Lecture13 Different Filesystem

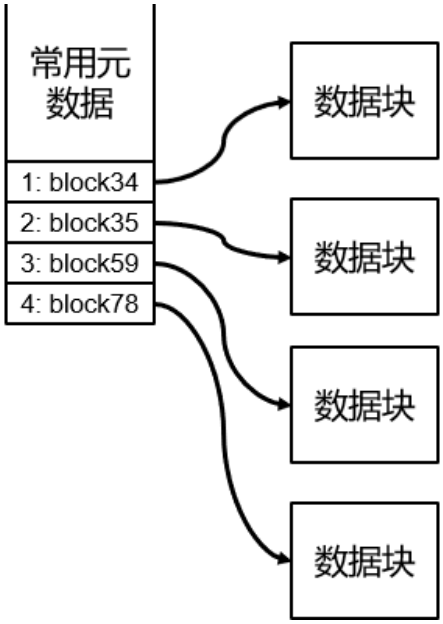
> Forum: 辨析同步与互斥
>
> Discuss: 互斥更倾向于exclusion，同步更倾向于order。互斥不在乎顺序，它只要求只有一方在critical section中；而同步的话，它的order是有具体的语义的，比如producer-consumer问题中，需要保证consumer的front指针一直在producer的rear指针之前这个顺序。

1. 文件系统使用磁盘块的基本单位是什么?
2. 一个文件的组织方式?
3. 空闲空间的组织方式?
4. 目录的结构是什么?
5. 是否支持硬链接?
6. 是否支持软链接?
7. 磁盘存储的整体布局是什么?
8. 如何根据文件名查找到一个文件?
9. 如何读取一个文件?
10. 如何为一个文件分配新的磁盘空间?
11. 如何挂载一个文件系统?

1. 回顾：Unix V6文件系统

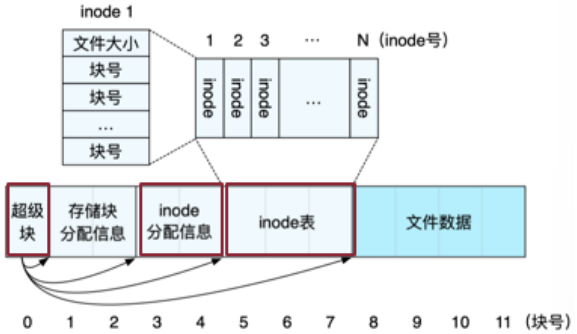
文件索引的节点：inode

- 常用的元数据
  - 文件类型
  - 文件大小
  - 链接数
  - 文件权限
  - 拥有用户/组
  - 时间（创建、修改、访问时间）
- 具体文件数据的位置



inode文件系统的存储布局

- **inode表：记录所有inode**
  - 可以看成inode的大数组
  - 每个inode使用作为索引
  - 此时，inode号即为文件名
- **inode分配信息（位图）**
  - 记录哪些inode已分配，哪些空闲
- **超级块：Super Block**
  - 记录磁盘块的大小、其他信息的起始磁盘块位置，等等
  - 是整个文件系统的元数据

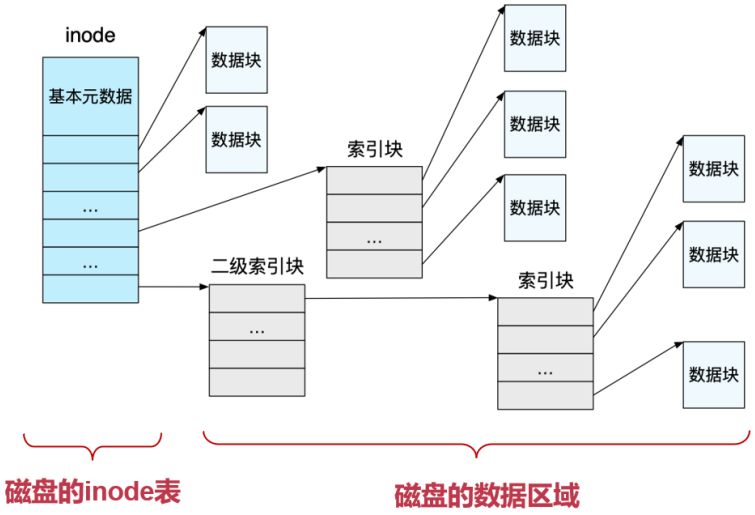


inode文件系统的基本操作

- 加载文件系统
  - 首先要读取超级块，然后找到其他信息
- 创建新文件
  - 根据inode分配信息找到空闲inode，将inode对应的bit设置为1
  - 返回inode在inode表中的索引，作为文件名
- 查找文件（根据inode号）
  - 在inode表中根据inode号定位该inode
- 删除文件
  - 在inode分配表中，将该inode对应的bit设置为0

多级inode

- 引入**索引块**：指向数据块；以及**二级索引块**：指向索引块；...
- 索引块（包括二级索引块）不在inode表的存储区域，而是在数据区域

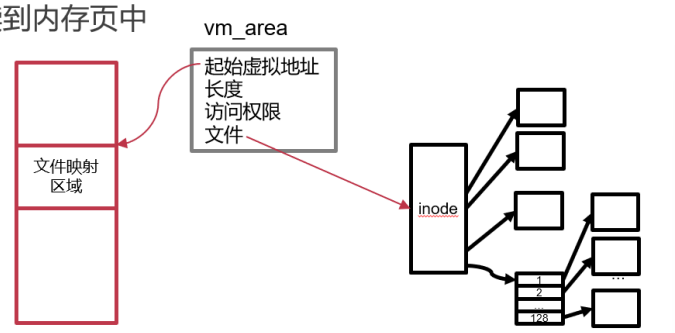


> Q: 多级inode和多级页表有什么相似和不同?
>
> - 相似：本质都是一个翻译的过程，inode是把文件的offset翻译成磁盘块号，页表是从va->pa
> - 不同之处
>
> 1. inode中存放的block指针一定是连续的，但是va不一定是连续的
> 2. inode是软件的实现，va->pa是由MMU硬件实现的，不太灵活
>
> 两种实现的pattern，可以用是否有空洞来区分

mmap()：用内存接口来访问文件

- mmap可将文件映射到虚拟内存空间中
  1. mmap时分配虚拟地址，并标记此段虚拟地址与该文件的inode绑定
  2. 访问mmap返回的虚拟地址时，触发缺页中断（page fault）
  3. 缺页中断处理函数，通过虚拟地址，找到该文件的inode
  4. 从磁盘中将inode中对应的数据读到内存页中
  5. 将内存页映射添加到页表中

```
fd = open("/OS/考试", O_RDWR);
addr = mmap(NULL, length, PROT_WRITE,
            MAP_SHARED, fd, 0);
memset(addr, 0, length);
```



mmap()：文件内存映射的优势

1. 对于随机访问，不用频繁的lseek
2. 减少系统调用次数
3. 可以减少数据copy

> 比如文件拷贝，数据无需经过中间的buffer

4. 访问的locality更好
5. 可以用madvice为内核提供访问提示，提高性能

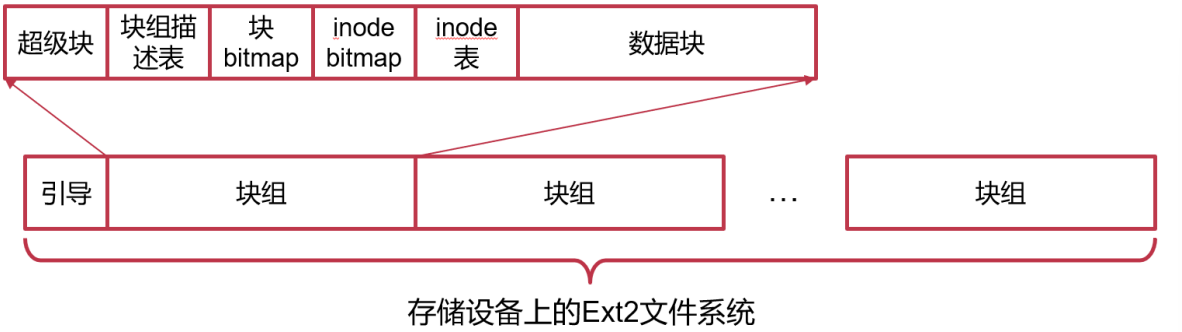
2. EXT2文件系统

存储布局

将磁盘分为多个块组，每个块组中都有超级块，互为备份

超级块（Super Block）记录了整个文件系统的元数据

块组描述表记录了块组中各个区域的位置和大小



使用区段(Extent)来优化

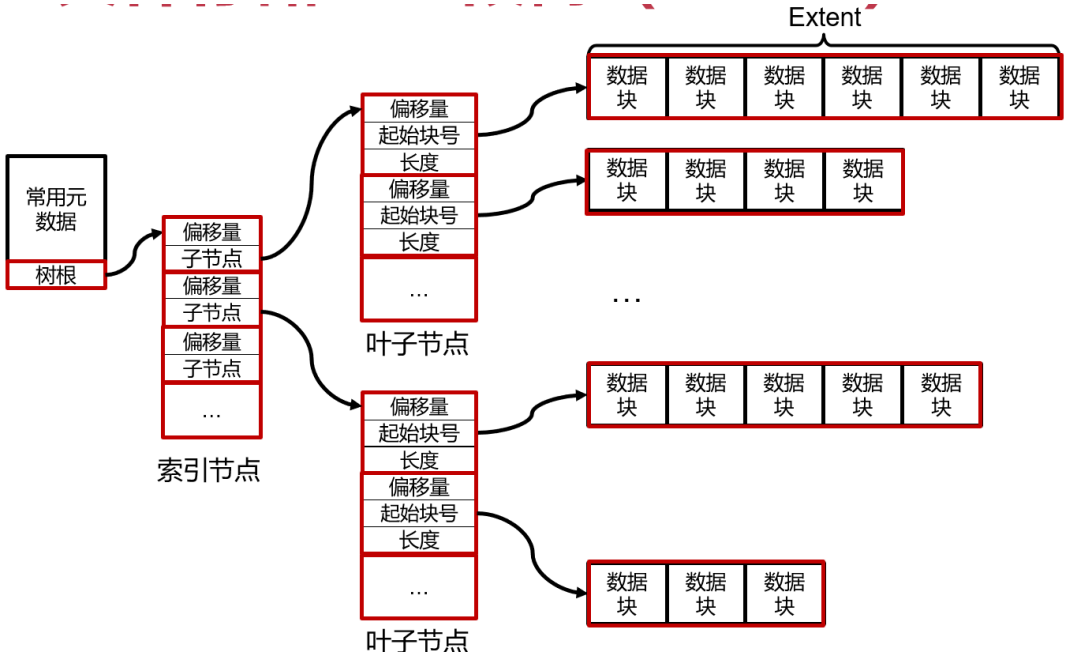
问题：在Ext2的设计中，保存一个1GB的视频文件，文件被拆成多少数据块？需要多少元数据来维护这些数据块？

如果这些数据块物理上连续，只需要保存**起始块地址和长度**即可！

区段（Extent）是由**物理上连续的多个数据块**组成

- 一个区段内的数据可以连续访问，无需按4KB数据块访问
- 可以减少元数据的数量

Ext4文件存储-区段树(Extent)

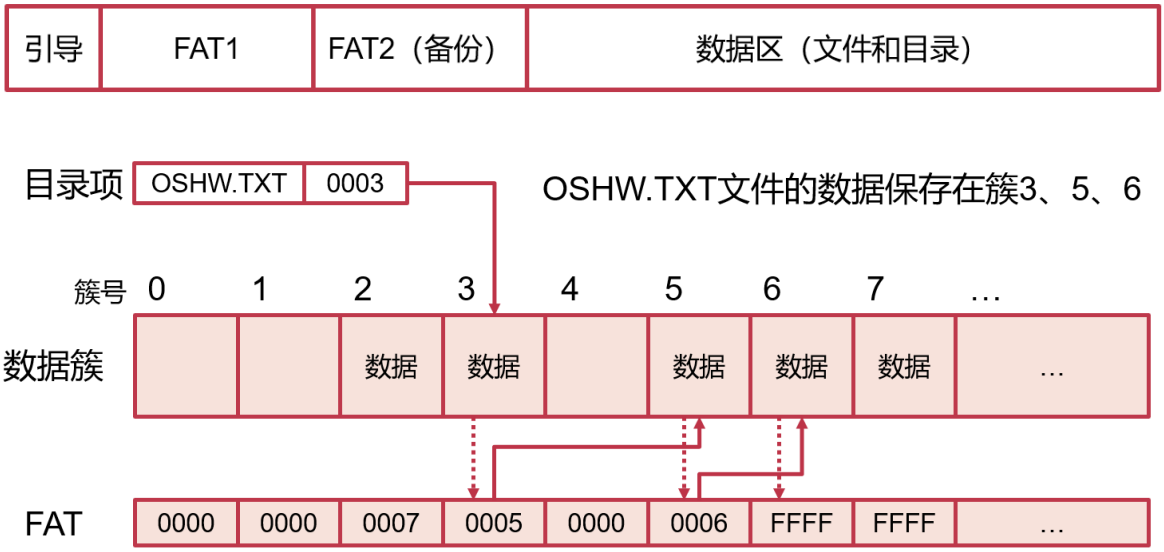


Ext2的目录文件

> 和Unix的差不多，复用常规文件格式，目录项连续存放

3. 基于Table的文件系统：FAT

FAT：文件分配表



FAT为每个数据簇增加了一个**next**指针，让簇可以串联在一起

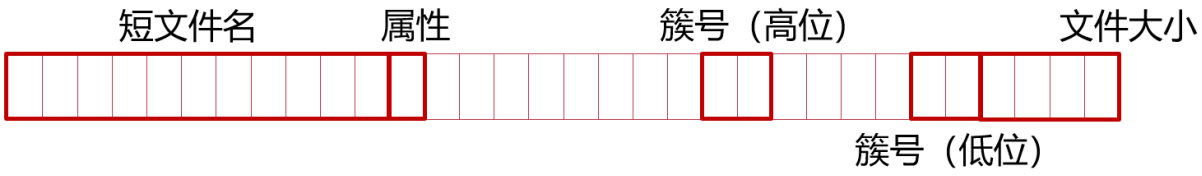
FAT32中的目录项

目录同样是一种（特殊的）文件 —— 与基于inode的文件系统一样

目录文件包含若干个目录项，每个目录项记录32个字节

四种目录项：

短文件名目录项、长文件名目录项、卷标目录项、"."和".."目录项



一些思考

> Q1：FAT32最大支持多大的单个文件？为什么？
>
> A1：最大支持4GB的文件，因为目录项中的文件大小用4bytes表示

> Q2：应该如何扩展FAT，使其能够支持更大的文件？

> A2：用更多字节保存filesize

> Q3：为什么U盘一般用FAT?

> A3：一方面是因为兼容性问题（Mac和Windows都兼容），还有一个因素是比较简单

> Q4：为什么FAT不支持link（硬链接）？

> A4：因为文件名成为了文件的元数据，所以不能hard link

> Q5：为什么有的时候存储的照片会花掉?

> A5：因为可能写到一般被拔掉，next指针位置指错了，只要一个错后面的都错

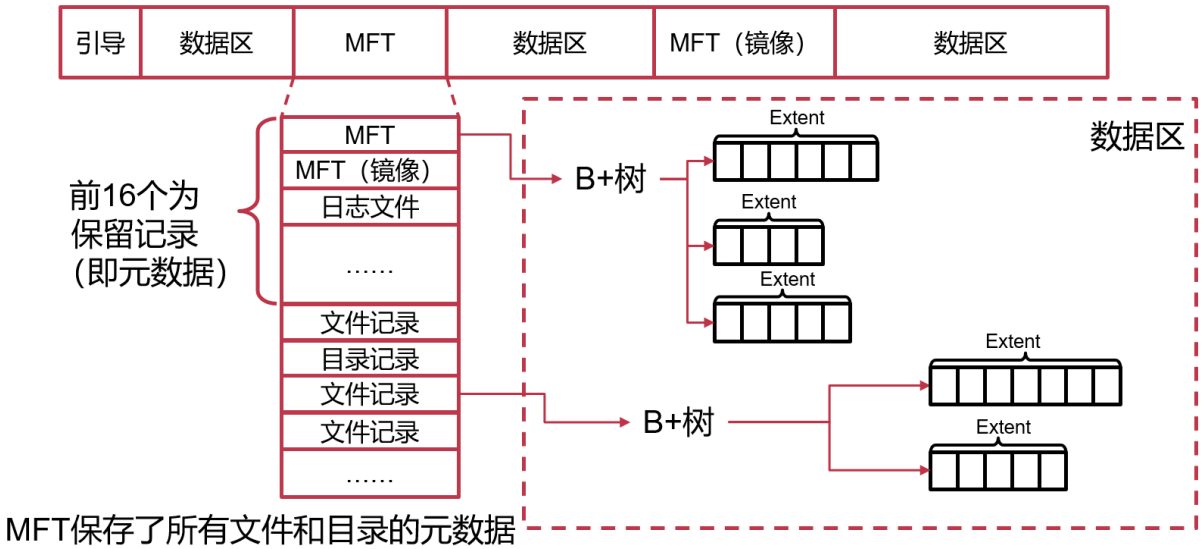
> Q6：为什么FAT会有大量的随机读写?

> A6：因为目录项对比和用next查询下一块数据

4. 基于数据库的文件系统：NTFS

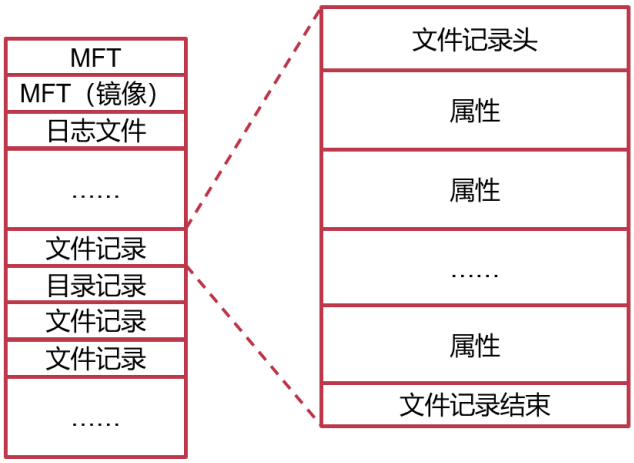
NTFS主文件表MFT

- **MFT是一个关系型数据库（from 微软文档）**
  - MFT中的每一行对应着一个文件
  - 每一列为这个文件的某个元数据
  - NTFS 中所有的文件均在 MFT 中有记录
  - 一般会预留整个文件系统存储空间的12.5%，专门保存MFT
- **一切皆文件**
  - NTFS 中的所有被分配使用的空间均被某个文件所使用
  - 用于存放文件系统元数据的空间，也会属于某个保留的元数据文件
  - 如：MFT本身，也是一个文件，其元数据保存在MFT中（递归了？）



MFT保存了所有文件和目录的元数据

主文件表记录



常用属性包括：

- 文件标准元数据（大小、时间等）
- 文件名
- 数据
- 索引根

NTFS数据保存位置和目录项

- 非常驻文件（大文件/目录）
  - 数据区的B+树和区段
- 常驻文件（小文件/目录）
  - 大小不超过MFT记录的最大值（1KB）
  - 内嵌在MFT中保存（在"数据"属性中）
- 目录项与硬链接
  - 包含文件名、文件ID（在MFT中的序号）
  - 支持硬链接：每个硬链接拥有一个单独的目录项

思考时间

> Q：为什么Everything查找文件这么快?
>
> A：因为文件的信息全都放在了MFT中，顺序存储

> Q：为什么NTFS存取小文件很高效?
>
> A：因为小于1KB的文件会直接放在MFT中

> Q：为什么ls也很快?
>
> A：