**Linux 文件系统剖析**

## 基本的文件系统体系结构

Linux 文件系统体系结构是一个对复杂系统进行抽象化的有趣例子。通过使用一组通用的 API 函数，Linux 可以在许多种存储设备上支持许多种文件系统。例如，read 函数调用可以从指定的文件描述符读取一定数量的字节。read 函数不了解文件系统的类型，比如 ext3 或 NFS。它也不了解文件系统所在的存储媒体，比如 AT Attachment Packet Interface（ATAPI）磁盘、Serial-Attached SCSI（SAS）磁盘或 Serial Advanced Technology Attachment（SATA）磁盘。但是，当通过调用 read 函数读取一个文件时，数据会正常返回。本文讲解这个机制的实现方法并介绍 Linux 文件系统层的主要结构。

[**回页首**](https://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/#ibm-pcon)

## 什么是文件系统？

首先回答最常见的问题，“什么是文件系统”。文件系统是对一个存储设备上的数据和元数据进行组织的机制。由于定义如此宽泛，支持它的代码会很有意思。正如前面提到的，有许多种文件系统和媒体。由于存在这么多类型，可以预料到 Linux 文件系统接口实现为分层的体系结构，从而将用户接口层、文件系统实现和操作存储设备的驱动程序分隔开。

### 挂装

在 Linux 中将一个文件系统与一个存储设备关联起来的过程称为挂装（mount）。使用 mount 命令将一个文件系统附着到当前文件系统层次结构中（根）。在执行挂装时，要提供文件系统类型、文件系统和一个挂装点。

为了说明 Linux 文件系统层的功能（以及挂装的方法），我们在当前文件系统的一个文件中创建一个文件系统。实现的方法是，首先用 dd 命令创建一个指定大小的文件（使用 /dev/zero 作为源进行文件复制）—— 换句话说，一个用零进行初始化的文件，见清单 1。

##### 清单 1. 创建一个经过初始化的文件

$ **dd if=/dev/zero of=file.img bs=1k count=10000**

10000+0 records in

10000+0 records out

$

现在有了一个 10MB 的 file.img 文件。使用 losetup 命令将一个循环设备与这个文件关联起来，让它看起来像一个块设备，而不是文件系统中的常规文件：

$ **losetup /dev/loop0 file.img**

$

这个文件现在作为一个块设备出现（由 /dev/loop0 表示）。然后用 mke2fs 在这个设备上创建一个文件系统。这个命令创建一个指定大小的新的 ext2 文件系统，见清单 2。

##### 清单 2. 用循环设备创建 ext2 文件系统

$ **mke2fs -c /dev/loop0 10000**

mke2fs 1.35 (28-Feb-2004)

max\_blocks 1024000, rsv\_groups = 1250, rsv\_gdb = 39

Filesystem label=

OS type: Linux

Block size=1024 (log=0)

Fragment size=1024 (log=0)

2512 inodes, 10000 blocks

500 blocks (5.00%) reserved for the super user

...

$

使用 mount 命令将循环设备（/dev/loop0）所表示的 file.img 文件挂装到挂装点 /mnt/point1。注意，文件系统类型指定为 ext2。挂装之后，就可以将这个挂装点当作一个新的文件系统，比如使用 ls 命令，见清单 3。

##### 清单 3. 创建挂装点并通过循环设备挂装文件系统

$ **mkdir /mnt/point1**

$ **mount -t ext2 /dev/loop0 /mnt/point1**

$ **ls /mnt/point1**

lost+found

$

如清单 4 所示，还可以继续这个过程：在刚才挂装的文件系统中创建一个新文件，将它与一个循环设备关联起来，再在上面创建另一个文件系统。

##### 清单 4. 在循环文件系统中创建一个新的循环文件系统

$ **dd if=/dev/zero of=/mnt/point1/file.img bs=1k count=1000**

1000+0 records in

1000+0 records out

$ **losetup /dev/loop1 /mnt/point1/file.img**

$ **mke2fs -c /dev/loop1 1000**

mke2fs 1.35 (28-Feb-2004)

max\_blocks 1024000, rsv\_groups = 125, rsv\_gdb = 3

Filesystem label=

...

$ **mkdir /mnt/point2**

$ **mount -t ext2 /dev/loop1 /mnt/point2**

$ **ls /mnt/point2**

lost+found

$ **ls /mnt/point1**

file.img lost+found

$

通过这个简单的演示很容易体会到 Linux 文件系统（和循环设备）是多么强大。可以按照相同的方法在文件上用循环设备创建加密的文件系统。可以在需要时使用循环设备临时挂装文件，这有助于保护数据。

[**回页首**](https://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/#ibm-pcon)

## 文件系统体系结构

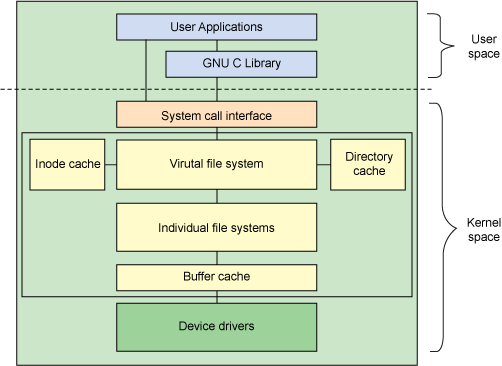
既然已经看到了文件系统的构造方法，现在就看看 Linux 文件系统层的体系结构。本文从两个角度考察 Linux 文件系统。首先采用高层体系结构的角度。然后进行深层次讨论，介绍实现文件系统层的主要结构。

[**回页首**](https://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/#ibm-pcon)

## 高层体系结构

尽管大多数文件系统代码在内核中（后面讨论的用户空间文件系统除外），但是图 1 所示的体系结构显示了用户空间和内核中与文件系统相关的主要组件之间的关系。

##### 图 1. Linux 文件系统组件的体系结构



用户空间包含一些应用程序（例如，文件系统的使用者）和 GNU C 库（glibc），它们为文件系统调用（打开、读取、写和关闭）提供用户接口。系统调用接口的作用就像是交换器，它将系统调用从用户空间发送到内核空间中的适当端点。

VFS 是底层文件系统的主要接口。这个组件导出一组接口，然后将它们抽象到各个文件系统，各个文件系统的行为可能差异很大。有两个针对文件系统对象的缓存（inode 和 dentry）。它们缓存最近使用过的文件系统对象。

每个文件系统实现（比如 ext2、JFS 等等）导出一组通用接口，供 VFS 使用。缓冲区缓存会缓存文件系统和相关块设备之间的请求。例如，对底层设备驱动程序的读写请求会通过缓冲区缓存来传递。这就允许在其中缓存请求，减少访问物理设备的次数，加快访问速度。以最近使用（LRU）列表的形式管理缓冲区缓存。注意，可以使用 sync 命令将缓冲区缓存中的请求发送到存储媒体（迫使所有未写的数据发送到设备驱动程序，进而发送到存储设备）。

这就是 VFS 和文件系统组件的高层情况。现在，讨论实现这个子系统的主要结构。

## 主要结构

Linux 以一组通用对象的角度看待所有文件系统。这些对象是超级块（superblock）、inode、dentry 和文件。超级块在每个文件系统的根上，超级块描述和维护文件系统的状态。文件系统中管理的每个对象（文件或目录）在 Linux 中表示为一个 inode。inode 包含管理文件系统中的对象所需的所有元数据（包括可以在对象上执行的操作）。另一组结构称为 dentry，它们用来实现名称和 inode 之间的映射，有一个目录缓存用来保存最近使用的 dentry。dentry 还维护目录和文件之间的关系，从而支持在文件系统中移动。最后，VFS 文件表示一个打开的文件（保存打开的文件的状态，比如写偏移量等等）。

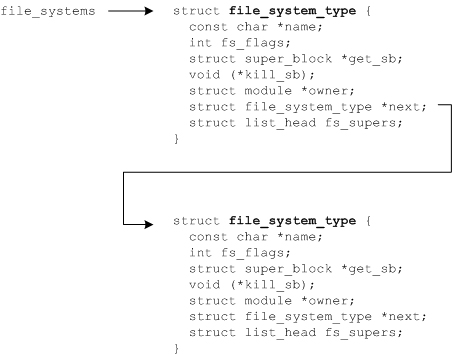
### 虚拟文件系统层

VFS 作为文件系统接口的根层。VFS 记录当前支持的文件系统以及当前挂装的文件系统。

可以使用一组注册函数在 Linux 中动态地添加或删除文件系统。内核保存当前支持的文件系统的列表，可以通过 /proc 文件系统在用户空间中查看这个列表。这个虚拟文件还显示当前与这些文件系统相关联的设备。在 Linux 中添加新文件系统的方法是调用 register\_filesystem。这个函数的参数定义一个文件系统结构（file\_system\_type）的引用，这个结构定义文件系统的名称、一组属性和两个超级块函数。也可以注销文件系统。

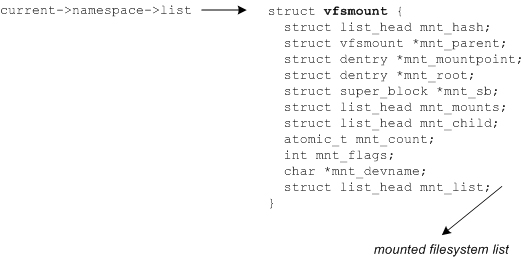
在注册新的文件系统时，会把这个文件系统和它的相关信息添加到 file\_systems 列表中（见图 2 和 linux/include/linux/mount.h）。这个列表定义可以支持的文件系统。在命令行上输入 cat /proc/filesystems，就可以查看这个列表。

##### 图 2. 向内核注册的文件系统



VFS 中维护的另一个结构是挂装的文件系统（见图 3）。这个结构提供当前挂装的文件系统（见 linux/include/linux/fs.h）。它链接下面讨论的超级块结构。

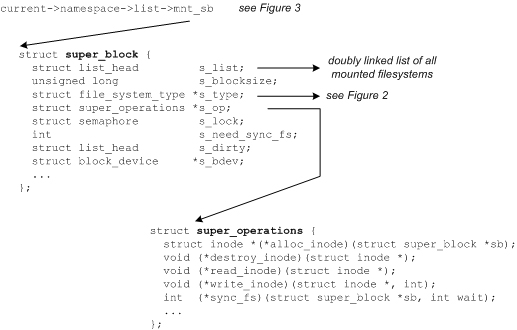
##### 图 3. 挂装的文件系统列表



### 超级块

超级块结构表示一个文件系统。它包含管理文件系统所需的信息，包括文件系统名称（比如 ext2）、文件系统的大小和状态、块设备的引用和元数据信息（比如空闲列表等等）。超级块通常存储在存储媒体上，但是如果超级块不存在，也可以实时创建它。可以在 ./linux/include/linux/fs.h 中找到超级块结构（见图 4）。

##### 图 4. 超级块结构和 inode 操作

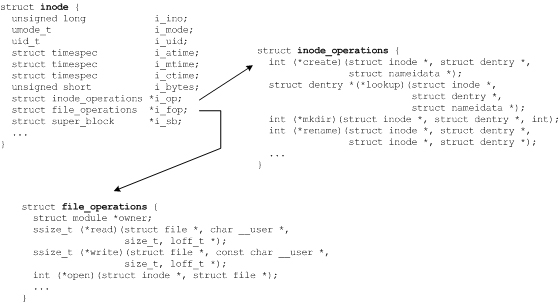


超级块中的一个重要元素是超级块操作的定义。这个结构定义一组用来管理这个文件系统中的 inode 的函数。例如，可以用 alloc\_inode 分配 inode，用 destroy\_inode 删除 inode。可以用 read\_inode 和 write\_inode 读写 inode，用 sync\_fs 执行文件系统同步。可以在 ./linux/include/linux/fs.h 中找到 super\_operations 结构。每个文件系统提供自己的 inode 方法，这些方法实现操作并向 VFS 层提供通用的抽象。

### inode 和 dentry

inode 表示文件系统中的一个对象，它具有惟一标识符。各个文件系统提供将文件名映射为惟一 inode 标识符和 inode 引用的方法。图 5 显示 inode 结构的一部分以及两个相关结构。请特别注意 inode\_operations 和 file\_operations。这些结构表示可以在这个 inode 上执行的操作。inode\_operations 定义直接在 inode 上执行的操作，而 file\_operations 定义与文件和目录相关的方法（标准系统调用）。

##### 图 5. inode 结构和相关联的操作



inode 和目录缓存分别保存最近使用的 inode 和 dentry。注意，对于 inode 缓存中的每个 inode，在目录缓存中都有一个对应的 dentry。可以在 ./linux/include/linux/fs.h 中找到 inode 和 dentry 结构。

### 缓冲区缓存

除了各个文件系统实现（可以在 ./linux/fs 中找到）之外，文件系统层的底部是缓冲区缓存。这个组件跟踪来自文件系统实现和物理设备（通过设备驱动程序）的读写请求。为了提高效率，Linux 对请求进行缓存，避免将所有请求发送到物理设备。缓存中缓存最近使用的缓冲区（页面），这些缓冲区可以快速提供给各个文件系统。

[**回页首**](https://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/#ibm-pcon)

## 有趣的文件系统

本文没有讨论 Linux 中可用的具体文件系统，但是值得在这里稍微提一下。Linux 支持许多种文件系统，包括 MINIX、MS-DOS 和 ext2 等老式文件系统。Linux 还支持 ext3、JFS 和 ReiserFS 等新的日志型文件系统。另外，Linux 支持加密文件系统（比如 CFS）和虚拟文件系统（比如 /proc）。

最后一种值得注意的文件系统是 Filesystem in Userspace（FUSE）。这种文件系统可以将文件系统请求通过 VFS 发送回用户空间。所以，如果您有兴趣创建自己的文件系统，那么通过使用 FUSE 进行开发是一种不错的方法。

[**回页首**](https://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/#ibm-pcon)

## 结束语

尽管文件系统的实现并不复杂，但它是可伸缩和可扩展的体系结构的好例子。文件系统体系结构已经发展了许多年，并成功地支持了许多不同类型的文件系统和许多目标存储设备类型。由于使用了基于插件的体系结构和多层的函数间接性，Linux 文件系统在近期的发展很值得关注。

## 参考资料

### 学习

* 您可以参阅本文在 developerWorks 全球站点上的 [英文原文](http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/?S_TACT=105AGX52&S_CMP=cn-a-l)。
* proc 文件系统提供了一种通过虚拟文件系统在用户空间和内核之间进行通信的新方式。“[使用 /proc 文件系统来访问 Linux 内核的内容](http://www.ibm.com/developerworks/cn/linux/l-proc.html)”（developerWorks，2006 年 3 月）介绍了 /proc 虚拟文件系统并演示它的使用方法。
* Linux 系统调用接口可以在用户空间和内核之间传递控制，从而调用内核 API 函数。“[使用 Linux 系统调用的内核命令](http://www.ibm.com/developerworks/cn/linux/l-system-calls/)”（developerWorks，2007 年）介绍了 Linux 系统调用接口。
* [Yolinux.com](http://www.yolinux.com/TUTORIALS/LinuxClustersAndFileSystems.html) 维护 Linux 文件系统、集群文件系统和性能计算集群的列表。还可以在 [File systems HOWTO](http://tldp.org/HOWTO/Filesystems-HOWTO.html) 中找到 Linux 文件系统的完整列表。[Xenotime](http://www.xenotime.net/linux/linux-fs.html) 也描述了许多文件系统。
* 关于 Linux 用户空间编程的更多信息，请参考本文作者所著的 [*GNU/Linux Application Programming*](http://www.charlesriver.com/Books/BookDetail.aspx?productID=91525)。
* 在 [developerWorks Linux 专区](http://www.ibm.com/developerworks/cn/linux/) 中可以找到为 Linux 开发人员准备的更多参考资料，还可以查阅 [最流行的文章和教程](http://www.ibm.com/developerworks/cn/linux/top10/index.html)。
* 查阅 developerWorks 上的所有 [Linux 技巧](http://www.ibm.com/developerworks/cn/views/linux/articles.jsp?view_by=search&search_by=linux+%E6%8A%80%E5%B7%A7) 和 [Linux 教程](http://www.ibm.com/developerworks/cn/views/linux/tutorials.jsp)。
* 随时关注 [developerWorks 技术活动和网络广播](http://www.ibm.com/developerworks/offers/techbriefings/?S_TACT=105AGX52&S_CMP=cn-a-l)。

### 获得产品和技术

* [Filesystem in Userspace](http://fuse.sourceforge.net/)（FUSE）是一个支持在用户空间中开发文件系统的内核模块。文件系统驱动程序实现将来自 VFS 的请求发送回用户空间。这是一种在不借助内核开发的情况下开发文件系统的好方法。如果您精通 Python，也可以通过 [LUFS-Python](http://simonwillison.net/2003/Dec/10/pythonFilesystems/) 用 Python 编写文件系统。
* 下载 [IBM 产品评估版](http://www.ibm.com/developerworks/downloads/?S_TACT=105AGX52&S_CMP=cn-a-l)，试用来自 DB2®、Lotus®、Rational®、Tivoli® 和 WebSphere® 的应用程序开发工具和中间件产品。

### 讨论

* 通过 [新的 developerWorks 空间](http://www.ibm.com/developerworks/spaces/?S_TACT=105AGX52&S_CMP=cn-a-l) 中的 blog、论坛、podcast 和社区主题，加入[developerWorks 社区](http://www.ibm.com/developerworks/community)。