微计算机系统设计第一次作业报告

2019310892 师天麾

设计与实现

按照课上所讲文件系统 7 层进行设计，以下介绍几个主要的类及功能。

源代码：https://github.com/sth1997/inode-based-filesystem

Block

一个块，里面保存着大小为 BLOCK_SIZE=4096 的 buffer。每个块使用一个文件来存储，模拟底层设备接口。

SuperBlock

继承 Block 类。记录着一些文件系统的相关信息，如 inode 数量、InodeBitmap 的起始和结束的 Block number、下一个空闲的 Inode number、下一个空闲的 Block number 等信息。

Bitmap

继承 Block 类。能够查看某个 bit 是 0（已被使用）还是 1（未被使用），以及查找下一个未被使用的位置。

BitmapMultiBlocks

由于一个 Bitmap 只有 4096 * 8 个 bit，容量有限。而 BitmapMultiBlocks 通过记录起始和结束的一段连续 Block num，将多个 Bitmap 组织为一个更大的 Bitmap。提供的接口与 Bitmap 类似。

Inode

继承 Block 类。一个 Inode 为一个普通文件或目录或软链接文件。

对于普通文件，Inode 中有一个 Indirect 指针以支持超过 4M 的文件。

对于目录，每 16 字节记录目录中的一项，其中 12 字节记录文件名，4 字节记录 Block num。

对于软链接文件，直接记录其源文件的绝对路径。

除了文件和目录相关的操作外，我也将一些与路径相关的操作写为 Inode 的函数或静态函数。

利用 fuse 的 high-level API 实现 https://github.com/libfuse/libfuse。

实现的接口包括：

- getattr, open, read, write, create, release, rename, truncate
- mkdir, opendir, readdir, rmdir
- link, unlink, symlink, readlink

环境

Ubuntu16.04、GoogleTest v1.10.0、libfuse2.9.7

测试

项目通过 cmake 构建，可以通过 build/bin/目录下的可执行文件执行单元测试。

编译安装完成后，可以执行 build.sh 来编译出 build/bin/inode_fuse。再运行 test_fuse.sh 即可运行 fuse 接口的测试。若要运行测试脚本

为避免助教因版本问题不方便测试，在这里贴上运行截图。

创建文件夹、创建文件、写文件、读文件成功：

```
./build/bin/inode_fuse test_mount

cd test_mount

mkdir dir1

cd dir1

echo "f1" > f1.txt

stat f1.txt
```

```
+ stat f1.txt
  File: f1.txt
  Size: 3               Blocks: 1          IO Block: 4096   regular file
Device: 35h/53d Inode: 3           Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800
Modify: 1970-01-01 08:00:00.000000000 +0800
Change: 1970-01-01 08:00:00.000000000 +0800
 Birth: -
```

硬链接成功（Llinks：2）：

```
link f1.txt f2.txt

stat f1.txt

stat f2.txt

cat f2.txt

echo "f2" > f2.txt

cat f1.txt
```

```
+ link f1.txt f2.txt
+ stat f1.txt
  File: f1.txt
  Size: 3               Blocks: 1          IO Block: 4096   regular file
Device: 35h/53d Inode: 3           Links: 2
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800
Modify: 1970-01-01 08:00:00.000000000 +0800
Change: 1970-01-01 08:00:00.000000000 +0800
 Birth: -
+ stat f2.txt
  File: f2.txt
  Size: 3               Blocks: 1          IO Block: 4096   regular file
Device: 35h/53d Inode: 4           Links: 2
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800
Modify: 1970-01-01 08:00:00.000000000 +0800
Change: 1970-01-01 08:00:00.000000000 +0800
 Birth: -
+ cat f2.txt
f1
+ echo f2
+ cat f1.txt
f2
```

删除硬链接成功（Links：1）：

```
rm f1.txt

stat f2.txt
```

```
+ rm f1.txt
+ stat f2.txt
  File: f2.txt
  Size: 3               Blocks: 1          IO Block: 4096   regular file
Device: 35h/53d Inode: 4           Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800
Modify: 1970-01-01 08:00:00.000000000 +0800
Change: 1970-01-01 08:00:00.000000000 +0800
 Birth: -
```

软链接文件夹成功：

```
cd ..

ln -s ${mountdir}"/dir1" dir2

ls -l dir2

cd dir2

cat f2.txt
```

```
+ ln -s /home/parallels/download/inode-based-filesystem/test_mount/dir1 dir2
+ ls -l dir2
lrw-rw-rw- 1 root root 63 Jan  1  1970 dir2 -> /home/parallels/download/inode-based-filesystem/test_mount/dir1
+ cd dir2
+ cat f2.txt
f2
```

软链接文件成功：

```
ln -s ${mountdir}"/dir2/f2.txt" f3.txt

readlink f3.txt

echo "f3" > f3.txt

cat f2.txt
```

```
+ ln -s /home/parallels/download/inode-based-filesystem/test_mount/dir2/f2.txt f3.txt
+ readlink f3.txt
/home/parallels/download/inode-based-filesystem/test_mount/dir2/f2.txt
+ echo f3
+ cat f2.txt
f3
```

删除软链接成功：

```
rm f2.txt

cat f3.txt
```

```
+ rm f2.txt
+ cat f3.txt
cat: f3.txt: No such file or directory
```

truncate、存储超过 4M 文件、rename 成功：

```
cd ..

echo "f4" > f4.txt

truncate f4.txt --size 6000000

mv f4.txt f5.txt

stat f5.txt
```

```
+ truncate f4.txt --size 6000000
+ mv f4.txt f5.txt
+ stat f5.txt
  File: f5.txt
  Size: 6000000         Blocks: 1465       IO Block: 4096   regular file
Device: 35h/53d Inode: 10          Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800
Modify: 1970-01-01 08:00:00.000000000 +0800
Change: 1970-01-01 08:00:00.000000000 +0800
 Birth: -
```

思考

当目录下有大量小文件时（成千上万），可能优化方法：

大量小文件会有大量 inode 元数据以及大量的存储空间浪费，因为每个文件至少被分配一块（比如 4K）来存储，但如果文件内容只有几字节或几十字节，那一个块的大部分空间就都被浪费了。一个可能的方法是将大量的小文件合成一个大文件来存储，只用 1 个 inode，同时记录（比如用数据库）每个小文件在大文件中的起始和结束位置。

文件系统不同层提供的功能，Fuse 的接口分别使用了哪些层的功能，以及分层必要性：

第一层 Block Layer：操作磁盘的块。

第二层 File Layer：将块组织为文件。

第三层 Inode Number Layer：使用 inode number 对文件进行编号与区分。

第四层 File Name Layer :使用对用户更友好的 “文件名 “来代替 inode number 区分文件。

第五层 Path Name Layer：通过文件夹，对多级文件进行组织。

第六层 Absolute Path Name Layer：为命名系统提供一个根（root）。

第七层 Symbolic Link Layer：合并多个文件系统。

Fuse 接口使用了第三、六、七层。（我了解到有的同学用 rust 的库，函数接口给出的都是文件名，而我是用的库给出的都是绝对路径，所以我是用到的是第六层，而没有用到第四、五层。）

分层可以很好地隐藏底层的实现细节，提供更简洁明了的接口，增加易用性。