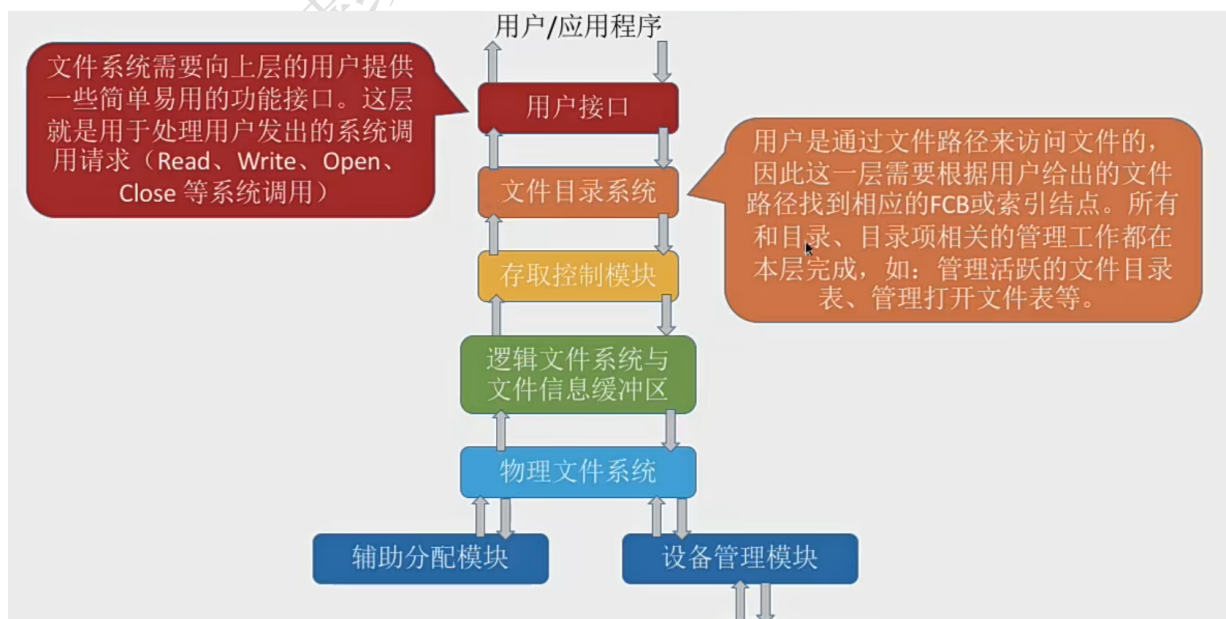


# Lecture 10: 知识 - 文件系统

## 文件系统的层次结构

### 概述

- 用户接口：
  - 文件系统需要向上层的用户提供一些简单易用的功能接口。这层就是用于处理用户发出的系统调用请求 (Read、Write、Open、Close 等系统调用)；
- 文件目录系统：
  - 用户是通过文件路径来访问文件的，因此这一层需要根据用户给出的文件路径找到相应的FCB 或索引结点；
- 存取控制模块：
  - 所有和目录、目录项相关的管理工作都在本层完成，如：管理活跃的文件目录表、管理打开文件表等；
- 逻辑文件系统与文件信息缓冲区：
  - 用户指明想要访问文件记录号这一层需要将记录号转换为对应的逻辑地址
- 物理文件系统：
  - 这一层需要把上一层提供的文件逻辑地址转换为实际的物理地址
- 辅助分配模块：
  - 负责文件存储空间的管理即负责分配和回收存储空间
- 设备管理模块：
  - 直接与硬件交互，负责和硬件直接相关的一些管理工作。如：分配设备、分配设备缓冲区、磁盘调度、启动设备、释放设备等
- 图示：



## 文件系统的全局结构

## 文件系统在外存

概念：文件系统在磁盘中的结构

- 主引导记录：
  - Master BootRecord，简称 MBR
  - 构成：
    - (1) 磁盘引导程序；
    - (2) 分区表；
  - 概念：
    - 位于磁盘的 0 号扇区，用来引导计算机，MBR 的后面是分区表，该表给出每个分区的起始和结束地址；
    - 表中的一个分区被标记为活动分区。当计算机启动时，BIOS 读入并执行 MBR；
    - MBR 做的第一件事是确定活动分区，读入它的第一块，即引导块；
- 引导块：
  - MBR 执行引导块中的程序后，该程序负责启动该分区中的操作系统；
  - 每个分区都是统一从一个引导块开始，即使它不含有一个可启动的操作系统，也不排除以后会在该分区安装一个操作系统；
    - Windows 系统称之为分区引导扇区；
  - 除了从引导块开始，磁盘分区的布局是随着文件系统的不同而变化的；
- 超级块：
  - Super Block
  - 概念：
    - 包含文件系统的所有关键信息，在计算机启动时，或者在该文件系统首次使用时，超级块会被读入内存；
  - 构成：
    - 分区的块的数量；
    - 块的大小；
    - 空闲块的数量和指针；
    - 空闲的 FCB 数量；
    - FCB 指针；

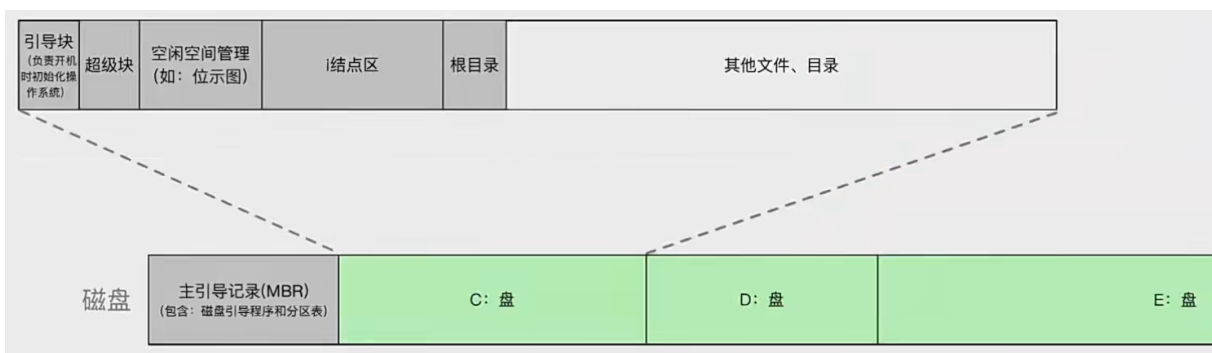
概念：文件系统在外存中的建立过程

- 概述：
  - 文件系统如何再外盘中逐渐建立；
- 第一步：物理格式化
  - 1. 低级格式化 - 划分扇区，检测坏扇区；
  - 2. 检测到坏扇区后，用备用扇区替换坏扇区；



- 第二步：高级格式化
  - 逻辑格式化：
    - 将磁盘分区 (分卷 Volume)，完成各分区的文件系统初始化；
    - 注意：逻辑格式化后，灰色部分就有实际数据了，白色部分还没有数据；

- 检查位视图：
  - 判断哪些盘块是否空闲；
- 超级块：
  - 迅速的找到若干个空闲的盘块；
- 索引区（i 节点区）：
  - 存放若干的索引节点；
  - 通过一个索引结点的下标，迅速的定位到一个索引节点；
- 根目录：
- 图示：

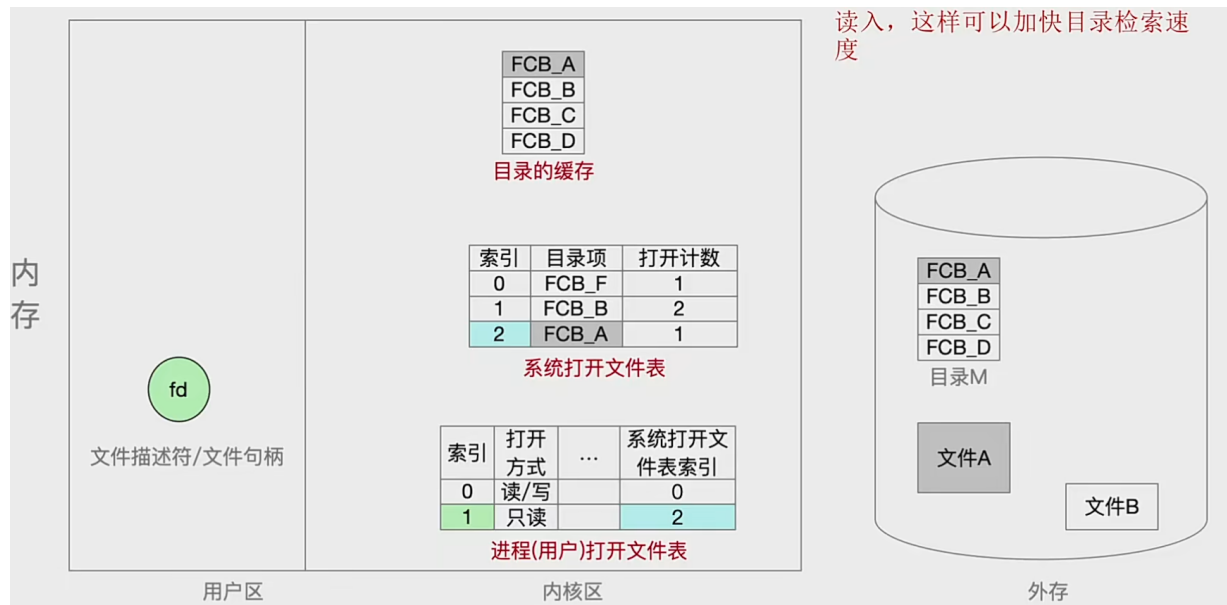


## 文件系统在内存

**概念：**文件系统在内存中的结构

- 目的：
  - 内存中文件系统的信息用于管理文件系统，并通过缓存提高性能；
- 构成：
  - 目录缓存：
    - 最近访问过的目录文件的数据，会被放在目录的缓存中；
  - 系统打开文件表：
    - 整个 OS 只有一张；
    - 包含每个打开文件的 FCB 副本、打开计数以及其他信息；
  - 进程打开文件表：
    - 每个进程一张；
    - 包含进程打开的文件的文件描述符；
    - 包含指向整个系统的打开文件表中对应表项的指针；
- 过程：
  1. Open 调用打开目录，找到目录 M，读入主存，放入目录 M 缓存 -> 检查目录项；
  2. 找到目标文件的 FCB，复制到系统打开文件表，表示这个文件被打开，设置打开计数为 1；
  3. 在进程打开文件表中新建一个条目，并返回文件描述符，并且建立一个指向系统打开文件表；
  4. 用户根据 fd 文件描述符，此时可以执行操作，如 read(fd)，在进程打开文件表中找到目标条目，然后根据索引信息，从系统打开文件表找到目标文件的 PCB，然后在找到目标文件；

• 图示：



# 外存空闲空间管理

**总结：**可用于文件系统空闲磁盘块管理的数据结构

- (1) 位图
- (2) 空闲链表法；
- (3) 空闲表法；
- (4) 成组链接法；
- (5) **文件分配表 FAT**；
  - 一个文件分配表的表项对应一个磁盘块，可以用一个特殊数字 (如 0)，标记磁盘块为空闲，因此文件分配表也可以用于管理空闲磁盘块；
- **真题：** 2024年
  - 题目：

26. 文件系统需占用部分外存空间记录空闲块位置，下列方法中，占用外存空间的大小与当前空闲块数量无关的是( )。

A. 位图法                      B. 空闲表法                      C. 成组链接法                      D. 空闲链表法
  - 答案：

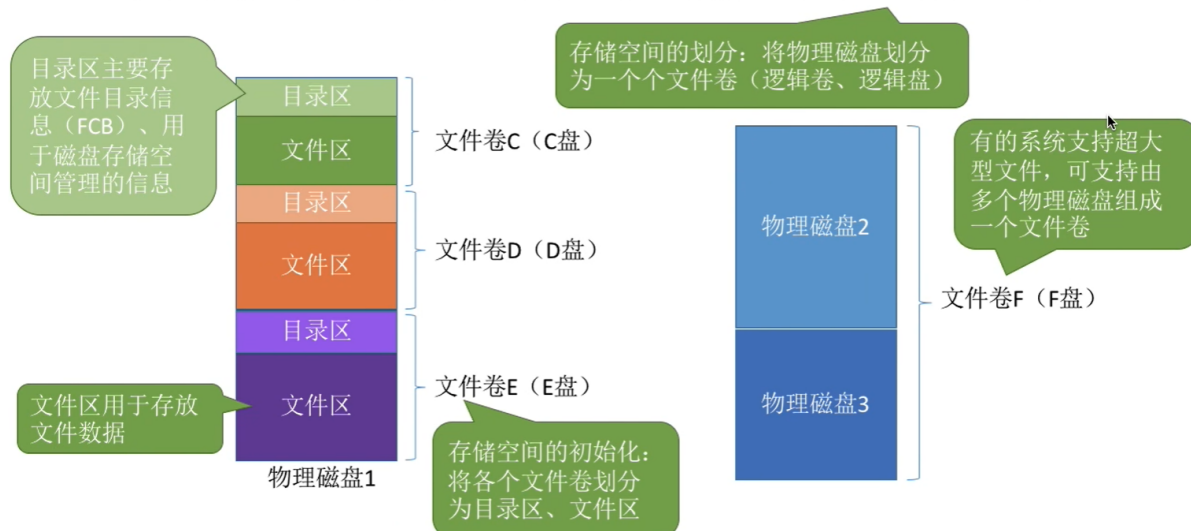
A
  - 分析：
    - 空闲表法需要用额外的一张表来记录所有空闲的磁盘块，而空闲链表法需要将空闲磁盘块连接在一起，因此这两种方法都是空闲块越多需要的额外空间越多；
    - 成组链接法把空闲块分为若干组，每 100 个空闲块为一组，每组的第一个空闲块记录了空闲块总数和下一组物理空闲块的物理盘块号，因此空闲块越多、所需空间越大；
    - 位示图用于表示一个磁盘块是否空闲，因此无论磁盘块是否空闲都占用空间，其大小仅取决于磁盘的**总块数**；

# 存储空间的划分与初始化

**概念：**存储空间的划分与初始化基础概念

- 初始化：
  - 将各个文件卷划分为目录区、文件区
- 文件卷：
  - 将物理磁盘划分为一个个文件卷 (逻辑卷、逻辑盘)
  - 有的系统支持超大型文件，可支持由多个物理磁盘组成一个文件卷；
- 目录区：
  - 主要存放文件目录信息 (PCB)、用于磁盘存储空间管理的信息；
- 图示：

安装 Windows 操作系统的时候，一个必经步骤是——为磁盘分区 (C: 盘、D: 盘、E: 盘等)



## 空闲表法

**概念：** 空闲表

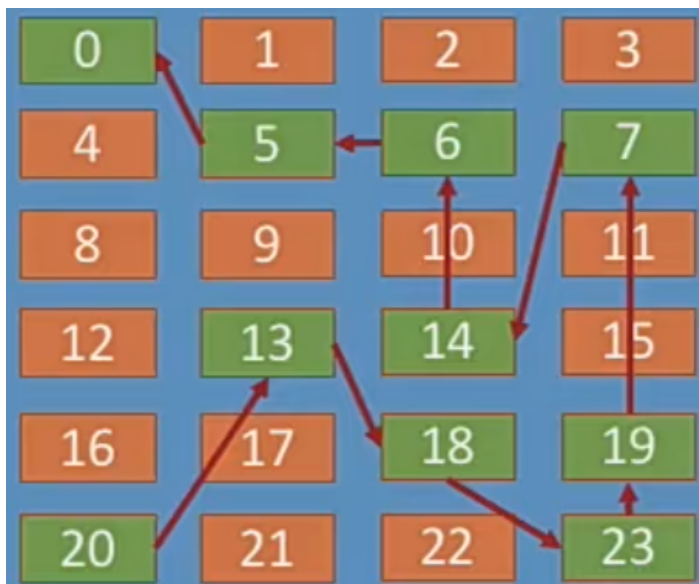
- 概念：
  - 记录每一个空闲盘块的块号以及其后续空闲盘块的数量，将其存储在表内；
- 概念： 空闲表法的回收
  - 与内存管理中的动态分区分配很类似，当回收某个存储区时需要有四种情况：
    - 1. 回收区的前后都没有相邻空闲区；
    - 2. 回收区的前后都是空闲区；
    - 3. 回收区前面是空闲区；
    - 4. 回收区后面是空闲区；
- 适用：
  - 适用于**连续分配方式**；
- 图示：



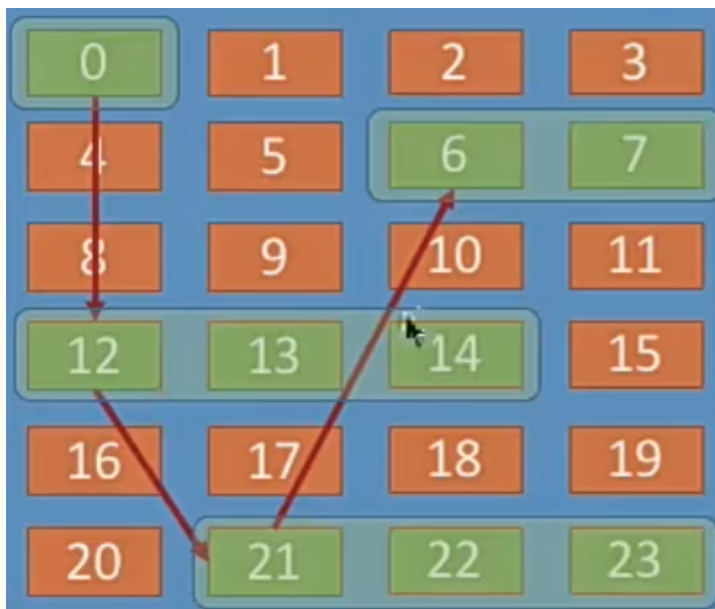
## 空闲链表法

**概念：**空闲链表法 - 两种方法

- 方法一：空闲盘块链
  - 概念：
    - 以**盘块**为单位，组成一条空闲链；
    - 操作系统保存着**链头**、**链尾**指针；
  - 分配：
    - 若某文件申请  $K$  个盘块，则从链头开始依次摘下  $K$  个盘块分配，并修改空闲链的链头指针；
  - 回收：
    - 回收的盘块依次挂到链尾，并修改空闲链的链尾指针；
  - 特点：
    - 优点：
      - 分配和回收一个盘块的过程非常简单；
    - 缺点：
      - 为一个文件分配盘块时可能要重复多次，效率较低；
      - 比较难得到连续的分配空间；
  - 图示：



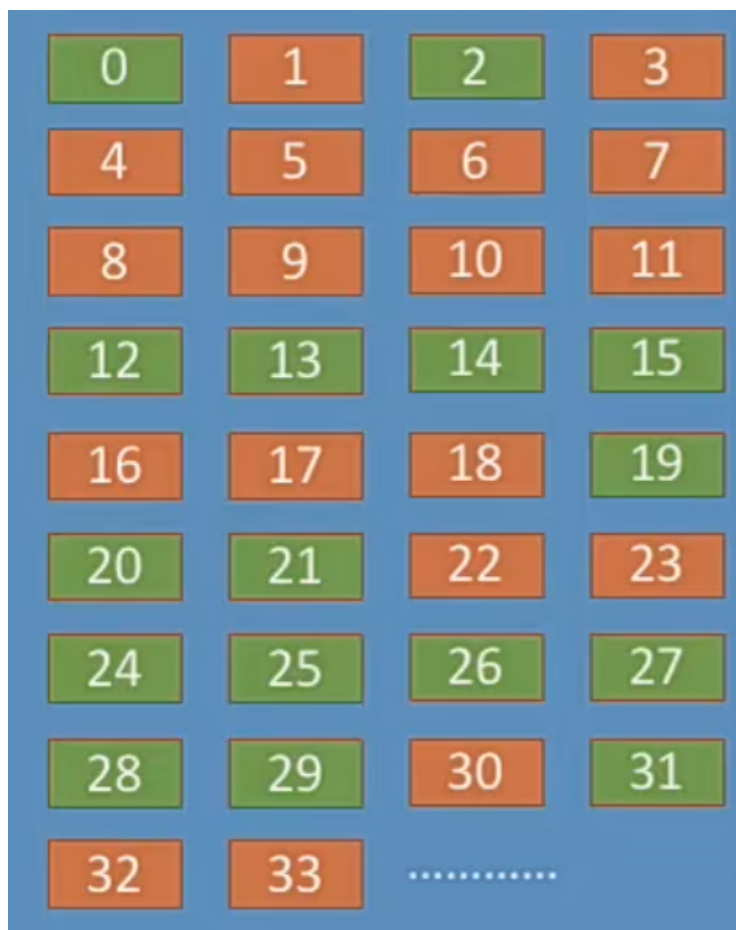
- 方法二：空闲盘区链
  - 概念：
    - 以盘区为单位，组成一条空闲链，连续的空闲盘块组成一个空闲盘区；
    - 空闲盘区中的第一个盘块内记录了盘区的长度、下一个盘区的指针；
  - 分配：
    - 若某文件申请  $k$  个盘块，则可以采用首次适应、最佳适应等算法，从链头开始检索，按照算法规则找到一个大小符合要求的空闲盘区分配给文件；
    - 若没有合适的连续空闲块，也可以将不同盘区的盘块同时分配给一个文件，注意分配后可能要修改相应的链指针、盘区大小等数据；
  - 回收：
    - 若回收区和某个空闲盘区相邻，则需要将回收区合并到空闲盘区中；
    - 若回收区没有和任何空闲区相邻，将回收区作为单独的一个空闲盘区挂到链尾；
  - 适用范围：
    - 离散分配、连续分配都适用；
    - 为个文件分配多个盘块时效率更高；
  - 特点：
    - 和空闲盘块链相反。回收和分配过程复杂，但回收和分配效率高；
  - 图示：



## 位示图法

**概念：**位示图法

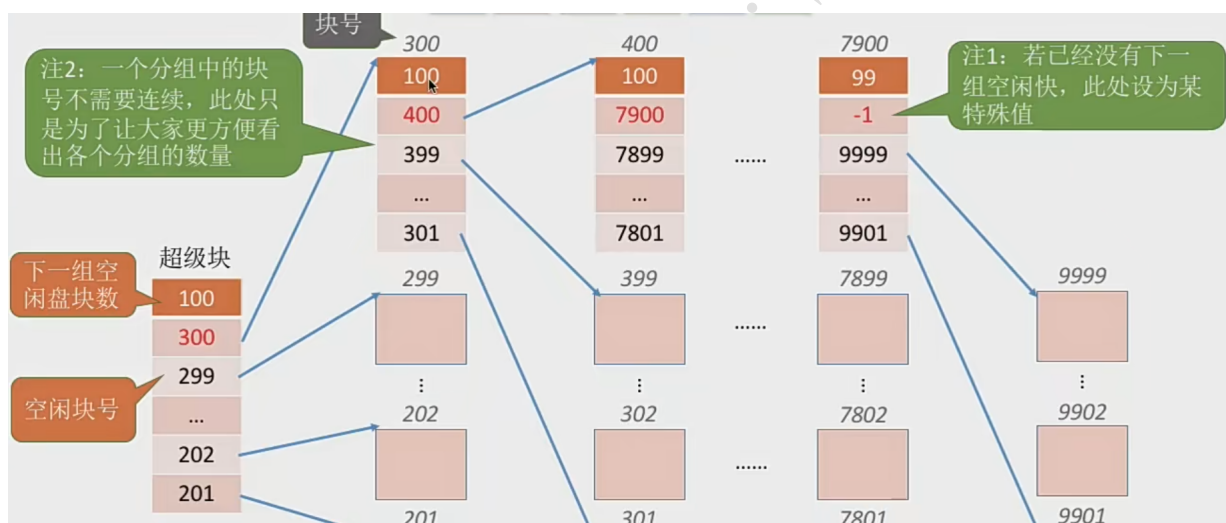
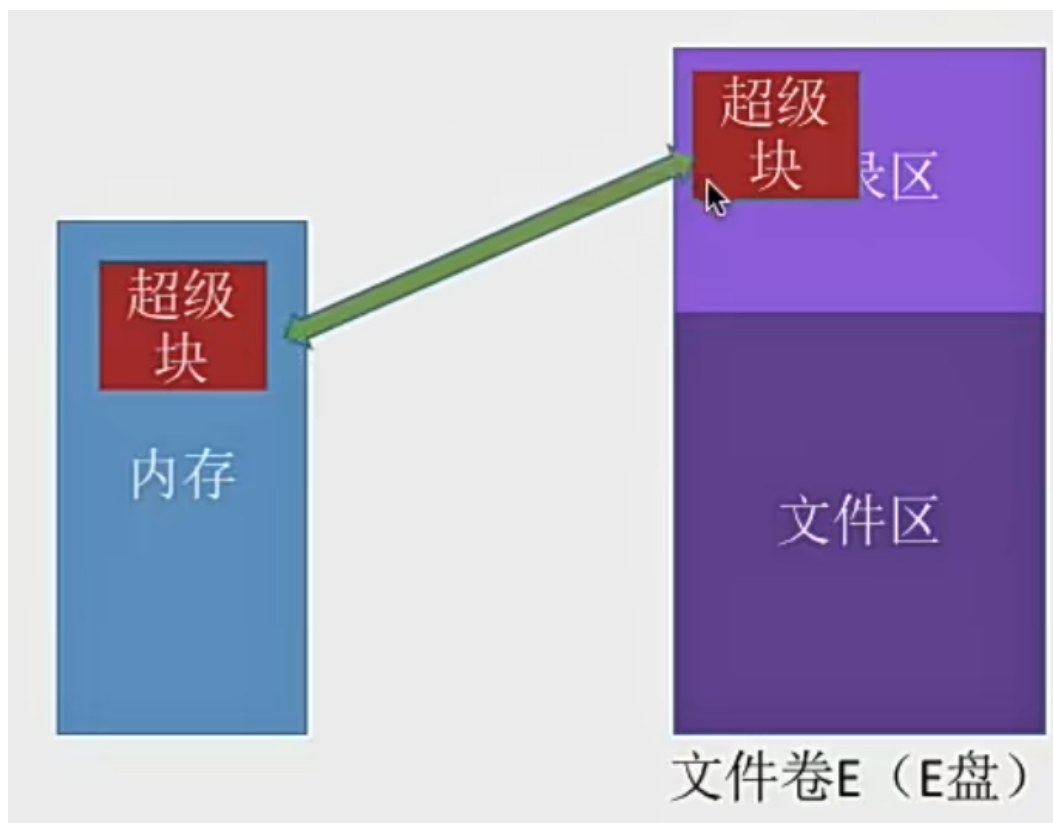
- 概念：
  - 每个**二进制位对应一个盘块**，“0”表盘块空闲，“1”代表盘块已分配；
  - 可以用 (字号，位号) 对应一个盘块号
- 计算：
  - (字号, 位号) = (i, j) 的二进制位对应的盘块号  $b = ni + j$
  - b 号盘块对应的字号  $i = b/n$ ，位号  $j = b \bmod (n)$
- 分配：
  - 若文件需要  $k$  个块：
    - 1. 顺序扫描位示图，找到  $k$  个相邻或不相邻的“0”；
    - 2. 根据字号、位号算出对应的盘块号，将相应盘块分配给文件；
    - 3. 将相应位设置为“1”；
- 回收：
  - 1. 根据回收的盘块号计算出对应的字号、位号；
  - 2. 将相应二进制位设为“0”
- 适用范围：
  - 连续分配、离散分配都适配；
- 注意：
  - 盘块号、字号、位号到底是从 0 开始还是从 1 开始
- 举例：
  - 在本例中，“0”表盘块空闲，“1”代表盘块已分配；
  - 位示图一般用连续的“字”来表示，如本例中个字的字长是 16 位，**字中的每一位对应一个盘块**；
  - 因此可以用 (字号，位号) 对应一个盘块号。当然有的题目中也描述为 (行号，列号)



## 成组链接法

**概念：**成组链接法

- 引入：
  - 空闲表法、空闲链表法不适用于大型文件系统，因为空闲表或空闲链表可能过大；
  - **UNIX** 系统中采用了成组链接法对磁盘空闲块进行管理；
- 概念：
  - 文件卷的目录区中**专门用一个磁盘块**作为“超级块”，当系统启动时需要将超级块读入内存，并且要保证内存与外存中的“超级块”数据一致；
  - 超级块类似于链头的作用，它永远保存了下一个分组的信息；
- 回收：
  - 假设每个分组最多为 **100** 个空闲块，此时第一个分组已有 **100** 个块，还要再回收一块；
  - 需要将超级块中的数据复制到新回收的块中，并修改超级块的内容，让新回收的块成为第一个分组；
- 超级块构成：
  - 1. 下一组空闲盘块数；
  - 2. 空闲块号，链接上各个空闲块；
- 图示：

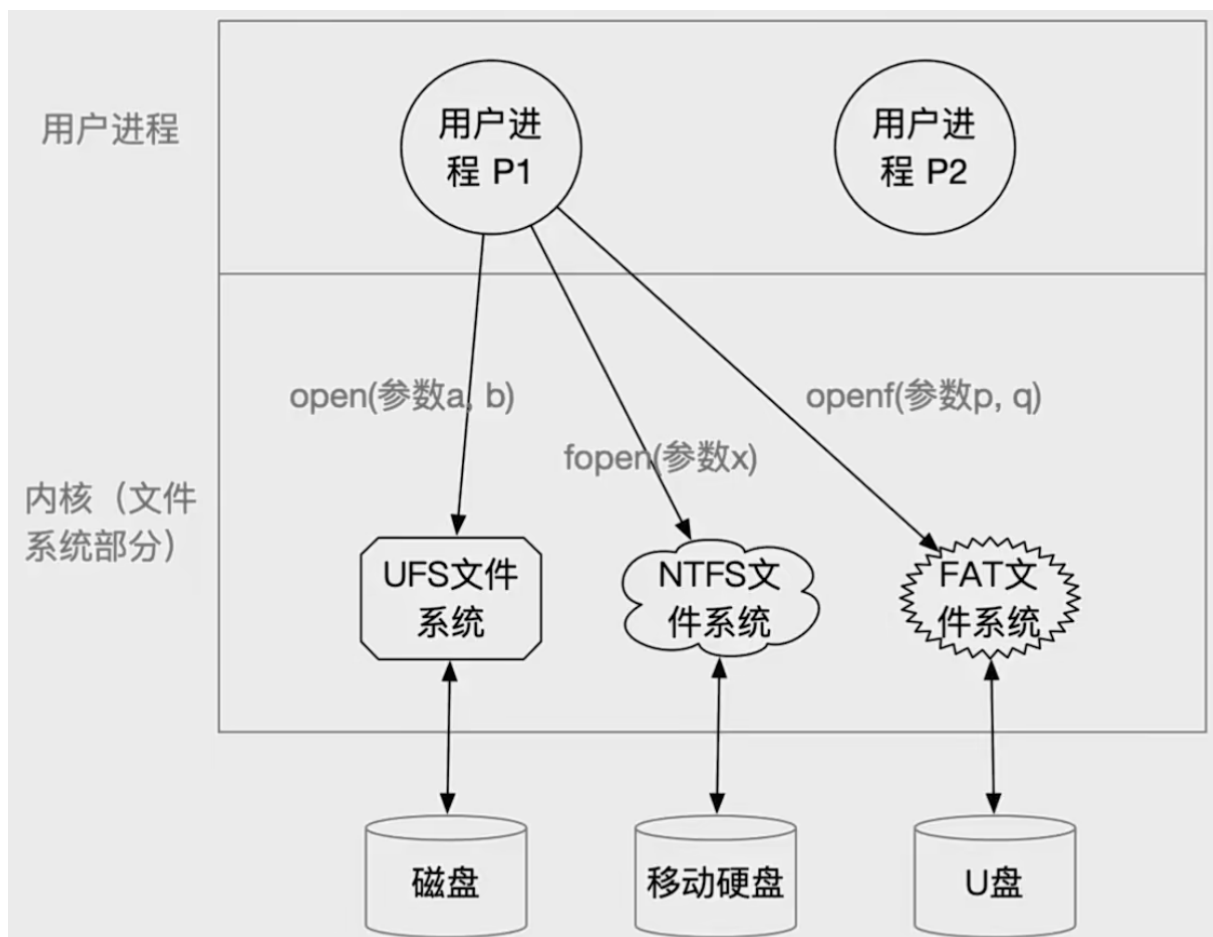


## 虚拟文件系统

### 普通文件系统

**概念:** 普通文件系统

- 概述:
  - 不同的硬件设备可能存在不同的文件系统, 因此其定义的各种接口可能不同;
  - 此时当用户想要使用不同的硬件中的文件系统打开文件时, 此时需要使用不同的函数代码;
  - 因此操作系统内核需要为上层用户进程提供统一标准的系统调用接口, 屏蔽底层具体文件系统的实现差异;
- 图示:



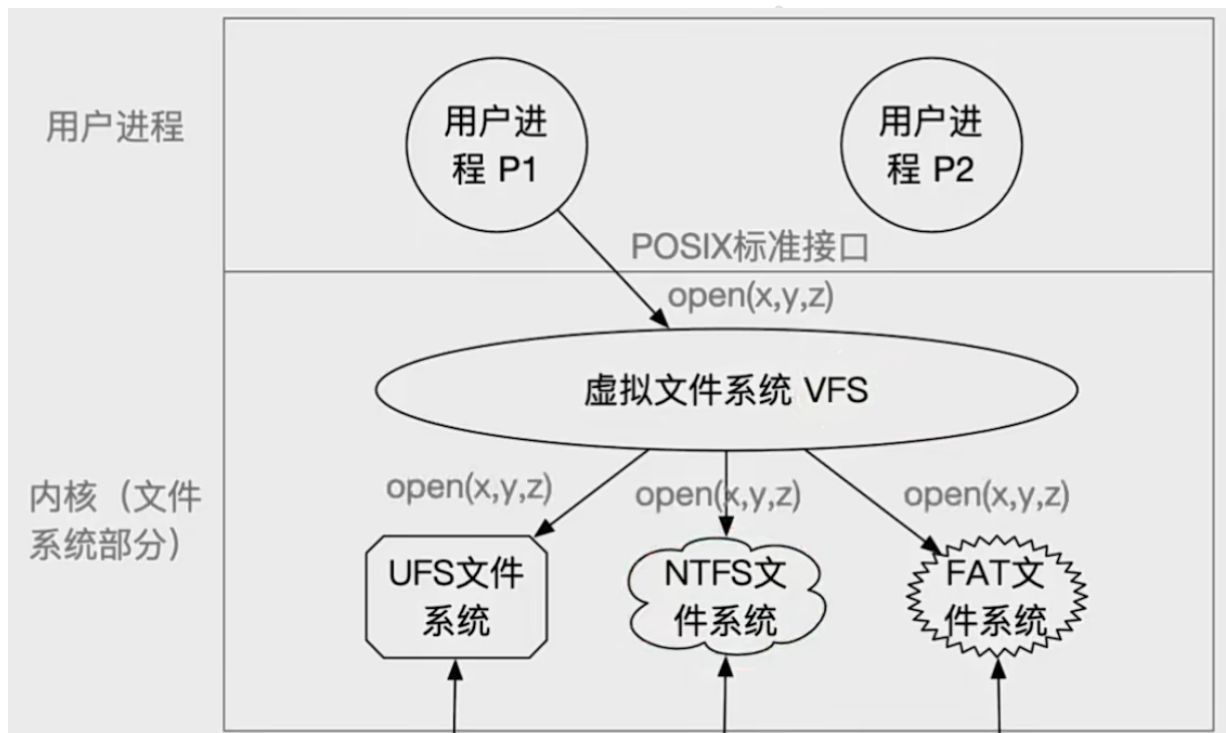
## 虚拟文件系统

**概念：**虚拟文件系统

- 概述：
  - 操作系统内核需要为上层用户进程提供统一标准的系统调用接口，屏蔽底层具体文件系统的实现差异；
- 要求：
  - **VFS** 要求下层的文件系统必须**实现某些规定的函数功能**；
    - 要求底层的文件系统必须支持如：`open/read/write` 的系统调用标准；
  - 一个新的文件系统想要在某操作系统上被使用，就必须满足该操作系统 **VFS** 的要求；
- 四种对象类型：
  - 超级块对象：
    - 表示一个已安装 (或称挂载) 的特定文件系统；
    - 超级块对象对应于**磁盘上特定扇区的文件系统超级块**，用于存储已安装文件系统的元信息；
    - 其操作方法包含一系列可在超级块对象上调用的操作函数；
      - 比如：分配 `inode`、销毁 `inode`、读 `inode`、写 `inode` 等；
  - 索引节点对象：
    - 表示一个特定的文件；
    - 索引节点和文件是一对一的关系，只有**当文件被访问时，才在内存中创建索引节点对象**；
    - 每个索引节点对象都会复制磁盘索引节点包含的一些数据；

- 索引节点对象还提供许多操作函数，如创建新索引节点、创建硬链接、创建新目录等；
- 目录项对象：
  - 表示一个特定的目录项；
  - 目录项对象是一个路径的组成部分，它包含：
    - 指向关联索引节点的指针；
    - 指向父目录和指向子目录的指针；
  - 不同于前面两个对象，目录项对象在磁盘上没有对应的数据结构，而是 VFS 在遍历路径的过程中，将它们逐个解析成目录项对象的；
- 文件对象：
  - 表示一个与进程相关的已打开文件；
  - 可以通过调用 `open()` 打开一个文件，通过调用 `close()` 关闭一个文件；
  - 文件对象和物理文件的关系类似于进程和程序的关系：
    - 文件对象仅是进程视角上代表已打开的文件，它反过来指向其索引节点；
  - 文件对象包含与该文件相关联的目录项对象，包含该文件的文件系统、文件指针等，还包含在该文件对象上的一系列操作函数；

• 图示：



**概念：**对 VFS 的改进

- 问题：
  - 不同的文件系统，表示文件数据结构各不相同打开文件后，其在内存中的表示就不同；
  - 对于虚拟文件系统，若来自 UFS 文件系统的：

|       |   |
|-------|---|
| 所有者   |   |
| 创建时间  |   |
| 总块数   | 1 |
| 共享计数  |   |
| 直接索引0 | 5 |
| 直接索引1 |   |
| 直接索引2 |   |
| 直接索引3 |   |
| 直接索引4 |   |
| 直接索引5 |   |
| 直接索引6 |   |
| 直接索引7 |   |
| 直接索引8 |   |
| 直接索引9 |   |
| 一级间接  |   |
| 二级间接  |   |
| 三级间接  |   |

- 来自 FAT 的：

|             |     |      |          |      |      |
|-------------|-----|------|----------|------|------|
| FAT文件系统的目录项 | 文件名 | 文件类型 | ...其他... | 文件大小 | 起始块号 |
|-------------|-----|------|----------|------|------|

- 改进：vnode 结点：

- 概述：

- 不管来自什么文件系统，相关的信息都存进 vnode 当中；
- 每打开一个文件，VFS 就在主存中新建一个 vnode，用统一的数据结构表示文件，无论该文件存储在哪个文件系统；
- 打开文件后，创建 vnode，并将文件信息复制到 vnode 中，vnode 的功能指针指向具体文件系统的函数功能；

- 函数功能指针：

- 函数功能指针存储了指向不同文件系统提供的函数功能；
- 比如当需要使用 FAT 中的 read 调用时，会访问 vnode 中的函数功能指针，找到指向 FAT 的函数功能中的 read 函数；



- 注意：
  - vnode 只存在于主存中，而 node 既会被调入主存，也会在外存中存储；

## 文件系统综合分析

### Unix 文件系统

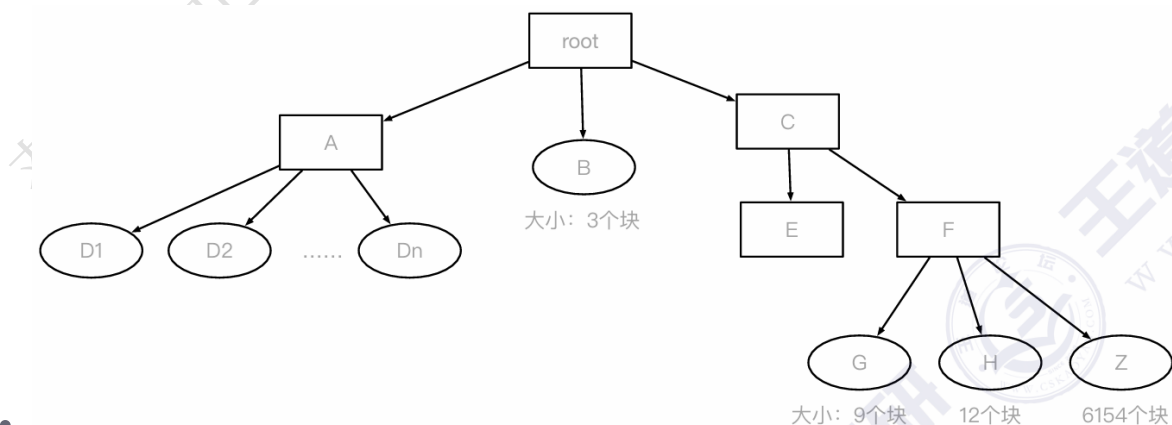
#### Unix 文件系统结构

**补充：**教程推荐

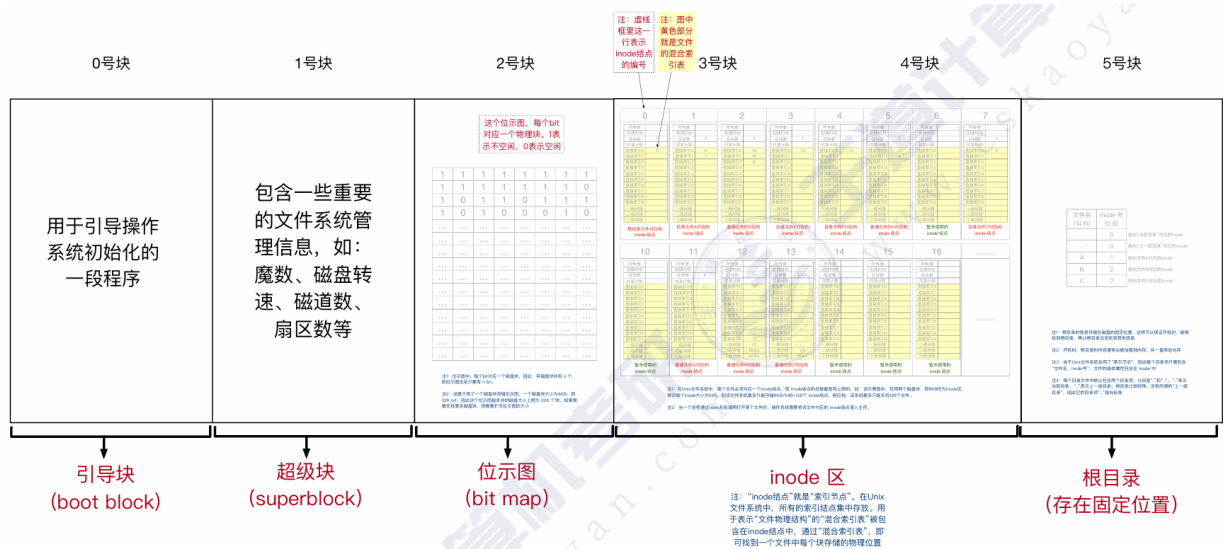
- [FAT 和 UNIX 文件系统 \(数据结构视角的文件系统; FAT 手册导读和目录树遍历\)](#) [南京大学 2022操作系统-P27] [哔哩哔哩 bilibili](#)

**概念：** Unix 文件系统基本结构

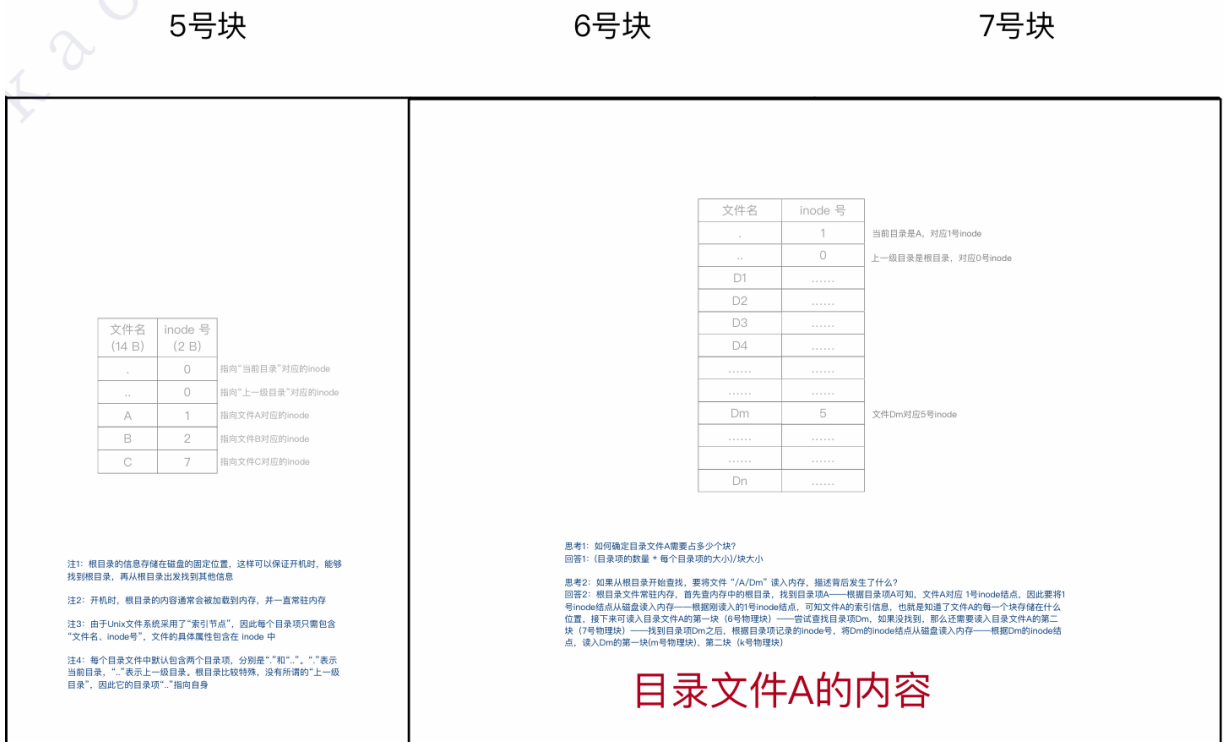
- 目录结构：
  - 图示：



- 注意：
  - 假设该磁盘中每个盘块的大小为 4 KB；
  - 下图为该文件系统中的目录结构示意，方形为目录文件，圆形为普通文件；
- 磁盘基本结构：



• 普通文件块结构：



概念：详细结构

- 位示图：
  - 图示：

这个位示图，每个bit  
对应一个物理块。1表  
示不空闲，0表示空闲

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
| 1   | 1   | 1   | 1   | 1   | 1   | 1   | 0   |
| 1   | 0   | 1   | 1   | 0   | 1   | 1   | 1   |
| 1   | 0   | 1   | 0   | 0   | 0   | 1   | 0   |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

- 注意：

- 位示图中，每个 bit 对应一个磁盘块，因此，若磁盘块共有 n 个，则位示图也至少要有 n bit；
- 这里只用了一个磁盘块存储位示图，一个磁盘块大小为 4 KB，即 32Kbit，因此这个位示图能支持的磁盘大小上限为 32K 个块。如果需要支持更多磁盘块，则需要扩充位示图的大小；

- inode 区

- 图示：

| 0                                                                                                                                            | 1                                                                                                                                            | 2                                                                                                                                            | 3                                                                                                                                            | 4                                                                                                                                            | 5                                                                                                                                            | 6                                                                                                                                            | 7                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 | 所有者<br>创建时间<br>总块数<br>共享计数<br>直接索引0<br>直接索引1<br>直接索引2<br>直接索引3<br>直接索引4<br>直接索引5<br>直接索引6<br>直接索引7<br>直接索引8<br>直接索引9<br>一级间接<br>二级间接<br>三级间接 |
| 根目录文件对应的<br>inode 结点                                                                                                                         | 目录文件A对应的<br>inode 结点                                                                                                                         | 普通文件B对应的<br>inode 结点                                                                                                                         | 目录文件E对应的<br>inode 结点                                                                                                                         | 目录文件F对应的<br>inode 结点                                                                                                                         | 普通文件Dm对应的<br>inode 结点                                                                                                                        | 暂未使用的<br>inode 结点                                                                                                                            | 目录文件C对应的<br>inode 结点                                                                                                                         |

• 注意：

- 黄色的部分为**混合索引表**；
- 虚线框里面表示 **inode** 编号；

• 概念：

- 在 **Unix** 文件系统中、每个文件必须对应一个 **inode** 结点，而 **inode** 结点的总数量是有上限的；
- 如：该示意图中、仅用两个磁盘块，即 **8KB** 作为 **inode** 区；
- 假设每个 **inode** 大小为 **64 B**，则该文件系统最多只能存储  $8\text{ KB}/64\text{ B}=128$  个 **inode** 结点，相应地，该系统最多只能支持 **128** 个文件；

• 根目录文件的块：

- 图示：

| 文件名<br>(14 B) | inode 号<br>(2 B) |
|---------------|------------------|
| .             | 0                |
| ..            | 0                |
| A             | 1                |
| B             | 2                |
| C             | 7                |

指向“当前目录”对应的inode

指向“上一级目录”对应的inode

指向文件A对应的inode

指向文件B对应的inode

指向文件C对应的inode

• 概念：

- 根目录的信息存储在**磁盘的固定位置**，这样可以保证开机时，能够找到根目录，再从根目录出发找到其他信息；
- 开机时，根目录的内容通常会被加载到内存，并**一直常驻内存**；
- 由于 **Unix** 文件系统采用了“索引节点”，因此每个目录项只需包含文件名、**inode** 号，文件的具体属性包含在 **inode** 中；
- 每个目录文件中默认包含两个目录项，分别是“.”和“..”
  - “.”表示当前目录；

- `..` 表示上一级目录。但根目录比较特殊，没有所谓的“上一级当前目录”，因此它的目录项“`..`”指向自身；

- 目录文件 A 的块：

- 图示：

| 文件名   | inode 号 |                     |
|-------|---------|---------------------|
| .     | 1       | 当前目录是A，对应1号inode    |
| ..    | 0       | 上一级目录是根目录，对应0号inode |
| D1    | .....   |                     |
| D2    | .....   |                     |
| D3    | .....   |                     |
| D4    | .....   |                     |
| ..... | .....   |                     |
| ..... | .....   |                     |
| Dm    | 5       | 文件Dm对应5号inode       |
| ..... | .....   |                     |
| ..... | .....   |                     |
| Dn    | .....   |                     |

- 思考：如何确定目录文件 A 需要占多少个块？

- 回答：  $(\text{目录项的数量} \times \text{每个目录项的大小}) / \text{块大小}$

- 思考：如果从根目录开始查找，要将文件“`/A/Dm`”读入内存，描述背后发生了什么？

- 回答：

- (1) 根目录文件常驻内存，首先查内存中的根目录，找到目录项 A；
- (2) 根据目录项 A 可知，文件 A 对应 1 号 inode 结点，因此要将 1 号 inode 结点从磁盘读入内存；

- 图示：目录文件 A 的 inode

|       |   |
|-------|---|
| 所有者   |   |
| 创建时间  |   |
| 总块数   | 2 |
| 共享计数  |   |
| 直接索引0 | 6 |
| 直接索引1 | 7 |
| 直接索引2 |   |
| 直接索引3 |   |
| 直接索引4 |   |
| 直接索引5 |   |
| 直接索引6 |   |
| 直接索引7 |   |
| 直接索引8 |   |
| 直接索引9 |   |
| 一级间接  |   |
| 二级间接  |   |
| 三级间接  |   |

目录文件A对应的  
inode 结点

- (3) 根据刚读入的 1 号 `inode` 结点，可知文件 A 的索引信息，也就是知道了文件 A 的每一个块存储在什么位置，接下来可读入目录文件 A 的第一块 (6 号物理块)；

| 文件名   | inode 号 |
|-------|---------|
| .     | 1       |
| ..    | 0       |
| D1    | .....   |
| D2    | .....   |
| D3    | .....   |
| D4    | .....   |
| ..... | .....   |
| ..... | .....   |
| Dm    | 5       |
| ..... | .....   |
| ..... | .....   |
| Dn    | .....   |

当前目录是A，对应1号inode

上一级目录是根目录，对应0号inode

文件Dm对应5号inode

- (4) 在第六块中、尝试查找目录项 `Dm`，如果没找到、那么还需要读入目录文件 A 的第二块 (7 号物理块)
- (5) 找到目录项 `Dm` 之后，根据目录项记录的 `inode` 号，将 `Dm` 的 `inode` 结点从磁盘读入内存；

|       |   |
|-------|---|
| 所有者   |   |
| 创建时间  |   |
| 总块数   | 2 |
| 共享计数  |   |
| 直接索引0 | 6 |
| 直接索引1 | 7 |
| 直接索引2 |   |
| 直接索引3 |   |
| 直接索引4 |   |
| 直接索引5 |   |
| 直接索引6 |   |
| 直接索引7 |   |
| 直接索引8 |   |
| 直接索引9 |   |
| 一级间接  |   |
| 二级间接  |   |
| 三级间接  |   |

### 目录文件A对应的 inode 结点

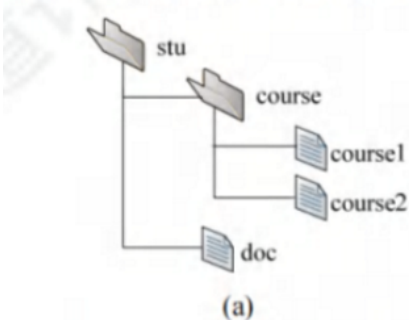
- (6) 根据 `Dm` 的 `inode` 结点，读入 `Dm` 的第一块(m 号物理块)、第二块(k 号物理块)

## Unix 文件系统分析

真题：2022 年

- 题目：

【2022 统考真题】某文件系统的磁盘块大小为 4 KB，目录项由文件名和索引节点号构成，每个索引节点占 256 字节，其中包含直接地址项 10 个，一级、二级和三级间接地址项各 1 个，每个地址项占 4 字节。该文件系统中子目录 stu 的结构如图(a)所示，stu 包含子目录 course 和文件 doc，course 子目录包含文件 course1 和 course2。各文件的文件名、索引节点号、占用磁盘块的块号如图(b)所示。请回答下列问题。



| 文件名     | 索引节点号 | 磁盘块号 |
|---------|-------|------|
| stu     | 1     | 10   |
| course  | 2     | 20   |
| course1 | 10    | 30   |
| course2 | 100   | 40   |
| doc     | 10    | x    |

(b)

- 1) 目录文件 stu 中每个目录项的内容是什么？
- 2) 文件 doc 占用的磁盘块的块号 x 的值是多少？
- 3) 若目录文件 course 的内容已在内存，则打开文件 course1 并将其读入内存，需要读几个磁盘块？说明理由。
- 4) 若文件 course2 的大小增长到 6 MB，则为了存取 course2 需要使用该文件索引节点的哪几级间接地址项？说明理由。

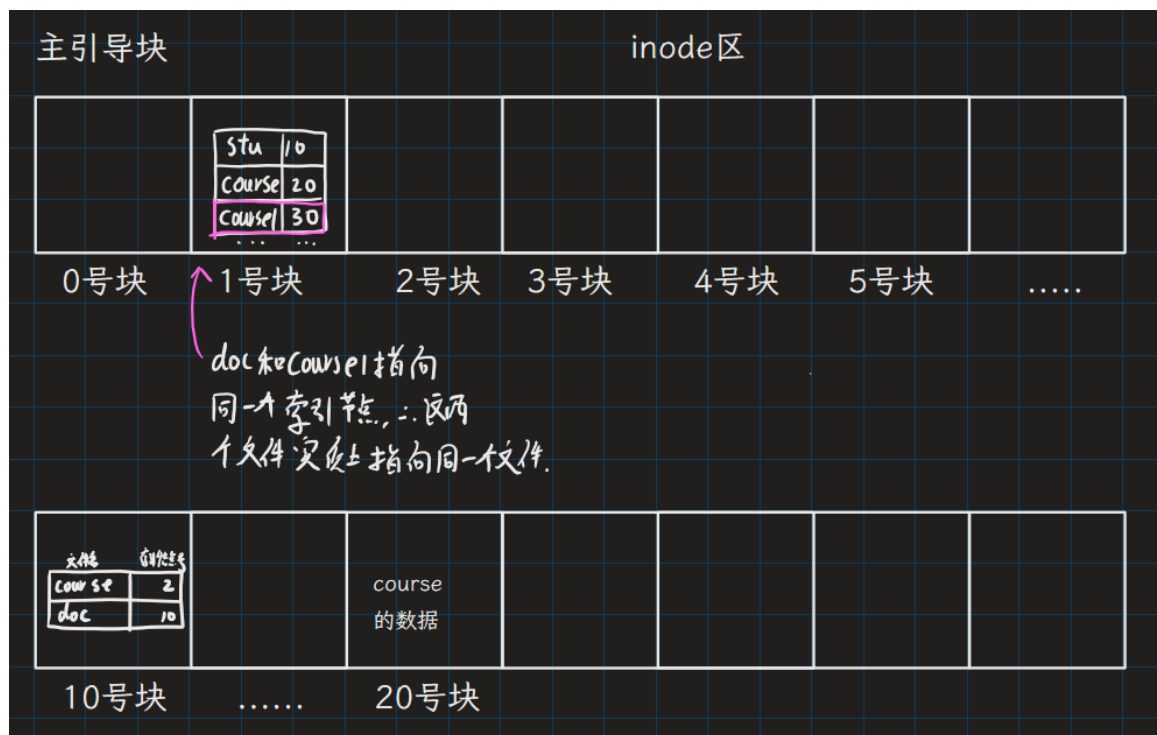
• 解析：

• 第一问：

| 文件名    | 索引节点号 |
|--------|-------|
| course | 2     |
| doc    | 10    |

• 第二问：

- 由于 doc 和 course1 的索引节点号相同，均为 10，因此两者共用一个索引节点，对应同一个文件（硬链接）；
- 由题目知 course1 的磁盘块号是 30，因此 doc 的磁盘块号也是 30，即 x 的值为 30；



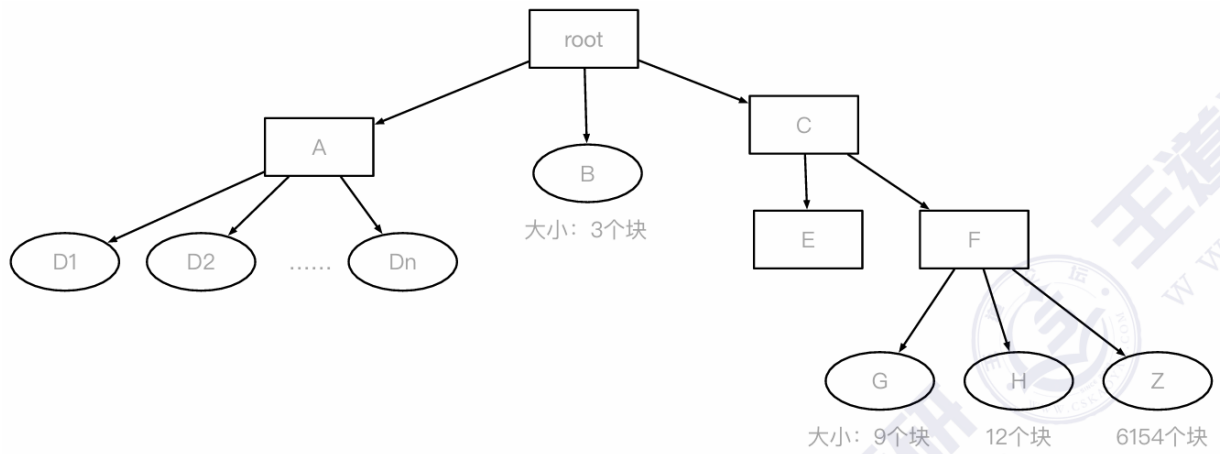
- 第三问：
  - 需要读 2 个磁盘块；
  - 先读其目录文件 `course`，由于目录文件 `course` 的内容已在内存中，即 `course1`、`course2` 对应的目录项已在内存中，根据 `course1` 对应的目录项可以知道其索引节点号；
  - 读 `course1` 的索引节点所在的磁盘块 1 号块（inode 区），根据 `course1` 的索引节点，可知该文件存储在 30 号磁盘块，因此可再读入 `course1` 的内容所在的磁盘块；
  - 因此：先在外存的 inode 区读入索引节点（一次访外存），然后再将 `course1` 所在内容读入（第二次访问外存）；
- 第四问：
  - 存取 `course 2` 需要使用索引节点的一级和二级间接地址项；
  - 6MB 大小的文件需要占用  $6\text{MB}/4\text{KB}=1536$  个磁盘块；
  - 直接地址项可以记录 10 个磁盘块号，一级间接地址块可以记录  $4\text{KB}/4\text{B}=1024$  个磁盘块号，二级间接地址块可以记录  $1024 \times 1024$  个磁盘块号，而  $10+1024 < 1536 < 10+1024+1024 \times 1024$ ；
  - 因此，6MB 大小的文件，需要使用一级间接地址项和二级间接地址项；

## FAT 文件系统

### FAT 文件系统结构

**概念：** FAT 文件系统基本结构

- 目录结构：

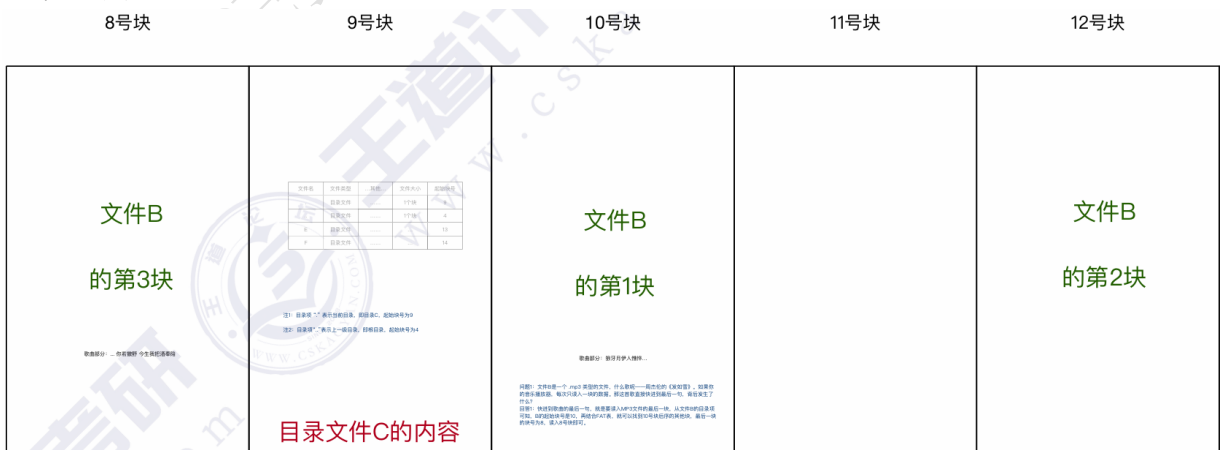


- 注意:
  - 假设该磁盘中每个盘块的大小为 4 KB;
  - 图为该文件系统中的目录结构示意, 方形为目录文件, 圆形为普通文件;

• 磁盘基本结构:



• 普通文件块结构:



概念: 详细结构

- FAT 表
- 图示:

| 当前块号 (隐含) | 下一块块号 |
|-----------|-------|
| 0         | -1    |
| 1         | 2     |
| 2         | 3     |
| 3         | -1    |
| 4         | -1    |
| 5         | 6     |
| 6         | 7     |
| 7         | -1    |
| 8         | -1    |
| 9         | -1    |
| 10        | 12    |
| 11        |       |
| 12        | 8     |
| 13        | -1    |
| 14        |       |
| 15        | ..... |
| 16        | -1    |
| .....     |       |
| 30        | 16    |
| 31        | 30    |
| .....     |       |

- 问题：FAT 表中，每个表项的大小对文件系统有什么影响？
  - 回答：
    - 1. 对文件系统支持的最大磁盘块数有影响，如：每个 FAT 表项 16bit，则最多只能表示  $2^{16}$  个块号
    - 2. 对 FAT 表的总大小有影响，每个表项的大小\*表项的数量=FAT 表的大小
- 问题：简述操作系统是如何使用 FAT 表的？
  - 回答：
    - 开机时，就会把整个 FAT 表读入内存，系统运行过程中，该表会一直常驻内存；
    - 也就是说，查 FAT 表通常不需要读磁盘，内存里本来就有这个表；
- 概念：FAT 下访问文件
  - 系统运行过程中，如果要访问某个文件，则首先要找到这个文件对应的目录项，从目录项中找到文件的起始块号；
  - 再根据起始块号，查 FAT 表，从而确认文件的后序的块都存放在哪里；

- 根目录：
  - 图示：

| 文件名 | 文件类型           | ...其他... | 文件大小           | 起始块号 |
|-----|----------------|----------|----------------|------|
| .   | 目录文件           | .....    | 1              | 4    |
| ..  | 目录文件           | .....    | 1              | 4    |
| A   | 目录文件           | .....    | 3              | 5    |
| B   | 普通文件<br>(.mp3) | .....    | 3个块<br>(2.5KB) | 10   |
| C   | 目录文件           | .....    | ...            | 9    |

- 注意：
  - 目录项 “.” 表示当前目录，即根目录，起始块号为 4
  - 问题：为什么根目录要存放在磁盘的固定位置？
    - 因为开机的时候，需要从根目录出发，找到操作系统初始化相关的各种文件；
    - 因此，将根目录存放在固定位置，那么开机时候取这个固定位置读入根目录的数据即可；

- 目录文件 A

- 图示：

| 文件名   | 文件类型        | ...其他... | 文件大小  | 起始块号  |
|-------|-------------|----------|-------|-------|
| .     | 目录文件        | ...其他... | 3个块   | 5     |
| ..    | 目录文件        | ...其他... | 1个块   | 4     |
| D1    | 普通文件 (.png) | .....    | ..... | ..... |
| D2    | 普通文件 (.jpg) | .....    | ..... | ..... |
| D3    | 普通文件 (.mp3) | .....    | ..... | ..... |
| D4    | 普通文件 (.txt) | .....    | ..... | ..... |
| ..... | .....       | .....    | ..... | ..... |
| ..... | .....       | .....    | ..... | ..... |
| Dm    | 普通文件        | .....    | 2个块   | m     |
| ..... | .....       | .....    | ..... | ..... |
| ..... | .....       | .....    | ..... | ..... |
| Dn    | 普通文件        | .....    | ..... | ..... |

- 概念：
  - 问题：如何确定目录文件 A 需要占多少个块？

- 回答:  $(\text{目录项的数量} \times \text{每个目录项的大小}) / \text{块大小}$
- 问题: 目录文件 A 必须占用连续的磁盘块吗?
  - 回答: 不用! 如果目录文件 A 继续变大, 还需要新的磁盘块, 那么通过查 FAT 表可以找到空闲磁盘块, 将其分配给文件 A 使用即可;
  - 当然, 本来文件 A 的最后一块是 7 号块, 如果给 A 再分配一个新的块, 就需要修改 FAT 表的 7 号表项, 指向新分配的块;
- 问题: 如果现在从根目录开始查找, 要将文件 “/A/Dm” 读入内存, 描述背后发生了什么?
  - 回答:
    1. 根目录常驻内存, 首先查内存中的根目录, 找到目录项 A;
    2. 根据目录项中记录的起始块号, 读入 A 的第一块 (5 号块);
    3. 尝试在 5 号块中找到 Dm 目录项:
      - 如果没找到, 那么通过查内存中的 FAT 表, 读入 A 的第二块 (6 号块), 尝试在 6 号块中找到 Dm 目录项;
      - 如果还没找到, 那么继续查 FAT 表, 并读入 A 的第三块 (7 号块), 尝试在 7 号块中找到 Dm 目录项;
    4. 找到 Dm 的目录项, 就可以确定 Dm 的起始块号为 m, 再结合 FAT 表就可以知道 m 号块之后是 k 号块, k 号块是文件 Dm 的最后一块。因此将 m、k 两个块读入内存即可;
  - 总结: 查询步骤
    - 访问内存根目录
    - -> 查内存中根目录中 A 目录项, 得到 A 的起始块号
      - 一次访存, 无磁盘 IO
    - -> 将目录 A 的第一块调入内存
      - 一次磁盘 IO, 一次写存
    - -> 查 Dm 文件的目录项, 得到 Dm 起始块号
      - 一次访存
    - -> 根据 FAT, 查 Dm 的后续块号
      - 一次访存, 因为 FAT 已经在内存中;

## FAT 文件系统分析

**真题:** 2016 年

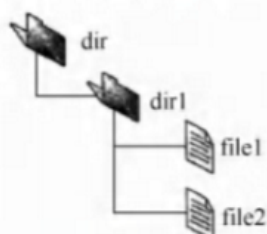
- 题目:

【2016 统考真题】某磁盘文件系统使用链接分配方式组织文件，簇大小为 4KB。目录文件的每个目录项包括文件名和文件的第一个簇号，其他簇号存放在文件分配表 FAT 中。

1) 假定目录树如下图所示，各文件占用的簇号及顺序如下表所示，其中 dir、dir1 是目录，file1、file2 是用户文件。请给出所有目录文件的内容。

2) 若 FAT 的每个表项仅存放簇号，占 2B，则 FAT 的最大长度为多少字节？该文件系统支持的文件长度最大是多少？

3) 系统通过目录文件和 FAT 实现对文件的按名存取，说明 file1 的 106, 108 两个簇号分别存放在 FAT 的哪个表项中。



| 文件名   | 簇号          |
|-------|-------------|
| dir   | 1           |
| dir1  | 48          |
| file1 | 100、106、108 |
| file2 | 200、201、202 |

4) 假设仅 FAT 和 dir 目录文件已读入内存，若需将文件 dir/dir1/file1 的第 5000 个字节读入内存，则要访问哪几个簇？

• 解析：

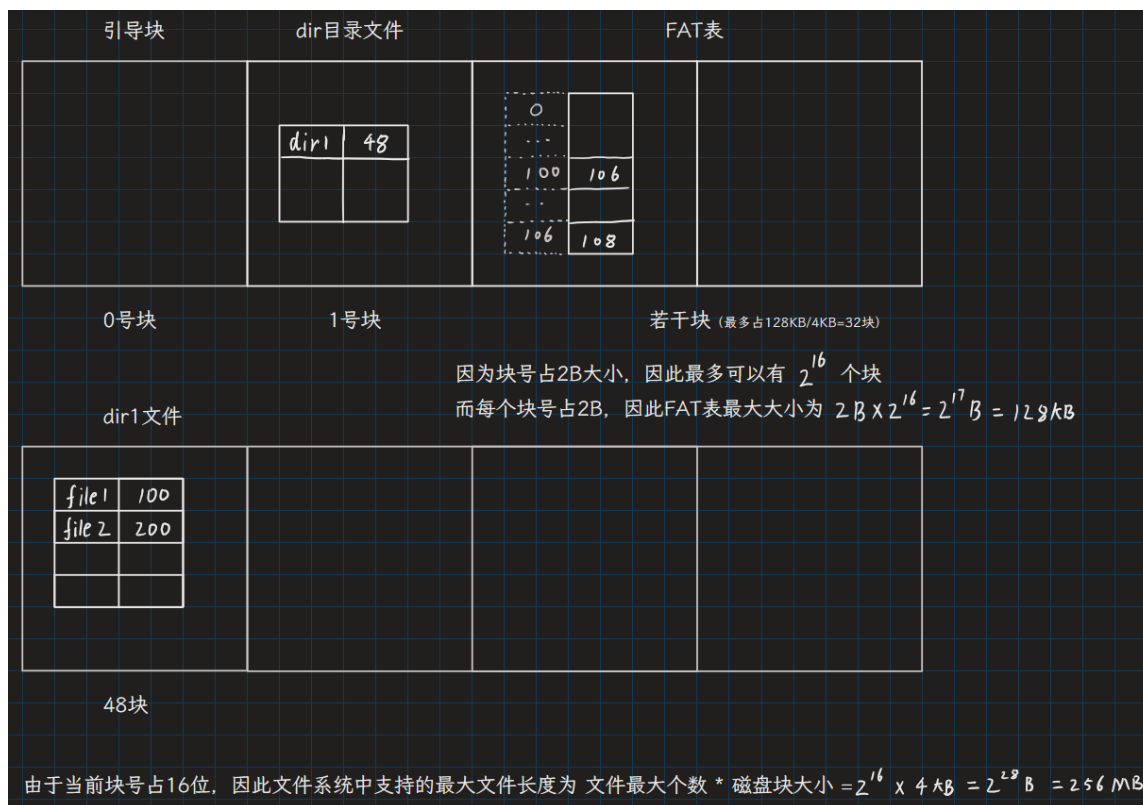
• 第一问：

注意区分以下概念：目录文件、目录项、目录树、用户文件、FAT表、FAT表项

| dir 目录文件 |    | dir1 目录文件 |     |
|----------|----|-----------|-----|
| 文件名      | 簇号 | 文件名       | 簇号  |
| dir1     | 48 | file1     | 100 |
|          |    | file2     | 200 |

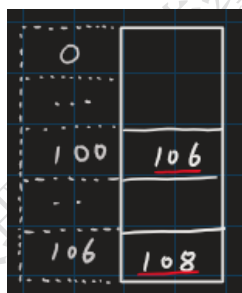
• 第二问：

- 问 FAT 的最大长度，首先需要知道当前文件系统运行的最多文件数量；
- 由于 FAT 表项占  $2B=16b$ ，因此最多可表示  $2^{16}$  个文件；
- 因此最多可以有  $2^{16}$  个表项，每个表项占  $2B$  大小，因此 FAT 最大长度为  $2^{16} * 2B$ ；
- 因为当前在 FAT 中最多可以有  $2^{16}$  个表项，而如果每个表项都链接同一个文件，则：
  - 文件系统中支持的最大文件长度为 文件最大个数 \* 磁盘块大小 = 256MB



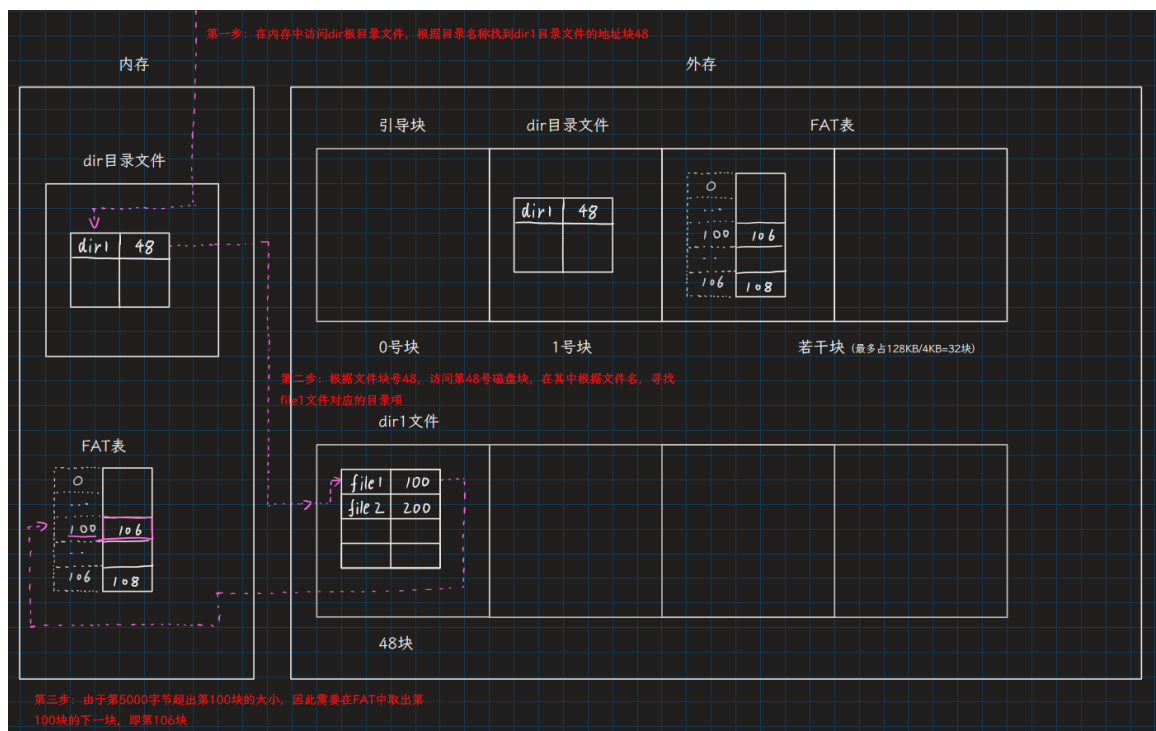
### 第三问:

- 系统使用 FAT 的流程:
- 开机时, 就会把整个 FAT 表读入内存, 系统运行过程中, 该表会一直常驻内存, 也就是说, 查 FAT 表不需要读磁盘, 内存里本来就有这个表;
- 系统运行过程中, 如果要访问某个文件, 则首先要找到这个文件对应的目录项, 从目录项中找到文件的起始块号, 再根据起始块号查 FAT 表, 从而确认文件的后序那些块存放在哪里;
- 因此若 file1 的 106 块文件, 存放在 FAT 的 100 块号表项中; 108 号文件块号存放在 106 号表项中;



### 第四问:

- 由于 FAT 和根目录 dir 都已经存入内存, 因此访问 dir 和 FAT 时不需要访问外存;



- 所以需要访问第48、第106块物理块

• 总结：

★ FAT最大长度 = 单个FAT表项大小 \* 最大文件数量

- 比如FAT表项为2B，即16位；
- 因为FAT表项有16位，因此最多可以有2的16次方个文件；
- 因此在FAT表中，最多有2的16次方的表项；
- 因此  $2B \times 2^{16} = 128KB$

★ 在FAT中，最大文件个数和FAT的表项大小有关，如果FAT表项为16位，则最多2的16次方个文件

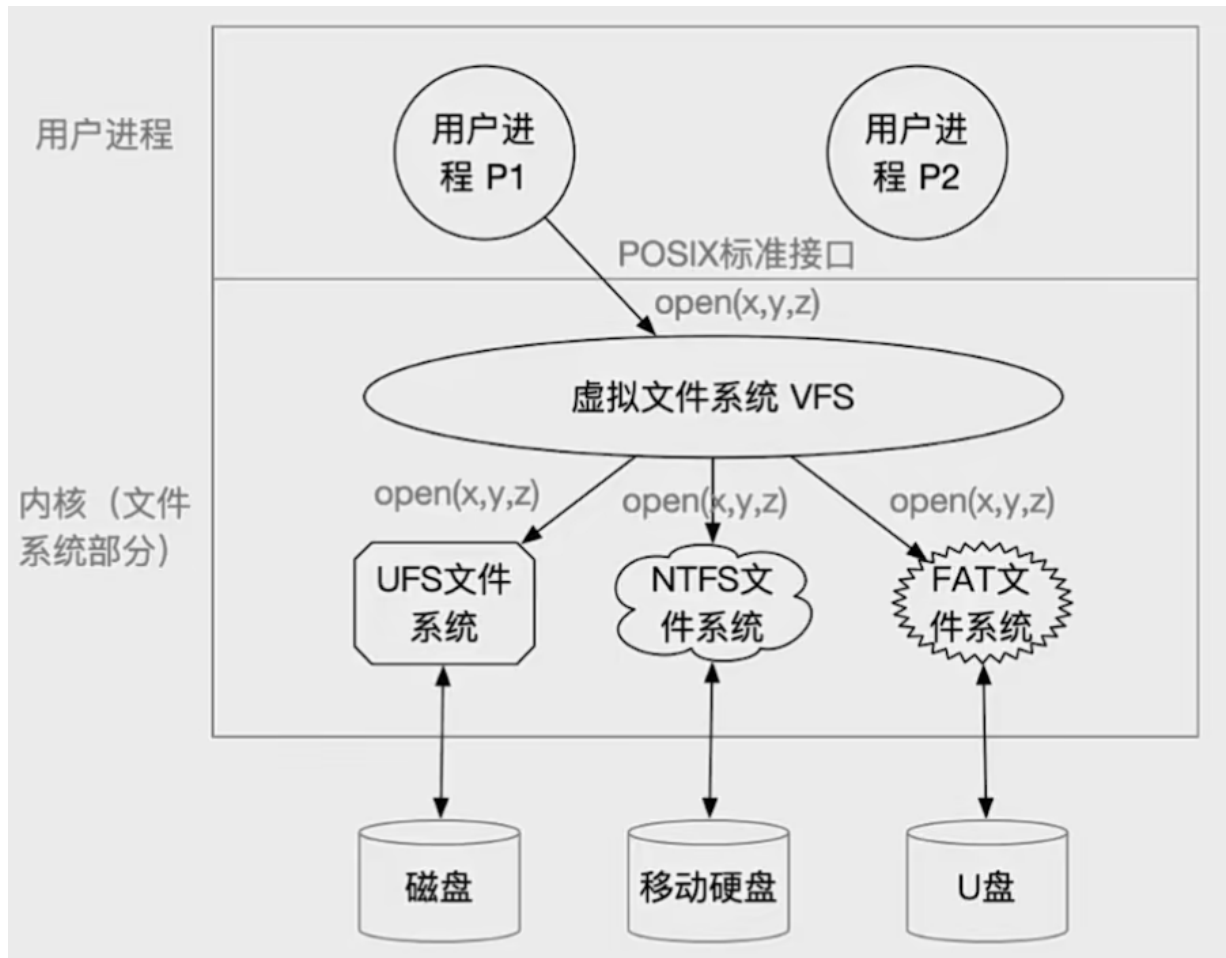
## 文件系统的挂载

**概念：**什么是挂载

- 概述：
  - 文件系统安装/装载 -> 如何将一个文件系统挂载到操作系统中；
- 实现：
  1. 在挂载表中注册：
    - 在 `VFS` 中注册新挂载的文件系统内存中的挂载表 (`mount table`)，其包含**每个文件系统的相关信息，包括文件系统类型、容量大小等**；
  2. 提供函数地址：
    - 新挂载的文件系统，要向 `VFS` 提供一个函数地址列表，如 `read`、`write` 等函数的地址；

- 3. 加到挂载点：
  - 将**新文件系统加到挂载点** (mount point), 也就是将新文件系统挂载在某个父目录下;

• 图示:



## Linux 中的文件系统

**引入:** Linux 中的文件系统

- 起源:
  - MINIX filesystem -> Extended Filesystem -> Second Extended Filesystem -> Ext3: 现在最为广泛使用的版本;
- 效率:
  - 1. Optimal Block Size - 最佳块大小 -> 块的大小会影响效率。加入当前大部分的文件平均长度都小于 1024 bit, 则将 Block 大小设置为 1024 字节将是合适的选择;
  - 2. 为分区选择合适的 inode 数量:

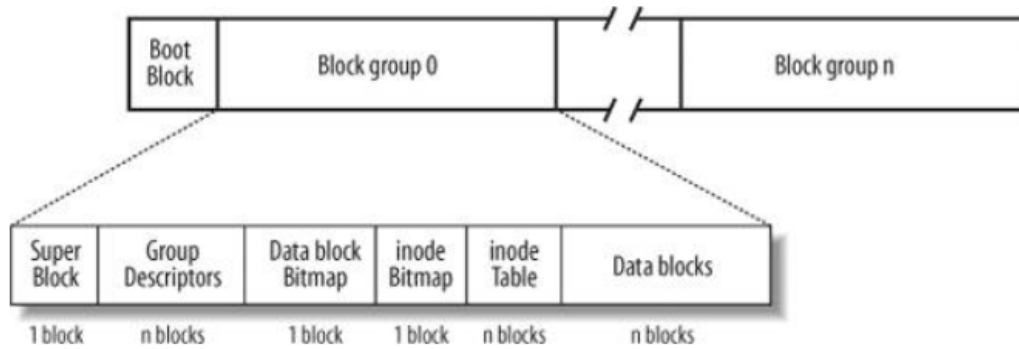
## Ext2 磁盘数据结构

**概念:** Ext2 磁盘数据结构

- 实现:
  - Ext2 文件系统, 磁盘上的第一个文件时 **Boot Block**, 随后的部分将会被分成若干个 **Block Group**;

- 所有的 **Block Group** 占据同样的大小，并且顺序的排列在磁盘中，因此内核可以简单的通过 **Index** 来定位每一个 **Block Group**；

• 图示：



## **Boot Block** 与内存碎片

• 概念：

- **Block Group** 减少了文件碎片，因为内核会尽量将文件的数据块放到同样大小的 **Block Group** 中；
- 如果一个块不拥有任何以下信息，则被认为是未被使用的；
- 所有的超级块以及组描述信息都会在每个 **Boot Block** 中存在一个副本，但在正常情况下，系统只会使用 **Block Group 0** 当中存储的超级块信息；
- 但是当 **block group 0** 数据发生错误时，此时系统将会指示 **e2fsck** 取找到之前产生的超级块以及组描述信息副本；

• 组成：

- 每一个 **Block Group** 具有以下信息：
  1. 文件系统的超级快的副本；
  2. 一组块描述符的副本；
  3. 数据块的 **bitmap**；
  4. **inode** 的 **bitmap**；
  5. **inode** 表；
  6. 若干个数据块；

• **Block Group** 的数量：

• 概念：

- **Block Group** 的数量取决于当前分区的大小以及每一个 **block** 的大小；
- 其中最主要的约束是 **bitmap** -> **bitmap** 用于确定当前 **Block Group** 中有哪些 **block** 是在被使用、有哪些是空闲的；

• 举例：

- 假设有 **32GB** 的 **Ext2** 分区，每个 **block** 的大小是 **4kb**
- 每一个 **4kb** 的 **bitmap** 描述了 **32k** 个数据块 ->  $32000 * 4kb = 128MB$
- 所以当前 **Block Group** 的数量 ->  $32GB / 128MB = 256$  -> 可以用 **256** 个 **Block Group**；