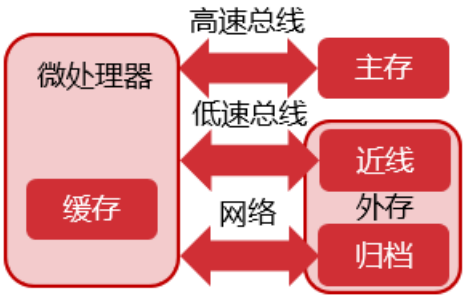
6.1空间的协调—外存储器（外存管理）

一.外存的特点

外存和内存的区别?



项目	外存	内存
总线	低速设备总线	高速内存总线
访问方式	系统调用	访存指令
距离CPU	一般很远	总是很近
容量	一般很大	总是很小
访问粒度	一般是块	总是按字节
连续访问带宽	一般很慢	总是极快
随机访问延迟	一般极慢	总是很快

外存——**最本质特征**就是**无法通过访存指令直接访问**，其他的特征都是辅助特征，，一个存储器是否是外存，不取决于存储器的物理属性，而是它在计算机架构中的访问属性

外存一般是只能**按块访问**、**不能高效随机访问的**。这是因为它们往往需要将容量做得很大，为缩减成本，在研发时就牺牲了其随机访问的能力。同时，很多资料需要永久保存，因此外存往往还是非易失的。这进一步牺牲了外存的访问速度和延迟， 使其性能全面落后于内存。

外存的特点:

1. 可靠性

   相较内存而言，外存需要跨上电周期**长时间储存数据**，因此其可靠性更加重要。很多外存技术为了容量极大地牺牲了原生物理存储单元的可靠性，因此需要**软件纠错措施**加以补强。

2. 保持时间

   外存能够在机器掉电后保持内容不变，但这种能力并非无限的。对于那些储存电荷的外存，一旦失去电力，其**数据便开始退化**，**在一段时间后便无法读写**。其它原理的外存往往也会由于介质的**逐渐劣化而影响数据的完整性**。

   准确地讲，在随时间劣化的是可靠性，保持时间仅仅是可靠性劣化速度的一个体现。当存储单元的可靠性恶化到无法靠软件纠错来补救，保持时间也就到头了。

   保持时间和可靠性在实践中受非常多因素影响，包括访问的累计次数、环境温度、介质工作模式等等。

3. 可擦写性

   部分外存介质如只读光盘等是不能够反复擦写的。它们只能接受一次写入，而后其内容便不可更改。当然，也存在可以反复擦写的光盘。常见的外存如硬盘等都是可以反复擦写的

4. 可覆写性

部分外存介质可以反复擦写，但是却**无法直接在已经写有内容的介质上直接覆盖写入新内容。**要复用已经写入过的介质，必须要执行**擦除**操作，该操作会清空一整片介质上的旧内容，然后这片介质才能接受新的写入。

5. 访问粒度

   外存的读（Read）、写（Write/Program）和擦除（Erase）粒度是相互独立的三个参数，它们都可以是字节或（大小不等的）块。比如，完全可能存在某种外存介质，它可以接受以字节读取，但必须按块写入，而擦除时则只接受全盘擦除。

6. 访问寿命（耐久性）

   外存的读、写和擦除寿命也是相互独立的三个参数。它们都标志着对其存储单元进行多少次相应操作后存储单元即告报废。绝大多数外存介质的读寿命都是无限长的，但其写寿命和擦除**（耐久性）**寿命则是有限的。一次写入/擦除操作称为一个循环，又称一次 P/E。我们经常用P/E数来衡量外存介质的寿命。

7. 访问延迟

   因外存介质的物理组织较为复杂，其**不同地址存储单元的访问延迟差别很大**，且该延迟与访问顺序可能存在很大关系。此外，读、写和擦除的延迟可以不同，而且差距还可以很大。

外存的内存化

随着技术发展，外存技术逐渐摆脱了不能随机访问、无法覆写、 耐久性差的传统印象。以FRAM和PCRAM为代表的外存技术逐渐可以在各项性能上与内存相媲美，同时拥有近乎无限的P/E数。 因此，这些技术逐渐衍生出可以直接插入内存插槽的形式，并可以当作系统内存使用。

内存的外存化

在DRAM上加上后备电池或者闪存，它就可以持久保存内容，此时称为NV（Non-Volatile）DRAM；再将它挂载到外存总线上，它就可以充当快速的外存。在SSD出现以前，这是非常常用的实现高速外存的手段。直到今天，这种手段仍在数据中心和加速器等场合有用武之地。

二.数据和文件系统

2.1文件的概念

分类：

1. 文本文件——由字符组成的文件，可以用文本编辑器打开
2. 二进制文件——由无约束二进制字节流组成的文件，用文本编辑器打开会乱码

定义：

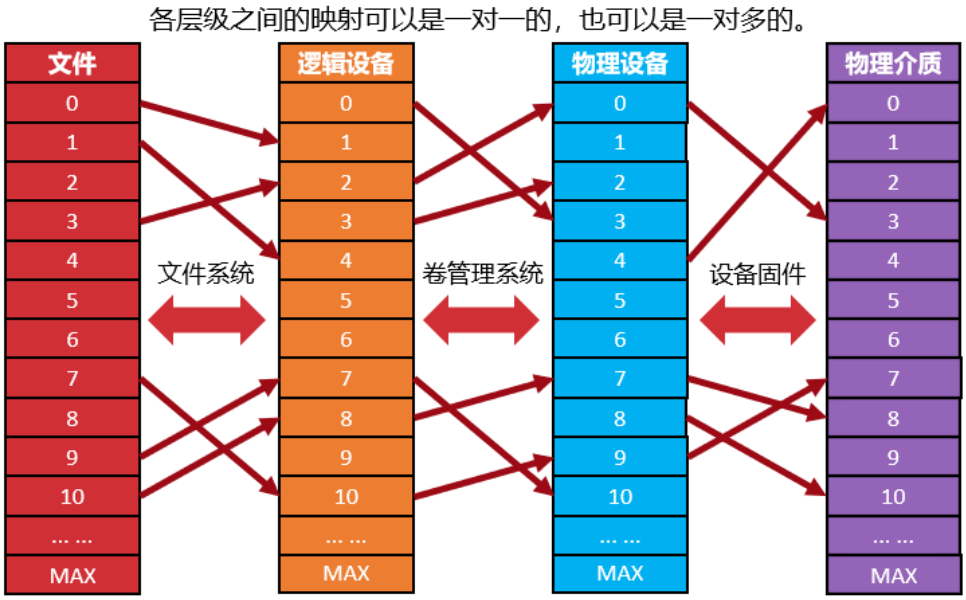
文件——一个具备一些**属性**的数据记录。我们可以根据其属性对相对应的数据记录进行**增（Create）删（Delete）改（Update）查 （Read；合称CRUD）。**它一般需要被**永久保存**，因而存储在非易失性**外存**上；可以看作是对外存的一种组织和抽象。

属性——文件的属性一般至少包括**文件名**，有时还包括**文件大小、 创建日期、修改日期、文件权限、所有者**等内容。除文件名外，它们一般都被储存在**文件控制块（File Control Block，FCB）**中

逻辑结构：

1. 字节流结构——最基本的结构，提供一个连续的存储空间，可以放置数据
2. 记录结构——提供一个固定大小的存储空间，可以放置大小固定的数据块
3. 索引结构——提供键值对，键—>值

物理结构——文件在外存上的物理保存形式，由多个中间映射层负责决定。



2.2 文件系统的概念

定义：

文件系统：

将文件中的**文件块**按照一定的方法最终**映射**到物理设备上的**物理块**，并在物理存储器特性的限制下**提供尽可能高的信息管理效率**。提供这种功能的**软件栈**称为文件系统。一般地，它**提供按名称和路径访问文件的功能。**

广义的文件系统包括这个映射中的所有层次。狭义的文件系统仅仅指从文件映射到逻辑设备这一个层次。

能不能将文件系统看成数据库？？

不能，文件系统更底层，更原始，提供的功能少，侧重数据的存储而不是查询，键值存储

关系型数据库则在文件系统的基础上提供更多数据抽象、数据查询、数据去重、数据统计、并发控制、故障恢复、权限控制等功能，可以看做是文件系统的升级版。

逻辑设备——逻辑上存在的外存设备。它可能是一个或数个物理外存设备按照某种方法拆分或组合产生的设备，内部的块称为逻辑块 （Logical Block）

分区——物理存储器按块号**分成**多个连续的逻辑存储器

阵列——将物理存储器按照一定规则**整合**到一个逻辑存储器

物理设备——实际存在的存储硬件设备

物理介质——物理设备中包含的实际存储介质

块的映射

早期：

外存和内存**区别不大**，接入系统中的存储器**是磁芯存储器和磁泡存储器**，它同时具备**非易失性和随机访问的能力**；其它存储器包括打孔卡片、纸带等需要人工装入计算机，它们可以看作是最早的外存。此时，访问任何一个文件块地址就是访问存储介质的物理块地址，因为**一个物理存储介质上只存储一个文件。**

磁盘的发明 温彻斯特（Winchester）硬盘等磁盘通过在平面磁介质上记录数 据，获得了较大的存储容量。此时，一个物理设备上可以储存 多个文件，这就需要将文件块的内容合理地安排到物理块上

容量的增大：

随着磁盘容量的进一步增大，**制造完美无瑕的存储介质变得不可能。**一片物理磁盘中，总有几个地方是有缺陷的，无法存储数据，这些缺陷称为**坏块。**然而，**存储设备必须给操作系统呈现连续而完美的设备存储空间**，因此要引入地址重映射的办法将这些**坏块屏蔽**掉。

在现代SSD中，**从设备块到物理块的翻译**由**闪存翻译层**（Flash Translation Layer，FTL）负责，它兼具**坏块屏蔽、磨损均衡和存储空间整理**等作用。

灵活性的提升：

随着硬盘容量的进一步增加，对硬盘管理的灵活性也在提升。 有时，我们希望**将硬盘分成多个区域**，每个区域负责一种或一 类用途，每个区域可以单独备份、迁移和管理；另一些时候， 我们希望**将多块硬盘组合成一个大的存储池**，用来承载单块硬盘难以承载的数据量，或者提高存储的可靠性。

此时便出现了逻辑设备的概念：它和物理设备之间可以是**一对 一、一对多、多对一或多对多。**在现代云计算中，存储层次更复杂，对存储灵活性的要求更高，甚至要求存储器的热插拔和程序的热迁移，因此逻辑设备与物理设备之间的映射越发复杂

路径和目录

文件系统一般以**树状**的方式组织文件

路径——文件在文件系统中的位置用**路径**表示。路径字符串由**一系列由分隔符隔开的子字符串组成**，它唯一确定一个文件。路径又可 以分为绝对路径和相对路径。分隔符一般是“/”或者“\”

目录——文件路径中由分隔符隔开的子字符串，又称文件夹。每一个目录都对应于一个文件组织层次。目录可以互相嵌套，一个目录下的其它目录称为它的子目录。

相对路径——从**某个目录D起始**，经由逐步查找能找到某文件F的路径，称为F相对于D的路径

绝对路径——某文件F相对于**系统根目录**的路径。也即从系统根目录出发，一步步跟随查找，能找到该文件F。

工作目录——某**应用程序**进行文件路径解析的**起始目录。**该程序中的一切路径都看做是对该目录的相对路径。

特殊目录——为了让**姐妹关系的目录**也能**使用相对路径进行互相引用**，每个目录下都设置两个子目录，它们是文件系统自动创建的。

- **目录“ . ” 当前目录。**这在一些需要指定某路径为当前目录的场合特别有用，比如指定程序的工作目录为当前目录就可以写“./”。
- **目录“ .. ”** 当前目录的**父目录。**引用这个目录就等于回到 当前目录的父目录。

2.3文件系统的组织

文件系统的性能指标：（只考虑文件系统—物理介质两个层次）

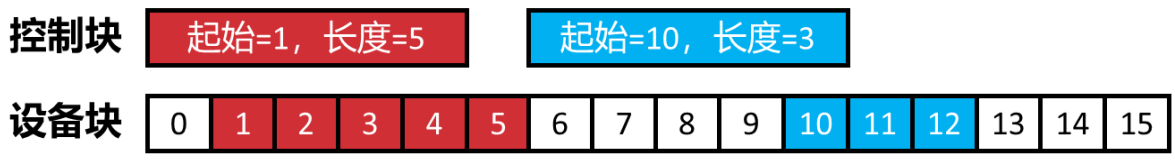
性能指标	评价方式
连续读取	MB/s，带宽
随机读取	IO/s，操作数量，或者ms，响应延迟
连续写入	MB/s，带宽
随机写入	IO/s，操作数量，或者ms，响应延迟
覆盖修改	MB/s，带宽
增加修改	MB/s，带宽
按名称查询文件	ms，响应延迟
按内容查询文件	ms，响应延迟
文件内寻址	ms，响应延迟
存储使用率	%，百分比
故障容错性	N/A

文件的存储：

1. 连续分配法

定义：将文件一个接一个的放置在外存上，**每个文件占据连续的物理介质块**

FCB只记录文件的起始块号和长度，就能唯一确定一个文件



优点：顺序和随机读写速度都快，不会出现内部碎片，管理数据结构短小精悍，存储介质利用率高，存储介质损坏只影响部分数据，数据可靠性高。

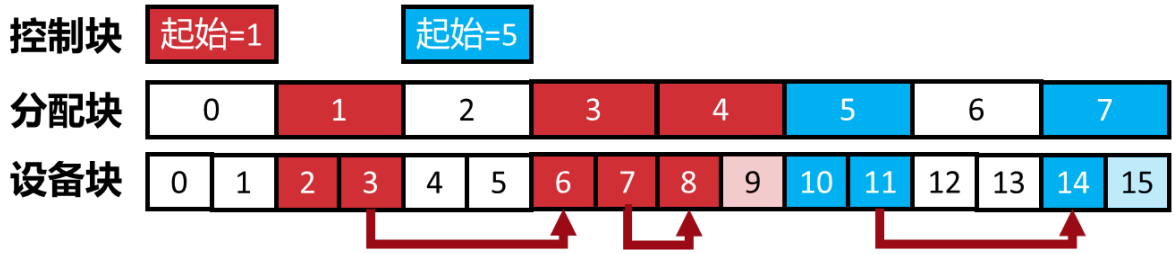
缺点：文件长度难以随意追加，可能存在外部碎片

适用场合：数据备份和归档等倾向于单次写多次读（Write Once Read Many， WORM）的场合。典型例子是用于LTO磁带的线性磁带文件系统 （Linear Tape File System，LTFS）

2. 链接分配法

定义：按照某个固定的块大小分配文件，每个块内部放置一个指针，指向下一个块。如果没有后续块，使用空指针。

FCB中只要记录文件的起始块号就能顺藤摸瓜找到整个文件；也可以说每个文件对应了一条块链。



优点：文件长度可以随意增加，不会产生外部碎片

缺点：顺序读写慢，随机读写更慢，管理数据结构包含指针，指针本身消耗存储介质，存储介质破坏，会导致指针丢失，所以数据的可靠性差。

问题：　分配块的大小的选择?

3. 链接分配法的改进

连续分配和链式分配的折中。

利用链表来避免外部碎片，利用可变长度块来避免内部碎片，

可变长度块称为**扩展**，内部包含指向下一个扩展的指针以及本扩展的长度，

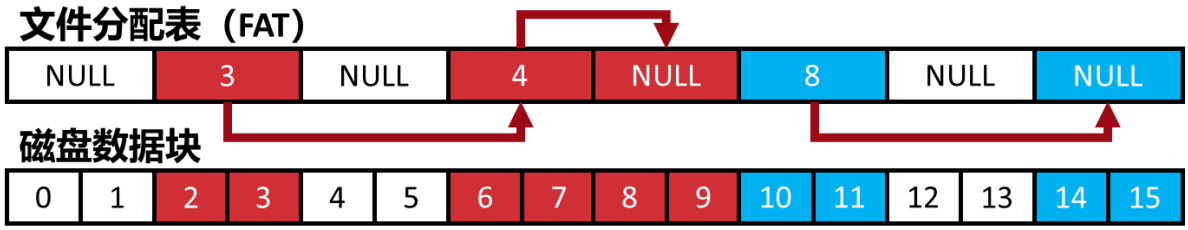
优点：文件长度可以随意增减，不会产生外部碎片，同时扩展的大小灵活，也不会产生内部碎片。

缺点：顺序读写有一定提升，随机读写还是慢，管理数据结构包含指针和扩展长度，在存储碎片化严重时候可能进一步消耗存储介质，存储介质破坏，会导致指针丢失，所以数据的可靠性差。

应用场景：这种文件系统是实用的最简单文件系统，其典型代表为exFAT。 exFAT支持按照固定块进行链表分配，也支持将整个文件放在一 个扩展中。但它不允许将整个文件分成两个或更多扩展。

4. 链接分配法的继续改进

链表备份——不将指针和扩展长度和文件结构存储在一起，而是分开，给他们单独的存储空间

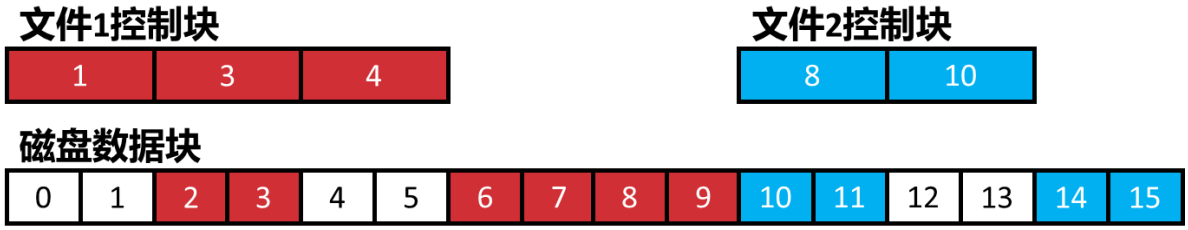


优点：管理结构就和数据分开，只需要单独备份管理信息就可以，管理结构中包含所有文件的所有块的组织信息，而且本身短小精悍，可以在盘上保存多个备份，只要还有一份管理信息在，整个文件系统就没有受结构性破坏，抗损性强，数据区受损只会引起数据损坏，不会导致文件系统崩溃。

随机访问提速： 小巧的管理结构可以被单独加载进内存。这样，随机寻址虽然 还是要做指针追逐，但是在内存中追逐指针速度要快得多。

5.索引分配法

索引分配——给每个文件创建一个线性索引表，每个表项记载对应于该逻辑块的物理块

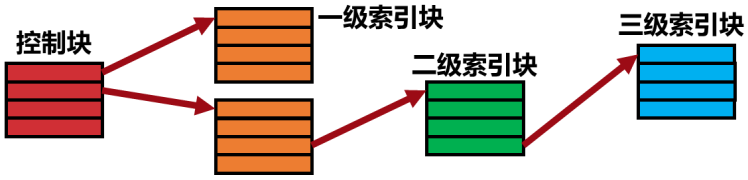


优点：连续和随机访问都快，抗损性能也强，、

确定：索引预留多了，浪费，索引预留少了，文件大小受限

6.索引分配的改进

多级索引——像组织页表，将多个索引表以层次的形式组织起来，每个层次负责翻译逻辑块号的一部分，最终得到物理块号，不使用的索引就不创建，不填充，虽然随机访问仍然引起指针追逐， 但是追逐的次数是有限的，也即索引的层数。



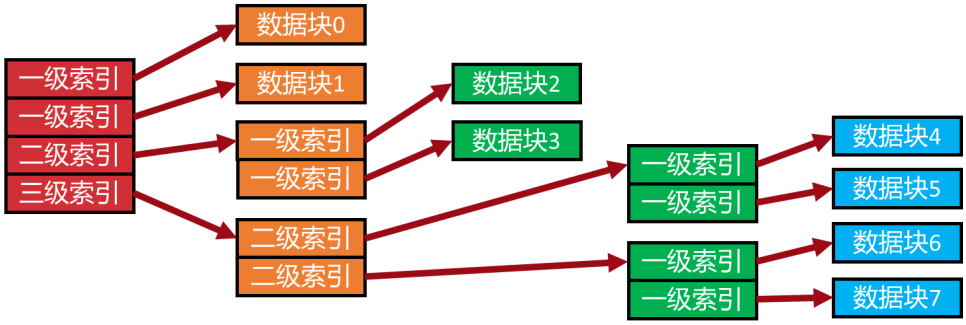
解决给文件预留多少索引的问题

稀疏文件——有一些文件仅占用整个逻辑空间中的几个很分散的逻辑块，而其 它块都是0。索引分配法可以轻松应对这种情况：不为那些全0的 逻辑块填充索引，也不为它们分配空间即可。

应用场景—— 除exFAT这种需要和极其古老的链式分配的FAT兼容的文件系统之外， 绝大多数现代文件系统都采取多级索引分配法。

7.索引分配法的继续改进

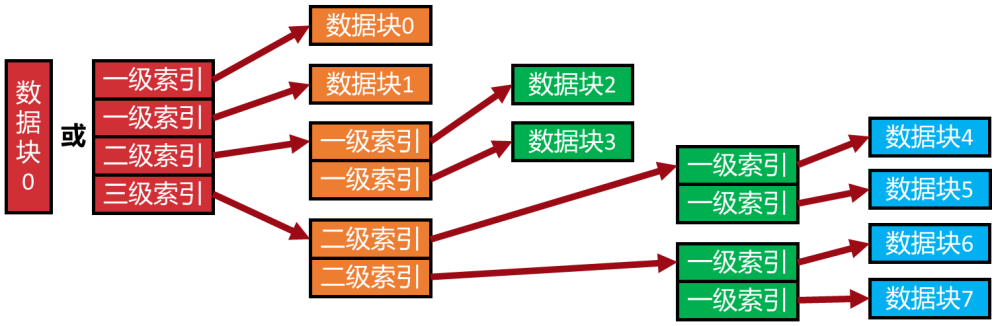
混合索引——文件的索引采取多级方法进行，但索引的级数随着文件块号的增加而增加。文件越靠前的部分，索引的级别越少。这样，小文件的存储效率就提高了



8.混合索引法的继续改进

直接内容法——文件非常小的时候，允许将文件内容直接记录再FCB中

从连续存储法的视角出发，这可以看作是连续存储法的一种应用；从索引存储 法的视角出发，这可以看作是“零级索引”，它们的内容直接就是文件内容。



空闲块——文件系统中暂时未被占用的物理块

怎么处理空闲块?

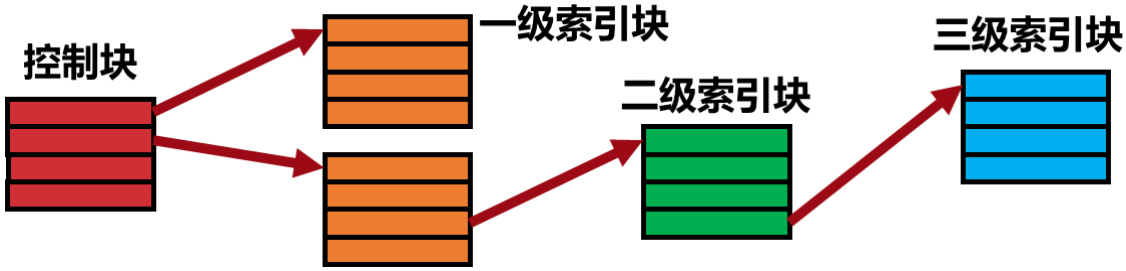
连续存储——将所有空闲块的地址和长度存储在表格中

链表存储——将所有空闲块放在链表中

索引存储——将所有空闲块放到多级索引中

存储方式	优点	缺点
连续存储	简单好实现，空闲块及其范围一目了然。分配和释放可以照内存分配的办法操作。	空闲块列表需要额外的空间来存储，而且该列表的大小是固定的，必须能够放下所有的物理块。
链表存储	相对而言也比较简单，分配单个块很容易。	分配和释放大量块时候会导致大量磁盘读写；找连续的扩展很麻烦。
索引存储	在分配单个块较容易的基础上，分配和释放大量块比较高效，因为索引表可以一次填充很多。	分配单个块需要做多级指针追逐。此外，结构复杂，找连续的扩展时候仍然很麻烦。

索引法的问题 一次分配多个块变得较高效了。但是，每次分配单个块都需要查询多级索引才能启动分配，启动成本太高。怎么解决这个问题?



1.分组索引法 空闲块可以先按照预定的数量分组，每个索引表都包含指向下一 个索引表的一个指针；除了这个指针之外，每组空闲块的地址直 接登记在该索引表内。先索引，再链接。

观察 比较该方法与分级索引法添加和分配单个空闲块和成组空闲块时 的操作成本。容易知道，成组空闲块的操作成本是差不多的，因 为主要的时间都花在了填充索引表格上。

单个添加 查找索引头指针指向的索引表。若它未满，将空闲块添加进列表 内；若它已满，将索引头指向它，然后将它初始化成索引表，将 它的后继指针指向索引头原本指向的索引表。

单个分配 查找空闲块头指针指向的索引表。若它未空，从它里面取出一个 空闲块；若它已空，将它本身取出，然后将索引头指向它原本指 向的索引表。

结论 比较可知，分组索引法绝大多数时候都只需要访问索引头指向的 那个索引表，不会像多级索引那样去做频繁的指针追逐。

我们之所以可以将空闲块表组织成这样，本质上是因为空闲块表 并非真正的文件：我们不关心空闲块之间的逻辑顺序，也不需要 维持一个逻辑地址到物理地址的映射。

进一步优化 正因为分组索引法绝大多数时候只需要访问索引头指向的索引表， 因此可以直接将那个表缓存到内存中。这样，绝大多数操作就是 在内存中访问线性表了，速度很快。

问题 一次分配单个和多个空闲块的问题都解决了，但是如果希望分配 物理上连续的扩展怎么办?

空闲块的组织：数据结构

直觉 将连续存储法中的空闲块区间按照大小做成红黑树。

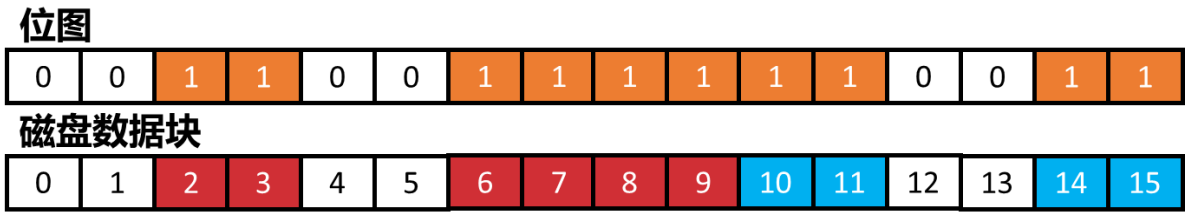
问题 红黑树的查询和修改要经过多次外存上的指针追逐，太慢。

解决方案 将空闲块区间组成的红黑树缓存在内存。

问题 连续存储法的空闲区表实在是太大了，放在硬盘上都嫌占地方， 放到内存还了得吗?

其它存储法 其它存储法根本没有办法做到这一点，因为它们不登记区间，只登记块。

2.位图法 用一系列二进制位对应每个块。如果该块被分配出去，那么位图 的对应位置就置1，否则置0。



问题 这个位图怎么查找呢？在硬盘上查找太慢了。

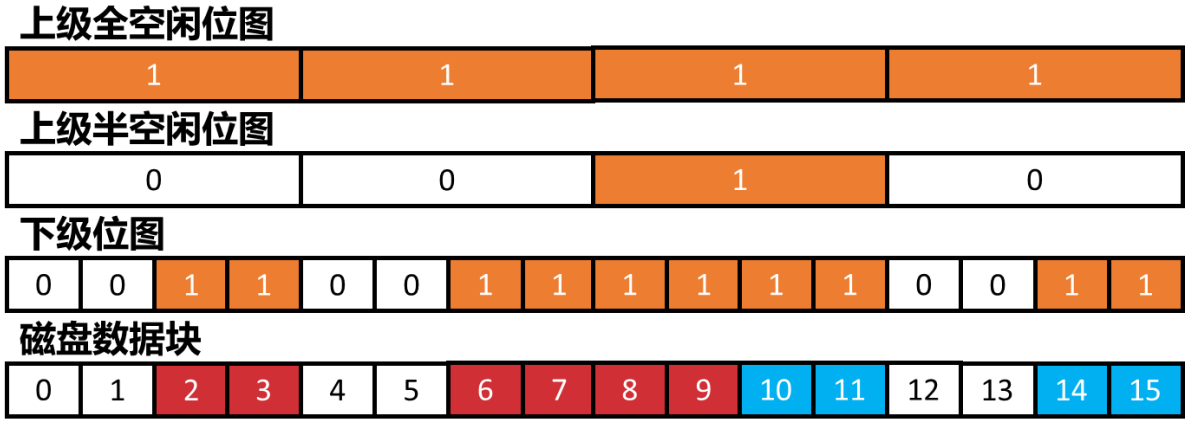
计算题：对于1TiB的磁盘，每4MiB一个块，位图大小是多少?

解决方案 既然位图所占空间不多，可将位图直接加载进内存。分配单个空 闲块只要找到第一个0所在的位置，分配连续的区间则是寻找足够 长的连续的0。释放空间就更简单，只要将那些1变成0就好了。

问题 但是找到第一个0，以及找到连续的足够长的0，看上去也不简单。

提示 位图可以按照不同粒度分级。

多级位图法 在位图法的基础上，使用多个不同粒度的位图。更高粒度的上级 位图可以有全空闲和半空闲两种：对于全空闲上级位图，若它的 某一位是0，代表其对应的整个下级位图区间的空间都是空闲的；对于半空闲上级位图，若它的某一位是0，则代表其对应的下级位 图区间中至少有一个空闲块。



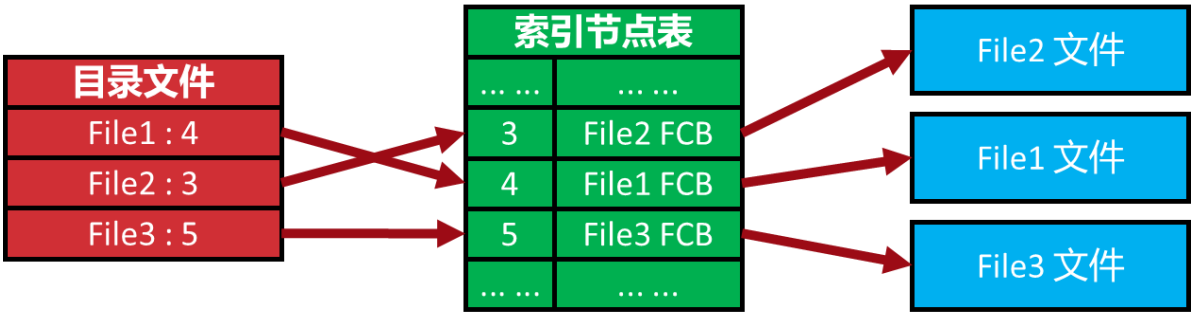
文件的查找：索引节点

思路 在FAT文件系统中，我们曾经把FAT从文件的数据块中分离开来。那 么这里我们同样可以把FCB里面不常变动的部分单独保存在一个表 里面，然后让目录文件去引用这个表就可以了。

索引节点 将FCB拆成两部分，文件名直接储存在目录文件中，而其它属性放
Index Node 在索引节点中。索引节点单独使用一个线性表存放，目录文件仅

inode 仅储存索引节点到索引节点编号的映射。这样，移动文件时完全 不需要触碰索引节点，仅需要触碰目录文件中的一行。

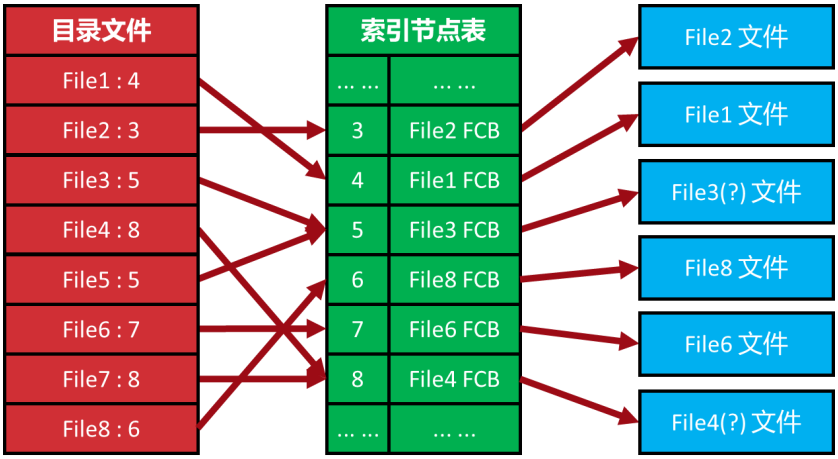
索引节点可以保存几个备份，目录文件就不需要保存多个备份了。 这样，一旦目录损坏，丢失的无非是文件名，文件的内容是不会 丢失的，进行文件恢复只要扫描索引节点表。



问题 如果两个目录文件中的不同文件名指向了同一个索引节点会怎样?

文件的引用：硬链接

硬链接 同一个文件在文件系统层面拥有两个文件名，**它们都指向相同的索引节点**，因而可以通过两个文件名访问同样的内容。这种通过 某个名称可以访问另一个名称所属内容的形式叫做"链接"，在 文件系统数据结构层面实现的链接称为硬链接。



问题一 链接这种结构有什么用途?

提示 文件编译、批处理、文件备份

问题二 两个文件名，谁是谁的硬链接?

文件名本质上就是对索引节点的硬链接，两个文件名的"地位" 是相同的。不过，在实践中常说第一个建立的是"文件本身"， 第二个则是文件的"硬链接"。

问题三 对于这种文件，对一个文件名做"删除"，会怎么样?

删除一个文件，实际上是删除对其索引节点的引用。文件系统会 记录一个索引节点被引用了多少次。当其所有的引用被删除时， 其数据块才会被释放。

文件的引用：软链接

软链接 一个独立于其目标的的链接文件，**该文件的内容是它指向的原文件的路径。**当应用程序操作软链接文件时，会通过某种方式 将操作转接到它链接的原文件上。

软链接不是文件系统级别的，而是在文件系统之上创建的、内 容特殊的独立文件。并非所有文件系统都支持硬链接，但一切 文件系统都必然支持软链接（为什么？）。

问题 既然软链接本质上不是文件系统的功能，谁来负责将操作转接 到原文件上呢?

内核负责 在部分操作系统中，由内核负责解析并转发操作。在打开一个 软链接文件时，内核将提取软链接文件中的实际路径，并据此 来打开真实的文件。这种软链接又叫符号链接（Symbolic Link；Symlink）。

应用程序负责 在部分操作系统中，应用程序需要自己识别打开的文件是否是 软链接，如果是则需要自行解析软链接并转去打开真实的文件。 这种软链接又叫做快捷方式（Shortcut；Launcher）。

文件的查找：索引节点表的改进

问题 索引节点耗尽了，需要扩充怎么办?

索引节点文件 索引节点的集合也可以看做一个特殊文件，因此它也可以用文件的 格式予以存储。当固定记录的方法不适合时，可以采用链表法、索 引法等方法存储。下图的例子是哪种存储方法?

问题 用于常规应用的EXT4并没有采用可追加索引节点的设计，但用于数 据中心和网络存储领域的ZFS等则允许追加。为何会有这种区别?

需求 EXT4文件系统仅供一般的系统盘等应用场景，文件的数量（~百万） 不大，文件系统的总大小（~TB）也不大，完全可以预留足够多的 节点。但XFS和ZFS则需要支持磁盘阵列海量存储器（~EB），且存 储器容量需要动态扩充，就不能采取一预留了之的方法了

文件的共享：多个进程打开同一个文件

文件共享 如果多个进程同时打开一个文件，它们对文件的访问次序是什 么样呢?

提示 访问分为读和写。可以分析不同的先后次序。

类别	先	后	影响
读-读 R-R	进程1读	进程2读	没影响，文件内容没变，怎么读都是读到一样的东西。
	进程2读	进程1读	
读-写 R-W	进程1读	进程2写	进程1读不到进程2写入的内容。
	进程2写	进程1读	进程1可以读到进程2写入的内容。
写-写 W-W	进程1写	进程2写	进程2写入的内容可能覆盖进程1写入的内容。
	进程2写	进程1写	进程1写入的内容可能覆盖进程2写入的内容。

观察 主要的问题出在写上。写会修改文件，和其它操作不能乱序。

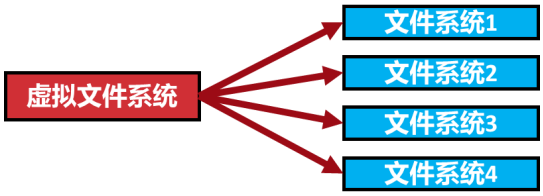
三.虚拟文件系统

3.1元文件系统

问题：不同文件系统有不同的组织形式，在文件，目录等方面不一而足，这就导致它们适用于不同的介质，并且适合不同的应用程序。这就使得外卖需要在一个操作系统中同时使用多个或所种文件系统，还有些时候，为了和其他操作系统兼容，外卖不得不**在系统中支持多个文件系统。**

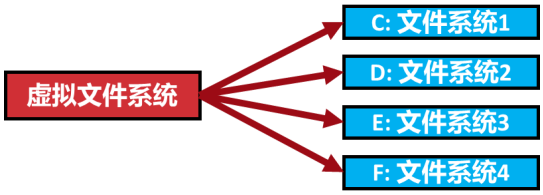
元文件系统

虚拟文件系统，系统中所有的问题件系统的更目录作为一个子目录出现在这个文件系统中，就可以用一种统一的路径描述来访问的各个文件系统，减轻了应用程序的负担。



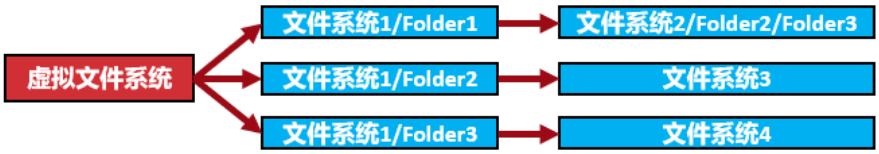
盘符法

仅允许将代表具体文件系统的子目录挂载在虚拟文件系统的根目录下，并且赋予不同文件系统独立的标志。这就是Windows的盘符机制



路径法

允许将任何一个文件系统的任何子目录挂载到任何文件系统的任何子目录下，并且一个文件系统及其子目录可以挂载多次。



3.2文件权限管理

访问控制矩阵ACM

记录不同保护域对文件操作的权限的矩阵，一般情况下是稀疏矩阵。

保护域	文件1	文件2	文件3	文件4
进程1	读，写			读
进程2	读，写			读
进程3		读	读，写	
进程4		读		执行

访问控制矩阵 Access Control List，ACL

按照每个文件记录不同保护域（进程）对文件操作的权限的列表，它储 存在文件的FCB中。进程访问该文件时，就从FCB中取出该文件的ACL，对 比查看进程是否有访问它的权限。

它可以看做ACM的按列压缩。

基于角色的访问控制 Role-Based Access Control，RBAC

按照角色记录对文件操作的权限，并同时记录每个用户属于哪些角色。

角色	用户
角色1	用户1，用户2
角色3	用户3
角色4	用户4

文件1 FCB	文件2 FCB	文件3 FCB	文件4 FCB
角色1：读，写	角色3：写	角色3：读，写	角色1：读
	角色4：读		角色4：执行

基于策略的访问控制 Policy-Based Access Control，PBAC

ABAC的另一种称呼，强调使用脚本而非描述列表来控制权限。每个访问 请求都被送入含有策略的脚本程序，并由程序来输出判别结果。比XML 更直观、更符合人类习惯，且可不受格式限制加入任意判断逻辑。

```
初始化 结果 = 拒绝;
如果 文件 == 文件1，文件2
{
        如果 角色 == 角色1，角色2
        {
                如果 活动 == 数据库管理 且 操作 == 读，写
                {
                        如果 时间 == 工作日早8点-晚6点
                                返回 结果 = 允许;
                }
        }
}
否则如果 角色 == 角色3 且 操作 == 读
        返回 结果 = 允许;
```

基于权能的访问控制 Capability-Based Access Control，CBAC

不将访问权限和文件结合，而是将其和具体的进程结合。每个进程都保 存自己的权能表（Capability Table，CT），其中每个权能都记载它可以对 某个文件做某些访问操作。它可以看做是ACM的按行压缩。

在进程启动时，其权能表为空，必须等待上级下发权能；权能可以在进 程运行的过程中由其管理者授予和撤销，这就解决了动态性的问题

进程1 PCB	进程2 PCB	进程3 PCB	进程4 PCB
文件1：读，写	文件1：读，写	用户3：读，写	用户1：读
文件4：读	文件4：读		用户2：读
			用户4：执行

自主访问控制 Discretionary Access Control，DAC

每个文件有一个“所有者”，该文件的权限可以由其“所有者”负责编辑。 对于RBAC、ABAC和PBAC，所有者负责编辑它拥有的文件的ACL、XML或 权限控制脚本；对于CBAC，所有者负责将其文件的访问权能添加到进程 的CT中。

强制访问控制 Mandatory Access Control，MAC

由系统中的一个或一组管理员统一编辑全部或分别编辑部分文件的权限。 无论是ACL、XML还是CT都由系统管理员集中操作，对文件的访问控制权 更集中。

问题一 DAC和MAC各有哪些优缺点?

DAC更灵活，适合用于宽松的资源共享环境；MAC则更严肃，适合企事 业单位等需要严格控制信息流转的环境。

问题二 系统中除了文件之外，还有其他设备等等。这些成分的权限管理能否也 参照文件进行?

3.3一切皆文件

硬件设备以及系统本身的运行状态等也可以使用文件抽象，使用和文件相同的权限管理模型和系统调用，着大大扩展了文件系统的内涵。

6.2空间的协调—文件系统（文件）

一.文件系统的实现

1.1系统文件总览

分类：

按来源划分：

- 系统文件——操作系统开发商提供的文件，包括内等关键部分。普通用户对这部分文件没有访问权限，只有在系统更新的时候才会被覆写
- 中间件文件——程序设计语言，驱动程序和中间件开发商提供的文件。普通用户只能读和执行文件，只有在安装新功能时才会被添加，一旦田间则不会被消除
- 用户文件——用户自行创建的文件，包括各种应用程序，数据库和文档等。
- 临时文件——由用于程序或系统生成的缓存性质的文件。这些文件一般体积小，零碎，而且在使用过一次后便无其他作用，需要定期清理。

按性质划分：

- 索引文件——存放关于其他文件学习的文件。目录文件，内容索引文件，软连接都可以看作索引文件的一种。
- 设备文件——实际上不是一个文件，但被抽象成一个文件的设备或系统运行状态信息，部分设备文件还可以用来做跨进程通信（IPC）
- 普通文件——存储用户信息的文件。

按权限划分：

- 不可访问文件——普通用户无权访问的文件
- 只读/写文件——普通用户只能读写的文件
- 可执行文件——含有进程的描述，可以用以启动进程的
- 隐藏文件——一般不展示给用户的文件

1.2分区：GPT

GUID（global unique ID）

一个128位的随机产生的ID，几乎可以保证世界上没有两个相同的。可以看成是某种哈希值标 签，也称为Universally Unique ID（UUID）。

磁盘分区：将一个物理硬盘互粉成若干个逻辑区域的过程

GPT（GUID partition Table）

是一种磁盘分区表的标准，用来管理硬盘上的分区，GUID表示全局唯一标识符，Partition Table代表磁盘分区表。

和MBR分区表相比，GPT支持更大的硬盘容量和更多的分区

GPT分区描述符：每个分区都有一个对应的分区描述符，用于描述该分区的信息，每个分区描述符都是一个128字节的数据结构，其中包含分区的其实地址，大小，GUID等信息。

字节地址	字节长度	描述	设置
0x00	16	分区类型GUID	系统自行决定类型与GUID的对应，该字段标识了该分区的类型，例如普通数据分区、EFI系统分区、Microsoft保留分区等。
0x10	16	分区GUID	随机生成一个，该字段标识了该分区的唯一标识符，也是全局唯一的。这个GUID可以用来在不同的系统中唯一地标识该分区，
0x20	8	分区起始LBA	这两个字段标识了该分区在磁盘上的起始位置和大小。通常情况下，每个分区的大小是以扇区为单位进行计算的。
0x28	8	分区结束LBA	-
0x30	8	分区属性	第0位置位代表系统固件，第1位置位代表操作系统恢复分区等。 该字段标识了该分区的一些属性，例如是否为只读分区、是否为隐藏分区等。
0x38	72	分区名	该字段标识了该分区的名称，可以用于在文件系统中显示分区的名称。 -

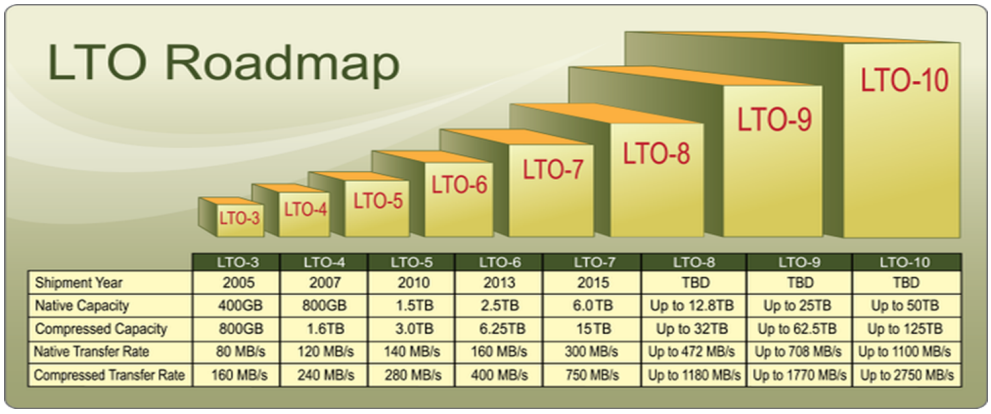
<table>
<tr><th>设备块地址</th><th colspan="4">描述</th></tr>
<tr><td>0</td><td colspan="4">MBR兼容头</td></tr>
<tr><td>1</td><td colspan="4">主GPT头</td></tr>
<tr><td>2</td><td>分区0描述符</td><td>分区1描述符</td><td>分区2描述符</td><td>分区3描述符</td></tr>
<tr><td>3</td><td>分区4描述符</td><td>分区5描述符</td><td>分区6描述符</td><td>分区7描述符</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>34</td><td>分区124描述符</td><td>分区125描述符</td><td>分区126描述符</td><td>分区127描述符</td></tr>
<tr><td>35</td><td colspan="4">分区0可用空间</td></tr>
<tr><td>...</td><td colspan="4">...</td></tr>
<tr><td>N-33</td><td colspan="4">分区127可用空间</td></tr>
<tr><td>N-32</td><td colspan="4" rowspan="3">分区描述符的备份</td></tr>
<tr><td>...</td></tr>
<tr><td>N-2</td></tr>
<tr><td>N-1</td><td colspan="4">GPT头的备份</td></tr>
</table>

1.3磁带：LTFS

线性磁带——数据磁带，容量大，价格便宜，适合备份大量数据使用，

只写，可以作为法律文书，商务合同等防篡改只读文档

发展史：



	LTO-3	LTO-4	LTO-5	LTO-6	LTO-7	LTO-8	LTO-9	LTO-10
Shipment Year	2005	2007	2010	2013	2015	TBD	TBD	TBD
Native Capacity	400GB	800GB	1.5TB	2.5TB	6.0TB	Up to 12.8TB	Up to 25TB	Up to 50TB
Compressed Capacity	800GB	1.6TB	3.0TB	6.25TB	15TB	Up to 32TB	Up to 62.5TB	Up to 125TB
Native Transfer Rate	80 MB/s	120 MB/s	140 MB/s	160 MB/s	300 MB/s	Up to 472 MB/s	Up to 708 MB/s	Up to 1100 MB/s
Compressed Transfer Rate	160 MB/s	240 MB/s	280 MB/s	400 MB/s	750 MB/s	Up to 1180 MB/s	Up to 1770 MB/s	Up to 2750 MB/s

磁带介质的特点：

1. 顺序读写：磁带是顺序读写的，不能随机读写
2. 价格便宜：单位容量价格便宜

适合磁带的文件系统：

1. 顺序读写性能好
2. 空间利用率高

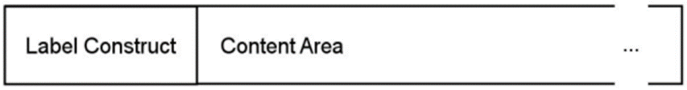
节约存储器的方法：

线性磁带文件系统LTFS

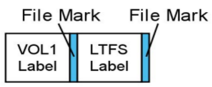
一种适合于磁带的文件系统。**顺序读写快、空间利用率高**，并自带压缩 功能：文件存储到介质上之前会被自动无损压缩，读取时则**自动解压缩**， 最大限度节省介质容量。

LTFS分区

分为标签区和内容区

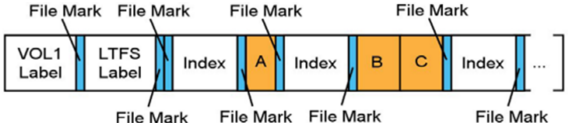


1. 标签区：



保存卷标

2. 内容区：



首先是索引区，该区域是应该XML文件，描述从文件名到文件内容位置的映射

如何修改文件?

无法直接追加

只能把更新后的文件卸载磁带最后面

这样就会存在一个老版本和一个新版本

代数——新版本的索引块中会标明一个比老版本更高的代数，用来判断哪个索引是最新的。

老版本的索引中有一个指针指向新版本的索引，

新版本中索引中有一个指针指向老版本的索引，这样就可以相互查找

唯一释放所有空间的方法就是重新初始化文件系统

缺点：浪费 空间

优点：可以做版本迭代

1.4硬盘EXT4

硬盘：EXT4

历史：

历史 EXT系列文件系统普遍被用于Linux的各个发行版，到目前为止一共发展了四代。它在个人PC和一般企业场景有不错的适用性。在四次升级中，EXT的基于inode索引的核心思路并未改变，因此保持着良好的向后兼容性。

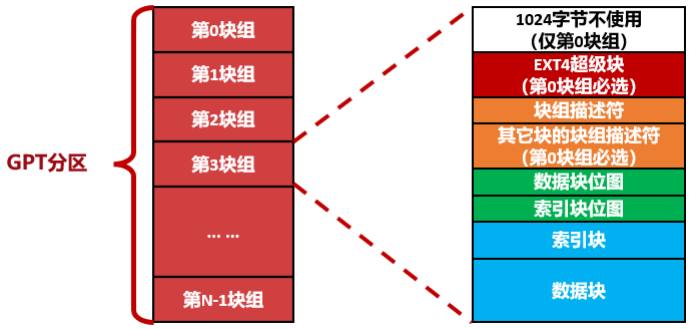
EXT 第一个版本，于1992年发布，作为Linux的原生文件系统。它是一个非常简单的文件系统，能应付最大2GB的文件。

EXT2 对EXT的一个升级，于1993年发布。从BFFS（Berkeley Fast File System）借来了一些概念。它仍然非常简单且没有日志功能，因此任何意外掉电都仍能造成文件系统损坏。能应付最大2TB的文件，文件系统的总大小可达32TB。

EXT3 对EXT2的一个升级，于2001年发布。增加了日志功能（后面详讲），因此掉电不容易损坏文件系统。能应付最大2TB的文件。

EXT4 对EXT3的一个升级，于2008年发布。能应付最大16TB的文件，文件系统的总大小可达1EB（1048576 TB）。

块组——EXT4将磁盘上的设备块按照一定的数量组成块组



块组也由自己的描述头进行描述

字节地址	字节长度	名称	描述
0x00	4	bg_block_bitmap_lo/hi	块位图地址
0x04	4	bg_inode_bitmap_lo/hi	inode位图地址
0x08	4	bg_inode_table_lo/hi	inode表地址

字节地址	字节长度	名称	描述
0x0C	4	bg_free_blocks_count_lo/hi	块组中空闲块数
0x0E	4	bg_free_inodes_count_lo/hi	块组中空闲inode数
0x12	2	bg_flags	块组状态（未初始化、初始化好、全零）
0x1E	2	bg_checksum	块组头校验和

超级块——EXT4文件系统的全局描述

字节地址	字节长度	名称	描述
0x00	4	s_inodes_count	总的inode数目
0x04	4	s_blocks_count_lo	总的块数
0x0C	4	s_free_blocks_count_lo	总的空闲块数
0x10	4	s_free_inodes_count	总的空闲inode数
0x14	4	s_first_data_block	第一个数据块的地址
0x20	4	s_blocks_per_group	每组的块数目
0x28	4	s_inodes_per_group	每组的inode数目
0x3A	2	s_state	当前状态（可用、有错误、恢复中）
0x78	16	s_volume_name	文件系统卷标
0xE0	4	s_journal_inum	日志文件的inode编号
0x3FC	4	s_checksum	超级块校验和

为什么超级快在每个块组都由一份可选拷贝？防止破坏的时候全部被破坏

索引节点inode

inode内部描述了文件的大部分属性

字节地址	字节长度	名称	描述
0x00	2	i_mode	文件模式（见后）
0x02	2	i_uid	所有者的ID
0x04	4	i_size_lo/hi	文件的总大小
0x08	4	i_atime	访问时间
0x0C	4	i_ctime	索引修改时间
0x10	2	i_mtime	数据修改时间
0x14	2	i_dtime	删除时间
0x1A	2	i_links_count	硬链接数量
0x1C	4	i_blocks_lo/hi	总块数
0x28	60	i_block	实际的索引树

文件模式i_mode——文件权限

读，写，执行三个独立权限，

用户分为三类：拥有者，本组用户，其他用户

i_mode可以叠加

i_mode值	名称	描述	解释
0x1	S_IXOTH	Execute，Others	其它用户可执行
0x2	S_IWOTH	Write，Others	其他用户可写
0x4	S_IROTH	Read，Others	其他用户可读
0x8	S_IXGRP	Execute，Group	本组用户可执行
0x10	S_IWGRP	Write，Group	本组用户可写
0x20	S_IRGRP	Read，Group	本组用户可读
0x40	S_IXUSR	Execute，Owner	拥有者可执行
0x80	S_IWUSR	Write，Owner	拥有者可写
0x100	S_IRUSR	Read，Owner	拥有者可读
0x200	S_ISVTX	Sticky Bit	仅拥有者和超级用户可删除子文件夹或文件

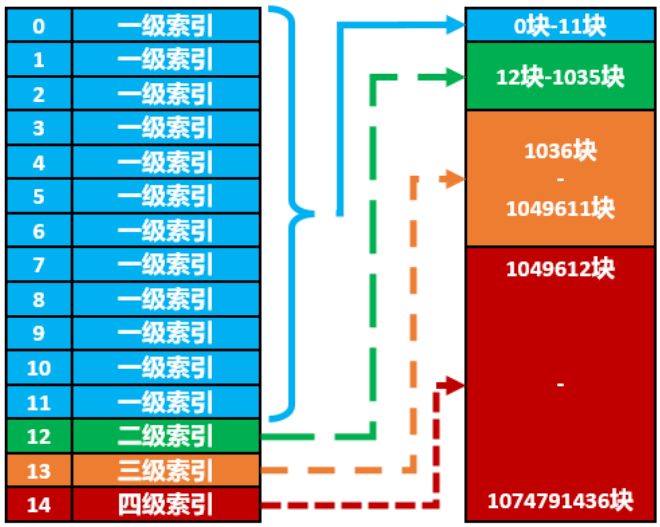
访问控制方法：DAC（自主访问控制）

RBAC（基于角色的访问控制）

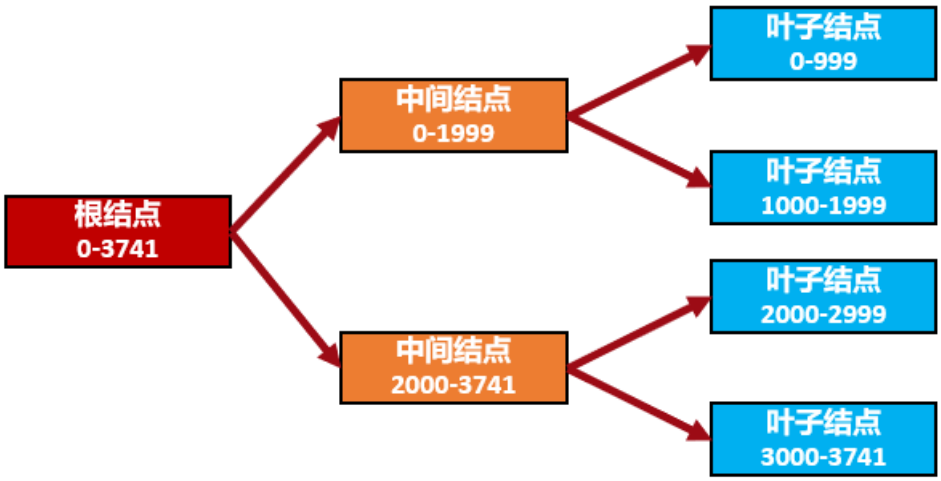
特殊文件的i_mode——imode前面还有些位，用来标记文件的性质（按性质分类）

i_mode值	名称	描述	解释
0x1000	S_IFIFO	FIFO File	有名管道（IPC）
0x2000	S_IFCHR	Character Device File	字符设备文件
0x4000	S_IFDIR	Directory	目录文件
0x6000	S_IFBLK	Block Device File	块设备文件
0x8000	S_IFREG	Regular File	常规文件
0xA000	S_IFLNK	Symbolic Link	符号链接文件
0xC000	S_IFSOCK	Socket	套接字文件（网络通信）

i_block内部结构



EXT4引入了**扩展记录树**的概念，不再使用固定的索引。一个扩展记录的 长度最多可达1000块以上。每颗树都分树头结点、中间结点和叶子结点 三种，叶子结点指向真正的扩展记录地址，中间结点则负责按照二分查 找法或多分查找法解析到叶子结点。每个结点本身的大小都是12字节



根结点的关键域（struct ext4_extent_header）

字节地址	字节长度	名称	描述
0x00	2	eh_magic	魔法数字，必须填写0xF30A
0x02	2	eh_entries	头之后的有效中间结点数（最多多少？）
0x06	2	eh_depth	当前节点深度；0表示下一级就指向数据

中间结点的关键域（struct ext4_extent_idx）

字节地址	字节长度	名称	描述
0x00	4	ei_block	该结点及其子结点覆盖的块起始地址
0x04	6	ei_leaf_lo/hi	下一级节点的块起始地址

叶子结点的关键域

字节地址	字节长度	名称	描述
0x00	4	ee_block	该结点覆盖的块起始地址
0x04	2	ee_len	该结点覆盖的块长度
0x06	6	ee_start_lo/hi	数据块起始地址

目录文件的内容

字节地址	字节长度	名称	描述
0x00	4	inode	该文件对应的inode号
0x04	2	rec_len	目录项的长度
0x06	1	name_len	文件名的长度
0x07	1	file_type	文件类型
0x08	255	name	文件名

文件类型（file_type）

类型	描述	类型	描述
0x0	未知	0x4	块设备文件
0x1	常规文件	0x5	有名管道文件
0x2	目录文件	0x6	套接字文件
0x3	字符设备文件	0x7	符号链接

根据文件名称得到哈希值，在查找哈系数的带具体的目录项

哈希树头结点（struct dx_root）

字节地址	字节长度	名称	描述
0x1C	1	hash_version	哈希函数
0x1E	1	indirect_levels	树深度
0x24	4	block	哈希值为0的文件块位置
0x28	-	entries	尽可能多的排好序的dx_entries

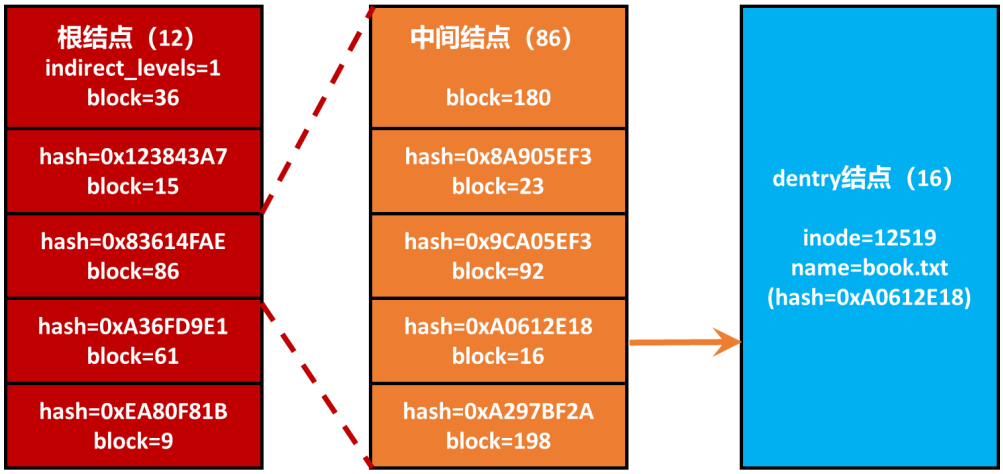
哈希树中间结点（struct dx_node）

字节地址	字节长度	名称	描述
0x0A	2	count	哈希树当前子结点数目
0x0E	4	block	子结点中最低哈希值的文件块位置
0x12	-	entries	尽可能多的排好序的dx_entries

哈希树叶子结点（struct dx_entry）

字节地址	字节长度	名称	描述
0x00	4	hash	哈希值；和上一条目的哈希值组成区间
0x04	4	block	含有相应条目的文件块，或下级节点

根据文件名称得到哈希值，在查找哈希数来找到目录项



抗损性 EXT4文件系统通过日志来防御意外掉电。

日志 Journal

文件系统在进行操作时，

（1）将操作写入日志，

（2）按照日志条目去操作文件系统，

（3）在完成这些操作后消除掉日志条目。

这样， 一旦掉电发生，就可以通过查询日志来了解哪些操作没有完成。这样， 每一个操作就都是原子化的事务（Transaction）。

日志文件系统 Journaling File System

带有日志恢复功能的文件系统。现代文件系统普遍都是日志文件系统。

问题 日志文件系统中，日志与文件系统掉电时的状态关系可能有哪些?

情况一 日志条目写入未完成。

情况二 日志条目写入完成，但操作未执行完毕。

情况三 操作已经执行完毕，但日志条目未被消除。

情况二与情况三怎么区分？怎么知道操作是否完成？

情况一 日志条目写入未完成。

解决方案 下次上电时直接删掉不完整的日志条目，相当于最后的操作从未发生。（装作无事发生；本次操作丢失）

情况二 日志条目写入完成，但操作未执行完毕。

解决方案 下次上电时根据日志条目**继续执行操作**，执行完毕后消 除掉日志条目。

情况三 操作已经执行完毕，但日志条目未被消除。

解决方案 下次上电时**检查日志条目描述的操作是否完成**，若完成则不再重复操作，若未完成则继续操作直到完成，完成后消除掉日志条目。

幂等性（Idempotence）

对某对象执行任意多次相同操作得到的**结果不变**。

日志文件系统的完整日志条目即具备幂等性，反复上电执行同一个日志条目得到的文件系统状态是一样的。

日志文件

EXT4的日志也是以文件格式存储的，称为日志文件。它的inode 编号传统上是8号，一般放置在靠近分区中间位置的某个块组内 的连续存储空间上。

在一般情况下，EXT4的日志**只存储文件系统管理结构操作**，并不储存实际数据，因此在掉电时只能保证文件系统结构的完整性， 正在操作的文件数据的完整性则无法保证。

除了将日志文件创建在文件系统所在的同一个逻辑设备上，EXT4 也允许在创建文件系统时为日志文件指定专用的存储设备。

问题 （1）为何要将日志文件放在连续存储空间上？

顺序追加的

（2）为何这个连续存储空间要靠近分区的中间位置？

访问的平均距离最短

（3）为何日志不能连实际数据一起存储？这里有什么考虑？

写日志需要时间，实际数据不重要

（4）将日志创建在其它存储设备上有什么好处？对这种设备的性能和可靠性有什么要求？

1.5闪存YAFFS

嵌入式系统：对工号，价格，体积都给长敏感的系统，通常只有一个核心处理器

NAND闪存的特点

读粒度 按页（如512字节）。

写粒度 按页（如512字节），写入1则存储单元值不变，写入0则存储单 元值变为0。

擦除粒度 按块（如2kB），擦除后该块内所有页的存储单元全部变成1。

缺陷 可能有部分坏块；坏块无法使用。

观察 闪存的读写粒度和擦除粒度是不一样的，这意味着追加容易，修 改难（有点像哪种介质？）。具有一定的可覆写性，但是只能将 1覆写成0，无法将0覆写成1。

日志结构文件系统 Log-structured File System, LFS

对文件进行修改时，不修改原有介质内容，而是在靠后的介质中追加内 容，并将原有介质内容通过某种方法作废。

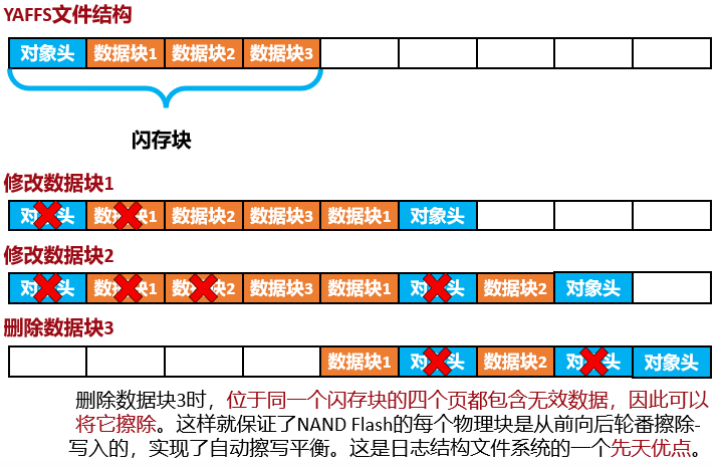
不要将它和日志文件系统混淆。日志文件系统是指文件系统具备一个恢 复日志，而日志结构文件系统则是指整个文件系统就是一个不断追加 的巨型日志。日志结构文件系统可以看作是日志文件系统的一个特例。

YAFFS Yet Another Flash File System

一种专用于嵌入式设备中的NAND Flash的文件系统，具备擦写平衡、垃 圾回收、坏块检测和规避等功能，可以直接和物理NAND Flash接口而无 需通过中间层，等于把SSD的FTL（Flash Translation Layer）和文件系统本 身集成在了一起。有两个版本，最新的YAFFS2能适配最新的NAND。

对象头 YAFFS中的FCB，记录文件的属性和它的块号。一个FCB的大小就是一个页。

数据块 YAFFS中的数据块，记录文件的数据。一个数据块的大小也是一个页。



删除数据块3时，位于同一个闪存块的四个页都包含无效数据，因此可以将它擦除。这样就保证了NAND Flash的每个物理块是从前向后轮番擦除-写入的，实现了自动擦写平衡。这是日志结构文件系统的一个先天优点。

YAFFS坏块检测

NAND Flash的块中数据并不是绝对可靠，因此需要配合纠错码（Error Correction Code，ECC）使用。ECC能够纠正一些错误的比特位，但其纠 错能力并非无限。当块中的错误太多而无法纠正时，就说这是坏块。

在出厂时，工厂会检测整个芯片，并标记一些块为坏块。随着使用，块 会磨损，坏块会变多。YAFFS对SLC Flash的策略是，如果三次读取都发生 ECC失败，则标记这个块为坏块，以后也不再使用它。

YAFFS主动垃圾回收

随着使用，Flash会逐渐碎片化。其每个块都是有效页和无效页夹杂的， 每写一页都要擦除含有有效页的一整块，再把有效页另找地方写回去。 这种现象叫写入放大（Write Amplification），在降低性能的同时会急剧 消耗Flash寿命。此时YAFFS会进行主动垃圾回收，将有效页尽量迁移到整 个块上，并擦除完全空白的块。在固态硬盘中，类似的功能称为TRIM

YAFFS标记数据

NAND Flash的每个页都会有一个备用（Spare）区，设计用来存放每个页 的ECC值以及坏块或坏页标记。YAFFS利用了这一点，将文件块的编号和有效标记等都一并放在这个区域。

问题 这样做有什么好处？给我们什么启发？

不占用主要的页空间，充分利用了Flash介质的物理特性，提高了存储效 率。在设计专用文件系统时，需要对介质的物理特性非常了解，并进行 针对性优化。

1.6池化

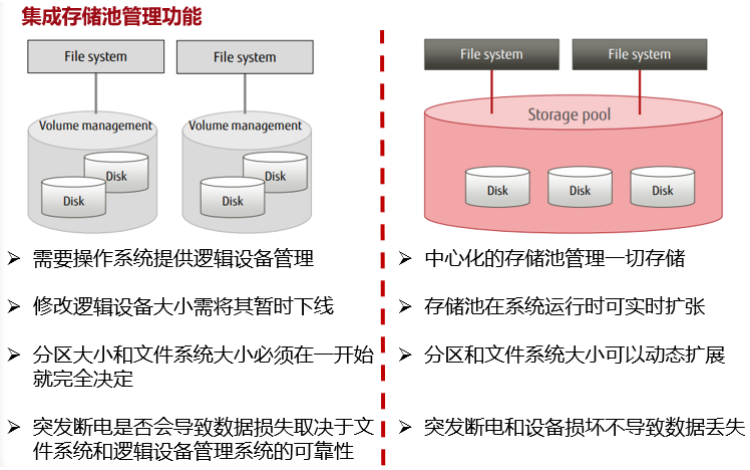
ZFS：zettabyte File System

设计用于海量存储的灵活型文件系统。

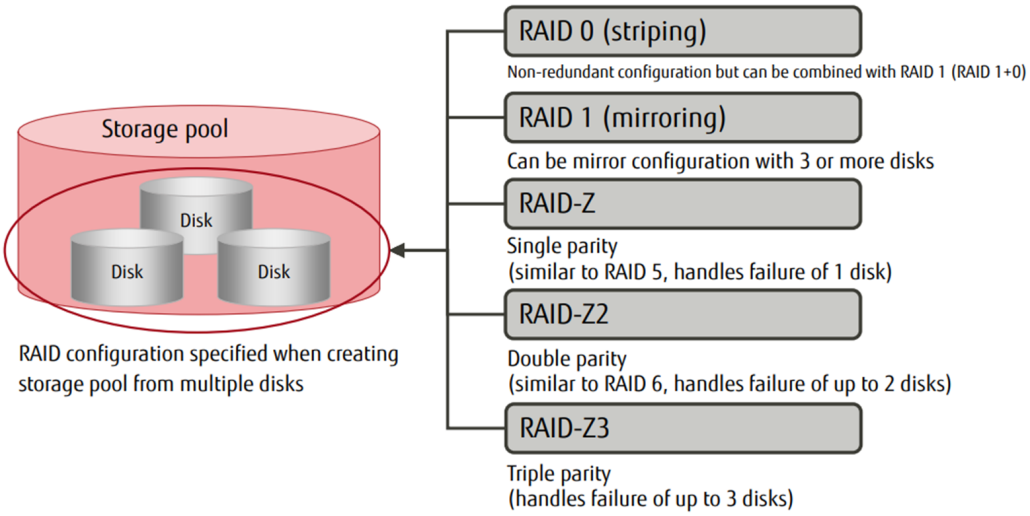
128位文件系统，inode 数量时可以增长的，因此支持的文件数量时无限的

简便的存储池管理：不需要逻辑卷管理器，文件系统可以直接池化并管理存储设备，并且管理命令非常易用，可以从存储池中随意添加和移除存储设备，存储池的内容个可用性在整个过程中不改变

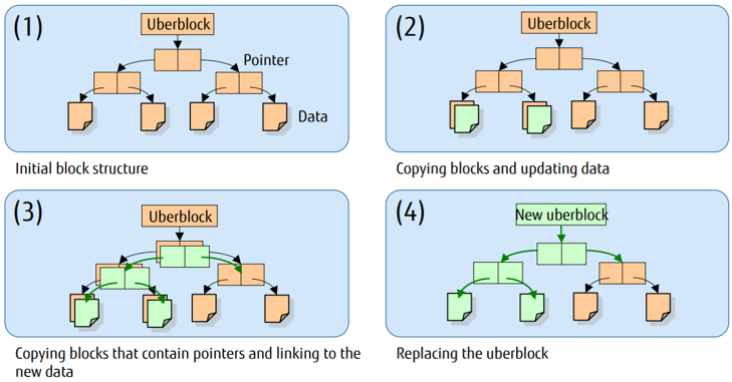
数据可靠性：内置RAID级别，设备损坏时数据可自修复



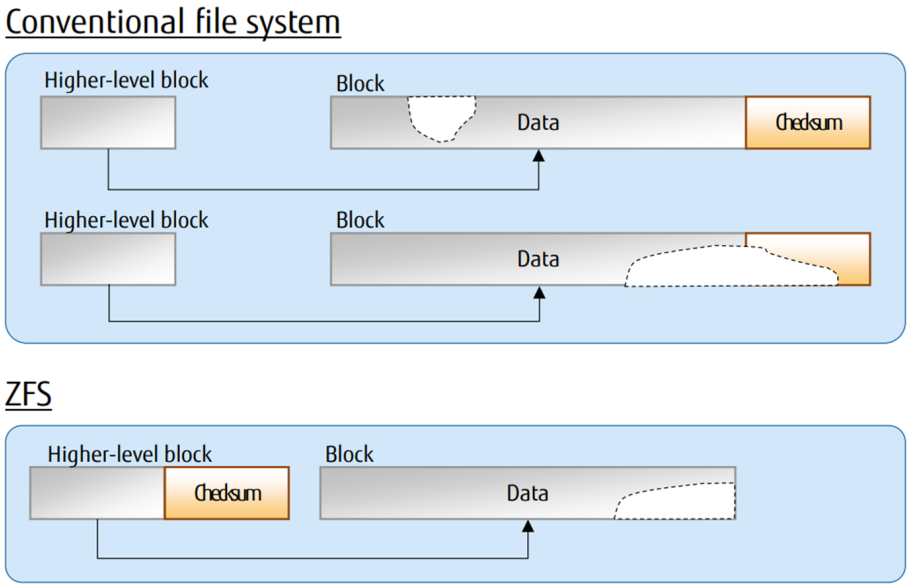
ZFS内置阵列功能：最多可以允许三个磁盘同时损坏



对抗突然掉电：Copy-on-Write



独立校验和：对抗连续介质损坏



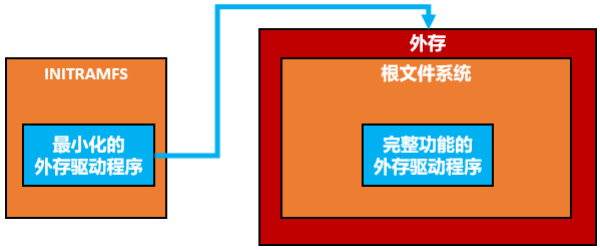
1.7启动INITRAMFS

系统启动

INITRAMFS:initial RAM File System

在启动器（Bootloader，如GRUB等）加载内核进内存时，也会在内存中 加载一个INITRAMFS，它具备基本的外存驱动程序。

内核启动后，先挂载INITRAMFS，然后从中加载外存的驱动程序模块。此 时内核才可以驱动外存，并将外存中的根文件系统挂载在INITRAMFS的根 目录下，覆盖掉INITRAMFS的所有内容（但INITRAMFS仍然被挂载着）。



1.8系统管理：PROC

PROC文件系统：被组织成文件形式的系统管理和状态信息。

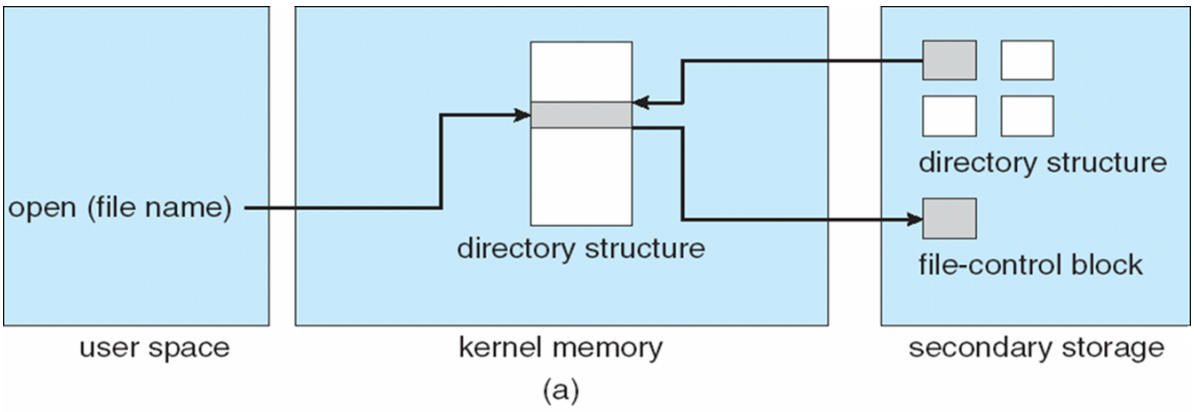
这样，文件的权限系统和访问方式也可以套用系统状态信息上，不需要再专门设计一套接口给它们。

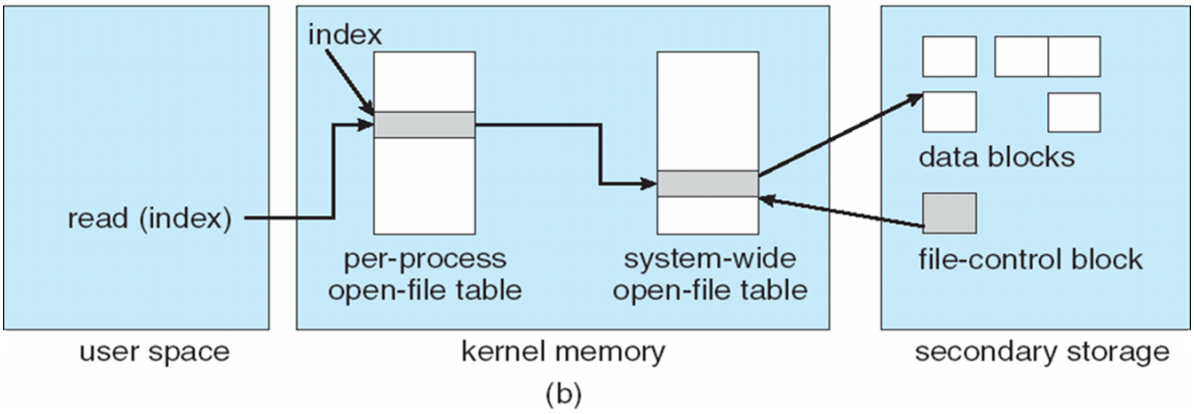
二.系统文件接口

2.1文件访问接口

C运行时库接口	Linux原生接口	Windows原生接口	功能
fopen	open/creat	CreateFile	打开或创建文件
-	mkdir	CreateDirectory	创建目录
fclose	close	CloseHandle	关闭文件
-	rmdir	RemoveDirectory	删除目录
-	rename	MoveFile	重命名或移动文件
fread	read	ReadFile	读文件
fscanf	-	-	格式化读文件
fwrite	write	WriteFile	写文件
fprintf	-	-	格式化写文件
fseek	lseek	SetFilePointer	移动文件指针
fstat	stat	GetFileAttributes	查询文件信息
-	do_ioctl(FS_IOC_SETFLAGS)	SetFileAttributes	设置文件信息
-	ftruncate	SetEndOfFile	截断文件
-	dup	DuplicateHandle	复制文件描述符
fflush	fsync	FlushFileBuffers	同步文件到磁盘

文件再内核内存中的描述结构





2.2访问控制接口

Linux原生接口	Windows原生接口	功能
stat	GetFileSecurity	获取文件权限
chmod/fchmod	SetFileSecurity	设置文件权限
access	-	测试文件权限
fchown	SetFileSecurity	改变文件所有者

磁盘配额限制 为了防止某用户使用过多磁盘空间，操作系统还允许限制某用 户的最大磁盘用量。

磁盘共享限制 部分操作系统提供将磁盘共享到网络的功能。网络可能会引入 额外的攻击面，因此共享时可以指定用户的访问权限。一个常 见的权限设定是只读，这样用户就只能远程读取磁盘上的文件 而无法修改它们。