，在运行中存在间断性，也有就绪、阻塞、运行三个基本状态

线程与进程

调度：线程是独立调度的基本单位，进程是拥有资源的基本单位

资源：进程拥有资源，线程不拥有系统资源但可以访问其进程的资源

并发性：都可以并发，提高了系统吞吐量

开销：创建线程的开销比较小，同进程内的多个线程可以共享进程的地址空间，且同步和通信比较容易

地址空间：进程间相互独立，进程内各线程可以共享

通信：IPC需要进程同步和互斥手段的辅助，线程间可以直接RW进程数据段来通信

线程特点

1. 轻型实体，不拥有资源但是有唯一的标识符与控制块

2. 同一个服务程序被不同用户调用的时候，操作系统把它们创建成不同的线程

3. 同一进程中的各个线程共享该进程所拥有的资源

4. 线程是处理机的独立调度单位，可以并发执行，若多CPU同时为一个进程服务可缩短进程处理时间

5. 线程被创建后，便开始其生命周期直到结束

多线程模型

many to one:多个用户级线程映射到一个内核级线程【优】线程管理在用户空间，效率较高【缺】若有线程在使用内核服务时阻塞，则整个进程都会被阻塞，无法进行多线程并发

one to one:将每个用户级线程映射到一个内核级线程【优】并发能力较强【缺】创建线程的开销大，影响性能

one to many:将n个用户级线程映射到m个内核级线程（n>m）【优】集前两者之所长

CPU调度评价指标

CPU利用率：尽量使资源利用率高

系统吞吐量：单位时间内CPU完成作业的数量，调度算法与方法对其影响较大

周转时间：作业提交到作业完成所需时间

周转时间=作业完成时间-作业提交时间内

平均周转时间=总周转时间/作业个数

带权周转时间=周转时间/实际运行时间

平均带权周转时间=总带权周转时间/个数

等待时间：进程处于等待处理机状态时间

响应时间：用户提交请求到系统首次产生响应所用时间

指标优化目标

满足特定系统用户的要求，考虑系统整体销量表，考虑调度算法的开销

经典调度算法

FCFS先来先服务：从就绪队列中选择最先进入该队列的进程，不可剥夺性【优】算法简单，对长作业与CPU繁忙型作业有利【缺】效率低，不利于CPU繁忙型作业

SJF短作业优先：从就绪队列中选择估计运行时间最短的进程，直到完成或发生阻塞【优】平均等待时间、平均周转时间最短【缺】长作业容易出现饥饿，不保证紧迫性作业，只是根据用户提供的估计时间判断

Priority优先级：选择优先级最高的作业

非剥夺式优先级调度算法：即使有更优的进程出现，会让前一进程结束

剥夺式优先级调度算法：有更优的进程出现时，立即暂停当前进程

静态优先级：在创建进程时确定优先级

动态优先级：动态调整优先级

系统进程>用户进程，交互型进程>非交互型进程，IO型进程>计算型进程

高响应比优先：

利于短作业，兼顾长作业，先到先服务

时间片轮转：适用于分时系统，按到达顺序将就绪队列排序，运行固定时间片【时间片大】退化为FCFS【过小】进程切换开销过大【因素】系统响应时间，就绪队列中的进程数目，系统的处理能力

多级反馈队列调度：时间片与优先级的综合1. 设置多个就绪队列并赋予不同优先级2. 优先级越高则时间片越小 3. 一个新进程进入内存后，放入1级队列的末尾，若能在一个时间片中完成则撤离，否则转入下一级队列末尾4. 最底层队列采用时间片轮转运行5. 只有高级队列清空时，才运行低级队列

产生同步问题的情况

直接制约关系，进程间因需要在某些位置上协调工作次序而等待、传递信息而产生的制约关系

资源竞争的3个问题

互斥：当一个进程进入临界区使用临界资源时，另一个进程必须等待。【空闲让进】【忙则等待】【有限等待】【让权等待】

死锁、饥饿

临界区问题：

定义：一次仅允许一个进程使用的资源称为临界区资源，对其的访问必须互斥地进行，访问临界区资源的代码即临界区

四个过程：

do{

    entry section;//进入区，检查可否进入临界区，若能进入则设置正在访问临界区的标志，阻止其他进程同时进入

    critical section;//临界区，进程中正在访问临界资源的代码

    exit section;//退出区，将正在访问临界区的标志清除

    remainder section;//剩余区，代码中的其他部分

}while(true)

临界区软件实现方法：

单标志法：设置公用整型变量turn，只允许下标为turn的进程访问->资源利用不充分

双标志法：设置数组flag，若flag[i]=false表示进程i未进入临界区->检查和修改操作不能一次进行

双标志法后检查：先设置自己为true再检查其他是否为false

Peterson’s：防止两个进程为进入临界区无限等待，设置turn同时检验另一个进程的状态（1、3的结合）

临界区硬件实现方法：

【优】适用于任意数目的进程，简单且容易验证，支持进程中有多个临界区【缺】进程进入临界区时耗费处理机时间，可能饥饿

中断屏蔽方法：禁止一切中断的发生，因为CPU只在发生中断时进行进程切换。关中断-临界区-开中断。【缺】将中断的权利交给用户，若一个进程关中断后不再开中断，系统可能因此终止，执行效率会降低，限制了处理机交替执行程序的能力

硬件指令方法：由硬件逻辑直接实现，不会被中断

TestAndSet指令：原子操作，读出指定标志后将其设为true

while TestAndSet(boolean\* lock){

    boolean old;old=\*lock;

    \*lock=true;return old;}

Swap指令：交换两个字的内容

key=true;

while(key) Swap(&lock,&key);

进程的临界区代码

lock=false;

进程的其他代码

信号量

wait(S)（P操作）：S.value- -表示进程请求一个资源，value为负数时表示资源被分配完

Signal(S)（V操作）：S.value++表示进程释放一个资源，若＋1后还是负数，则需要调用wakeup将队列中第一个等待进程唤醒

信号量实现同步

semaphore S=0;

P1(){

    ...

    V(S);//告诉P2语句已完成

}

P2(){

    ...

    P(S);//接收到P1已完成信号

    y;

}

信号量实现互斥

semaphore S=1;//可用资源为1

P(){

    P(S)

    ...进入临界区

    V(S)

}

Bounded-buffer problem

一个餐厅20人吃饭，只有一个门，每次只能通过一个人

semaphore empty=20; //最多可容纳人数

semaphore mutex=1;//一次可过一个人

cobegin

Pi(){

    ...

    P(empty);//可容纳人数-1

    P(mutex);//互斥使用门

    进门

    V(mutex);

    用餐

    P(mutex);//互斥使用门

    出门

    V(mutex);

    V(empty);//可容纳人数+1

}

生产者消费者

某寺庙有小和尚、老和尚若干，有一水缸，由小和尚提水入缸供老和尚饮用，水缸可容纳10桶水，水取自同一井中，每次只能容一个桶去睡，水桶总数3个，每次入缸取水仅为1桶水，且不可同时进行。

semaphore well=1;

semaphore vat=1;

semaphore empty=10;

semaphore full=0;

semaphore pail=3;

//老和尚

while(1){

    p(full);p(pail);p(vat);

    打水一桶;v(vat);v(empty);喝水

    v(pail);}

//小和尚

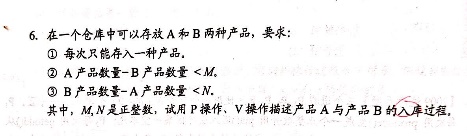
while(1){

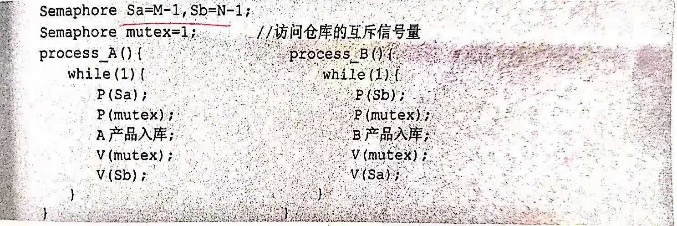
    p(empty);p(pail);p(well);

    从井中打水;v(well);p(vat);

    将水倒入水缸;v(vat);v(full);

    v(pail);}





死锁定义

多个进程因竞争资源而造成的一种互相等待的僵局，若无外力作用，这些进程都将无法向前推进

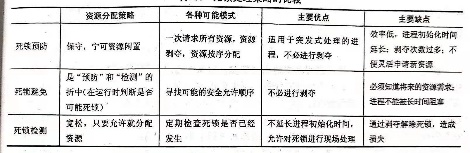
产生死锁的4个必要条件

互斥条件mutual exclusion：一段时间内某资源只能被一个进程占用

不剥夺条件no preemption：只能由获得资源的进程自己释放资源

请求并保持条件hold & wait：进程被阻塞时，不会释放已获得的资源

循环等待条件circular waiting：存在一种资源的循环等待链

死锁预防

破坏四个必要条件之一

1. 允许系统资源能被共享使用

2. 一个进程所拥有的资源会被暂时剥夺

3. 采用预先静态分配，在运行前进程一次性申请完全部所需资源

4. 采用顺序资源分配法

死锁的避免

系统安全状态：系统在进行资源分配前，应先计算此次分配的安全性【安全状态】系统能按某种推进顺序为每个进程分配资源

不安全状态=死锁状态+…

银行家算法：

Need=Max-Allocation

死锁定理

S为死锁的条件是当且仅当S状态的资源分配图不可被完全简化

主存连续分配

固定分区分配：每个分区只装入一道作业。

分区大小相等：用于用一台计算机去控制多个相同对象的场合，缺乏灵活性

分区大小不等：划分为多个较小分区、适量中等分区和少量大分区

动态分区分配：系统中分区的大小和数目可变。紧凑技术（compaction），系统不时对进程进行移动和整理，以克服外部碎片

FirstFit: 空闲分区以地址递增顺序连接【优】最好最快【缺】内存低地址区有很多小分区

BestFit: 空闲分区以容量递增顺序连接【缺】性能差，最多的外部碎片

WorstFit: 空闲分区以容量递减顺序连接【缺】导致没有可用的大内存块

NextFit: 由FF演变，从上次位置继续【缺】内存末尾存在很多小分区

分页基本概念

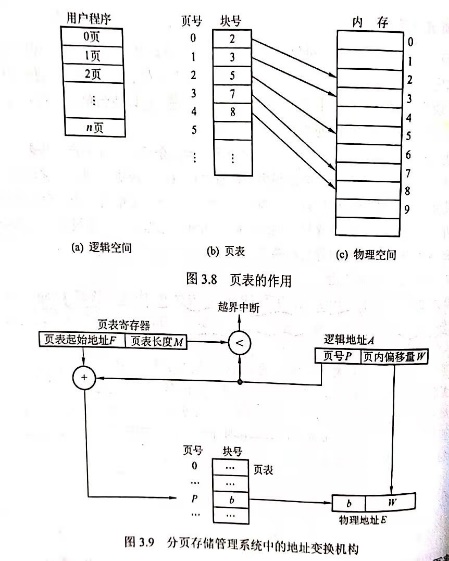
Page：进程中的页

Page Frame：内存中的块

Block：外存的分块

Page Table：页表

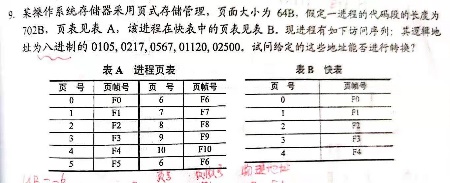
分页模型原理



逻辑地址转换

若有页面大小L为1KB，页号2对应的物理块b=8，逻辑地址A=2500的物理地址如下：

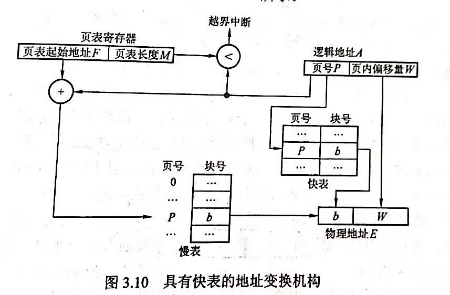
页号P=2500/1k=2，页内偏移量W=2500%1k=452，查到页号2对应的块号为8，物理地址E=8\*1024+452=8644



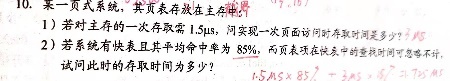
页面大小64B故页内位移6位，702/64=11故需要11个页面。0105逻辑页号1，由快表得页帧号为F1，页内位移为5故物理地址为(F1,5)。0567逻辑页号5，不在快表中，在内存页表可查到页帧号为F5，业内位移为55故物理地址(F5,55)。02500逻辑页号21，越界中断

TLB快表

用来存放当前访问的若干页表项，以加快地址变换过程

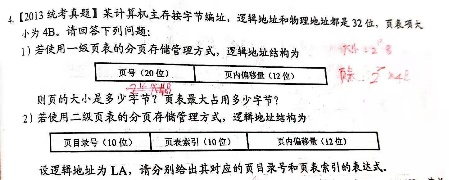


有效访问时间



1）3ms 2）1.5\*0.85+3\*0.15=1.725ms

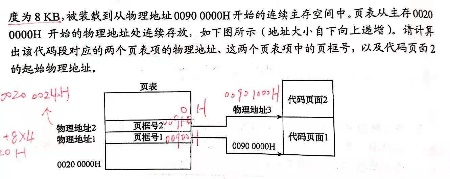
二级分页

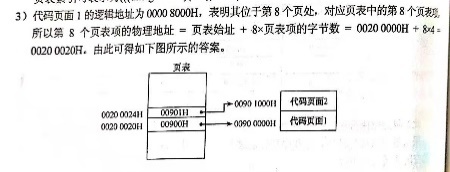


1）页大小为2^12B，页表项数为2^20，页表占2^20\*4B

2）页目录号(((unsigned int)(LA))>>22)&0x3FF,页索引号(((unsigned int)(LA))>>12)&0x3FF

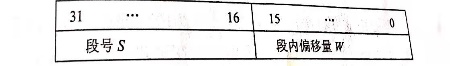
C:\Users\ljy28\AppData\Local\Temp\WeChat Files\676109982030285448.jpg





基本分段储存

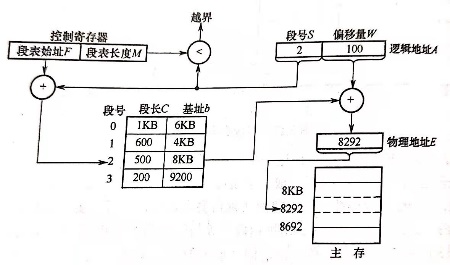
按照用户进程中的自然段划分逻辑空间，段内要求连续

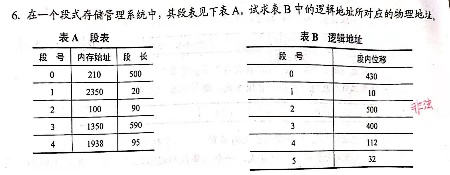


段表：

C:\Users\ljy28\AppData\Local\Temp\WeChat Files\29959941818128072.jpg

分段储存地址转换





（0,430）对应210+430=640。（1,10）对应2350+10=2450。（2,500）内存始值100，段长90<500，非法地址。（5,32）段表中不存在第5段，非法地址。

虚拟内存按需调页机制demand paging

只有程序执行需要时才载入页，那些从未访问的页不会调入到物理内存

缺页page-fault

当进程试图访问那些尚未调入到内存的页时，对标记为无效的访问会产生缺页

将缺页的进程阻塞，若内存中有空闲块，则分配一个块装入要调入的页，并修改页表，否则淘汰某页（若被淘汰页曾被修改，还需要将其写回外存）

输入内部中断，且一条指令中可以有多次缺页中断

COW copy on write

让两个进程共享同一份物理内存

linux中引入了“写时复制”技术，也就是只有进程空间的各段的内容要发生变化时，才会将父进程的内容复制一份给子进程。

内核只为新生成的子进程创建虚拟空间结构，它们来复制于父进程的虚拟内存结构，但是不为这些段分配物理内存，它们共享父进程的物理空间，当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间。

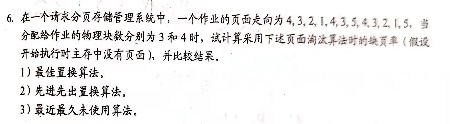
页面置换

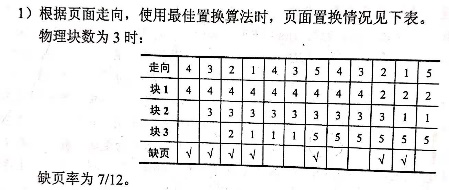
OPT：淘汰未来预测最长时间内不再被访问的页面，区分LRU的过去

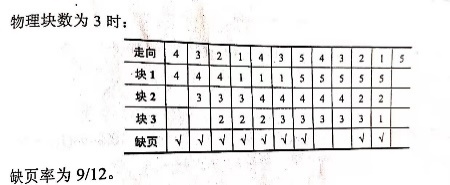
FIFO：淘汰最早进入内存的页面，基于队列实现，可能出现Belady异常

LRU：淘汰过去最长时间未访问的页面

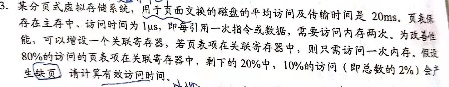
CLOCK：（使用位，修改位）：00>01>10>11，每轮扫描后将1置为0







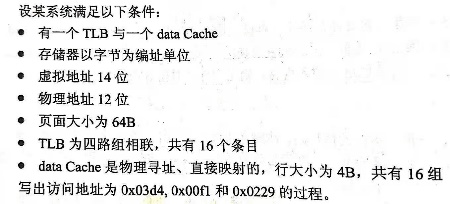




80%在TLB，访问耗时1μs

18%在内存，访问耗时1+1μs

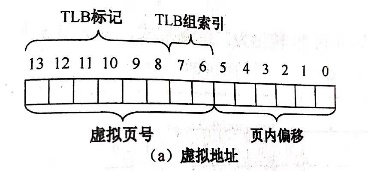
2%缺页中断，访问耗时1μs+1μs+20ms+1μs



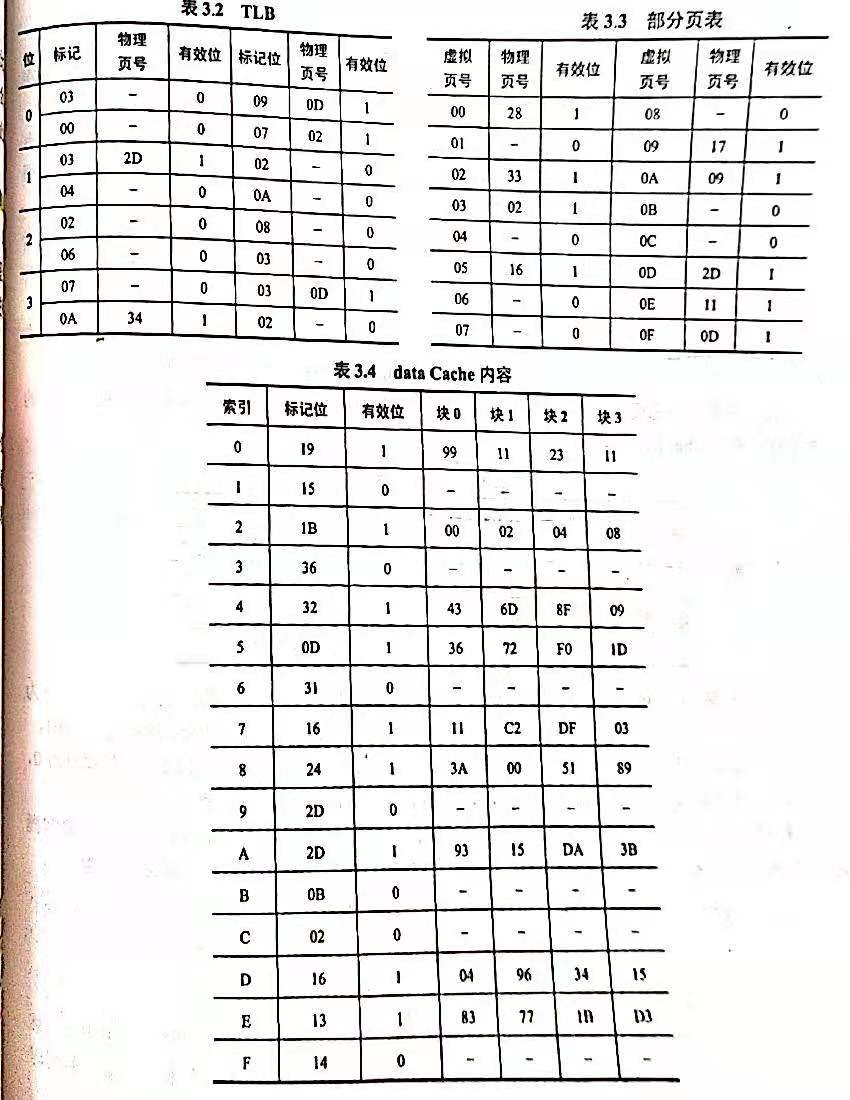
页面大小64B->页内偏移地址6位

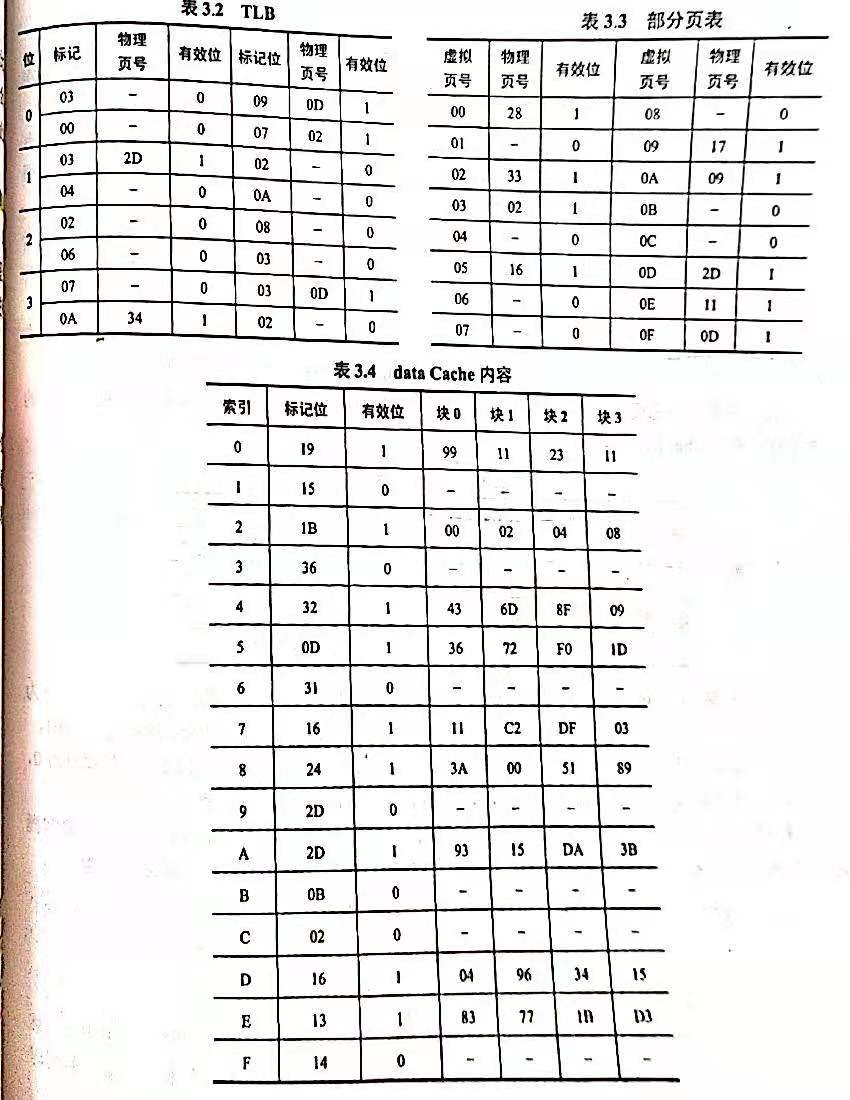
虚拟页号14-6=8位，TLB四路组相联，故虚拟页号中低2位为TLB组索引，高6位为TLB标记

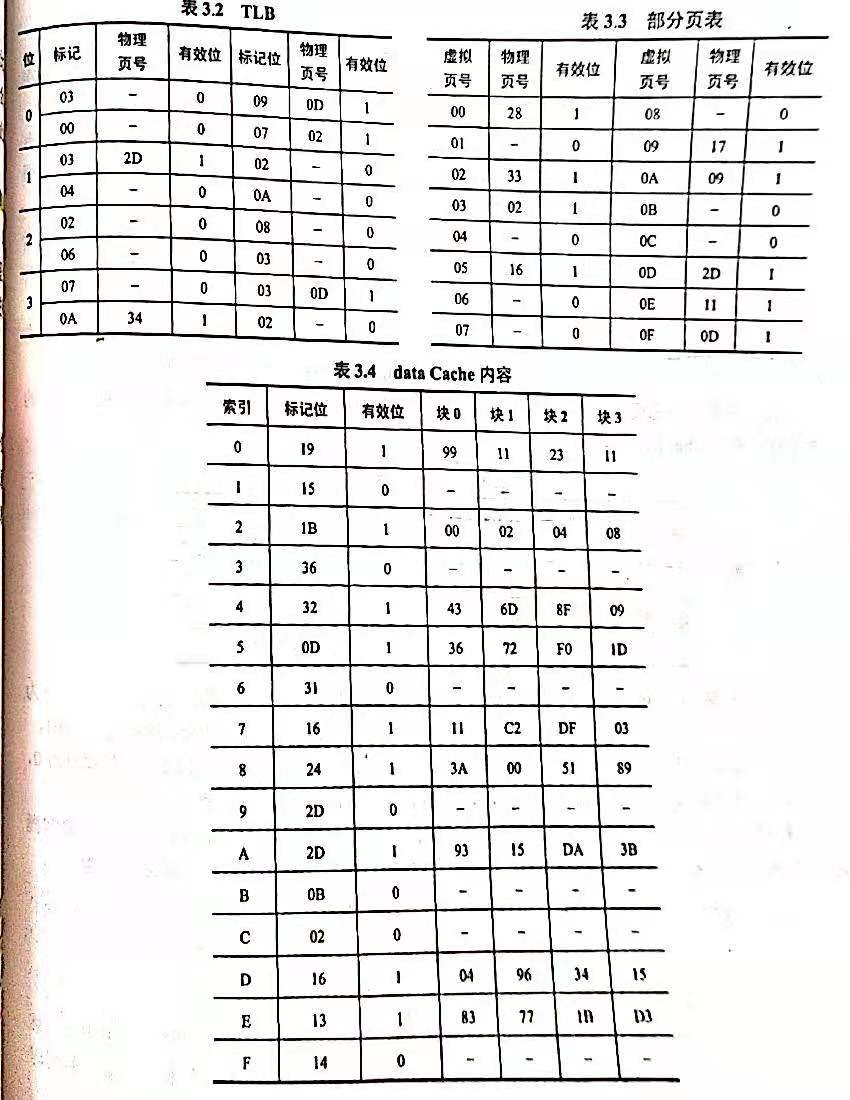
cache行大小4B，物理地址中低2位位快偏移，cache共16组故cache索引4位，剩下6位为cache标记











对于0x03d4：组索引3，TLB标记0x03，TLB中第三组恰有03项，故页面在主存中，对应物理地址0d，拼接页内地址010100，得物理地址0x354

对于0x00f1：组索引为3，TLB标记0x00，TLB中无该项，页表中虚拟页号为0x03，页表第三行有效位为1故在主存中，物理地址为000010，拼接页内地址110001，得物理地址0x0b1

对于0x0229：组索引0，TLB标记0x02，不在TLB中，虚拟页号0x08不在页表中，故不在主存中，产生缺页中断

文件结构基本属性

数据项+记录+文件

Name: only information kept in human-readable form. Identifier：unique tag (number) identifies file within file system. Type: needed for systems that support different types. Location: pointer to file location on device. Size: current file size. Protection: controls who can do reading, writing, executing. Time, date, and user identification: data for protection, security, and usage monitoring. Information about files are kept in the directory structure, which is maintained on the disk

文件信息被保存在目录结构中，目录结构保存在外存上

目录条目包含文件名称及其唯一的标识符用于定位其他属性的信息

文件基本操作

1. 创建文件create：（2个必要步骤）在文件系统中为文件找到空间，在目录中为新文件创造条目

2. 写文件write：执行一个系统调用，指明文件名称和要写入文件的内容。对于给定的文件名称，系统搜索目录以查找文件位置，系统必须为该文件维护一个写位置的指针，每当发生写操作，都要更新该指针

3. 读文件read：执行一个系统调用，指明文件名称和要读入文件快的内存位置。搜索目录已找到相关目录项，系统维护一个读位置的指针，每当发生读操作都要更新该指针。读或写都是用同一指针。

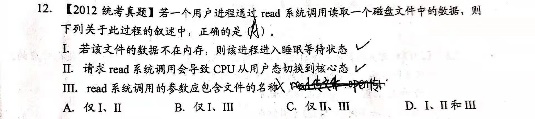
4. 文件重定位reposition within file：按某条件搜索目录，将当前文件位置设为给定值，不会读写文件

5. 删除文件delete：先从目录中找到要删除文件的目录项，使之成为空项，然后回收该文件所占用的存储空间

6. 截断文件truncate：允许文件所有属性不变，并删除文件，即将其长度设为0并是放弃其空间

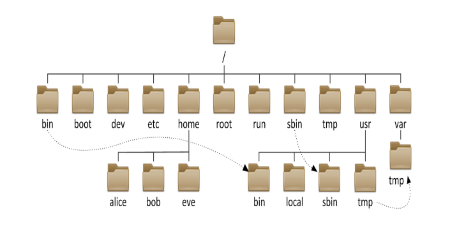
open打开文件/close关闭文件

使用系统调用 open将指明文件的属性（包括该文件在外存上的物理位置）从外存复制到内存打开文件表的一个表目中，并将该表目的编号（即索引）返回给用户。操作系统维护一个打开文件表（open-file table）。当用户需要一个文件操作时，可通过该表的一个索引指定文件，可忽略搜索环节。

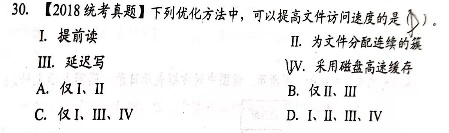


open传参文件名，read传参open返回的文件描述符

LINUX结构对文件&目录数据结构的表现

  
目录被组织成一个:单根倒置树结构，文件系统从根目录开始，用/来表示。文件名称区分大小写,以.开头的为隐藏文件,路径用/来进行分割

/bin : 提供用户使用的基本命令，存放二进制命令，不允许关联到独立分区，OS启动会用到里面的程序。/boot:用于存放引导文件,内核文件,引导加载器./sbin:管理类的基本命令，不能关联到独立分区，OS启动时会用到的程序（重要的命令通常处于bin，不重要的则安装在sbin）。/lib:存放系统在启动时依赖的基本共享库文件以及内核模块文件. 系统使用的函数库的目录也存放了大量的脚本库文件，程序在执行过程中，需要调用时会用到/lib64:存放64位系统上的辅助共享库文件./etc：系统配置文件存放的目录，该目录存放系统的大部分配置文件和子目录，不建议在此目录下存放可执行文件/home:普通用户主目录，当新建账户时，都会分配在此，建议单独分区，并分配额外空间用于存储数据。/root: 系统管理员root的宿主目录，系统第一个启动的分区为/，所以最好将/roo和/放置在一个分区下。/media:便携式移动设备挂载点目录/mnt:临时文件系统挂载点/dev:设备（device）文件目录，存放linux系统下的设备文件，访问该目录下某个文件，相当于访问某个设备，存放连接到计算机上的设备（终端、磁盘驱动器、光驱及网卡等）的对应文件 (b 随机访问，c 线性访问)/opt:第三方应用程序的安装位置/srv：服务启动之后需要访问的数据目录，存放系统上运行的服务用到的数据，如www服务需要访问的网页数据存放在/srv/www内。/tmp:存储临时文件， 任何人都可以访问,重要数据一定不要放在此目录下。/usr：应用程序存放目录，/usr/bin 存放保证系统拥有完整功能而提供的应用程序， /usr/share 存放共享数据，/usr/lib 存放不能直接运行的，却是许多程序运行所必需的一些函数库文件，\_/usr/local 存放软件升级包，第三方应用程序的安装位置，/usr/share/doc \_系统说明文件存放目录。/var：放置系统中经常要发生变化的文件，如日志文件。/var/log 日志目录及文件./var/tmp:保存系统两次重启之间产生的临时数据./proc: 用于输出内核与进程信息相关的虚拟文件系统，目录中的数据都在内存中，如系统核心，外部设备，网络状态，由于数据都存放于内存中，所以不占用磁盘空间/sys: 用于输出当前系统上硬件设备相关的虚拟：文件系统./selinux：存放selinux相关的信息安全策略等信息.



文件的逻辑结构

方便用户而设计

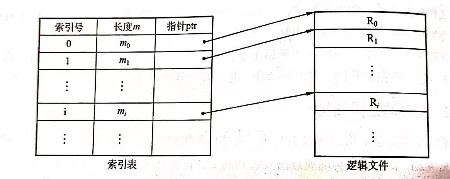
无结构文件（流式文件）：数据按顺序组织成记录并积累、保存，是有序相关信息项的集合，字节为单位，访问只能通过穷举搜索

有结构文件（记录式文件）：

1. 顺序文件: 文件中记录（通常是定长的）按顺序排列。【优】读写效率高，只有顺序文件才能记录在磁盘上【缺】对单条记录的增删改查比较困难

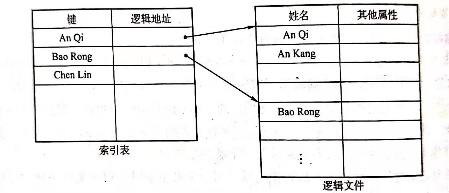
①串结构：顺序与关键字无关，由时间决定②顺序结构：记录按关键字顺序排列

2. 索引文件:



对于定长记录文件，可用i\*L获得第i条记录的相对地址。但对于不定长记录文件，需要用ΣLi+i才能获得第i条记录地址，可建立一张索引表（本身是定长记录的顺序文件）来加快访问

3. 索引顺序文件：



将顺序文件中的记录分为若干组，并为他们建立一张索引表，表中为每组第一条记录建立一个索引项，包含其关键字值和指向该记录的指针。

对于n条记录的顺序文件，平均需要查找N/2次。在索引顺序文件中，将N条记录分为组，索引表中有个表项，每组有条记录，平均查找次

4. 散列文件Hash File：给定记录的键值或通过散列函数转换的键值，直接决定记录的物理地址，没有顺序的特性【优】很高的存取速度【缺】可能引起冲突

文件控制块FCB

一个FCB就是一个文件目录项，每创建一个文件，系统就分配一个FCB并存放在文件目录中。

1. 基本信息：如文件名、物理位置、逻辑结构、物理结构

2. 存取控制信息：文件存取权限

3. 使用信息：文件建立时间、修改时间

索引文件中，索引表的每个表项中汉语相应记录的关键字和存放该记录的逻辑地址。

顺序文件进行检索时，首先从FCB中读出文件的第一个盘块号，而对索引文件进行检索时，应先从FCB中读出文件索引块开始的地址

对于一个具有三级索引的文件，存取一条记录通常需要访问4次磁盘

索引结点

采用文件名和文件描述信息分开的办法，文件描述信息单独形成一个成为索引结点的数据结构

文件保护的方法

口令，用户权限表，存取控制

相对于加密保护机制，访问控制机制的安全性较差。因为访问控制的级别和保护力度小，因此灵活性相对较高。访问控制由系统实现，否则系统本身的安全性无法得到保障

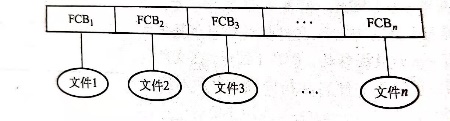
系统级安全管理包括注册和登录

文件的目录结构

在用户所需要的文件名和文件之间提供一种映射（按名存取）

Sigle- directory for all users:整个文件系统只建立一张目录表，每个文件占一个目录项

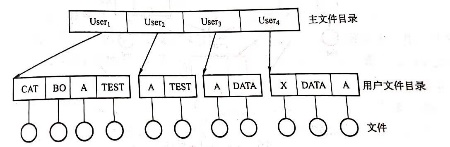
naming problem+ grouping problem



Separate directory for each user:将文件目录分为主文件目录(Master File Directory, MFD）和用户文件目录(User File Directory, UFD)

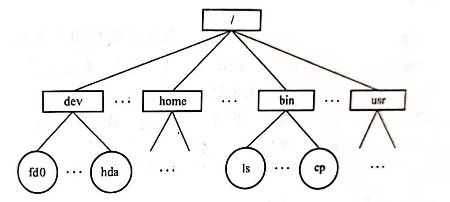
【缺】path name+ efficient searching+ no grouping capability

【优】can have the same file name for different user



tree structured directories:用户要访问某个文件时，用文件的路径名标识文件，由根目录出发到所找文件通路上所有目录名与数据文件名用/连接而成。进程对各文件的访问都是相对于当前目录进行的

efficient searching+ group capability+ current directory(working directory)



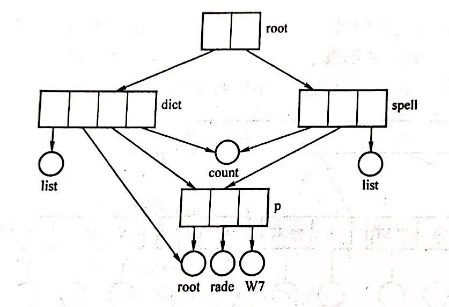
delete a file: rm<file-name>

creating a new subdirectory is done in current directory: mkdir<dir-name>

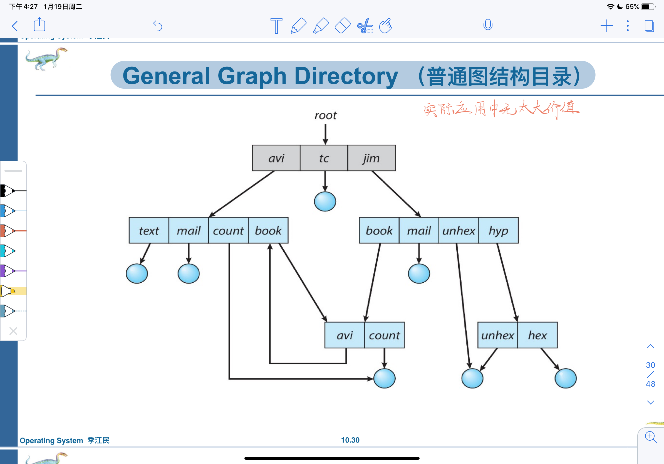
acyclic-Graph directories: 在树形目录结构中增加一些指向同一结点的有向边

逆向指针backpointers: using a daisy chain organization. we can delete all pointers variable size records a problem

表项保留计数Entry-hold-count solution: 记录结点的共享链条数量，只有计数器为0时才能真正删除该结点



general graph directory：allow only links to file not subdirectories; garbage collection; every time a new link is added use a cycle detection algorithm to determine whether it is ok



文件系统结构

application programs: The code that is making a file request

logical file system: 管理元数据（文件系统的所有结构数据，不包括实际数据），根据给定符号文件名来管理目录结构，通过FCB来维护文件结构

file-organization module：知道文件及其逻辑块和物理块，空闲空间管理器

basic file system：向合适的设备驱动程序发送一般命令就可对磁盘上的物理块进行读写

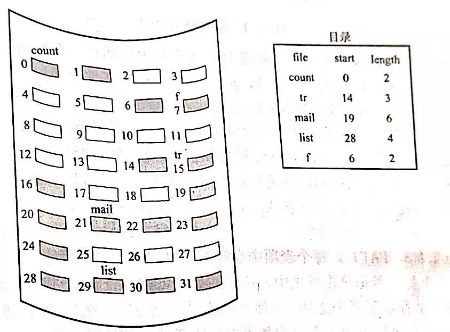
IO control：由设备驱动程序和中断处理程序组成，实现内存与磁盘之间的信息转移

文件分配方式

对应于文件的物理结构，指如何为文件分配磁盘块

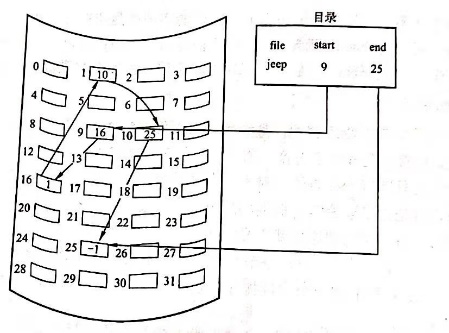
contiguous allocation连续分配：要求每个文件在磁盘上占有一组连续的块【缺】wasteful of space, files cannot grow【优】random access, simple

用第一块的磁盘地址和连续块的数量来定义。支持顺序访问和直接访问。反复增删文件后会产生外部碎片，因此知识和长度固定的文件

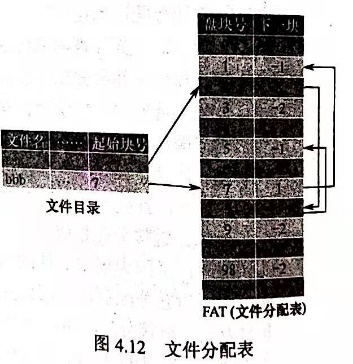


Linked Allocation链接分配：采用离散分配方式，消除外部碎片，提高磁盘空间利用率，根据文件需求动态分配盘块

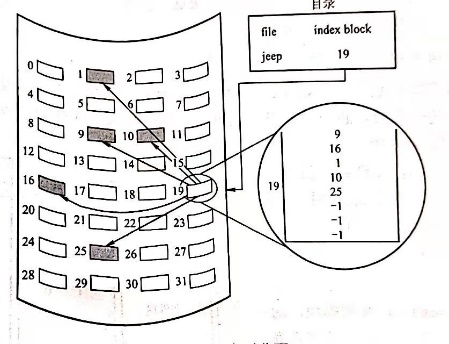
隐式链接：每个目录项都有一个指向文件首块的指针，该指针初始化为NULL以标识空文件，大小字段为0.【缺】无法直接访问盘块，只能通过指针顺序访问文件，且盘块指针会消耗一定的存储空间。稳定性差，运行过程中由于软件/硬件错误导致链表中的指针丢失或损坏，会导致文件数据的丢失。



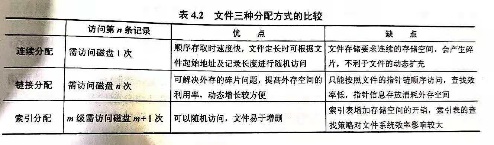
显式链接：把用于链接文件各物理块的指针从块末尾提取出来，显式地存放在内存地一张连接表中，在整个磁盘中仅设置一张文件分配表（file allocation table, FAT）。查找FAT的过程是在内存中进行了，可以提高检索速度、减少访问磁盘的次数。



Index Allocation索引分配：将每个文件的所有盘块号集中放在一起构成索引块表

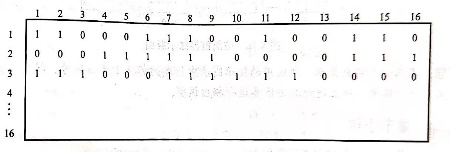


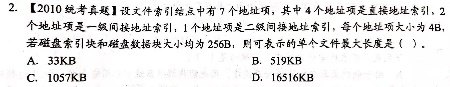
索引块的第i个条目指向文件的第i个块



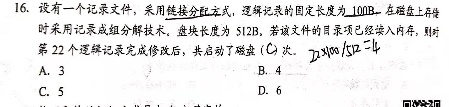
bitmap位示图法

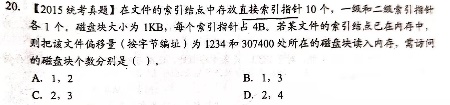
利用二进制的1位表示磁盘中一个盘块的使用情况，为0时表示对应的盘块空闲，为1时表示对应的盘块已分配。



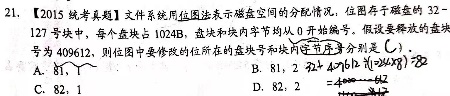


每个磁盘索引块有256/4=64个地址项，因此4个直接地址索引指向的数据块大小为4\*256B；2个一级间接索引包含的直接地址索引数为2\*(256/4)，即其指向的数据块大小为2\*(256/4)\*256B；1个二级间接索引所包含的直接地址索引数为(256/4)\*(256/4)，所指向的数据块大小为(256/4)\*(256/4)\* 256B。加起来是1057kB。



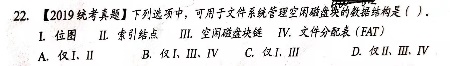


10个直接索引指针指向的数据块大小为10\*1KB=10KB。每个索引指针占4B，所以每个磁盘块可存放1KB/4B=256个索引指针。一级索引指针指向的数据块大小为256\*1KB=256KB，二级索引指针指向的数据块大小为256\*256\*1KB=2^16KB=64MB。偏移量1234时，1234B<256KB，由直接索引指针可得到其所在的磁盘块地址，文件的索引结点已在内存中，因此地址可直接得到，仅需1次访问磁盘。偏移量307400时，10+256KB<307400B<64MB，可知该偏移量的内容在二级索引指针所指向的某个磁盘块中，索引结点已在内存中，因此访盘2次得到文件所在的磁盘块地址，再访盘1次读出内容，共需3次访盘

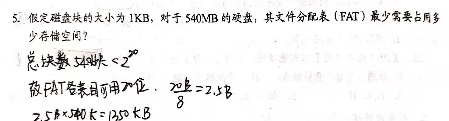


盘块号=起始块号+（盘块号/1024\*8）=32+409612/（1024\*8）=82

块内字节号=（盘块号%（1024\*8））/8=1



B



磁盘调度优化指标

寻找时间：活动头磁盘再读写信息前，将磁头移动到指定磁道所需要的时间。由跨越n条磁道的时间与启动磁臂的时间s组成：Ts=m\*n+s

延迟时间：磁头定位到某一磁道的扇区所需要的时间，设磁盘的旋转速度为r，则Tr=1/2r

传输时间：从磁盘读出或向磁盘写入数据所经历的时间，取决于每次读写的字节数b与磁盘的旋转速度Tt=b/rN，其中r是磁盘每秒的转数，N为一个磁道上的字节数

磁盘调度算法

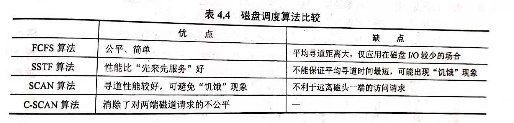
FCFS先来先服务：根据进程请求访问磁盘的先后顺序进行调度。具有公平性，若只有少量进程需要访问，且大部分请求都是访问族聚的文件扇区，则有望达到较好的性能。若有大量进程竞争使用磁盘，则这种算法在性能上接近随机调度。

SSTF最短寻找时间优先：选择离当前磁头所在磁道最近的磁道，使每次的寻找时间最短。不能保证平均寻找时间最小，但性能更好，会导致饥饿。

SCAN扫描算法：在SSTF基础上规定磁头运动方向，对最近扫描过的区域不公平，访问局部性较差。

C-SCAN循环扫描算法：再SCAN基础上，返回时直接快速移动到起始端

LOOK调度/C-LOOK调度：在SCAN/C-SCAN基础上，磁头移动只需要到达最远端的一个请求



交换分区swap-space

virtual memory uses disk space as an extension of main memory

swap-space can be carried out in two forms:

1. in the normal file system like Windows

2. in a separate disk partition like linux, unix

BSD allocates swap space when process starts; holds text segment(the program) and data segment.

kernel uses swap maps to track swap-space use

solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

RAID

定义：RAID (Redundant Array of Independent Disks)即独立磁盘冗余阵列，通常简称为磁盘阵。RAID 中主要有三个关键概念和技术：镜像Mirroring，数据条带Data Stripping，数据校验Data parity

Inexpensive -> Independent

RAID – multiple disk drives provides reliability via redundancy

Increases the mean time to failure

Frequently combined with NVRAM to improve write performance

RAID is arranged into six different levels

Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

Disk striping(条带化)uses a group of disks as one storage unit

RAID schemas improve performance and improve the reliability of the storage system by storing redundant data

\* Mirror(镜像)or shadowing(RAID 1) keeps duplicate of each disk

\* Striped mirrors(RAID 1+0) or mirrored stripes(RAID 0+1) provides high performance and high reliability

\* Block interleaved parity(RAID 4,5,6)uses much less redundancy

RAID within a storage array can still fail if the array fails, so automatic replication of the data between arrays is common

Frequently, a small number of hot-spare disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

用处：提供比单个硬盘更高的储存性能和提供数据备份技术

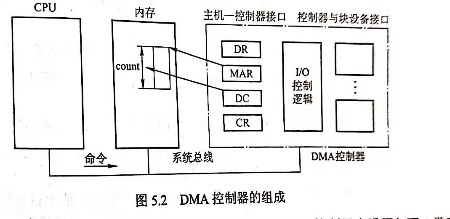
DMA方式

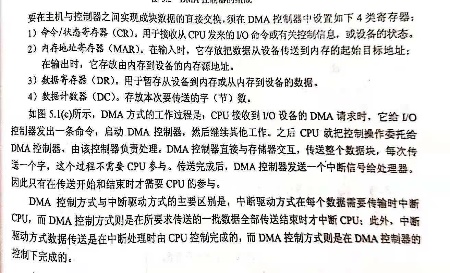
在IO设备和内存之间开辟直接的数据交换通路，彻底解放CPU

1. 基本单位是数据块

2. 所传送的数据，直接从设备送入内存或反之

3. 仅在传送一个或多个数据块的开始和结束时，才需CPU干预，整块数据的传送是在DMA控制器的控制下完成的





LINUX指令

2．命令功能：用来压缩和解压文件。tar本身不具有压缩功能。他是调用压缩功能实现的

3．命令参数：必要参数有如下：

-A 新增压缩文件到已存在的压缩 -B 设置区块大小 -c 建立新的压缩文件-d 记录文件的差别 -r 添加文件到已经压缩的文件 -u 添加改变了和现有的文件到已经存在的压缩文件 -x 从压缩的文件中提取文件 -t 显示压缩文件的内容 -z 支持gzip解压文件 -j 支持bzip2解压文件 -Z 支持compress解压文件 -v 显示操作过程 -l 文件系统边界设置 -k 保留原有文件不覆盖 -m 保留文件不被覆盖 -W 确认压缩文件的正确性

可选参数如下：

-b 设置区块数目 -C 切换到指定目录

-f 指定压缩文件 --help 显示帮助信息

--version 显示版本信息

4．常见解压/压缩命令

tar

解包：tar xvf FileName.tar

打包：tar cvf FileName.tar DirName

（注：tar是打包，不是压缩！）

.gz

解压1：gunzip FileName.gz

解压2：gzip -d FileName.gz

压缩：gzip FileName

.tar.gz 和 .tgz

解压：tar zxvf FileName.tar.gz

压缩：tar zcvf FileName.tar.gz DirName

1. man或info 获得每个Linux命令的帮助手册， 或命令名 –help.

2.w或who 显示当前用户名字(包括时间)

3.用户名whoami ;操作系统名uname ;计算机系统域名uname -n ;计算机系统CPU uname -p

用户ID id -u ;组ID和组名字 id -g ; id -n ;同一组中的其他用户名 groups ;系统管理员给主目录设置的权限 ls -d ;

4.详细时间date ;日历cal 7 2010或cal 7

cal -j 12变成天数

5.当前目录路径pwd ;已运行时间uptime

6.修改文件或者目录的时间属性 touch ;复制文件或目录 cp ;为文件或目录改名、或将文件 mv ;删除一个文件或者目录 rm ;开启一个新的虚拟终端以开始一个程序 open ;从文件描述符读取内容 read ;寄一封邮件给其他使用者 write ;关闭一个文件描述符 close ;创建一个管道 pipe ;创建一个沟通终点 socket ;创建先进先出(FIFO)管道 mkfifo ;执行一个壳命令 system ;输出或格式化数据 printf

7.获得主目录 cd ~或echo $HOME$或echo ${HOME} ;显示以t字母开头的头文件名 ls -l t\*.h

8.主目录下查找隐含文件 ls -a|grep ‘^\.’

9.显示inode号 ls -I ;显示所有信息 ls -l

10.查看文件类型 stat 文件名 ;文件内容类型 file 文件名 ;创建大文件man bash > 文件名 ;创建中文件 man cat > 文件名 ; 显示文件的开始12行内容 head -12 文件名 ; 显示文件的最后5行内容 tail -n 5 文件名 ; 显示文件从第6行开始到结束全部行 tail –n +6 文件名;

显示文件字节数、字数和行数 wc file1 file2 ;给出字节数; 搜索目录，显示创建时间在file之后的文件及其路径 find -newer file ; 搜索文件students的文本，并把首字母在A-H之间的行打印出来 grep ‘/^[A-H]/’ students ; 搜索文件students的文本，并把首字母是A和H的行打印出来grep ‘/^[A,H]/’ students

11.复制后时间不变 cp -p 文件1 文件2

12.用长列表格式显示文件或目录 ls -al file

13.修改权限 chmod 755 file

14.显示当前掩码 umask ;设置掩码 umask 077

15.编译c代码 gcc -o 可执行文件 file.c

16.删除目录下所有内容 rm -rf 目录

17.创建硬链接: ln file newfile.hard ;创建软链接: ln -s file newfile.soft

18.使用的Linux系统中有多少个进程在运行 ps -Al | wc -l ;(wc -l扣除域名的一行)查看进程PID ps -Al ;有多少个sh进程ps | grep sh |wc -l

19.使得date 、uname –a 、who 和ps并发执行date & uname -a & who & ps ;当只有前

面一个命令执行成功后，才能执行后面一个命令date && uname -a && who && ps

20.普通文件符号是 f; 目录文件符号是 d; 符号链接文件符号是l.查找文件名使用的是find，统计行数使用的是wc –l .eg. find -type f | wc -l

21.计算命令ls -l的输出中的字符数、单词数和行数ls -l | wc

22.搜索并显示你主目录下foobar文件的绝对路径，错误信息重定向到/dev/null中 realpath -P foorbarr 2>/dev/null ;搜索你主目录下foobar文件，保存它的绝对路径到foobar.path文件中，错误信息写到/dev/null中 realpath -P foorbar >foorbar.path 2>/dev/null

Lab2

1.更新软件源apt-get update ; 安装需要的环境apt-get install kernel-package libncurses5-dev libssl-dev

2.解开压缩内核xz -d linux-4.6.tar.xz ; 解压后得到linux-4.6.tar. 执行tar xvf linux-4.6.tar

3.打内核补丁xz -d patch-4.6.xz | patch -p1

4.清除目录下所有配置文件和先前生成核心时产生的.o文件make mrproper

5.编译内核make bzImage -jN ; 对选择的模块进行编译make modules -jN

6.安装模块make modules\_install; 安装内核make install

7.应用grub配置启动文件

mkinitramfs 4.6.0 -o /boot/initrd.img-4.6.0

update-grub2

8.内核模块装入insmod helloworld.ko ;输入dmesg查看输出;使用lsmod命令查看;卸载这个模块rmmod helloworld

9.c程序执行: gcc user.c 生成a.out文件, 执行 ./a.out 运行程序

Lab4

#mount –t myext2 –o loop ./fs.new /mnt

#cd /mnt

#mknod myfifo p

mknod: `myfifo': Operation not permitted

第一行命令：将fs.new mount到/mnt目录下。

第二行命令：进入/mnt目录，也就是进入fs.new这个myext2文件系统。

第三行命令：执行创建一个名为myfifo的命名管道的命令。

第四、五行是执行结果：第四行是我们添加的myext2\_mknod函数的printk的结果；第五行是返回错误号EPERM结果给shell，shell捕捉到这个错误后打出的出错信息。

#!/bin/bash

/sbin/losetup -d /dev/loop2

/sbin/losetup /dev/loop2 $1

/sbin/mkfs.ext2 /dev/loop2

dd if=/dev/loop2 of=./tmpfs bs=1k count=2

.angeMN $1 ./tmpfs

dd if=./fs.new of=/dev/loop2

/sbin/losetup -d /dev/loop2

rm -f ./tmpfs

第一行 表明是shell程序。第三行 如果有程序用了/dev/loop2了，就将它释放。第四行 用losetup将第一个参数代表的文件装到/dev/loop2上第五行 用mkfs.ext2格式化/dev/loop2。也就是用ext2文件系统格式格式化我们的文件系统。第六行 将文件系统的头2K字节的内容取出来，复制到tmpfs文件里面。第七行 调用程序changeMN读取tmpfs，复制到fs.new，并且将fs.new的magic number改成0x6666第八行 再将2K字节的内容写回去。第九行 把我们的文件系统从loop2中卸下来。第十行 将临时文件删除。

错题

· 库函数不是操作系统提供给应用程序的接口：库函数是高级语言中提供的与系统调用对应的函数（也有些库函数语系统调用无关），属于用户程序，是系统调用的上层

· 操作系统与用户通信接口通常包括：shell（命令解析器），命令解释器（命令接口），广义指令（系统调用）不包括：缓存管理指令（系统缓存全部由操作系统管理，对用户是透明的）

·多道程序设计的设计原则：制约性、间断性、共享性

· 计算机开机后，操作系统最终会被加载到RAM

· 提高单机资源利用率的关键技术是多道程序设计技术

· 实时操作系统必须在被控制对象规定时间内处理来自外部的事件

· 实时系统的进程调度，通常采用抢占式的优先级高者优先

· 分时系统追求的目标是比较快速响应用户

·用户程序在用户态下要使用特权指令引起的中断属于访管中断

·广义指令（系统调用）只能在核心态下执行

·操作系统程序可以执行特权指令

·子程序调用只需保存程序断点，即该指令的下一条指令的地址

·中断调用子程序不仅需要保存断点（PC），还需要保存程序状态字寄存器（PSW）的内容

·处理外部中断时，操作系统应该保存通用寄存器的内容（PSWR），程序计数器的内容（PC）由中断隐指令在硬件中自动完成

·执行系统调用的过程：传递系统调用参数->执行陷入trap指令->执行相应的服务程序->返回用户态

· 线程包含CPU现场，可以独立执行程序

· 线程没有自己独立的地址空间，它共享其所属进程的空间

· 并发进程的封闭性：执行结果与速度无关

· 线程是一种特殊的进程

· 一个进程是程序在一个数据集上的一次运行过程· 同一程序经过多次创建，运行在不同的数据集上，形成了不同的进程· 系统线程被不同的进程调用，还是相同的线程

· 降低进程优先级的合理时机是进程时间片用完

· 信号量分为互斥量和资源量· 互斥量初值为1，表示临界区只允许一个进程进入，若小于0则绝对值表示临界区外等待进入的进程数量· 资源量初值为0，小于0表示所有资源已全部用完，等待的进程数就是其绝对值

· 临界资源与共享资源的区别在于，在一段时间内能否允许被多个进程并发使用

· PV操作是一种低级的进程通信原语

· 共享程序段可能同时被多个进程使用，所以必须可重入编码。

一个文件系统中有一个20MB大文件和一个20KB小文件，当分别采用连续、链接、链接索引、二级索引和LINUX分配方案时，每块大小为4096B，每块地址用4B表示。问：

（1）各文件系统管理的最大文件是多少？

连续分配：理论上是不受限制，可大到整个磁盘文件区。

隐式链接：由于块的地址为4字节，所以能表示的最多块数为 232=4G，而每个盘块中存放文件大小为4092字节。链接分配可管理的最大文件为：4G×4092B=16368GB

链接索引：由于块的地址为4字节，所以最多的链接索引块数为 232=4G，而每个索引块有1023个文件块地址的指针，盘块大小为4KB。假设最多有n个索引块，则1023×n+n=232，算出n=222，链接索引分配可管理的最大文件为：4M10234KB=16368GB

二级索引：由于盘块大小为4KB，每个地址用4B表示，一个盘块可存1K个索引表目。二级索引可管理的最大文件容量为4KB×1K×1K＝4GB。

LINUX混合分配：LINUX的直接地址指针有12个，还有一个一级索引，一个二级索引，一个三级索引。因此可管理的最大文件为48KB＋4MB+4GB＋4TB。

（2）每种方案对大、小两文件各需要多少专用块来记录文件的物理地址(说明各块的用途)？

连续分配：对大小两个文件都只需在文件控制块FCB中设二项，一是首块物理块块号，另一是文件总块数，不需专用块来记录文件的物理地址。

隐式链接：对大小两个文件都只需在文件控制块FCB中设二项，一是首块物理块块号，另一是末块物理块块号；同时在文件的每个物理块中设置存放下一个块号的指针。

一级索引：对20KB小文件只有5个物理块大小，所以只需一块专用物理块来作索引块，用来保存文件的各个物理块地址。对于20MB大文件共有5K个物理块，由于链接索引的每个索引块只能保存（1K－1）个文件物理块地址（另有一个表目存放下一个索引块指针），所以它需要6块专用物理块来作链接索引块，用于保存文件各个的物理地址。

二级索引：对大小文件都固定要用二级索引，对20KB小文件，用一个物理块作第一级索引，用另一块作二级索引，共用二块专用物理块作索引块，对于20MB大文件，用一块作第一级索引，用5块作第二级索引，共用六块专用物理块作索引块。

LINUX的混合分配：对20KB小文件只需在文件控制块FCB的i\_addr[15]中使用前5个表目存放文件的物理块号，不需专用物理块。对20MB大文件，FCB的i\_addr[15]中使用前12个表目存放大文件前12块物理块块号（48K），用一级索引块一块保存大文件接着的1K块块号（4M），剩下还有不到16M，还要用二级索引存大文件以后的块号，二级索引使用第一级索引1块，第二级索引4块（因为4KB×1K×4=16 M）。总共也需要6块专用物理块来存放文件物理地址。

（3）如需要读大文件前面第5.5KB的信息和后面第（16M＋5.5KB）的信息，则每个方案各需要多少次盘I/O操作?

连续分配：为读大文件前面和后面信息都需先计算信息在文件中相对块数，前面信息相对逻辑块号为5.5K／4K=1（从0开始编号），后面信息相对逻辑块号为（16M＋5.5K）/4K=4097。再计算物理块号＝文件首块号＋相对逻辑块号，最后化一次盘I/O操作读出该块信息。

链接分配：为读大文件前面5.5KB的信息，只需先读一次文件头块得到信息所在块的块号，再读一次第1号逻辑块得到所需信息，共2次。而读大文件16MB＋5.5KB处的信息，逻辑块号为（16M＋5.5K）/4092=4107，要先把该信息所在块前面块顺序读出，共化费4107次盘I／O操作，才能得到信息所在块的块号，最后化一次I/O操作读出该块信息。所以总共需要4108次盘I／O才能读取（16MB+5.5KB）处信息。

链接索引：为读大文件前面5.5KB处的信息，只需先读一次第一个索引块得到信息所在块的块号，再读一次第1号逻辑块得到所需信息，共化费2次盘I／O操作。为读大文件后面16MB+5.5KB处的信息，(16MB+5.5KB)/(4KB×1023)=4,需要先化5次盘I／O操作依次读出各索引块，才能得到信息所在块的块号，再化一次盘I/O操作读出该块信息。共化费6次盘I／O操作。

二级索引：为读大文件前面和后面信息的操作相同，首先进行一次盘I／O读第一级索引块，然后根据它的相对逻辑块号计算应该读第二级索引的那块，第一级索引块表目号=相对逻辑块号／1K，对文件前面信息1／1K＝0，对文件后面信息4097／1K＝4，第二次根据第一级索引块的相应表目内容又化一次盘I／O读第二级索引块，得到信息所在块块号，再化一次盘I／O读出信息所在盘块，这样读取大文件前面或后面处信息都只需要3次盘I／O操作。

LINUX混合分配：为读大文件前面5.5KB处信息，先根据它的相对逻辑块号，在内存文件控制块FCB的i\_addr第二个表目中读取信息所在块块号，而只化费一次盘I／O操作即可读出该块信息。为读大文件后在（16MB＋5.5KB）信息，先根据它的相对逻辑块号判断要读的信息是在二级索引管理范围内，先根据i\_addr内容化一次盘I／O操作读出第一级索引块，再计算信息所在块的索引块号在第一级索引块的表目号为（4097-12-1024）／1024＝2，根据第一级索引块第3个表目内容再化费一次盘I／O操作，读出第二级索引块，就可以得到信息所在块块号，最后化一次盘I／O读出信息所在盘块，这样总共需要3次盘I／O操作才能读出文件后面的信息。