CH9

文件系统——程序员角度

文件

需要的原因:

1. 虽然内存速度快，但是内存中的数据是易失性的。而文件中的数据是持续性的，在进程终止后还能继续存在
2. 可以作为一个共享对象被进程同时访问

定义:文件是操作系统提供的信息储存的统一的逻辑视图。

用户视角:逻辑辅助存储的最小分配单元

OS视角:文件是一个逻辑储存单元，是一种抽象数据类型

分类

type:可执行文件、对象、源代码、文本、多媒体等

文件属性

ATTR	INFO
name	文件名，人可以看得懂的
identifier	一个独特的标签
type	类型(可执行、文本等)
location	一个指针指向设备和设备上该文件的位置
size	文件大小
time/date	最后使用、创建、最后修改
protection	权限(rwx)

文件属性是因文件系统而异

权限问题:三位(owner、group、others)，采用比特标记权限RWX,如775表示所有者和组内成员可读写执行、其他人只能读和执行

name的相关问题

如:/home/os/test.txt

/home/os/test.txt为路径名(全机唯一，内核会把路径名分割去为一系列的数据地址，因此必然是唯一的)

test.txt为文件名(不唯一，仅在该文件夹唯一)

操作

1. create

   分配空间、在目录中分配条目

2. write

   修改文件名、文件内容(写指针)

3. read

   读文件名、文件内容(读指针)

4. reposition

   重定向文件指针、需要随机访问

5. delete

   释放空间、删除目录项

6. truncate

   截断、仅仅保留属性

file open

```
FILE*fopen(const chat*filename,const char*mode)
```

会返回一个FILE*,file是一个在stdio.h中定义的结构

fopen()会为FILE结构开辟一片内存(堆分配)

文件描述符

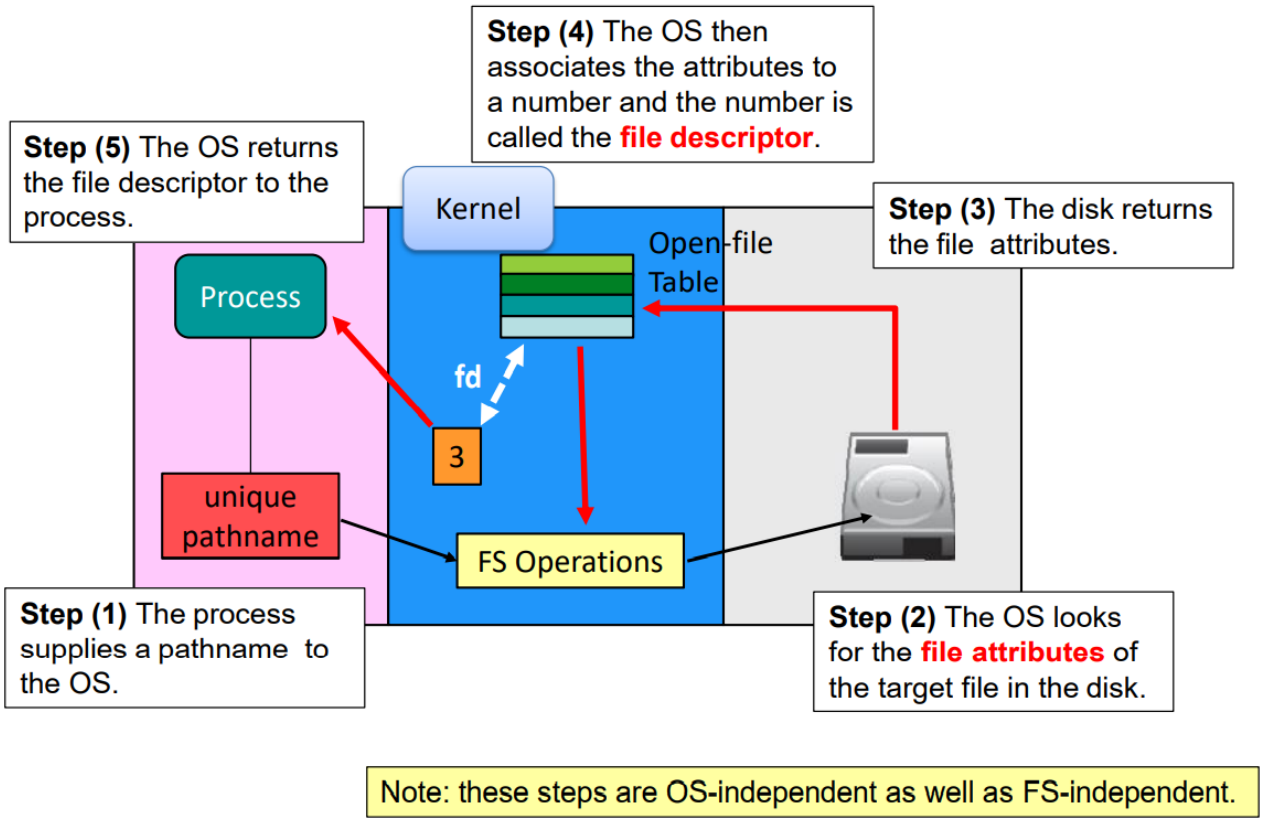
FILE	FD
stdin	0
stdout	1
stderr	2
other	>2

open file table

原因:很多文件系统操作要查找目录去定位文件，浪费性能

定义:当open()调用后，OS会维护一个包含所有打开文件的表(每个进程和系统范围的表)。当文件不再使用、调用close()时，就会被关闭

打开文件的详情



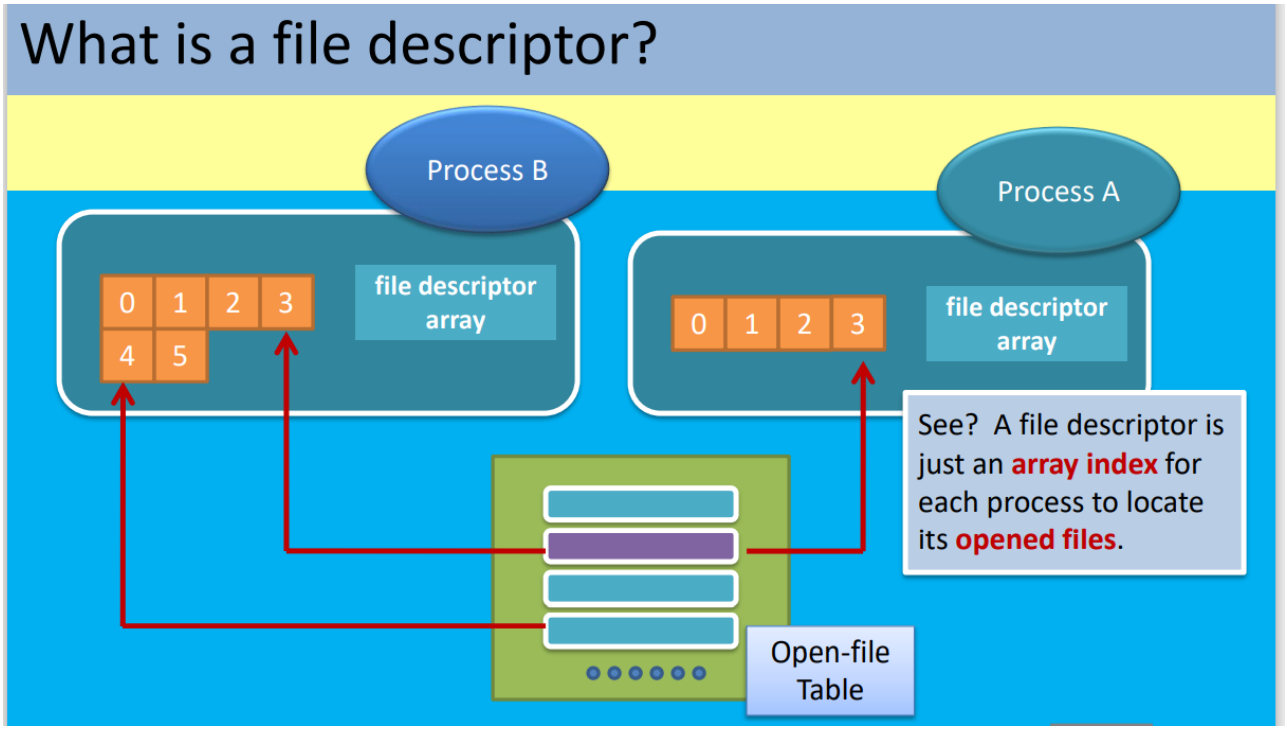
1. 进程提供路径名给OS
2. 通过文件系统服务、OS查找文件属性
3. 硬盘返回文件属性
4. 文件把文件属性和一个数字联系起来，即文件描述符
5. OS把文件描述符返回

因此打开文件仅仅包含路径名和文件属性、没有文件内容

file read

1. 进程提供文件描述符给OS
2. OS利用文件描述符读取文件属性定位所需的数据
3. 硬盘返回数据给内核、cache到缓冲区
4. OS把数据填到缓冲区返回给进程

文件描述符



不同进程的文件描述符结构一致

read()和write()

read():

1. 检查是否到达文件的尽头(offset>filesize)
2. 读取相应数据
3. 文件数据保存在内核cache
4. 内核cache把数据写到buffer返回给进程

write():

1. 把数据写到缓冲区
2. 判断是否修改数据长度，如果是，就修改文件大小
3. 系统调用结束并返回
4. 不时地把缓冲区的数据写到硬盘

目录项

一个目录项就是一个文件

利用目录项定位文件

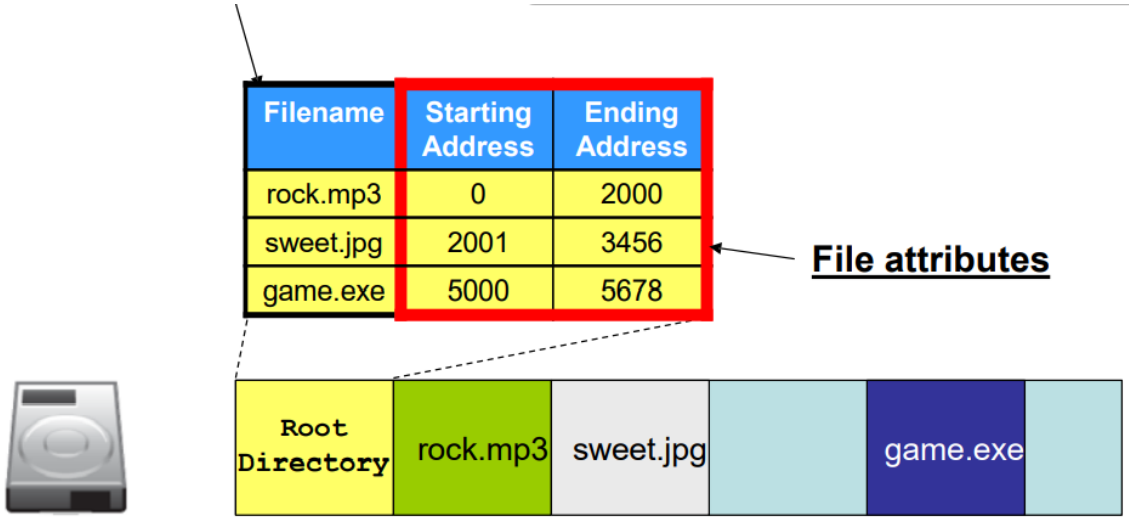
1. 进程把/bin/ls提供给OS
2. OS分割地址并得到“/”，传递给根目录
3. 硬盘把根目录的文件返回
4. OS在目录文件查找"bin"
5. 找到后，在/bin查找在其中的文件

文件系统设计

存数据的几个规则:

1. 可检索性
2. 高效性
3. 释放占用的空间

v1.0



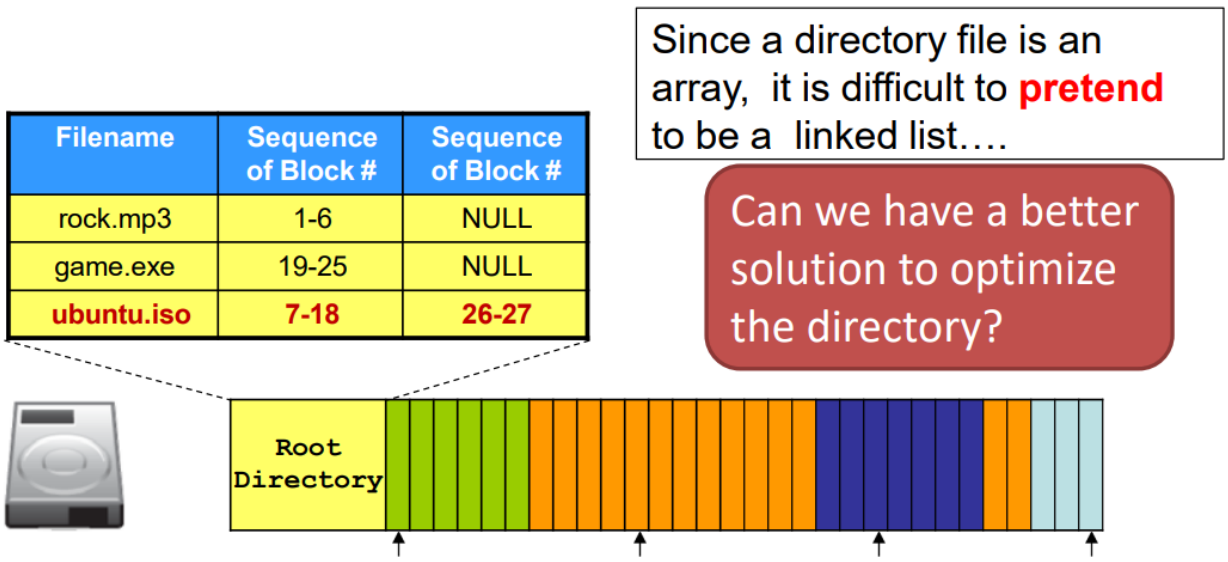
书本式储存

优点:组织简单、删除十分方便

缺点:无法应对文件增大、内部碎片化很严重、大文件存储麻烦

例子:ISO9660

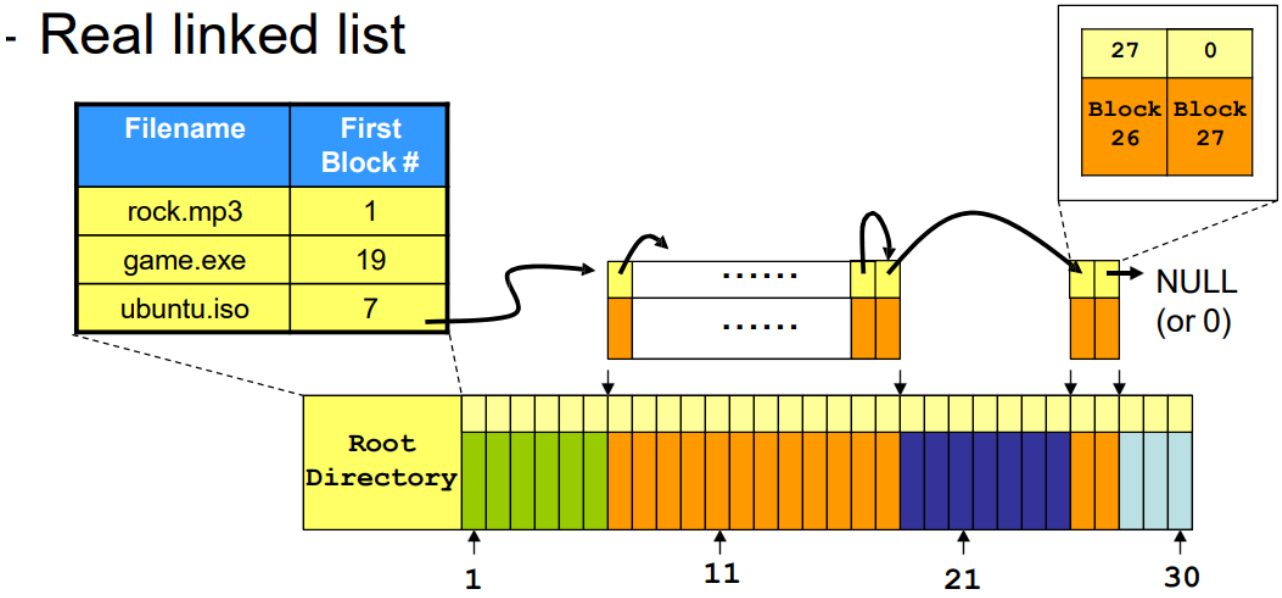
V2.x



分割加链表

问题:目录项很复杂

解决:从数据项中取4字节做链表



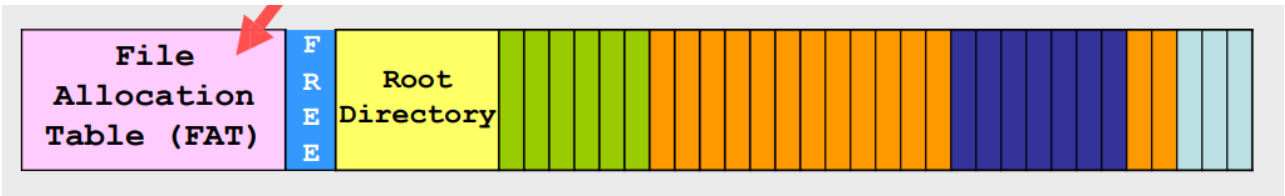
分割+链表

优点:外部碎片化得到缓解、文件大小改变很自由、空闲块管理容易实现

缺点:随机访问性能差，访问某个块之前要先遍历。

内部碎片化

v2.2 FAT



区别:链表由FAT表统一管理

查找文件详解

1. 找到文件的第一个块
2. 在FAT表中查找下一个块
    a. 直到下一个块为0(表示这是最后一个了)

随机查找性能：

Block #	1	…	6	7	…	18	19	…	25	26	27	28	29	30
Next Block #	2	…	0	8	…	26	20	…	0	27	0	29	30	0

FAT表会被cache进内核，加速随机访问性能

FAT分类:

文件储存的最小单元——簇

File System	FAT12	FAT16	FAT32
Cluster address length	12 bits	16 bits	32 bits (28?)
Number of clusters	4K	64K	256M

优点:各方面性能都很优秀(需要cache)，随机访问优秀

缺点:时间换空间，部分缓存很难

小结

1.0:连续分配

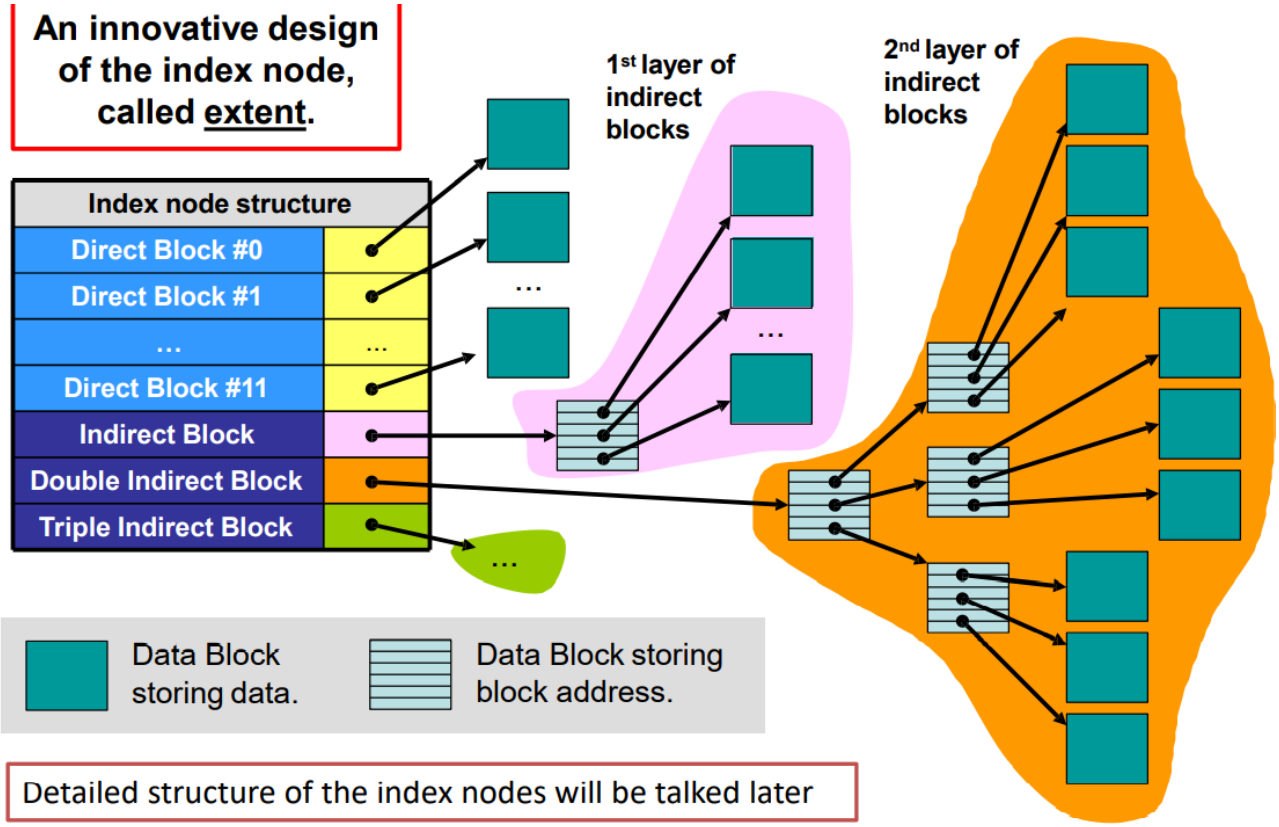
2.0:块式、根目录很复杂

2.1:块式+链表，随机访问性能差

2.2:FAT,需要cache

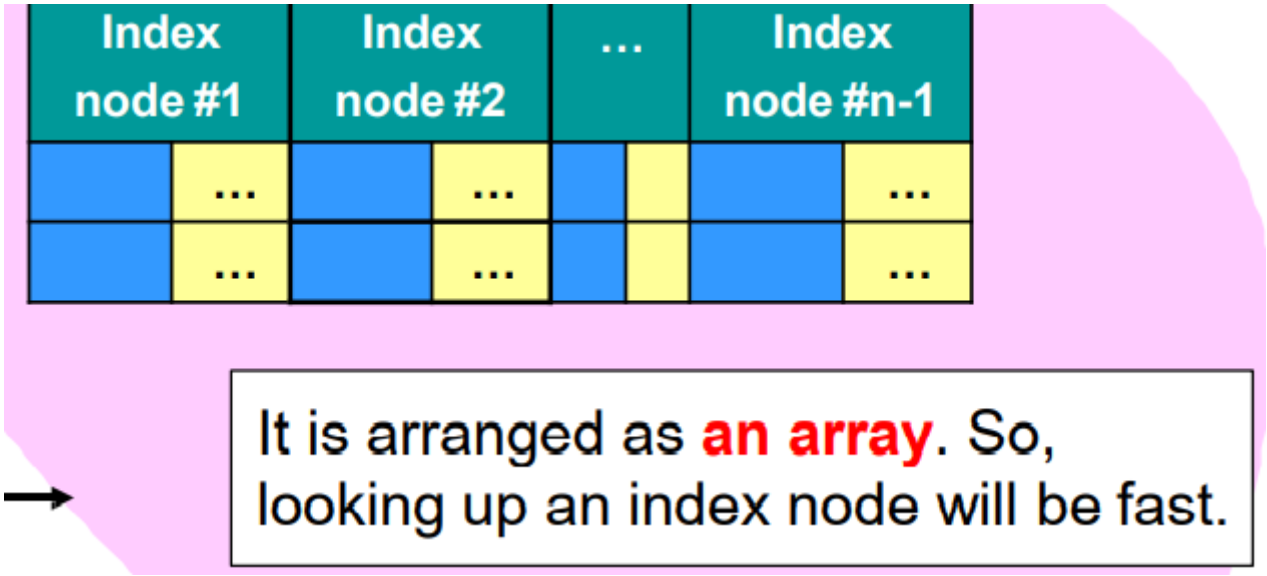
v3.0 I-node

采取多级映射方式



direct block储存的是数据块地址，indirect block储存的是地址块的地址,假定一个块大小为2^x

$$size_1 = 12 \times 2^x$$
$$size_2 = 2^{x-2} \times 2^x$$
$$size_3 = 2^{x-2} \times 2^{x-2} \times 2^x$$
$$totalsieze = size_1 + size_2 + size_3 = 2^{4x-6}$$

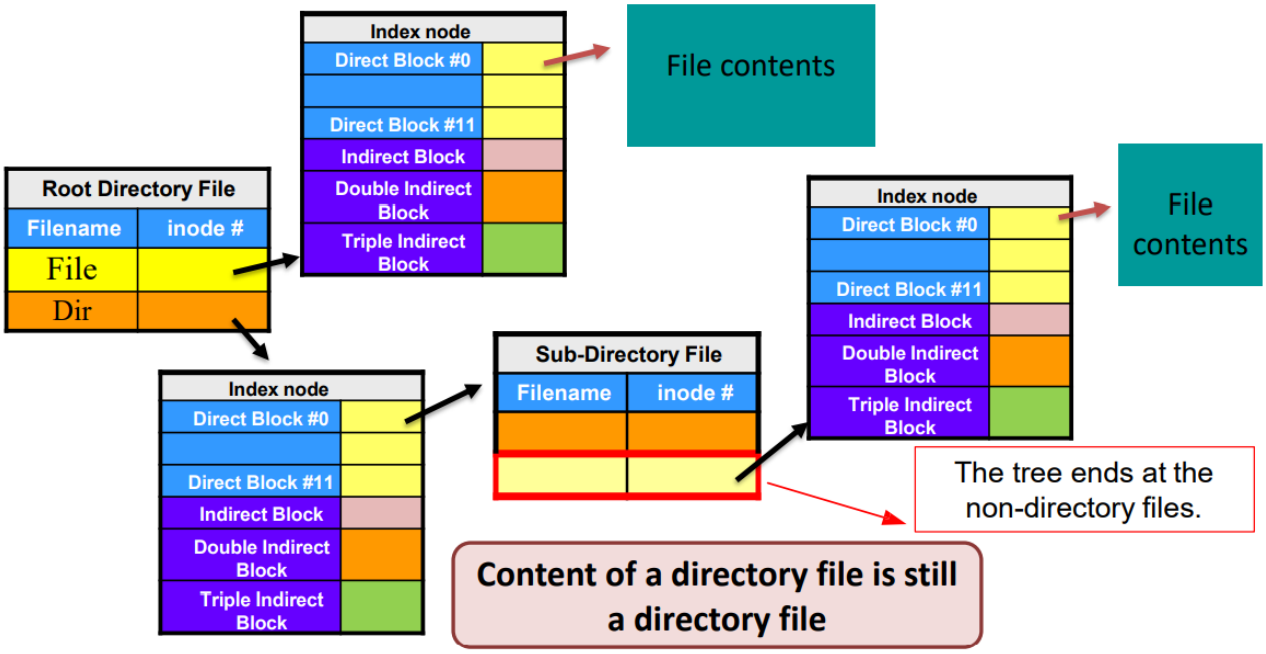


index node以数组形式组织,查找速度很快

根目录和子目录

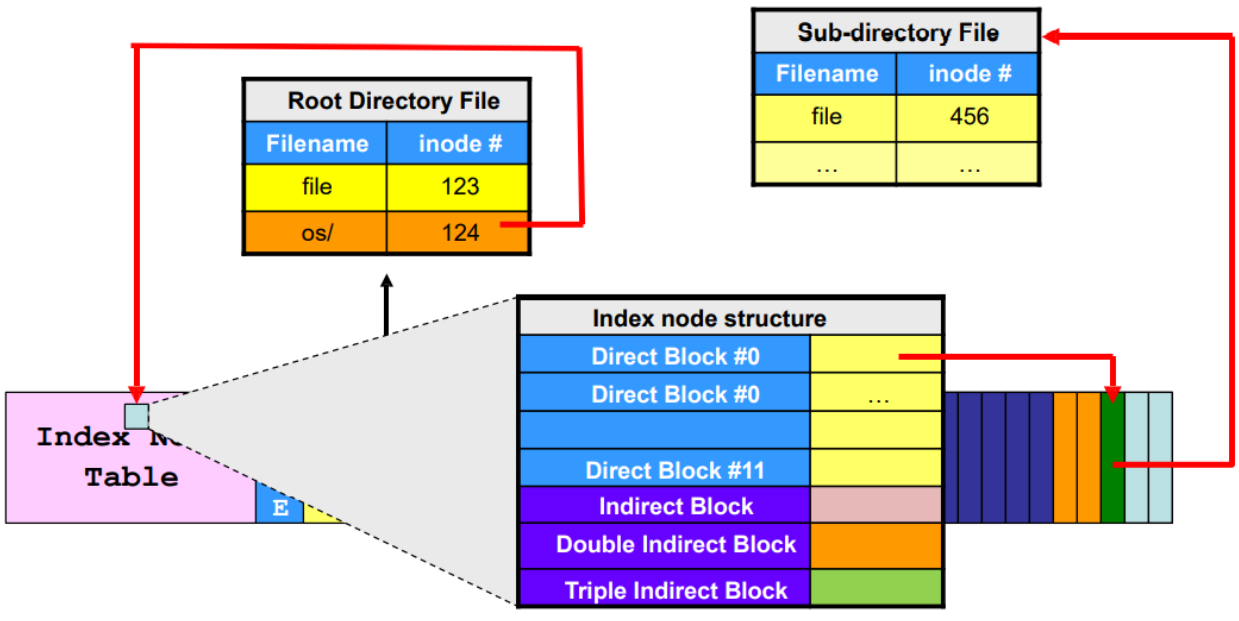
子目录在FS中也被当成文件，也有inode和FAT项

遍历目录结构



1. 对于一个目录，从block获取其中文件的iNode信息。
2. 利用iNode信息进入到下级目录，搜索想要的文件
3. 如此递归下去直到找到想要的文件

例子:查找/os/file



1. 先从根目录找到os/文件夹的iNode

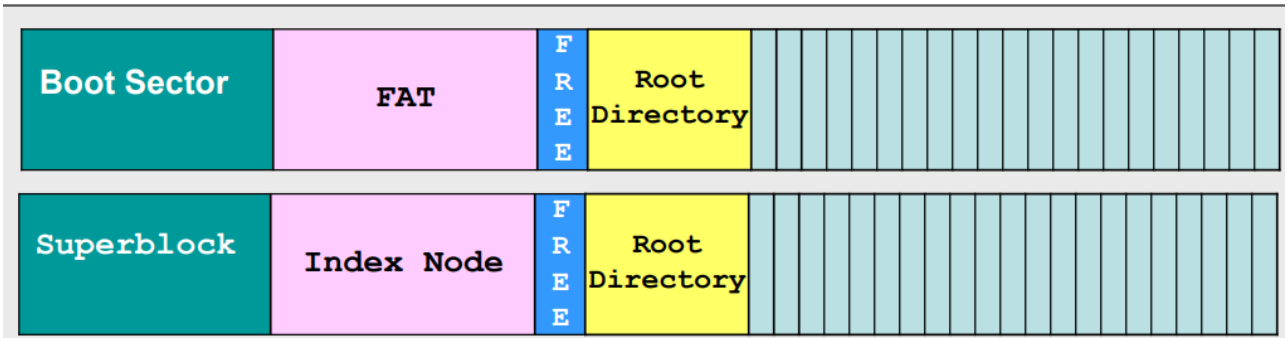
2. 从iNode表中找到os/文件夹的内容
3. 遍历其中的内容，找到file对应的iNode
4. 回到iNode表中获取到其对应的相关信息

文件系统信息和分割

硬盘中会储存相应的文件系统信息，如：

1. 块的大小
2. 已分配块的数目
3. 空闲块的数目
4. 根目录的位置
5. 文件分配信息的位置
6. 文件分配信息的大小

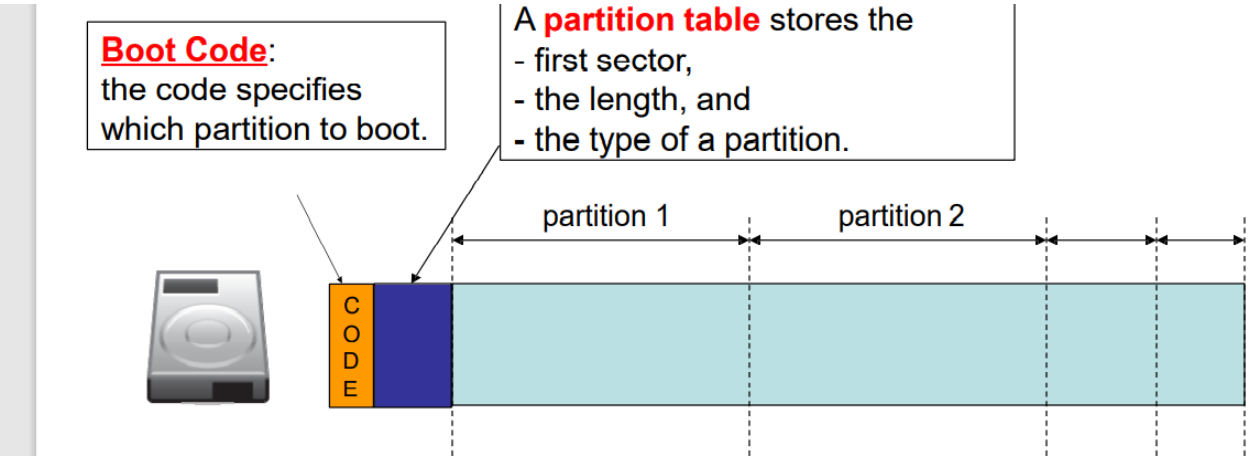
不能储存在操作系统，因为这是因硬件而异的



分为启动扇区和超级块两种

分割

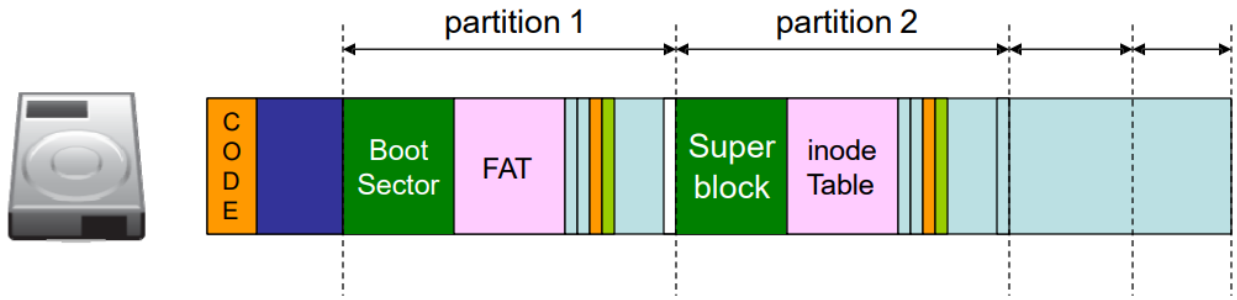
原因:大文件系统会带来很大的FAT因此需要分割



分割后还要有启动代码

优点：

1. 性能更佳
2. 多系统
3. 数据管理



最终形态