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。

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

```
1    $ dd if=/dev/zero of=file.img bs=1k count=10000
2    10000+0 records in
3    10000+0 records out
4    $
```

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：

```
1    $ losetup /dev/loop0 file.img
2    $
```

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

```
1    $ mke2fs -c /dev/loop0 10000
2    mke2fs 1.35 (28-Feb-2004)
3    max_blocks 1024000, rsv_groups = 1250, rsv_gdb = 39
4    Filesystem label=
5    OS type: Linux
6    Block size=1024 (log=0)
7    Fragment size=1024 (log=0)
8    2512 inodes, 10000 blocks
9    500 blocks (5.00%) reserved for the super user
10   ...
11   $
```

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

```
1    $ mkdir /mnt/point1
2    $ mount -t ext2 /dev/loop0 /mnt/point1
3    $ ls /mnt/point1
4    lost+found
5    $
```

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

```
1    $ dd if=/dev/zero of=/mnt/point1/file.img bs=1k count=1000
2    1000+0 records in
3    1000+0 records out
4    $ losetup /dev/loop1 /mnt/point1/file.img
5    $ mke2fs -c /dev/loop1 1000
6    mke2fs 1.35 (28-Feb-2004)
7    max_blocks 1024000, rsv_groups = 125, rsv_gdb = 3
8    Filesystem label=
9    ...
10   $ mkdir /mnt/point2
11   $ mount -t ext2 /dev/loop1 /mnt/point2
12   $ ls /mnt/point2
13   lost+found
14   $ ls /mnt/point1
15   file.img lost+found
16   $
```

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。

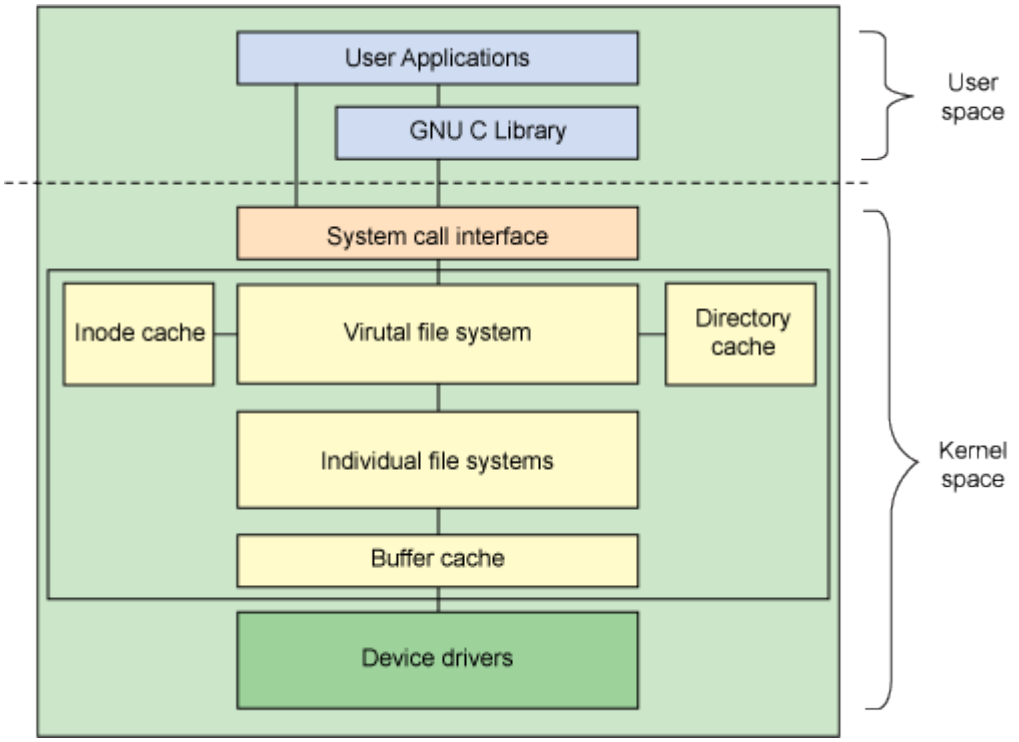
文件系统体系结构

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。

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）。