第16卷 第11期
2006年11月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol.16 No.11
Nov. 2006

Linux虚拟文件系统机制

廖光忠
(武汉科技大学 计算机科学与技术学院，湖北 武汉 430081)

摘 要：论述了Linux文件系统的逻辑关系和逻辑结构，对VFS中几个主要的数据结构进行了分析，剖析了Linux文件系统中一个物理文件系统的安装与注册、VFS的产生，以及通过VFS来管理和访问物理文件系统的内核工作机制。对开发及设计出适合不同需求的专用文件系统有着参考意义。

关键词：虚拟文件系统；超级块；安装；注册；访问
中图分类号：TP311 **文献标识码**：A **文章编号**：1673-629X(2006)11-0114-03

Research on Mechanism of Virtual File System of Linux

LIAO Guang-zhong
(Computer Sci. & Tech. School, Wuhan Univ. of Sci. and Tech., Wuhan 430081, China)

Abstract: Tells logical relation and structure of file system, gives several analysis of chief data structure. The installation and registration of a physical system and production of VFS and kernel working mechanism of managing and accessing through VFS physical file system in Linux deeply are analyzed. It has referenced significance to develop and design special file system which suits the different demand.

Key words: virtual file system; super block; installation; registration; access

1 虚拟文件系统

Linux支持各种不同的文件系统是通过VFS实现的，不同的物理文件系统具有不同的组织结构和不同的处理方式，为了能够处理各种不同的物理文件系统，操作系统必须把它们所具有的特性进行抽象，并建立一个面向各种物理文件系统的转换机制，通过这个转换机制，把各种不同物理文件系统转换为一个具有统一共性的虚拟文件系统[1]，VFS实际上向Linux内核和系统中运行的进程提供了一个处理各种物理文件系统的公共接口，通过这个接口使得不同的物理文件系统看来都是相同的。VFS并不是一种物理的文件系统，它仅是一套转换机制，它在系统启动时建立，在系统关闭时消失，并且仅存在于内存空间。所以，VFS并不具有一般物理文件系统的实体。在VFS提供的接口中包含向各种物理文件系统转换用的一系列数据结构，如VFS超级块、VFS的inode等，同时还包含对不同物理文件系统进行处理的各种操作函数的转换入口，为用户程序提供一个统一的、抽象的、虚拟的文件系统界面。

图1给出了Linux内核的虚拟文件系统与它的真实文件系统之间的关系。虚拟文件系统要管理在任何时间装配的所有不同的文件系统，所以它要维护一些描述整个虚拟文件系统和真实的被装配的文件系统的数据结构。与EXT2文件系统的inode节点一样，VFS的inode节点也用于描述系统中的文件、目录以及虚拟文件系统的内容、拓扑结构。

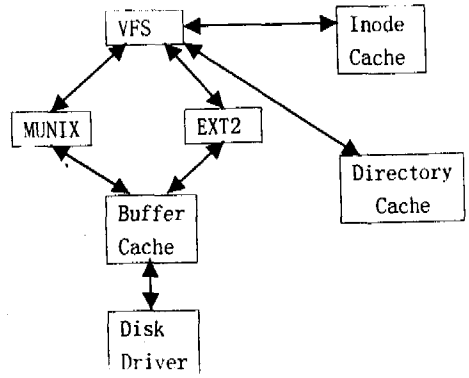

图1 文件系统的逻辑结构图

在每个文件系统初始化时，它向VFS进行注册。这个过程发生在系统启动操作系统自我初始化的过程中，真实的文件系统或者是安装在内核中的，或者是作为内核的可载入模块。文件系统模块只有在系统需要它们时才会被载入。每当包含文件系统的块设备被装配时(包括根文件系统)，VFS都会读入它的超级块。每种类型文件系统的超级块读例程必须要确定出整个文件系统的拓扑结构，并把这些信息映射到VFS超级块数据结构中[2]。VFS文件系统包含了系统中所有装配文件系统的列表和它们的VFS超级块信息。每个VFS超级块包含执行某些特殊功能的例程的信息和指向它们的指针。且包含一个指向文

收稿日期：2006-03-09
作者简介：廖光忠(1969-)，男，湖北武汉人，硕士，讲师，研究方向为计算机网络及信息安全、多媒体、系统软件开发。

件系统第一个 VFS inode 节点的指针。对根文件系统来说,这个指针指向的 inode 节点是用来表示根目录的。在系统进程访问目录和文件时,系统会调用搜索系统中 VFS inode 节点的进程。由于系统中的每个文件和目录都是由一个 VFS inode 节点表示的,所以有一些 inode 节点会被经常访问。这些经常被访问的 inode 节点被记录在缓存中以加速访问过程。如果要访问的 inode 节点不在 inode 缓存中,那么系统会调用文件系统的专门例程来读入相应的 inode 节点。

2 VFS 数据结构

2.1 VFS 文件系统的超级块

每个装配的文件系统由一个 VFS 超级块来表示[3],VFS 超级块中主要包括下列几个域:

·设备:该文件系统所在的块设备的设备标识。

·inode 指针:mounted inode 节点指针指向文件系统中的第一个 inode 节点。

·块长度:以字节为单位表示的该文件系统的块长度。

·超级块操作:指向该文件系统的一组超级块例程。

·文件系统类型:指向装配的文件系统

·file_system_type:数据结构的指针。

·文件系统特征:指向该文件系统所需信息的指针。

2.2 VFS 文件系统的 inode 节点

像 EXT2 文件系统一样,VFS 文件系统的每个文件、目录等对象都是由一个 VFS inode 节点表示的。每个 VFS inode 节点的信息都是由文件系统的专门例程从下层文件系统的信息中获得的。VFS inode 节点只存在于内核的存贮空间中,只要它们对系统有用就一直被记录在 VFS inode 节点的缓存中,VFS inode 节点包括下列域:

* 设备:该 VFS inode 节点表示的文件系统对象所在物理设备的标识。

* inode 号:inode 节点的节点号,在本文件系统中是唯一的。设备和 inode 号一起可以在 VFS 中唯一标识该 VFS inode 节点。

* 模式:像 EXT2 文件系统的这个域一样,它表示对 VFS inode 节点的访问权限。

* 用户标识:拥有者标识。

* 时间创建、更改和写的时间。

* 块长度:用字节为单位表示的本文件数据块的长度。

* inode 操作:指向例程地址块的指针。这些例程是该文件系统专有的,用于操作该 inode 节点。

* 计数:当前使用的本 VFS inode 节点的系统进程数,值为 0 表示本 inode 节点可以被自由地丢弃或重用。

* 锁:本域用于锁定 VFS inode 节点。例如当它被文件系统读出时,VFS 文件系统可以锁定它。

* 脏:表示该 VFS inode 节点是否被更改过。如果有改动,那么下层文件系统也要作相应的更改操作[4]。

3 VFS 实现机制

3.1 注册文件系统

在建立 Linux 内核时,可以选择支持的文件系统。在内核被建立后,文件系统的启动代码包含对所有安装在内核中的文件系统的初始化例程的调用。在文件系统模块被载入时,它向内核注册;在卸载时向内核注消。每个文件系统的初始化例程向 VFS 文件系统注册,并由一个 file_system_type 数据结构来表示。该数据结构中包含文件系统的名字和指向它的 VFS 超级块读例程的指针。每个 file_system_type 数据结构包含下列信息:

* 超级块读例程:这个例程在文件系统的实例被装配时由 VFS 文件系统调用。

* 文件系统名:该文件系统的名称,如 ext2。

* 所需设备:这个文件系统需要支持设备吗?并不是所有的文件系统都要有保存它们的设备。

3.2 装配文件系统

超级用户要装配文件系统时,Linux 内核必须先验证传入系统调用中的参数的有效性。例如:mount 命令会向内核传送三部分信息:文件系统名、包含该文件系统的块设备和新文件系统在当前文件系统拓扑结构中的装配位置。VFS 文件系统首先查找待装配的文件系统。它通过查看由 file_systems 指向的表中的每个 file_system_type 数据结构,来搜索已知文件系统的列表。如果它找到了一个匹配名,就知道该文件系统是内核所支持的。从 file_system_type 结构中,VFS 文件系统可以获得读该文件系统超级块的文件系统专有例程的地址。如果它找不到匹配的文件系统,那么只要内核支持按需载入内核模块的话,这时内核在继续处理之前会要求内核守护进程载入适当的文件系统模块。如果由 mount 命令传送来的物理设备还没有被装配的话,VFS 文件系统就必须要找到作为新文件系统的装配点的目录的 inode 节点。该 VFS inode 节点可能在 inode 缓存中,也可能从装配点文件系统的物理块设备上读出。一旦 VFS 找到了 inode 节点,就要检查该 VFS inode 节点是否代表着一个目录以及是否有其他的文件系统装配在这里。因为同一个目录不能作为一个以上文件系统的装配点。这时,VFS 文件系统的装配代码要分配一个 VFS 超级块,并把装配信息传递给这个文件系统的超级块读例程。所有系统的 VFS 超级块都被记录在 super_block 数据结构的 super_blocks 向量中,每次装配操作都必须分配一个超级块。超级块的读例程用从物理设备读取的信息填充 VFS 超级块的各域。对 EXT2 文件系统来说,这种信息的映射只是读入 EXT2 文件系统的超级块,并用它来填充 VFS 文件系统的超级块。无论什么类型的文件系统,填充 VFS 超级块意味着文件系统必须从它所在的块设备中读取描述信息,如果块设备无法读取或块设备中没有这种类型的文件系统,mount 命令就会失败。

每个装配的文件系统由 vfsmount 数据结构来表示。

它们被放在由 vfsmntlist 指向的队列链表中。另一个指针——vfsmntail 指向表中的最后一项，nmru_vfsmnt 指针指向最近被使用的文件系统。每个 vfsmount 结构包括记录该文件系统的块设备的设备号、文件系统的装配目录和文件系统装配时分配的 VFS 超级块的指针。每个 VFS 超级块除了包含指向其对应文件系统的根 inode 节点的指针外，还指向它对应文件系统的 file_system_type 数据结构。每个文件系统的 inode 节点在本文件系统加载后一直驻留在 VFS inode 节点的缓存中。

3.3 在虚拟文件系统中查找文件

为了在 VFS 文件系统中查找某个文件的 VFS inode 节点，VFS 文件系统必须以每次一个目录的方式来解析文件名，查找代表文件名中间目录的 VFS inode 节点。每个目录查找过程都包含对代表它父目录的 VFS inode 节点记录的文件系统专有的查找过程的调用。由于总是能通过该文件系统的超级块找到该文件系统的根 VFS inode 节点，所以上面的办法是可行的。每当真实文件系统查找一个 inode 节点时，系统都会先检查目录缓存。如果目录缓存中没有该目录项，那么真实文件系统可以通过下层的文件系统或者在 inode 缓存中找到要找的 VFS inode 节点。

3.4 卸载文件系统

如果系统的某些进程正在使用该文件系统的某个文件的话，该文件系统不能卸载。如果有某些进程正在使用要卸载的文件系统的话，在 VFS inode 节点的缓存中就会有来自该文件系统的 VFS inode 节点。检查程序通过在 inode 节点表中查找来自于该文件系统所在设备的 inode 节点可以检测出这个问题。如果装配的文件系统的 VFS 超级块被标记为"脏"，这说明该超级块被改动过，所以文件系统要把它写回到硬盘上的文件系统中。一旦超级块被回写到硬盘上，由 VFS 超级块占用的存贮区就被归还给内核的自由存贮区池。最后为装配操作建立的 vfsmount 数据结构与 vfsmntlist 解除链接并释放掉[5]。

4 VFS 的系统调用

在 Linux 的 VFS 中提供了具体文件系统统一的操作界面，依赖于 VFS 的多种操作函数的数据结构。在这些结构体中，文件系统共用的多个操作函数的函数指针组成结构体。属性的赋值由各个文件系统根据需要进行，以保证不同的文件系统都可以利用该数据结构实现具体的操作[3]。文件操作数据结构是实现对文件的搜索、读写、更新等操作。节点操作数据结构是实现节点的创建、查找、删除等操作。超级块操作函数实现的是超级块的对节点的读取、写回、释放、删除等操作和超级块的释放、写回、获取文件系统状态、再次安装等。目录项操作是目录缓冲区常用的操作函数。Linux 文件系统的系统调用实现了用户进程访问文件系统。VFS 提供给用户大量的系统调用，在 VFS 的源程序中，系统调用占据了大量的编码。Linux 内核源程序提供的系统调用统一的格式为：

```
asm linkge retm_type sys_name(argument)
{
……
}
```

5 结束语

Linux 文件系统是一个优秀的文件系统。它的文件系统与 UNIX 操作系统有很多共同点，也有自身的特点。该文详细分析了 Linux 文件系统的建立、撤销过程，包括了文件系统的注册和注销，文件系统的安装和卸载过程。在整个文件系统的建立过程中，利用了文件系统类型链表和文件系统安装链表两个链表型数据结构。对文件系统的建立和撤消过程的研究，对于理解 Linux 文件系统的工作流程和实现方法起着关键的作用。

参考文献：

[1] 毛德操，胡希明. Linux 内核源代码情景分析(上册)[M]. 杭州：浙江大学出版社，2001.

[2] 胡晓明，王 强. Linux 核心源代码分析[M]. 北京：人民邮电出版社，2000：50－100.

[3] 陈莉君. Linux 操作系统内核分析[M]. 北京：人民邮电出版社，2000.

[4] 韩德志，钟 铭. Windows NT 文件系统驱动程序原理及设计方法[J]. 微机发展，2001，11(4)：63－65.

[5] 顾喜梅，顾宝根. Linux 虚拟文件系统实现机制研究[J]. 微机发展，2002，12(1)：60－62.

(上接第 113 页)

4 结束语

通过在原理和特点上与其它同类技术进行比较，得出在实际开发中 Hibernate 的优势。Hibernate 框架技术将开发人员从繁琐的代码中解放出来，使软件项目的开发效率大大提高。相信随着技术的不断改进，Hibernate 必将有更大的发展。

参考文献：

[1] 王 鑫. 基于 Hibernate 的 O/R 映射数据持久化的研究[J]. 中南民族大学学报：自然科学版，2005(3)：85－88.

[2] 梁添才，皮佑国. J2EE 数据持久层的 ORM 设计模式[J]. 深圳信息职业技术学院学报，2005(Z1)：22－26.

[3] 方 巍，孙 涌，张书奎. 整合 Struts 和 Hibernate 的 Web 系统应用[J]. 计算机与现代化，2005(12)：43－45.

[4] 田 珂，谢世波，方 马. J2EE 数据持久层的解决方案[J]. 计算机工程，2003(22)：94－96.

[5] Ebersole S. Hibernate Core for Java[EB/OL]. 2006－01－28. http://www.hibernage.org.