中国科学院大学
University of Chinese Academy of Sciences

操作系统课程 B0911010Y-01

文件系统实现

中国科学院大学计算机与控制学院
中国科学院计算技术研究所

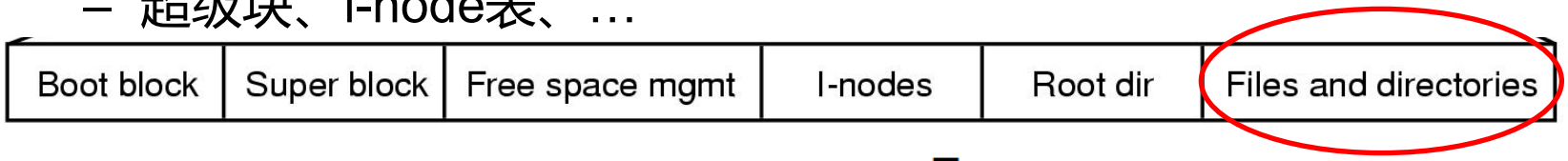
2021-12-8

内容提要

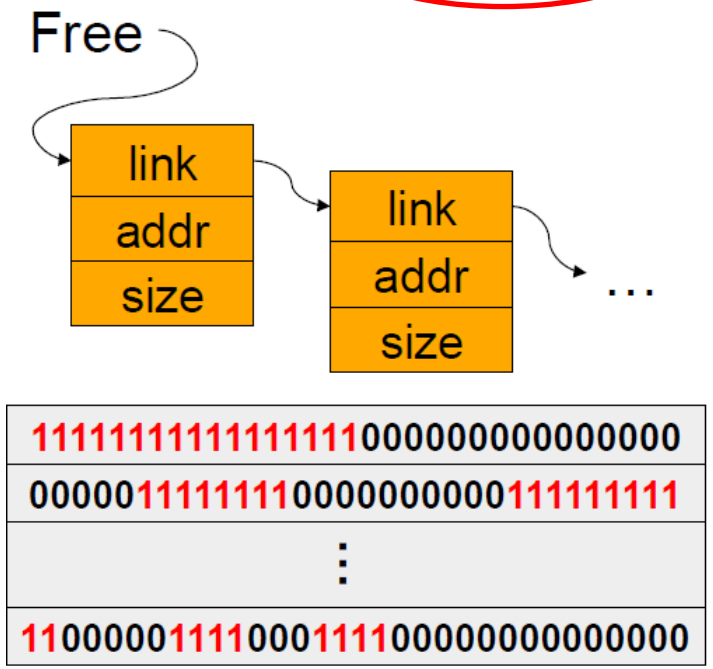
- 磁盘空间管理
- 文件块索引结构
- 目录项的组织结构
- 文件缓存
- FFS（UNIX文件系统）

磁盘空间管理

- 磁盘布局
  - 超级块、i-node表、…



- 空闲数据块
  - 空闲块链表
  - 空闲块位图（bit map）
  - …



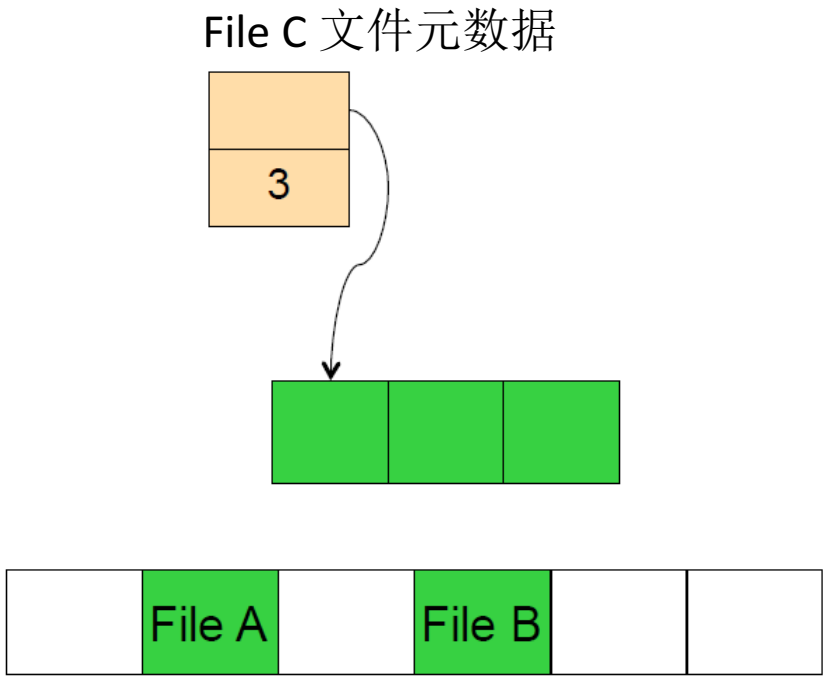
- 已用数据块
  - 位于i-node中的文件块索引：定位每个文件块在磁盘上的位置

内容提要

- 磁盘空间管理
- 文件块索引结构
- 目录项的组织结构
- 文件缓存
- FFS（UNIX文件系统）

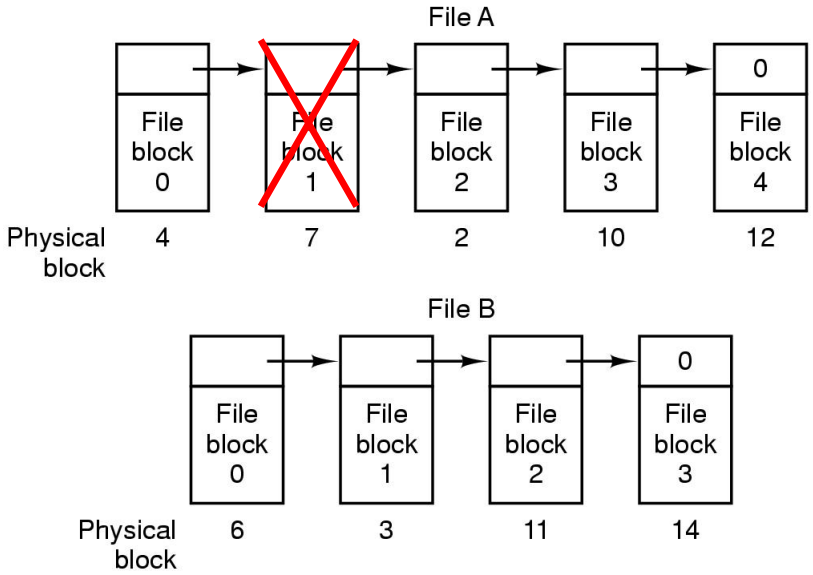
连续分配

- 分配连续的磁盘块给文件
  - 文件粒度分配
  - 位图：找到N个连续的“0”
  - 链表：找到size>=N的区域
- 文件元数据
  - 记录第一个块的地址
  - 块的个数N
- 优点
  - 顺序访问性能高
  - 随机访问时定位数据块也容易
- 缺点
  - 不知道文件最终多大，无论创建时，还是写数据块时
  - 文件难以“变大”
  - 外部碎片化：如果文件C需要3个块怎么办？



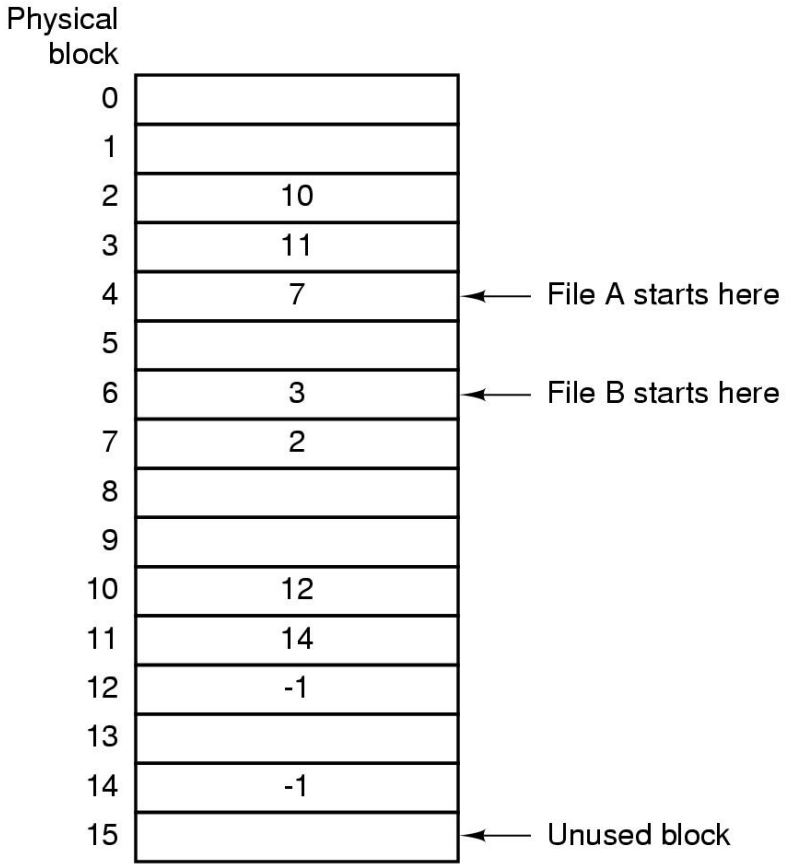
文件块索引：链表结构

- 分配不连续的磁盘块给文件
  - 块粒度分配
- 文件元数据
  - 记录第一块的地址
  - 每个块指向下一块的地址
  - 最后一块指向NULL
- 优点
  - 无外部碎片，而且
    文件“变大”很容易
  - 空闲空间链表：与文件块类似
- 缺点
  - 随机访问性能极差：定位数据块需按指针顺序遍历链表
  - 可靠性差：一个块坏掉意味着其余的数据全部“丢失”
  - 块内有效数据的大小不一定再是2的幂



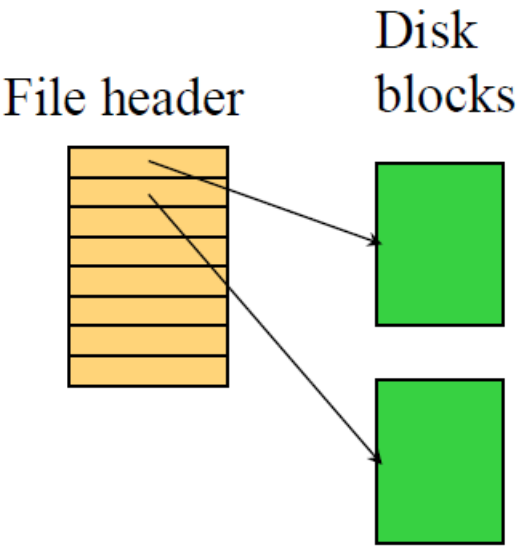
文件块索引：文件分配表（FAT）

- 一张有N个项的表，假设磁盘有N块
  - 每个磁盘块有一个表项：要么为空，要么为该文件下一块的地址
  - 位于磁盘分区的头部
- 文件元数据
  - 记录第一块的地址：链表头指针
  - 每个磁盘块全部存数据，无指针
- 优点
  - 简单
  - 文件块大小为2的幂
- 缺点
  - 随机访问性能不好
    - 定位数据需要查找FAT表
  - 浪费空间
    - 额外的空间存储FAT表



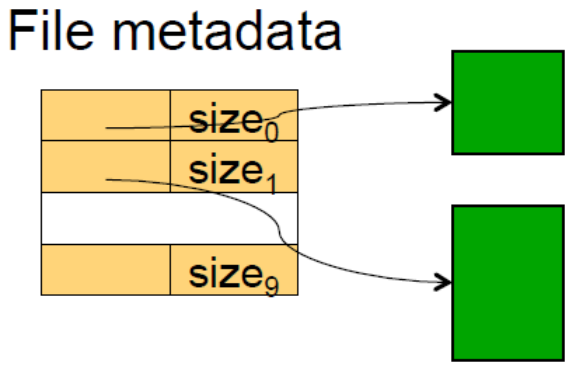
文件块索引：单级索引

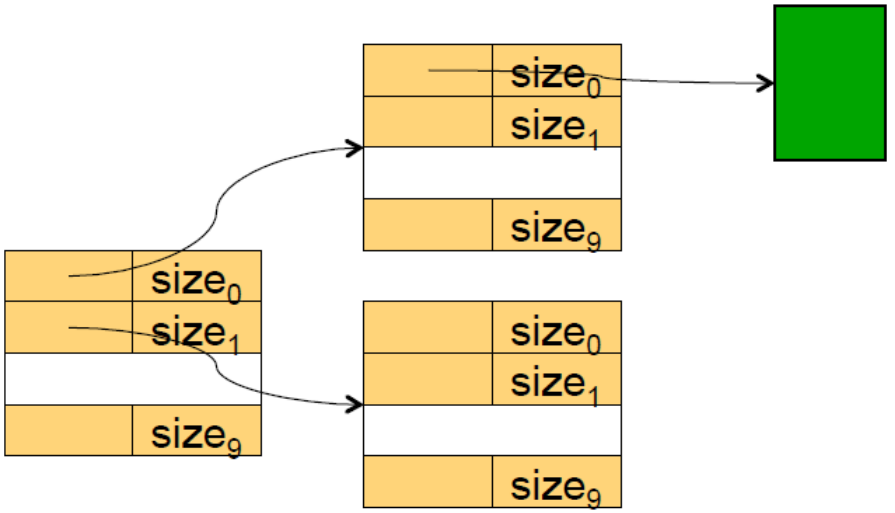
- 文件元数据
  - 用户定义文件长度上限max size
  - file header ：一个指针数组
    指向每个块的磁盘地址
- 优点
  - 文件在限制内可变大
  - 随机访问性能高：数据块直接定位
- 缺点
  - 不灵活，文件长度难以事先知道
    - 如果max size不够大就难办了
    - 如果max size过大，浪费空间



文件块索引：两级索引（Cray-1 DEMOS ）

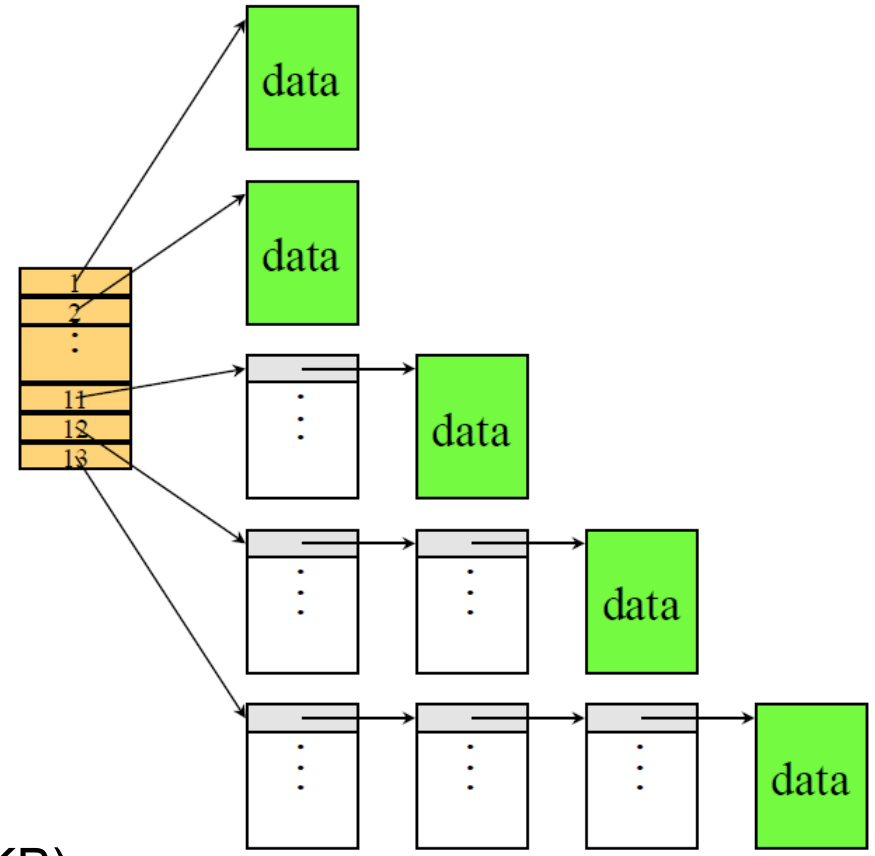
- 思路：
  - 采用连续分配，允许不连续
  - 不定长分配
- 文件元数据
  - 小文件有10个指针
    指向10个可变长度段（base, size）
  - 大文件有10个间接指针，每个指向可变长度的间址块
- 优点
  - 支持文件变大（最大10GB）
- 缺点
  - 外部碎片





文件块索引：多级索引（UNIX）

- 块粒度分配
- 文件元数据：13个指针
  - 10个直接指针
  - 第11个指针：一级间接指针
  - 第12个指针：二级间接指针
  - 第13个指针：三级间接指针
- 优点
  - 小文件访问方便
  - 支持文件变大
- 缺点
  - 文件有上限，16GB多 (每个块1KB)
  - 大量寻道
- 如何找到文件的第23块、第5块、第340块？



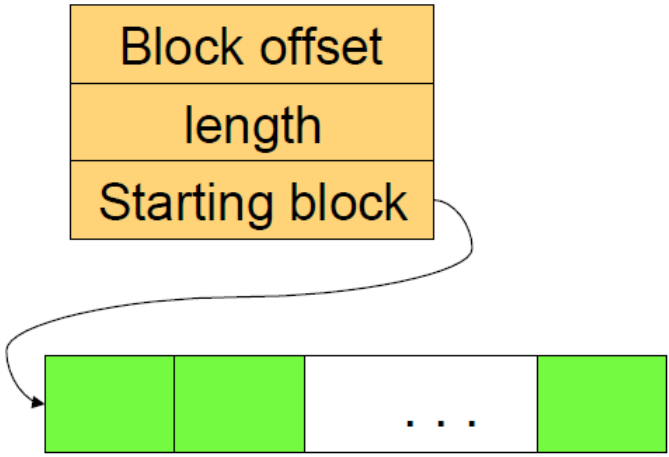
最初UNIX的i-node

- mode: 文件类型，访问权限，
- link count: 指向该i-node的目录项个数
- uid：文件所有者的uid
- gid：文件所有者的gid
- file size
- 时间戳：创建时间ctime，修改时间mtime，访问时间atime

- 10个指向数据块的指针
- 一级间接指针
- 二级间接指针
- 三级间接指针

文件块索引： Extents

- Extent是若干个连续磁盘块（长度不固定）
  - 同一extent中的所有块：要么都是空闲块，要么都属于某个文件
  - extent：<starting block, length>
- XFS提出的方法
  - 无论文件块还是空闲块都采用extents来组织
    - 块大小为8KB
    - Extent的大小 <= 2M个块
  - 文件块索引： extent的映射关系
    - <文件块号, 块数 > → <磁盘块号, 块数>
    - 采用B+树
  - 文件元数据
    - 记录B+树的根节点地址



A. Sweeney, et al., **Scalability in the XFS File System**, ATEC'96 (ATC'96)
https://www.usenix.org/legacy/publications/library/proceedings/sd96/sweeney.html

内容提要

- 磁盘空间管理
- 文件块索引结构
- 目录项的组织结构
- 文件缓存
- FFS（UNIX文件系统）

目录访问

- 路径解析
  - 在目录里查找指定目录项：文件名
- 修改目录
  - 创建/删除目录、创建/删除文件、硬链接、符号链接、 rename
  - 在目录里添加/删除目录项
- Readdir
  - 扫描目录内容

目录项采用什么组织结构？

ino	name
1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

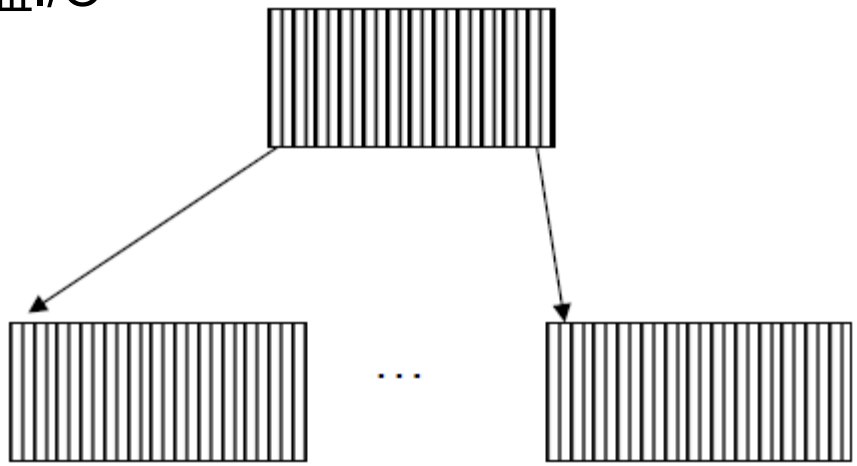
线性表

- 原理
  - <文件名，ino>线性存储
    - 每一项不定长：<ino, 名字长度, 下一项起始偏移，名字>
  - 创建文件
    - 先查看是否有重名文件
    - 如果没有，在表末增加一个entry：<ino, newfile>
  - 删除文件
    - 用文件名查找
    - 删除匹配的entry
    - 紧缩：将之后的entry都向前移动
- 优点
  - 空间利用率高
- 缺点
  - 大目录性能差：线性查找，目录项数据从磁盘读取时，磁盘I/O多
  - 删除时紧缩很耗时

<1, .><1, ..> <4, bin> <7, dev><14, lib> <9, etc> <6, usr> <8, tmp> ……

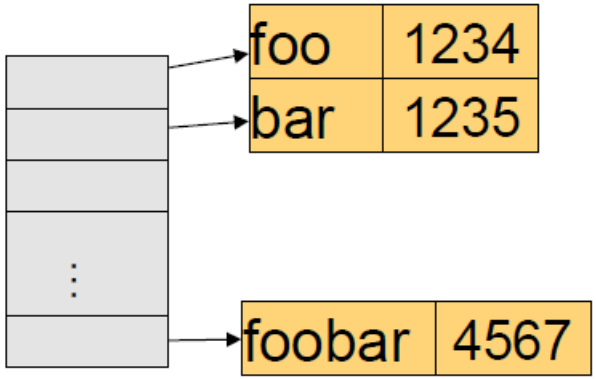
使用B树索引

- 原理
  - 在磁盘上使用B树索引目录的数据块
  - 用B树来存储<文件名，ino>，以文件名排序(字典序)
  - 创建/删除/查找：在B树中进行
- 优点
  - 大目录性能高：B树查找减少磁盘I/O
- 缺点
  - 小目录不高效
  - 占用更多空间
  - 实现复杂



使用哈希表索引

- 原理
  - 在VFS中使用哈希表索引目录项
  - 用哈希表将文件名映射到ino
    - hash_func(filename) → hval → bucket
    - linear lookup “filename” in the bucket
  - 文件名是变长的
  - 创建/删除需要分配/回收空间
- 优点
  - 简单
  - 查找速度快
- 缺点
  - 哈希表空间不好估计，表大了容易浪费空间，表小了容易产生哈希冲突



创建文件或目录

- 例子：创建文件“/home/os21/fs01.ppt”
  - 解析父目录“/home/os21”，得到其ino，假设为100
  - 读取其i-node，检查用户是否具有创建的权限
  - 根据i-node，读取父目录的内容
  - 查找是否已经存在名字为“fs01.ppt”的目录项
  - 如果找到，同时flag有O_EXCL或为目录，则返回失败，否则转
  - 为“fs01.ppt”分配 一个空闲的i-node，假设其ino为116
  - 填充i-node的内容: ino, size, uid, gid, ctime, mode, …
  - 在父目录的内容中添加一个目录项<“fs01.ppt”, 116>
  - 修改父目录的i-node: size, atime, mtime
  - 把修改写到磁盘: ”fs01.ppt”的i-node，父目录的i-node、父目录内容
  - 创建一个打开文件结构，指向“fs01.ppt”的i-node
  - 分配一个空闲的打开文件结构指针，指向打开文件结构
  - 返回指针的数组下标

删除文件或目录

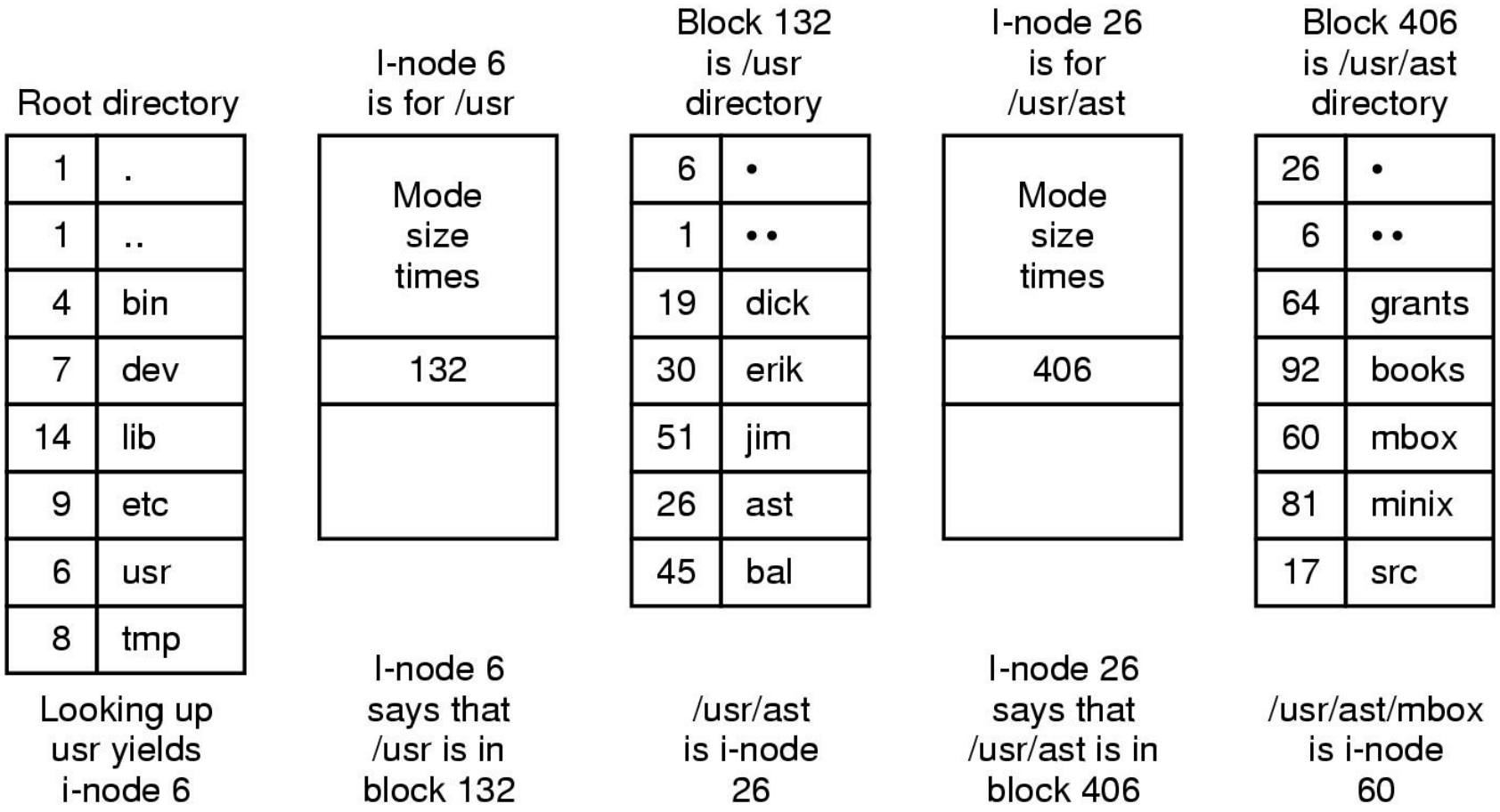
- 例子：删除文件“/home/os21/fs01.ppt”
  - 路径解析父目录“/home/os21”，得到其ino为100
  - 读取其i-node，检查用户是否具有删除的权限
  - 根据i-node，读取父目录的内容
  - 查找是否已经存在名字为“fs01.ppt”的目录项
  - 如果不存在，则返回失败
  - 得到“fs01.ppt”的ino为116
  - 如果nlink为1，则释放i-node及文件块，否则 nlink--
  - 在父目录的内容中删除目录项<“fs01.ppt”, 116>
  - 修改父目录的i-node: size, atime, mtime
  - 把修改写到磁盘: i-node、父目录 i-node、父目录内容、空闲块
  - 返回

内容提要

- 磁盘空间管理
- 文件块索引结构
- 目录项的组织结构
- 文件缓存
- FFS（UNIX文件系统）

路径解析

- 例子：Looking up */usr/ast/mbox* in UNIX

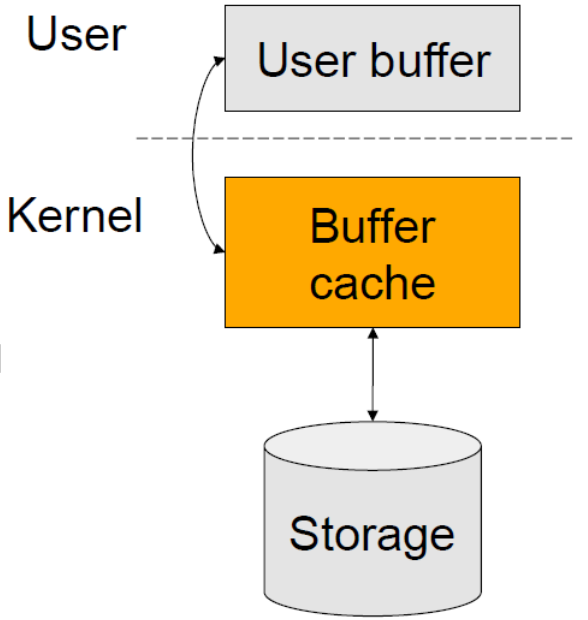


路径解析性能

- 读一个文件: /home/foo，假设读它的第一块
  - 读根目录的i-node和它的第一块
  - 读home目录的i-node和它的第一块
  - 读foo文件的i-node和它的第一块
- 写一个新文件: /home/foo，假设只写入一个块
  - 读根目录的i-node和它的第一块
  - 读home目录的i-node和它的第一块
  - 创建文件foo
  - 写foo的第一块
- 路径解析产生很多I/O

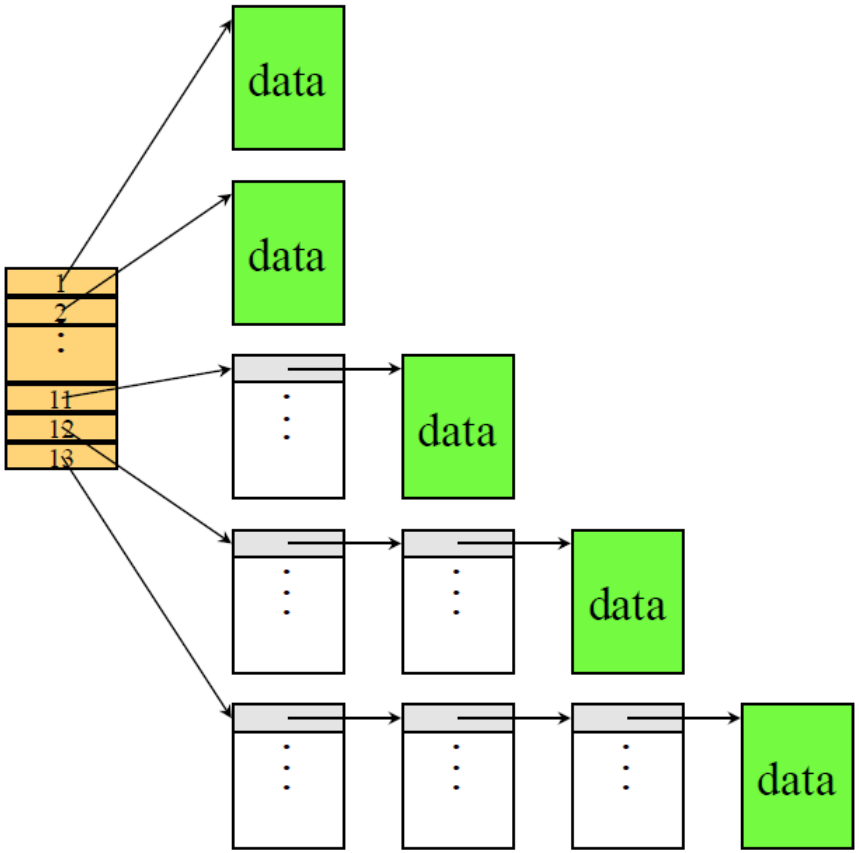
文件缓存 (file buffer cache/page cache)

- 使用内核空间的一部分内存来缓存磁盘块
- 读操作read()：先检查该块是否在缓存中
  - 在：将缓存块的内容拷贝到用户buffer
  - 不在：分配一个缓存块（可能需要替换）
    把磁盘块读到缓存
    再把缓存块拷贝到用户buffer
- 写操作write()：先检查该块是否在缓存中
  - 在：将用户buffer的内容拷贝到缓存块中
  - 不在：分配一个缓存块（可能需要替换）
    将用户buffer的内容拷贝到缓存块中
  - 将该缓存块写回磁盘（根据缓存管理策略）
- 缓存设计问题
  - 缓存什么、缓存大小、何时放进缓存、替换谁、写回策略



缓存什么

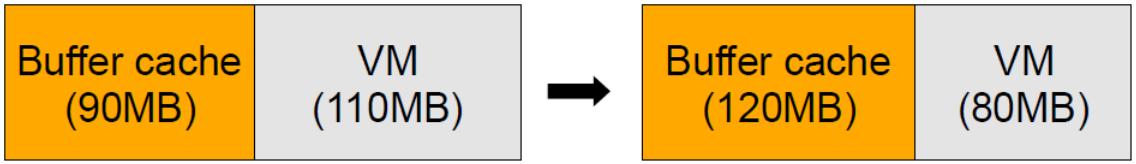
- 不同类型的块
  - i-nodes
  - 间址块
  - 目录
  - 文件块



缓存时所有的块都同等对待么？

缓存大小

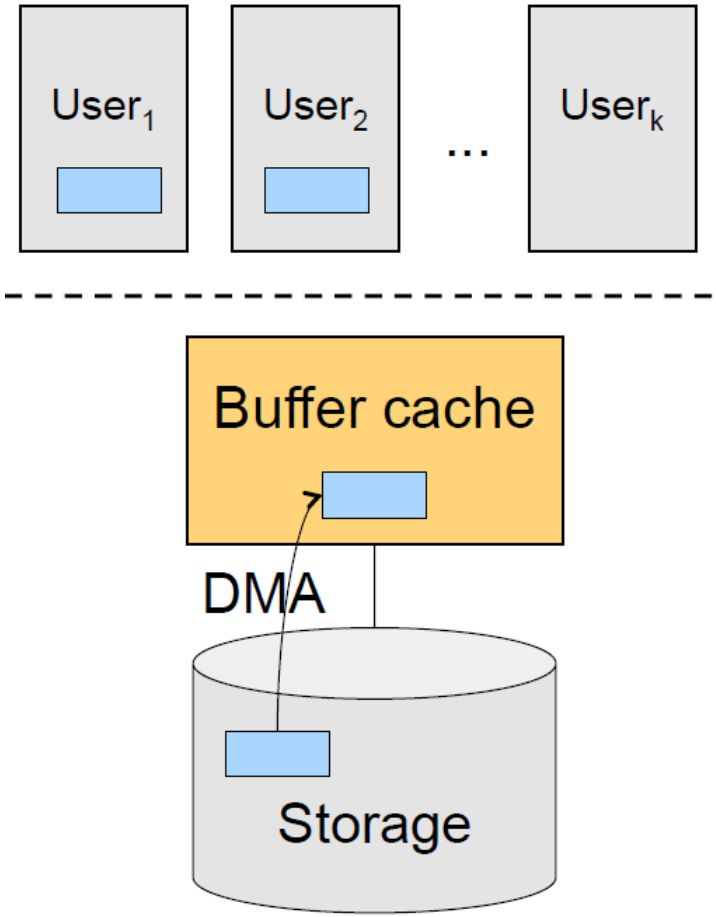
- 文件缓存与VM竞争有限的内存空间
- 两种方法
  - 固定大小
    - 用特权命令设置文件缓存大小



  - 可变大小
    - 文件缓存和VM都按需申请内存：页替换
    - 文件缓存大小不可控

为什么缓存位于内核空间

- DMA
  - DMA需要绑定的物理内存（pin）
- 多用户进程
  - 共享缓存
- 通常的替换策略
  - 全局LRU
  - 进程工作集



预取

- 何时放进缓存：按需取 vs. 预取
- 文件访问具有局部性
  - 时间局部性
  - 空间局部性
- 最优
  - 在要用之前刚好预取进来
- 通常的策略
  - 针对顺序访问的预取：访问第i块时，预取随后的k个块
    - 文件块尽量分配连续的磁盘块
    - Linux采用此方法
  - 针对 i-node的预取：在读取目录项时，同时读取对应的i-nodes
- 高级策略
  - 预取同一目录下的所有小文件

写回策略

- 写操作
  - 数据必须写到磁盘才能持久化
- 缓存中的数据何时写到磁盘上
- Write through (直写)
  - 每个写操作，不仅更新缓存块，而且立即更新磁盘块
  - 好处：简单 & 可靠性高，最新数据都落盘
  - 坏处：磁盘写没有减少
- Write back (回写)
  - 每个写操作，只更新缓存块，并将其标记为“脏”(dirty)
  - 之后再将它写到磁盘
  - 写操作快&减少磁盘写：缓存吸纳多次写(小)，批量写磁盘(大)

回写有什么问题吗？

写回的复杂性

- 丢数据
  - 宕机时，缓存中的“脏”数据将全部丢失
  - 推迟写磁盘 ➔ 更好的性能，但损失更大
- 什么时候写回磁盘
  - 当一个块被替换出缓存（evict）时
  - 当文件关闭时
  - 当进程调用 fsync时
  - 固定的时间间隔（UNIX是30秒）
- 问题
  - 执行写操作的进程并不知道数据什么时候落盘了
    - fsync: 用户显式写回数据
  - 不能保证不丢数据: 宕机或掉电可能发生在任何时候

文件系统 vs. 虚存

- 相似点
  - 位置透明性：用户不感知物理地址
  - 大小无关性：固定粒度分配 (块/页)，不连续分配
  - 保护：读/写/执行权限
- FS比VM容易的地方
  - FS的地址转换可以慢
  - 文件比较稠密（空洞少）， 经常是顺序访问
  - 进程地址空间非常稀疏，通常是随机访问
- FS比VM难的地方
  - 路径解析可能引入多次I/O
  - 文件缓存的空间（内存）总是不够的
  - 文件长度差距大：很多不足10KB，有些又大于GB
  - FS的实现必须是可靠的

虚存页表 vs. 文件块索引

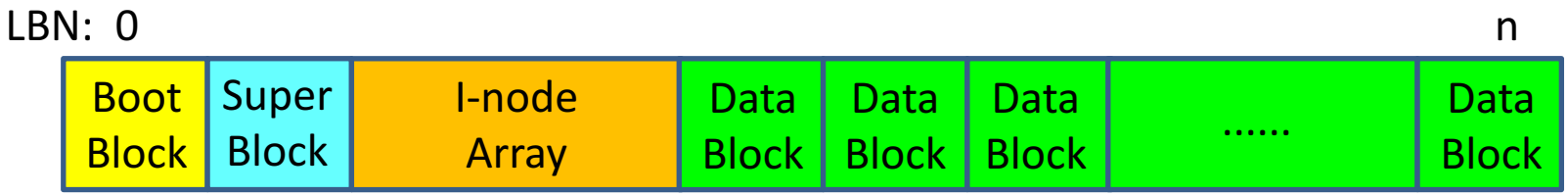
页表

- 维护进程地址空间与物理内存的映射关系
- 虚页号→物理页框号
- 查检访问权限、地址合法性
- 硬件实现地址转换，如果映射关系在TLB中，一个cycle就完成转换

文件块索引

- 维护文件块与磁盘块之间的映射关系
- 文件块号→磁盘逻辑块号
- 查检访问权限、地址合法性
- 软件(OS)实现地址转换，可能需要多次磁盘I/O

示例：最初的UNIX FS

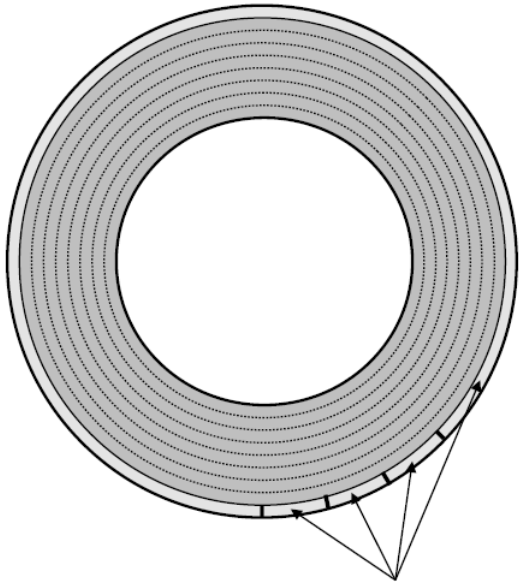


- 简单的磁盘布局
  - 文件块大小 = 扇区大小（512B）
  - i-node区在前，数据区在后
  - 空闲块/inode链表：SuperBlock中记录头指针
- 文件块索引采用三级间址，目录采用线性表

- 存在的问题
  - 带宽很低：顺序访问只有20KB/s，~2%的磁盘带宽 (1984)

导致带宽低的原因

- 数据块的存储位置
  - 数据块存储在内层的柱面
  - i-node存储在外层的柱面
- 频繁长距离寻道
  - i-node与其数据块离得很远
  - 同一目录里的文件，其i-node也离得很远
  - 一个文件的数据块散布在磁盘上任意位置
    - 即使顺序读写文件→随机磁盘I/O
- 未考虑给文件分配连续磁盘块
  - 空闲块采用链表组织
  - 链表上相邻的块，其物理地址不连续
- 小粒度访问多
  - 采用512B的小块
  - 无法发挥磁盘带宽

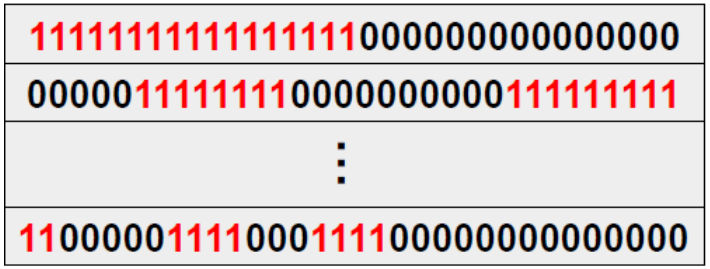


i-node array

示例：BSD FFS（Fast File System）

- 大文件块：4KB或8KB vs. 512B
  - 数据块大小记录在超级块中
  - 空间利用率问题
    - 小文件
    - 大文件的最末一块可能非常小
  - FFS的解决办法：数据块划分为若干更小的子块 (分片)
  - 子块为512B，每块8/16个子块

- 位图（bitmap）：取代空闲块链表
  - 尽量连续分配
  - 预留10%的磁盘空间

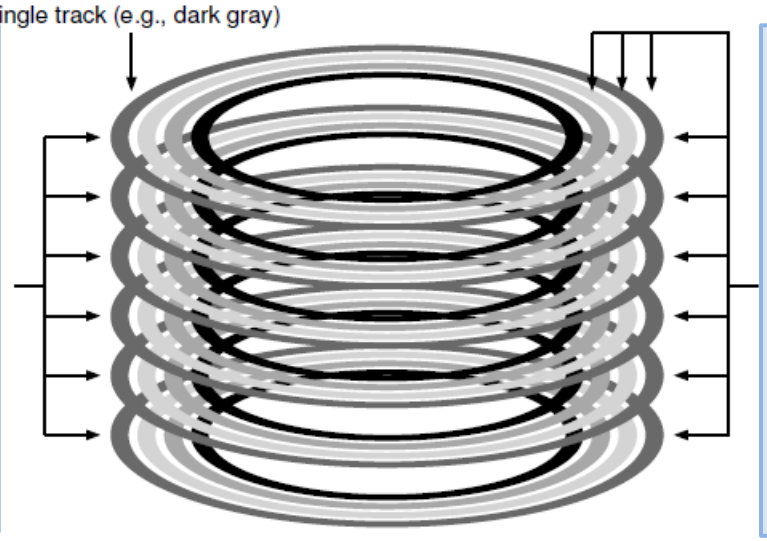


M. McKusick, et al., **A Fast File System for UNIX**, TOCS 1984, http://dl.acm.org/citation.cfm?doid=989.990

FFS的磁盘布局

- 柱面组 (Cylinder Group)
  - CG ：每N个连续的柱面为一个CG

柱面：
所有盘片上半径相同的磁道构成一个柱面。



柱面组：
N个连续的柱面。
此例中柱面组是3个连续的柱面。

  - 把磁盘划分为若干柱面组，将文件和目录分散存储于每个柱面组
  - 每个CG类似一个sub FS
    - 包含超级块、空闲i-node位图、空闲块位图、 i-node表、数据块

FFS的放置策略

- 减少长距离寻道
  - 原则：把相关的东西放在同一CG
- 目录放置
  - 选择CG：目录个数少 & 空闲i-node个数多 & 空闲块多
- 文件放置
  - 文件i-node选择其目录所在的CG
  - 文件块选择其i-node所在的CG
    - i-node与文件块一起读的概率是只读i-node的4倍
    - 例如，ls
- 大文件
  - 应避免它占满一个CG
  - i-node所在CG：前10块 (直接指针指向)
  - 每个间址块及其指向的块放在同一CG (4MB)
  - 不同间址块及其指向的块放在不同CG

FFS的效果

- 性能提升
  - 读性能：s5fs（UNIX FS） 29 KB/s vs. FFS 221 KB/s，7.6x
    CPU利用率: s5fs 11% vs. FFS 43%
  - 写性能：s5fs 48 KB/s vs. FFS 142KB/s, 2.9x
    CPU利用率: s5fs 29% vs. FFS 43%
  - 小文件性能提升

- 进一步的优化空间
  - 块粒度分配和多级索引，大文件访问不高效
    - 用extent来描述连续的数据块（XFS, ext4, etc）
  - 元数据采用同步写，影响小文件性能
    - 异步写，并保证一定顺序
    - 日志

总结

- 数据块管理
  - 空闲块组织：空闲块链表、块位图、 extent 的B 树
  - 空闲块分配：块粒度分配、extent粒度分配（extent内连续，extent间不连续）
  - 文件块索引：块的多级索引、extent的B+树
- 目录项的组织结构
  - 线性表、B树、hash
- 文件缓存
  - 缓存、预取
  - 写回策略

总结

- FFS：根据磁盘的特性来设计FS
  - 减少长距离寻道 & 小粒度访问
  - CG & 大块 & 位图 & 连续分配
  - Co-location @ CG: 文件块与其i-node、文件与其父目录