《操作系统原理》实验报告

实验名称	文件系统	实验序号	5
实验日期	2023/05/08	实验人	盖乐

一、实验题目

下面的练习检查了 UNIX 或 Linux 系统上文件和 inode 之间的关系。在这些系统上，文件用 inode 表示。也就是说，一个 inode 是一个文件（反之亦然）。您可以在随本文提供的 Linux 虚拟机上完成此练习。您也可以在任何 Linux、UNIX 或 Mac OS X 系统上完成练习，但需要创建两个名为 file1.txt 和 file3.txt 的简单文本文件，其内容是唯一的句子。

二、相关原理与知识

1）inode 相关知识

文件储存在硬盘上，硬盘的最小存储单位叫做“扇区”（Sector）。每个扇区储存 512 字节（相当于 0.5KB）。操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个块（block）。这种由多个扇区组成的块，是文件存取的最小单位。块的大小，最常见的是 4KB，即连续八个 sector 组成一个 block。

文件数据都储存在块中，那么很显然，我们还必须找到一个地方储存文件的元信息，比如文件的创建者、文件的创建日期、文件的大小等等。这种储存文件元信息的区域就叫做 inode，中文译名为"索引节点"。每一个文件都有对应的 inode，里面包含了与该文件有关的一些信息。

inode 包含文件的元信息，具体来说有以下内容：

* 文件的字节数
* 文件拥有者的 User ID
* 文件的 Group ID
* 文件的读、写、执行权限
* 文件的时间戳，共有三个：ctime 指 inode 上一次变动的时间，mtime

指文件内容上一次变动的时间，atime 指文件上一次打开的时间。

* 链接数，即有多少文件名指向这个 inode

* 文件数据 block 的位置

可以用 stat 命令，查看某个文件的 inode 信息：

$ stat file_name

总之，除了文件名以外的所有文件信息，都存在 inode 之中。

inode 也会消耗硬盘空间，所以硬盘格式化的时候，操作系统自动将硬盘分成两个区域。一个是数据区，存放文件数据；另一个是 inode 区（inode table），存放 inode 所包含的信息每个 inode 节点的大小，一般是 128 字节或 256 字节。inode 节点的总数，在格式化时就给定，一般是每 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%。

查看每个硬盘分区的 inode 总数和已经使用的数量，可以使用 df 命令。

查看每个 inode 节点的大小，可以用如下命令：

$ sudo dumpe2fs -h /dev/hda | grep "Inode size"

由于每个文件都必须有一个 inode，因此有可能发生 inode 已经用光，但是硬盘还未存满的情况。这时，就无法在硬盘上创建新文件。

用户通过文件名，打开文件，实际上，系统内部这个过程分成三步：首先，系统找到这个文件名对应的 inode 号码；其次，通过 inode 号码，获取 inode 信息；最后，根据 inode 信息，找到文件数据所在的 block，读出数据。

Unix/Linux 系统中，目录（directory）也是一种文件。打开目录，实际上就是打开目录文件。

目录文件的结构非常简单，就是一系列目录项（dirent）的列表。每个目录项，由两部分组成：所包含文件的文件名，以及该文件名对应的 inode 号码。 由于 inode 号码与文件名分离，这种机制导致了一些 Unix/Linux 系统特有的现象。

1. 有时，文件名包含特殊字符，无法正常删除。这时，直接删除 inode 节点，就能起到删除文件的作用。

2. 移动文件或重命名文件，只是改变文件名，不影响 inode 号码。

3. 打开一个文件以后，系统就以 inode 号码来识别这个文件，不再考虑文件名。因此，通常来说，系统无法从 inode 号码得知文件名。

第 3 点使得软件更新变得简单，可以在不关闭软件的情况下进行更新，不需要重启。因为系统通过 inode 号码，识别运行中的文件，不通过文件名。更新的时候，新版文件以同样的文件名，生成一个新的 inode，不会影响到运行中的文件。等到下一次运行这个软件的时候，文件名就自动指向新版文件，旧版文件的 inode 则被回收。

2）硬链接和软链接

硬链接

· 具有相同 inode 节点号的多个文件互为硬链接文件；

· 删除硬链接文件或者删除源文件任意之一，文件实体并未被删除；

· 只有删除了源文件和所有对应的硬链接文件，文件实体才会被删除；

· 硬链接文件是文件的另一个入口；

· 可以通过给文件设置硬链接文件来防止重要文件被误删；

· 创建硬链接命令 ln 源文件硬链接文件；

· 硬链接文件是普通文件，可以用 rm 删除；

· 对于静态文件（没有进程正在调用），当硬链接数为 0 时文件就被删除。

注意：如果有进程正在调用，则无法删除或者即使文件名被删除但空间不会释放。

软链接

· 软链接类似 windows 系统的快捷方式；

· 软链接里面存放的是源文件的路径，指向源文件；

· 删除源文件，软链接依然存在，但无法访问源文件内容；

· 软链接失效时一般是白字红底闪烁；

· 创建软链接命令 ln -s 源文件 软链接文件；

· 软链接和源文件是不同的文件，文件类型也不同，inode 号也不同；

· 软链接的文件类型是“l”，可以用 rm 删除。

区别：

原理上，硬链接和源文件的 inode 节点号相同，两者互为硬链接。软连接和源文件的 inode 节点号不同，进而指向的 block 也不同，软连接 block 中存放了源文件的路径名。

实际上，硬链接和源文件是同一份文件，而软连接是独立的文件，类似于快捷方式，存储着源文件的位置信息便于指向。

使用限制上，不能对目录创建硬链接，不能对不同文件系统创建硬链接，不能对不存在的文件创建硬链接；可以对目录创建软连接，可以跨文件系统创建软连接，可以对不存在的文件创建软连接。

3）文件索引

当要对大数据文件进行随机读取时，一种方法是先全部读入内存，以数组形式存储，通过数组索引下标形式进行访问，缺点是要占用大量的内存，我们知道对计算机而言内存是相当宝贵的。另一种方法就是建立文件索引，通过文件索引查找数据。思路：把每一行字符串的数据首地址记录下来，存入数组，再通过文件指针访问数组中的存储地址所指向的数据。

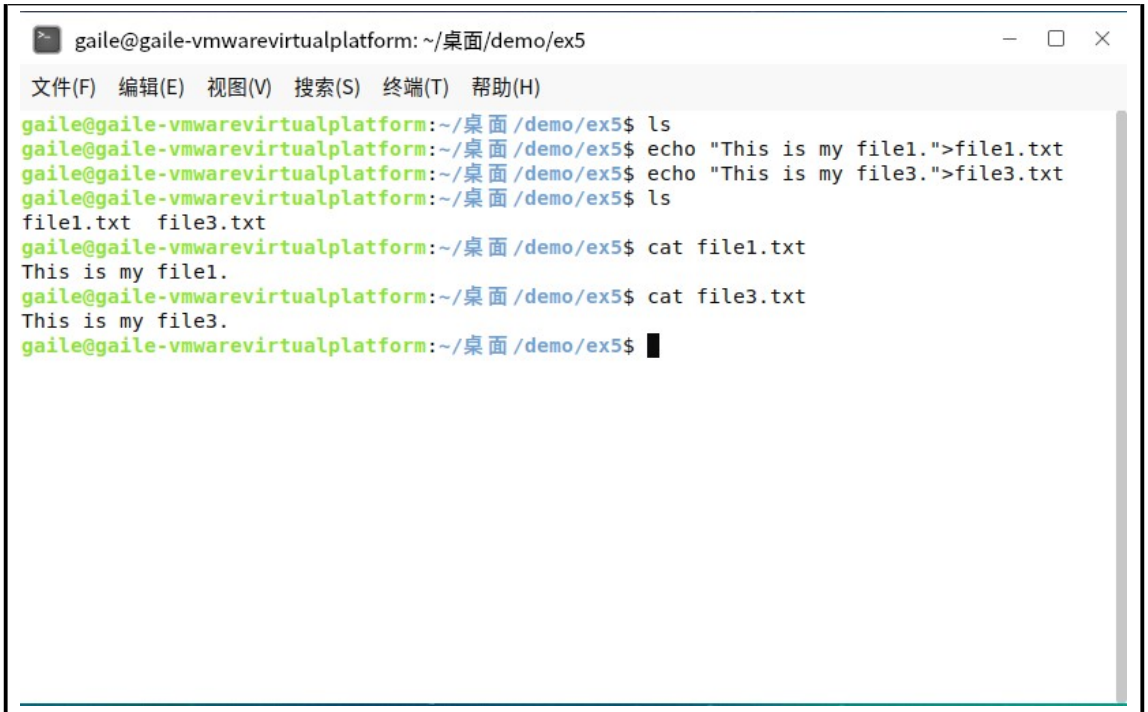
程序步骤：

1. 读取文件中一共有多少行数据
2. 得到每行数据在文件中的地址
3. 写入索引文件
4. 载入索引文件到内存或者直接从索引文件读取每行数据所在的地址
5. 随机读取想要读取的行数

三、实验过程

• 硬链接

首先，按照题目要求创建 file1.txt 和 file3.txt 两个文件，并写入不同的内容。



查看 file1.txt 与 file3.txt 的 inode 号分别为 134029 和 135642。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls -li
总用量 8
134029 -rw-rw-r-- 1 gaile gaile 18 5月  11 13:47 file1.txt
135642 -rw-rw-r-- 1 gaile gaile 18 5月  11 13:47 file3.txt
```

之后通过 ln 创建 file1.txt 的硬链接 file2.txt ，查看 inode 号可以发现 file1.txt 与 file2.txt 相同，均为 134029。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ln file1.txt file2.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls -li
总用量 12
134029 -rw-rw-r-- 2 gaile gaile 18 5月  11 13:47 file1.txt
134029 -rw-rw-r-- 2 gaile gaile 18 5月  11 13:47 file2.txt
135642 -rw-rw-r-- 1 gaile gaile 18 5月  11 13:47 file3.txt
```

查看 file2.txt 的内容发现和 file1.txt 内容相同，之后改变 file2.txt 的内容，发现 file1.txt 的内容也被改变。

这是因为在 Linux 中，硬链接（hard link）是指向同一个 inode 的不同文件名。当创建硬链接时，操作系统只是在文件系统中添加一个新的文件名，该文件名与原文件名共享同一个 inode，文件的数据块和元数据（如权限、所有者、创建时间等）。

因此，当通过 ln 命令创建一个硬链接 file2.txt 指向 file1.txt 时，这两个文件实际

上是同一个文件，它们的内容和元数据都是一致的。

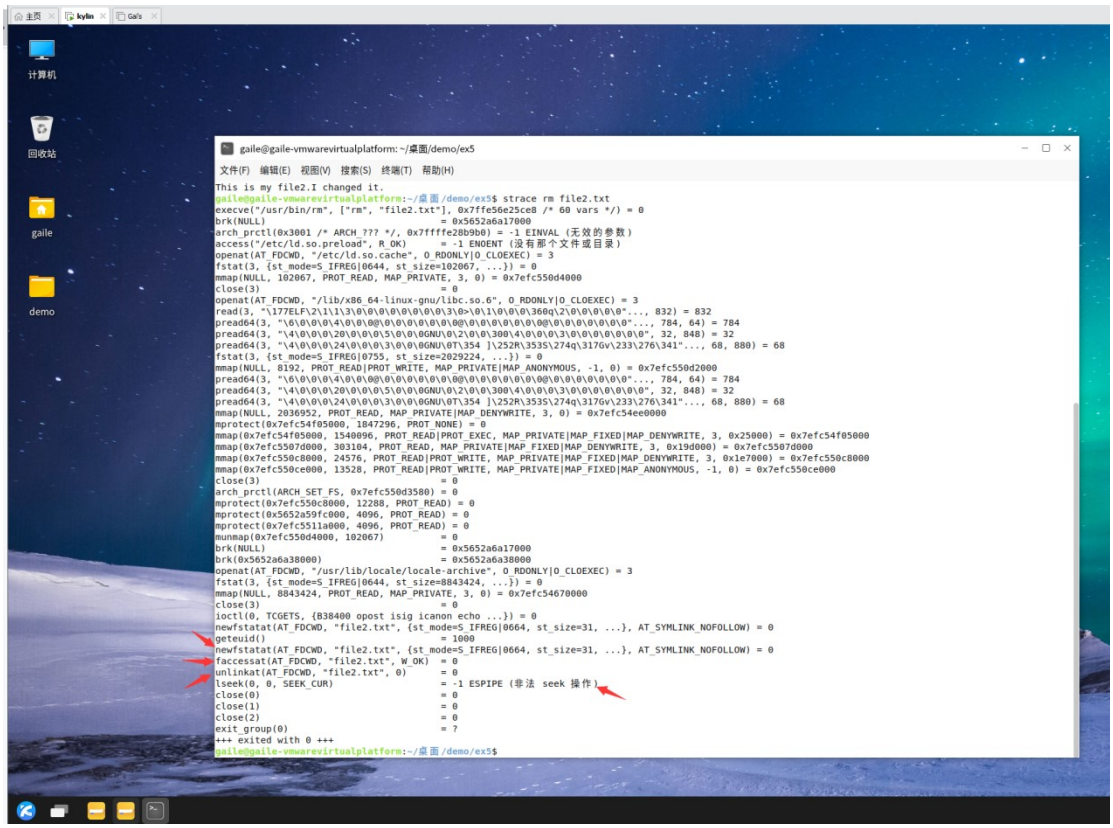
当你修改其中一个文件的内容时，实际上是在修改这个文件的数据块，而这个数据块同时被这两个文件共享，因此另一个文件的内容也会被改变，因为它们都指向相同的数据块。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file2.txt
This is my file1.
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ echo "This is my file2.I changed it.">file2.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file2.txt
This is my file2.I changed it.
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file1.txt
This is my file2.I changed it.
```

删除 file1.txt ，发现 file2.txt 的 inode 和内容都没有发生改变。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ rm file1.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls -li
总用量 8
134029 -rw-rw-r-- 1 gaile gaile 31 5月  11 13:54 file2.txt
135642 -rw-rw-r-- 1 gaile gaile 18 5月  11 13:47 file3.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file2.txt
This is my file2.I changed it.
```

之后删除 file2.txt 并用 strace 查看相关系统调用，可以看到 newfstatat 获取文件状态，之后使用 faccessat2 获取对该文件的权限，最后使用 unlinkat 将文件删除。最后使用 lseek 将指针指向标准输入的当前位置，但是返回值为-1，报错非法 seek。



• 软链接

使用 ln -s 命令创建 file3.txt 的软链接 file4.txt ，查看 inode 发现 file4.txt 的 inode 与 file3.txt 并不相同。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ln -s file3.txt file4.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls
file3.txt  file4.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls -li
总用量 4
135642 -rw-rw-r-- 1 gaile gaile 18 5月  11 13:47 file3.txt
133954 lrwxrwxrwx 1 gaile gaile  9 5月  11 14:05 file4.txt -> file3.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$
```

接下来我们尝试向 file4.txt 中写入内容，观察，我们发现 file3.txt 中内容同时被改变，这是由于文件链接的机制导致的，我们对软链接文件的修改本质上是修改源文件，软连接文件的作用仅仅是作为对源文件的一个“索引”，在这里笔者认为或许可以类比为 C 语言中指针与指针所指向区域的关系

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ echo "This is my file4.I changed it.">file4.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file4.txt
This is my file4.I changed it.
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file3.txt
This is my file4.I changed it.
```

尝试删除 file3.txt，查看 file4.txt 发现其变为红色。这个时候我们发现当我们再次尝试获取 file4.txt 的内容时我们是无法获取到的，这是因为 file4.txt 所链

接的源文件已经不存在了。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ rm file3.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls -li
总用量 0
133954 lrwxrwxrwx 1 gaile gaile 9 5月  11 14:05 file4.txt -> file3.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$
```

查看 file4.txt 的内容，发现报错：没有该文件，说明软链接所创建的是源文件的一个快捷方式，软链接中存放的是源文件的路径，修改软链接可以对源文件造成修改。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file4.txt
cat: file4.txt: 没有那个文件或目录
```

之后向 file4.txt 中写入内容，查看当前文件夹下的文件，发现 file3.txt 被重新创建，inode 仍然与 file4.txt 不同，且内容与 file4.txt 相同，符合了我们上面的判断， file4.txt 中存放的是 file3.txt 的路径。

```
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ echo "This is my file4.I changed it.">file4.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls
file3.txt  file4.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ ls -li
总用量 4
135639 -rw-rw-r-- 1 gaile gaile 31 5月  11 14:09 file3.txt
133954 lrwxrwxrwx 1 gaile gaile  9 5月  11 14:05 file4.txt -> file3.txt
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file4.txt
This is my file4.I changed it.
gaile@gaile-vmwarevirtualplatform:~/桌面/demo/ex5$ cat file3.txt
This is my file4.I changed it.
```

四、实验结果与分析

在本次实验中，我通过实践深入理解了 Linux 文件系统中硬链接和软链接之间的区别。硬链接相当于一个文件的不同入口，每次创建硬链接都会使 inode 节点中的“链接数”加 1。每删除一个该 inode 对应的文件名会使该 inode 节点中的“链接数”减 1。当这个值减到 0 时，就表明没有文件名指向这个 inode，系统就会回收这个 inode 号码以及所对应的 block 区域。在创建目录时，默认会生成两个目录项："."和".."。前者的 inode 号码就是当前目录的 inode 号码，等同于当前目录的“硬链接”；后者的 inode 号码就是当前目录的父目录的 inode 号码，等同于父目录的“硬链接”。因此，任何一个目录的“硬链接”总数总是等于 2 加上它的子目录总数（包括隐藏目录）。修改硬链接的权限会修改该 inode 对应的所有文件名的权限，因为文件系统内部使用的是 inode 号而不是文件名。

软链接中存放的是源文件的路径，读取软链接时，系统会自动将访问者导向

源文件，因此无论打开的是哪一个软链接，最终读取的都是源文件。软链接依赖于源文件而存在，因此如果源文件被删除，所有指向它的软链接也会失效。在修改软链接权限时，软链接的权限不会发生更改，而是会对源文件的权限进行更改。

通过这次实验，我更加深入了解了 Linux 文件系统中硬链接和软链接的工作原理，也体会到了 Linux 文件系统的简单易上手。

五、问题总结

本实验比较基础，遇到的问题无需总结。

六、参考资料

[1] 理解 inode https://www.ruanyifeng.com/blog/2011/12/inode.html

[2] linux 文件管理（inode、文件描述符表、文件表）

https://blog.csdn.net/wwwlyj123321/article/details/100298377

[3] Linux 硬链接和软连接的区别与总结

https://xzchsia.github.io/2020/03/05/linux-hard-soft-link/

[4] Linux 软连接和硬链接

https://www.cnblogs.com/itech/archive/2009/04/10/1433052.html