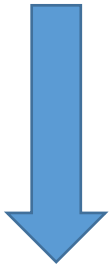
持久化：文件系统实现

蒋炎岩

南京大学 | 计算机软件研究所 | 系统与软件分析研究组

文件系统实现

- 文件系统：一个数据结构(目录、文件)及其操作



如何把目录/文件操作翻译成磁盘操作？

- 磁盘：按固定大小块(block)读写的一个大数组
  - read(#blk, buf); write(#blk, buf)
  - 在本次课中默认块大小为4KB
    - 扇区不是512B么 → 块大小可以大于扇区大小

解决问题：The SYSTEM Way

- 把“实现一个文件系统”转换成以下问题：
  - 文件系统提供怎样的抽象(文件/目录操作)?
  - 文件系统中的目录和文件有何特点?
  - 文件系统的访问模式有何特点?
- 然后再开始考虑具体设计和实现问题
  - 用什么样的数据结构来表示一个文件/一个目录?
  - 存储在磁盘的什么位置……?

在开始之前……

文件和目录：小调研

Summary	Findings
Most files are small	Roughly 2K is the most common size
Average file size is growing	Almost 200K is the average
Most bytes are stored in large files	A few big files use most of the space
File systems contains lots of files	Almost 100K on average
File systems are roughly half full	Even as disks grow, file systems remain ~50% full
Directories are typically small	Many have few entries; most have 20 or fewer

实证研究(Empirical Study)

- 实证研究是计算机系统/计算机软件研究中很重要的一部分
  - 真实系统的需求、行为往往未必是你想象的那样
  - 必须对实际系统的行为进行调研

- 例子
  - 对system workload的study、建模、构造 (例如Cloud Workloads)
  - 对bug模式的study (例如并发bug ≈ deadlock + AV + OV)
  - 对行为的study (例如Github上的开源项目)

计算机系统研究

- 一份(operating) system研究工作[1]
  - It presents a real system, either by a global survey of an entire system or by a selective examination of specific themes embodied in the system.
  - It presents a system that is unimplemented but utilizes ideas or techniques that you feel the technical community should know.
  - It addresses a topic in the theoretical areas, for example, performance modelling or security verification.
- 评价方法
  - original ideas, reality, lessons, choices, context, focus, presentation, writing style

[1] R. Levin and D. Redell. How (and How Not) to Write a Good Systems Paper. In *SIGOPS Operating Systems Review*, 17(3), 1983.

实现文件系统

- 在文件/目录操作时尽可能减少磁盘读写
- 文件如何在磁盘上表示?
- 目录如何在磁盘上表示?

UNIX文件系统：基础设计

文件的属性

- 一个文件有两部分数据
  - 文件的元数据(metadata, inode in the UNIX world)
    - 文件名
    - 访问权限 (试图执行不能执行的程序将会返回permission denied)
    - 时间信息 (Make工具用修改时间判断是否需要更新)
  - 文件的数据(data)，就是个大数组
    - 对于普通文件，操作系统不管其中数据的含义
    - 目录也是数据，也可以看成是文件(存储目录里的文件信息)

Inode (Index Node)

- 文件的元数据 (ext4为例)
  - mode - chmod里的mode drwxrwxrwx
  - uid, gid - 权限
  - size - 大小
  - a/c/m-time (access, creation, modification) – 访问时间
    - 1970年1月1日的秒数；UNIX世界末日：2038年1月19日
  - links_count – 链接数目(引用计数)
  - blocks – 块数目
  - flags – 标志
  - ...

文件系统设计：管理三种数据

- 管理三种数据
  - inode (文件的metadata)
  - 文件的数据(块)

- 因此需要
  - inode bitmap (i) – 用于分配inode
  - data bitmap (d) – 用于分配数据块
  - inode region
  - data region
  - super block (文件系统信息)

例子(参考书)

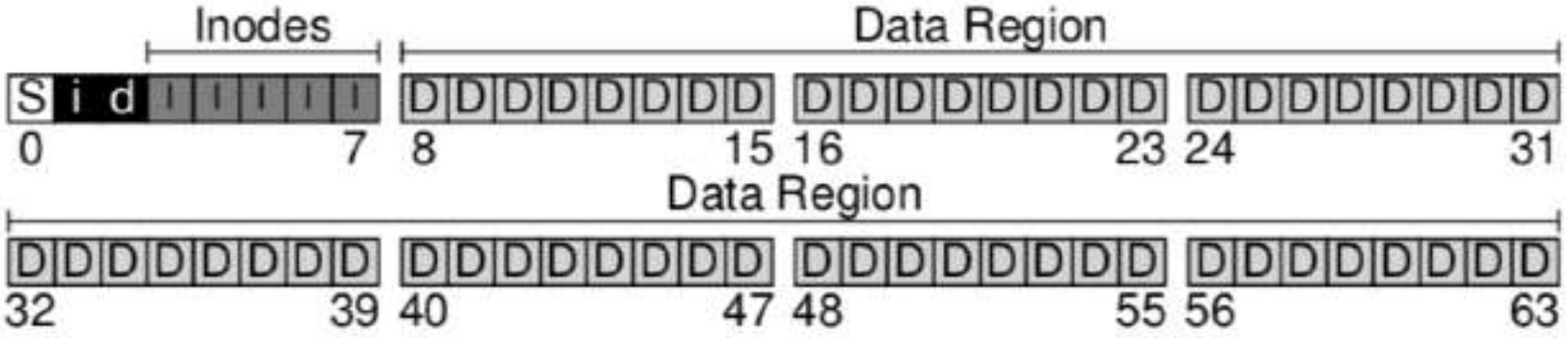
- 给你64个4K blocks的迷你磁盘
- 你如何设计文件系统在磁盘上的layout
- 如何支持
  - 文件操作：read, write, lseek
  - 目录操作：mkdir, link, unlink

Super Block – 文件系统信息

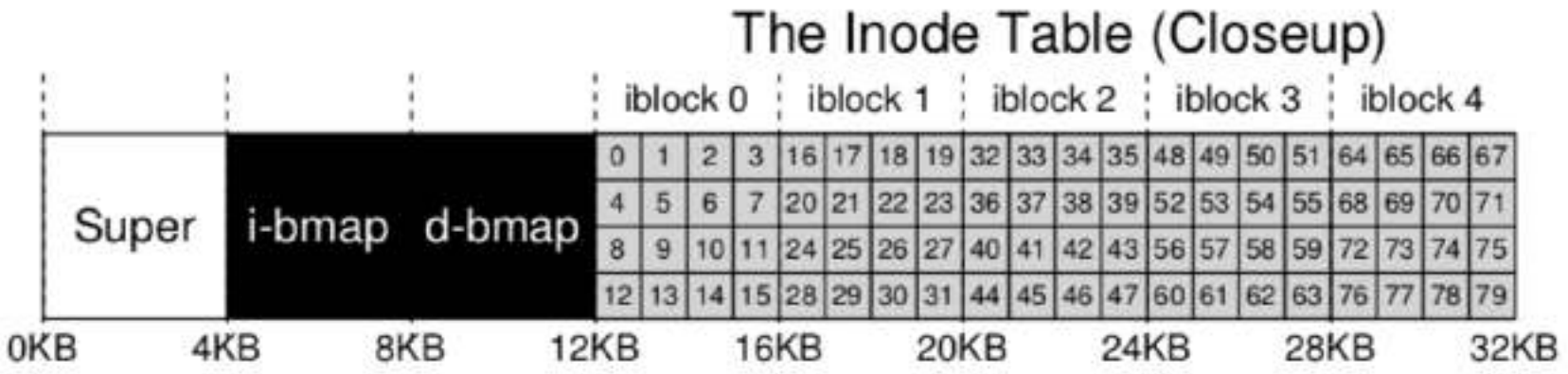
- 文件系统的统计信息
  - inode数量；数据块数量；……
  - df命令可以查看系统中的空间/inode使用情况
    - 都存储在super block，否则需要遍历整个文件系统
    - FAT的统计信息在boot block里(连super block都省了)

磁盘上的数据结构

- superblock (1 block)
  - 结构体，直接存放文件系统元数据
- inode/data bitmap (2 blocks)
  - bitset (4K * 8 = 32K位)
- inode array (256B/inode * 80, 5 blocks)
- data blocks (56 blocks)



磁盘上的数据结构(cont'd)



文件的磁盘表示

文件是数据结构

- 抽象地说，文件是一个映射
  - $f(i) = b$ 把偏移量i映射到某个字节数值
  - ELF文件$f(0) = 0$x7f, $f(1) =$ 'E', $f(2) =$ 'L', $f(3) =$ 'F'

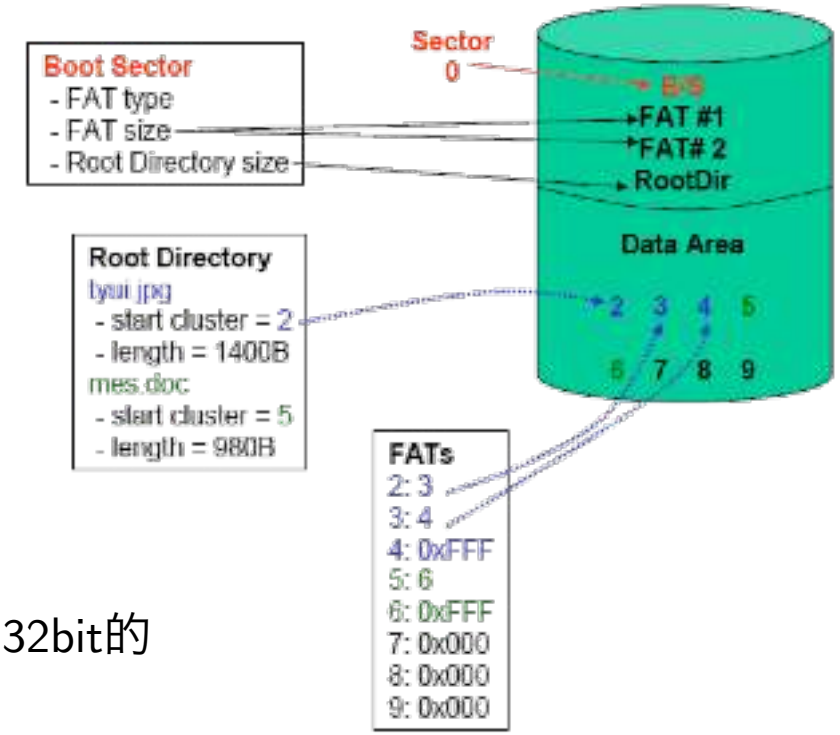
- 支持的主要操作
  - read 读取连续的字节
  - write 写入连续的字节，并且允许超过当前文件大小
  - lseek 改变读写的位置

- 类似于std::vector<char>，但对随机访问的要求没那么高

想法1：链接

- 为每个block维护next指针
  - 在磁盘上存储struct block *next[NUMBLOCKS]

- 这就是File Allocation Table (FAT)
  - 优点
    - FAT缓存在内存，查找不慢
  - 缺点
    - FAT损坏 = gg (怎么办？)
    - 文件碎片问题(为什么？)
    - FAT32不支持4GB文件是因为size是32bit的



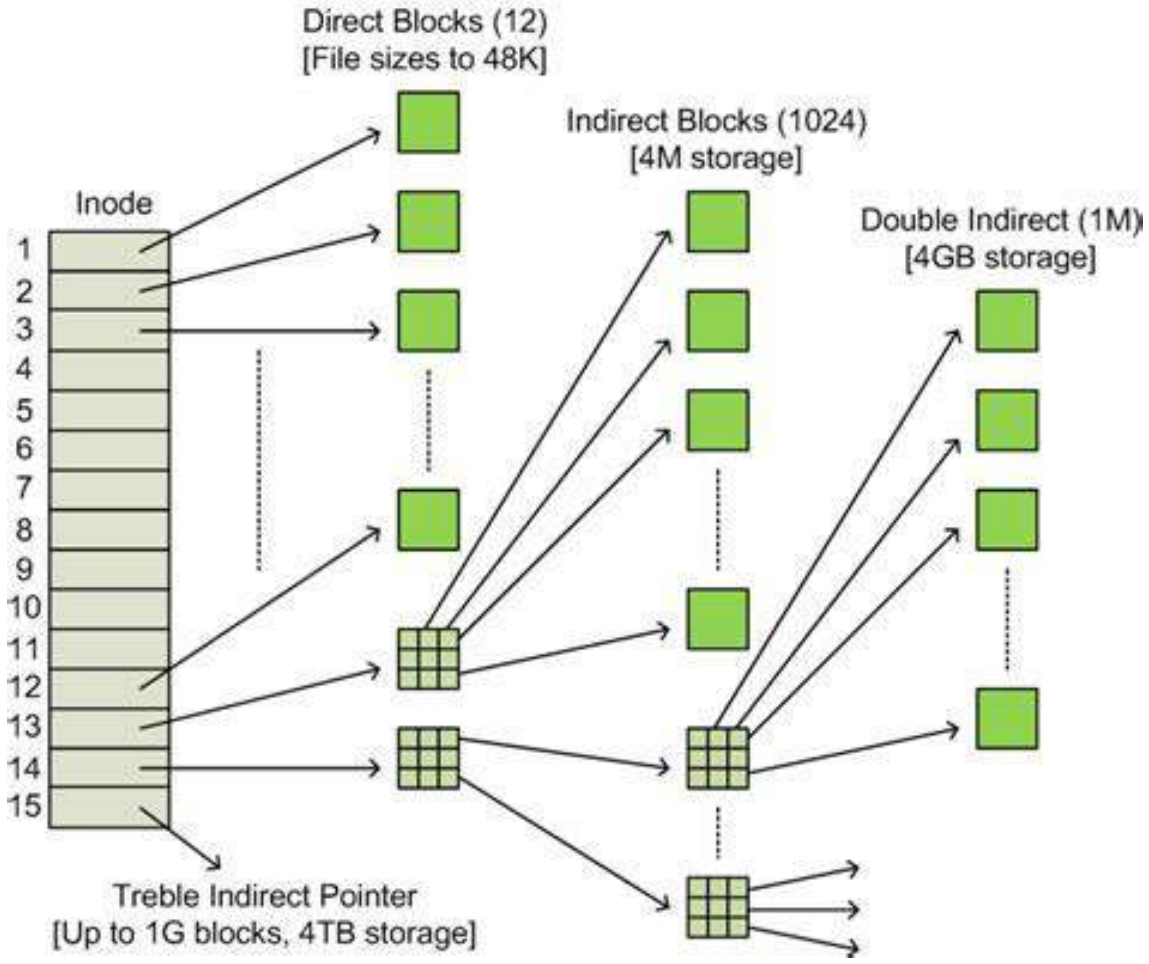
想法2：索引

- 观察：大部分文件是小文件(2KB)；而大文件就是很大

- 索引的设计是不对称的(System的美学)
  - 12个直接索引(12 blocks, 48KB)
  - 1个一级间接索引(1024 blocks, 4MB)
  - 1个二级间接索引(1024^2 blocks, 4GB)

- 好处？坏处？
  - (+) 对于不超过48KB的文件，不需要读/写额外的索引块
  - (+) 对于任何文件，读写头部都是高效的
  - (+) 没有“碎片”
  - (-) 索引块浪费了一定的空间，需要访问时加载

索引：图示


Direct Blocks (12)
[File sizes to 48K]
Indirect Blocks (1024)
[4M storage]
Double Indirect (1M)
[4GB storage]
Inode
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Treble Indirect Pointer
[Up to 1G blocks, 4TB storage]

目录的磁盘表示

目录是数据结构 && 目录是文件

- 目录是一种映射$f(name) = file$
- 目录也是以文件的形式存储(目录文件支持read, write, lseek), 保存目录项(directory entries)
  - 还记得OSLab0扫描目录时用的API吗?

- 目录文件要支持更复杂的映射修改操作
  - mkdir (增加新的目录)
  - link, unlink (目录中记录的管理)
  - ……

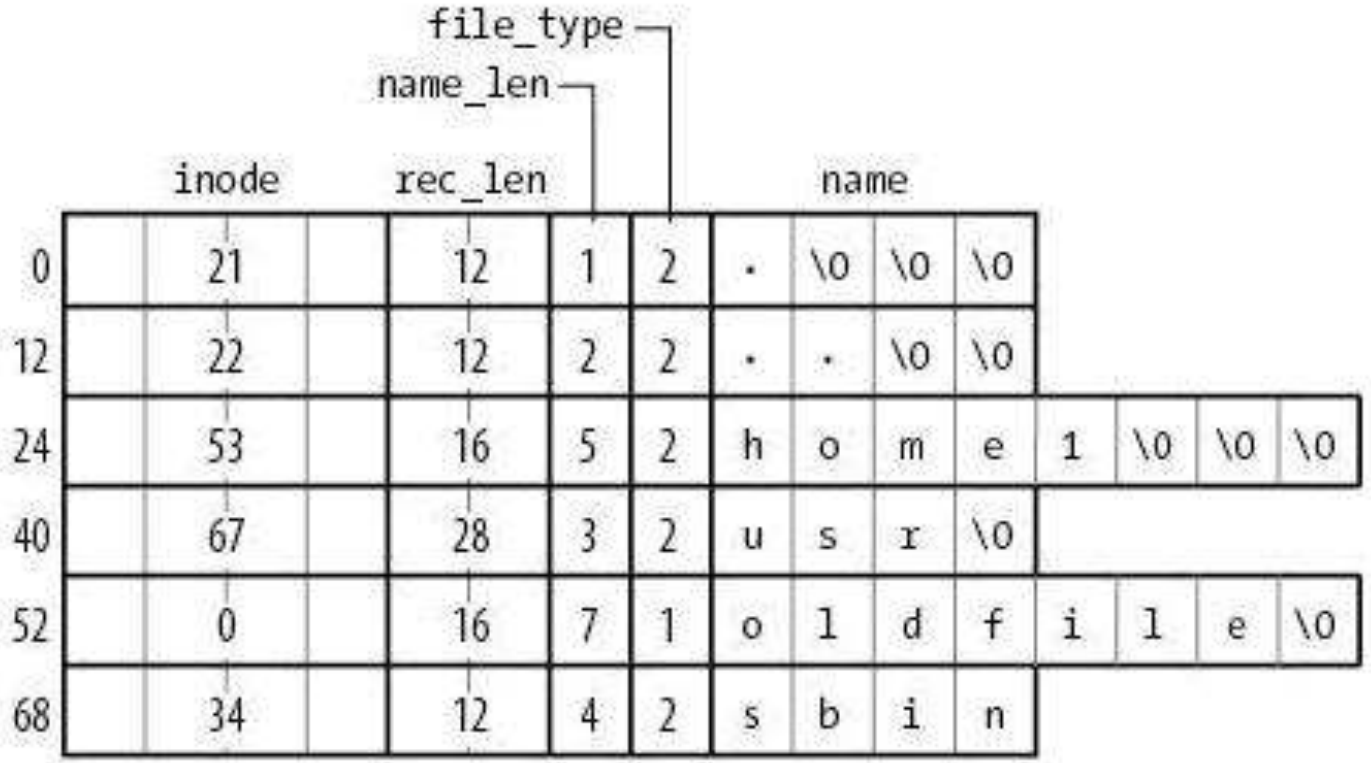
FAT：固定长度的目录项

- FAT16只有8+3字节的文件名
  - autoexec.bat (不知道可有用过的同学？)
  - C:\progra~1 (超过8个字符时的消歧义方法)

- FAT32用连续的目录项表示长文件名

- 好处：插入删除都很容易

ext2: 按顺序存储变长的目录项



ext2: 目录项结构

- 目录中数据的存储方式和文件相同

```
struct ext2_dir_entry_2 {
  __le32 inode; /* inode number */
  __le16 rec_len; /* Directory entry length */
  __u8 name_len; /* Name length */
  __u8 file_type;
  char name[]; /* File name, up to EXT2_NAME_LEN */
};
```

- 未解决的问题
  - 获取目录下的文件需要读出所有内容
  - 修改目录要么留下“洞”，要么需要很多磁盘I/O
  - 越来越复杂……

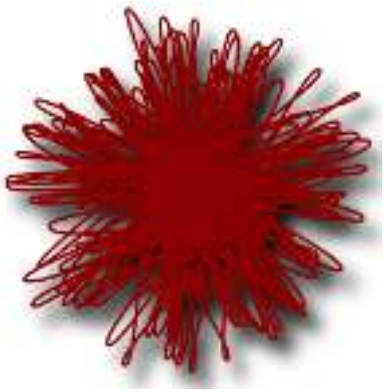
实现文件系统

性能：相比于内存，磁盘是低速设备

- 在memory hierarchy里遇到过这些问题
  - CPU – L1 – L2 – L3 – Memory
- 磁盘也可以用类似的方法解决
  - 缓存
    - 一个block在内存中，只有一段时间以后才被写回到磁盘
  - 预取
    - 根据文件访问的模式，预测未来会访问的块，在访问发生前取出
- 因为软件实现，有更大的灵活性

可靠性：文件系统实现的bug

- 文件系统涉及大量的数据处理
  - 假如磁盘总是由文件系统维护，通常没有问题
  - 但如果恶意的第三方构造磁盘镜像……
- 谁说Kernel代码没有bug?
  - fuzzer (Windows USB stick of death)
  - Linux kernel file system cross-checker[1]



fuzzing
(按在地上摩擦)

[1] C. Min, S. Kashyap, B. Lee, C. Song, T. Kim. Cross-checking Semantic Correctness: The Case of Finding File System Bugs. In *Proc. of SOSP*, 2015.

可靠性：崩溃一致性(Crash Consistency)

- 缓存的数据……断电就没了……

- 操作系统提供了缓存控制API

```
void sync(void);
int syncfs(int fd);
int fsync(int fd);
int fdatasync(int fd);
```

- 以及POSIX其实并没有规定write的数据按什么顺序、写入磁盘
  - 于是有了应用程序的crash consistency bugs

最后

- 考虑更多的需求
  - inode动态分配
  - SSD友好
  - 可靠性/崩溃一致性
  - 快照/回退
  - 逻辑分区、压缩、校验、……
- 到底应该如何设计我们的文件系统?
  - ZFS (the last word in file system), btrfs (开源版本)
  - that's more about theory (engineering不解决全部问题)