# 文件系统实现

## 文件系统结构

磁盘具有两个优势

- 磁盘可以原地重写； 可以从磁盘上读取一块， 修改该块， 并写回到原来的位置。
- 磁盘可以直接访问它包含信息的任何块。 因此， 可以简单按顺序或随机地访问文件；并且从一个文件切换到另一个文件只需移动读写磁头， 并且等待磁盘旋转。

为了提高 I/O 效率， 内存和磁盘之间的 I/O 传输以块为单位执行。 每个块具有一个或多个扇区。

文件系统 提供提供高效和便捷的磁盘访问， 以便允许轻松存储、 定位、 提取数据。 文件系统有两个截然不同的设计问题。

- 第一个问题是， 如何定义文件系统的用户接口。 这个任务涉及定义文件及其属性、 所允许的文件操作、 组织文件的目录结构。
  第二个问题是， 创建算法和数据结构， 以便映射逻辑文件系统到物理外存设备
- 基本文件系统 只需向适当设备驱动程序发送通用命令， 以读取和写入磁盘的物理块。
- 文件组织模块 知道文件及其逻辑块以及物理块。
- 逻辑文件系统 管理元数据信息。

## 文件系统实现

概述

在磁盘上， 文件系统可能包括如下信息： 如何启动存储在那里的操作系统、 总的块数、空闲块的数量和位置、 目录结构以及各个具体文件等。

- (每个卷的) 引导控制块 可以包含从该卷引导操作系统的所需信息。 如果磁盘不包含操作系统， 则这块的内容为空。
- (每个卷的) 卷控制块 包括卷 ( 或分区) 的详细信息， 如分区的块的数量 、 块的大小、 空闲块的数量和指针 、 空闲的 FCB 数量和 FCB 指针等。
- ( 每个文件系统的) 目录结构用于组织文件。
- 每个文件的 FCB 包括该文件的许多详细信息。 它有一个唯一的标识号， 以便与目录条目相关联。

内存中的信息用于管理文件系统并通过缓存来提高性能。 这些数据在安装文件系统时被加载， 在文件系统操作期间被更新， 在卸载时被丢弃。

- 内存中的安装表 包含每个安装卷的有关信息。
- 内存中的目录结构的缓存含有最近访问目录的信息。 ( 对于加载卷的目录， 它可以包括一个指向卷表的指针。)
- 整个系统的打开文件表 包括每个打开文件的 FCB 的副本以及其他信息。
- 每个进程的打开文件表 包括一个指向整个系统的打开文件表中的适当条目的指针， 以及其他信息。
- 当对磁盘读出或写入时， 缓冲区保存文件系统的块。

分区与安装

分区可以是 “生的” （或原始的， 空白的）， 没有文件系统； 或者 “熟的”， 含有文件系统。当没有合适的文件系统时， 可以使用原始磁盘

引导信息可以存储在各自分区中，引导信息通常是一系列连续的块， 可作为映像加载到内存。映像的执行从预先定义的位置， 如第一字节， 开始。 这个引导加载程序相应地了解足够的文件系统结构， 从而能够找到并加载内核， 并开始执行它。 它不仅包括指令以便启动一个具体的操作系统。

根分区, 包括操作系统内核和其他系统文件， 在启动时安装。 其他卷可以在引导时自动安装或以后手动安装， 这取决于操作系统。

虚拟文件系统

数据结构和程序用于隔离基本系统调用的功能与实现细节。 因此， 文件系统的实现由三个主要层组成

- 通过定义一个清晰的 VFS 接口， 它将文件系统的通用操作和实现分开。 VFS 接口的多个实现可以共存在同一台机器上， 允许透明访问本地安装的不同类型的文件系统。
- 它提供了一种机制， 以唯一表示网络上的文件。 VFS 基于称为虚拟节点或 v 节点 的文件表示结构， 它包含一个数字指示符以唯一表示网络上的一个文件。 (在一个文件系统内， UNIX inode 是唯一的。)

第二层称为虚拟文件系统 ( VFS) 层。 VFS 层提供两个重要功能

VFS 区分本地文件和远程文件； 根据文件系统类型， 进一步区分本地文件

VFS 根据文件系统类型调用特定文件类型的操作以便处理本地请求， 通过调用NFS 协议程序来处理远程请求。 文件句柄可以根据相应的 vnode 来构造， 并作为参数传递给这些程序。 实现文件系统类型或远程文件系统协议的层属于架构的第三层

## 目录实现

线性列表

目录实现的最简单方法是， 采用文件名称和数据块指针的线性列表。 这种方法编程简单， 但执行费时。

目录条目的线性列表的真正缺点是， 查找文件需要线性搜索。 目录信息使用频繁； 如果对其访问很慢， 用户会注意到的。

哈希表

哈希表根据文件名称获得一个值， 并返回线性列表内的一个元素指 针。 因此， 它大大地减少了目录搜索的时间。 插入和删除也是比较直截了当的， 虽然必须做出一些规定来避免碰撞

哈希表的主要困难是， 它的通常固定的大小和哈希函数对大小的依赖性。

或者， 可以采用溢出链接的哈希表。 哈希表的每个条目可以是链表而不是单个值， 可以采用向链表增加新的条目来解决冲突。

## 分 配 方 法

连 续 分 配

连续分配 方法要求， 每个文件在磁盘上占有一组连续的块。 磁盘地址为磁盘定义了一个线性排序。

文件的连续分配可以用首块的磁盘地址和连续的块数来定义。

连续分配文件的访问非常容易。 对于顺序访问，文件系统会记住上次引用的块的磁盘地址，如需要可读入下一块对于直接访问一个文件的从块b开始的第i 块， 可以直接访问块 b+i

不过， 连续分配也有一些问题。 一个难题是，为新文件找到空间。

连续分配的另一个问题是， 确定一个文件需要多少空间。

链接分配

链接分配解决了连续分配的所有问题。 采用链接分配， 每个文件是磁盘块的表；磁盘块可能会散布在磁盘的任何地方。 目录包括文件第一块和最后一块的指针。

然而， 链接分配确实有缺点。 主要问题是， 它只能有效用于顺序访问文件。

另一个缺点是指针所需的空间。 如果指针需要使用 512 字节块的 4 字节， 则 0.78% 的磁盘空间会用于指针， 而不是其他信息。 每个文件需要比原来稍多的空间。

这个问题的通常解决方案是， 将多个块组成簇，并按簇而不是按块来分配。

链接分配的另一个问题是可靠性

链接分配的一个重要变种是文件分配表 FAT) 的使用。 这个简单而有效的磁盘空间分配方法用于 MS-DOS 操作系统。 每个卷的开头部分的磁盘用于存储该表。 在该表中， 每个磁盘块都有一个条目， 并可按块号来索引。

索引分配

链接分配解决了连续分配的外部碎片和大小声明的问题。 然而， 在没有 FAT 时，链接分配不能支持高效的直接访问， 因为块指针与块一起分散在整个磁盘上， 并且必须按序读取。

索引分配 通过将所有指针放在一起， 即索引块 解决了这个问题

每个文件都有自己的索引块， 这是一个磁盘块地址的数组。 索引块的第i 个条目指向文_件的第i 个块。 目录包含索引块的地址

性能

当操作系统选择合适方法来实现时， 存储效率和数据块访问时间都是重要依据。

对于任何类型的访问， 连续分配只需访问一次就能获得磁盘块。 由于可以在内存中容易地保存文件的开始地址， 所以可以立即计算第i 块 （或下一块） 的磁盘地址， 并直接读取。

对于链接分配， 也可以在内存中保留下一块的地址， 并直接读取。 对于顺序访问， 这种方法很好; 然而， 对于直接访问， 对第 i 块的访问可能需要读i 次磁盘。 这个问题表明了，为什么链接分配不适用于需要直接访问的应用程序

索引分配更为复杂。 如果索引块已在内存， 则可以进行直接访问。 然而， 在内存中保存索引块需要相当大的空间。 如果没有这个内存空间， 则可能必须先读取索引块， 再读取所需的数据块。 对于两级索引， 可能需要读取两次索引块。

## 空闲空间管理

为了跟踪空闲磁盘空间， 系统需要维护一个空闲空间列表 空闲空间列表记录了所有空闲磁盘空间， 即未分配给文件或目录的空间。

位向量

空闲空间列表按位图 或位向量来实现。 每块用一个位来表示。 如果块是空闲的，位为 1; 如果块是分配的， 位为 0

这种方法的主要优点是， 在查找磁盘上的第 1 个空闲块和《 个连续的空闲块时相对简单和高效。

除非整个位向量都保存在内存中(并时而写入磁盘以便恢复)， 否则位向量就低效。 将位向量完全保存在内存中， 对于较小的磁盘是可能的， 对于较大的就不一定。

链表

将所有空闲磁盘块用链表链接起来， 将指向第一空闲块的指针保存在磁盘的特殊位置上， 同时也将其缓存在内存中。 这个第一个块包含下一个空闲磁盘块的指针

组

空闲列表方法的一个改进是， 在第一个空闲块中存储n 个空闲块的地址。 这些块的前n-1 个确实为空。 最后一块包含另外个空闲块的地址， 如此继续。

计数

不是记录 n 个空闲块的磁盘地址， 而是记录第一块的地址和紧跟第一块的连续空闲块的数量这样， 空闲空间列表的每个条目包括磁盘地址和数量。

空间图

对于空闲空间的管理， ZFS 采用了组合技术， 来控制数据结构的大小并最小化管理这些数据结构所需的 I/O。

## 效率与性能

效率

磁盘空间的有效使用在很大程度上取决于磁盘分配和目录算法。

保存在文件目录条目 内的数据类型也需要加以考虑。 通常， 要记录 “最后写日期”， 以提供给用户， 并确定是否需要备份给定文件。

性能

有些系统有一块独立内存以用作缓冲区缓存假设其中的块将很快再次使用。 其他系统采用页面缓存 来缓存文件数据。 页面缓存采用虚拟内存技术，将文件数据按页面而不是按面向文件系统的块来缓存。 采用虚拟地址来缓存文件数据， 与采用物理磁盘块来缓存相比， 更为高效， 因为访问接口是通过虚拟内存而不是文件系统。

## 恢 复

统崩溃可能导致磁盘文件系统数据结构 （ 如目录结构、 空闲块指针和空闲 FCB指针） 的不一致。

一致性检查

对于检测， 每个文件系统的所有元数据的扫描可以肯定或否定系统的一致性。 不过， 这种扫描可能需要分钟或数小时， 而且在每次系统启动时都应进行。

文件系统可能在文件系统的元数据中记录其状态。 在任何元数据修改的开始， 设置状态位以表示元数据正在修改。 如果所有元数据的更新成功完成， 则文件系统可以清除位。 然而， 如果状态位保持置位， 则运行一致性检查程序。

基于日志的文件系统

日志可能是文件系统的一个单独的部分， 甚至在单独的磁盘上。 采用分开读 / 写磁头可以减少磁头竞争和寻道时间， 会更有效但也更复杂

如果系统崩溃， 日志文件可能包含零个或多个事务。 它包含的任何事务虽然已经由操作系统提交了， 但是还没有完成到文件系统， 所以现在必须完成。 交易可以从指针处执行， 直到工作完成， 因此文件系统结构仍能保持一致。

其他解决方法

保留旧的指针和块， 则创建了快照

ZFS 采用更为创新的方法来实现磁盘一致性。 就像 WAFL一样， 它从不会覆盖块。 然而， ZFS 更进一步， 它提供所有元数据和数据块的校验和。

备份和恢复

磁盘有时故障， 所以必须注意确保因故障而丢失的数据不会永远丢失。 为此， 可以采用系统程序将磁盘数据备份 到另一存储设备， 如磁带或其他硬盘。 单个文件或整个磁盘的恢复， 只需要从备份中恢复数据就可以了。

## NFS

概述

NFS 将一组互连的工作站视作一组具有独立文件系统的独立机器。 目的是，允许透明(根据显式请求) 共享这些文件系统。 共享是基于客户机- 服务器关系的。

安装协议

安装协议 在客户机和服务器之间建立初始逻辑连接。 在 Solaris 中，每台机器在内核外都有一个服务器进程来执行这个协议功能

NFS 协议

NFS 协议为远程文件提供了一组 RPC 操作。 这些程序包括以下操作

- 搜索目录内的文件
- 读取一组目录条目
- 操作链接和目录
- 访问文件属性
- 读写文件

路径名称转换

NFS 的路径名称转换， 将路径名称解析成单独的目录条目或组件