Project 6 File System设计文档

中国科学院大学

袁峥

2018.1.19

# 文件系统初始化设计

（1）请用图表示你设计的文件系统对磁盘的布局（布局上可以不考虑boot block的大小，直接从逻辑地址第0块开始），并说明各部分占用的磁盘空间大小，例如superblock，inode的元数据等

设计每个block为4KB，对于4GB的磁盘一共有1024\*1024=1048576个block。

superblock及其备份各占一个block，inode\_bit\_map占16个block，datablock\_bit\_map占32个block，inode\_table占16384个block，剩余1032142个block为数据区。

（2）你如何实现superblock的备份？如何判断superblock损坏，以及当有一个superblock损坏时你的文件系统如何正常启动？

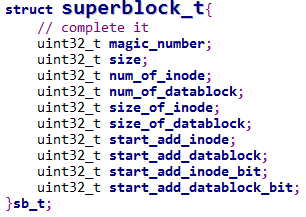
理论上，在整个4G磁盘的第一个block和最后一个block为superblock，其中第一份为主superblock，后一份为备份，两份同时修改，当第一份被破坏时，读取第二块中的内容。在实现时，由于为了便于测试两块superblock全部被破坏时的情况，若将备份块放在磁盘最后一块，数据覆盖耗时太长，因此将备份块放在了第二块。

判断superblock是否损坏可以采用将superblock所在块中的所有字节进行异或，并将结果记录在superblock中，读取时判断异或结果是否正确，如果不正确则表示已经被破坏。在实现时，采用了另一种较为简易的方面，对p6fs文件系统设置了一个magic number，并记录在superblock块的开头，读取时判断该数是否正确，如果错误则表示已经被破坏。

当有一个superblock损坏时，初始化函数会自动读取备份块中的数据，判断备份块中的数据是否也被破坏，如果被破坏则重新创建文件系统，否则根据备份块中的数据组织磁盘结构。

（3）请列出你设计的superblock和inode数据结构，并阐明各项含义。请说明你设计的文件系统能支持的最大文件大小，最多文件数目，以及单个目录下能支持的最多文件/子目录数目。

superblock数据结构：



magic\_number为自定义的p6fs的魔数，用于判断superblock是否损坏。

size表示该p6fs文件系统的大小，在本实验中为4GB。

num\_of\_inode为文件系统中最大支持的inode数，为512\*1024。

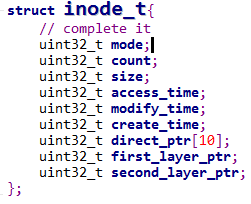
num\_of\_datablock为文件系统中最大支持的数据块数，为1032142。

size\_of\_inode为每个inode元数据的大小，为128B。

size\_of\_datablock为每个数据块的大小，为4096B。

start\_add\_inode、start\_add\_datablock、start\_add\_inode\_bit、start\_add\_datablock\_bit分别为p6fs文件系统中inode块、数据块、inode的bitmap、datablock的bitmap的起始地址。

inode数据结构：



mode为inode节点所表示的文件类型：文件、目录、软链接以及9位操作权限。

count为该inode的引用计数。

size为该inode所代表的文件的大小。

access\_time、modify\_time、create\_time分别为该inode的上次访问时间、上次修改时间和创建时间。

direct\_ptr、first\_layer\_ptr、second\_layer\_ptr分别为该inode所代表的文件内容的10个直接指针、1个一级间址指针和1个二级间址指针。

最大文件大小：10\*4KB+1024\*4KB+1024\*1024\*4KB=4GB+4MB+40KB。

最多文件数目：即superblock中的num\_of\_inode，为512\*1024。

单个目录下能支持的最多文件/子目录数目：在进行目录解析时，考虑到p6fs的实际情况，至采用了一级间址指针的解析，没有进行二级间址指针的解析。每个文件在目录中占用64B，因此单个目录下最多支持的文件数为67109504。

（4）请说明你设计的文件系统的块分配策略，按需分配还是有设计其他分配策略？

块分配策略为按需分配，在p6fs\_init、p6fs\_write、p6fs\_symlink、p6fs\_mkdir等函数中需要新块时进行分配，同时在p6fs\_truncate、p6fs\_rmdir、p6fs\_unlink、p6fs\_rename等函数释放数据块时进行回收。

（5）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

在设计上superblock的备份块应该和主superblock块的距离离的相对远一点比较好，因此在设计上本来将superblock分别放在4G磁盘的第一块和最后一块，但是在实现后的测试时发现，如果想要将两个块都破坏到需要往磁盘内写入4G内容，速度较慢，因此为了测试两个superblock都被破坏的情况，将备份superblock放在离主superblock较近的地方。

# 文件操作设计

（1）请说明mknod涉及的操作流程

1、解析路径是否存在，如果存在则返回EEXIST。

2、解析父路径是否存在，如果不存在则返回ENOENT。

3、分配inode，并在inode中写入相关信息。

4、在父目录页中写入新文件的信息，并更新父目录的inode。

（2）请说明link和unlink涉及的操作流程

link：

1、解析路径path和newpath，如果newpaht存在则返回EEXIST，如果path不存在则返回ENOENT。

2、解析path的父路径是否存在，如果不存在则返回ENOENT。

3、更新inode中的count（引用计数）及modify\_time。

4、在newpath的父目录中加入newpath的信息，并更新其父目录的inode。

unlink：

1、解析路径是否存在，如果不存在则返回ENOENT。

2、如果inode的count大于1，将count减1，并更新modify\_time。

3、如果inode的count等于1，删除inode及其所包含的datablock，并在父目录中删除其路径，更新inode。

（3）请说明open涉及的操作流程

1、解析路径是否存在，如果不存在则返回ENOENT。

2、分配文件描述符，并填入ino及flag信息。

3、将fileInfo->fh指向分配的文件描述符。

（4）请说明read涉及的操作流程

1、根据fileInfo->fh中的ino获取inode信息。

2、查看fileInfo->fh中的flag，判断是否读权限。

3、先定位到offset位置，然后从该位置开始读取size大小到buf。

4、更新inode的access\_time。

（5）请说明write涉及的操作流程

1、根据fileInfo->fh中的ino获取inode信息。

2、查看fileInfo->fh中的flag，判断是否写权限。

3、先判断当前文件大小是否小于offset+size，若小于则分配满足offset+size的datablock。

4、先定位到offset位置，然后从该位置开始写size大小的buf中的内容。

5、更新inode的access\_time、modify\_time、size。

（6）请说明truncate涉及的操作流程

1、解析路径是否存在，如果不存在则返回ENOENT。

2、如果newsize大于原文件的size，则在原文件后面补newsize-size大小的0。

3、如果newsize小于原文件的size，则将原文件大于newsize的部分删除，并回收相应的datablock。

4、更新inode中的size、modify\_time、access\_time信息。

（7）请说明release涉及的操作流程

释放文件描述符，并将其fileInfo->fh赋为NULL。

（8）请说明rename涉及的操作流程

1、解析路径path和newpath，如果newpaht存在则返回EEXIST，如果path不存在则返回ENOENT。

2、在路径path的父目录中删除path的信息，并更新其inode。

3、在路径newpath的父目录中增加newpath的信息，并更新其inode。

（9）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

在文件操作中比较复杂的是read和write中对文件读写时，需要根据offset和size来对文件进行定位，这部分需要根据具体大小来解析到其在哪一级文件块指针，在解析的过程中要分类讨论的情况较多，而且较容易出错。

# 目录操作设计

（1）请说明mkdir的操作流程

1、解析需要创建的目录路径是否存在，如果已经存在则返回EEXIST。

2、解析创建目录的父目录是否已经存在，如果不存在则返回ENOENT。

3、分配inode和datablock，分别存放新目录的元数据和目录页。

4、在新分配的datablock中放入“.”和“..”的信息。

5、在创建目录的父目录的目录页中放入新创建目录的信息，并更新父目录的inode信息。

（2）请说明rmdir的操作流程

1、解析需要删除的目录路径是否存在，如果不存在则返回ENOENT。

2、查看需要删除的目录是否为空目录，如果不是空目录则返回ENOTEMPTY。

3、删除目录的目录页及inode并更新bitmap进行回收。

4、在删除目录的父目录中删除新删除的目录信息，并更新其inode。

（3）请说明readdir的操作流程

1、解析路径是否存在，如果不存在则返回ENOENT。

2、根据目录页大小逐项读取目录页中的信息，并通过filler函数放入buf。

（4）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

这个实验的代码量比较大，在一开始的设计上做的不是很好，后来发现在目录和文件部分有许多代码其实可以复用，在代码复用这一块做的不是特别好，导致写的代码比较长。

# 其他操作设计

（1）请说明symlink涉及的操作流程

1、解析路径link，如果已经存在则返回EEXIST。

2、解析link的父路径，如果不存在则返回ENOENT。

3、分配新的inode及datablock，在datablock中写入path，并更新inode信息（mode设为软链接）。

4、更新link的父目录页，加入link的信息，并更新其父目录的inode。

（2）请说明readlink涉及的操作流程

1、解析路径path，如果不存在则返回ENOENT。

2、判断其inode的mode是否为S\_IFLNK（软链接），如果不是则返回EINVAL。

3、从inode所指向的datablock中读出路径并放入link中。

4、更新inode中的access\_time信息。

（3）请说明getattr涉及的操作流程

1、解析路径是否存在，如果不存在则返回ENOENT。

2、将inode中的mode、count、size、modify\_time、access\_time、create\_time存入statbuf中的相应项中。

3、通过fuse\_get\_context函数获得uid和gid。

（4）请说明utime涉及的操作流程

1、解析路径是否存在，如果不存在则返回ENOENT。

2、将inode中的modify\_time及access\_time赋为ubuf中的相应项数据。

（5）请说明chmod涉及的操作流程

1、解析路径是否存在，如果不存在则返回ENOENT。

2、更新inode中的mode信息。

（6）请说明statfs涉及的操作流程

填写statInfo中的f\_bsize、f\_frsize、f\_blocks、f\_bfree、f\_bavail、f\_files、f\_ffree、f\_favail的信息。

（7）请说明init涉及的操作流程

读取磁盘中的第一个block，查看其中的magic\_number是否正确，若不正确则读取备份superblock中的信息，查看magic\_number是否正确，若两个块中的magic\_number都不正确则进入mkfs，否则进入mount。

mkfs：

1、将superblock块中应保存的值同时存入主superblock及备份superblock。

2、同时清空磁盘和内核中的inode\_bitmap及block\_bitmap。

3、分配inode及datablock，存放根目录的信息。

mount：

1、将superblock中相关数据存入内核。

2、将磁盘中的inode信息存入内核。

3、扫描inode\_bitmap及block\_bitmap，计算剩余数，并将bitmap存入内核。

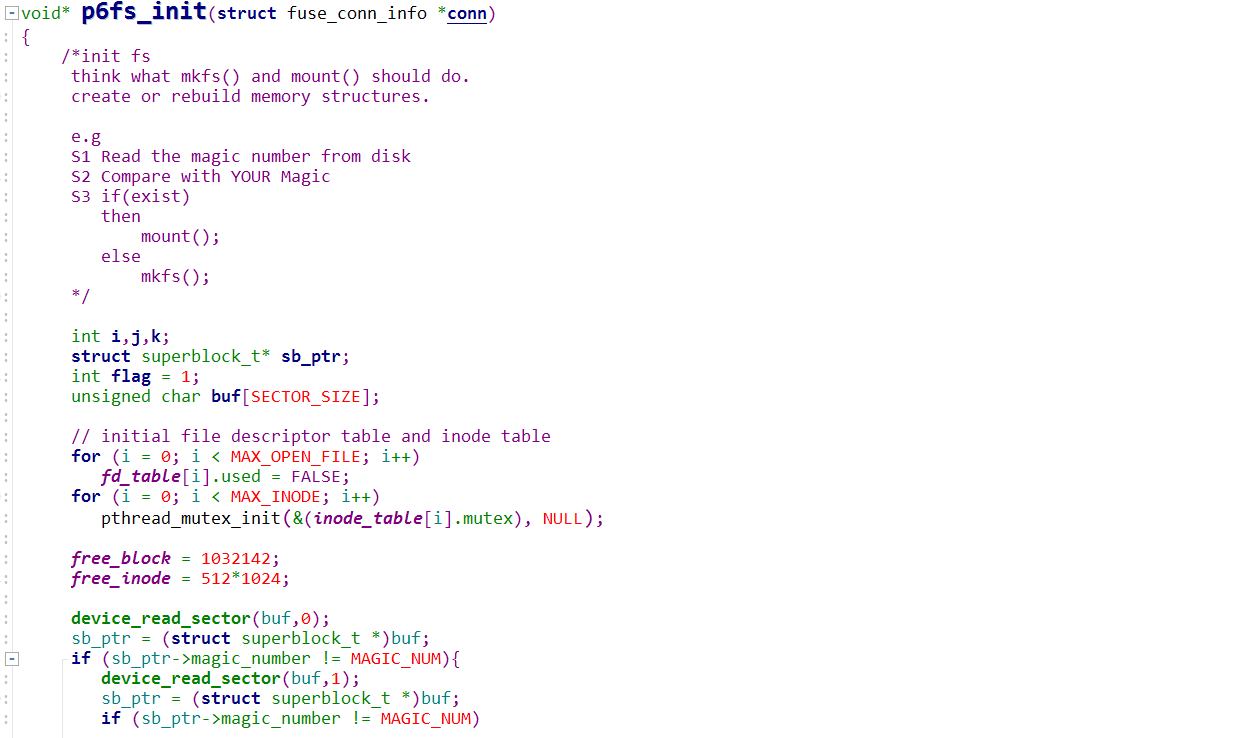
（8）请说明destroy涉及的操作流程

释放内核中的inode，并关闭磁盘及log\_file。

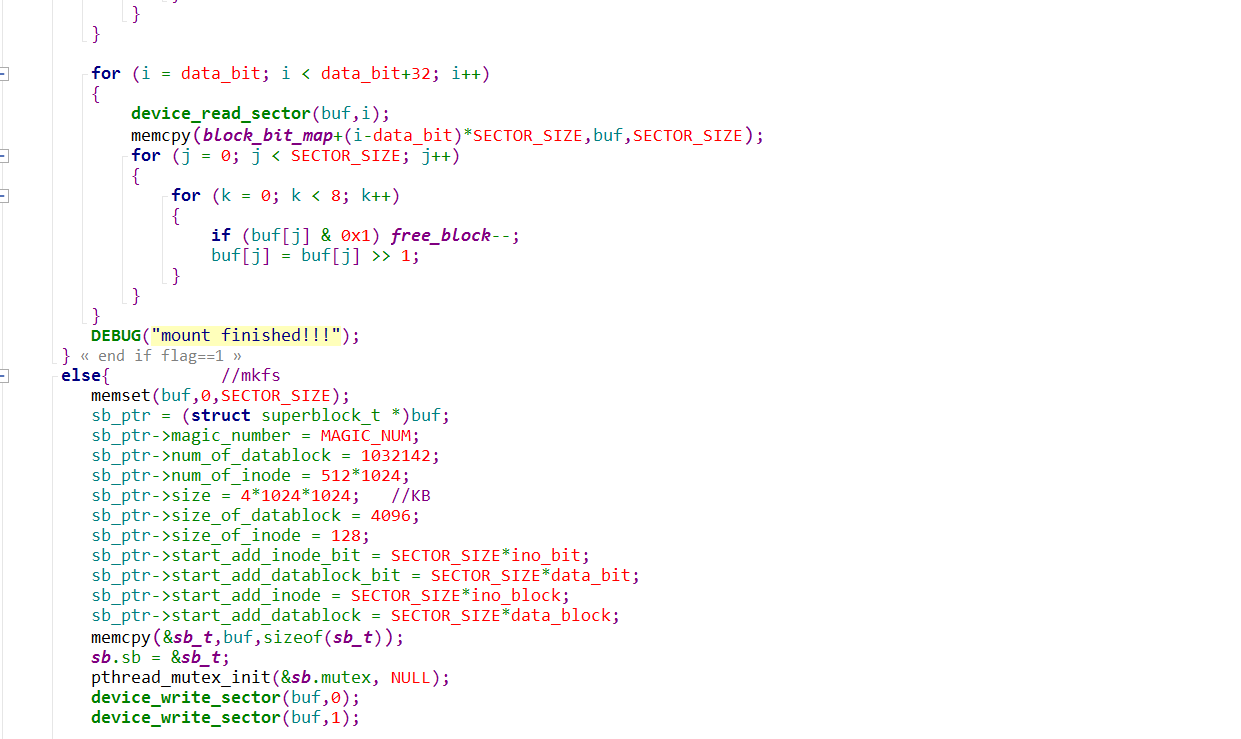
# 关键函数功能

请列出上述各项功能设计里，你觉得关键的函数或代码块，及其作用

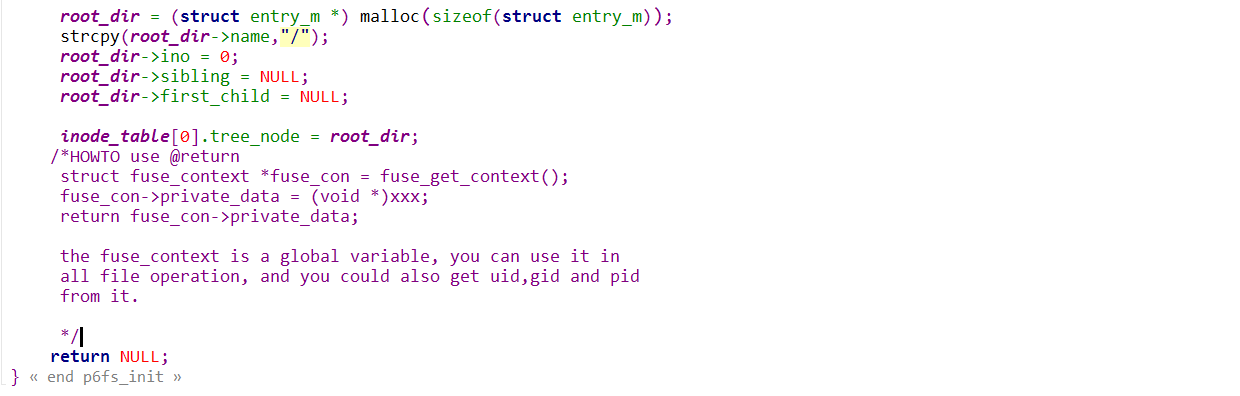
1、p6fs\_init（common.c）











首先根据主superblock及备份superblock中的magic\_number判断进入mount还是mkfs。

然后分别进行挂载或者创建文件系统。

2、p6fs\_read（common.c）



