中国矿业大学计算机学院

2019-2020(2)本科生 Linux 操作系统课程作业

内容范围 文件系统

指 标 点 1.2 占 比 50%

学生姓名 袁孝健 学 号 06172151

专业班级 信息安全 2017-01 班

任课教师 杨东平

课程基础理论掌握程度	熟练 □	较熟练 □	一般 □	不熟练 □
综合知识应用能力	强 □	较强 □	一般 □	差 □
作业内容	完整 □	较完整 □	一般 □	不完整 □
作业格式	规范 □	较规范 □	一般 □	不规范 □
作业完成状况	好 □	较好 □	一般 □	差 □
工作量	饱满 □	适中 □	一般 □	欠缺 □
学习、工作态度	好 □	较好 □	一般 □	差 □
抄袭现象	无 □	有 □ 姓名:		
存在问题				
总体评价				

综合成绩: 任课教师签字: 年 月 日

目　　录

1 详述 Linux 的节点 inode

1.1 什么是 inode

inode 译成中文就是索引节点，它用来存放档案及目录的基本信息，包含时间、档名、使用者及群组等。

我们知道文件是存储在硬盘上的，而硬盘的最小单位叫做"扇区"，每个扇区存储 512 字节；多个（通常是连续的八个）这样的扇区组成"块"，而块则是文件存取的最小单位。操作系统为了提高效率，在读取硬盘的时候，通常一次性读取多个"扇区"，也就是一个"块"。因此文件数据实际上都存储在"块"中，那么也就需要用一个区域来存储这些文件的信息，这种储存文件元信息的区域就叫做 inode。

实际上文件系统在创建时，就会把存储区域分为两大连续的存储区域。一个用来保存文件系统对象的元信息数据，即由 inode 组成的表，每个 inode 默认是 256 字节或者 128 字节。另一个用来保存"文件系统对象"的内容数据，即划分的"扇区"以及由扇区组成的"块"。一个文件系统的 inode 的总数是固定的。这限制了该文件系统所能存储的文件系统对象的总数目。典型的实现下，所有 inode 占用了文件系统 1%左右的存储容量。

1.2 inode 包含的信息

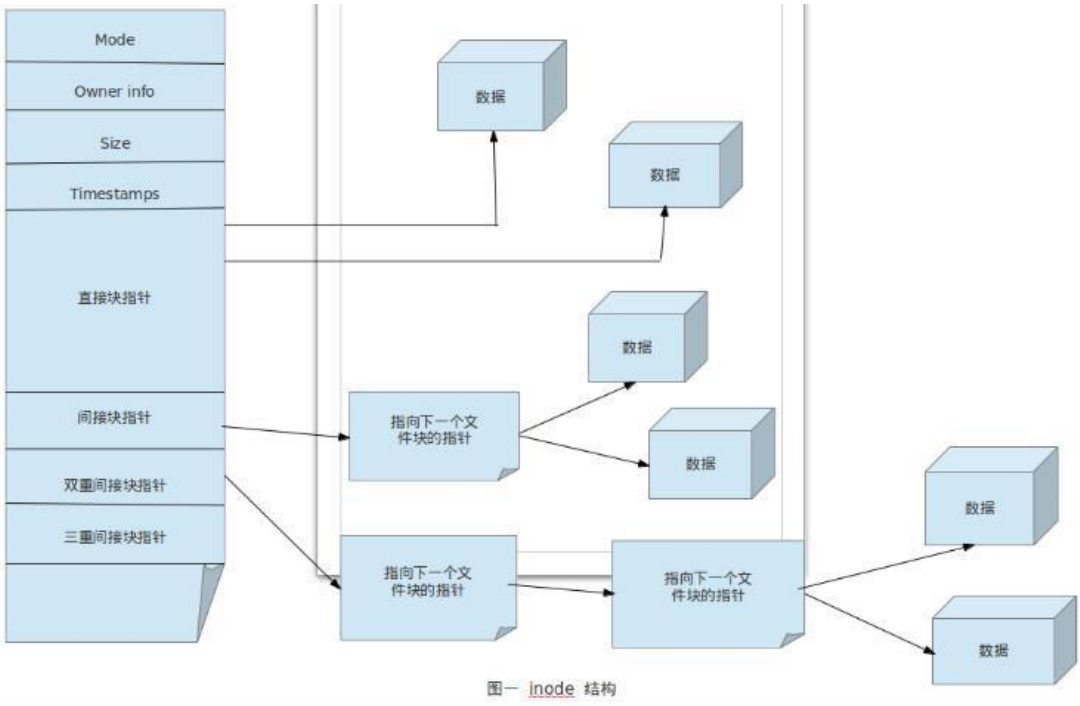
POSIX 标准强制规范了文件系统的行为。每个"文件系统对象"必须具有如下信息，即 inode 中包含的文件信息：

- 以字节为单位表示的文件大小。
- 设备 ID，标识容纳该文件的设备。
- 文件所有者的 User ID。
- 文件的 Group ID
- 文件的模式（mode），确定了文件的类型，以及它的所有者、它的 group、其它用户访问此文件的权限。
- 额外的系统与用户标志（flag），用来保护该文件。
- 3 个时间戳，记录了 inode 自身被修改（ctime, inode change time）、文件内容被修改（mtime, modification time）、最后一次访问（atime, access time）的时间。
- 1 个链接数，表示有多少个硬链接指向此 inode。
- 到文件系统存储位置的指针。通常是 1K 字节或者 2K 字节的存储容量为基本单位。

在 Linux 下可以用 stat 命令，查看某个文件的 inode 信息：

```
ubuntu@VM-0-14-ubuntu:~/learn$ stat cumt.txt
  File: cumt.txt
  Size: 38              Blocks: 8          IO Block: 4096   regular file
Device: fc01h/64513d    Inode: 272437      Links: 1
Access: (0664/-rw-rw-r--)  Uid: (  500/  ubuntu)   Gid: (  500/  ubuntu)
Access: 2020-04-07 17:17:33.125509341 +0800
Modify: 2020-04-07 17:17:31.257509650 +0800
Change: 2020-04-07 17:17:31.269509648 +0800
 Birth: -
```

1.3 inode 表结构



图一 inode 结构

（1）直接块指针：

前 12 个直接指针，直接指向存储数据的区域。如 Blocks 大小为4×1024KB，前 12 个直接指针就可以保存 48KB 的文件。

（2）间接块指针:

设每个指针占用 4 个字节,则以及指针指向的 Blocks 可以保存$\left(\frac{4\times1024}{4}\right)$KB,可指向 1024 个 Blocks，一级指针可存储文件数据大小为$1024 \times (4 \times 1024)\text{KB} = 4\text{MB}$。

（3）双重间接块指针：

同样Blocks大小为4×1024,则二级指针可保存Blocks指针数量为$\left(\frac{4\times1024}{4}\right) \times \left(\frac{4\times1024}{4}\right)$,则二级指针保存文件数据大小为$(1024 \times 1024) \times (4 \times 1024) = 4\text{GB}$。

（4）三重间接块指针：

以次类推三级指针可以储存文件数据大小为$(1024 \times 4 \times 1024 \times 1024) \times (4 \times 1024) = 4TB$

1.4 inode 的大小

inode 的大小一般是 128 字节或 256 字节，一般每 1KB 或每 2KB 就设置一个 inode。假定在一块 1GB 的硬盘中，每个 inode 节点的大小为 128 字节，每 1KB 就设置一个 inode，那么 inode table 的大小就会达到 128MB，占整块硬盘的 12.8%。

查看硬盘分区的 inode 总数和已使用的数量，可以使用 df 命令(-i 显示 inode 信息)：

```
ubuntu@VM-0-14-ubuntu:~$ df -i
Filesystem      Inodes  IUsed   IFree IUse% Mounted on
udev            226573    400  226173    1% /dev
tmpfs           234688   1963  232725    1% /run
/dev/vda1      3276800 224823 3051977    7% /
tmpfs           234688     13  234675    1% /dev/shm
tmpfs           234688      5  234683    1% /run/lock
tmpfs           234688     18  234670    1% /sys/fs/cgroup
tmpfs           234688     10  234678    1% /run/user/500
```

1.5 inode 号码

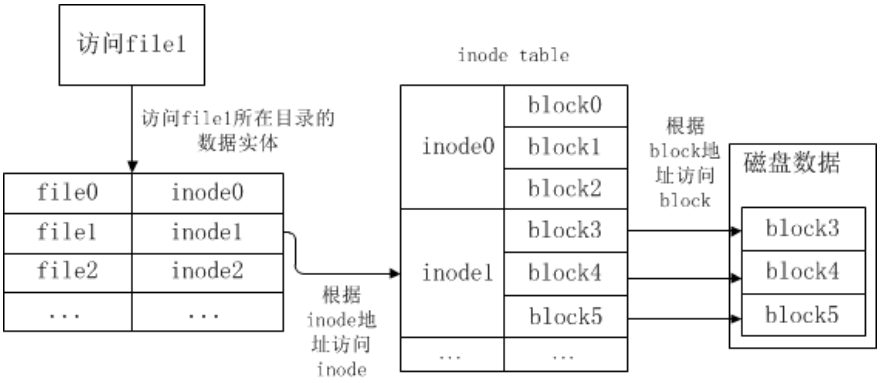
每个 inode 都有一个号码，Unix/Linux 系统内部使用 inode 号码（而非文件名）来识别不同文件。

当用户通过文件名打开文件时，实际上系统内部的过程如下：

① 找到改文件名对应的 inode 号码；

② 通过 inode 号码获取 inode 信息（包括文件数据所在的 block）；

③ 读出数据。



ls -i 命令列出整个目录文件，即文件名和 inode 号(箭头标的即为 inode 号)：

```
ubuntu@VM-0-14-ubuntu:~/learn$ ls -i
267808 1.sh               272437 cumt.txt           272425 nameedpipe_write
267276 beexec             267794 doexec             272430 nameedpipe_write.c
267818 beexec.c           267820 doexec.c           272426 sort.sh
267798 comprehensive      272421 guess.sh           272427 test2.sh
267799 comprehensive2     272429 judgeAlive.sh      272428 test.sh
272434 comprehensive2.c   272432 nameedpipe_read    272424 transform.sh
272422 comprehensive.c    272431 nameedpipe_read.c
ubuntu@VM-0-14-ubuntu:~/learn$ ls -i cumt.txt
272437 cumt.txt
```

1.6 inode 的优点

（1）对于有些无法删除的文件可以通过删除 inode 节点来删除；

（2）移动或者重命名文件，只是改变了目录下的文件名到 inode 的映射，并不需要实际对硬盘操作；

（3）删除文件的时候，只需要删除 inode，不需要实际清空那块硬盘，只需要在下次写入的时候覆盖即可（这也是为什么删除了数据可以进行数据恢复的原因之一）；

（4）打开一个文件后，只需要通过 inode 来识别文件。

2 详述硬链接与软链接

2.1 概述

在 Linux 下面的连接文件有两种——软连接和硬连接，虽然都是连接文件，但两者却有很大的区别。一种是类似于 Windows 的快捷方式功能的文件（或目录），这种连接称为软连接；另一种则是通过文件系统的 inode 连接来产生新文件名，而不是产生新文件，这种称为硬连接。

创建连接文件就是使用 ln 命令：

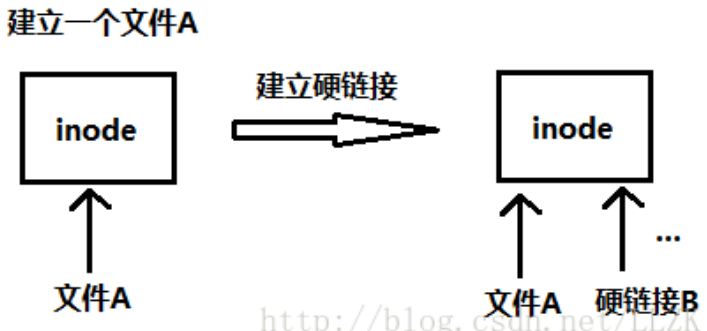
- 创建硬连接为 ln file1 file2，其中 file2 为 file1 的硬连接。
- 创建软连接为 ln -s file1 file2，其中 file2 为 file1 的软连接。

2.2 硬链接

2.2.1 含义

我们知道，在 Linux 下，每个文件都会占用一个 inode，文件内容由 inode 的记录来指向,而想要读取文件,必须要经过目录记录的文件名来指向正确的 inode 号码才能读取，也就是说，其实文件名只与目录有关，但是文件的内容则与 inode 有关。

而硬链接就是在某个目录下新建一条文件名连接到某 inode 号码的关联记录而已，简单来说，就是有多个文件名（不是多个文件）对应到同一个 inode 号码。



2.2.2 特点

（1）不论是修改源文件，还是修改硬链接文件，另一个文件中的数据都会发生改变。

（2）不论是删除源文件，还是删除硬链接文件，只要还有一个文件存在，这个文件都可以被访问。

（3）硬链接不会建立新的 inode 信息，也不会更改 inode 的总数。

（4）硬链接不能跨文件系统（分区）建立，因为在不同的文件系统中，inode 号是重新计算的。

（5）硬链接不能链接目录，因为如果给目录建立硬链接，那么不仅目录本身需要重新建立，目录下所有的子文件，包括子目录中的所有子文件都需要建立硬链接，这对当前的 Linux 来讲过于复杂。

（6）硬链接不会占用 inode 和 block。

2.2.3 具体演示

（1）源文件与硬链接访问到的文件内容相同：

```
ubuntu@VM-0-14-ubuntu:~/learn$ echo "I'm a student from CUMT." > test_file
ubuntu@VM-0-14-ubuntu:~/learn$ ln test_file hard_link_test
ubuntu@VM-0-14-ubuntu:~/learn$ cat hard_link_test
I'm a student from CUMT.
```

（2）源文件与硬链接的 inode 相同：

```
ubuntu@VM-0-14-ubuntu:~/learn$ ls -i test_file hard_link_test
267804 hard_link_test  267804 test_file
```

（3）修改某一个 inode 对应的文件内容，则该 inode 对应的所有文件名的内容均改变：

```
ubuntu@VM-0-14-ubuntu:~/learn$ echo "Hello,World!" > hard_link_test
ubuntu@VM-0-14-ubuntu:~/learn$ cat test_file
Hello,World!
```

（4）直到 inode 对应的所有文件名均被删除，该文件内容才会被删除：

```
ubuntu@VM-0-14-ubuntu:~/learn$ rm test_file
ubuntu@VM-0-14-ubuntu:~/learn$ cat hard_link_test
Hello,World!
```

2.2.4 查找文件

当我们查找一个硬链接文件，如/root/hard_link_test 时，要经过以下步骤：

（1）首先找到根目录的 inode（根目录的 inode 号系统已知是 2），然后判断用户是否有权限访问根目录的 block。

（2）如果有权限，则可以在根目录的 block 中访问到/root 的文件名及对应的 inode 号。

（3）通过/root/目录的 inode 号，可以查找到/root/目录的 inode 信息，接着判断用户是否有权限访问/root/目录的 block。

（4）如果有权限，则可以从/root/目录的 block 中读取到 hard_link_test 文件的文件名及对应的 inode 号。

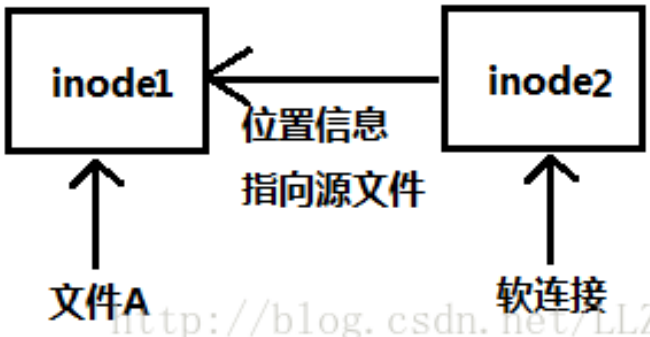
（5）通过 hard_link_test 文件的 inode 号，就可以找到 hard_link_test 文件的 inode 信息，接着判断用户是否有权限访问 hard_link_test 文件的 block。

（6）如果有权限，则可以读取 block 中的数据，即完成了/root/hard_link_test 文件的读取与访问。

2.3 软链接

2.3.1 含义

相对于硬连接，软链接（symbolic link），软链接基本上就是在创建一个独立的文件（拥有自己的 inode 号），而这个文件会让数据的读取指向指向它连接的那个文件的文件名，即只是利用文件来作为指向的操作。可以理解为，软链接文件也是一个文本文件，不过它里面包含了源文件的位置信息。所以，当源文件被删除后，软链接会“打不开”。



2.3.2 特点

（1）不论是修改源文件，还是修改软链接文件，另一个文件中的数据都会发生改变。

（2）删除软链接文件，源文件不受影响。而删除原文件，软链接文件将找不到实际的数据，从而显示文件不存在。

（3）软链接会新建自己的 inode 信息和 block，只是在 block 中不存储实际文件数据，而存储的是源文件的文件名及 inode 号。

（4）软链接可以链接目录。

（5）软链接可以跨分区。

（6）软链接会占用 inode 和 block。

2.3.3 具体演示

（1）源文件与软链接访问到的文件内容相同：

```
ubuntu@VM-0-14-ubuntu:~/learn$ echo "I'am a student from CUMT." > test_file
ubuntu@VM-0-14-ubuntu:~/learn$ ln -s test_file soft_link_test
ubuntu@VM-0-14-ubuntu:~/learn$ cat soft_link_test
I'am a student from CUMT.
```

（2）修改源文件或软链接的内容，其链接的文件内容也会更改：

```
ubuntu@VM-0-14-ubuntu:~/learn$ echo "Hello World" > soft_link_test
ubuntu@VM-0-14-ubuntu:~/learn$ cat test_file
Hello World
ubuntu@VM-0-14-ubuntu:~/learn$ echo "Hello" > test_file
ubuntu@VM-0-14-ubuntu:~/learn$ cat soft_link_test
Hello
```

（3）源文件与软链接各自拥有不同的 inode：

```
ubuntu@VM-0-14-ubuntu:~/learn$ ls -i test_file soft_link_test
267812 soft_link_test  267804 test_file
```

（4）删除源文件，则软链接失效无法打开：

```
ubuntu@VM-0-14-ubuntu:~/learn$ cat soft_link_test
cat: soft_link_test: No such file or directory
```

2.3.4 查找文件

当我们查找一个软链接文件，如/root/soft_link_test 时，要经过以下步骤：

（1）首先找到根目录的 inode 索引信息，然后判断用户是否有权限访问根目录的 block。

（2）如果有权限访问根目录的 block，就会在 block 中查找到/root/目录的 inode 号。

（3）接着访问/root/目录的 inode 信息，判断用户是否有权限访问/root/目录的 block。

（4）如果有权限，就会在 block 中读取到软链接文件 soft_link_test 的 inode 号。

（5）通过软链接文件的 inode 号，找到了 soft_link_test 文件 inode 信息，判断用户是否有权限访问 block。

（6）如果有权限，就会发现 soft_link_test 文件的 block 中没有实际数据，仅有源文件 test 的 inode 号。

（7）接着通过源文件的 inode 号，访问到源文件 test 的 inode 信息，判断用户是否有权限访问 block。

（8）如果有权限，就会在 test 文件的 block 中读取到真正的数据，从而完成数据访问。

2.4 硬链接与软链接的优缺点

2.4.1 硬链接优缺点

硬连接比较安全，因为即使某一个目录下的关联数据被删除了也没有关系，只要有任何一个目录下存在着关联数据，那么该文件就不会被删除，而且硬连接还不需要耗用 inode 和 block，但是硬连接也有其限制，就是不能跨文件系统也不能连接到目录。在硬连接中，删除文件时，只有当连接数为 0 时，才能文件真正删除，否则只会把文件的连接数减 1。

2.4.2 软连接优缺点

软件连接比较灵活，可以连接到文件和目录，但是它会耗用 inode 和 block，不过这对于系统来说其实不算什么，但是如果目标文件被删除了，从最后一幅图可以看出，整个环节就会无法继续下去，会发生无法通过连接文件读取的问题。其实软连接就你 Windows 中的快捷方式一样。

2.4.3 应用场景

（1）软链接会占用 inode 值，如果存在大量软连接，可能造成硬盘空间未满，inode 值却耗尽的问题。（软连接很小，但是却至少要占一个 block），但这种情况很少发生。

（2）根据硬链接的特性，可以为重要文件创建多个硬链接，防止误操作删除导致的问题。

（3）如果遇到了磁盘空间占满的情况，但又忘记配置 LVM 的情况，但是文件不是太大的话，可以将数据迁移到其他磁盘，通过软连接访问。